20

UPDATING IN THE "DECISION AIDS FOR TUNNELING"

by

Christoph Haas

ENG

Diploma in Civil Engineering
Swiss Federal Institute of Technology (1998)

Submitted to the Department of Civil and Environmental Engineering
in Partial Fulfillment of the Requirements for the Degree of

Master of Science in Civil and Environmental Engineering

at the

Massachusetts Institute of Technology

June 2000

© 2000 Massachusetts Institute of Technology
All rights reserved

Signature of Author.....
Department of Civil and Environmental Engineering
April 17, 2000

Certified by.....
Herbert H. Einstein
Professor of Civil Engineering and Environmental Engineering
Thesis Supervisor

Accepted by.....
Daniele Veneziano
Professor of Civil Engineering and Environmental Engineering
Chairman, Department Committee on Graduate Studies

ENG

UPDATING IN THE “DECISION AIDS FOR TUNNELING”

by

Christoph Haas

Submitted to the Department of Civil and Environmental Engineering
on April 17, 2000 in Partial Fulfillment of the Requirements for the Degree of
Master of Science in Civil and Environmental Engineering

ABSTRACT

The “Decision Aids for Tunneling” (DAT) are a procedure and computer code which can be used to assess uncertainties which are caused by the geologic / geotechnical conditions and from the construction process. In particular the DAT can be used to predict construction costs and time. The computer code determines the overall uncertainty which results from the many individual uncertainties that are present in a tunneling project. These individual uncertainties are input parameters and must be specified by the user.

While construction is in progress there is a need for updated predictions. These can be used for improvements in scheduling, resource allocation, financial planning and so on.

This work presents an updating procedure and associated code which allows one to refine predictions during construction. Updating not only involves replacing the original prediction by actual data from the excavation but also includes a learning effect. This uses information from the actual excavation to arrive at an improved prediction for the unexcavated part. The Updating Module of the DAT is a tool which helps the user refining input parameters by deriving relevant information from construction data and presenting it together with original input. Graphical presentation of original input, construction information and updated input assists in the interpretation of the information. A mathematical model which is based on Bayes’ theorem makes the code also capable of suggesting updated input parameters in some cases. An example project shows that the learning effect has a significant impact on the precision of the prediction and reduces the uncertainty substantially.

Thesis Supervisor: Herbert H. Einstein

Title: Professor of Civil and Environmental Engineering

Acknowledgments

First and foremost my thanks go to Prof. Einstein. His supportive manner made it a great pleasure to work with him. He has the great ability to guide a student while at the same time encouraging the students initiative.

Of course I owe a lot to my parents, since it was their support during my whole life which finally led my way to MIT.

Special thanks go to Claude Indermitte for his help and patient explanations about Simsuper during the course of my research.

For the many hours we spent together I would also like to thank my friends both in and outside of MIT, above all Catalina, Evelyne, Jorge, Martin, Pong, Rosanne and Dave, Sandy and Yun.

Table of Contents

1. Introduction.....	9
2. The “Decision Aids for Tunneling”	12
2.1 Purpose	12
2.2 Simulations	13
2.3 Geology Module	14
2.4 Construction Module	18
2.5 Resource Module	20
2.6 Program Results / Output	20
2.7 The Need for an Updating Process	21
3. The Markov Process.....	23
3.1 Introduction and Summary	23
3.2 Basic Elements of the Markov Process	25
3.3 State Prediction at a Future Point	28
3.4 Input Parameters for the Markov Process in the DAT	32
4. The Updating Process	33
4.1 General Framework	33
4.2 Bayes’ Theorem	33
4.3 Treatment of Different Types of Input Data	35
5. Program Input.....	38
5.1 General Input Characteristics	38
5.2 Tunnel Network and Tunnel Geometries	40
5.3 Tunnel Information	42
5.4 Areas and Zones	43
5.5 Geotechnical Parameter Sets, Parameters and Parameter States	45
5.6 Parameters for the Markov Process	48
5.7 Information Points	49
5.8 Ground Classes and Combined Ground Classes	50
5.9 Correlation Numbers	51
5.10 Methods, Headings, Activity Networks and Cycle Lengths	52
5.11 Activities and Equations	53
5.12 Method Variables and General Variables	55
5.13 Association of Geometries, Ground Classes and Methods	55
5.14 Learn Parameters	56
5.15 Method Change Delays	57
5.16 Repositories, Resources and Stock Levels	58
5.17 Daily Fluctuations vs. Long-Term Averages	58
6. Observations	60
6.1 Observed Parameters	60
6.2 Data Collection and Project Database	65
7. The Updating Process in Detail	68
7.1 Recapitulation of General Framework	68
7.2 Parameters of the Markov Process	69

7.3	Variables and Cycle Lengths	72
7.4	Semi-Deterministic Definition of Geotechnical Parameters	75
7.5	Deterministic Definition of Geotechnical Parameters	78
7.6	Tunnel Delays	79
7.7	Association of Geometries, Ground Classes and Methods	82
7.8	Method Change Delays	84
7.9	Tunnel Detail Information	85
7.10	Creation of Updated Input Files	86
7.11	Possible Extensions for Updating Variables and Cycle Lengths	88
7.12	Possible Extension for Updating Transition Probabilities	95
7.13	Possible Extension for Updating Associations between Geometries, Ground Classes and Methods	96
8.	Mathematical Background	97
8.1	Updating with Continuous Probability Distributions	97
8.2	Updating Markov Parameters	98
8.3	Properties of the Triangular PDF with Spikes	116
8.4	Updating Variables and Cycle Lengths as a Stochastic Process	120
8.5	The PDF of the Inverse of a Random Variable	122
9.	Example Project	127
9.1	Description of Project	127
9.2	Initial (prior) Predictions	132
9.3	Observations	137
9.4	Updating Process	138
9.5	Updated (posterior) Predictions and Comparison	143
10.	Conclusion and Outlook.....	149
10.1	Conclusion	149
10.2	Outlook	150
	References.....	151
	Appendices (see separate table of contents).....	153

Updating in the “Decision Aids for Tunneling”

1. Introduction

Tunneling very often involves high degrees of uncertainty, more so than many other areas of civil engineering. These uncertainties stem from two major problems: The geologic conditions are never exactly known and, particularly for deep and long tunnels, pre-construction information may be very sparse. But even if the geologic conditions are known, there is still considerable uncertainty about the construction process itself. Advance rates and costs depend on such things as the reliability and performance of the equipment as well as the skills and morale of the workers. In addition there is a basic variability about the mechanical process itself. And even if geologic conditions are thought to be known, many unforeseen events can still occur due to minor geologic details.

As a consequence, advance rates and costs can never be exactly predicted. It is however possible to determine a range over which they vary. This can be done with any kind of probability distribution or frequency plot (Figure 1.1, Figure 1.2).

Figure 1.1: Histogram for Construction Costs

Figure 1.2: Probability Distribution for Construction Costs

The uncertainty associated with a tunneling project influences all major associated decisions:

- the decision to plan and realize the project at all
- the selection of alternatives for different alignments and shaft locations
- the selection of excavation methods and design cross-sections
- the decision to perform additional exploration

The challenge is to formally represent these uncertainties in order to formalize the decision process. A particular difficulty is to quantify the total uncertainty about a project, given the numerous individual uncertainties. To address these issues, the DAT (Decision Aids for Tunneling) were developed (see e.g. Einstein et al., 1998). The DAT are programmed in C/C++ and are based on the MOTIF and XWINDOW system.

A typical result of the DAT is a scattergram of total construction time and total construction cost. This scattergram reflects the overall uncertainty associated with a particular project (Figure 1.3).

Figure 1.3: Time-Cost-Scattergram

Typically the DAT have been applied at an early stage in the project life, almost exclusively before construction actually started. Therefore any program input is based on expert knowledge (like anticipated geologic conditions) and/or experience from previous projects (like probability distributions for costs per tunnel meter and advance rates).

However, once construction is in progress, more and more information from the actual project will be available. The uncertainty about the excavated part disappears. What remains is uncertainty about the unexcavated part, but since that part becomes smaller and smaller, overall uncertainty obviously decreases (Figure 1.4). In addition to this, it

may be possible to further reduce the uncertainty about the unexcavated part by drawing conclusions from information obtained in the excavated part.

It will therefore be possible to make more accurate predictions. This has various benefits. The most important ones are an optimum allocation of resources (including finances) and an improved scheduling of secondary works such as ventilation, lighting, road paving and similar.

Figure 1.4: Uncertainty about construction progress before and after construction start

This thesis presents a formal way of combining observations and initial predictions in order to obtain more refined predictions during the construction process. This procedure will henceforth be called the updating process.

The first part of the thesis presents the different modules of the DAT and explains its mode of operation, as well as some underlying concepts. Then a framework for the updating process is presented.

The second part describes the various input data of the DAT and explains the updating process and its implementation in a new module, the “updating module”, in detail.

The third part presents an example project. It was not possible to apply the updating process to a real project during the time this thesis was written. Therefore a virtual project has been created.

The thesis ends with some conclusions about this work and comments on future extensions.

2. The “Decision Aids for Tunneling”

2.1 Purpose

The “Decision Aids for Tunneling” (DAT) were developed to capture uncertainties in tunneling. The following issues can be addressed:

- A decision whether to undertake a project at all can be made on the basis of risk analysis. The risk of exceeding certain cost and time limits, together with other risks (e.g. operational, environmental, political) can be compared to potential benefits.
- A comparison of alternatives with regard to alignment, intermediate access tunnels, as well as excavation methods should be made not only on the basis of mean values but also of associated uncertainties, as shown in Figure 2.1 and Figure 2.2.
- When assessing the value of exploration, the main purpose is generally not to lower the mean cost and time, but rather to limit uncertainties. The DAT allow one to estimate the potential reduction of uncertainties through exploration, before this exploration is actually performed. Thereby its value can be estimated.
- Cost- and time-distributions of the form of Figure 2.2 can be used as a basis for contracts or insurance coverage. The contractor may be required to bear uncertainties up to a certain limit and for everything beyond that, insurance coverage will be bought.

Figure 2.1: Comparison of Alternatives on the Basis of Distributions (similar cost distribution; different time distribution)

Figure 2.2: Time- or Cost-Distributions of two Alternatives with Identical Means (Alternative 2 has greater probabilities of long durations)

Figure 2.3: Reduction of Uncertainty due to Exploration

It is generally not sufficient to just rely on mean values and best case / worst case scenarios. In order to express uncertainty in terms of monetary value, the entire distribution must be considered. This becomes quite obvious when one tries to answer the questions “How bad is my worst case?”, “Will it ever occur?”, “Is it possible that things go even worse than the worst case?”. Eventually one has to answer the question of how *probable* the best case and worst case are. This can not be done unless the entire probability distribution is known.

The challenge with tunnels, as with any other system, is to find the overall uncertainty which results from countless small uncertainties. Only in very simple cases are there analytical solutions which allow one to combine a number of probability distributions into one single distribution. In real world an analytical approach is not feasible.

2.2 Simulations

The most commonly used approach to tackle the above problem are Monte-Carlo simulations. This approach is also used in the DAT.

The DAT actually simulates the construction of the tunnel network. This simulation is done many times – typically hundreds or thousands of times. Each simulation run uses a different set of parameters values (e.g. geology / geotechnical parameters, construction performance etc.). Each value is selected by playing a lottery from a probability density

function (like Figure 1.2, p. 9), which has to be defined by the user. This probability density function (pdf) describes the uncertainty about one particular parameter.

The simulation by the DAT is essentially a two step process. The first step generates a possible profile of geologic (and/or geotechnical) parameters. This is done by the Geology Module (section 2.3, "Geology Module"). The second step is performed by the Construction Module (section 2.4, "Construction Module", p. 18). This module simulates the construction process in that particular geologic profile. A third module, the Resource Module (section 2.5, "Resource Module", p. 20), runs parallel with the Construction Module and traces the flow of resources during the construction process.

Let us first look at only one particular parameter, for example the advance rate of a tunnel boring machine (TBM) under certain geologic conditions. The parameter is specified to have a pdf like Figure 1.2. In each simulation the DAT play a lottery and thereby select a different parameter value. The pdf defines the chances of selecting a particular value. Therefore, if a large number of simulations is performed, the frequency plot of all generated values (like Figure 1.1, p. 9) will approximate the pdf. In other words, the Monte-Carlo method reproduces the pdf's of all individual parameters. The approximation improves with larger numbers of simulations.

Of course one is interested in overall results, such as total cost, total construction time, than in the individual parameter values. Each simulation leads to one result such as a time-cost pair. One can perform a large number of simulations and present the result values in a frequency plot. The frequency plot is an approximation of the true probability density function of the real world problem, where again the accuracy increases with the number of simulations.

It is interesting to plot total cost vs. total time on the same diagram. This leads to a scattergram as shown in Figure 1.3 (p. 10). Every point in the scattergram represents one simulation.

2.3 Geology Module

The Geology Module (Chan 1980) generates possible geologic profiles along the tunnel (or tunnel network). The ultimate goal is to obtain a *ground class profile*. The ground class

profile will then be related to the Construction Module (section 2.4, “Construction Module”, p. 18), which associates construction methods with ground classes.

The process to create ground class profiles is based on a somewhat complex structure, which gives the user great flexibility and is therefore widely applicable:

The top level of the structure are areas. An area is a linear domain and typically represents a part of the tunnel system. Consider the following (schematic) tunnel network:

Figure 2.4: Tunnel Network

There are two areas, one being the main tunnel (including the auxiliary tunnel) and one being the access shaft.

Areas are further subdivided into zones. Zones are considered to be *statistically* homogeneous. This means that geologic parameters in every individual zone can be described by some kind of a statistical process, which remains the same over the length of the zone.

Typically zones are of the same geologic origin. A possible zone pattern for the area “tunnel” in the figure above could be:

Figure 2.5: Zone Pattern

Zones are the first major source of uncertainty. In many cases it will already be difficult to know the sequence of zones. The exact start and end positions of the zones area often highly uncertain. This input can therefore be made in a probabilistic manner, as will be described later in section 5.4 (“Areas and Zones”, p. 43).

Every zone has a set of geotechnical parameters (e.g. lithology, jointing, water inflow) and each parameter has some parameter states (e.g. sparse, intermediate and intense jointing). Parameter states of every parameter follow each other in a certain sequence.

Figure 2.6: Sequence of Parameter States in Zone “metamorphic rocks”

The exact sequence of parameter states is even less certain than the sequence of zones. However, one may have good estimates about the proportions in which the different parameter states occur. In other words, one does not know *exactly* which state follows which and how long one state persists, but one may have information about the *statistical properties*. The goal is therefore to find a statistical process which generates sequences of parameter states and which has the estimated statistical properties. One such process is the Markov Process. The Markov process will be explained thoroughly in chapter 3 (“The Markov Process”, p. 23).

Once a complete profile of geotechnical parameter states is generated, one can go about defining ground classes. A ground class is a combination of parameter states (of different geotechnical parameters). Verbally ground classes could for example be defined as follows:

- If jointing is “sparse” and water inflow is “little”, the ground class is “good conditions”.
- If jointing is “intermediate” and water inflow is “little”, the ground class is “medium conditions”.
- Whenever jointing is “intense”, the ground class is “bad conditions”.
- Whenever water inflow is “much”, the ground class is “bad conditions”.

One thus obtains a ground class profile for every zone and thereby along the whole tunnel alignment.

jointing:	sparse	intermediate	sparse	intense	intermediate
water inflow:	little	much	little	much	little
ground class:	good cond.	m.c.	bad c.	good cond.	bad conditions
					medium c.

Figure 2.7: Ground Class Profile (using the example definitions in the text above)

The DAT also allows one to define parameters which are independent of a particular zone. A typical example are faults. It may be known that there is fault zone in a certain part of the area, even though it may not be known whether there are sedimentary or metamorphic rocks there. It is thus possible to define a parameter “fault zone” for the whole area. This parameter would have the two states “exists” or “doesn’t exist”.

In this case there is both a zone ground class and an area ground class. The area ground class could be “bad conditions” if there is a fault zone and “neutral” if there is none. Based on the zone ground class and area ground class a combined ground class is defined. Verbally they would be defined as follows:

- If the area ground class is “bad conditions”, the combined ground class is always “bad conditions”
- If the area ground class is “neutral”, the combined ground class is equal to the zone ground class.

area parameter: fault zone	doesn't exist	exists	doesn't exist
area ground class:	neutral	bad conditions	neutral
zone ground class:	good cond.	m.c.	bad c.
	good cond.	bad conditions	medium c.
combined ground class:	good cond.	m.c.	bad c.
	good c.	bad conditions	medium c.

Figure 2.8: Combined Ground Class Profile (using the example definitions in the text above)

Eventually there will be a combined ground class profile for the whole tunnel network. This forms the basis with which the Construction Module associates such things as construction methods and their associated costs and construction times.

2.4 Construction Module

The Construction Module simulates the actual construction process. This can be done at almost any level of detail.

The main focus of the module is on the “method”. A method is a series of activities which advance the tunnel by one round (or cycle). A round is typically one cycle in drilling-and-blasting or one cycle of a tunnel boring machine advance. Every activity has its associated cost and time. The order in which activities are performed is defined in an activity network.

Figure 2.9: Activity Network

In its most simple form the activity network only consists of one arc. All tasks that need to be performed in one round are aggregated into one single arc. At the other extreme it is possible to go into great detail and precisely model parallel and sequential activities.

Many construction methods involve multiple headings. Especially in bad geotechnical conditions the excavation of the tunnel cross-section is separated in two or more parts. These parts are excavated with a certain distance between them.

Figure 2.10: Tunnel Excavation with Heading and Benching

In such a case, separate activity networks must be specified for every heading or drift.

The decision, which method is used in a particular part of the tunnel depends on the combined ground class (which is a result of the Geology Module) and the tunnel geometry. In Figure 2.4, for example, “main tunnel” and “auxiliary tunnel” are parallel to each other. Therefore they share the same ground class profile. Quite obviously though,

different methods are applied for construction. To address this issue, “geometry” is introduced. Geometry reflects various tunnel properties such as size or shape. For the purpose of the DAT, if under equal ground conditions, two different design and construction methods are applied for two tunnels, these two tunnels must have different geometries.

It is frequently observed, that construction performance is low at the beginning and increases gradually to reach a certain limit. This is known as the “learning curve”. There are many reasons for it:

- Workers must get used to the peculiarities of the actual project or excavation method
- Workers must get used to each other in order to have good team work
- Machinery may have teething troubles.
- The project organization and the associated flow of information is not yet streamlined

Figure 2.11: Learning Curve

In the DAT this learning curve can be specified by simply defining some points on it.

Sometimes switching between methods also causes delays. A new machine may have to be installed or there may be a “mini learning curve” because workers must get used to the new method. This case is provided for as well: In the DAT the user can specify “method change delays”.

2.5 Resource Module

The Resource Module tracks the flow of resources during the construction process. A comprehensive model was developed by Halabe (Halabe, 1995). However, only a simplified version of it is implemented in the current version of the DAT.

In the current implementation the model involves resources, repositories and activities. Resources can be anything that is an input or output of the construction process. Typical examples are concrete, steel, muck but also man-hours.

The resource needs (or output) of each activity determine the amount of resources that are taken from (or brought to) the repositories. Depending on the current position of the heading within the tunnel, resources are taken from a particular repository.

Repositories have a capacity and stock levels. If the capacity is exceeded, no more resources (e.g. muck) can be brought to the repository and the activity comes to a halt. Similarly, if the current stock level of a resource is zero, it is no longer available and activities needing it are forced to stop.

2.6 Program Results / Output

The most frequently used form of output are time-cost scattergrams, as shown in Figure 1.3 and Figure 2.12. From these scattergrams it is simple to generate frequency plots for costs and time.

Figure 2.12: Time-Cost-Scattergram with Frequency Plots

Figure 2.13: Time-Position-Graph

The time-position-graph (Figure 2.13) shows the construction progress at any point in time. In particular it can be seen where opposite tunnels meet (shaded areas).

A possible output of the resource module is a graph of stock levels vs. time (Figure 2.14).

Figure 2.14: Stock Level vs. Time Graph

2.7 The Need for an Updating Process

Typically the DAT have been applied at an early stage in the project life, almost exclusively before construction actually started. Therefore any program input is based on expert knowledge (like anticipated geologic conditions) and/or experience from previous projects (like probability distributions for costs per tunnel meter and advance rates).

However, once construction is in progress, more and more information from the actual project will be available. It will therefore be possible to make more accurate predictions. There are two basic ways of reducing uncertainty; one very obvious and one somewhat less obvious.

Replace predicted progress by actual progress

As construction progresses, the uncertainty about the excavated part of the tunnel disappears completely. In fact, what once was a prediction has become history. Geologic conditions can be observed and mapped and it will be known almost exactly how much time and money was spent so far. What remains is uncertainty about the unexcavated part, but since that part becomes smaller and smaller, overall uncertainty obviously decreases (Figure 2.15, left).

Figure 2.15: Reduction in Uncertainty:

- Left:** only by replacing predicted progress with actual progress
(prediction for the remaining part is parallel to initial prediction)
- Right:** Refinement of the prediction based on observation leads to an additional reduction in uncertainty.

Refine predictions based on observations

Less obviously, the uncertainty about the unexcavated part can also be reduced. Observations in the excavated part very often allow one to refine predictions for the unexcavated part (Figure 2.15, right). Examples are jointing intensity (if it is assumed that the unexcavated and the excavated part are of the same geologic origin) or advance rates for a certain construction method.

As construction progresses, it is vital to have periodically updated predictions about the remaining part of the construction. This allows one to allocate resources in an optimal way. Examples are financial planning but also re-allocation of machinery and manpower in cases where the critical path changes.

It is common practice to replace the predicted progress by the observed one. However, predictions for the remaining part are typically not refined. One reason is certainly the lack of a consistent and formal way of doing so.

The Updating Module of the DAT provides a framework for the updating process in the context of the DAT. Some aspects of the methodology and mathematical background provided in this thesis may also be more widely applicable.

3. The Markov Process

3.1 Introduction and Summary

The Markov process is a statistical process which is used by the DAT to generate sequences of parameter states. It can be set up in a way such that the generated parameter state sequence fulfills some pre-determined statistical conditions. The reason for using the Markov Process, is that the exact sequence is usually uncertain, but there may be some knowledge about the statistical properties of the sequence. The Markov process can generate parameter state sequences that both reflect our knowledge (statistical properties) as well as our uncertainties (the exact parameter state at any location is random).

The Markov process is a well known example of a stochastic process. A stochastic process involves random variables which are functions of a “time” parameter. For example the number N of people in a queue can be regarded as a random variable which depends on time. Thus at a given time t , the number of people is a random function $N(t)$ which has a certain probability distribution $P_N(n, t)$

An example is given in Figure 3.1, where $P_N(n, t)$ is the probability mass function (pmf) of $N(t)$. $P_N(n, t_1)$ and $P_N(n, t_2)$ are shown as pmf's of N at times t_1 and t_2 respectively. Thus for example at $t = t_1$, $P[N = 2] = 0.25$ while at $t = t_2$, $P[N = 2] = 0.5$.

Figure 3.1: Example Probability Mass Function of a Stochastic Process at two different times t_1 and t_2

The Markov process is characterized by a single-step memory. In order to determine the next step all previous steps except for the most recent one are neglected. This a

restrictive condition, but the most recent step should usually be the most important one in order to determine the next step. The advantages of a single-step process are simpler calculations and the fact that often full probability density functions can be found.

If the previous example about the number of people in a queue is assumed to be a Markov process, then the number of people in the queue at time t_{n+1} depends only on the number of people in the queue at time t_n and not on the numbers at earlier times t_{n-1} , t_{n-2} , ..., t_0 .

With respect to the DAT, what was called “time” above is equivalent to the position along the tunnel axis where position is identified by the distance ℓ from a certain fixed point, such as the portal of the tunnel.

Figure 3.2: Definition of Position

Parameter states behind the tunnel face depend only on the parameter state *at* the tunnel face. In order to predict the parameter state at a certain small distance behind the tunnel face, one needs to ask two basic questions, which lead to two sets of input parameters which are used by the DAT (see section 5.6, “Parameters for the Markov Process”, p. 48).

1. How long will the current parameter state (at the tunnel face) persist? It is basically the question of how long, on average, a parameter state persists. This leads to the first set of input parameters: mean lengths of each parameter state.
2. Which parameter state follows, once the current parameter state does no longer persist? This leads to the second set of input parameters: transition probabilities from parameter state to parameter state.

The following chapters will explain the Markov process in more detail, including its mathematical formulation. It will be shown, how the two sets of input parameters mentioned above, govern the Markov process. In order to understand the subject of this

thesis it is however not necessary to read these chapters. The mathematically less inclined can therefore directly proceed to chapter 4 ("The Updating Process", p. 26).

What follows is largely taken from a Master's Thesis by Chan (1981). This thesis represents the first use of the Markov process in the DAT and explains the process exhaustively.

3.2 Basic Elements of the Markov Process

Central to the Markov process are the concepts of state (in the context of the DAT: geotechnical parameter state), state transition and extent. These three basic elements are introduced in the following sub-sections.

State

The states of a random variable are the possible values that it can take. For example, for the geotechnical parameter "lithology", the parameter states r can be defined as follows:

- $r = 1$: Schist
- $r = 2$: Metaquartzite
- $r = 3$: Diorite
- $r = 4$: Quartzite

such that " $r = 3$ " means that the state lithology is Diorite.

State Transition

A geotechnical parameter X at a certain position ℓ can be regarded as a random variable $x(\ell)$. As ℓ increases from 0, $x(\ell)$ changes its value (Figure 3.3). Each of these changes is called a state transition. If at a certain position $x(\ell) = i$, the probability that the next state is j is denoted P_{xij} , the transition probability from state i to j .

Figure 3.3: State Transitions

For example if in Figure 3.2 the current state $x(\ell_0) = k$, then the probability that the next state is 2 is P_{x_k2} . Since the “next state” is always assumed to be different from the present state, it follows that $P_{x_{ii}} = 0$.

Extent

After a parameter $X(\ell)$ has entered into a certain state x_i at ℓ_0 , the interval for which X will remain in state x_i is called the extent HX_i of state x_i at ℓ_0 . HX_i can be thought of as the “horizontal thickness” of state x_i and is depicted in Figure 3.4.

Figure 3.4: State Extent

The DAT use a continuous space Markov process, i.e. one in which space (position) is measured with a continuous scale. One has to define the transition intensity coefficient c_{x_i} of state x_i of parameter X such that $c_{x_i} d\ell$ is the probability that a state transition is made (i.e. the extent terminates) within the infinitesimal interval $d\ell$ (Figure 3.5), given that state x_i exists at the beginning of the interval. Thus the probability that a transition occurs within the interval from $\ell = \ell_1$ to $\ell = \ell_1 + d\ell$ is $c_{x_i} d\ell$.

Figure 3.5: Infinitesimal interval $d\ell$

Neglecting the small probability that there is more than one transition within $d\ell$, the pdf of HX_i can be derived by first considering the cdf (cumulative density function) of HX_i :

Eq. 3-1: $F_{HX_i}(h) = P[HX_i < h]$

If h is divided into m equal segments of infinitesimal length $d\ell$ each, then

$$\begin{aligned} P[HX_i > h] &= P[\text{no transition occurs within } h] \\ &= P[\text{no transition occurs within each of the } m \text{ segments}] \\ &= \lim_{m \rightarrow \infty} (1 - c_{X_i} d\ell)^m \end{aligned}$$

Eq. 3-2: $P[HX_i > h] = \lim_{m \rightarrow \infty} (1 - c_{X_i} \frac{h}{m})^m = e^{-c_{X_i} h}$

From **Eq. 3-1** it follows:

$$P[HX_i > h] = 1 - F_{HX_i}(h), \text{ and therefore}$$

Eq. 3-3: $F_{HX_i}(h) = 1 - e^{-c_{X_i} h}$

By differentiating both sides of **Eq. 3-3**, the pdf of the extent HX_i is given by

Eq. 3-4: $f_{HX_i}(h) = c_{X_i} e^{-c_{X_i} h}$

which is the familiar exponential distribution with mean $1 / c_{X_i}$ and standard deviation $1 / c_{X_i}$. In other words, the extent of a state is exponentially distributed under the single-step memory assumption of the Markov process.

3.3 State Prediction at a Future Point

Based on the Markov process concept the probability of a parameter X being in a certain state at a future point can be calculated. This probability is of great interest since the state prediction of X at a point ahead of the tunnel face is often desired. The situation is depicted in Figure 3.6, in which the probability of X being in state x_j at an interval u from the tunnel face ℓ_0 is wanted, given that the state at the tunnel face $X(\ell_0) = x_i$. This probability cannot be found easily since within the interval u any number of transitions (including no transition) can take place. It is therefore expedient to introduce matrix notations which express calculations in compact form.

Figure 3.6: State Prediction After Interval u

As will be shown later, in order to predict states at “future points”, one needs the transition intensity matrix $\underline{A}_X$ of parameter X such that

$\underline{A}_X = \{a_{Xij}\}$, where

$$a_{Xij} = \begin{cases} -c_{X_i} & (i = j) \\ c_{X_i}P_{Xij} & (i \neq j) \end{cases}$$

and hence

$$\text{Eq. 3-5: } \underline{A}_X = \begin{bmatrix} -c_{X1} & c_{X1}P_{X12} & \dots & c_{X1}P_{X1n} \\ c_{X2}P_{X21} & -c_{X2} & \dots & c_{X2}P_{X2n} \\ \dots & \dots & \dots & \dots \\ c_{Xn}P_{Xn1} & c_{Xn}P_{Xn2} & \dots & -c_{Xn} \end{bmatrix}$$

$\underline{A}_X$ contains c_{X_i} and P_{Xij} and hence defines the Markov process completely. $\underline{A}_X$ is especially useful in making state probability predictions which are discussed in the following sub-sections.

Interval Transition Probability

For the situation shown in Figure 3.6, given that the parameter X is in state x_i at the tunnel face, the probability that X is in state x_j at a distance u behind the tunnel face is required. This problem of state prediction can be solved by introducing the interval transition probability matrix (see also Veneziano, 1980):

$$\underline{V}_X(u) = \{v_{xij}(u)\}, \text{ where}$$

$$v_{xij}(u) = P[X \text{ will be in state } x_j \text{ after an interval } u, \text{ given the present state is } x_i]$$

Generally $\underline{V}_X(u)$ satisfies the forward Kolmogorov differential equation

$$\text{Eq. 3-6: } \frac{d\underline{V}_X(u)}{du} = \underline{V}_X(u) \cdot \underline{A}_X$$

To prove **Eq. 3-6**, let du be a small interval. Then

$$v_{xij}(u+du) = v_{xij}(u) \cdot v_{xij}(du) + \sum_{k \neq j} v_{xik}(u) \cdot v_{xkj}(du)$$

$$\text{Eq. 3-7: } v_{xij}(u+du) = v_{xij}(u) \cdot (1 - c_{xi} du) + \sum_{k \neq j} v_{xik}(u) \cdot a_{xkj} du$$

Thus

$$v_{xij}(u+du) - v_{xij}(u) = -v_{xij}(u) \cdot c_{xi} du + \sum_{k \neq j} v_{xik}(u) \cdot a_{xkj} du$$

Dividing both sides by du and taking the limit as du approaches zero:

$$\text{Eq. 3-8: } \frac{dv_{xij}(u)}{du} = -v_{xij}(u) \cdot c_{xi} + \sum_{k \neq j} v_{xik}(u) \cdot a_{xkj}$$

Eq. 3-8 is identical to **Eq. 3-6** which is in matrix form. The solution of **Eq. 3-6** can be written as:

$$\text{Eq. 3-9: } \underline{V}_X(u) = \exp[u \cdot \underline{A}_X] = I + u \cdot \underline{A}_X + \frac{1}{2} \cdot u^2 \cdot \underline{A}_X^2 + \dots + \frac{1}{m!} \cdot u^m \cdot \underline{A}_X^m$$

In practical cases this series may converge very quickly and one can use only a few terms to get satisfactory accuracy. If convergence is not quick or high accuracy is needed, one

can use the spectral resolution of $\underline{A}_X$ (Cox and Miller, 1965, pp. 183-184). Another way to find a closed-form expression for $\underline{V}_X(u)$ is by using exponential transforms (Howard, 1971, p. 710).

State Probabilities

According to the results of the previous sub-section, if a parameter X is in state x_i at ℓ_0 , the probability that X will be in state x_j at (ℓ_0+u) is $v_{xij}(u)$. When the state of X at ℓ_0 is not known deterministically, but only a probability mass function (Figure 3.1) $P_X(x)$ at ℓ_0 is given, the probability of finding state x_j at an interval u later can still be found. Let $s_{xj}(u)$ be the probability of having state x_j at (ℓ_0+u) :

$$\begin{aligned} s_{xj}(u) &= P[X \text{ is in state } x_j \text{ after an interval } u] \\ &= \sum_{i=1}^n P_X(i) \cdot P[X \text{ is in state } x_j \text{ after an interval } u, \text{ given the present state is } x_i] \end{aligned}$$

$$\text{Eq. 3-10: } s_{xj}(u) = \sum_{i=1}^n P_X(i) \cdot v_{xij}(u) = \sum_{i=1}^n s_{xi}(0) \cdot v_{xij}(u)$$

since $s_{xi}(0) = P_X(i)$ by the definition of $s_{xj}(u)$ above.

To express **Eq. 3-10** in a more compact form, let $\underline{S}_X(u)$ be the row vector of state probabilities such that

$$\underline{S}_X(u) = \{s_{x1}(u), s_{x2}(u), s_{x3}(u), \dots, s_{xn}(u)\}$$

Then **Eq. 3-10** can be rewritten as

$$\text{Eq. 3-11: } \underline{S}_X(u) = \underline{S}_X(0) \cdot \underline{V}_X(u) = \underline{S}_X(0) \cdot \exp[\underline{A}_X \cdot u]$$

Limiting State Probabilities

As the interval u increases, the effect of the present state on the probabilities of future states at an interval u later becomes smaller and smaller. When u approaches infinity, the probability of finding a certain state x_j at an interval u later becomes a limiting constant and is independent of the present state x_i . This limiting constant is called a limiting state probability v_{xj} and is given by

Eq. 3-12: $v_{Xj} = \lim_{u \rightarrow \infty} v_{Xij}(u)$

Furthermore, let $\underline{S}_X = \{v_{X1}, v_{X2}, \dots, v_{Xn}\}$ be the limiting state probability vector. When the transition intensity matrix $\underline{A}_X$ is given, v_{Xj} can be found by first differentiating **Eq. 3-11** with respect to u :

Eq. 3-13: $\frac{d\underline{S}_X(u)}{du} = \underline{S}_X(0) \frac{d \exp[\underline{A}_X \cdot u]}{du} = \underline{S}_X(0) \cdot \exp[\underline{A}_X \cdot u] \cdot \underline{A}_X = \underline{S}_X(u) \cdot \underline{A}_X$

As u approaches infinity, $\underline{S}_X(u)$ approaches $\underline{S}_X$ and **Eq. 3-13** becomes:

$$\frac{d\underline{S}_X}{du} = \underline{S}_X \cdot \underline{A}_X$$

Since $\underline{S}_X$ is a constant

Eq. 3-14: $\frac{d\underline{S}_X}{du} = \underline{S}_X \cdot \underline{A}_X = 0$

which rewrites into a system of linear equations:

Eq. 3-15:

$$\begin{array}{ccccccc} v_{X1}(-c_{X1}) & + v_{X2}(c_{X2}P_{X21}) & + \dots & + v_{Xn}(c_{Xn}P_{Xn1}) & = & 0 \\ v_{X1}(c_{X1}P_{X12}) & + v_{X2}(-c_{X2}) & + \dots & + v_{Xn}(c_{Xn}P_{Xn2}) & = & 0 \\ \dots & \dots & \dots & \dots & \dots & \\ v_{X1}(c_{X1}P_{X1n}) & + v_{X2}(c_{X2}P_{X2n}) & + \dots & + v_{Xn}(-c_{Xn}) & = & 0 \end{array}$$

Equations **Eq. 3-15** are linearly dependent because if all the equations are added together, the left-hand-side vanishes (the coefficients of v_{Xi} vanish) and is identically equal to the right-hand-side. One more equation is thus necessary, which is

Eq. 3-16: $v_{X1} + v_{X2} + \dots + v_{Xn} = 1$

since the parameter must occupy one and only one state at a time.

Thus solving $(n-1)$ equations of **Eq. 3-15** simultaneously with **Eq. 3-16** will lead to the values of v_{Xi} . On the other hand, if $\underline{V}_X$ is already found in closed form (see previous subsection), then v_{Xi} can easily be found by taking the limit as u approaches infinity.

The physical significance of v_{Xi} is that it is the relative percentage of the occurrence of state j . If in a certain region state j (e.g. granite) of a parameter X (e.g. lithology) occurs

70% of the time, then $v_{xi} = 0.7$. For a tunnel of length L in such a region, the expected total length of granite would be $v_{xi} L = 0.7 L$.

3.4 Input Parameters for the Markov Process in the DAT

We have seen in the previous section that the Markov process has the following two basic parameters:

- the transition intensity coefficients c_{xi}
- the transition probabilities from state x_i to x_j , P_{xij}

with $i, j = 1, 2, \dots, n$ for a parameter with n different states.

We have also derived, that the average extent of any parameter state $\ell_{mi} = 1 / c_{xi}$.

For an expert it is easier to estimate the average extent of a parameter state, than its reciprocal value. The input parameters used in the DAT are therefore the average extents (or mean lengths) ℓ_{mi} and the transition probabilities P_{xij} . Remember also that we found $P_{xij} = 0$ if $i = j$.

For more details as to how these parameters are defined in the DAT, refer to section 5.6 ("Parameters for the Markov Process", p. 48).

4. The Updating Process

4.1 General Framework

The general idea of the updating process is to use new information in order to improve a prediction. It is assumed that a prediction has already been made and that new information has become available after this initial prediction was made. The updating process then leads to a updated prediction. The initial prediction is called *prior* and denoted by a ' sign. The updated prediction is called *posterior* and denoted by a " sign.

In very general terms the posterior prediction is a function of the prior prediction and the new information:

$$P'' = f(P'; I)$$

The type of the function depends on the nature of the prediction and the information and is not necessarily a function in the common mathematical sense.

4.2 Bayes' Theorem

Bayes' theorem is a special case of an updating process and a very important principle of probabilistic theory.

It is an updating process applied to probabilities. In contrast to the above paragraph, where P (P' and P'') describes any sort of prediction, P now stands explicitly for a probability. More specifically it stands for the probability that the true state of a certain parameter Z is Z_i . Examples are:

- The probability that the lithology at a certain point along a proposed tunnel alignment is shale.
- The probability that the market demand growth for cars worldwide within the next three years is 5%.
- The probability that a particular research program will be a success.

New information can be obtained by performing any kind of test. If the test is perfect, the true state of parameter Z is known and the updating process is obsolete. However, most tests are not perfect. Tests sometimes say $Z = Z_1$ when in fact $Z = Z_2$.

One of the simplest tests is often to wait. In the 2nd example above, waiting one year is a test. After one year it will be easier to predict the market demand growth for the three year period because only the growth for the remaining two years remains uncertain. The test is however not perfect: Just because growth was 5% in the first year does not mean that it will be 5% for the whole three year period.

If a test X is performed, the probability of its outcome being X_i is denoted by $P(X = X_i)$. It is important to note that $P(X = X_i)$ depends on the true state of the parameter Z . If this is not the case the test is useless because it does not say anything about the parameter Z . More specifically, we are interested in the probability of the result of the test X being X_i , given that the true state of parameter Z is Z_i . This is written as $P(X = X_i | Z = Z_i)$.

Bayes' theorem relates the posterior probability $P''(Z = Z_i)$ to the prior probability $P'(Z = Z_i)$ and $P(X = X_i | Z = Z_i)$.

Eq. 4-1:
$$P''(Z = Z_i | X = X_i) = N \cdot P'(Z = Z_i) \cdot P(X = X_i | Z = Z_i)$$

where: Z :	parameter of interest
Z_i :	state i of parameter Z
X :	test
X_i :	outcome of test
N :	normalizing constant
$P'(Z = Z_i)$:	prior probability of Z_i
$P''(Z = Z_i X = X_i)$:	posterior probability Z_i , given the outcome of the test being X_i
$P(X = X_i Z = Z_i)$:	probability of the outcome being X_i if the true state of Z is Z_i (also referred to as the likelihood of Z_i , given the outcome X_i)

In words:

The posterior probability of the true state of parameter Z being Z_i , given the outcome of the test X is X_i , is the product of a normalizing constant, the prior probability of the true state of parameter Z being Z_i and the probability of the outcome of the test X being X_i , given that the true state of the parameter Z is Z_i .

The normalizing constant is chosen such, that the sum of the posterior probabilities of all possible states of parameter Z equals 1. This is a basic axiom of probability theory.

4.3 Treatment of Different Types of Input Data

Bayes' theorem is widely applicable. There are some limitations, though. It is sometimes difficult to solve an updating problem by using Bayes' theorem – either because the analytical solution becomes very complex or because the updating problem becomes impractical. The remainder of this section presents various contexts in which an updating problem may occur, explains the application of Bayes' theorem and points out practical difficulties.

The parameter describes a probability density function

In chapter 3 ("The Markov Process", p. 23) it was assumed that the lengths over which certain geotechnical parameters persist in a statistically homogeneous zone are exponentially distributed. This parameter "mean length", m , is sought. The test consists of measuring some lengths $l_1, l_2, \dots, l_N$ (during the initial stages of the excavation). The measured lengths are results of imperfect tests. Even if the measured lengths are, say, 50m, 67m, 72m and 79m, there is still a chance that the mean (of all lengths, including those that are yet unknown) is more than 100m.

For every set of measured lengths $l_1, l_2, \dots, l_N$ the probability of measuring these lengths, given that the true parameter of the exponential distribution m is m_i , $P(l = l_1, l_2, \dots, l_N | m = m_i)$ can be evaluated. Together with the prior probabilities $P'(m = m_i)$ for every possible m_i Bayes' theorem can be applied without any particular problem.

There is a continuous range of possible parameter states

In the above example the true state of the parameter m can be any positive number. Consequently, there is a continuous range of possible states m_i of the parameter m . The prior and posterior predictions for the parameter m are therefore in the form of probability density functions (pdf, see Figure 1.2, p. 9) rather than individual probabilities for a finite number of discrete values. Therefore Bayes' theorem needs to be extended to continuous pdf's, which can be done without much difficulty. This is explained in section 8.1 ("Updating with Continuous Probability Distributions", p. 97).

Only the most likely parameter state is predicted

For reasons of practicality, a prediction is sometimes restricted to the determination of the most likely parameter state. It is obvious that such a prediction causes less work, because it is not necessary to predict the probabilities for each parameter state. From the standpoint of probability theory, such a prediction is incomplete, though. If the most likely parameter state is predicted it would also be necessary to say *how* likely it is (i.e. its probability of occurring) and what the other parameter states might be.

Because a prediction as described above is not founded on sound probabilistic principles, Bayes' theorem is not applicable.

An example in the context of the DAT are transition probabilities. (see section 5.6, "Parameters for the Markov Process", p. 48). The transition probability T_{ij} (probability that the parameter state following state i is state j) is the parameter, and the probability values (for example: $T_{i1} = 0.3$, $T_{i2} = 0.5$, $T_{i3} = 0.7$) are the parameter states. There is in fact a continuous range of parameter states within the interval $[0; 1]$. In the DAT however, only the most likely probability value is used as input.

A second example is the association of geometries, ground classes and methods (see section 5.11, "Activities and Equations", p. 53). Here the parameter is the probability A_{ijk} that method k is applied if the tunnel has geometry i and the ground class is j . The probability (e.g. $A_{ijk1} = 0.8$) is the parameter state. Again, only the most likely parameter state (probability A_{ijk}) is used as input, whereas there is in fact a continuous range within the interval $[0; 1]$.

Consequently neither transition probabilities nor the association of geometries, ground classes and methods can be updated by using a mathematical procedure. The updating process must involve human judgment.

The parameter is inherently subject to fluctuations

Consider that the parameter is the daily advance rate of a TBM in a certain geologic formation. This parameter can be measured exactly every day the TBM advances. Even though this is a perfect test, the measured advance rate will be different every day. The reason is that the parameter changes its state (parameter state is a particular value, e.g. 7.9 m/day) daily. These daily advance rates follow a certain probability density function.

This pdf is to be predicted. The exact shape of the pdf is not so important, though. What really matters are the mean and the standard deviation. The situation which was just described applies not only to cycle lengths (see section 5.10, “Methods, Headings, Activity Networks and Cycle Lengths”, p. 52) but also to method variables and general variables (see 5.12, “Method Variables and General Variables”, p. 55).

The prior prediction is a pdf – in the DAT it is a triangular distribution with spikes (see Figure 5.1 middle, p. 38). This is, however, not sufficient. It is also necessary to have a measure of how confident the expert is about the prior prediction. This is illustrated by the following paragraph:

If there is a large number of measurements, the prior prediction does not matter any more – it can be assumed that the frequency plot of the measurements (see Figure 1.1, p. 9) approximates the true pdf very closely. On the other hand, if there are only few measurements, the frequency plot is very unreliable and the expert is more likely to rely on the prior prediction. The question arises immediately, how many experiments are considered to be “a large number” or “a few”. At some number there is a “break even” point, where the expert relies on the prior prediction and on the frequency plot equally. This “break even number” depends on the confidence of the expert in the prior prediction and is therefore a way of measuring this confidence.

Even if one known this “break even number”, the updating process can become mathematically quite complex. Section 7.11 (“Possible Extensions for Updating Variables and Cycle Lengths”, p. 88) will present two extensions to the DAT which would allow one to use an updating process which is based on a mathematical model. The downside of both extensions is the fact that additional input is required when prior predictions are made (for example the “break even number” described above).

Because the definition of prior input was already felt to be quite demanding in earlier applications of the DAT, it was decided not to implement any of the two extensions into the current version of the DAT. Besides it is felt that too many factors cannot be captured by a purely mathematical updating process (e.g. trends in advance rates over time). Human judgment is necessary anyway to sensibly update cycle lengths and variables.

5. Program Input

It is not the purpose of this thesis to explain the program functions of the DAT in detail. The subject of this thesis is rather the updating process. An important part is, however, the application of the theoretical background in the practical context of the DAT, which includes an application example. In order to understand this part of the thesis it is necessary to understand the structure of the program and the different ways in which user input is required. With the updating process in mind, emphasis is particularly on probabilistic input, i.e. input that can be updated.

5.1 General Input Characteristics

The program works absolutely independent of units of measurement. The user has to choose units for basic dimensions like length, time or weight and then keep them throughout the program use. Typical units for length are meters or feet, whereas time is usually measured in days, hours or shifts.

Whenever input is of probabilistic nature, the program works with the following three probability density functions:

- triangular distribution (Figure 5.1, left)
- triangular distribution with spikes (Figure 5.1, middle)
- uniform distribution (Figure 5.1, right)

**Figure 5.1: Probability Density Functions (from left to right):
triangular – triangular with spikes - uniform**

Triangular probability density functions (pdf) have proven to be of great practical value for estimating a parameter. In this respect they are superior to otherwise more widely used pdf's like the normal or log-normal distributions.

The normal distribution (see Figure 5.2) is theoretically undesirable because it puts a finite probability on negative values and because it is symmetric. Experience shows that parameter estimates are quite often skewed to the right. Particularly time estimates may have a long tail end, i.e. the shortest and most likely length of a task can be quite close, whereas the longest possible length of the same task may be much greater because of any kind of unexpected delay.

Figure 5.2: Normal and Lognormal Probability Density Functions
(both have mean = 10 and standard deviation = 5)

The lognormal distribution (see Figure 5.2) is skewed to the right and does exclude negative values. Theoretically, it can therefore be well used for parameter estimates. It's parameters are difficult to estimate, though. Already the mean value can be a problem and the standard deviation is almost impossible to estimate directly. Alternatively it is possible to estimate the minimum and maximum values of the parameter in question. If one assumes that the minimum value is the 5% (or 10%) fractile value and the maximum is the 95% (or 90%) fractile value, three pieces of information (minimum, mean, maximum) are available to determine two parameters. In other words, the lognormal pdf can only fit two of the three pieces of information. This is not satisfactory.

In contrast, the estimation of the three parameters (min – mode- max) of the triangular pdf is very straightforward and intuitively simple. However, without spikes, the probability of the minimum and maximum values are infinitesimally small. This contradicts to some real

world situations, where the minimum and maximum values have a finite probability of occurring. Imagine that the parameter is the daily advance rate of a TBM. The minimum advance rate is 0 m/day and due to various reasons this minimum may occur a significant number of times, for example on 5% of all working days. This situation can't be captured by the triangular pdf and for this reason the triangular pdf with spikes is introduced. The spike places a finite probability on the minimum and/or maximum values.

Sometimes all values within a certain interval are equally likely. This situation is captured by the uniform pdf. It has only two parameters – the boundaries of the interval.

5.2 Tunnel Network and Tunnel Geometries

The tunnel network defines the sequence in which multiple tunnels are excavated. The network consists of nodes, which are connected by arcs. If there is only one tunnel, the “network” is one simple arc with a start node and an end node. More complex networks are interpreted according to the rules of “activity-on-arc” (AOA) networks.

Every arc has a direction (from the start node to the end node) which determines the direction in which the tunnel is excavated.

Consider the following (schematic) tunnel system layout (Figure 5.3). Tunnel excavation starts at the very left and right end of “main tunnel”, at the right end of “auxiliary tunnel” and at the top of “intermediate access shaft”. Once the bottom of “intermediate access shaft” is reached, excavation of the auxiliary tunnel to the right and of the main tunnel in both directions starts.

Figure 5.3: Tunnel System Layout

The corresponding tunnel network looks as follows (Figure 5.4):

Figure 5.4: Tunnel Network

The main tunnel must be modeled with a total of four arcs because it will be excavated from four different access points. Note also that the network is purely schematic and does not say anything about the true geometric relation of the tunnels. The length of an arc does not say anything about the length of the associated tunnel or the time which is necessary to excavate the tunnel. In the example above (Figure 5.4) the left part of the main tunnel could be much longer than part to the right of the shaft.

Excavation at any start node can only start, if either one of the following two conditions is fulfilled:

- the node is not at the same time an end node of another tunnel
- the tunnel for which this node is an end node, has reached the end node, i.e. tunnel excavation is completed.

If there are multiple tunnels ending at a particular node, one must make a difference between "AND" nodes and "OR" nodes. With "OR" nodes it is sufficient that *one* of the tunnels which end at this node are completed in order to start excavation of a tunnel which starts with this node. With "AND" nodes, *all* tunnels must be completed (Figure 5.5).

Figure 5.5: "AND" node and "OR" node

Of course it is possible to delay the excavation start of any tunnel. This is explained in section 5.3 ("Tunnel Information", p. 42).

Even though two tunnels may run parallel and therefore encounter the same geotechnical conditions, they may advance at very different rates or cost. They may even be excavated using completely different excavation methods. To address this issue, “geometry” is introduced. Geometry reflects various tunnel properties such as size, shape or support method. For the purpose of the DAT, two tunnels have the same geometry, if under equal ground conditions, the same design and construction method is applied. For the simulator it is irrelevant what the differences are – it is only important that there are input data (e.g. advance rates, cost information) available for all geometries. In the DAT geometry is therefore nothing more than a number. In the example above (Figure 5.4) the auxiliary tunnel has a smaller excavation area than the main tunnel. The two tunnels are therefore assigned different geometry numbers.

5.3 Tunnel Information

The main parameters determining total construction time and costs are advance rates and costs per meter of each individual method. These parameters are covered later, in chapters 5.11 (“Activities and Equations”, p. 53) and 5.12 (“Method Variables and General Variables”, p. 55). Apart from these main parameters there is various information about tunnels which is relevant to the construction simulation. A tunnel may

- have delays at certain points in time
- have fix costs associated with it
- have time-dependent costs associated with it
- draw resources from or deposit waste to various repositories, depending on how far excavation has advanced
- change tunnel slope (e.g. start excavation with upward slope and later change to downward slope) at certain points along the tunnel

Delays are very uncertain parameters. Both *when* they occur and *how long* they last is very often only vaguely known – if they can be predicted at all. In the DAT the start of the delay is a deterministic input. It is however possible, to make the start time-dependent (e.g. major maintenance of the TBM *every 6 months*) or position-dependent (e.g. change in machine configuration after exiting an initial, *200 m long* soft soil part). The duration of the delay is a probabilistic input, using a triangular pdf (see Figure 5.1, p. 38).

Fix costs are costs that are associated with a tunnel, independent of the actual excavation performance. These include installations at the construction site, most notably machinery, housing and offices for the staff. They can be probabilistic input, also using a triangular pdf.

Time-dependent costs are incurred on a per-time basis. Examples are electricity, fresh water, surveillance and the like. They can be probabilistic input, also using a triangular pdf.

Note that the costs specified here are usually only of secondary importance. The cost associated with the core part of the construction process (i.e. the excavation) should be defined by using method variables.

5.4 Areas and Zones

The top level of the input data structure are areas. An area is a linear structure and typically represents the alignment of the tunnel. In the example in Figure 5.3 there are two areas, one being the main tunnel (including auxiliary tunnel) and one being the access shaft. If two tunnels are parallel and sufficiently close, they are considered to run in the same area. Sufficiently close means that the geotechnical conditions do not differ significantly between the two tunnels.

The definition of areas is deterministic. They have a name and a deterministic length. Areas may also have a parameter set (see section 5.5, “Geotechnical Parameter Sets, Parameters and Parameter States”, p. 45).

Areas are subdivided into zones. Zones are considered to be *statistically* homogeneous. This means that geologic parameters in every individual zone can be described by some kind of a statistical process, which remains the same over the length of the zone.

Typically zones correspond to geologic formations or units. A possible zone pattern for the area “tunnel” in Figure 5.3 could be:

Figure 5.6: Zone Sequence

Zones are a major source of uncertainty. In many cases it is already difficult to know the exact sequence of zones. The exact start and end positions of the zones area often highly uncertain. This input is therefore of probabilistic nature. The *sequence* of zones is deterministic, i.e. it is assumed that it is known which zone follows which one. The extent of zones is probabilistic. The extent of zones can be defined by their length, their end position (relative to the beginning of the area) or both. Both length and end position are specified using a triangular pdf with spikes or a uniform pdf (see Figure 5.1, middle and right). If both length and end position are specified, the DAT ensures that the actual values used are within both specified brackets for every simulation.

#	zone name	what ?	p _{min}	min	mode	max	p _{max}
1	northern sedimentary rocks	end position	0.10	1'500	1'800	2'200	0.10
2	metamorphitic rocks	length	0.0	5'000	-1	6'000	0.0
3	intrusion	length	0.0	500	750	900	0.0
		end position	0.0	7'200	8'100	8'500	0.0
4	southern sedimentary rocks	end position	0.0	10'000	10'000	10'000	0.0

Note that “-1” in mode of zone 2 signifies that the uniform probability density function (see Figure 5.1, right) is to be used.

The spikes in the triangular pdf are particularly useful if the expert is uncertain whether a particular zone exists at all. In such a case the minimum length is 0, and p_{min} is the probability that the zone does not exist.

5.5 Geotechnical Parameter Sets, Parameters and Parameter States

Every zone as defined in section 5.4 (“Areas and Zones”, p. 43) has a set of geotechnical parameters (e.g. jointing intensity, lithology, water inflow). Each geotechnical parameter has a number of parameter states (e.g. sparse, intermediate, intense for the parameter “jointing”). The parameter set defines not only *which* geotechnical parameters are used but also *how* their parameter states are distributed within the zone, i.e. two zones using the same set of parameters do not necessarily use the same parameter set. Hence, only if two zones are statistically identical do they use the same parameter set.

Consider the example in Figure 5.6 (p. 44). Both the zone “sedimentary rocks” and the zone “intrusion” may be described by the parameters “compressive strength” and “jointing intensity” – i.e. they have the same geotechnical parameters. But the “compressive strength” in “sedimentary rocks” may range from “very low” (e.g. shale) to “high” (e.g. quartzitic sandstone), whereas in “intrusion” it may only vary between “high” and “very high”. Because the two zones are geotechnically different, they have different parameter sets.

In contrast, the two zones “sedimentary rocks” at the beginning and at the end of the area “tunnel” may have the same parameter set if they are geotechnically very similar, for example because they have the same geologic origin.

The remainder of this section describes the ways, in which the distribution of the states of a particular geotechnical parameter can be specified. Eventually each zone is subdivided into a sequence of segments.

In this thesis “segment” is used as a generic term to describe an element of a sequence, where the property in question (such as the parameter state or the ground class – see section 5.8, “Ground Classes and Combined Ground Classes”, p. 50), *does* change from one element to the next one but *does not* change within any particular element (see Figure 5.7).

Each change of the geotechnical parameter state is therefore a segment boundary. This subdivision into segments can be specified explicitly by the user or it can be done by the DAT, using a Markov process (see chapter 3, “The Markov Process”, p. 23). It is done for

every geotechnical parameter separately. The following example (Figure 5.7) shows the case where the zone “metamorphic rocks” has the two geotechnical parameters “jointing” and “water inflow”.

Figure 5.7: Sequence of Parameter States in Zone “metamorphic rocks”

There are three ways of defining this sequence of segments and their parameter states:

- If the sequence in which the parameter states follow each other is known, but their extent is uncertain, the input is semi-deterministic. Semi-deterministic input is very similar to the way input for zones (see section 5.4, “Areas and Zones”, p. 43) is made. Again the length or the end position can be specified using a triangular pdf with spikes (see Figure 5.1 middle, p. 38). In contrast to zones it is not possible to define both length and end position.

Consider the case where the geotechnical parameter is “fault” and its states are “present” and “not present”. It may be known that there is a fault within a particular zone, but its exact location and extent may be unknown.

fault not present	fault present	fault not present
-------------------	---------------	-------------------

The sequence of the parameter “fault” has therefore three segments. Input for this case can be for example:

seg.	parameter state	what ?	$p_{\min}$	min	mode	max	$p_{\max}$
1	not present	end position	0.0	1'400	1'650	1'900	0.0
2	present	length	0.0	50	150	300	0.0
3	not present	end position	0.0	3'050	3'200	3'300	0.0

Note that the end position of the last segment must coincide with the end position of the zone, i.e. in the above case the zone would also have the end position pdf with $\min = 3'050$, $\text{mode} = 3'200$ and $\max = 3'300$. Note also that the beginning of the first segment coincides with the beginning of the zone, which in turn is defined by the end position of the previous zone (not shown in the above table).

- If not only the sequence in which parameter states follow each other, but also their extent is known, the input is deterministic. Every segment has an exact end position and a parameter state.

Consider again the example with the geotechnical parameter “fault” under the previous bullet. Input may now look like:

seg.	parameter state	end position
1	not present	1'650
2	present	1'800
3	not present	3'200

Again the beginning of the first segment coincides with the end position of the previous zone. If there is no preceding zone, the begin position of the first segment equals 0.

- In most cases the exact sequence of parameter states is very uncertain. Nevertheless the expert may have a good idea of the average extent of each parameter state. This makes it advisable to use a Markov process to generate the parameter state sequence. For more details about the theoretical background see chapter 3 (“The Markov Process”, p. 23). The way in which the parameters for the Markov process are defined is explained in the following section.

5.6 Parameters for the Markov Process

The Markov process comprises two sets of parameters: mean lengths and transition probabilities.

As was explained earlier (chapter 3, “The Markov Process”, p. 23), it is assumed that the lengths (= extents) of each parameter state are exponentially distributed. The parameter $\lambda = 1 / m$ (m = mean length) of the exponential pdf (Figure 5.8) needs to be predicted. Because m is uncertain, it is predicted using a triangular pdf (Figure 5.9). Note that two pdf's are involved in this process: The expert predicts the mean length (using a triangular pdf) which is in turn the parameter for the exponential pdf used by the Markov process.

Figure 5.8: Exponential Distribution

Figure 5.9: Triangular Probability Distribution for Mean Length m^*

It is also necessary to define the way parameter states follow each other. This is also done probabilistically. The transition probability T_{ij} is the probability that parameter state j follows parameter state i . Transition probabilities are best defined in a matrix.

Consider the geotechnical parameter “jointing” which has the three parameter states “sparse”, “intermediate” and “intense”. The transition probability matrix may look as follows:

T_{ij}	sparse	intermediate	intense
sparse	—	0.7	0.3
intermediate	0.6	—	0.4
intense	0.2	0.8	—

The table reads “from row to column”, i.e. the probability that “intermediate” follows “intense” equals 0.8. Note that there is no transition probability between equal

parameter states because transitions cannot be fictitious (e.g. from “sparse” to “sparse”). Note also that the sum of probabilities in every row must equal 1.0, because the choice of parameter states is exhaustive, i.e. either “intermediate” or “intense” *must* follow “sparse”.

Transition probabilities are not known exactly. Nevertheless, for practical reasons, it was decided not to make this input probabilistic. Program input simply requires the most likely value for every transition probability.

5.7 Information Points

Information points can be the result of any kind of exploration. Exploration will result in perfect or imperfect knowledge about the geotechnical conditions at certain locations along the tunnel. This knowledge influences the Markov process. This is explained in the Masters thesis by Chan (1980), where information points are referred to as “boreholes”.

Input is in the form of a table which is headed by the area and exact location of the information point within the area. The body of the table is a list of various parameter states and the probabilities with which they are believed to exist. If the information is perfect, the probabilities equal 1.0.

Consider again the example in Figure 5.7, p. 46. Assume that geoseismic exploration has been performed in the area “tunnel” at a location approx. 3’100 m from the beginning of the area. The results are imperfect, but based on the exploration data the expert makes the following estimate:

area: tunnel **location:** 3’100

geotechnical parameter	parameter state	probability
jointing	sparse	0.75
	intermediate	0.25
water	little	0.60
	much	0.40

Note that the sum of the probabilities of all parameter states for each parameter must equal 1.0.

5.8 Ground Classes and Combined Ground Classes

Ground classes are the link between the Geology Module (section 2.3, “Geology Module”, p. 14) and the Construction Module (section 2.4, “Construction Module”, p. 18). A ground class is a combination of parameter states (of different parameters). Verbally, ground classes could be defined as follows:

- If jointing is “sparse” and water inflow is “little”, the ground class is “good conditions”.
- If jointing is “intermediate” and water inflow is “little”, the ground class is “medium conditions”.
- Whenever jointing is “intense”, the ground class is “bad conditions”.
- Whenever water inflow is “much”, the ground class is “bad conditions”.

One thus obtains a ground class profile for every zone and thereby along the whole tunnel alignment.

jointing:	sparse	intermediate	sparse	intense	intermediate
water inflow:	little	much	little	much	little
ground class:	good cond.	m.c.	bad c.	good cond.	bad conditions
					medium c.

Figure 5.10: Ground Class Profile (using the example definitions in the text above)

Ground class definitions are an entirely deterministic issue. Consequently no updating is possible in this matter.

The DAT also allows one to define parameters which are independent of a particular zone. A typical example are faults. It may be known that there is fault zone in a certain part of the area, even though it may not be known whether there are sedimentary or metamorphic rocks there. It is thus possible to define a parameter “fault zone” for the whole area. This parameter would have the two states “exists” or “doesn’t exist”.

In this case there is both a zone ground class and an area ground class. The area ground class could be “bad conditions” if there is a fault zone and “neutral” if there is none. Based on the zone ground class and area ground class a combined ground class is defined. Verbally they would be defined as follows:

- If the area ground class is “bad conditions”, the combined ground class is always “bad conditions”
- If the area ground class is “neutral”, the combined ground class is equal to the zone ground class.

area parameter: fault zone	doesn't exist			exists		doesn't exist	
area ground class:	neutral			bad conditions		neutral	
zone ground class:	good cond.	m.c.	bad c.	good cond.	bad conditions		medium c.
combined ground class:	good cond.	m.c.	bad c.	good c.	bad conditions		medium c.

Figure 5.11: Combined Ground Class Profile (using the example definitions in the text above)

The definition of combined ground classes is also deterministic and therefore can't be updated.

Eventually there will be a combined ground class profile for all tunnels in a network. This forms the basis from which the Construction Module derives such things as construction methods and their associated costs and construction times.

5.9 Correlation Numbers

Sometimes geotechnical parameters are correlated. It is for example more likely that water inflow is “much” if jointing is “intense” or, conversely, it is less likely that water inflow is “much”, if jointing is “sparse”. The first example describes a positive correlation and the second example describes a negative correlation.

The DAT can only handle negative correlations. These are defined by correlation numbers, which should not be confused with the commonly known correlation coefficients. The correlation number is a number between 0 and 1. If the correlation number of a combination of parameter states (e.g. jointing “sparse”, water inflow “much”) equals 0, this combination of parameter states does never occur. If the correlation number of a combination equals 1, this combination is not more or less likely to occur than any other combination.

Correlation numbers are not known exactly. Nevertheless, for practical reasons, it was decided not to make this input probabilistic. Program input simply requires the most likely value for every correlation number. Correlation numbers of 1 are assumed as a default value for any combination of parameter states and need not be defined explicitly.

5.10 Methods, Headings, Activity Networks and Cycle Lengths

A method is a series of activities that are performed in order to advance the tunnel by one round or cycle. A round is typically one cycle in drilling-and-blasting or one cycle of a tunnel boring machine. Every activity has its associated cost and time. The order in which activities are performed is defined in an activity network.

Figure 5.12: Activity Network

In its most simple form the activity network only consists of one arc. All tasks that need to be performed in one round are aggregated into one single arc. At the other extreme it is possible to go into great detail and precisely model parallel and sequential activities. The activity network consists of the same elements and follows the same logic as the tunnel network (see section 5.2, “Tunnel Network and Tunnel Geometries”, p. 40), with the only difference that the arcs represent activities rather than tunnels.

The “cycle length” defines how much the tunnel advances during one cycle of the activity network. This can be an uncertain quantity. Cycle lengths are therefore defined probabilistically using a triangular pdf with spikes (Figure 5.1, middle, p. 38).

Many construction methods involve multiple headings. Especially in bad geotechnical conditions the excavation of the tunnel cross-section is separated in two or more parts. These parts are excavated with a certain distance between them.

Figure 5.13: Tunnel Excavation with Heading and Benching

In such a case, separate activity networks must be specified for every heading.

The definition of methods and headings is deterministic. However, most input that is related to these is of probabilistic nature, as described in the following sections.

5.11 Activities and Equations

Activities are the elements of the activity network. (see previous section). The same activity may appear in multiple activity networks, i.e. is used for multiple methods (for example the activity “drill holes” is used both in method “drilling & blasting with shotcrete only” as well as “drilling and blasting with shotcrete and rock bolts”). Activities are associated with two properties, namely the *time* it takes to perform the activity and the associated *costs*. If the Resource Module (section 2.5, “Resource Module” p. 20) is used it is also important to know what resources are used (e.g. steel, concrete) or produced (e.g. muck) during the activity.

The time and cost of an activity are specified in equations, i.e. functions of variables which are defined by the user. For the understanding of this thesis it is not necessary to know the syntax rules of these equations or the functions that can be used. The equations themselves are not part of the updating process, only the variables (see following section) are of probabilistic nature and need to be updated. Still, two functions are worth describing: `round_length()` and `cycle_percentage()`.

The function `round_length()` yields the length a heading advances during a particular cycle. The function `cycle_percentage()` is the quotient [`round length` / `cycle length`]. To understand the difference between round length and cycle length two cases have to be considered:

a) ordinary construction cycle

In an ordinary construction cycle all activities for all headings in a method are performed according to the activity network and the headings advance according to their cycle length. In this case the round length equals the cycle length and `cycle_percentage()` equals therefore 1.

b) truncated construction cycle

A truncated construction cycle occurs if the cycle length of a heading is greater than the remaining extent of a ground class. In Figure 5.14 the last cycle length evaluated for

method “fast” is longer than the remaining extent of ground class “good”. However, the construction with method “fast” cannot continue into the ground class “bad”. The construction cycle is therefore truncated and the round length equals only the remaining length of the ground class “good”. The function `cycle_percentage()` describes what percentage of an ordinary construction cycle has been performed in this truncated cycle.

Figure 5.14: Truncated Cycle

The question arises immediately whether the costs and duration of this truncated construction cycle are evaluated correctly. Usually the costs and time of a cycle depend directly on the length of the cycle. Sometimes this is defined explicitly in equations by using the function `round_length()`, as in the following example (`cost_per_meter` is a method variable):

$$\text{cost} = \text{round_length}() * \text{cost_per_meter} \quad (\text{a})$$

In this case the costs of a truncated cycle are automatically adjusted for the truncation because the function `round_length()` yields the truncated length.

In other cases variables are chosen such that they are directly related to a cycle, for example (`cost_per_cycle` is again a method variable):

$$\text{cost} = \text{cost_per_cycle} \quad (\text{b})$$

Now the costs of all cycles, including the truncated ones, are equal to the variable “`cost_per_cycle`”. In order to adjust costs for truncation the equation should therefore be rewritten as

$$\text{cost} = \text{cost_per_cycle} * \text{cycle_percentage}() \quad (\text{c})$$

In an ordinary cycle `cycle_percentage()` equals 1 and equations (b) and (c) are equivalent. In a truncated cycle equation (c) reduces costs according to the fraction of the actual cycle length compared to the length of a full cycle.

Therefore, in most cases, either the function `round_length()` or `cycle_percentage()` should appear in an equation. There may however be cases where this is not appropriate, such as activities where the lion share of the costs consists of setting up machinery, which is independent of the actual cycle length.

5.12 Method Variables and General Variables

Variables are an important part of activity equations (see previous section). There are general variables and method variables. General variables are globally defined. In contrast, method variables need to be defined for every method separately. Remember that activities can be used in multiple methods. However, the same activity may not cost the same or take the same amount of time in different methods. Therefore the cost (or duration) of a particular activity is usually a method variable. The DAT use the variable value depending on the method in which the activity occurs.

Consider as an example the following cost equation for the activity “rockbolting”:

$$\text{rockbolt_cost} = \text{bolts_in_method} * \text{cost_per_bolt}$$

The variable `cost_per_bolt` is a general variable because it is independent of the method used (assuming the same type of bolt is used among different methods). The variable `bolts_in_method` clearly depends on the method and is defined as a method variable for every method which has the activity “bolting” in its activity network.

Both general and method variables are often uncertain. They are defined using a triangular pdf with spikes (Figure 5.1, middle, p. 38).

5.13 Association of Geometries, Ground Classes and Methods

At any point along the tunnel, the Construction Module (section 2.4, “Construction Module”, p. 18) selects a method based on the current ground class and the geometry of the tunnel. The method determines the actions to be performed in order to excavate the

next round (cycle). This decision which method to choose need however not be deterministic. It is possible that even though geometry and ground class are the same, the selected method is not. The user must specify probabilities with which certain methods are selected, given a particular ground class and geometry. The input is a simple table (Figure 5.15).

geometry	ground class	method	probability
1	bad conditions	steel arches / large Ø	0.75
	bad conditions	shotcrete + bolts / large Ø	0.25
	good conditions	shotcrete only / large Ø	1.00
	...	...	...
2	bad conditions	shotcrete + bolts / small Ø	1.00
	...	...	...

Figure 5.15: Association of Geometry, Ground Class and Method

These probabilities are of course uncertain themselves. But again, for practical reasons, input is quasi-deterministic and requires only the most likely value.

5.14 Learn Parameters

It is frequently observed, that construction performance is low at the beginning and increases gradually to reach a certain limit. This phenomenon is known as the learning curve. There are many reasons for it:

- Workers must get used to the peculiarities of the actual project
- Workers must get used to each other in order to have good team work
- Machinery may have teething troubles.
- The project organization and the associated flow of information is not yet streamlined

In the DAT this learning curve can be specified simply by defining some points on it. The learning curve is difficult to predict and even in hindsight it may be difficult to determine.

Consider for example a learning curve for daily advance rates in a certain geotechnical environment. Because of various influences, the advance rate has an inherent variability. Measured advance rates therefore exhibit considerable scatter.

Because efficiency can be below 100% both due to learning curve effects or due to the inherent variability in the advance rates it can be difficult to extract the learning curve from measured data.

Nevertheless input of the learning curve is deterministic. It is the best estimate based on past records or the experience of experts.

Figure 5.16: Learning Curve

5.15 Method Change Delays

Sometimes switching between methods also causes delays. A new machine may have to be installed or there may be a “mini learning curve” because workers must get used to the new method. The user can therefore specify “method change delays”. This is done deterministically in a simple table:

from method	to method	delay
TBM w/ shotcrete only	TBM w/ shotcrete and bolts	0.3
TBM w/ shotcrete only	TBM w/ steel arches and shotcrete	0.5
TBM w/ shotcrete and bolts	TBM w/ steel arches and shotcrete	0.5
...	...	...

Figure 5.17: Method Change Delays Table (delay in days)

“Delay” occurs whenever the method changes from “from method” to “to method”.

5.16 Repositories, Resources and Stock Levels

The Resource Module which is currently implemented in the DAT has only limited capabilities. All input is deterministic and consequently no updating is necessary. The Resource Module is therefore not treated any further in this thesis. For a detailed discussion of the subject refer to Halabe (1995).

5.17 Daily Fluctuations vs. Long-Term Averages

The values of cycle lengths (section 5.10, "Methods, Headings, Activity Networks", p. 52), method and general variables (section 5.12, "Method Variables and General Variables", p. 55) are usually subject to daily fluctuations. This is, however, only one source of uncertainty. The average value around which they fluctuate can be equally uncertain. In fact, the effect of daily fluctuations in long tunnels cancels out and only the uncertainty about the long-term average is important. On the other hand, in short tunnels, the effect of daily fluctuations can outweigh the uncertainty about the long-term average.

It is evident that in some cases daily fluctuations and in other cases long-term averages are important. The DAT can accommodate both situations with a switch, which can have the positions "daily fluctuation" or "long-term average". This switch can be chosen for each method individually. This allows one, for example, to use "daily fluctuation" for a short access shaft and "long-term average" for (longer) tunnels connecting access shafts.

In the position "daily fluctuation" all input that is made for cycle lengths, general and method variables is considered to represent the uncertainty about daily fluctuations. Consequently, when the DAT run simulations, these values are picked from the probability density functions (pdf) in every round (often one round represents one day) of the simulation anew.

If the switch is in position "long-term average", the pdf's for cycle lengths, general and method variables are considered to represent long-term averages. The simulator therefore picks these values from the pdf's only once at the beginning of each simulation. As a consequence, *during any particular simulation*, the time and costs associated with performing one cycle of such a method, as well as the cycle length itself, remain constant.

Since general variables can be used by several methods it is important to make sure that all methods using one particular general variable have the same switch position. It is

obvious that a general variable cannot have the meaning of “long-term average” for one method while at the same time having the meaning of “daily fluctuation” for an other method.

There are boundary cases, where the daily fluctuations and their long-term average are both significant sources of uncertainty. With the current version of the DAT, this case is not well provided for and the user must decide on one of the two options mentioned above. Section 7.11 (“Possible Extensions for Updating Variables and Cycle Lengths”, p. 88) presents a possible extension of the DAT (in “Extension 2: Treat daily fluctuations and long-term averages separately”) which would allow one to incorporate both sources of uncertainty into the DAT. The section also discusses, why it was decided *not* to implement this extension.

6. Observations

The updating process is basically a comparison of prior input with actual observations made within a certain timeframe. This chapter describes what data must be recorded and discusses several issues regarding the data collection process.

6.1 Observed Parameters

Generally speaking, all those parameters for which one wants to refine the input, i.e. which one wants to update, need to be observed.

Geologic / Geotechnical Parameters

First of all one has to record the beginnings and ends of all zones. This is done by listing the encountered end positions of the zones. The beginning of each zone is defined by the end of the previous zone. It is also necessary to record the actual position of the tunnel face as the “current end position” of the zone the tunnel is in right now. Although this does not yet tell us the exact end position of that particular zone, it tells us that the zone exists at all and also its minimum end position. Of course, this number has to be marked accordingly in order not to confuse it with a true end position. (see Figure 6.2, 5'561 being the current position of the tunnel face)

Figure 6.1: Example of Tunnel Advancing Forwards (left arrow) and Backwards (right arrow)

<i>zone</i>	<i>end position</i>	<i>status</i>
1. northern sedimentary rocks	2'550	observed
2. metamorphic rocks	4'674	observed
3. intrusion	5'561	current
4. southern sedimentary rocks		

Figure 6.2: Observed Zones (tunnel advancing forwards)
(unit in this example: meters)

Note that if a tunnel is excavated backwards (in the negative direction of the coordinate system of the area, see Figure 6.1), the zone end is the location where the zone is entered (see Figure 6.3). In this case the current location of the tunnel face does not appear on the table.

<i>zone</i>	<i>end position</i>	<i>status</i>
1. northern sedimentary rocks		
2. metamorphic rocks		
3. intrusion	7'850	current
4. southern sedimentary rocks	10'000	observed

Figure 6.3: Observed Zones (tunnel advancing backwards)
(unit in this example: meters)

Within each zone it is necessary to record the encountered geotechnical parameter states. This is done by recording the parameter state, beginning, end and status of each segment (a segment being a “slice” of the tunnel between two state transitions, Figure 6.4). The records are somewhat redundant because usually the end of one segment coincides with the beginning of the next one and it would be sufficient to record only one of the two. There are some reasons why both end and beginning should be recorded:

- One can see in the previous paragraph about zones that the current location of the tunnel face does not appear in the records if only end positions are recorded and if the tunnel advances backwards. In this case it is *necessary* to record the beginning too.
- For a tunnel advancing backwards it is not entirely clear whether the end of a segment is where the segment is exited (i.e. the smaller coordinate) or where the segment ends

in the direction of the coordinate system (i.e. the larger coordinate). Consequently it is easy to confuse the two. However, if both of them are recorded one can always define the smaller number to be the beginning and the larger one to be the end – independent of how they were recorded.

- There are many reasons why the records can be incomplete or erroneous, especially since usually many different people are involved in recording and transmitting the data. Many errors can be detected by checking whether the end of one segment really does coincide with the beginning of the next one.

As with zones, the current position of the tunnel is marked by setting “status” to “current”. Note also that such a table has to be created for every geotechnical parameter.

zone: *intrusion*

<i>parameter state</i>	<i>begin position</i>	<i>end position</i>	<i>status</i>
...	...	...	...
jointing: intense	4'550	4'610	observed
jointing: medium	4'610	4'875	observed
jointing: slight	4'875	5'120	observed
jointing: medium	5'120	5'561	current

Figure 6.4: Observed Parameter States (unit in this example: meters)

Construction Methods

In every tunnel which is being excavated one has to record the applied (construction) methods. The table follows the same logic as the one above (Figure 6.4) for geotechnical parameter states. It is however not necessary to record the status. As far as methods are concerned, it is unimportant where the tunnel face is.

Beside the record of applied construction methods it is also important to record cycle lengths as well as values of method variables and general variables. For all of them it is not sufficient just to record a (possibly large) number of individual values. It is just as important to record the time (e.g. working days since construction started) during which these values were observed (Figure 6.6). This is the only way to discover trends with time.

These are likely to exist and can be very important. In particular it will be possible to determine learning curves.

tunnel: *shaft – south exit (main tunnel)*

<i>method</i>	<i>begin position</i>	<i>end position</i>
...	...	...
TBM with steel arches	4'550	4'630
TBM with rock bolts	4'630	4'910
TBM with shotcrete only	4'910	5'120
TBM with rock bolts	5'120	5'561

Figure 6.5: Applied Construction Methods (unit in this example: meters)

method: *TBM with shotcrete and rock bolts*

<i>heading</i>	<i>time of observation</i>	<i>observed cycle length</i>	<i>variable name</i>	<i>time of observation</i>	<i>observed value</i>
1	110	2.5	cost_per_meter	110	5'400
	111	2.5		111	5'315
	112	2.6		112	5'175
	113	2.6		113	4'980
	114	2.7		114	4'950

Figure 6.6: Record of Cycle Lengths (left) and Method Variable Values (right)
(units in this example: days, meters/day and \$/meter)

Note that a method can have multiple headings, each with different cycle lengths as well as multiple variables.

Delays

There are two kinds of delays that need to be recorded: tunnel delays and method change delays.

Tunnel Delays

Tunnel delays can be defined in the DAT as being time-dependent (i.e. delay starts at a certain point in time) or position-dependent (i.e. delay starts at a certain location along the

tunnel). However, a delay that is observed is neither time- nor position-dependent – it is just there. In order to be capable to match observed delays to initial predictions it is necessary to record both time and location where the delay started, together with its duration (Figure 6.7). If a delay is still going on currently, it must be marked accordingly. The duration up to now is then a minimum value for its total duration.

<i>tunnel</i>	<i>start time</i>	<i>start position</i>	<i>duration</i>	<i>still going on ?</i>
shaft – south exit (main)	50	3'000	50	no
shaft – south exit (main)	130	4'550	20	no
shaft – south exit (aux)	50	3'000	25	no
shaft – south exit (aux)	315	5'561	15	yes

Figure 6.7: Observed Tunnel Delays (units in this example: days, meters)

Method Change Delays

Method change delays occur when the construction method must be changed. They can have various causes (see section 2.4, “Construction Module”, p. 18). Similar to cycle lengths there may be a trend with time (for example something like a learning curve) and therefore the time of every observation must be recorded too.

<i>from method</i>	<i>to method</i>	<i>time of observation</i>	<i>observed duration</i>
TBM with shotcrete only	TBM with rock bolts	120	0.25
TBM with rock bolts	TBM with steel arches	135	0.40
TBM with steel arches	TBM with shotcrete only	151	0.35
TBM with shotcrete only	TBM with rock bolts	179	0.20
...	...	...	...

Figure 6.8: Observed Method Change Delays (unit in this example: days)

6.2 Data Collection and Project Database

Data collection is the most work intensive part of the updating process. In fact, the time and resources required to collect all necessary data and feed them into the DAT can be prohibitively high.

One has to be aware though, that during construction many data are recorded anyway in some way or another. Nevertheless there is some additional effort required in connection with the DAT:

- 1) some data need to be collected which would not be collected otherwise
- 2) the data which are collected anyway must be stored according to a framework / structure which is compatible to the one used by the DAT
- 3) the data which are usually stored somewhere else must be transferred to the DAT

The **first point** depends very much on the actual project. It can involve a significant effort but it may also not be a problem at all. Generally speaking it will probably become less important in the future because with modern data processing tools there is a tendency to collect more and more data.

The **second point** is merely a question of good planning. It is always difficult and requires a lot of work to change the structure in which data are stored. This point has a technical and an organizational aspect:

- A very basic question is of course, whether data are stored electronically or on paper. Nowadays most data are probably stored electronically. Nevertheless there may be graphic intensive data (such as geologic profiles, jointing patterns) that are still collected manually on paper. But even if the data are stored electronically there are many possible formats that can be used (text documents, pictures, data tables, databases). This also plays into the third point.

- At least as important as the technical side is the organizational one. The data must be stored in a structure that reflects the structure of the observations (see previous section) in the DAT. In particular the level of detail should be the same.

For example if the parameter “jointing” in the DAT has the three states “intense”, “intermediate” and “sparse”, the data must be collected using this same system of subdividing “jointing” into three states.

Another example are variables: If the activity network includes <drilling – blasting – remove muck> and the costs are assigned to each of these activities separately (through variables) then these costs must also be recorded separately during construction.

These decisions about what data will be collected and how they will be collected must be made before actual construction starts. Data collection must be well planned and the necessary tools must be provided in time.

The **third point** is a mostly technical question. In whatever format the data are stored, they have to be transferred to the DAT.

The worst alternative is to type the data manually in the data entry forms of the DAT. This requires a prohibitively large amount of work.

It is considered a major advantage of this new version / extension of the DAT that it is programmed based on Microsoft® Access®. Being part of the widely used MS Office family it has the great benefit that data can be copied from other Office applications (notably Excel®) simply by using copy & paste operations. This provides an easy way to transfer large amounts of data in a short time.

If the data of a project are stored in a database it is possible to connect the DAT directly to this database. Initially this requires some additional programming effort. After this, however, the data transfer can be completely automated. For large projects which go through the updating process several times it may well be worth the effort.

All the above issues are best dealt with if data collection is discussed early on in the project and on a high executive level.

The author strongly recommends the creation of a project database which stores all relevant data of the whole project in one central database. This ensures that data are collected consistently by the various companies and persons involved in the project. The process of creating the database itself (if done responsibly) inevitably triggers the

discussion of the points mentioned above. In addition to this, such a database provides easy access to project data to all persons involved in the project. It is no longer necessary that these people are physically in the same place as the database, since data can also be queried over an intranet or even over the world wide web, while sensitive data can still be protected by access control.

7. The Updating Process in Detail

7.1 Recapitulation of General Framework

The general idea of the updating process is to use new information in order to improve a prediction. It is assumed that a prediction has already been made and that new information has become available after this initial prediction was made. The updating process then leads to an updated prediction. The initial prediction is called *prior* and denoted by a ' sign. The updated prediction is called *posterior* and denoted by a " sign.

In very general terms the posterior prediction is a function of the prior prediction and the new information:

$$P'' = f(P'; I)$$

The type of the function depends on the nature of the prediction and the information and is not necessarily a function in the common mathematical sense. The updating process can be applied to any situation where a prior prediction is updated with new information. It very often – if not always – involves human judgment. In the DAT it is applied to all input data which is of probabilistic nature.

Because the updating process usually involves human judgment, it is very important to present the necessary information in a condensed and practical manner. In fact, one of the main problems when updating predictions for such a complex system as a tunnel or tunnel network, is not to lose oversight over all available data. Besides, oftentimes new information cannot be used in its raw state and therefore must be processed in some way or another. This processing of information and its presentation together with prior predictions is the main purpose of the Updating Module of the DAT.

Bayes' theorem is a special case of an updating process and a very important principle of probabilistic theory. It is an updating process applied to probabilities. In contrast to the above paragraph, where P (P' and P'') describes any sort of prediction, P now stands explicitly for a probability. Bayes' theorem can, but does not have to involve human judgment, i.e. it can be performed purely computationally on the basis of the prior input

and the new information. In the DAT it is only used to update the “mean length” parameter of the Markov process.

7.2 Parameters of the Markov Process

The Markov process (chapter 3, “The Markov Process”, p. 23) has two sets of parameters, the mean lengths of every geotechnical parameter state and the transition probabilities from one state to another.

Mean Length

The mean lengths are the only parameters that are currently updated using Bayes’ theorem. More precisely, Bayes’ theorem is used to make a suggestion to the user, as to what the posterior (= updated) prediction should be. However, the user can always override the program’s suggestion by using his or her judgment. For more details on the mathematical aspect of this application of Bayes’ theorem, see section 8.2 (“Updating Markov Parameters”, p. 98).

Generally speaking, the posterior prediction is influenced by the prior prediction and by the observed segment lengths of that particular geotechnical parameter state. It is therefore important to present this information to the user.

The window in which the “mean length” parameter is updated consists of a left and a right part (see Figure 7.1).

In the top left part, the user selects the parameter set and the geotechnical parameter (of the selected parameter set) which has to be updated. This is done by pull-down menus which show a list of available parameter sets and geotechnical parameters, respectively. The list in the left part presents the sequence in which the states of the selected geotechnical parameter have occurred, together with the length of each segment. It also lists the zone (or area) in which this state was observed.

Remember that a parameter set can be used in multiple zones if these zones are believed to be statistically similar (see section 5.5, “Geotechnical Parameter Sets, Parameters and Parameter States”, p. 45). This zone/area information can be important to decide whether it is justified that the same parameter set is used for two or more zones. If the observed lengths in one zone are consistently different from those observed in an other zone, this

can indicate that they are not statistically similar. This, in turn, is a reason not to use the same parameter set for these two zones.

Update Parameters for Markov Generation Mode

[Print Preview / Print](#)

Parameter Set: **4**
Parameter: **Jointing**

Parameter State: **intense**

[Back to Summary](#)

[Open Transition Probabilities](#)

[Calculate Posterior](#)

Zone/ Area	Parameter State	Endposition	Length
Z0	little	150.0	150.0
Z0	medium	210.0	60.0
Z0	intense	230.0	20.0
Z0	little	330.0	100.0
Z0	medium	410.0	80.0
Z0	intense	450.0	40.0
Z0	medium	500.0	50.0

	Min	Mode	Max	Start Prob
Prior:	10.0	20.0	80.0	0.00
Posterior:	37	24.6	74.5	0.00

Figure 7.1: Sample Window to Update "mean length" Parameter

The right part of the window displays information for one particular state of the geotechnical parameter selected to the left. In the upper part it shows the prior and posterior predictions for the "mean length" parameter (do not confuse geotechnical parameters like 'Jointing' with the parameters of the Markov process, mean length and transition probability). Remember that these "mean length" parameters are triangular distributions, as can be seen in the graph in the lower part. The user can, of course, change the values of the posterior prediction, which will immediately change the graph. The graph shows the prior triangular pdf, the likelihood function and the posterior triangular pdf. The likelihood function indicates how likely each value for the "mean length" parameter is, given the observed values. Note that it is solely based on the observations and not on the prior prediction.

The "calculate posterior" button calculates the posterior pdf by applying Bayes' theorem to the prior and the observed values and fills the calculated values in the appropriate fields in

the upper right part. As a default, the DAT starts with suggesting these calculated values as posterior input.

Transition Probabilities

The transition probabilities are uncertain parameters. Nevertheless, for practical reasons, their input in the DAT is deterministic, i.e. only one value (the most likely one) for each transition probability is used as input. Still, because they are not known precisely, they must be updated. The predicted value can be compared to the observed one, which requires human judgment. Transition probabilities are not observed directly. Instead the sequence in which geotechnical parameter states occur is recorded and this information must be processed. From the recorded sequence of states, the transitions from one state to an other can simply be counted and from the counted numbers the observed transition probabilities can be obtained.

Assume the geotechnical parameter “lithology” has the states ‘gneiss’, ‘granite’ and ‘shale’. The counted transitions are: gneiss → granite: 5; gneiss → shale: 3; granite → shale: 4; and so on. The observed transition probability from gneiss to shale can be found as:

$$T_{\text{gneiss} \rightarrow \text{shale}} = \frac{\text{number of transitions from gneiss to shale}}{\text{number of transitions from gneiss to any state}} = \frac{3}{5 + 3} = 0.375$$

This number is compared to the prior value (see Figure 7.2). In order to arrive at a posterior value, one must weight the prior value and the observed value. This weighting is done purely by the user.

This weighting involves human judgment. It is obvious that the observed value is weighted more strongly, if it is based on a large number of observed transitions. The total number of observed transitions from every state (i.e. the denominator in the above equation) is therefore a necessary piece of information in the updating process.

In Figure 7.2, only two transitions were observed from all three parameter states. Therefore the observed transition probabilities are not relevant and the prior probabilities are also taken as posterior input.

Imagine however, that 20 transitions had been observed, instead of only 2. In such a

case one would weight the observed values strongly and the posterior input would be almost exclusively chosen on the basis of the observed values.

As a default the DAT suggest the prior transition probabilities also as posterior transition probabilities.

Update Parameters for Markov Generation Mode

Print Preview / Print

Parameter Set: 4
Parameter: Jointing

Zone/ Area	Parameter State	Endposition	Length
Z0	little	150.0	150.0
Z0	medium	210.0	60.0
Z0	intense	230.0	20.0
Z0	little	330.0	100.0
Z0	medium	410.0	80.0
Z0	intense	450.0	40.0
Z0	medium	500.0	50.0

Update Transition Probabilities					
Show Trans. Matrix					
Print Prev. / Print					
from parameter state	# of transitions	to parameter state	prior	observed	posterior
intense	2	little	0.20	0.50	0.20
		medium	0.80	0.50	0.80
little	2	intense	0.10	0.00	0.10
		medium	0.90	1.00	0.90
medium	2	intense	0.50	1.00	0.50
		little	0.50	0.00	0.50

Figure 7.2: Sample Window to Update Transition Probabilities

In section 7.12 ("Possible Extension for Updating Transition Probabilities", p. 95) the author proposes an extension which would allow the DAT to suggest posterior transition probabilities on its own, based on computations involving both prior input and observed transitions.

7.3 Variables and Cycle Lengths

Variables and cycle lengths are treated in absolutely the same manner. In this section, all comments about variables also apply to cycle lengths.

Variables are subject to daily fluctuations. In addition to this their long-term average is also uncertain. Program input (prior and posterior) either tries to predict daily fluctuations or their long-term average (see section 5.17, "Daily Fluctuations vs. Long-Term Averages", p. 58) by specifying a triangular probability density function with spikes (see Figure 5.1 middle, p. 38). During construction daily values are recorded. From these data a frequency plot (see Figure 7.3, upper right graph, titled "Observed") can be drawn and their average can be calculated.

The updating process is slightly different, depending on whether daily fluctuations or long-term averages have been specified as prior input.

In the case of daily fluctuations, the updating process is basically a comparison between the prior prediction of the triangular pdf with spikes and the frequency plot of observed data. This comparison is given in the upper right graph of Figure 7.3, where the prior pdf, the frequency plot and the posterior pdf are drawn. The upper left part shows the method as well as the variable name (if variables are updated) or the heading number (if cycle lengths are updated).

If one is dealing with long-term averages, the prior prediction must be compared to the average of the observed values. Although the frequency plot provides some insight into the variation in daily values, for the purpose of the updating process, it is only the average that counts.

It would be misleading to indiscriminately create a frequency plot and calculate the average of all observed data. Oftentimes variables show a trend with time. By creating a frequency plot only, such a trend would not be revealed. It is therefore necessary to create a second plot which shows the observed values together with the time at which they were observed. In the example of Figure 7.3 the value-time-plot (lower half) reveals that

- this particular method has been applied twice so far, from days 0 to 21 and days 120 to 144
- that the measured cycle lengths of the two periods are distinctly different
- that the cycle lengths in the first few days are somewhat lower, very much like a learning curve (see section 5.14, "Learn Parameters", p. 56)

This is the point at which human judgment starts to influence the decision as to what the posterior pdf should look like. The expert may judge that some of the data are not relevant and therefore should not be included in the frequency plot. The DAT therefore allows one to filter out all data that were observed before a certain time limit. This time limit can be set by the user in the upper center of Figure 7.3. In the example in Figure 7.3 all observations before time = 120 are discarded and only data after time = 120 are included in the frequency plot. Note that in the lower graph the discarded data points are plotted with little crosses instead of diamonds. Different rationales lead to different decisions as to which data should be discarded. Some examples:

- the data are considered to be equally relevant, i.e. it is considered pure coincidence that the values in the second period are substantially higher
→ discard no data (set time limit to 0)
- the data in the first period are not considered relevant because some improvements to the excavation process have led to the increase in the second period which can be maintained in the future
→ discard all data of the first period (as shown in Figure 7.3)
- in addition to the previous point the five first days of the second period are considered to be part of a learning curve for the improved method and are therefore not relevant for the remainder of the excavation
→ discard all data before time = 125 (set time limit to 125)

Update Cycle Lengths

[Print Preview / Print](#)

[Back to Summary](#)

Method: **Shaft D&B w/ Shotcrete**

Updated: ☐

Heading: **0**

Cycle Length	Prior	Posterior
prob. min.	0.00	0.00
min	3.0	2.6
mode	3.4	3.9
max	3.7	4.7
prob. max.	0.00	0.00
mean	3.37	3.73

discard all data observed before
timelimit:
120
obs. average:
3.65

Figure 7.3: Sample Window to Update Variables or Cycle Lengths

So far it was only discussed which observed data points are relevant and should be included in the frequency plot and the calculation of the average. The updating process itself is a comparison of this frequency plot or observed average value with the prior pdf.

This, once more, involves human judgment. The question is how the prior pdf and the frequency plot / observed average should be weighted against each other.

In the case where the prior input represents daily fluctuations, if the frequency plot is the result of a long series of observed data one would probably choose the posterior pdf such that the frequency plot is matched very closely. Conversely, if there are only few data, one would rather rely on the prior prediction.

In the case where the prior input represents long-term averages, if there is a long series of observed data, one would be quite confident that the future long-term average is close to the observed average and the posterior pdf would be chosen very narrowly around the observed average. On the other hand, if there are only few data, one would still rely on the prior prediction.

Practically the updating is done by changing the values in the “posterior” column (top left part in Figure 7.3). Every time these values are changed, the graph on the right hand side and the mean of the posterior pdf (at the bottom of the “posterior” column) is immediately adjusted. The posterior values can be changed until the user is satisfied with the resulting graph and mean of the posterior pdf. In Figure 7.3 the posterior values were adjusted such that the posterior graph matches the frequency plot of observations as well as possible, i.e. in this example the pdf represents daily fluctuations.

With the current input data structure of the DAT, the decision on how to define the posterior pdf can only be made by human judgment. However, section 7.11 (“Possible Extensions for Updating Variables and Cycle Lengths”, p. 88) proposes two possible extensions to the DAT which would allow the program to make suggestions for the posterior pdf. Even those extensions would still require the expert to set the time limit, i.e. human judgment would still be necessary in the updating process.

7.4 Semi-Deterministic Definition of Geotechnical Parameters

Updating semi-deterministic input is almost exclusively a matter of human judgment. The prior input is a sequence of parameter states where each segment has uncertain length and end position. This sequence is compared to the actually observed sequence (Figure 7.4, p. 76). When the DAT process observed data, the program tries to match the observed sequence with the prior one. If it can match an observed segment to a predicted

one, it changes its status (see Figure 7.4, first column of right part) to “Observed”. If the matched segment is the one in which the tunnel face is currently, status becomes “Current”. All unmatched segments are “Posterior”.

The matching process starts at the beginning of the observed sequence. It compares the parameter state and the end position of the first observed and the first prior segment. If the parameter state is the same and if the observed end position is within the bracket [min end pos; max end pos] of the prior input (plus an additional tolerance of $\pm 10\%$ of the length of the observed segment), the DAT consider the two to be a successful match. The program then moves to the next segment on both sides and so on.

By default the DAT suggest the following input for posterior segments: (the figures are examples for a geotechnical parameter with three states: high – medium – low)

Update Parameters for Semi-Deterministic Generation Mode

[Print Preview / Print](#)

[Back to Summary](#)

Parameter Set:		0		Parameter:		Faulting								
Order Parameter State		Gen. Mode	Prob.	Min	Mode	Max	Prob. Max	Gen. Mode	Prob.	Min	Mode	Max	Prob. Max	
1	none		0.00	0.0	0.0	0.0	0.00	Observed		0.00	450.0	450.0	450.0	0.00
☒	insert segment	End Pos	0.00	300.0	400.0	500.0	0.00	Both		0.00	450.0	450.0	450.0	0.00
2	slight		0.00	10.0	50.0	100.0	0.00	Observed		0.00	30.0	30.0	30.0	0.00
☒	insert segment	Length	0.00	0.0	0.0	0.0	0.00	Both		0.00	480.0	480.0	480.0	0.00
3	severe		0.00	20.0	30.0	50.0	0.00	Observed		0.00	20.0	20.0	20.0	0.00
☒	insert segment	Length	0.00	0.0	0.0	0.0	0.00	Both		0.00	500.0	500.0	500.0	0.00
4	slight		0.00	20.0	50.0	80.0	0.00	Observed		0.00	30.0	30.0	30.0	0.00
☒	insert segment	Length	0.00	0.0	0.0	0.0	0.00	Both		0.00	530.0	530.0	530.0	0.00
4.5	none		0.00	0.0	0.0	0.0	0.00	Observed		0.00	1,750.0	1,750.0	1,750.0	0.00
☒	insert segment	Length	0.00	0.0	0.0	0.0	0.00	Both		0.00	2,280.0	2,280.0	2,280.0	0.00
4.75	slight		0.00	0.0	0.0	0.0	0.00	Observed		0.00	170.0	170.0	170.0	0.00
☒	insert segment	Length	0.00	0.0	0.0	0.0	0.00	Both		0.00	2,450.0	2,450.0	2,450.0	0.00
5	none		0.00	0.0	0.0	0.0	0.00	Current		0.00	650.0	850.0	1,050.0	0.00
☒	insert segment	End Pos	0.00	3,100.0	3,300.0	3,500.0	0.00	End Pos		0.00	2,800.0	3,300.0	3,500.0	0.00
6	slight		0.00	100.0	150.0	200.0	0.00	Posterior		0.00	100.0	150.0	200.0	0.00
☒	insert segment	Length	0.00	0.0	0.0	0.0	0.00	Length		0.00	3,200.0	3,450.0	3,700.0	0.00
7	severe		0.00	50.0	80.0	100.0	0.00	Posterior		0.00	50.0	80.0	100.0	0.00
☒	insert segment	Length	0.00	0.0	0.0	0.0	0.00	Length		0.00	3,250.0	3,530.0	3,800.0	0.00
8	none		0.00	0.0	0.0	0.0	0.00	Posterior		0.00	3,100.0	3,570.0	3,950.0	0.00
☒	insert segment	End Pos	0.00	6,900.0	7,100.0	7,200.0	0.00	End Pos		0.00	6,900.0	7,100.0	7,200.0	0.00

Figure 7.4: Sample Window to Update Semi-Deterministic Input

During the updating process it is possible that segments are expected to occur that were initially not expected. In this case a new segment must be inserted into the sequence. This can be done with the “insert segment” button in Figure 7.4.

- If the prior segment is matched with an observed one, then its posterior input is the end position of the observed segment.

- If a prior segment is matched with an observed one, which has status “Current”, then the posterior input for that segment is the same as the prior input. However, lengths and end positions cannot be smaller than the observed ones.

- If a segment in the prior input does not exist (i.e. it was not observed even though the segments before and after it were successfully matched), its posterior input is min length = mode length = max length = 0.

- If a segment was observed which was not predicted (i.e. could not be matched with any prior segment), the DAT insert a new segment which has the length and end position of the observed segment. (see also Figure 7.4, p. 76: the segments numbered 4.5 and 4.75 were inserted by the DAT while processing observations)

- if the prior segment has not been matched yet, because the excavation has not yet advanced into that part of the tunnel, the posterior input for the segment is the same as prior input.

Note that the DAT do not make any new suggestions about the unexcavated part. The program only updates the input related to the excavated part. It thereby provides the basis on which the expert can refine the predictions for the unexcavated part.

7.5 Deterministic Definition of Geotechnical Parameters

The term “deterministic” may be misleading. If input were really deterministic there would be no need for an updating process because geotechnical parameters could be defined precisely once and for all. Sometimes, however, input may be made deterministically even though one is somewhat uncertain about it. Deterministic input involves much less effort than probabilistic input. That’s why it can be justified to define even uncertain parameters

deterministically if their associated uncertainty is small and/or they are of lesser importance.

The updating process is nothing more than a comparison of the prior sequence of parameter states and the actually observed sequence (Figure 7.5). As was described in the previous section, the DAT try to match observed segments to prior ones and change their status (rightmost column in Figure 7.5) accordingly.

Update Parameters for Deterministic Generation Mode

[Print Preview / Print](#)[Back to Summary](#)

Parameter Set: 0 ▾

Parameter: Overburden ▾

Prior			Observed / Posterior			
Parameter State		Endpos		Parameter State	Endpos	Status
low	▾	1,000.0	✕ insert segment	low	1,000.0	Observed
high	▾	2,350.0	✕ insert segment	high	2,400.0	Observed
low	▾	3,100.0	✕ insert segment	low	2,800.0	Current
high	▾	5,400.0	✕ insert segment	high	5,400.0	Posterior
low	▾	6,500.0	✕ insert segment	low	6,500.0	Posterior
high	▾	8,100.0	✕ insert segment	high	8,150.0	Current
low	▾	10,000.0	✕ insert segment	low	10,000.0	Observed

Figure 7.5: Sample Window to Update Deterministic Input

Again it is up to the expert to refine the end positions of those segments in the unexcavated part (those marked with “Current” and “Posterior”).

7.6 Tunnel Delays

Updating tunnel delays is mostly a matter of human judgment. Again, the DAT’s role is to present prior predictions together with delays that actually occurred. In addition to this the DAT try to match predicted and observed delays, thereby taking some work from the expert.

It was mentioned already in section 5.3 (“Tunnel Information”, p. 42) that in the DAT a delay can be defined as time-dependent or position-dependent and that the start time/position is defined deterministically. Only the duration of a delay is defined probabilistically.

A prior prediction and an observed delay are considered a successful match when the following conditions are met:

- the observed duration is within the brackets of the predicted duration [min; max] where an additional tolerance of $\pm 20\%$ is granted, i.e. the observed duration is within the interval $[0.8 \cdot \text{duration min}; 1.2 \cdot \text{duration max}]$
- if the delay is predicted as time-dependent:
The difference between the predicted and the observed start time is less than the observed duration.
- if the delay is predicted as position-dependent:
The predicted start position is within $\pm 25\%$ of the observed start position.

If an observed delay is successfully matched to a predicted one, the DAT automatically suggest the start time/position and duration of the observed delay as posterior input for that delay.

If an observed delay cannot be matched to a predicted one, the DAT automatically create a new delay, which shows duration = 0 in prior input and the observed start time/position and duration as posterior input.

If a predicted delay cannot be matched, but a later delay was successfully matched, it is assumed that this particular delay did not occur and the DAT suggest duration = 0 as posterior input.

Finally, if any other predicted delay cannot be matched, its predicted start time/position and duration are also suggested as posterior input.

The bottom part of the update window (Figure 7.6) presents a table of all delays of a particular tunnel (which can be chosen in the top left part). The prior prediction is shown on the left, the posterior prediction on the right side. The middle column shows the status of each delay. A delay can be "Observed", "Current" (it is still going on) or "Posterior" (it is the posterior prediction for a delay in the future). According to their status, delays are plotted in different colors in the two plots above.

The lower plot shows time-dependent delays. This plot does only have a time axis (horizontal). An "Observed" delay is shown as a box from its start time (left boundary) to its end time (right boundary). For example, in Figure 7.6, there is an observed delay from $t = 220$ to $t = 240$. "Current" and "Posterior" delays, as well as prior predictions of delays, are shown as a box in which the right boundary (the end time) is subdivided into two lines.

In other words, there is an upper, a middle and a lower right corner. These show the predicted min, mode and max end times, respectively (see Figure 7.7)

Figure 7.6: Sample Window to Update Tunnel Delays

Figure 7.7: Explanation of the Time-Dependent Delay Plot

The upper plot shows position-dependent delays. This plot has a position axis (horizontal) which shows the start position and a time axis (vertical) which shows the duration. An “Observed” delay is a vertical line with two points with its horizontal position at the start position and the length of the line according to the observed duration. “Current” and “Posterior” delays, as well as prior predictions of delays, are shown as a vertical line with three points at the upper end. The uppermost stands for the max predicted duration, the middle one for the mode and the lower one for the min duration (see Figure 7.6, upper graph).

The graphical presentation facilitates the updating of posterior predictions. As was mentioned at the beginning of the section, this process is completely in the hands of the expert.

7.7 Association of Geometries, Ground Classes and Methods

The associations between geometries, ground classes and methods define which construction method is used, given that a certain ground class is present and given that the tunnel has a particular geometry. This association is not deterministic in the sense that with the same ground class and geometry, one has more than one choice of method. The user must specify which methods are part of the set of choices and with what probability they are chosen. These probabilities are uncertain and need to be updated. Note that even though these probabilities are uncertain, the user only specifies one value – the most likely one.

The predicted probabilities must be compared to the observed ones. Even though the latter are not directly observed, they can be derived from the observed parameter states and the observed (applied) methods. This derivation of observed probabilities is done by the DAT when observations are processed.

For a given tunnel with geometry m the association probability is defined as:

$$AP_{mno} = \frac{\text{length over which method } o \text{ is used when ground class } n \text{ is present}}{\text{total length of ground class } n \text{ being present}} = \frac{l_{no}}{l_n}$$

Consequently the DAT have to derive a ground class sequence from the observed parameter state sequences. This process is explained in section 2.3 (“Geology Module”, p. 14). Once the ground class sequence is derived it can be compared to the sequence of applied methods and the two terms in the above equation can be calculated (Figure 7.8).

ground class	good conditions	bad conditions	good	bad cond.	good conditions
method	full face	heading & benching			full face
length _{good/full face}
length _{good}

Figure 7.8: Calculation of Association Probabilities

The updating of the association probability itself is based on expert judgment. The DAT performs the processing of the observed information and presents the results together with prior probability values (Figure 7.9, 2nd and 3rd column from the right). Based on this the expert defines posterior probability values (Figure 7.9, rightmost column). An important additional piece of information is the “total length observed” (Figure 7.9, 3rd column from the left). It is the total length over which the ground class in question was observed, i.e. it is the value of l_n in the above equation. This information tells the expert how relevant the observed association probabilities are. If the “total length observed” is small compared to the length of the tunnel, the observed probabilities may not be very relevant. The greater “total length observed”, the more relevant are the observed probabilities.

Update Probabilistic Association of Geometries, Ground Classes and Methods							Back to Main Menu
Add Associations							Print Preview / Print
geo- metry	ground class name	total length observed	method name	prior	observed	posterior	
1	bad conditions	760	TBM w/ Concrete Lining fast	0.00	0.07	0.00	✕
			TBM w/ Concrete Lining slow	1.00	0.93	1.00	✕
1	good conditions	2820	TBM w/ Concrete Lining fast	1.00	0.99	1.00	✕
			TBM w/ Concrete Lining medium	0.00	0.01	0.00	✕
1	medium conditions	2170	TBM w/ Concrete Lining fast	0.00	0.09	0.00	✕
			TBM w/ Concrete Lining medium	1.00	0.77	1.00	✕
			TBM w/ Concrete Lining slow	0.00	0.13	0.00	✕

Figure 7.9: Sample Window to Update Associations between Geometries, Ground Classes and Methods

It is important to note that the sum of all association probabilities for any given combination of geometry and ground class must equal 1, i.e. at any particular point along the tunnel one and only one of the specified methods *must* be selected.

By default the DAT always suggest the prior probabilities as posterior input.

In section 7.13 (“Possible Extension for Updating Associations between Geometries, Ground Classes and Methods”, p. 96) the author proposes an extension which would allow the DAT to suggest posterior association probabilities by using a mathematical model which involves both prior input and observed probabilities.

7.8 Method Change Delays

Method change delays are defined deterministically (see section 5.15, “Method Change Delays”, p. 57). Nevertheless they can be uncertain and therefore the Updating Module provides the possibility to update them (Figure 7.10). The upper left part of Figure 7.10 shows a table with all specified method change delays and their prior (3rd column) and posterior value (4th column). The graphs only show information related to the method change delay which is currently marked by the cursor (the fourth one in Figure 7.10). The lower plot shows the observed delays on a time scale. This may expose trends in the duration of the delay (in Figure 7.10 there seems to be a trend towards decreasing durations). The upper right graph shows a frequency plot of the observed delays (in Figure 7.10 it indicates that the range of values is from 0.05 to 0.13 and that the mean is around 0.075). It is up to the user to compare the graphs and the prior prediction in order to arrive at a posterior prediction for the method change delay.

Update Method Change Delays

Add Change Delay

Print Preview / Print

Back to Summary

from method:	to method:	Prior:	Posterior:	
Shaft D&B w/ Shotcrete 1	Shaft D&B w/ Shotcrete 8	0.024	0.024	↗↘
Shaft D&B w/ Shotcrete 6	Shaft D&B w/ Shotcrete 8	0.040	0.040	↗↘
	Shaft D&B w/ Shotcrete 1	0.024	0.024	↗↘
Small TBM w/ Bolts & Sh	Small TBM w/ Steel Archi	0.080	0.075	↗↘
Small TBM w/ Shotcrete (	Small TBM w/ Bolts & Sh	0.040	0.038	↗↘
	Small TBM w/ Steel Archi	0.080	0.070	↗↘
Small TBM w/ Steel Archi	Small TBM w/ Bolts & Sh	0.056	0.060	↗↘
	Small TBM w/ Shotcrete (	0.040	0.042	↗↘

Figure 7.10: Sample Window to Update Method Change Delays

7.9 Tunnel Detail Information

Part of the tunnel detail information requires some updating. This is, however, more in the sense of making sure that the simulations with the updated input produce correct results, rather than in the sense of refining predictions.

One must be aware that the DAT simulate only the unexcavated part. In the case of updated input tunnels begin no longer at their real start position but rather at the current position of their tunnel face. As a consequence, the DAT would only evaluate the costs and construction time of the unexcavated part of the tunnel. This is not desirable, though, because the results of the predictions of the various update phases would not be comparable. To overcome this problem, it is possible to specify the time and costs associated that part of the tunnel which has already been excavated. Three pieces of information must be provided:

- the construction time for the excavated part or, in other words, the time at which excavation of the remaining part starts
- the costs associated with the excavated part of each tunnel
- the new begin and end positions of each tunnel (i.e. the unexcavated part of the tunnel)

Update Tunnel Information + Create Input Files

[Print Preview / Print](#)[Back to Menu](#)[CREATE FILES](#)

Note: Fix costs must include all cost incurred up to present and all future costs that are not included elsewhere

current time

tunnel	complete	fix cost min	mode	max	current position	new end position	400.00
Access Shaft	☒	18,598,000	18,598,000	18,598,000	500	500	
North Exit - Shaft	☐	23,700,000	24,000,000	24,500,000	1,500	3,000	
Shaft - South Exit (Aux)	☐	7,850,000	7,935,000	8,200,000	4,140	14,459	
Shaft - South Exit (Main)	☐	4,500,000	4,800,000	5,150,000	3,260	17,603	
South Exit - Shaft (Aux)	☐	37,500,000	37,700,000	37,950,000	14,459	4,140	
South Exit - Shaft (Main)	☐	47,400,000	48,100,000	48,700,000	17,603	3,260	

Figure 7.11: Sample Window to Update Tunnel Information

The time at which excavation of the remaining part starts is entered in the top right field “current time” in Figure 7.11.

A review of section 5.3 (“Tunnel Information”, p. 42) reveals that one can specify fix costs for tunnels, i.e. costs that are added to the overall constructions cost independent of the

geology and construction process. In updated input fix costs must also include the costs associated with the excavated part. Fix cost are therefore the sum of:

- fix costs that were incurred in the past
- costs of the excavation process (i.e. from the construction activities) incurred in the past
- fix costs that are expected for the future (e.g. dismantling of a TBM or of construction facilities)

The first and second points are costs that are known exactly, but there is still uncertainty about the last point. Therefore fix costs are specified using a triangular probability density function (see Figure 5.1, left, p. 38).

The new begin and end positions of the tunnels are usually calculated by the DAT, when observations are processed. This information can be derived from the sequence of applied construction methods (see sub-section “Construction Methods”, p. 62), which shows which part of each tunnel has already been excavated.

7.10 Creation of Updated Input Files

The creation of input files for the simulator is largely a technicality. The Updating Module of the DAT must simply write the data in a format which can be read by the simulator. There are no particular difficulties if files are created from initial input. Once files are created from updated input, however, there are some issues that must be dealt with.

As was mentioned in the previous section, the simulator does only simulate the unexcavated part of the tunnel. Difficulties arise if there are two parallel tunnels which have advanced to different positions within an area (see Figure 7.12).

In Figure 7.12 zone B is subdivided into a known part (zone B1) and an unknown part (zone B2). Zone A is known completely and zone C is still unknown. “Known” means that all geologic/geotechnical parameters are known and not necessarily the full extent of the zone (the extent of zone A is known, the extent of zone B is not). This subdivision into zones B1 and B2 is necessary if at least one of the geotechnical parameters in it has generation mode “Markov”, which is usually the case. Because zone B1 is now known deterministically it is no longer appropriate to use Markov generation mode, but the input data structure of the DAT does not allow one to use two different generation modes for a

particular geotechnical parameter in the same zone. In the case of generation mode “semi-deterministic” a subdivision is not necessary because the semi-deterministic sequence can have a deterministic (i.e. min endpos = mode endpos = max endpos) and probabilistic part.

Figure 7.12: Two Tunnels in the same Area with Different Construction Progress

Let us assume first that tunnel 2 does not exist at all. In this case zone A and the known part of zone B, zone B1, could be neglected. One could introduce a dummy zone which covers zones A and B1 and the “new” zone B would only consist of the unexcavated part, zone B2. Unfortunately such a system breaks down if there is a second tunnel which has advanced less far than the first one (tunnel 2 in Figure 7.12), because zone B1 can no longer be neglected.

To solve the problem, zone B must be subdivided into two zones, B1 and B2. In zone B1 all parameters have generation mode “deterministic”, in zone B2 their generation mode is like the original one used in zone B. As a matter of fact, zone B simply becomes zone B2 (with reduced length) and zone B1 is newly created. Zone B1 must also have a new parameter set, because the parameter set defines the generation modes of the individual parameters and these are different in zones B1 and B2. In the new parameter set all parameters have generation mode “deterministic”. The deterministic segments are created from the observations in this zone.

Zone A could still be neglected, i.e. replaced by a dummy zone. Nevertheless it is decided to keep the zone as it is and only change the generation mode of its parameters to “deterministic”. Again, deterministic segments are created from the observations in this zone. A difficulty arises if the parameter set of a known zone is also used by an unknown

zone (for example zone C uses the same parameter set as zone A – which is possible). In this case a new parameter set must be created for the known zone (e.g. zone A).

7.11 Possible Extensions for Updating Variables and Cycle Lengths

There are two problems related to the way variables (method variables and general variables) and cycle lengths are currently treated by the DAT:

- They cannot be updated according to any mathematical rule. Updating necessarily requires expert judgment in weighting the prior probability density function and the new information gathered.
- The current input in the DAT is in the form of a triangular pdf with spikes (see Figure 5.1 middle, p. 38). This pdf represents either daily fluctuations or long-term averages. However, if the pdf represents daily fluctuations it implicitly also defines the expected long-term average value in a deterministic manner: It can simply be calculated from the pdf. In short tunnels it may be that the uncertainty about the long-term average is not significant but there are boundary cases where the long-term average is also a substantial source uncertainty and should therefore not be defined deterministically.

The author wishes to propose two possible (but mutually exclusive) extensions of the DAT. Both of them would allow one to update variables and cycle lengths in a mathematical / analytical way and thereby solve the first problem. The benefit of this is, that the DAT would be capable of making suggestions about the posterior pdf's, as it is already the case with Markov lengths (section 7.2, sub-section "Mean Length", p. 69). One should be aware though, that the first extension can only be applied if the input for variables and cycle lengths represents daily fluctuations. The second extension can be applied in either case.

Only the second extension can solve the second problem. The downside for both extensions is the fact that additional input is required when prior predictions are made. Because the definition of prior input was already felt to be quite demanding in earlier applications of the DAT, it was decided not to implement any of the two extensions in the current version of the DAT.

The first extension introduces the input parameter “confidence level” which determines how prior input and new information should be weighted against each other. The second extension introduces a completely new concept of how variables and cycle lengths should be defined. It treats the long-term average and daily fluctuations separately.

In both cases variables and cycle lengths are treated absolutely equally. For improved readability the remainder of this section refers only to variables, but everything applies to cycle lengths as well.

Extension 1: Introducing a “confidence level”

Note: This extension can only be applied to the case where the input represents daily fluctuations.

It was mentioned above, that currently the long-term average of variables is implicitly defined by the pdf which is used for daily fluctuations. The long-term average equals the mean μ' of the pdf (see Figure 7.13). In addition to this, also the variance v' of the daily values can be evaluated from the pdf.

Figure 7.13:PDF for Daily Fluctuations with Mean μ'

Assume now that this variable (say, for example, the daily advance rate of a particular method) has been measured on n different days and all days are considered to be equally representative, i.e. nothing unusual occurred on these days. The statistics of the measured values is defined by their mean m and their variance v .

Without any further information it is impossible to combine the prior input (μ', v') and the new information (m, v) in order get to a posterior input (μ'', v'') . The reason is that we don't

know how to weight the two pieces of information. If the expert felt very confident about his prior prediction he or she would be inclined to almost neglect the new information and attribute the difference to some kind of temporary reason which would not persist. In contrast, if the expert felt very vague about the prior prediction, he or she would heavily rely on the new information. Consequently it is necessary not only to ask the expert for a prior prediction but also for his or her “confidence level” about the prediction.

It is suggested to measure the confidence level by asking the following question:

“Assume that n daily values have been measured, all considered to be representative, and their mean is m . With what number of measurements n_{CL} would you expect your predicted mean and the measured mean to be equally likely?”

In the special case where the actual number of measurements n equals n_{CL} the posterior input (μ'', v'') can be found as a direct result of the above question:

$$\text{Eq. 7-1: } \mu'' = \frac{\mu' + m}{2}; \quad v'' = \frac{v' + v}{2}$$

If n does not equal n_{CL} , μ'' and v'' can be obtained by using weighted averages:

$$\text{Eq. 7-2: } \mu'' = \frac{n_{CL} \cdot \mu' + n \cdot m}{(n_{CL} + n)}; \quad v'' = \frac{n_{CL} \cdot v' + n \cdot v}{(n_{CL} + n)}$$

The number n_{CL} is a very straightforward way of measuring the confidence level. Note however, that the confidence levels for the mean and the variance are assumed to be the same. This is usually not the case because it is more difficult to estimate a variance than a mean. In particular, the variance is typically underestimated (de Neufville, 1999).

By applying **Eq. 7-2** one obtains the mean and variance of the posterior pdf. This is not sufficient though, to fully define the pdf because it has five parameters (prob. min, min, mode, max, prob. max). Three of them need to be found in some other way. It is suggested to apply the following constraints:

- prob. min and prob. max are not updated, i.e. $(\text{prob. min})'' = (\text{prob. min})'$
- the ratio $(\text{mode} - \text{min}) / (\text{max} - \text{mode})$ [I_1/I_2 in Figure 7.14 left] should be the same for the prior and posterior pdf.

The posterior pdf can be obtained by subjecting the prior pdf to a sequence of two geometric transformations. These will lead to a posterior pdf with the desired mean and variance, while maintaining the above two constraints.

- 1) The first geometric transformation (Figure 7.14 left) involves only the min and max of the triangular pdf. The transformation is a horizontal stretching or contraction with its fixed point at the mode. The amount of stretching/contraction is chosen such that the transformed pdf has the desired variance v'' . Because the total area of the pdf must remain 1.0, the height h of the triangle must be adjusted accordingly. Note that this transformation obeys both constraints above.
- 2) The second geometric transformation (Figure 7.14 right) is a simple horizontal translation such that the transformed pdf has the desired mean. The translation does not affect the variance nor does it violate the two constraints above. Therefore the pdf obtained with these two transformations has exactly the desired properties.

Figure 7.14: Geometric Transformations: stretching/contraction (left) and translation (right) (dashed line: before transformation, full line: after transformation)

If the stretching factor k is best found iteratively. A good approximation is

Eq. 7-3:
$$k \approx \sqrt{\frac{v''}{v'}}$$

The translation length t is obtained by

Eq. 7-4:
$$t = \mu'' - \mu'$$

For a mathematical derivation of these equations see section 8.3 (“Properties of the Triangular PDF with Spikes”, p. 116)

This extension represents a very straightforward and practical way of updating input for variables and cycle lengths. One should be aware though, that this is by no means based on a rigorous probabilistic approach.

Extension 2: Treat daily fluctuations and long-term averages separately

In analogy to the Markov process (chapter 3, “The Markov Process”, p. 23) we can think of the daily fluctuations as a stochastic process which generates variable values according to the triangular pdf with spikes which is defined in the input of the DAT. In its current form (if the input represents daily fluctuations) the process is defined deterministically, i.e. the parameters of the pdf are fixed. This does not have to be so. One could define the pdf of the stochastic process (Figure 7.15 left) by defining some parameters probabilistically. In particular one could define the mean of the stochastic process probabilistically (Figure 7.15 right).

Currently, there are two ways in which the DAT evaluate variable values (see section 5.17, “Daily Fluctuations vs. Long-Term Averages”, p. 58). The DAT play a lottery to choose a variable value from the pdf which is defined in the input. This is done either only once at the beginning of each simulation or each time a variable is used during the simulation.

Figure 7.15:PDF for Daily Fluctuations (left) and Long-Term Average (right)
The left pdf depends on the parameter m , which is selected from the right pdf

With this extension, the DAT would first play a lottery at the beginning of every simulation to choose a mean value from the pdf in Figure 7.15, right. This mean value defines the exact shape of the pdf to the left. Each time a variable is used during this particular simulation, the DAT would again play a lottery to choose a variable value from the left pdf. The process would be restarted with the next simulation by playing a lottery for a new mean value.

The challenge is to find these two pdf's by questioning experts. Since the left pdf depends on the mean which is generated from the pdf to the right, one has to create rules according to which the left pdf is defined once the mean is determined from the right pdf. There is a large number of possible rules. The most sensible two are:

- 1) The expert specifies the shape of the left pdf for any mean. Once the mean is generated from the right pdf, the left pdf is translated such that its mean coincides with the generated one. This is exactly what is done in the second geometric transformation in the previous sub-section (see Figure 7.14, right)

If we denote the mean of the left pdf before the transformation by μ and the mean generated from the right pdf by m , the amount of translation required is given by:

$$t = m - \mu$$

There are however two problems with this translation:

Very often the min parameter of the triangular pdf is explicitly defined to be zero and the prob. min parameter set to some value $p_{\min} > 0$ because the variable has a certain chance to be zero (example: the cycle length of a TBM has a 10% chance to be zero, i.e. $\min = 0$ and $p_{\min} = 0.10$). One has to be aware that the translation also changes the min parameter which makes input as described above impossible.

For exactly the same reason it is possible that the min parameter becomes negative. If the mean generated from the right pdf is smaller than the mean of the (not yet translated) left pdf the translation has to be to the left. Negative values are meaningless for most variables and in particular for cycle lengths.

- 2) The min parameter is held fixed and the left pdf is subjected to stretching/contraction until the mean coincides with the mean generated from the right pdf. This is almost the same geometric transformation as the one shown in Figure 7.14, left, except for the fact that in this case the fixed point is min, rather than mode (as in Figure 7.14).

If we denote the mean of the left pdf before the transformation by μ and the mean generated from the right pdf by m , the stretching factor k is found by:

$$k = \frac{m - \min}{\mu - \min}$$

By doing this stretching instead of the translation proposed above, one can avoid the problems related to the translation completely. One has to be aware though, that this transformation affects not only the mean but also the variance. The variance increases or decreases approximately by the square of the stretching factor k .

A more serious problem arises if the left pdf has a small variance but the pdf for the mean (right) has a large variance. In other words: the daily fluctuations are expected to be small but the average is very uncertain. It is evident that in this case it does not make sense to keep $\min$ fixed because it is at least as uncertain as the average.

Considering the advantages and disadvantages of the two rules, one can conclude that the choice must be made depending on the actual situation. It is suggested to use rule 2 if $\min = 0$ in the pdf of daily fluctuations (Figure 7.14, left). Otherwise it is better to use rule 1. Note that if rule 2 is only used with $\min = 0$, the formula for k simplifies to $k = m / \mu$.

Updating the parameter m of the stochastic process

Because the daily fluctuations are now treated as a stochastic process with uncertain mean m , it is possible to update the pdf for the mean m (Figure 7.15, right). If one has a number of observations $x_1, x_2, \dots, x_n$ one can calculate the likelihood (see section 4.2, "Bayes' Theorem", p. 33) for every m .

The likelihood and the prior pdf for the mean m lead to a posterior pdf for the mean m . In theory the updating process is therefore quite straightforward. There are however serious practical problems arising from the fact that both pdf's in Figure 7.15 have so many discontinuities. The updating problem becomes analytically intractable.

If rule 2 is used to find the pdf of daily fluctuations there is an additional problem because the variance changes with the mean. This makes a reasonable solution for the updating process impossible.

If rule 1 is used, updating is possible if some adjustments are made. For the purpose of the updating process it is necessary to transform both pdf's to normal distributions (while

maintaining the means and variances), update the right pdf as a normal distribution and then transform it back to a triangular form.

The mathematical details of this updating process are given in section 8.4 (“Updating Variables and Cycle Lengths as a Stochastic Process”, p. 120)

7.12 Possible Extension for Updating Transition Probabilities

It was mentioned already in section 7.2 (“Parameters of the Markov Process”, p. 69) that although transition probabilities are uncertain, they are defined deterministically. The updating process is basically a comparison of the prior input with the observed transition probabilities. It was also mentioned that the number of transitions that have taken place is a measure of how reliable the observed probabilities are.

The updating problem is therefore very similar to the one for variables and cycle lengths (see previous section) where the mean of the pdf of daily fluctuations is to be updated based on a prior prediction and a number of observations.

In fact, the concept of the first proposed extension in the previous section (Extension 1: Introducing a “confidence level”, p. 89) is equally applicable to transition probabilities.

In this context the “confidence level” is the number of transitions that must be observed in order to make the expert feel that his/her prior prediction for a transition probability is equally likely as the observed probability.

The posterior transition probability TP'' can be found analogously to **Eq. 7-2** as a weighted average of the prior probability TP' and the observed probability TP_o , where n is the number of observed transitions and n_{CL} is the confidence level.

Eq. 7-5:
$$TP'' = \frac{n_{CL} \cdot TP' + n \cdot TP_o}{(n_{CL} + n)}$$

With the additional information of the confidence level, the DAT could therefore make a suggestion for the posterior transition probabilities on its own, by applying **Eq. 7-5**.

7.13 Possible Extension for Updating Associations between Geometries, Ground Classes and Methods

The reasoning that was used in the previous section can also be applied to association probabilities. An indication for the relevance of observed probabilities is the total length over which a certain geometry/ground class combination was observed. The relevance increases with this length. In this context the “confidence level” is the total length over which a particular geometry/ground class combination must have been observed in order to make the expert feel that his/her prior prediction for an association probability is equally likely as the observed probability.

The posterior association probability AP'' can again be found analogously to **Eq. 7-2** as a weighted average of the prior probability AP' and the observed probability AP_o , where l is the total length over which the geometry/ground class in question was observed and l_{CL} is the confidence level.

Eq. 7-6:
$$AP'' = \frac{l_{CL} \cdot AP' + l \cdot AP_o}{(l_{CL} + l)}$$

With the additional information of the confidence level, the DAT could therefore make a suggestion for the posterior association probabilities on its own, by applying **Eq. 7-6**.

However, the author feels, that the concept of the confidence level as a length, the way it is used in this section is less straightforward and less easily accessible for the user than the confidence level as a number of transitions, as it was used in the previous section.

8. Mathematical Background

This chapter covers the mathematical background for the updating process. The mathematically less inclined reader may skip this chapter and still perfectly understand what follows. It is however important that the concept of the updating process, as described in chapter 4 ("The Updating Process", p. 26) is well understood.

8.1 Updating with Continuous Probability Distributions

When Bayes' theorem was introduced in section 4.2 ("Bayes' Theorem", p. 33) it was assumed that there is only a limited number of possible outcomes and observations. Bayes' theorem was therefore formulated as (Eq. 4-1):

$$\text{Eq. 8-1: } P''(Z = Z_i | X = X_i) = N \cdot P'(Z = Z_i) \cdot P(X = X_i | Z = Z_i)$$

where: Z:	parameter of interest
Z_i :	state i of parameter Z
X:	test
X_i :	outcome of test
N:	normalizing constant
$P'(Z = Z_i)$:	prior probability of Z_i
$P''(Z = Z_i X = X_i)$:	posterior probability Z_i , given the outcome of the test being X_i
$P(X = X_i Z = Z_i)$:	probability of the outcome being X_i if the true state of Z is Z_i (also referred to as the likelihood of Z_i , given the outcome X_i)

In many cases, however, there is a continuous range of possible outcomes. Instead of discrete probabilities P (P' or P'') there are probability distributions f (f' or f''), such as in Figure 1.2, p. 9. In addition to this, multiple tests with multiple observations $x_1, x_2, x_3, \dots, x_n$ may be available.

Bayes' theorem for continuous functions is formulated as:

$$\text{Eq. 8-2: } f''(z) = N L(z | x_1, \dots, x_n) f'(z)$$

where: Z:	state variable of interest
z:	outcome of Z (note that there is a continuous range of possible outcomes z and that they are therefore not indexed)
x_i :	observations
N:	normalizing constant
$f'(z)$:	prior probability distribution of Z
$f''(z)$:	posterior probability distribution of Z
$L(z \mid x_1, \dots, x_n)$:	likelihood of z, given the observations $x_1, x_2, x_3, \dots, x_n$

The likelihood of z given the observations x_i is the probability of observing the x_i if the true state of Z is z. It can therefore be rewritten as

$$\text{Eq. 8-3: } L(z \mid x_1, \dots, x_n) = P[(x_1 \mid Z = z) \cap (x_2 \mid Z = z) \cap (x_3 \mid Z = z) \cap \dots \cap (x_n \mid Z = z)]$$

Note the similarity of this expression to $P(X \mid Z_i)$ in **Eq. 8-1**. Like $f'(z)$ and $f''(z)$, $L(z \mid x_i)$ is now a continuous function.

It is a basic property of probability density functions that the area under the function equals 1, i.e.

$$\text{Eq. 8-4: } \int_{-\infty}^{\infty} f(x) dx = 1$$

The normalizing constant N in **Eq. 8-2** is chosen such, that this condition is fulfilled.

8.2 Updating Markov Parameters

In the Markov process segment lengths x_i are exponentially distributed with a mean $m^* = 1/\lambda$ (Figure 8.1).

$$\text{Eq. 8-5: } g(x) = \lambda \cdot e^{-\lambda x} = \frac{1}{m^*} \exp\left(-\frac{x}{m^*}\right)$$

Remember that in the context of the DAT, segment lengths are lengths over which certain parameter states occur along the tunnel alignment. The parameter m^* (or λ) of this distribution is therefore not known – unless tunnel excavation is finished and all segment lengths have been recorded.

Therefore, in the DAT, the input for the parameter m^* is in the form of a probability density function itself, specifically a triangular distribution (Figure 8.2).

Figure 8.1: Exponential Distribution

Figure 8.2: Triangular Probability Distribution for Mean Length m^*

The triangular distribution is the one that is to be updated. The prior probability distribution for m^* is designated by $f'(m)$, its posterior by $f''(m)$. It is further assumed that a set of segment lengths has already been observed: $x_1, \dots, x_n$.

The probability of observing a segment length x , given that the true value of the parameter m^* equals m , is (applying **Eq. 8-5**):

$$\text{Eq. 8-6: } P(x | m^* = m) = \frac{1}{m} \exp\left(-\frac{x}{m}\right)$$

The likelihood of $m^* = m$, given the single observation x is therefore

$$\text{Eq. 8-7: } L(m^* = m | x) = \frac{1}{m} \exp\left(-\frac{x}{m}\right)$$

and the likelihood of $m^* = m$, given multiple observations $x_1, \dots, x_n$ is

$$L(m^* = m | x_1, \dots, x_n) = \frac{1}{m} \exp\left(-\frac{x_1}{m}\right) \cdot \frac{1}{m} \exp\left(-\frac{x_2}{m}\right) \cdot \dots \cdot \frac{1}{m} \exp\left(-\frac{x_n}{m}\right)$$

$$\text{Eq. 8-8: } L(m^* = m | x_1, \dots, x_n) = \left(\frac{1}{m}\right)^n \exp\left(-\frac{x_1 + x_2 + \dots + x_n}{m}\right)$$

By introducing $X = x_1 + x_2 + \dots + x_n$ and combining **Eq. 8-8** and **Eq. 8-5**, we finally get

$$\text{Eq. 8-9: } f''(m) = N \cdot \left(\frac{1}{m}\right)^n \exp\left(-\frac{X}{m}\right) \cdot f'(m)$$

From this expression it is obvious that $f''(m)$ is no longer a triangular function. Because the DAT input requires a triangular probability distribution, an exact analytical solution for $f''(m)$ within the DAT is not possible.

As an approximation, the goal is to find a triangular probability distribution which shares the most important statistical properties of the exact analytical posterior distribution – whatever its shape might be.

What follows first are discussions of the likelihood function, the triangular probability density function (pdf) and the analytically exact posterior pdf. Afterwards the solution to the above approximation problem will be shown.

Discussion of the likelihood function

The likelihood function in **Eq. 8-9**, (see Figure 8.3) can be rewritten as

$$\text{Eq. 8-10: } L(m) = m^{-n} e^{-Xm^{-1}}$$

Figure 8.3: Likelihood Function: The right diagram is a magnification of the left one and includes the tangent to the curve in the lower point of inflection.

Its first and second derivatives are

$$\text{Eq. 8-11: } L'(m) = [Xm^{-(n+2)} - nm^{-(n+1)}]e^{-Xm^{-1}}$$

$$\text{Eq. 8-12: } L''(m) = [n(n+1)m^2 - 2X(n+1)m + X^2]m^{-(n+4)}e^{-Xm^{-1}}$$

The point of maximum is found with the condition $L'(m) = 0$:

$$\text{Eq. 8-13: } [Xm^{-(n+2)} - nm^{-(n+1)}] = 0 \Rightarrow m_{\max} = X/n$$

and the two points of inflection are found with the condition $L''(m) = 0$:

$$\text{Eq. 8-14: } [n(n+1)m^2 - 2X(n+1)m + X^2] = 0 \Rightarrow m_{\inf 1, \inf 2} = X \left[\frac{1}{n} \pm \sqrt{\frac{1}{n^2(n+1)}} \right]$$

Another characteristic point is the point of maximum curvature. Its exact location can be found by the condition $L^{(3)}(m) = 0$. This would lead to a cubic equation, which cannot generally be solved in closed form. However, the location of the point of maximum curvature can be reasonably well approximated by the intersection of the tangent in the lower point of inflection with the horizontal axis (see Figure 8.3, right side). This point is at

$$\text{Eq. 8-15: } m_{\text{intersec}} = X \frac{(n+2)\sqrt{n+1} - 2 \cdot (n+1)}{n^2 \cdot \sqrt{n+1}}$$

The area under the likelihood function is:

$$\text{Eq. 8-16: } \int_0^\infty L(m) = \int_0^\infty m^{-n} e^{-Xm^{-1}} = X^{-(n-1)} \cdot (n-2)! \text{ for } n > 1$$
$$\int_0^\infty L(m) \rightarrow \infty \quad \text{for } n = 1$$

These properties of the likelihood function are important because the function needs to be plotted on the screen with a minimum of necessary points (for maximum execution speed). The plot best approximates the function, if the above characteristic points are used as support points in the plot.

Discussion of the triangular probability density function

In this paragraph some basic relations of the triangular pdf are discussed. The pdf has three parameters: a minimum, mode and maximum. For convenience these are further on referred to as follows (Figure 8.4):

minimum $\equiv a$, mode $\equiv b$, maximum $\equiv c$; peak value: $f(\text{mode}) = f(b) = h$; (see Figure 8.4).

It is a basic property of probability density functions that the area under them equals 1. The area of the triangle is:

$$A = \frac{1}{2} \cdot h \cdot (c - a) = 1; \text{ and therefore } h = \frac{2}{c - a}$$

Figure 8.4: Notation in Triangular PDF (left) and Centroids of both Parts (right)

Quite obviously, $f(m)$ has a discontinuity at $m = b$. Therefore, the two parts of the triangle must be handled separately. The line equations $f_1(m)$ and $f_2(m)$ of the two respective parts result from basic geometry:

Eq. 8-17: $f_1(m) = \frac{h(m-a)}{b-a}; \quad f_2(m) = \frac{h(c-m)}{c-b}$

By substituting for h and introducing

Eq. 8-18: $u_1 = \frac{2}{(c-a)(b-a)}; \quad u_2 = \frac{2}{(c-a)(c-b)}$

these equations simplify to

Eq. 8-19: $f_1(m) = u_1 \cdot (m - a); \quad f_2(m) = u_2 \cdot (c - m)$

In the remainder of this sub-section, statistical properties of the triangular pdf will be determined.

Mean (1st order moment)

By definition the mean value $E(m)$ of a pdf can be written as

$$\text{Eq. 8-20: } E(m) = \int_{-\infty}^{\infty} f(m) m \, dm$$

In the case of a triangular pdf this translates into: (using relation 7b)

$$\text{Eq. 8-21: } E(m) = \int_a^b u_1(m-a) m \, dm + \int_b^c u_2(c-m) m \, dm$$

However, a simpler way is using the weighted average of the means of the two parts of the triangle. The means of the two parts are the locations of their respective centroids m_{A1} and m_{A2} (see Figure 8.4). The expressions for the centroids (= means) are as follows:

$$\text{Eq. 8-22: } m_{A1} = \frac{2b+a}{3}; \quad m_{A2} = \frac{2b+c}{3}$$

The weights must be assigned according to the areas of the two parts. These are

$$\text{Eq. 8-23: } A_1 = \frac{h \cdot (b-a)}{2} = \frac{b-a}{c-a}; \quad A_2 = \frac{h \cdot (c-b)}{2} = \frac{c-b}{c-a}$$

It follows therefore for the mean of the triangular pdf (note that $A_1 + A_2 = 1$):

$$\begin{aligned} \text{Eq. 8-24: } E(m) &= \frac{A_1 \cdot m_{A1} + A_2 \cdot m_{A2}}{(A_1 + A_2) = 1} = \frac{(2b+a)(b-a) + (2b+c)(c-b)}{3(c-a)} = \\ E(m) &= \frac{2b^2 - ab - a^2 + bc - 2b^2 + c^2}{3(c-a)} = \frac{c^2 - a^2 + b(c-a)}{3(c-a)} = \underline{\underline{\frac{a+b+c}{3}}} \end{aligned}$$

Variance (2nd order moment)

By definition the 2nd order moment $E(m^2)$ of a pdf can be written as

$$\text{Eq. 8-25: } E(m^2) = \int_{-\infty}^{\infty} f(m) m^2 \, dm$$

In the case of a triangular pdf this translates into: (using relation 7b)

$$\text{Eq. 8-26: } E(m^2) = \int_a^b u_1(m-a)m^2 dm + \int_b^c u_2(c-m)m^2 dm$$

Again a more direct physical approach is used:

In a physical context the above integral defines the moment of inertia J of the area under the pdf relative to the dependent (f -) axis. It is the sum of the moments of inertia of the two triangular parts:

$$\text{Eq. 8-27: } J = J_1 + J_2$$

J_1 (and similarly J_2) can be found using the following law (Steiner's law):

$$\text{Eq. 8-28: } J_1 = J_{1,\text{local}} + A_1 \cdot d^2$$

where: $J_{1,\text{local}}$: moment of inertia of the part around its own centroid

A_1 : area of the part

d : distance of the centroid of the part from the dependent axis

The moment of inertia of the left triangle (part 1) around its own centroid is

$$\text{Eq. 8-29: } J_{1,\text{local}} = \frac{(b-a)^3 h}{36}$$

By combining **Eq. 8-22**, **Eq. 8-23** and **Eq. 8-27** to **Eq. 8-29**, the second order moment of the pdf is therefore

$$\begin{aligned} E(m^2) &= J_1 + A_1 \cdot m_{A1}^2 + J_2 + A_2 \cdot m_{A2}^2 = \\ E(m^2) &= \frac{(b-a)^3 h}{36} + \frac{(b-a)h}{2} \left(\frac{2b+a}{3} \right)^2 + \frac{(c-b)^3 h}{36} + \frac{(c-b)h}{2} \left(\frac{2b+c}{3} \right)^2 \\ E(m^2) &= \frac{h}{36} \left[(b-a)(b^2 - 2ab + a^2 + 8b^2 + 8ab + 2a^2) + \right. \\ &\quad \left. + (c-b)(c^2 - 2bc + b^2 + 8b^2 + 8bc + 2c^2) \right] = \\ E(m^2) &= \frac{h}{36} \left[(b-a)(9b^2 + 6ab + 3a^2) + (c-b)(9b^2 + 6bc + 3c^2) \right] = \\ E(m^2) &= \frac{h}{12} \left[3b^3 + 2ab^2 + a^2b - 3ab^2 - 2a^2b - a^3 + \right. \\ &\quad \left. + 3b^2c + 2bc^2 + c^3 - 3b^3 - 2b^2c - bc^2 \right] = \end{aligned}$$

$$E(m^2) = \frac{h}{12} [-ab^2 - a^2b - a^3 + b^2c + bc^2 + c^3]$$

$$E(m^2) = \frac{h}{12} [(c^3 - a^3) + b(c^2 - a^2) + b^2(c - a)]$$

$$E(m^2) = \frac{(c^3 - a^3) + b(c^2 - a^2) + b^2(c - a)}{6(c - a)}$$

$$\text{Eq. 8-30: } E(m^2) = \frac{(a^2 + ac + c^2) + b(a + c) + b^2}{6} = \underline{\underline{\frac{a^2 + b^2 + c^2 + ab + ac + bc}{6}}}$$

The variance can be found by the applying a basic relationship, which applies to all pdf's

$$\text{Eq. 8-31: } \text{Var}(m) = E(m^2) - [E(m)]^2$$

Note: Remembering that the area under the pdf is 1 by definition, it is quite obvious that

Eq. 8-31 is just another application of Steiner's law (**Eq. 8-28**), where $\text{Var}(m)$ corresponds to J_{local} , $E(m^2)$ to J and $E(m)$ to d .

Inserting **Eq. 8-24** and **Eq. 8-30** into **Eq. 8-31** leads to a an expression of the variance in terms of a , b , and c :

$$\text{Var}(m) = \frac{a^2 + b^2 + c^2 + ab + ac + bc}{6} - \left[\frac{a + b + c}{3} \right]^2 =$$

$$\text{Var}(m) = \frac{a^2 + b^2 + c^2 + ab + ac + bc}{6} - \frac{a^2 + b^2 + c^2 + 2ab + 2ac + 2bc}{9} =$$

$$\text{Eq. 8-32: } \text{Var}(m) = \frac{a^2 + b^2 + c^2 - ab - ac - bc}{18}$$

Skewness (3rd order moment):

By definition the 3rd order moment $E(m^3)$ of a pdf can be written as

$$\text{Eq. 8-33: } E(m^3) = \int_{-\infty}^{\infty} f(m) m^3 dm$$

In the case of a triangular pdf this translates into: (using relation 7b)

$$\text{Eq. 8-34: } E(m^3) = \int_a^b u_1(m-a)m^3 dm + \int_b^c u_2(c-m)m^3 dm$$

The solution is found by performing the integration

$$\begin{aligned} E(m^3) &= \int_a^b u_1(m^4 - am^3) dm + \int_b^c u_2(cm^3 - m^4) dm = \\ E(m^3) &= \left[u_1 \left(\frac{m^5}{5} - \frac{am^4}{4} \right) \right]_a^b + \left[u_2 \left(\frac{cm^4}{4} - \frac{m^5}{5} \right) \right]_b^c \\ E(m^3) &= \frac{u_1}{20} (4b^5 - 5b^4a + a^5) + \frac{u_2}{20} (c^5 - 5b^4c + 4b^5) = \\ E(m^3) &= \frac{(4b^5 - 5b^4a + a^5)}{10(c-a)(b-a)} + \frac{(c^5 - 5b^4c + 4b^5)}{10(c-a)(c-b)} \\ E(m^3) &= \frac{(4b^5 - 5b^4a + a^5)(c-b) + (c^5 - 5b^4c + 4b^5)(b-a)}{10(c-a)(b-a)(c-b)} = \end{aligned}$$

after simplifying

$$\text{Eq. 8-35: } E(m^3) = \frac{a^5(c-b) + b^5(a-c) + c^5(b-a)}{10(c-a)(b-a)(c-b)}$$

and finally after lengthy polynomial divisions:

$$\text{Eq. 8-36: } E(m^3) = \underline{\underline{\frac{1}{10}[a^3 + b^3 + c^3 + a^2(b+c) + b^2(a+c) + c^2(a+b) + abc]}}$$

Discussion of the analytically exact posterior PDF

The relevant equations for this paragraph have already been derived above. They are just copied for convenience and are not given new numbers. The analytically exact posterior pdf $f''(m)$ is defined as (repeating **Eq. 8-9**)

$$f''(m) = N \cdot \left(\frac{1}{m} \right)^n \exp\left(-\frac{X}{m}\right) \cdot f'(m)$$

The triangular prior pdf has two parts which are defined as

$$f'_1(m) = u'_1 \cdot (m - a'); \quad f'_2(m) = u'_2 \cdot (c' - m)$$

where (repeating **Eq. 8-18** and **Eq. 8-19**)

$$u'_1 = \frac{2}{(c' - a')(b' - a')}; \quad u'_2 = \frac{2}{(c' - a')(c' - b')}$$

It is obvious that the posterior pdf has two parts as well. Because $f'_1(m)$ is equal to zero everywhere outside of the interval $[a; b]$, this is also the case for $f''_1(m)$. For the same reason $f'_2(m)$ equals zero everywhere outside the interval $[b; c]$. This fact will become important later in the sub-section "Mode", p. 109.

By combining **Eq. 8-9** and **Eq. 8-19**, the posterior pdf of the two parts can be written as

$$\text{Eq. 8-37:} \quad f_1''(m) = Nu'_1 \left[\left(\frac{1}{m} \right)^{n-1} - a' \left(\frac{1}{m} \right)^n \right] \exp\left(-\frac{X}{m}\right)$$

$$\text{Eq. 8-38:} \quad f_2''(m) = Nu'_2 \left[c' \left(\frac{1}{m} \right)^n - \left(\frac{1}{m} \right)^{n-1} \right] \exp\left(-\frac{X}{m}\right)$$

Normalizing constant N

The normalizing constant N can be found by applying the rule, that the area under any pdf equals 1.

$$\text{Eq. 8-39:} \quad \int_{a'}^{b'} f_1''(m) dm + \int_{b'}^{c'} f_2''(m) dm = 1$$

By applying **Eq. 8-37** and **Eq. 8-38** and dividing by N:

$$\begin{aligned} \text{Eq. 8-40:} \quad & u'_1 \int_{a'}^{b'} \left[\left(\frac{1}{m} \right)^{n-1} - a' \left(\frac{1}{m} \right)^n \right] \exp\left(-\frac{X}{m}\right) dm + \\ & + u'_2 \int_{b'}^{c'} \left[c' \left(\frac{1}{m} \right)^n - \left(\frac{1}{m} \right)^{n-1} \right] \exp\left(-\frac{X}{m}\right) dm = \frac{1}{N} \end{aligned}$$

The integral

$$\text{Eq. 8-41: } \int_p^q \left(\frac{1}{m}\right)^r \exp\left(-\frac{X}{m}\right) dm$$

will appear frequently in the remaining part of the paragraph. To simplify notation an auxiliary function is therefore defined as follows:

$$\text{Eq. 8-42: } S(r, p, q) = \int_p^q \left(\frac{1}{m}\right)^r \exp\left(-\frac{X}{m}\right) dm$$

The solution to this integral will be presented at the end of this section ("The auxiliary function", p. 114).

Eq. 8-40 can now be rewritten as:

$$\text{Eq. 8-43: } \frac{1}{N} = u'_1 \cdot [S(n-1, a', b') - a' S(n, a', b')] - u'_2 \cdot [S(n-1, b', c') - c' S(n, b', c')]$$

Mean (1st order moment)

The mean of the posterior pdf can be determined easily by using the auxiliary function. By definition it is

$$\text{Eq. 8-44: } E''(m) = \int_{a'}^{b'} m \cdot f_1''(m) dm + \int_{b'}^{c'} m \cdot f_2''(m) dm =$$

By expanding for $f_1''(m)$ and $f_2''(m)$

$$\begin{aligned} E''(m) = & Nu'_1 \int_{a'}^{b'} \left[\left(\frac{1}{m}\right)^{n-1} - a' \left(\frac{1}{m}\right)^n \right] m \exp\left(-\frac{X}{m}\right) dm + \\ & + Nu'_2 \int_{b'}^{c'} \left[c' \left(\frac{1}{m}\right)^n - \left(\frac{1}{m}\right)^{n-1} \right] m \exp\left(-\frac{X}{m}\right) dm \end{aligned}$$

and using the auxiliary function, **Eq. 8-42**

$$\begin{aligned} \text{Eq. 8-45: } E''(m) = & N \cdot u'_1 \cdot [S(n-2, a', b') - a' S(n-1, a', b')] - \\ & - N \cdot u'_2 \cdot [S(n-2, b', c') - c' S(n-1, b', c')] \end{aligned}$$

Higher order moments

Analogously to the above equations for the mean, higher order moments can be found easily. The 2nd order moment is

$$E''(m^2) = \int_{a'}^{b'} m^2 \cdot f_1''(m) dm + \int_{b'}^{c'} m^2 \cdot f_2''(m) dm =$$

Eq. 8-46:
$$E''(m^2) = N \cdot u'_1 \cdot [S(n-3, a', b') - a' \cdot S(n-2, a', b')] -$$
$$- N \cdot u'_2 \cdot [S(n-3, b', c') - c' \cdot S(n-2, b', c')]$$

and finally the 3rd order moment is

$$E''(m^3) = \int_{a'}^{b'} m^3 \cdot f_1''(m) dm + \int_{b'}^{c'} m^3 \cdot f_2''(m) dm =$$

Eq. 8-47:
$$E''(m^3) = N \cdot u'_1 \cdot [S(n-4, a', b') - a' \cdot S(n-3, a', b')] -$$
$$- N \cdot u'_2 \cdot [S(n-4, b', c') - c' \cdot S(n-3, b', c')]$$

Mode

The modes of $f_1''(m)$ and $f_2''(m)$ are found by requiring the first derivative to equal zero:

Eq. 8-48:
$$\frac{df_1''(m)}{dm} = 0 \text{ and } \frac{df_2''(m)}{dm} = 0$$

The solutions of the two equations are named mode₁ and mode₂ respectively. From earlier in this section (**Eq. 8-37**):

$$f_1''(m) = Nu'_1 \left[\left(\frac{1}{m} \right)^{n-1} - a' \left(\frac{1}{m} \right)^n \right] \exp\left(-\frac{X}{m}\right)$$

and therefore:

$$\frac{df_1''(m)}{dm} = N u'_1 \left[X \left(\frac{1}{m} \right)^{n+1} - (n-1) \left(\frac{1}{m} \right)^n - X a' \left(\frac{1}{m} \right)^{n+2} + n a' \left(\frac{1}{m} \right)^{n+1} \right] \exp\left(-\frac{X}{m}\right) = 0$$

Multiplying by m^{n+2} and dividing by $N \cdot u'_1 \cdot \exp(-X/m)$ leads to

$$\text{Eq. 8-49: } Xm - (n-1)m^2 - Xa' + na'm = 0 \Rightarrow (n-1)m^2 - (X+na')m + Xa' = 0$$

The roots of this quadratic equation are:

$$\text{Eq. 8-50: } \text{mode}_1 = \frac{(X+na') \pm \sqrt{X^2 - 2Xna' + (na')^2 + 4Xa'}}{2n-2}$$

In the above equation the positive sign is relevant. Note that the function $f_1''(m)$ also has a minimum at $m_{\min} < a'$ as can be seen in Figure 8.5. This minimum is associated with the negative sign in **Eq. 8-50** but not relevant for this discussion because it is outside of the interval $[a'; c']$. Remember that the two functions $f_1''(m)$ and $f_2''(m)$ equal 0 outside of the intervals $[a'; b']$ and $[b'; c']$, respectively.

Figure 8.5: Analytically exact posterior pdf, $f_1''(m)$ and $f_2''(m)$
 In this example $\text{mode}_1 = 145$, $\text{mode}_2 = 65$. The true mode of $f_1''(m)$ equals mode_1 , whereas the true mode of $f_2''(m)$ equals $b' = 200$

mode₂ can be found similarly:

Eq. 8-51:
$$\text{mode}_2 = \frac{(X + nc') \pm \sqrt{X^2 - 2Xnc' + (nc')^2 + 4Xc'}}{2n - 2}$$

where the negative sign is relevant, because the positive sign is associated with the minimum of $f_2''(m)$ at $m_{\min} > c'$.

If mode₁ falls outside the interval $[a'; b']$, the true mode of part 1 is b' :

Because $f_1''(m) = 0$ outside of the interval $[a'; b']$, the true mode of part 1 must be within the interval. The fact that mode₁ lies outside the interval and that $a' = a'' = 0$ tells us that $f_1''(m)$ is monotonically increasing in the interval and therefore the maximum value is at $m_{\max} = b'$.

Analogously, if mode₂ falls outside the interval $[b'; c']$, the true mode of part 2 is also b' . This fact is illustrated by Figure 8.5, where the true mode of part 2 is $b' = 200$ rather than mode₂ = 65. If both mode₁ and mode₂ fall outside of their respective intervals, the true mode of both parts is at b' and therefore the posterior mode b'' equals b' .

Approximation of analytically exact posterior pdf with a triangular pdf

In order to optimally approximate the analytically exact posterior pdf with a triangular pdf, the posterior triangular pdf must have statistical properties which are equal or close to the statistical properties of the analytically exact pdf.

Since the triangular posterior pdf has three parameters (a'' , b'' , c''), it is possible to define them such that three of its properties equal their counterparts in the analytically exact posterior pdf. In other words: It is possible to pick any three properties of the analytically exact posterior pdf and find a triangular posterior pdf which exactly matches these three properties.

The most important properties of any pdf are usually:

- mean (= 1st order moment)
- variance (or 2nd order moment)
- skewness (or 3rd order moment)
- mode

Note that if the means and 2nd order moments are equal to each other, the variances are also equal by definition, because they can be calculated directly using **Eq. 8-31**. Similarly the skewness can be calculated from the 1st, 2nd and 3rd order moments.

The condition that the above four properties be equal, results in the following four equations (from **Eq. 8-24**, **Eq. 8-30** and **Eq. 8-36**). Note that a", b" and c" are the unknowns whereas E"(m), E"(m²), E"(m³) and the mode of the analytically exact posterior pdf are calculated as described in the previous sub-section.

$$\text{Eq. 8-52: } E''(m) = \frac{a'' + b'' + c''}{3}$$

$$\text{Eq. 8-53: } E''(m^2) = \frac{a''^2 + b''^2 + c''^2 + a''b'' + b''c'' + a''c''}{6}$$

$$\text{Eq. 8-54: } E''(m^3) = \frac{[a''^3 + b''^3 + c''^3 + a''^2(b'' + c'') + b''^2(a'' + c'') + c''^2(a'' + b'') + a''b''c'']}{10}$$

$$\text{Eq. 8-55: } \text{mode}(\text{analytically exact pdf}) = b''$$

where: E"(m): 1st order moment of analytically exact posterior pdf

E"(m²): 2nd order moment of analytically exact posterior pdf

E"(m³): 3rd order moment of analytically exact posterior pdf

Only three of the four equations above can be satisfied. It becomes clear, that with any choice, the problem results in a non-linear system or three equations for three unknowns. By neglecting **Eq. 8-54**, it is however possible to reduce the system to two linear and one quadratic equation. The problem becomes thereby analytically tractable in a straightforward way.

Solution of the system of equations

Parameter b" can be found trivially, once the mode of the analytically exact posterior pdf is found. Knowing b", the following equations are used to determine a" and c" (from **Eq. 8-52** and **Eq. 8-53**):

$$\text{Eq. 8-56: } a'' + b'' + c'' = 3 \cdot E''(m)$$

$$\text{Eq. 8-57: } a''^2 + b''^2 + c''^2 + a''b'' + a''c'' + b''c'' = 6 \cdot E''(m^2)$$

Eq. 8-56 can be solved for a''

$$a'' = 3 \cdot E(m) - b'' - c''$$

and inserted into **Eq. 8-57**. At the same time the following substitutions are made for ease of notation: $E''(m) = \mu$, $E''(m^2) = \omega$. **Eq. 8-57** then rewrites

$$\begin{aligned} 9\mu^2 + b''^2 + c''^2 - 6\mu b'' - 6\mu c'' + 2b''c'' + b''^2 + c''^2 + \\ + 3\mu b'' - b''^2 - b''c'' + 3\mu c'' - b''c'' - c''^2 + b''c'' = 6\omega \end{aligned}$$

and simplifies to

$$\text{Eq. 8-58: } c''^2 + c''(b'' - 3\mu) + b''(b'' - 3\mu) + 9\mu^2 - 6\omega = 0$$

This is a quadratic equation for c'' , the roots of which are

$$\begin{aligned} \text{Eq. 8-59: } c'' &= \frac{1}{2} \left(3\mu - b'' \pm \sqrt{b''^2 - 6b''\mu + 9\mu^2 - 4b''^2 + 12b''\mu - 36\mu^2 + 24\omega} \right) \\ c'' &= \frac{1}{2} \left(3\mu - b'' \pm \sqrt{24\omega - 3b''^2 + 6b''\mu - 27\mu^2} \right) \end{aligned}$$

Note that the equations are symmetric as far as a'' and c'' are concerned. The larger root (positive sign) is therefore the solution for c'' , and the smaller root (negative sign) the solution for a'' .

Negative a''

It may happen that the above solution leads to slightly negative values for a'' , even though $f_1''(m < 0) = 0$, because the triangular posterior pdf is only an approximation of the analytically exact posterior pdf.

In this case a'' is fixed at $a'' = 0$. Because there are only two free parameters (b'' and c'') left, one of **Eq. 8-52**, **Eq. 8-53** and **Eq. 8-55** can't be fulfilled. The criterion that b'' equals the mode of the analytically exact posterior pdf is considered least important and therefore revoked. Instead b'' and c'' are determined solely on the basis of **Eq. 8-56** and **Eq. 8-57**. These simplify greatly because $a'' = 0$:

$$\text{Eq. 8-60: } b'' + c'' = 3 \cdot E(m)$$

$$\text{Eq. 8-61: } b''^2 + c''^2 + b''c'' = 6 \cdot E(m^2)$$

Solving **Eq. 8-60** for b'' and inserting into **Eq. 8-61** results in

$$(3\mu - c'')^2 + c''^2 + (3\mu - c'')c'' = 6\omega$$

$$c''^2 - 3\mu c'' + 9\mu^2 - 6\omega = 0$$

The roots of the quadratic equation are (where $b'' < c''$)

$$\text{Eq. 8-62: } c'' = \frac{1}{2} \left[3\mu + \sqrt{24\omega - 27\mu^2} \right]; \quad b'' = \frac{1}{2} \left[3\mu - \sqrt{24\omega - 27\mu^2} \right]$$

The auxiliary function

In the section "Discussion of the analytically exact posterior PDF", p. 106, the following auxiliary function was introduced (Eq. 8-42):

$$\text{Eq. 8-42: } S(r, p, q) = \int_p^q \left(\frac{1}{m} \right)^r \exp\left(-\frac{X}{m}\right) dm$$

Using the substitution

$$\text{Eq. 8-63: } \lambda = 1/m \Rightarrow \frac{d\lambda}{dm} = -\frac{1}{m^2} \Rightarrow dm = -d\lambda \cdot m^2$$

Eq. 8-42 rewrites as

$$\text{Eq. 8-64: } S(r, p, q) = \int_{1/q}^{1/p} \lambda^{r-2} \exp(-X\lambda) d\lambda$$

For $r \leq 2$ see the paragraph "Special cases" below. For $r \geq 2$ partial integration can be used:

$$S(r, p, q) = \left[\frac{\lambda^{r-2}}{-X} \exp(-X\lambda) - \frac{r-2}{-X} \int \lambda^{r-3} \exp(-X\lambda) d\lambda \right]_{1/q}^{1/p}$$

It can be perceived that it is necessary to partially integrate $(r-2)$ times. The result is a series

$$S(r,p,q) = \exp(-X\lambda) \left[\frac{\lambda^{r-2}}{-X} - \frac{(r-2)\lambda^{r-3}}{(-X)^2} + \frac{(r-2)(r-3)\lambda^{r-4}}{(-X)^3} - \dots \right]_{1/q}^{1/p}$$

$$\text{Eq. 8-65: } S(r,p,q) = \left[-\exp(-X\lambda) \sum_{k=0}^{r-2} \frac{(r-2)! \cdot \lambda^k}{k! \cdot X^{r-1-k}} \right]_{1/q}^{1/p}$$

If $S(r-1, p, q)$ is known, $S(r, p, q)$ can also be calculated recursively

$$\text{Eq. 8-66: } S(r, p, q) = \frac{r-2}{X} \cdot S(r-1, p, q) + \left[\exp(-X\lambda) \frac{\lambda^{r-2}}{X} \right]_p^q$$

Special cases

For $r \leq 2$ the auxiliary function $S(r, p, q)$ can't be solved by partial integration. A special case is $r = 1$:

$$\text{Eq. 8-67: } S(1, p, q) = \int_p^q \left(\frac{1}{m} \right) \exp\left(-\frac{X}{m}\right) dm$$

The solution of the integral is an infinite series (Bronstein et. al, 1993). Note that once more the substitution in **Eq. 8-63** is used:

$$\text{Eq. 8-68: } S(1,p,q) = \int_{1/q}^{1/p} \frac{\exp(-X\lambda)}{\lambda} d\lambda = \left[\ln \lambda + \frac{-X\lambda}{1} + \frac{(-X\lambda)^2}{2 \cdot 2!} + \dots \right]_{1/q}^{1/p}$$

$$S(1, p, q) = - \left[\ln\left(\frac{1}{m}\right) - \frac{X}{m} + \frac{X^2}{m^2 \cdot 2 \cdot 2!} - \frac{X^3}{m^3 \cdot 3 \cdot 3!} + \dots \right]_p^q$$

For large ratios (X/m) the series converges very slowly and is numerically a problem because it is a sequence of differences of large numbers, which drastically decreases accuracy. It is therefore considered better to perform this integration numerically. At this point it should be noted that the function, which is to be integrated, is in fact the likelihood function (see "Discussion of the likelihood function", p. 100), **Eq. 8-9**.

For $r \leq 0$ the value of the integral can only be found recursively (Bronstein et. al., 1993):

$$S(r, p, q) = \int_{1/q}^{1/p} \frac{\exp(-X\lambda)}{\lambda^{2-r}} d\lambda = \frac{1}{1-r} \left[-\frac{\exp(-X\lambda)}{\lambda^{1-r}} \right]_{1/q}^{1/p} - X \int_{1/q}^{1/p} \frac{\exp(-X\lambda)}{\lambda^{1-r}} d\lambda$$

Eq. 8-69:
$$S(r, p, q) = \frac{1}{1-r} \left[\frac{\exp(-X\lambda)}{\lambda^{1-r}} \right]_{1/p}^{1/q} - X \cdot S(r+1, p, q)$$

This recursion needs to be performed k times until $S(r+k, p, q)$ becomes a known quantity. $S(1, p, q)$ is already known and therefore partial integration needs to be performed $k = (1-r)$ times. Note that r is negative. The result is a series:

$$S(r, p, q) = \exp(-X\lambda) \left[\frac{1}{(1-r)\lambda^{(1-r)}} - \frac{X}{(1-r)(-r)\lambda^{(-r)}} + \dots + \frac{(-X)^{(-r)}}{(1-r)!\lambda} \right]_{1/p}^{1/q} + \frac{(-X)^{(1-r)}}{(1-r)!} S(1, p, q)$$

Re-substituting m for $1/\lambda$ and compacting the notation of the series leads to

Eq. 8-70:
$$S(r, p, q) = \left[\exp\left(-\frac{X}{m}\right) \cdot \sum_{k=0}^{-r} \frac{(-r-k)! (-X)^k}{(1-r)! m^{(r+k-1)}} \right]_p^q + \frac{(-X)^{(1-r)}}{(1-r)!} S(1, p, q)$$

The recursive form is obtained directly from **Eq. 8-69** by substituting m for $1/\lambda$

Eq. 8-71:
$$S(r, p, q) = \left[\exp\left(-\frac{X}{m}\right) \cdot \frac{m^{(1-r)}}{(1-r)} \right]_p^q - \frac{X}{1-r} S(r+1, p, q)$$

8.3 Properties of the Triangular PDF with Spikes

The properties of the triangular pdf with spikes are important in section 7.11 (“Possible Extensions for Updating Variables and Cycle Lengths”, p. 88). The section also includes geometric transformations like stretching and translation. These issues will be dealt with in three sub-sections later in this section.

In sub-section “Discussion of the triangular probability density function”, p. 102, we found for the triangular pdf without spikes (repeating **Eq. 8-24**):

$$E(m) = \frac{a+b+c}{3}$$

and (repeating **Eq. 8-30**)

$$E(m^2) = \frac{a^2 + b^2 + c^2 + ab + ac + bc}{6}$$

The area of the triangle itself in the triangular pdf with spikes equals no longer 1, but is now $A_{\text{triangle}} = (1 - p_{\min} - p_{\max})$, where $p_{\min}$ and $p_{\max}$ are the probabilities prob. min and prob. max respectively (i.e. the areas of the respective spikes).

Figure 8.6: Notation for the Triangular PDF with Spikes

The expected value of the triangular pdf with spikes is (applying **Eq. 8-24**):

$$E(m) = A_{\text{triangle}} \cdot E(m)_{\text{triangle}} + p_{\min} \cdot a + p_{\max} \cdot c$$

Eq. 8-72:
$$E(m) = (1 - p_{\min} - p_{\max}) \frac{a+b+c}{3} + p_{\min} \cdot a + p_{\max} \cdot c$$

Similarly the second order moment is (applying **Eq. 8-30**):

$$E(m^2) = A_{\text{triangle}} \cdot E(m^2)_{\text{triangle}} + p_{\min} \cdot a^2 + p_{\max} \cdot c^2$$

Eq. 8-73:
$$E(m^2) = (1 - p_{\min} - p_{\max}) \frac{a^2 + b^2 + c^2 + ab + ac + bc}{6} + p_{\min} \cdot a^2 + p_{\max} \cdot c^2$$

The variance is found from **Eq. 8-31**:

$$\text{Var}(m) = E(m^2) - [E(m)]^2$$

The effect of a translation by a length t

If the whole pdf is translated by t it has the following properties ($'$ denotes the translated pdf), using **Eq. 8-72** and **Eq. 8-31**:

$$\begin{aligned}\text{Eq. 8-74: } E'(m) &= (1 - p_{\min} - p_{\max}) \frac{(a+t) + (b+t) + (c+t)}{3} + p_{\min} \cdot (a+t) + p_{\max} \cdot (c+t) \\ E'(m) &= E(m) + t\end{aligned}$$

$$\text{Var}'(m) = \text{Var}(m) \text{ by inspection}$$

$$\begin{aligned}\text{Eq. 8-75: } E'(m^2) &= \text{Var}'(m) + [E'(m)]^2 = E(m^2) - [E(m)]^2 + [E(m) + t]^2 \\ E'(m^2) &= E(m^2) + 2 \cdot t \cdot E(m) + t^2\end{aligned}$$

The effect of stretching the pdf by a factor k , with fixed point at $m = \text{mode}$

We can first rewrite $E(m)$ (**Eq. 8-72**) by subtracting b in each term and then adding it again separately:

$$\begin{aligned}E(m) &= (1 - p_{\min} - p_{\max}) \frac{(a-b) + 3b + (c-b)}{3} + \\ &\quad + p_{\min} (a-b) + p_{\min} b + p_{\max} (c-b) + p_{\max} b\end{aligned}$$

$$\text{Eq. 8-76: } E(m) = (1 - p_{\min} - p_{\max}) \frac{(a-b) + (c-b)}{3} + p_{\min} (a-b) + p_{\max} (c-b) + b$$

The stretching transformation affects the terms $(a-b)$ and $(c-b)$ linearly (see also Figure 7.14 left, p. 91) and it does not affect the value of b at all. Therefore

$$\begin{aligned}\text{Eq. 8-77: } E'(m) &= (1 - p_{\min} - p_{\max}) \frac{k(a-b) + k(c-b)}{3} + p_{\min} k(a-b) + p_{\max} k(c-b) + b \\ E'(m) - b &= k \cdot [E(m) - b]\end{aligned}$$

Note that $E'(m)$ can be expressed as a function of $E(m)$, k and b . In theory $E'(m^2)$ can analogously be expressed as a function $E(m^2)$, k and b . However, the expression consists of very lengthy mathematical terms, such that $E'(m^2)$ is best re-evaluated by applying **Eq. 8-73** to the transformed pdf. From $E'(m)$ and $E'(m^2)$ **Eq. 8-31** leads to $\text{Var}'(m)$.

An approximation of the influence of the stretching factor k on $E'(m^2)$ is found by neglecting the spikes. Without spikes, $\text{Var}(m)$ was found to be (from “Discussion of the triangular probability density function”, p. 102, repeating **Eq. 8-32**)

$$\text{Var}(m) = \frac{a^2 + b^2 + c^2 - ab - ac - bc}{18}$$

which can be rewritten as

$$\text{Eq. 8-78: } \text{Var}(m) = \frac{(a-b)^2 + (b-c)^2 + (c-a)^2}{36}$$

Again the terms $(a-b)$, $(b-c)$ and $(c-a)$ are affected linearly by the transformation and therefore

$$\text{Eq. 8-79: } \text{Var}'(m) = k^2 \cdot \text{Var}(m)$$

Remember that this is only true for the triangular pdf without spikes, but since $p_{\min}$ and $p_{\max}$ are usually small, this is a reasonable approximation for the triangular pdf with spikes.

The effect of stretching the pdf by a factor k , with fixed point at $m = \min$

Very much like the previous sub-section, we rewrite $E(m)$ (**Eq. 8-72**) this time by subtracting a in each term and then adding it again separately:

$$E(m) = (1 - p_{\min} - p_{\max}) \frac{3a + (b-a) + (c-a)}{3} + p_{\min} a + p_{\max} (c-a) + p_{\max} a$$

$$\text{Eq. 8-80: } E(m) = (1 - p_{\min} - p_{\max}) \frac{(b-a) + (c-a)}{3} + p_{\max} (c-a) + a$$

Again, the stretching transformation affects the terms $(b-a)$ and $(c-a)$ linearly and therefore

$$\text{Eq. 8-81: } E'(m) - a = k \cdot [E(m) - a]$$

Also **Eq. 8-79** still holds as an approximation. Note that for its derivation a , b and c are used absolutely interchangeably. This expresses the fact that the fixed point of the stretching transformation does not matter.

8.4 Updating Variables and Cycle Lengths as a Stochastic Process

In section 7.11 ("Possible Extensions for Updating Variables and Cycle Lengths", p. 88) it was discussed how variables and cycle lengths can be treated as a stochastic process with uncertain mean. In theory one can therefore use Bayesian updating analogously to updating Markov process parameters (see section 8.2, "Updating Markov Parameters", p. 98).

In practice the problem is however almost intractable because both the pdf for the stochastic process and the pdf for its uncertain parameter, the mean m , have triangular shape. With so many discontinuities involved, Bayesian updating becomes almost impossible.

For the case where the variance of daily fluctuations is assumed to be known (rule 1 in section 7.11, p. 93), the problem can be simplified by replacing the triangular pdf's by normal distributions while maintaining the statistical properties (mean, variance). The stochastic process which describes daily fluctuations becomes therefore a normal process with known variance.

The problem of updating a normal stochastic process with known variance has already been solved elsewhere (e.g. Raiffa & Schlaifer, 1962) and it is sufficient to repeat some of the main points.

The normal process has one unknown parameter, its mean μ . For this mean we have a prior pdf which we would like to update in order to arrive at a posterior pdf. It is advisable to choose the prior pdf such that the posterior pdf (which is essentially the product of the prior pdf and the likelihood function, see section 8.1, "Updating with Continuous Probability Distributions", p. 97) is of the same type of function as the prior one. This can be achieved by selecting the prior pdf to be a normal distribution. The normal distribution has the two parameters m' and v' (prior) or m'' and v'' (posterior), where m is the mean and v the variance. These parameters are to be updated.

Let us now assume that n daily values have been measured and denote their average by avg . Let us further denote the variance of the normal process (i.e. the expected variance of daily values = variance of the pdf in Figure 7.15 left, p. 92) by σ^2 .

The parameters of the normal distribution for μ , m' and v' , can be updated as follows:

$$\text{Eq. 8-82: } m'' = \frac{m' + \text{avg} \cdot \frac{n \cdot v'}{\sigma^2}}{1 + \frac{n \cdot v'}{\sigma^2}}; v'' = \frac{v'}{1 + \frac{n \cdot v'}{\sigma^2}}$$

A few points should be noted:

- The expression for the mean is a weighted average of m' and avg . The weight for m' is 1 and the weight for avg is $(n \cdot v' / \sigma^2)$.
- The weight of the observed average avg increases with the number of observations n , which is intuitively correct. This effect is diminished by a large variance σ^2 of the stochastic process, because if the process itself has a large variance a larger number of observations is required to get a reasonable estimate of the mean.
- The weight of avg also increases with the prior variance v' , i.e. the more uncertain we are about the true mean, the more we weight observations.
- The posterior variance v'' (and thereby the uncertainty about the true mean m) decreases with the number of observations, which is again intuitively correct. Again this effect is diminished by a large variance σ^2 of the stochastic process.
- The posterior variance v'' increases with the prior variance v' . Note that v' in the numerator has a greater influence than the expression $[1 - (n \cdot v' / \sigma^2)]$ in the denominator. The more uncertain we are in our prior prediction the more uncertain we will still be in our posterior prediction.

Going back to the original problem, which involves triangular pdf's, it is now necessary to find a new triangular pdf which has the above statistical properties. This is done analogously to the sub-section 'Extension 1: Introducing a "confidence level"' in section 7.9, p. 89. Again the prior triangular pdf is subjected to two geometric transformations such that its mean and variance have the desired values of m'' and v'' , respectively. The transformations involve stretching by a factor k with fixed point at the mode and subsequently a translation by t . Note that in contrast to section 7.11, this triangular pdf does not have any spikes and that therefore both t and k can be found exactly:

$$\text{Eq. 8-83: } k = \sqrt{\frac{v''}{v'}}; t = m'' - m'$$

8.5 The PDF of the Inverse of a Random Variable

If a random variable X follows a certain probability density function $f_X(x)$ it is often necessary to find the pdf of the inverse of X . We call the inverted random variable Y and its pdf $f_Y(y) = f_X(1/x)$.

In the DAT this case arises for example in the following situation:

Assume that daily advance rate v [m/day] is defined as a method variable. The total time T required to excavate a tunnel segment of length L with this method is found by:

Eq. 8-84: $T = L / v$

However, v is uncertain and defined by a pdf $f(v)$. Typically we are therefore interested in the expectation of T , $E(T)$. This is found by:

Eq. 8-85: $E(T) = L \cdot E(1/v) \neq L / E(v)$

It is very important to realize that the expected value of $1/v$, $E(1/v)$ *does not equal* $1/E(v)$. Instead we have to find the pdf of $(1/v)$ and find the correct expected value from this pdf.

Figure 8.7: PDF of Original Random Variable (left) and PDF of Inverted Random Variable (right)

Let us look at an interval around x^* in the original pdf, $[x^* - \Delta x; x^* + \Delta x]$, see Figure 8.7, left. The probability that the random variable X takes on a value in this interval is (assuming Δx is very small):

Eq. 8-86: $P_{X^*} = 2 \cdot \Delta x \cdot f_X(x^*)$

The inverse of all values within the original interval will be within a new interval, formed by the inverted boundaries of the original interval, $\left[\frac{1}{x + \Delta x}; \frac{1}{x - \Delta x} \right]$. (Note that the upper boundary of the original interval defines the lower boundary of the inverted interval and vice versa.

The central reasoning in this section is the following:

The probability P_{Y^*} that the inverted random variable Y takes on a value within the inverted interval must equal the probability P_{X^*} that the original random variable X takes on a value within the original interval.

This translates into the following equation:

$$f_{X^*}(x) \cdot 2 \cdot \Delta x = f_{Y^*}(y) \cdot 2 \cdot \Delta y = f_{Y^*}(y) \cdot \left(\frac{1}{x - \Delta x} - \frac{1}{x + \Delta x} \right)$$

Eq. 8-87: $f_{X^*}(x) \cdot 2 \cdot \Delta x = f_{Y^*}(y) \cdot \left(\frac{(x + \Delta x) - (x - \Delta x)}{(x - \Delta x) \cdot (x + \Delta x)} \right) = f_{Y^*}(y) \cdot \left(\frac{2\Delta x}{x^2 - (\Delta x)^2} \right)$

Again, $(\Delta x)^2$ is a very small quantity and can be neglected. We then solve for $f_{Y^*}(y)$:

Eq. 8-88: $f_Y(y^*) = f_X(x^*) \cdot x^2 = f_X(x^*) \cdot y^{-2}$

This result is valid for all continuous probability density functions.

The following two sub-sections highlight the two special cases which are important in the context of the DAT: the uniform and the triangular pdf.

The Inverted Uniform PDF

The uniform pdf is defined by:

$$\text{Eq. 8-89: } f_X(x) = \begin{cases} (b-a)^{-1} & a \leq x \leq b \\ 0 & x \text{ outside } [a;b] \end{cases}$$

The inverted uniform pdf is therefore defined by (applying **Eq. 8-88**):

$$\text{Eq. 8-90: } f_Y(y) = \begin{cases} y^{-2} \cdot (b-a)^{-1} & 1/b \leq y \leq 1/a \\ 0 & y \text{ outside } [1/b; 1/a] \end{cases}$$

Its mean is found by the integral

$$E(y) = \frac{1}{b-a} \int_{1/b}^{1/a} y^{-2} \cdot y \cdot dy = \frac{1}{b-a} \int_{1/b}^{1/a} y^{-1} \cdot dy = \frac{1}{b-a} [\ln(y)]_{1/b}^{1/a} =$$

$$\text{Eq. 8-91: } E(y) = \frac{1}{b-a} \left[\ln\left(\frac{1}{a}\right) - \ln\left(\frac{1}{b}\right) \right] = \frac{\ln(b) - \ln(a)}{b-a}$$

And similarly its 2nd order moment

$$\text{Eq. 8-92: } E(y^2) = \frac{1}{b-a} \int_{1/b}^{1/a} y^{-2} \cdot y^2 \cdot dy = \frac{1}{b-a} \int_{1/b}^{1/a} dy = \frac{1}{b-a} \left[\frac{1}{a} - \frac{1}{b} \right] = \frac{b-a}{(b-a) \cdot ab} = \frac{1}{ab}$$

The Inverted Triangular PDF

In the sub-section “Discussion of the triangular probability density function” in section 8.2 (p. 98) the pdf of the two parts of the triangle were found to be as follows ($a = \min$, $b = \text{mode}$, $c = \max$), repeating **Eq. 8-18** and **Eq. 8-19**:

$$f_1(x) = u_1 \cdot (x - a); \quad f_2(x) = u_2 \cdot (c - x)$$

Where u_1 and u_2 are defined as:

$$u_1 = \frac{2}{(c-a)(b-a)}; \quad u_2 = \frac{2}{(c-a)(c-b)}$$

By combining **Eq. 8-19** and **Eq. 8-88** the inverted triangular can be written as

Eq. 8-93: $f_{1Y}(y) = u_1 \cdot (x - a) \cdot x^2$; $f_{2Y}(y) = u_2 \cdot (c - x) \cdot x^2$

This is rewritten in terms of $y = 1 / x$:

Eq. 8-94: $f_{1Y}(y) = u_1 \cdot (y^{-3} - a \cdot y^{-2})$; $f_{2Y}(y) = u_2 \cdot (c \cdot y^{-2} - y^{-3})$

The mean is found by the integral:

$$E(y) = u_1 \int_{1/b}^{1/a} (y^{-3} - a \cdot y^{-2}) \cdot y \, dy + u_2 \int_{1/c}^{1/b} (c \cdot y^{-2} - y^{-3}) \cdot y \, dy$$
$$E(y) = u_1 \left[-y^{-1} - a \cdot \ln(y) \right]_{1/b}^{1/a} + u_2 \left[c \cdot \ln(y) + y^{-1} \right]_{1/c}^{1/b}$$

Eq. 8-95: $E(y) = u_1 [b - a + a \cdot (\ln a - \ln b)] + u_2 [b - c + c \cdot (\ln c - \ln b)]$

By multiplying out $u_1 \cdot (b - a)$ and $u_2 \cdot (b - c)$

$$E(y) = \frac{2(b-a)}{(b-a)(c-a)} - \frac{2(c-b)}{(c-b)(c-a)} + u_1 \cdot a \cdot [\ln(a) - \ln(b)] + u_2 \cdot c \cdot [\ln(c) - \ln(b)]$$

It can be seen easily that the first two terms cancel each other and therefore

Eq. 8-96: $E(y) = u_1 \cdot a \cdot [\ln(a) - \ln(b)] + u_2 \cdot c \cdot [\ln(c) - \ln(b)]$

Very similarly the 2nd order moment is found

$$E(y^2) = u_1 \int_{1/b}^{1/a} (y^{-3} - a \cdot y^{-2}) \cdot y^2 \, dy + u_2 \int_{1/c}^{1/b} (c \cdot y^{-2} - y^{-3}) \cdot y^2 \, dy$$
$$E(y^2) = u_1 [\ln(y) - a \cdot y]_{1/b}^{1/a} + u_2 [c \cdot y - \ln(y)]_{1/c}^{1/b}$$
$$E(y^2) = u_1 \left[a \cdot \left(\frac{1}{b} - \frac{1}{a} \right) + \ln(b) - \ln(a) \right] + u_2 \left[c \cdot \left(\frac{1}{b} - \frac{1}{c} \right) + \ln(b) - \ln(c) \right]$$
$$E(y^2) = u_1 \left[\frac{a(a-b)}{a \cdot b} + \ln(b) - \ln(a) \right] + u_2 \left[\frac{c(c-b)}{b \cdot c} + \ln(b) - \ln(c) \right]$$

By multiplying $u_1 \cdot (b - a)$ and $u_2 \cdot (b - c)$ out

$$E(y^2) = \frac{-2(b-a)}{b(b-a)(c-a)} + \frac{2(c-b)}{b(c-b)(c-a)} + u_1 \cdot [\ln(b) - \ln(a)] + u_2 \cdot [\ln(b) - \ln(c)]$$

Again the first two terms cancel each other and therefore

Eq. 8-97: $E(y^2) = u_1 \cdot [\ln(b) - \ln(a)] + u_2 \cdot [\ln(b) - \ln(c)]$

The variance is determined by applying **Eq. 8-31**, (see p. 105)

9. Example Project

The example project which will be presented in this chapter was created by the author to demonstrate and check the functions of the Updating Module of the DAT, as well as to get an idea of the effect of the updating procedure.

It was not possible to work on a real project during the time the Updating Module of the DAT was developed. Although there were a number of real-world applications to make predictions in the design stage (e.g. Swiss Trans-Alpine Tunnels, see Dudt and Descoeudres, 1997) these projects are not yet ready to be used with an updating process because construction has not yet started or not yet yielded a significant amount of new data.

9.1 Description of Project

The following lines provide only a qualitative description of the model project. More details, including exact numbers, can be found in Appendix A.1.

Units: The DAT are absolutely unit-free, which means that all input has to be in the same basic units for time, length and currency as well as combinations thereof. This example project uses the units **meters, days and dollars**.

Tunnel Network

The project consists of a relatively simple network of 3 tunnels. The main tunnel has a length of 20'000m and is subdivided into two unequally long parts (3'000m, 17'000m) by an intermediate access shaft of 500 m length. An auxiliary tunnel runs parallel along the longer part of the main tunnel. For ease of reference it is assumed that these tunnels are oriented in north-south direction.

Figure 9.1: Schematic Tunnel Network

Initially there are four tunnel faces: shaft top, north exit, south exit (main) and south exit (aux). Once the shaft reaches its bottom, two new faces (one for the main and one for the auxiliary tunnel) are opened towards the south exit. *No new face will be created towards the north exit.* This project consists therefore of a total of six tunnel faces, each of which is represented by a “tunnel” in the DAT. The resulting DAT tunnel network is shown in Figure 9.2.

Figure 9.2: Tunnel Network in the DAT

There are three different geometries: geometry 1 (main tunnels), geometry 2 (auxiliary tunnels) and geometry 3 (access shaft).

A number of delays are expected during the construction process:

- Access Shaft:
 - a short delay halfway down the shaft
 - a long delay to excavate a cavern at the shaft bottom
- North Exit – Shaft Bottom:-
 - a long delay before construction starts at the north exit
 - a long delay for a cavern halfway between the shaft top and shaft bottom
- Shaft – South Exit (Main):-
 - a medium delay before construction starts at the south exit
 - a medium delay before construction starts at the shaft bottom, *after the cavern there is finished*

- Shaft – South Exit (Aux.): - two short delays somewhere between the shaft bottom and the south exit

Delays are of the order of 25 days (short), 60 days (medium) or 150 days (long).

Areas

This project consists of two areas: “Shaft” between the shaft top and bottom and “Tunnel” between north and south exit (see Figure 9.1).

Zones

The geology along the area “Tunnel” can be subdivided into four zones. “Northern Sedimentary Rocks” and “Southern Sedimentary Rocks” are the northernmost and southernmost zones respectively. Between these two zones there are the zones “Metamorphic Rocks” and “Intrusion”. The boundaries between the zones are not known precisely, i.e. they are defined probabilistically.

Figure 9.3: Zone Pattern in Area “Tunnel”

The area “Shaft” consists of only one zone, named “Shaft Zone”.

Parameter Sets

“Northern Sedimentary Rocks” and “Southern Sedimentary Rocks” are assumed to have the same geotechnical properties, i.e. for these two the same parameter set is used. The other two zones have a different parameter set each. These three parameter sets which are used in zones in the area “Tunnel” have only one geotechnical parameter, “Overall Condition”. This parameter includes all relevant detailed parameters that would influence the assessment of the overall condition for the excavation process. It has the three parameter states “good”, “medium” and “bad” and it is assumed to follow a Markov process.

The “Shaft Zone” has a parameter set which consists of two geotechnical parameters, “Water” (water inflow) and “Jointing”. The former has the two states “little” and “much”. The latter has the three states “intense”, “intermediate” and “sparse”. Both parameters are assumed to follow a Markov process. In addition to this it is expected that some parameter state combinations are less likely than others, in particular:

- very unlikely: - “Water: little” and “Jointing: intense”
- “Water: much” and “Jointing: sparse”
- quite unlikely: - “Water: much” and “Jointing: intermediate”

In addition to these zone parameter sets, there is an area parameter set for the area “Tunnel”. It is expected that there are a number of fault zones. The influence of these fault zones on the overall condition for the excavation depends on the overburden. The area parameter set consists therefore of the two geotechnical parameters “Faulting” and “Overburden”. The former can assume the states “none”, “slight” and “severe” whereas the latter can only assume the states “high” or “low”. “Overburden” can be defined deterministically. “Faulting” is defined semi-deterministically.

There is no area parameter set in the area “Shaft”.

Ground Classes and Combined Ground Classes

In the case of the four zones in the area “Tunnel” the ground class corresponds directly to the state of the parameter “Overall Condition”, i.e. if the parameter state is “good”, the ground class is “good conditions” and so on.

In the “Shaft Zone” there are also three ground classes “good shaft conditions”, “medium shaft conditions” and “bad shaft conditions”. These depend on the parameter state combination of the two parameters “Jointing” and “Water”.

The area “Tunnel” can have the following ground classes: “fault neutral”, “fault important” and “fault serious”, depending on the overburden and the state of the parameter “Faulting”.

The combined ground classes for the area “Tunnel” are found as follows:

- If the area ground class is “fault neutral”, the combined ground class equals the zone ground class

- If the area ground class is “fault important”, the combined ground class is one level worse than the zone ground class (e.g. if zone ground class is “good conditions” the combined ground class is “medium conditions”)
- If the area ground class is “fault serious”, the combined ground class is always “bad conditions”

Methods, Cycle Lengths and Variables

There are three geometries and three ground classes (per area). Consequently there are nine different methods (Figure 9.4).

	geometry 1 (main tunnel)	geometry 2 (auxiliary tunnel)	geometry 3 (access shaft)
good conditions	TBM w/ Concrete Lining fast	Small TBM w/ Shotcrete only	Shaft D&B w/ Shotcrete 6cm
medium conditions	TBM w/ Concrete Lining medium	Small TBM w/ Shotcrete & Bolts	Shaft D&B w/ Shotcrete 15cm
bad conditions	TBM w/ Concrete Lining slow	Small TBM w/ Steel Arches & Shotcrete	Shaft D&B w/ Shotcrete & Bolts

Figure 9.4: Associations of Geometries, Ground Classes and Methods

Each method has one heading. The activity network of each heading is highly simplified and consists of only one activity, named “do_it_all”. This activity comprises all individual activities that would be performed during one ordinary working day. Consequently the cycle length defines the daily advance rate.

The time required for an ordinary cycle is therefore 1 [day] and the time equation would simply be “do_it_all_t = 1”. However, there can be method change delays and there are also learn parameters. In addition the function “cycle_percentage()” is used (see section 5.11, “Activities and Equations”, p. 53). The complete time equation writes therefore:

$$\text{“do_it_all_t = cycle_percentage() / this_meth_eff() + meth_change_delay()”}$$

The costs of one cycle depend on the value of the method variable “cost_per_meter” and the cycle length of this particular cycle. The equation is therefore:

$$\text{“do_it_all_c = cost_per_meter * round_length()”}$$

9.2 Initial (prior) Predictions

With the input data as described in the previous section an initial prediction for total construction costs and time is made. It results in the following time-cost scattergram (Figure 9.5) and time-position diagram (Figure 9.6).

Figure 9.5: Time-Cost Scattergram for Initial Input

In order to check the proper operation of the simulator, the expected construction times and costs are calculated in a spread-sheet based on analytical solutions, including:

- limiting state probabilities in the Markov process (see sub-section "Limiting State Probabilities", p. 30, in section 3.3)
- the expectation of a variable with triangular pdf (see sub-section "Discussion of the triangular probability density function", p. 102, in section 8.2)
- the expectation of the inverse of a variable with triangular pdf (see sub-section "The Inverted Triangular PDF", p. 124, in section 8.5)

The spread-sheet can be found in Appendix A.1. A general outline of the spread-sheet calculation is given below:

Figure 9.6: Time-Position Plot for Initial Input (black: maximum time, gray: minimum time)
The hatched areas describe the possible times and positions for breakthrough
(vertical: auxiliary tunnel, horizontal: main tunnel)

First, the average length of each zone is calculated (using b) and then within each zone the percentage of each ground class (using a) is determined. This results in the average extent of each ground class along the tunnels (main tunnel, auxiliary tunnel, access shaft). Each combination of ground class and geometry/tunnel defines deterministically which method must be used (see Figure 9.4, p. 131). Therefore the average length over which each method is used in a tunnel can be computed. For each method the time and costs to excavate this average length are calculated. This requires computing the average cost per meter (using b)

$$\text{cost} = \text{method_length} * \text{cost_per_meter}$$

and the average of the inverse of the cycle length (using c)

$$\text{time} = \text{method_length} * (1 / \text{cycle_length}).$$

Construction times for each method and each tunnel *along the critical path* are summed up. The critical path includes the access shaft and the part of the main tunnel between the shaft bottom and the south exit. To this sum the average length (using b) of all delays along the critical path must be added and finally this result can be divided by two, because the critical path is excavated from two sides simultaneously.

Total construction costs are found by summing up the costs of all methods in *all* tunnels and by adding the averages of tunnel fix costs (using b).

The spread-sheet calculation can not consider the following aspects:

- learn parameters
- method change delays
- starting probabilities for parameter states in the Markov generation mode
- correlation numbers
- information points

Comparing simulation results of the example project with the spread-sheet calculation would therefore produce misleading results because the above issues influence the simulator results. Instead, the spread-sheet calculation must be compared to a simplified version of the example project, which does exclude the above aspects. The results of the simulation of the simplified project and the spread-sheet calculation compare as follows:

	simulation (1000 simul.)	spread-sheet calculation
average construction time	1069.4 days	1069.8 days
average construction costs	\$ 396'398'000	\$ 396'378'000

The differences are extremely small, which confirms the reliability of the simulator.

Some very interesting observations about sources of uncertainty can be made by varying the way in which simulations are run. There are three basic sources of uncertainty in the DAT:

- geology-related: zones, Markov parameters, semi-deterministic segments
- method-related: cycle lengths / variables
- tunnel-related: delays / fix costs

By using four different settings for the simulations it is actually possible to find out how much each of these three sources contribute to the overall uncertainty. The following table shows the four settings together with the standard deviation (relative to the average) of the simulation result that is associated with each setting (Figure 9.7).

	treatment of variables and cycle lengths ¹	number of geology simulations	number of construction simulations per geology	standard deviation of total construction time	standard deviation of total construction costs
A	long-term average	1000	1	5.41%	3.29%
B	long-term average	1	200	2.96%	2.10%
C	daily fluctuations	1000	1	4.53%	2.39%
D	daily fluctuations	1	200	0.68%	0.23%

Figure 9.7: Time and Cost Standard Deviations (as a percentage of total time, cost) with Different Simulation Settings

These four settings include the three sources of uncertainty in varying combinations (Figure 9.8). Setting A is the original setting that was used for the simulations described earlier in this section. It includes all sources of uncertainty. Setting B excludes the uncertainty about the geology because all construction simulations are run with the same geology. Setting C excludes the uncertainty about method variables and cycle lengths. This is so because if variables (also cycle lengths) are treated as daily fluctuations they will almost perfectly average out during every single simulation, and thereby, *on average per simulation*, always use the same variable value and cycle length. Setting D, finally, excludes the uncertainty about the geology, the variables and cycle lengths.

¹ see section 5.17, "Daily Fluctuations vs. Long-Term Averages", p. 58

setting	geology-related uncertainties	method-related uncertainties	tunnel-related uncertainties
A	X	X	X
B		X	X
C	X		X
D			X

Figure 9.8: Sources of Uncertainties in Simulator Settings

These three sources of uncertainty are practically independent, although not completely:

In a geology simulation that results in a large percentage of one particular ground class (e.g. “good conditions”) the uncertainty in cycle lengths and costs per meter of methods which are used with this ground class (e.g. “TBM w/ concrete lining fast”) have more influence than others. Therefore the standard deviations with settings B and D depend somewhat on the result of the single geology simulation.

Under the assumption of complete independence among these sources of uncertainties, the variance of any combination of them is expected to be the sum of the variances of the parts of the combination. We can therefore write:

$$\text{Eq. 9-1: } \text{Var}(A) = \text{Var}(\text{geology}) + \text{Var}(\text{method}) + \text{Var}(\text{tunnel})$$

$$\text{Eq. 9-2: } \text{Var}(B) = \text{Var}(\text{method}) + \text{Var}(\text{tunnel})$$

$$\text{Eq. 9-3: } \text{Var}(C) = \text{Var}(\text{geology}) + \text{Var}(\text{tunnel})$$

$$\text{Eq. 9-4: } \text{Var}(D) = \text{Var}(\text{tunnel})$$

It follows immediately from these four equations that

$$\text{Eq. 9-5: } \text{Var}(A) = \text{Var}(B) + \text{Var}(C) - \text{Var}(D)$$

which can be rewritten in terms of standard deviations:

$$\text{Eq. 9-6: } \text{Std}(A) = \sqrt{\text{Std}(B)^2 + \text{Std}(C)^2 - \text{Std}(D)^2}$$

This is an important consistency check for the simulations. The simulated time and costs of the example project are consistent indeed, as can be seen by writing Eq. 9-5 with actual numbers from Figure 9.7 (and using relative standard deviations):

$$\text{time: } 5.41\% \approx \sqrt{2.96\% + 4.53\% - 0.68\%} = 5.37\%$$

$$\text{cost: } 3.29\% \approx \sqrt{2.10\% + 2.93\% - 0.23\%} = 3.18\%$$

The differences are quite small and not considered to be significant. It is also interesting to see how much these three sources of uncertainty contribute to the overall uncertainty (Figure 9.9), which can be computed easily from **Eq. 9-1** to **Eq. 9-4**:

	geology-related std. dev.	method-related std. dev.	tunnel-related std. dev.	total standard deviation
Time	4.48%	2.88%	0.68%	5.41%
Cost	2.38%	2.09%	0.23%	3.29%

Figure 9.9: Contributions of the Three Sources of Uncertainties

Note that unlike variances, standard deviations of individual uncertainties do not add up to total standard deviation.

9.3 Observations

“Observations” are assumed based on initial input. They basically describe the excavation process during the first 400 working days. The following guidelines are used for uncertain parameters (zone extents, extents of geotechnical parameter states, cycle lengths, etc.):

- The assumed observations are within the range [min; max] defined by the initial input (for example the “observed” average cycle length for a method is within the minimum and maximum of the triangular pdf which was defined as initial input).
- The position of the assumed observations within this range [min; max] varies from one parameter to the other. Some observations are intentionally assumed such that they are close to the minimum or maximum whereas others are close to the most likely value.
- In a few cases the assumed observations are actually outside the predicted range. For example one semi-deterministic segment which was predicted in the area parameter set was not observed at all. Another example is a short unexpected delay.

It is believed that these guidelines give the example project a sense of reality. It is unrealistic to expect observed values to exactly match the most likely values of the initial prediction. Details about the observations can be found in Appendix A.2.

9.4 Updating Process

Appendix A.3 shows in detail how parameters of the example project are updated based on their initial prediction and observed data.

There are a number of key input parameters of the DAT on which the updating process must focus. These input parameters are generally characterized as follows:

- They have a significant initial uncertainty.
- There is new information available from observations that can substantially reduce this uncertainty
- They have an significant influence on the unexcavated part of the tunnel or tunnel network

In the example project the following input parameters meet the above criteria and are considered key parameters:

- Markov process parameters for the four zones in the main tunnel
- cycle lengths and “cost_per_meter” variables for methods used in the main and auxiliary tunnels

Many parameters do not meet these criteria for various reasons (in parentheses). Nevertheless they still should be updated:

- zone boundaries (in the special case of this project the observed zones do not allow one to draw any conclusions about the unobserved zone boundaries)
- semi-deterministic segments (same reason as for zone boundaries)
- deterministic segments (there is little or no uncertainty)
- tunnel delays (same reason as for zone boundaries)
- method change delays (they don't have a significant influence on the total construction time because they do only exist in the auxiliary tunnel and the shaft, the former of which is not on the critical path and the latter of which is already excavated at the time of the updating)

- associations between geometries, ground classes and methods (the observed probabilities are very close to the predicted ones)

Finally there are some parameters that do not need to be updated at all because they have no influence whatsoever on the unexcavated part. These are all those related to the access shaft because the shaft is already finished at the time of the updating and none of the geotechnical parameters or methods are relevant to the main tunnel:

- Markov process parameters for the shaft zone
- cycle lengths and method variables of methods used in the shaft excavation.

It is worthwhile elaborating on the key input parameters mentioned above.

Markov Lengths

The probability density functions for Markov lengths are the only parameters for which updating does not require human judgment and for which the Updating Module of the DAT can make suggestions on its own. Because these suggestions are based on solid theoretical principles (Bayes' theorem) they are accepted as posterior input.

Figure 9.10: Updating the PDF for Markov Lengths: prior (circles) and posterior (diamonds) pdf with likelihood function (smooth line).

As a general rule, the posterior pdf differs significantly from the prior pdf in the following two cases:

- there is a large number of observed segments (e.g. Figure 9.10, left, 24 observations)
- there is a medium number of observed segments and the average length of the observed segments is close to the maximum of the prior prediction (e.g. Figure 9.10, right, 10 observations)

This is explained by the fact that in a Markov process only 37% of all segments are longer than the mean length. A long segment is therefore a stronger indication for a long mean length than a short segment is for a short mean length. This is also reflected by the likelihood function which has a steep slope left of its peak and a long flat slope towards increasing lengths.

Transition Probabilities

Transition probabilities (TP) are not defined probabilistically although they are uncertain (see also section 5.6, "Parameters for the Markov Process", p. 48). They are updated based on a comparison of prior and observed TP's. This involves human judgment. As a general guideline for this example project a lot of weight is given to observed TP's if the number of observed transitions is greater than 10. If this number is smaller than 5 the posterior TP is chosen very close to the prior prediction.

It is interesting to compare limiting state probabilities of the Markov process for the geotechnical parameter "Overall Condition" with prior and posterior parameters. These are influenced by changes in both the mean length and transition probabilities (Figure 9.11).

zone	good conditions		medium conditions		bad conditions	
	prior	posterior	prior	posterior	prior	posterior
north. & south. sedimentary rocks	29%	23%	50%	52%	22%	25%
metamorphic rocks	27%	29%	54%	53%	19%	17%
intrusion	42%	45%	35%	42%	23%	13%

Figure 9.11: Prior and Posterior Limiting State Probabilities for "Overall Condition"

In the zones "Northern Sedimentary Rocks" and "Southern Sedimentary Rocks" there is a shift towards worse conditions, whereas the zone "Metamorphic Rocks" remains more or less unchanged. In contrast, the zone "Intrusion" shows a shift towards better conditions. It is also interesting to check whether these differences (posterior – prior) stem from changes in transition probabilities or from changes in the pdf for mean lengths (see Figure 9.12). This table is created by first only updating mean lengths and recording the differences in limiting state probabilities due to updating mean lengths [$\Delta p(ML)$].

Afterwards also transition probabilities are updated and again the differences in limiting state probabilities (compared to the first updating stage) are recorded [$\Delta p(TP)$], this time due to updating transition probabilities.

zone	good conditions		medium conditions		bad conditions	
	$\Delta p(ML)$	$\Delta p(TP)$	$\Delta p(ML)$	$\Delta p(TP)$	$\Delta p(ML)$	$\Delta p(TP)$
north. & south. sedimentary rocks	+ 1.8%	- 7.0%	- 3.7%	+ 5.5%	+ 1.9%	+ 1.5
metamorphic rocks	+ 0.7%	+ 1.4%	- 0.1%	- 0.7%	- 0.5%	- 0.7%
intrusion	- 2.5%	+ 5.3%	+ 5.9%	+ 1.5%	- 3.4%	- 6.9%

Figure 9.12: Changes in Limiting State Probabilities due to Changes in Mean Lengths (left column) and Transition Probabilities (right column)

It can be seen that in this example project updating transition probabilities has generally a greater impact than updating mean lengths. This can be explained by the fact that most observed segment lengths fit well with the predicted mean length whereas observed transition probabilities are sometimes quite different from the predicted ones.

Cycle Lengths

For all methods there is a long record of observed cycle lengths. One can therefore be quite certain about the average cycle length. Some variation is still probable because of factors that make cycle lengths in the future systematically different from those in the past. There are reasons for changes both for increased and decreased cycle lengths in the future. Wear on machinery (especially TBM), for example, could force one to have more frequent maintenance intervals. On the other hand there may still be ways to improve the efficiency of the excavation process or even better machinery (e.g. drills) may speed up the excavation.

In a real-world application experts would use their judgment in the updating process. For this example project the author created the following guidelines for updating cycle lengths based on his own judgment:

- the mean of the posterior pdf should equal the observed mean
- the spread between the min and max value should be between 8% and 10% of the mean

In general the differences in cycle length between methods used in different ground classes have decreased due to the updating process, i.e. the observed cycle lengths in “fast” methods are lower than expected and those in “slow” methods are higher than expected (Figure 9.13).

ground class	main tunnel		auxiliary tunnel	
	prior	posterior	prior	posterior
good conditions	22.0	20.5	23.7	21.0
medium conditions	11.0	10.9	14.7	15.1
bad conditions	5.17	6.10	6.80	7.75

Figure 9.13: Mean of the PDF of Cycle Lengths Before and After Updating

Method Variable “cost_per_meter”

Everything that was written about the way in which cycle lengths are updated also applies to the “cost_per_meter” variables.

The effect of the updating process on variables is slightly different from the effect on cycle lengths. In the auxiliary tunnel predictions are very close to the observations. In the main tunnel the observed averages are about equal to predictions for bad conditions but substantially higher for good and medium conditions.

ground class	main tunnel		auxiliary tunnel	
	prior	posterior	prior	posterior
good conditions	10'667	11'433	3'167	3'200
medium conditions	12'167	13'067	7'000	7'100
bad conditions	17'333	17'100	10'333	10'983

Figure 9.14: Mean of the PDF of “cost_per_meter” Variables Before and After Updating

Remember that all “observations” were assumed arbitrarily within the range predicted by the initial input (see section 9.3, “Observations”, p. 137) and that no general conclusions can be drawn from the actual numbers.

9.5 Updated (posterior) Predictions and Comparison

Posterior predictions are made based on two levels of updating (see also section 2.7, “The Need for an Updating Process”, p. 21):

- a) partly updated: replace predictions for excavated part of the tunnel system with actual data, do not change prediction for the unexcavated part
- b) fully updated: in addition to a) update parameters that affect the prediction for the unexcavated part as described in the previous section.

The time-cost scattergrams of a) and b) together with the initial prediction are shown on the same scale in Figure 9.16 to Figure 9.18 on the following page. The averages and standard deviations of the three cases are shown in Figure 9.15.

	total construction time (days)		total construction cost (\$M)	
	average	standard dev.	average	standard dev.
Initial Input	1097	59.4	394.2	12.96
Partly Updated	1030	44.4	395.0	9.60
Fully Updated	978	25.5	401.4	6.38

Figure 9.15: Total Construction Time and Costs for Initial Input and two Levels of Updating

Figure 9.16:
Time-Cost
Scattergram
for Initial
Input

Figure 9.17:
Time-Cost
Scattergram
for Partly
Updated
Input

Figure 9.18:
Time-Cost
Scattergram
for Fully
Updated
Input

In order to confirm the reliability of the simulations of the updated project, a simplified version of the fully updated project is used to compare the results of the simulation with a spread-sheet calculation (see Appendix A.3). The simplifications and the spread-sheet are explained in section 9.2 ("Initial (prior) Predictions", p. 132). The results compare as follows:

	simulation (1000 simul.)	spread-sheet calculation
average construction time	987.1	986.7 days
average construction costs	\$ 403'730'000	\$ 404'208'000

The differences are extremely small, which confirms that the simulations are reliable.

From a comparison of Figure 9.16 to Figure 9.18 it is obvious that full updating has very significant benefits in further reducing uncertainty both about time and costs (as compared to partial updating). In addition to this it actually leads to a significantly improved prediction of the expected construction time and costs.

Already partial updating indicates that the expected construction time is lower than predicted by the initial input (see shift to the left in Figure 9.19). Full updating predicts an additional shift towards lower expected construction times.

Note that the fact that updating leads to a prediction of higher costs and lower construction time is a result of arbitrarily assuming "observations" (see section 9.3, "Observations", p. 137).

Time vs. Number of Simulation Results

Figure 9.19: Frequency Plots for Total Construction Time
(light gray: Initial Input, dark gray: Partly Updated, black: Fully Updated)

Partial updating neither significantly reduces uncertainty about construction costs nor does it indicate a shift in the expected costs (Figure 9.20). Only full updating indicates that costs will be higher than originally expected and it also reduces uncertainty.

Cost vs. Number of Simulation Results

Figure 9.20: Frequency Plot of Total Construction Costs
(light gray: Initial Input, dark gray: Partly Updated, black: Fully Updated)

It is interesting to analyze changes in standard deviation due to the three sources of uncertainty mentioned in section 9.2 ("Initial (prior) Predictions", p. 132). This is shown in Figure 9.21 for time and Figure 9.22 for costs.

Figure 9.21: Changes in the Standard Deviation of Total Construction Time due to Updating

Partial updating reduces the total uncertainty (measured by the standard deviation) in construction time by about 20% whereas full updating can reduce it by 50%. Tunnel-related uncertainty almost vanishes in both cases.

Full updating reduces method-related uncertainty even by 80%. However, this does not contribute significantly to a reduction in total uncertainty because it is dominated by geology-related uncertainty.

Figure 9.22: Changes in the Standard Deviation of Total Construction Costs due to Updating

The overall picture in cost uncertainties is very similar to time uncertainties. Partial updating leads to a reduction in total uncertainty by 25% whereas full updating can again reduce it by 50%. Tunnel-related uncertainties are still present (reduced by about 50%) but not significant compared to the total uncertainty. Full updating has again a particularly strong effect on method-related uncertainty.

The time-position diagrams further illustrate the benefits of full updating. The set of likely breakthrough times / positions (hatched areas in Figure 9.23) has significantly decreased.

Figure 9.23: Time-Position Graphs for Initial Input (top), Partly Updated Project (left) and Fully Updated Project (right)
The hatched areas define possible breakthrough points for the main tunnel (horizontal) and auxiliary tunnel (vertical).

10. Conclusion and Outlook

10.1 Conclusion

Tunneling involves many uncertainties that need to be considered when making predictions about construction costs and time. The Decision Aids for Tunnel (DAT) allow one to assess all these individual uncertainties and combine them in order to arrive at the overall uncertainty.

The presented work discusses the need for a proper updating procedure during the actual construction process and presents such a procedure. Updated predictions can be used in many ways, for example to improve scheduling, financial planning and resource allocation. Proper updating is more than just replacing the predictions about the excavated part of the tunnel with actual data. It is also a learning process which uses information from the excavated part to refine predictions about the unexcavated part. This learning process can significantly improve the precision of the predictions.

In this thesis the presented procedure is applied to an example project. In the example project full updating (with learning effect) approximately doubled the reduction in uncertainty as compared to partial updating (without learning effect). Full updating not only reduces the uncertainty but also indicates trends in construction costs and time. In the particular case it indicated that construction costs will probably be higher than originally expected, whereas construction time is very likely to be lower than expected. These two trends were not revealed by partial updating only.

The implementation of this updating process in the DAT puts the emphasis on presenting relevant data to experts (the users of the Updating Module) such that they can quickly arrive at more precise predictions for individual uncertainties. This often involves processing construction data to make them comparable to input parameters of the DAT. Generally a graphical presentation of the information further facilitates the updating process for the experts. In one case the DAT are also able to suggest updated input parameters by comparing the initial input parameters with actual data from the construction. In general, however, the decision about how to update each input parameter stays with the expert. At least to some extent, updating always requires human judgment.

Therefore, the thesis also provides guidelines which assist the user in his judgment during the updating process.

A significant part of the total effort required for the updating process is in collecting data during construction. However, much of the data would be collected anyway and can also be used for the updating process. The key to minimize the additional effort lies in a well-planned overall concept for data acquisition.

A number of extensions to the Updating Module have been suggested in this thesis. All of them aim at making the Updating Module capable of suggesting updated values for a greater number of input parameters. However, these improvements of the updating process all come at the expense of a more complex procedure for defining initial input parameters. Experience has shown that defining input parameters is already quite demanding. Adding these extensions might therefore prove counter-productive and it was decided not to implement them right now but to describe them in sufficient detail to facilitate future implementation.

10.2 Outlook

Based on this research the following recommendations for future work can be made:

At this point only a simplified version of the Resource Module is implemented. Apart from implementing a more comprehensive Resource Module, a future extension could also include updating uncertain parameters in this module, such as resource demands and/or production.

Future research could also consider applying the DAT to assess risks which involve the completed tunnel, particularly on a long-term basis. Examples are

- the adverse effects of aggressive water on concrete linings, which depend on such things as the geologic conditions, the concrete quality, the quality of sealing work (if present) and other factors which are difficult to predict
- the occurrence of swelling rock and associated damages to the tunnel lining.

Although these examples are long-term effects that occur long after the excavation is completed (and consequently also after the geologic conditions are known) a use of the DAT can still be justified because one usually has to decide on a tunnel design *before* the exact geologic conditions are known.

References

- Bronstein, I.N., Semendjajew K.A., Musiol G., Mühlig H.; Taschenbuch der Mathematik; Verlag Harri Deutsch, Thun, Frankfurt a. M., 1993
- Chan, M.H.C.; A Geological Prediction and Updating Model in Tunneling; S.M. Thesis submitted at the Massachusetts Institute of Technology, 1981
- Cox, D.R., Miller, H.D.; The Theory of Stochastic Processes; Methuen and Co. Ltd., London, 1965
- De Neufville, R., Clark, J., Field, F.R.; 1999; "Risk Assessment"; lecture notes course ESP.71, Massachusetts Institute of Technology
- Dudt, J.-P., Descoeudres, F.; Basistunnel Gotthard und Loetschberg: Bausimulationen mit den "Entscheidungshilfen für den Tunnelbau"; Ecole Polytechnique Fédérale de Lausanne, Institut des Sols, Roches et Fondations; Report R385, Feb. 1997
- Einstein, H.H.; Indermitte, C.A.; et al.: Creating the Basis for Risk Assessment in Tunneling – The Decision Aids for Tunneling (DAT); Conference on Reducing Risk in Tunnel Design and Construction, Basel, Dec. 1998
- Halabe, V.B.; Resource Modeling for DAT; Ph.D. Thesis submitted at the Massachusetts Institute of Technology, 1995
- Howard, R.A.; Dynamic Probabilistic Systems, Vols. I and II; John Wiley and Sons, Inc., New York, 1971
- Raiffa, H., Schlaifer R.; Applied Statistical Decision Theory; Division of Research, Harvard Business School, Boston, 1962
- Veneziano, D; 1980; "Assessment of System Behavior"; lecture notes course 1.154, Massachusetts Institute of Technology

Appendix

Appendix A: Model Project	155
A.1 Initial Input	155
A.2 Observations	172
A.3 Updating	179
Appendix B: Database Structure and Code	212
B.1 Database Structure and Naming Conventions	212
Naming Conventions	213
Update Phases	214
Areas and Zones	214
Tunnels, Tunnel Network, Tunnel Delays	215
Parameter Sets, Parameters and Parameter States	216
Methods, Cycle Lengths, Method Variables and Method Change Delays	217
Ground Classes, Correlations and Associations	217
General Variables	218
Resources and Repositories	218
Others	219
B.2 Relevant Code	219
Module: Auxiliary	220
Module: Checker	221
Module: Export	236
Module: Graphs	262
Module: Import	274
Module: ManageUpdatePhase	296
Module: Observations	301
Module: Results	324
Module: Transfer	329
Form_fAssocAdd	332
Form_fAssocPost	333
Form_fChangeDelaysAdd	333
Form_fChangeDelaysPost	334
Form_fCycleMain	335
Form_fGenVariablesMain	336
Form_fMarkovMain	338
Form_fNonMarkovSemiDeterPrior	339
Form_fStockAdd	340
Form_fStocksPost	341
Form_fTransMain	341
Form_fTunDelaysMain	341
Form_fTunDelaysPost	342
Form_fUpdatePhases	343
Form_fVariablesMain	344
Form_fZonesPrior	346
Form_ofZonesSeq	347
Form_pfAreasMain	349

Form_pfAreasZones	349
Form_pfCorrelMain	351
Form_pfGCMain	354
Form_pfGenVariablesMain	358
Form_pfGeometires	359
Form_pfMarkovMainForm	359
Form_pfMethodHeadsMain	360
Form_pfMethodHeadsSub	360
Form_pfMethodsLearnMain	361
Form_pfMethodsSub	362
Form_pfMethodVariablesMain	362
Form_pfMethodVariablesSub	363
Form_pfNonMarkovMain	363
Form_pfParamMain	364
Form_pfParamSet	365
Form_pfParamSetParam	366
Form_pfTunnelDetailsMain	372
Form_pfTunnelNetTunnels	373

Appendix A: Model Project

A.1 Initial Input

Definition of Tunnel Network

North Exit - Shaft		Geometry: 1		Area: Tunnel			
opp. tunnel:	start pos	0.0	fix cost	2,500,000	3,000,000	3,500,000	
	end pos.	3,000.0	time cost	0	0	0	
Delays	start	dur. min	mode	max	Repositories	up to position	up/down changes
time-dependent	0	150	180	200			
position-dependent	1500	120	150	180			
Shaft - South Exit (Main)		Geometry: 1		Area: Tunnel			
opp. tunnel:	start pos	3,000.0	fix cost	1,000,000	1,500,000	2,500,000	
South Exit - Shaft (Main)	end pos.	20,000.0	time cost	0	0	0	
Delays	start	dur. min	mode	max	Repositories	up to position	up/down changes
position-dependent	0	40	50	60			8,000.0
South Exit - Shaft (Main)		Geometry: 1		Area: Tunnel			
opp. tunnel:	start pos	20,000.0	fix cost	3,500,000	4,000,000	4,500,000	
Shaft - South Exit (Main)	end pos.	3,000.0	time cost	0	0	0	
Delays	start	dur. min	mode	max	Repositories	up to position	up/down changes
time-dependent	0	50	70	100			
time-dependent	299	0	0	0			
Shaft - South Exit (Aux)		Geometry: 2		Area: Tunnel			
opp. tunnel:	start pos	3,000.0	fix cost	500,000	700,000	1,000,000	
South Exit - Shaft (Aux)	end pos.	20,000.0	time cost	0	0	0	
Delays	start	dur. min	mode	max	Repositories	up to position	up/down changes
time-dependent	500	10	20	30			8,000.0
South Exit - Shaft (Aux)		Geometry: 2		Area: Tunnel			
opp. tunnel:	start pos	20,000.0	fix cost	2,000,000	2,500,000	3,000,000	
Shaft - South Exit (Aux)	end pos.	3,000.0	time cost	0	0	0	
Delays	start	dur. min	mode	max	Repositories	up to position	up/down changes
position-dependent	7000	20	30	40			

Access Shaft

Geometry: 3 Area: Shaft

opp. tunnel:	start pos	0.0	fix cost	7,500,000	9,350,000	11,150,000
	end pos.	500.0	time cost	0	0	0

Delays	start	dur. min	mode	max	Repositories	up to position	up/down changes
position-dependent	200	20	30	40			
position-dependent	500	130	150	170

Nodes	type	position x	position y
in aux. tunnel	OR	500.0	500.0
in main tunnel	OR	500.0	300.0
north exit	OR	0.0	400.0

shaft bottom	OR	200.0	400.0
shaft top	OR	200.0	700.0
south exit	OR	800.0	400.0

Arcs	type	start node	end node	tunnel
0	normal	north exit	shaft bottom	North Exit - Shaft
1	normal	shaft bottom	in aux. tunnel	Shaft - South Exit (Aux)
2	normal	shaft bottom	in main tunnel	Shaft - South Exit (Main)
3	normal	south exit	in aux. tunnel	South Exit - Shaft (Aux)
4	normal	south exit	in main tunnel	South Exit - Shaft (Main)
5	normal	shaft top	shaft bottom	Access Shaft

Initial Definition of Areas and Zones

Area: Shaft				Length: 500.0		Parameter Set:		
				prob. min	min	mode	max prob. max	
0	Shaft Zone			length:	0.00	0.0	0.0	0.00
	param. set: 4	gen. mode: End Pos	endpos	0.00	500.0	500.0	500.0	0.00

Area: Tunnel				Length: 20,000.0		Parameter Set: 0		
				prob. min	min	mode	max prob. max	
0	Northern Sedimentary Rocks			length:	0.00	0.0	0.0	0.00
	param. set: 1	gen. mode: End Pos	endpos	0.10	2,200.0	2,600.0	2,900.0	0.10
1	Metamorphic Rocks			length:	0.00	7,500.0	-1.0	9,000.0
	param. set: 2	gen. mode: Length	endpos	0.00	0.0	0.0	0.0	0.00
2	Intrusion			length:	0.00	5,000.0	6,000.0	7,000.0
	param. set: 3	gen. mode: Both	endpos	0.00	16,000.0	17,000.0	17,500.0	0.00
3	Southern Sedimentary Rocks			length:	0.00	0.0	0.0	0.00
	param. set: 1	gen. mode: End Pos	endpos	0.00	20,000.0	-1.0	20,000.0	0.00

Initial Input for Parameter Sets and Parameters

Parameters	Parameter States
Faulting	none severe slight
Jointing	intense intermediate sparse
Overall Condition	bad good medium
Overburden	high low
Water	little much

Param. Set	Parameter	generation mode	start. prob.
0	Faulting Overburden	Semi-Deterministic Deterministic	User Input User Input
1	Overall Condition	Markov	User Input
2	Overall Condition	Markov	Steady State
3	Overall Condition	Markov	Steady State
4	Jointing Water	Markov Markov	User Input User Input

Initial Input for (Semi-) Deterministic Generation Mode

Parameter Set: 0

Parameter: Faulting

Semi-Deterministic Segments

	parameter state		prob. min	min	mode	max	prob. max
1	none	length:	0.00	0.0	0.0	0.0	0.00
	gen. mode: End Pos	endpos:	0.00	300.0	400.0	500.0	0.00
2	slight	length:	0.00	50.0	100.0	150.0	0.00
	gen. mode: Length	endpos:	0.00	0.0	0.0	0.0	0.00
3	severe	length:	0.00	20.0	30.0	50.0	0.00
	gen. mode: Length	endpos:	0.00	0.0	0.0	0.0	0.00
4	slight	length:	0.00	20.0	50.0	80.0	0.00
	gen. mode: Length	endpos:	0.00	0.0	0.0	0.0	0.00
5	none	length:	0.00	0.0	0.0	0.0	0.00
	gen. mode: End Pos	endpos:	0.00	4,000.0	4,400.0	4,700.0	0.00
6	slight	length:	0.00	100.0	150.0	200.0	0.00
	gen. mode: Length	endpos:	0.00	0.0	0.0	0.0	0.00
7	severe	length:	0.00	50.0	80.0	100.0	0.00
	gen. mode: Length	endpos:	0.00	0.0	0.0	0.0	0.00
8	none	length:	0.00	0.0	0.0	0.0	0.00
	gen. mode: End Pos	endpos:	0.00	13,200.0	13,400.0	14,000.0	0.00
9	slight	length:	0.00	100.0	150.0	200.0	0.00
	gen. mode: Length	endpos:	0.00	0.0	0.0	0.0	0.00
10	severe	length:	0.00	20.0	80.0	150.0	0.00
	gen. mode: Length	endpos:	0.00	0.0	0.0	0.0	0.00
11	slight	length:	0.00	50.0	150.0	300.0	0.00
	gen. mode: Length	endpos:	0.00	0.0	0.0	0.0	0.00
12	none	length:	0.00	0.0	0.0	0.0	0.00
	gen. mode: End Pos	endpos:	0.00	20,000.0	-1.0	20,000.0	0.00

Parameter Set: 0

Parameter: Overburden

Deterministic Segments

parameter state	end position	parameter state	end position	parameter state	end position
low	1,000.0	high	2,350.0	low	3,700.0
high	12,800.0	low	16,500.0	high	18,100.0
low	20,000.0				

Initial Input for Markov Generation Mode

Parameter Set: 1		Parameter: Overall Condition			Start Prob.:	User Input
Mean Lengths		min	mode	max	start prob	
bad		30.0	60.0	90.0	1.00	
good		60.0	90.0	120.0	0.00	
medium		50.0	120.0	190.0	0.00	
Transition Probabilities			from param. state		to param. state	prob
good	medium	0.50	good	bad	0.50	
medium	good	0.50	medium	bad	0.50	
bad	good	0.30	bad	medium	0.70	
Parameter Set: 2		Parameter: Overall Condition			Start Prob.:	Steady State
Mean Lengths		min	mode	max	start prob	
bad		30.0	-1.0	80.0	0.33	
good		80.0	-1.0	160.0	0.33	
medium		80.0	-1.0	160.0	0.34	
Transition Probabilities			from param. state		to param. state	prob
good	medium	0.80	good	bad	0.20	
medium	good	0.35	medium	bad	0.65	
bad	good	0.20	bad	medium	0.80	
Parameter Set: 3		Parameter: Overall Condition			Start Prob.:	Steady State
Mean Lengths		min	mode	max	start prob	
bad		30.0	70.0	100.0	0.33	
good		80.0	120.0	160.0	0.33	
medium		40.0	60.0	120.0	0.34	
Transition Probabilities			from param. state		to param. state	prob
good	medium	0.67	good	bad	0.33	
medium	good	0.50	medium	bad	0.50	
bad	good	0.33	bad	medium	0.67	

Parameter Set: 4 Parameter: Jointing Start Prob.: User Input

Mean Lengths		min	mode	max	start prob
intense		10.0	20.0	30.0	0.00
intermediate		30.0	40.0	50.0	0.00
sparse		50.0	75.0	90.0	1.00
Transition Probabilities			from param. state	to param. state	prob
sparse	intermediate	0.90	sparse	intense	0.10
intermediate	sparse	0.50	intermediate	intense	0.50
intense	sparse	0.20	intense	intermediate	0.80

Parameter Set: 4 Parameter: Water Start Prob.: User Input

Mean Lengths		min	mode	max	start prob
little		30.0	50.0	70.0	1.00
much		10.0	20.0	40.0	0.00
Transition Probabilities			from param. state	to param. state	prob
little	much	1.00	much	little	1.00

Information Points

Identifier: **Boring A**

Parameter Set: 2 Position: 6,000.0 Use Likelihood: no

parameter and parameter state	prob
Overall Condition good	0.80
Overall Condition medium	0.20

Identifier: **Boring B**

Parameter Set: 3 Position: 12,000.0 Use Likelihood: no

parameter and parameter state	prob
Overall Condition good	0.50
Overall Condition medium	0.35
Overall Condition bad	0.15

Initial Definition of Correlation Numbers

Parameter Set: 4

parameter \ correl. no	0.2	0.3	0.7
Jointing	sparse	intense	intermediate
Water	much	little	much

Definition of Ground Classes

Parameter Set: 0

parameter \ GC	fault important	fault important	fault neutral	fault neutral
Faulting	severe	slight	slight	none
Overburden	low	high	low	< any state>
	fault serious			
	severe			
	high			

Parameter Set: 1

parameter \ GC	bad conditions	good conditions	medium conditions
Overall Condition	bad	good	medium

Parameter Set: 2

parameter \ GC	bad conditions	good conditions	medium conditions
Overall Condition	bad	good	medium

Parameter Set: 3

parameter \ GC	bad conditions	good conditions	medium conditions
Overall Condition	bad	good	medium

Parameter Set: 4

parameter \ GC	bad shaft condition	good shaft conditio	medium shaft condi	medium shaft condi
Jointing	< any state>	< any state>	sparse	intense
Water	much	little	much	little

Definition of Combined Ground Classes

area ground class	zone ground class	combined ground class
< any area ground class >	bad conditions	bad conditions
< any area ground class >	bad shaft conditions	bad shaft conditions
< any area ground class >	good conditions	good conditions
< any area ground class >	good shaft conditions	good shaft conditions
< any area ground class >	medium conditions	medium conditions
< any area ground class >	medium shaft conditions	medium shaft conditions
fault important	good conditions	medium conditions
fault important	medium conditions	bad conditions
fault serious	good conditions	bad conditions
fault serious	medium conditions	bad conditions

Initial Input for General Variables and Activities

Activities:	priority
do_it_all	0
cost: do_it_all_c = round_length()*cost_per_meter	
time: do_it_all_t = method_change_delay()+cycle_percentage()/this_meth_eff()	

There are no general variables in this project.

Definition of Activity Networks

Method: Shaft D&B w/ Shotcrete & Bolts								Heading 0
Nodes	type	position x	position y	Arcs	type	start node	end node	activity
end	OR	800.0	400.0	0	normal	start	end	do_it_all
start	OR	200.0	400.0

This activity network is used for ALL of the following methods.

Definition of Methods

method name: unified round cycle length

Shaft D&B w/ Shotcrete & Bolts ☐ -1.0

Headings	distance:	min	max	cycle length:	prob. min	min	mode	max	prob. max
Heading 0	0.0	0.0		0.00	1.9	2.0	2.1	0.00	

Variables	prob. min	min	mode	max	prob. max
cost_per_meter	0.00	22500	25000	28000	0.00

Learn Parameters	up to cycle	efficiency	up to cycle	efficiency	up to cycle	efficiency
5	0.50	10	0.75			

Shaft D&B w/ Shotcrete 15cm ☐ -1.0

Headings	distance: min	max	cycle length: prob. min	min	mode	max	prob. max
Heading 0	0.0	0.0	0.00	2.8	3.0	3.2	0.00

Variables	prob. min	min	mode	max	prob. max
cost_per_meter	0.00	15000	17000	19000	0.00

Learn Parameters	up to cycle	efficiency	up to cycle	efficiency	up to cycle	efficiency
4	0.50	8	0.75			

Shaft D&B w/ Shotcrete 6cm ☐ -1.0

Headings	distance: min	max	cycle length: prob. min	min	mode	max	prob. max
Heading 0	0.0	0.0	0.00	4.2	4.7	5.2	0.00

Variables	prob. min	min	mode	max	prob. max
cost_per_meter	0.00	11500	13000	15000	0.00

Learn Parameters	up to cycle	efficiency	up to cycle	efficiency	up to cycle	efficiency
4	0.50	8	0.75			

Definition of Methods

method name: unified round cycle length

Small TBM w/ Bolts & Shotcrete ☐ -1.0

Headings	distance: min	max	cycle length: prob. min	min	mode	max	prob. max
Heading 0	0.0	0.0	0.00	12.0	15.0	17.0	0.00
Variables			prob. min	min	mode	max	prob. max
cost_per_meter			0.00	6000	7000	8000	0.00
Learn Parameters	up to cycle	efficiency	up to cycle	efficiency	up to cycle	efficiency	
5	0.50	10	0.75				

Small TBM w/ Shotcrete only ☐ -1.0

Headings	distance: min	max	cycle length: prob. min	min	mode	max	prob. max
Heading 0	0.0	0.0	0.00	21.0	24.0	26.0	0.00
Variables			prob. min	min	mode	max	prob. max
cost_per_meter			0.00	2900	3200	3400	0.00
Learn Parameters	up to cycle	efficiency	up to cycle	efficiency	up to cycle	efficiency	
4	0.50	8	0.75				

Small TBM w/ Steel Arches & Shotcre ☐ -1.0

Headings	distance: min	max	cycle length: prob. min	min	mode	max	prob. max
Heading 0	0.0	0.0	0.10	5.0	7.0	9.0	0.00
Variables			prob. min	min	mode	max	prob. max
cost_per_meter			0.00	9000	10500	11500	0.00
Learn Parameters	up to cycle	efficiency	up to cycle	efficiency	up to cycle	efficiency	
10	0.50	20	0.75				

Definition of Methods

method name: unified round cycle length

TBM w/ Concrete Lining fast ☐ -1.0

Headings	distance: min	max	cycle length: prob. min	min	mode	max	prob. max
Heading 0	0.0	0.0	0.00	19.0	22.0	25.0	0.00

Variables	prob. min	min	mode	max	prob. max
cost_per_meter	0.00	9500	10500	12000	0.00

Learn Parameters	up to cycle	efficiency	up to cycle	efficiency	up to cycle	efficiency
10	0.50	20	0.75			

TBM w/ Concrete Lining medium ☐ -1.0

Headings	distance: min	max	cycle length: prob. min	min	mode	max	prob. max
Heading 0	0.0	0.0	0.00	9.0	11.0	13.0	0.00

Variables	prob. min	min	mode	max	prob. max
cost_per_meter	0.00	10500	12000	14000	0.00

Learn Parameters	up to cycle	efficiency	up to cycle	efficiency	up to cycle	efficiency
10	0.50	20	0.75			

TBM w/ Concrete Lining slow ☐ -1.0

Headings	distance: min	max	cycle length: prob. min	min	mode	max	prob. max
Heading 0	0.0	0.0	0.00	4.0	5.0	6.5	0.00

Variables	prob. min	min	mode	max	prob. max
cost_per_meter	0.00	15500	17500	19000	0.00

Learn Parameters	up to cycle	efficiency	up to cycle	efficiency	up to cycle	efficiency
10	0.50	20	0.75			

Definition of Method Change Delays

from method	to method	delay	to method	delay
Shaft D&B w/ Shotcrete 15c	Shaft D&B w/ Shotcrete & Bolt	0.024		
Shaft D&B w/ Shotcrete 6cm	Shaft D&B w/ Shotcrete & Bolt	0.04	Shaft D&B w/ Shotcrete 15cm	0.024
Small TBM w/ Bolts & Shotcrete	Small TBM w/ Steel Arches &	0.08		
Small TBM w/ Shotcrete only	Small TBM w/ Bolts & Shotcrete	0.04	Small TBM w/ Steel Arches &	0.08
Small TBM w/ Steel Arches	Small TBM w/ Bolts & Shotcrete	0.056	Small TBM w/ Shotcrete only	0.04

Initial Association of Geometries, Ground Classes and Methods

ground class	method	prob	method	prob
Geometry: 1				
good conditions	TBM w/ Concrete Lining fast	1.00		
medium conditions	TBM w/ Concrete Lining medi	1.00		
bad conditions	TBM w/ Concrete Lining slow	1.00		
Geometry: 2				
good conditions	Small TBM w/ Shotcrete only	1.00		
medium conditions	Small TBM w/ Bolts & Shotcr	1.00		
bad conditions	Small TBM w/ Steel Arches &	1.00		
Geometry: 3				
bad shaft conditions	Shaft D&B w/ Shotcrete & Bo	1.00		
good shaft conditions	Shaft D&B w/ Shotcrete 6cm	1.00		
medium shaft conditions	Shaft D&B w/ Shotcrete 15c	1.00		

Calculation of Expected Construction Time and Cost

(Simplified Initial Input)

Calculation of average cycle lengths:

	min	mode	max	average
Shaft D&B w/ Shotcrete & Bolts	1.90	2.00	2.10	2.00
Shaft D&B w/ Shotcrete 15cm	2.80	3.00	3.20	3.00
Shaft D&B w/ Shotcrete 6cm	4.20	4.70	5.20	4.69
Small TBM w/ Shotcrete only	21.00	24.00	26.00	23.62
Small TBM w/ Bolts & Shotcrete	12.00	15.00	17.00	14.59
Small TBM w/ Steel Arches & Shotcrete	5.00	7.00	9.00	6.90
TBM w/ Concrete Lining fast	19.00	22.00	25.00	21.93
TBM w/ Concrete Lining medium	9.00	11.00	13.00	10.94
TBM w/ Concrete Lining slow	4.00	5.00	6.50	5.12

A) Shaft Zone

Shaft Zone					mean length:	500
	E(length)	Transition Probabilities			Extent Share	Avg. Extent
<i>Jointing</i>		intense	interm.	sparse		
intense	20		0.80	0.20	11.8%	59
intermediate	40	0.50		0.50	42.1%	210
sparse	72	0.10	0.90		46.1%	231
<i>Water</i>		little	much			
little	50		1.00		68%	341
much	23	1.00			32%	159

Extent share of parameter combinations:

Methods used for parameter combinations:

Jointing\Water	little	much		little	much
intense	8%	4%		intense	shotcrete 15cm
intermediate	29%	13%		intermediate	shotcrete 6cm
sparse	31%	15%		sparse	shotcrete 6cm

Method	Avg. Extent	Avg. Cycle Length	Avg. Cost per meter	Time	Cost
Shaft D&B w/ Shotcrete & Bolts	86	2.0	25167	43	2,156,575
Shaft D&B w/ Shotcrete 15cm	114	3.0	17000	38	1,931,818
Shaft D&B w/ Shotcrete 6cm	301	4.7	13167	64	3,958,962
				145	8,047,355
Tunnel Delays / Fix Costs				180	9,333,333
Total Time and Cost				325	17,380,688

B) Northern Sedimentary Rocks

				mean length:	2567
	E(length)	Transition Probabilities		Extent Share	Known Extent
<i>Overall Condition</i>		good	medium	bad	
good	90		0.50	0.50	28.5%
medium	120	0.50		0.50	49.6%
bad	60	0.30	0.70		21.9%

<i>Main Tunnel</i>	<i>remaining length</i>	<i>2567</i>		<i>unknown length</i>	<i>2567</i>
			Extent m	Extent Share	
area ground classes:	fault serious		0	0.0%	
	fault important		33	1.3%	
	fault neutral		2534	98.7%	
Method	Avg. Extent	Avg. Cycle Length	Avg. Cost per meter	Time	Cost
TBM w/ Concrete Lining fast	721	21.9	10667	33	7,693,710
TBM w/ Concrete Lining mediu	1267	10.9	12167	116	15,415,352
TBM w/ Concrete Lining slow	578	5.1	17333	113	10,025,812
	2567			262	33,134,874
Tunnel Delays / Fix Costs				327	3,000,000
Total Time and Cost				588	36,134,874

C) Southern Sedimentary Rocks

	E(length)	Transition Probabilities		mean length:	3183
				Extent Share	Known Extent
<i>Overall Condition</i>		good	medium	bad	
good	90		0.50	0.50	28.5%
medium	120	0.50		0.50	49.6%
bad	60	0.30	0.70		21.9%
<i>Main Tunnel</i>	<i>remaining length</i>	<i>3183</i>		<i>unknown length</i>	<i>3183</i>
			Extent m	Extent Share	
area ground classes:	fault serious		0	0.0%	
	fault important		0	0.0%	
	fault neutral		3183	100.0%	
Method	Avg. Extent	Avg. Cycle Length	Avg. Cost per meter	Time	Cost
TBM w/ Concrete Lining fast	906	21.9	10667	41	9,666,482
TBM w/ Concrete Lining mediu	1580	10.9	12167	144	19,224,452
TBM w/ Concrete Lining slow	697	5.1	17333	136	12,082,493
	3183			322	40,973,427
Tunnel Delays / Fix Costs				73	4,000,000
Total Time and Cost				395	44,973,427

<i>Auxiliary Tunnel</i>	<i>remaining length</i>	<i>3183</i>			
			Extent m	Extent Share	
area ground classes:	fault serious		0	0.0%	
	fault important		0	0.0%	
	fault neutral		3183	100.0%	
Method	Avg. Extent	Avg. Cycle Length	Avg. Cost per meter	Time	Cost
Small TBM w/ Shotcrete only	906	23.6	3167	38	2,869,949
Small TBM w/ Bolts & Shotcret	1580	14.6	7000	108	11,060,341
Small TBM w/ Steel Arches & S	697	6.9	10333	101	7,202,931
	3183			248	21,133,221
Tunnel Delays / Fix Costs					2,500,000
Total Time and Cost				248	23,633,221

D) Metamorphic Rocks

					mean length:	8250
					Extent Share	Known Extent
<i>Overall Condition</i>	E(length)	Transition Probabilities				
		good	medium	bad		
good	120		0.80	0.20	27.1%	
medium	120	0.35		0.65	54.2%	
bad	55	0.20	0.80		18.6%	
<i>Main Tunnel</i>	<i>remaining length</i>	8250			<i>unknown length</i>	8250
				Extent m	Extent Share	
area ground classes:		fault serious		77	0.9%	
		fault important		150	1.8%	
		fault neutral		8023	97.2%	
Method	Avg. Extent	Avg. Cycle Length	Avg. Cost per meter	Time	Cost	
TBM w/ Concrete Lining fast	2176	21.9	10667	99	23,208,499	
TBM w/ Concrete Lining medium	4392	10.9	12167	402	53,439,114	
TBM w/ Concrete Lining slow	1682	5.1	17333	329	29,156,456	
	8250			830	105,804,069	
Tunnel Delays / Fix Costs				50	1,666,667	
Total Time and Cost				880	107,470,736	
<i>Auxiliary Tunnel</i>	<i>remaining length</i>	7817				
				Extent m	Extent Share	
area ground classes:		fault serious		77	1.0%	
		fault important		150	1.9%	
		fault neutral		7590	97.1%	
Method	Avg. Extent	Avg. Cycle Length	Avg. Cost per meter	Time	Cost	
Small TBM w/ Shotcrete only	2058	23.6	3167	87	6,518,366	
Small TBM w/ Bolts & Shotcrete	4157	14.6	7000	285	29,099,751	
Small TBM w/ Steel Arches & S	1601	6.9	10333	232	16,546,694	
	7817			604	52,164,811	
Tunnel Delays / Fix Costs				20	733,333	
Total Time and Cost				624	52,898,145	

E) Intrusion

				mean length:	6000
	E(length)	Transition Probabilities		Extent Share	Known Extent
<i>Overall Condition</i>		good	medium	bad	
good	120		0.67	0.33	42.1%
medium	73	0.50		0.50	34.5%
bad	67	0.33	0.67		23.4%
<i>Main Tunnel</i>	<i>remaining length</i>	<i>6000</i>		<i>unknown length</i>	<i>6000</i>
area ground classes:		fault serious		Extent m	Extent Share
		fault important		0	0.0%
		fault neutral		83	1.4%
				5917	98.6%
Method	Avg. Extent	Avg. Cycle Length	Avg. Cost per meter	Time	Cost
TBM w/ Concrete Lining fast	2492	21.9	10667	114	26,581,645
TBM w/ Concrete Lining medium	2076	10.9	12167	190	25,253,665
TBM w/ Concrete Lining slow	1432	5.1	17333	280	24,828,861
	6000			583	76,664,171
Tunnel Delays / Fix Costs					
Total Time and Cost				583	76,664,171
<i>Auxiliary Tunnel</i>	<i>remaining length</i>	<i>6000</i>			
area ground classes:		fault serious		Extent m	Extent Share
		fault important		0	0.0%
		fault neutral		83	1.4%
				5917	98.6%
Method	Avg. Extent	Avg. Cycle Length	Avg. Cost per meter	Time	Cost
Small TBM w/ Shotcrete only	2492	23.6	3167	105	7,892,010
Small TBM w/ Bolts & Shotcrete	2076	14.6	7000	142	14,529,108
Small TBM w/ Steel Arches & S	1432	6.9	10333	208	14,801,628
	6000			455	37,222,746
Tunnel Delays / Fix Costs				30	
Total Time and Cost				485	37,222,746
Summary			Length		Cost
Shaft			500		17,380,688
Main Tunnel			20,000		265,243,208
Auxiliary Tunnel			17,000		113,754,111
Total Construction Time and Costs				1069.8	396,378,008

A.2 Observations

Observed Parameter State Sequences

Area: **Shaft**

Length:

500.0

Parameter Set:

order	zone name			status	length	end pos.	param. set
0	Shaft Zone			Observed	0.0	500.0	4
parameter state		beginpos.	endpos.	partly visible			
parameter: Jointing				sparse		0.0	28.0 ☐
intense		28.0	56.0	☐ intermediate		56.0	112.0 ☐
sparse		112.0	188.0	☐ intermediate		188.0	235.0 ☐
sparse		235.0	322.0	☐ intense		322.0	343.0 ☐
intermediate		343.0	410.0	☐ intense		410.0	423.0 ☐
intermediate		423.0	456.0	☐ sparse		456.0	500.0 ☐
parameter: Water				little		0.0	56.0 ☐
much		56.0	85.0	☐ little		85.0	178.0 ☐
much		178.0	219.0	☐ little		219.0	322.0 ☐
much		322.0	353.0	☐ little		353.0	410.0 ☐
much		410.0	429.0	☐ little		429.0	500.0 ☐

Area: **Tunnel**

Length: **20,000.0** Parameter Set: **0**

parameter state	beginpos.	endpos.	partly visible		
parameter: Faulting					
slight	335.0	392.0	☐	none	0.0 335.0 ☐
none	447.0	1,500.0	☒	severe	392.0 447.0 ☐
slight	4,111.0	4,140.0	☒	none	3,000.0 4,111.0 ☒
none	14,535.0	20,000.0	☐	slight	14,459.0 14,535.0 ☒
parameter: Overburden					
high	1,003.0	1,500.0	☒	low	0.0 1,003.0 ☐
high	3,697.0	4,140.0	☒	low	3,000.0 3,697.0 ☒
high	16,509.0	18,103.0	☐	low	4,140.0 16,509.0 ☒
				low	18,103.0 20,000.0 ☐

order	zone name	status	length	end pos.	param. set
0	Northern Sedimentary Rocks	Current	0.0	1,500.0	1

parameter state	beginpos.	endpos.	partly visible		
parameter: Overall Condition					
medium	50.0	192.0	☐	bad	0.0 50.0 ☐
medium	247.0	335.0	☐	bad	192.0 247.0 ☐
medium	417.0	539.0	☐	bad	335.0 417.0 ☐
medium	621.0	699.0	☐	good	539.0 621.0 ☐
medium	752.0	872.0	☐	bad	699.0 752.0 ☐
bad	938.0	1,003.0	☐	good	872.0 938.0 ☐
bad	1,080.0	1,114.0	☐	medium	1,003.0 1,080.0 ☐
medium	1,199.0	1,259.0	☐	good	1,114.0 1,199.0 ☐
medium	1,312.0	1,422.0	☐	bad	1,259.0 1,312.0 ☐
medium	1,481.0	1,500.0	☒	bad	1,422.0 1,481.0 ☐

order	zone name	status	length	end pos.	param. set
1	Metamorphic Rocks	Current	0.0	4,140.0	2

parameter state	beginpos.	endpos.	partly visible		
parameter: Overall Condition					
good	3,105.0	3,204.0	☐	medium	3,000.0 3,105.0 ☒
bad	3,344.0	3,374.0	☐	medium	3,204.0 3,344.0 ☐
medium	3,498.0	3,548.0	☐	good	3,374.0 3,498.0 ☐
medium	3,589.0	3,697.0	☐	bad	3,548.0 3,589.0 ☐
medium	3,854.0	3,966.0	☐	good	3,697.0 3,854.0 ☐
medium	4,015.0	4,128.0	☐	bad	3,966.0 4,015.0 ☐
				bad	4,128.0 4,140.0 ☒

order	zone name	status	length	end pos.	param. set
2	Intrusion	Current	0.0	17,223.0	3

parameter state	beginpos.	endpos.	partly visible		
parameter: Overall Condition				medium	17,223.0 17,122.0 ☐
bad	17,122.0	17,090.0	☐	medium	17,090.0 16,995.0 ☐
good	16,995.0	16,847.0	☐	medium	16,847.0 16,750.0 ☐
good	16,750.0	16,605.0	☐	bad	16,605.0 16,565.0 ☐
medium	16,565.0	16,429.0	☐	good	16,429.0 16,290.0 ☐
medium	16,290.0	16,153.0	☐	good	16,153.0 16,048.0 ☐
medium	16,048.0	15,940.0	☐	good	15,940.0 15,859.0 ☐
medium	15,859.0	15,783.0	☐	good	15,783.0 15,658.0 ☐
medium	15,658.0	15,452.0	☐	bad	15,452.0 15,415.0 ☐
medium	15,415.0	15,305.0	☐	good	15,305.0 15,119.0 ☐
medium	15,119.0	14,963.0	☐	good	14,963.0 14,821.0 ☐
bad	14,821.0	14,786.0	☐	good	14,786.0 14,720.0 ☐
medium	14,720.0	14,535.0	☐	good	14,535.0 14,459.0 ☒

order	zone name	status	length	end pos.	param. set
3	Southern Sedimentary Rocks	Observed	0.0	20,000.0	1

parameter state	beginpos.	endpos.	partly visible		
parameter: Overall Condition				bad	20,000.0 19,957.0 ☐
medium	19,957.0	19,895.0	☐	bad	19,895.0 19,840.0 ☐
medium	19,840.0	19,797.0	☐	good	19,797.0 19,740.0 ☐
medium	19,740.0	19,653.0	☐	bad	19,653.0 19,567.0 ☐
medium	19,567.0	19,490.0	☐	bad	19,490.0 19,446.0 ☐
good	19,446.0	19,407.0	☐	bad	19,407.0 19,331.0 ☐
medium	19,331.0	19,228.0	☐	bad	19,228.0 19,200.0 ☐
good	19,200.0	19,103.0	☐	medium	19,103.0 19,054.0 ☐
bad	19,054.0	18,991.0	☐	medium	18,991.0 18,871.0 ☐
good	18,871.0	18,789.0	☐	medium	18,789.0 18,755.0 ☐
bad	18,755.0	18,688.0	☐	medium	18,688.0 18,602.0 ☐
good	18,602.0	18,514.0	☐	medium	18,514.0 18,338.0 ☐
bad	18,338.0	18,287.0	☐	medium	18,287.0 18,248.0 ☐
good	18,248.0	18,192.0	☐	medium	18,192.0 18,143.0 ☐
good	18,143.0	18,065.0	☐	bad	18,065.0 17,977.0 ☐
medium	17,977.0	17,899.0	☐	bad	17,899.0 17,842.0 ☐
good	17,842.0	17,764.0	☐	medium	17,764.0 17,603.0 ☐
bad	17,603.0	17,561.0	☐	medium	17,561.0 17,526.0 ☐
bad	17,526.0	17,471.0	☐	good	17,471.0 17,372.0 ☐
medium	17,372.0	17,254.0	☐	bad	17,254.0 17,223.0 ☐

Observed Tunnel Delays and Sequence of Applied Methods

tunnel name	begin pos.	end pos.	opposite tunnel
Shaft - South Exit (Main)	3,000.00	20,000.00	South Exit - Shaft (Main)

Tunnel Delays

start time	start position	duration	going on
321.00	3,000.00	57.0	☐

Method Sequence

method	beginpos.	endpos.			
TBM w/ Concrete Lining mediu	3,000.0	3,105.0	TBM w/ Concrete Lining fast	3,105.0	3,206.0
TBM w/ Concrete Lining mediu	3,206.0	3,260.0			

tunnel name	begin pos.	end pos.	opposite tunnel
North Exit - Shaft	0.00	3,000.00	

Tunnel Delays

start time	start position	duration	going on				
0.00	0.00	175.0	☐	344.00	1,500.00	66.0	☒

Method Sequence

method	beginpos.	endpos.			
TBM w/ Concrete Lining slow	0.0	50.0	TBM w/ Concrete Lining mediu	50.0	192.0
TBM w/ Concrete Lining slow	192.0	247.0	TBM w/ Concrete Lining mediu	247.0	335.0
TBM w/ Concrete Lining slow	335.0	447.0	TBM w/ Concrete Lining mediu	447.0	539.0
TBM w/ Concrete Lining fast	539.0	621.0	TBM w/ Concrete Lining mediu	621.0	699.0
TBM w/ Concrete Lining slow	699.0	752.0	TBM w/ Concrete Lining mediu	752.0	872.0
TBM w/ Concrete Lining fast	872.0	938.0	TBM w/ Concrete Lining slow	938.0	1,003.0
TBM w/ Concrete Lining mediu	1,003.0	1,080.0	TBM w/ Concrete Lining slow	1,080.0	1,114.0
TBM w/ Concrete Lining fast	1,114.0	1,199.0	TBM w/ Concrete Lining mediu	1,199.0	1,259.0
TBM w/ Concrete Lining slow	1,259.0	1,312.0	TBM w/ Concrete Lining mediu	1,312.0	1,422.0
TBM w/ Concrete Lining slow	1,422.0	1,481.0	TBM w/ Concrete Lining mediu	1,481.0	1,500.0

tunnel name	begin pos.	end pos.	opposite tunnel
Shaft - South Exit (Aux)	3,000.00	20,000.00	South Exit - Shaft (Aux)

Method Sequence

method	beginpos.	endpos.			
Small TBM w/ Bolts & Shotcret	3,000.0	3,105.0	Small TBM w/ Shotcrete only	3,105.0	3,204.0
Small TBM w/ Bolts & Shotcret	3,204.0	3,344.0	Small TBM w/ Steel Arches & S	3,344.0	3,374.0
Small TBM w/ Shotcrete only	3,374.0	3,498.0	Small TBM w/ Bolts & Shotcret	3,498.0	3,548.0
Small TBM w/ Steel Arches & S	3,548.0	3,589.0	Small TBM w/ Bolts & Shotcret	3,589.0	3,697.0
Small TBM w/ Shotcrete only	3,697.0	3,854.0	Small TBM w/ Bolts & Shotcret	3,854.0	3,966.0
Small TBM w/ Steel Arches & S	3,966.0	4,015.0	Small TBM w/ Bolts & Shotcret	4,015.0	4,111.0
Small TBM w/ Steel Arches & S	4,111.0	4,140.0			

tunnel name	begin pos.	end pos.	opposite tunnel
South Exit - Shaft (Main)	20,000.00	3,000.00	Shaft - South Exit (Main)

Tunnel Delays

start time	start position	duration	going on				
0.00	20,000.00	65.0	☐	299.00	17,373.00	18.0	☐

Method Sequence

method	beginpos.	endpos.			
TBM w/ Concrete Lining slow	20,000.0	19,957.0	TBM w/ Concrete Lining mediu	19,957.0	19,895.0
TBM w/ Concrete Lining slow	19,895.0	19,840.0	TBM w/ Concrete Lining mediu	19,840.0	19,797.0
TBM w/ Concrete Lining fast	19,797.0	19,737.0	TBM w/ Concrete Lining mediu	19,737.0	19,656.0
TBM w/ Concrete Lining slow	19,656.0	19,565.0	TBM w/ Concrete Lining mediu	19,565.0	19,493.0
TBM w/ Concrete Lining slow	19,493.0	19,448.0	TBM w/ Concrete Lining fast	19,448.0	19,411.0
TBM w/ Concrete Lining slow	19,411.0	19,333.0	TBM w/ Concrete Lining mediu	19,333.0	19,224.0
TBM w/ Concrete Lining slow	19,224.0	19,200.0	TBM w/ Concrete Lining fast	19,200.0	19,103.0
TBM w/ Concrete Lining mediu	19,103.0	19,051.0	TBM w/ Concrete Lining slow	19,051.0	18,991.0
TBM w/ Concrete Lining mediu	18,991.0	18,864.0	TBM w/ Concrete Lining fast	18,864.0	18,775.0
TBM w/ Concrete Lining mediu	18,775.0	18,744.0	TBM w/ Concrete Lining slow	18,744.0	18,687.0
TBM w/ Concrete Lining mediu	18,687.0	18,605.0	TBM w/ Concrete Lining fast	18,605.0	18,506.0
TBM w/ Concrete Lining mediu	18,506.0	18,338.0	TBM w/ Concrete Lining slow	18,338.0	18,288.0
TBM w/ Concrete Lining mediu	18,288.0	18,245.0	TBM w/ Concrete Lining fast	18,245.0	18,186.0
TBM w/ Concrete Lining mediu	18,186.0	18,145.0	TBM w/ Concrete Lining fast	18,145.0	18,057.0
TBM w/ Concrete Lining slow	18,057.0	17,975.0	TBM w/ Concrete Lining mediu	17,975.0	17,899.0
TBM w/ Concrete Lining slow	17,899.0	17,842.0	TBM w/ Concrete Lining fast	17,842.0	17,770.0
TBM w/ Concrete Lining mediu	17,770.0	17,603.0			

tunnel name	begin pos.	end pos.	opposite tunnel
South Exit - Shaft (Aux)	20,000.00	3,000.00	Shaft - South Exit (Aux)

Method Sequence

method	beginpos.	endpos.			
Small TBM w/ Steel Arches & S	20,000.0	19,957.0	Small TBM w/ Bolts & Shotcret	19,957.0	19,895.0
Small TBM w/ Steel Arches & S	19,895.0	19,797.0	Small TBM w/ Shotcrete only	19,797.0	19,740.0
Small TBM w/ Bolts & Shotcret	19,740.0	19,653.0	Small TBM w/ Steel Arches & S	19,653.0	19,567.0
Small TBM w/ Bolts & Shotcret	19,567.0	19,490.0	Small TBM w/ Steel Arches & S	19,490.0	19,331.0
Small TBM w/ Bolts & Shotcret	19,331.0	19,228.0	Small TBM w/ Steel Arches & S	19,228.0	19,200.0
Small TBM w/ Shotcrete only	19,200.0	19,103.0	Small TBM w/ Bolts & Shotcret	19,103.0	19,054.0
Small TBM w/ Steel Arches & S	19,054.0	18,991.0	Small TBM w/ Bolts & Shotcret	18,991.0	18,871.0
Small TBM w/ Shotcrete only	18,871.0	18,755.0	Small TBM w/ Steel Arches & S	18,755.0	18,688.0
Small TBM w/ Bolts & Shotcret	18,688.0	18,602.0	Small TBM w/ Shotcrete only	18,602.0	18,514.0
Small TBM w/ Bolts & Shotcret	18,514.0	18,338.0	Small TBM w/ Steel Arches & S	18,338.0	18,287.0
Small TBM w/ Bolts & Shotcret	18,287.0	18,143.0	Small TBM w/ Shotcrete only	18,143.0	18,065.0
Small TBM w/ Steel Arches & S	18,065.0	17,977.0	Small TBM w/ Bolts & Shotcret	17,977.0	17,899.0
Small TBM w/ Steel Arches & S	17,899.0	17,842.0	Small TBM w/ Shotcrete only	17,842.0	17,764.0
Small TBM w/ Bolts & Shotcret	17,764.0	17,603.0	Small TBM w/ Steel Arches & S	17,603.0	17,471.0
Small TBM w/ Shotcrete only	17,471.0	17,372.0	Small TBM w/ Bolts & Shotcret	17,372.0	17,254.0
Small TBM w/ Steel Arches & S	17,254.0	17,223.0	Small TBM w/ Bolts & Shotcret	17,223.0	17,122.0
Small TBM w/ Steel Arches & S	17,122.0	17,090.0	Small TBM w/ Bolts & Shotcret	17,090.0	16,995.0
Small TBM w/ Shotcrete only	16,995.0	16,847.0	Small TBM w/ Bolts & Shotcret	16,847.0	16,750.0
Small TBM w/ Shotcrete only	16,750.0	16,605.0	Small TBM w/ Steel Arches & S	16,605.0	16,565.0
Small TBM w/ Bolts & Shotcret	16,565.0	16,429.0	Small TBM w/ Shotcrete only	16,429.0	16,290.0
Small TBM w/ Bolts & Shotcret	16,290.0	16,153.0	Small TBM w/ Shotcrete only	16,153.0	16,048.0
Small TBM w/ Bolts & Shotcret	16,048.0	15,940.0	Small TBM w/ Shotcrete only	15,940.0	15,859.0
Small TBM w/ Bolts & Shotcret	15,859.0	15,783.0	Small TBM w/ Shotcrete only	15,783.0	15,658.0
Small TBM w/ Bolts & Shotcret	15,658.0	15,452.0	Small TBM w/ Steel Arches & S	15,452.0	15,415.0
Small TBM w/ Bolts & Shotcret	15,415.0	15,305.0	Small TBM w/ Shotcrete only	15,305.0	15,119.0
Small TBM w/ Bolts & Shotcret	15,119.0	14,963.0	Small TBM w/ Shotcrete only	14,963.0	14,821.0
Small TBM w/ Steel Arches & S	14,821.0	14,786.0	Small TBM w/ Shotcrete only	14,786.0	14,720.0
Small TBM w/ Bolts & Shotcret	14,720.0	14,535.0	Small TBM w/ Shotcrete only	14,535.0	14,459.0

tunnel name		begin pos.	end pos.	opposite tunnel			
Access Shaft		0.00	500.00				
Tunnel Delays							
start time	start position	duration	going on				
80.00	227.00	25.0	☐	184.00	500.00	136.0	☐
Method Sequence							
method	beginpos.	endpos.					
Shaft D&B w/ Shotcrete 6cm	0.0	28.0	Shaft D&B w/ Shotcrete 15cm	28.0	43.0		
Shaft D&B w/ Shotcrete 6cm	43.0	56.0	Shaft D&B w/ Shotcrete & Bolts	56.0	85.0		
Shaft D&B w/ Shotcrete 6cm	85.0	178.0	Shaft D&B w/ Shotcrete 15cm	178.0	188.0		
Shaft D&B w/ Shotcrete & Bolts	188.0	219.0	Shaft D&B w/ Shotcrete 6cm	219.0	322.0		
Shaft D&B w/ Shotcrete & Bolts	322.0	353.0	Shaft D&B w/ Shotcrete 6cm	353.0	410.0		
Shaft D&B w/ Shotcrete & Bolts	410.0	429.0	Shaft D&B w/ Shotcrete 6cm	429.0	500.0		

A.3 Updating

Update Definition of Zones

Area: **Shaft** Length: **500.0**

		prob min	min	length mode	max	prob max	prob min	min	endpos mode	max	prob max
0	Shaft Zone										
Prior	End Pos	0.00	0.0	0.0	0.0	0.00	0.00	500.0	500.0	500.0	0.00
Observed	End Pos	0.00	0.0	0.0	0.0	0.00	0.00	500.0	500.0	500.0	0.00

Area: **Tunnel** Length: **20,000.0**

		prob min	min	length mode	max	prob max	prob min	min	endpos mode	max	prob max
0	Northern Sedimentary Rocks										
Prior	End Pos	0.00	0.0	0.0	0.0	0.00	0.10	2,200.0	2,600.0	2,900.0	0.10
Current	End Pos	0.00	1,500.0	2,600.0	2,900.0	0.00	0.00	2,200.0	2,600.0	2,900.0	0.00
1	Metamorphic Rocks										
Prior	Length	0.00	7,500.0	-1.0	9,000.0	0.00	0.00	0.0	0.0	0.0	0.00
Current	End Pos	0.00	1,240.0	-1.0	9,000.0	0.00	0.00	9,700.0	-1.0	11,900.0	0.00
2	Intrusion										
Prior	Length	0.00	5,000.0	6,000.0	7,000.0	0.00	0.00	16,000.0	17,000.0	17,500.0	0.00
Current	End Pos	0.00	0.0	0.0	0.0	0.00	0.00	17,223.0	17,223.0	17,223.0	0.00
3	Southern Sedimentary Rocks										
Prior	End Pos	0.00	0.0	0.0	0.0	0.00	0.00	20,000.0	-1.0	20,000.0	0.00
Observed	End Pos	0.00	0.0	0.0	0.0	0.00	0.00	20,000.0	20,000.0	20,000.0	0.00

Update Deterministic Sequence of Parameter States

Parameter Set: 0

Parameter: Overburden

Updated: ☐

Prior:		Observed/Posterior:		
parameter state	endpos	status	parameter state	endpos
low	1,000.0	Observed	low	1,003.0
high	2,350.0	Current	high	2,350.0
low	3,700.0	Current	low	3,697.0
high	12,800.0	Current	high	12,800.0
low	16,500.0	Current	low	16,509.0
high	18,100.0	Observed	high	18,103.0
low	20,000.0	Observed	low	20,000.0

Update Semi-Deterministic Sequence of Parameter States

Parameter Set: 0

Parameter: Faulting

Updated: ☐

			prob min	min	length mode	max	prob max	prob min	min	endpos mode	max	prob max
1	Prior	End Pos	0.00	0.0	0.0	0.0	0.00	0.00	300.0	400.0	500.0	0.00
	Observed	End Pos	0.00	335.0	335.0	335.0	0.00	0.00	335.0	335.0	335.0	0.00
2	Prior	Length	0.00	50.0	100.0	150.0	0.00	0.00	0.0	0.0	0.0	0.00
	Observed	End Pos	0.00	57.0	57.0	57.0	0.00	0.00	392.0	392.0	392.0	0.00
3	Prior	Length	0.00	20.0	30.0	50.0	0.00	0.00	0.0	0.0	0.0	0.00
	Observed	End Pos	0.00	55.0	55.0	55.0	0.00	0.00	447.0	447.0	447.0	0.00
4	Prior	Length	0.00	20.0	50.0	80.0	0.00	0.00	0.0	0.0	0.0	0.00
	Observed	Length	0.00	0.0	0.0	0.0	0.00	0.00	0.0	0.0	0.0	0.00
5	Prior	End Pos	0.00	0.0	0.0	0.0	0.00	0.00	4,000.0	4,400.0	4,700.0	0.00
	Current	End Pos	0.00	0.0	0.0	0.0	0.00	0.00	4,111.0	4,111.0	4,111.0	0.00
6	Prior	Length	0.00	100.0	150.0	200.0	0.00	0.00	0.0	0.0	0.0	0.00
	Current	Length	0.00	100.0	150.0	200.0	0.00	0.00	4,140.0	4,261.0	4,311.0	0.00
7	Prior	Length	0.00	50.0	80.0	100.0	0.00	0.00	0.0	0.0	0.0	0.00
	Posterior	Length	0.00	50.0	80.0	100.0	0.00	0.00	4,261.0	4,341.0	4,411.0	0.00
8	Prior	End Pos	0.00	0.0	0.0	0.0	0.00	0.00	13,200.0	13,400.0	14,000.0	0.00
	Posterior	End Pos	0.00	8,789.0	9,059.0	9,739.0	0.00	0.00	13,200.0	13,400.0	14,000.0	0.00
9	Prior	Length	0.00	100.0	150.0	200.0	0.00	0.00	0.0	0.0	0.0	0.00
	Posterior	Length	0.00	100.0	150.0	200.0	0.00	0.00	13,300.0	13,550.0	14,200.0	0.00
10	Prior	Length	0.00	20.0	80.0	150.0	0.00	0.00	0.0	0.0	0.0	0.00
	Posterior	Length	0.00	20.0	80.0	150.0	0.00	0.00	13,320.0	13,630.0	14,350.0	0.00
11	Prior	Length	0.00	50.0	150.0	300.0	0.00	0.00	0.0	0.0	0.0	0.00
	Current	End Pos	0.00	0.0	0.0	0.0	0.00	0.00	14,535.0	14,535.0	14,535.0	0.00
12	Prior	End Pos	0.00	0.0	0.0	0.0	0.00	0.00	20,000.0	-1.0	20,000.0	0.00
	Observed	End Pos	0.00	5,465.0	5,465.0	5,465.0	0.00	0.00	20,000.0	20,000.0	20,000.0	0.00

Update Markov Process: Mean Lengths

Parameter Set: 1

Parameter: Overall Condition

Updated: ☐

Parameter State:

good

	Prior	Posterior
min	60.0	58.4
mode	90.0	90.0
max	120.0	114.0
start prob	0.00	0.00

Parameter State:

medium

	Prior	Posterior
min	50.0	58.8
mode	120.0	96.4
max	190.0	149.0
start prob	0.00	1.00

Parameter State:

bad

	Prior	Posterior
min	30.0	36.8
mode	60.0	60.0
max	90.0	81.5
start prob	1.00	0.00

Parameter Set: 2

Parameter: Overall Condition

Updated: ☐

Parameter State:

good

	Prior	Posterior
min	80.0	63.9
mode	-1.0	127.0
max	160.0	173.0
start prob	0.33	0.29

Parameter State:

medium

	Prior	Posterior
min	80.0	71.6
mode	-1.0	105.0
max	160.0	178.0
start prob	0.34	0.54

Parameter State:

bad

	Prior	Posterior
min	30.0	26.8
mode	-1.0	40.0
max	80.0	91.3
start prob	0.33	0.17

Parameter Set: 3

Parameter: Overall Condition

Updated: ☐

Parameter State:

good

	Prior	Posterior
min	80.0	84.9
mode	120.0	120.0
max	160.0	160.0
start prob	0.33	0.45

Parameter State:

medium

	Prior	Posterior
min	40.0	59.4
mode	60.0	95.0
max	120.0	123.0
start prob	0.34	0.42

Parameter State:

bad

	Prior	Posterior
min	30.0	24.8
mode	70.0	66.3
max	100.0	93.5
start prob	0.33	0.13

Update Transition Probabilities

Parameter Set: 1 Parameter: Overall Condition Updated: ☐

from parameter state	# of transitions	to parameter state	prior	observed	posterior
bad	21	good	0.30	0.14	0.15
		medium	0.70	0.86	0.85
good	12	bad	0.50	0.42	0.45
		medium	0.50	0.58	0.55
medium	24	bad	0.50	0.63	0.60
		good	0.50	0.38	0.40

Parameter Set: 2 Parameter: Overall Condition Updated: ☐

from parameter state	# of transitions	to parameter state	prior	observed	posterior
bad	3	good	0.20	0.33	0.25
		medium	0.80	0.67	0.75
good	3	bad	0.20	0.00	0.15
		medium	0.80	1.00	0.85
medium	6	bad	0.65	0.67	0.65
		good	0.35	0.33	0.35

Parameter Set: 3 Parameter: Overall Condition Updated: ☐

from parameter state	# of transitions	to parameter state	prior	observed	posterior
bad	4	good	0.33	0.50	0.35
		medium	0.67	0.50	0.65
good	10	bad	0.33	0.10	0.15
		medium	0.67	0.90	0.85
medium	10	bad	0.50	0.30	0.35
		good	0.50	0.70	0.65

Update Cycle Lengths

Method: Small TBM w/ Bolts & Shotcrete

Heading: 0

Updated: ☐

Prior
prob. min. 0.00
min 12.0
mode 15.0
max 17.0
prob. max 0.00

Posterior
prob. min. 0.00
min 14.5
mode 15.2
max 15.7
prob. max 0.00

mean values: prior: 14.67 posterior 15.13 observed: 15.17

discard all data observed before timelimit 20

posterior input represents: long-term averages

Method: Small TBM w/ Shotcrete only

Heading: 0

Updated: ☐

mean values: prior: 23.67 posterior: 21.00 observed: 21.07

discard all data observed before timelimit 50

posterior input represents: long-term averages

Method: Small TBM w/ Steel Arches & Shotcrete

Heading: 0

Updated: ☐

mean values: prior: 6.80 posterior: 7.75 observed: 7.82

discard all data observed before timelimit 10

posterior input represents: long-term averages

Method: TBM w/ Concrete Lining fast

Heading: 0

Updated: ☐

mean values: prior: 22.00 posterior: 20.50 observed: 20.42

discard all data observed before timelimit 100

posterior input represents: long-term averages

Method: TBM w/ Concrete Lining medium

Heading: 0

Updated: ☐

mean values: prior: 11.00 posterior 10.93 observed: 10.90

discard all data observed before timelimit 0

posterior input represents: long-term averages

Method: TBM w/ Concrete Lining slow

Heading: 0

Updated: ☐

mean values: prior: 5.17 posterior: 6.10 observed: 6.10

discard all data observed before timelimit 0

posterior input represents: long-term averages

Update Method Variables

Method: **Small TBM w/ Bolts & Shotcrete**

Variable: **cost_per_meter**

Updated: ☐

Prior
prob. min. 0.00
min 6,000.0
mode 7,000.0
max 8,000.0
prob. max 0.00

Posterior
prob. min. 0.00
min 6,800.0
mode 7,100.0
max 7,400.0
prob. max 0.00

mean values: prior: **7,000.00** posterior **7,100.00** observed: **7,115.90**

discard all data observed before timelimit: 0

posterior input represents: **long-term averages**

Method: Small TBM w/ Shotcrete only

Variable: cost_per_meter

Updated: ☐

Prior
prob. min. 0.00
min 2,900.0
mode 3,200.0
max 3,400.0
prob. max 0.00

Posterior
prob. min. 0.00
min 3,100.0
mode 3,200.0
max 3,300.0
prob. max 0.00

mean values: prior: 3,166.67 posterior 3,200.00 observed: 3,206.38

discard all data observed before timelimit: 0

posterior input represents: long-term averages

Method: Small TBM w/ Steel Arches & Shotcr Variable: cost_per_meter

Updated: ☐

Prior
prob. min. 0.00
min 9,000.0
mode 10,500.0
max 11,500.0
prob. max 0.00

Posterior
prob. min. 0.00
min 10,500.0
mode 11,000.0
max 11,450.0
prob. max 0.00

mean values: prior: 10,333.33 posterior 10,983.33 observed: 10,984.22

discard all data observed before timelimit: 0

posterior input represents: long-term averages

Method: TBM w/ Concrete Lining fast

Variable: cost_per_meter

Updated: ☐

Prior
prob. min. 0.00
min 9,500.0
mode 10,500.0
max 12,000.0
prob. max 0.00

Posterior
prob. min. 0.00
min 11,000.0
mode 11,400.0
max 11,900.0
prob. max 0.00

mean values: prior: 10,666.67 posterior 11,433.33 observed: 11,418.49

discard all data observed before timelimit: 100

posterior input represents: long-term averages

Method: TBM w/ Concrete Lining medium

Variable: cost_per_meter

Updated: ☐

Prior
prob. min. 0.00
min 10,500.0
mode 12,000.0
max 14,000.0
prob. max 0.00

Posterior
prob. min. 0.00
min 12,500.0
mode 13,050.0
max 13,650.0
prob. max 0.00

mean values: prior: 12,166.67 posterior 13,066.67 observed: 13,043.45

discard all data observed before timelimit: 0

posterior input represents: long-term averages

Method: TBM w/ Concrete Lining slow

Variable: cost_per_meter

Updated: ☐

Prior
prob. min. 0.00
min 15,500.0
mode 17,500.0
max 19,000.0
prob. max 0.00

Posterior
prob. min. 0.00
min 16,300.0
mode 17,100.0
max 17,900.0
prob. max 0.00

mean values: prior: 17,333.33 posterior 17,100.00 observed: 17,091.10

discard all data observed before timelimit: 0

posterior input represents: long-term averages

Update Associations of Geometries, Ground Classes and Methods

Geometry: 1

ground class	length observed	method applied	prior	observed	posterior
bad conditions	1299	TBM w/ Concrete Lining f	0.00	0.01	0.00
		TBM w/ Concrete Lining	0.00	0.02	0.00
		TBM w/ Concrete Lining	1.00	0.97	1.00
good conditions	1229	TBM w/ Concrete Lining f	1.00	0.98	1.00
		TBM w/ Concrete Lining	0.00	0.02	0.00
		TBM w/ Concrete Lining	0.00	0.00	0.00
medium conditions	2557	TBM w/ Concrete Lining f	0.00	0.02	0.00
		TBM w/ Concrete Lining	1.00	0.98	1.00
		TBM w/ Concrete Lining	0.00	0.01	0.00

Geometry: 2

ground class	length observed	method applied	prior	observed	posterior
bad conditions	1079	Small TBM w/ Steel Arch	1.00	1.00	1.00
good conditions	2267	Small TBM w/ Bolts & Sh	0.00	0.02	0.00
		Small TBM w/ Shotcrete	1.00	0.96	1.00
		Small TBM w/ Steel Arch	0.00	0.02	0.00
medium conditions	3335	Small TBM w/ Bolts & Sh	1.00	0.97	1.00
		Small TBM w/ Shotcrete	0.00	0.01	0.00
		Small TBM w/ Steel Arch	0.00	0.02	0.00

Geometry: 3

ground class	length observed	method applied	prior	observed	posterior
bad shaft conditions	110	Shaft D&B w/ Shotcrete	1.00	1.00	1.00
good shaft conditions	352	Shaft D&B w/ Shotcrete	1.00	1.00	1.00
medium shaft conditio	38	Shaft D&B w/ Shotcrete	1.00	0.66	1.00
		Shaft D&B w/ Shotcrete	0.00	0.34	0.00

Update Tunnel Delays

Tunnel: Access Shaft

Construction Start Time 0

Updated: ☐

position-dependent delays

time-dependent delays

	type of delay	start	duration min	mode	max
Prior	position-dependent	200	20	30	40
Observed	position-dependent	227	25	25	25
Prior	position-dependent	500	130	150	170
Observed	position-dependent	500	136	136	136

Tunnel: North Exit - Shaft

Construction Start Time 0

Updated: ☐

position-dependent delays

time-dependent delays

	type of delay	start	duration min	mode	max
Prior	time-dependent	0	150	180	200
Observed	time-dependent	0	175	175	175
Prior	position-dependent	1,500	120	150	180
Current	position-dependent	1,500	120	150	180

Tunnel: Shaft - South Exit (Aux)

Construction Start Time 320

Updated: ☐

position-dependent delays

time-dependent delays

	type of delay	start	duration min	mode	max
Prior	time-dependent	500	10	20	30
Posterior	time-dependent	500	10	20	30

Tunnel: Shaft - South Exit (Main)

Construction Start Time 320

Updated: ☐

position-dependent delays

time-dependent delays

	type of delay	start	duration min	mode	max
Prior	position-dependent	0	40	50	60
Observed	position-dependent	0	57	57	57

Tunnel: South Exit - Shaft (Aux)

Construction Start Time 0

Updated: ☐

position-dependent delays

time-dependent delays

	type of delay	start	duration min	mode	max
Prior	position-dependent	7,000	20	30	40
Posterior	position-dependent	7,000	20	30	40

Tunnel: South Exit - Shaft (Main)

Construction Start Time

0

Updated: ☐

position-dependent delays

time-dependent delays

	type of delay	start	duration min	mode	max
Prior	time-dependent	0	50	70	100
Observed	time-dependent	0	65	65	65
Prior	time-dependent	299	0	0	0
Observed	time-dependent	299	18	18	18

Update Method Change Delays

from method: to method: prior posterior

Shaft D&B w/ Shotcrete 15cm Shaft D&B w/ Shotcrete & Bolts 0.02 0.02

Shaft D&B w/ Shotcrete 6cm Shaft D&B w/ Shotcrete & Bolts 0.04 0.04

Shaft D&B w/ Shotcrete 6cm Shaft D&B w/ Shotcrete 15cm 0.02 0.02

from method:

to method:

prior posterior

Small TBM w/ Bolts & Shotcrete

Small TBM w/ Steel Arches & Shotcrete

0.08

0.08

Small TBM w/ Shotcrete only

Small TBM w/ Bolts & Shotcrete

0.04

0.04

Small TBM w/ Shotcrete only

Small TBM w/ Steel Arches & Shotcrete

0.08

0.07

from method:	to method:	prior	posterior
Small TBM w/ Steel Arches & Shotcrete	Small TBM w/ Bolts & Shotcrete	0.06	0.06

Small TBM w/ Steel Arches & Shotcrete	Small TBM w/ Shotcrete only	0.04	0.04
---------------------------------------	-----------------------------	------	------

Calculation of Expected Construction Time and Cost

(Simplified Fully Updated)

Calculation of average cycle lengths:

	min	mode	max	average
Shaft D&B w/ Shotcrete & Bolts	1.90	2.00	2.10	2.00
Shaft D&B w/ Shotcrete 15cm	2.80	3.00	3.20	3.00
Shaft D&B w/ Shotcrete 6cm	4.20	4.70	5.20	4.69
Small TBM w/ Shotcrete only	20.00	21.00	22.00	20.99
Small TBM w/ Bolts & Shotcrete	14.50	15.20	15.70	15.13
Small TBM w/ Steel Arches & Shotcrete	7.30	7.80	8.30	7.79
TBM w/ Concrete Lining fast	19.5	20.5	21.5	20.49
TBM w/ Concrete Lining medium	10.30	11.00	11.50	10.93
TBM w/ Concrete Lining slow	5.70	6.10	6.50	6.10

A) Shaft Zone

	E(length)	Transition Probabilities			mean length:	0
		Extent Share				Avg. Extent
<i>Jointing</i>		intense	interm.	sparse		
intense	20		0.80	0.20	12%	0
intermediate	40	0.50		0.50	42%	0
sparse	72	0.10	0.90		46%	0
<i>Water</i>		little	much			
little	50		1.00		68%	0
much	23	1.00			32%	0

Extent share of parameter combinations:

Methods used for parameter combinations:

Jointing\Water	little	much	intense	little	much
intense	8%	4%	intense	shotcrete 15cm	shotcrete & bolts
intermediate	29%	13%	intermediate	shotcrete 6cm	shotcrete & bolts
sparse	31%	15%	sparse	shotcrete 6cm	shotcrete 15cm

Method	Avg. Extent	Avg. Cycle Length	Avg. Cost per meter	Time	Cost
Shaft D&B w/ Shotcrete & Bolts	0	2.0	25167	0	0
Shaft D&B w/ Shotcrete 15cm	0	3.0	17000	0	0
Shaft D&B w/ Shotcrete 6cm	0	4.7	13167	0	0
				0	0

Tunnel Delays / Fix Costs	320	18,598,000
Total Time and Cost	320	18,598,000

B) Northern Sedimentary Rocks

	E(length)	Transition Probabilities			mean length:	2567
		Extent Share				Known Extent
<i>Overall Condition</i>		good	medium	bad		
good	87		0.55	0.45	23.3%	
medium	101	0.40		0.60	51.5%	
bad	59	0.15	0.85		25.2%	

<i>Main Tunnel</i>	<i>remaining length</i>	<i>1067</i>		<i>unknown length</i>	<i>1067</i>
			Extent m	Extent Share	
area ground classes:	fault serious		0	0.0%	
	fault important		0	0.0%	
	fault neutral		1067	100.0%	
Method	Avg. Extent	Avg. Cycle Length	Avg. Cost per meter	Time	Cost
TBM w/ Concrete Lining fast	249	20.5	11433	12	2,844,202
TBM w/ Concrete Lining mediu	549	10.9	13067	50	7,171,345
TBM w/ Concrete Lining slow	269	6.1	17100	44	4,601,186
	1067			107	14,616,733
Tunnel Delays / Fix Costs				494	24,066,667
Total Time and Cost				601	38,683,400

C) Southern Sedimentary Rocks

	E(length)	Transition Probabilities		mean length:	2777
		good	medium	bad	Known Extent
Overall Condition					
good	87		0.55	0.45	23.3%
medium	101	0.40		0.60	51.5%
bad	59	0.15	0.85		25.2%
<i>Main Tunnel</i>	<i>remaining length</i>	<i>0</i>		<i>unknown length</i>	<i>0</i>
			Extent m	Extent Share	
area ground classes:	fault serious		0	0.0%	
	fault important		0	0.0%	
	fault neutral		0	100.0%	
Method	Avg. Extent	Avg. Cycle Length	Avg. Cost per meter	Time	Cost
TBM w/ Concrete Lining fast	0	20.5	11433	0	0
TBM w/ Concrete Lining mediu	0	10.9	13067	0	0
TBM w/ Concrete Lining slow	0	6.1	17100	0	0
	0			0	0
Tunnel Delays / Fix Costs				358	41,358,069
Total Time and Cost				358	41,358,069

<i>Auxiliary Tunnel</i>	<i>remaining length</i>	<i>0</i>			
			Extent m	Extent Share	
area ground classes:	fault serious		0	0.0%	
	fault important		0	0.0%	
	fault neutral		0	100.0%	
Method	Avg. Extent	Avg. Cycle Length	Avg. Cost per meter	Time	Cost
Small TBM w/ Shotcrete only	0	21.0	3200	0	0
Small TBM w/ Bolts & Shotcrete	0	15.1	7100	0	0
Small TBM w/ Steel Arches & S	0	7.8	10983	0	0
	0			0	0
Tunnel Delays / Fix Costs				231	22,669,760
Total Time and Cost				231	22,669,760

D) Metamorphic Rocks

				mean length:	8233
E(length)	Transition Probabilities			Extent Share	Known Extent
<i>Overall Condition</i>		good	medium	bad	
good	121		0.85	0.15	29.2%
medium	118	0.35		0.65	53.4%
bad	53	0.25	0.75		17.4%
<i>Main Tunnel</i>	<i>remaining length</i>	7973		<i>unknown length</i>	7093
			Extent m	Extent Share	
area ground classes:		fault serious	77	1.1%	
		fault important	150	2.1%	
		fault neutral	6866	96.8%	
Method	Avg. Extent	Avg. Cycle Length	Avg. Cost per meter	Time	Cost
TBM w/ Concrete Lining fast	2285	20.5	11433	112	26,130,072
TBM w/ Concrete Lining medium	4179	10.9	13067	382	54,605,582
TBM w/ Concrete Lining slow	1509	6.1	17100	248	25,802,287
	7973			741	106,537,941
Tunnel Delays / Fix Costs				80	4,816,667
Total Time and Cost				821	111,354,608
<i>Auxiliary Tunnel</i>	<i>remaining length</i>	6660			
			Extent m	Extent Share	
area ground classes:		fault serious	77	1.2%	
		fault important	150	2.3%	
		fault neutral	6433	96.6%	
Method	Avg. Extent	Avg. Cycle Length	Avg. Cost per meter	Time	Cost
Small TBM w/ Shotcrete only	1878	21.0	3200	89	6,009,375
Small TBM w/ Bolts & Shotcrete	3480	15.1	7100	230	24,711,538
Small TBM w/ Steel Arches & Shotcrete	1302	7.8	10983	167	14,295,597
	6660			486	45,016,511
Tunnel Delays / Fix Costs				100	7,995,000
Total Time and Cost				586	53,011,511

E) Intrusion

E) Intrusion				mean length: 6423		
	E(length)	Transition Probabilities			Extent Share	Known Extent
Overall Condition		good	medium	bad		
good	122		0.85	0.15	44.9%	991
medium	92	0.65		0.35	41.9%	1114
bad	62	0.35	0.65		13.2%	112
Main Tunnel	remaining length	5875			unknown length	3658
area ground classes:				Extent m	Extent Share	
		fault serious		0	0.0%	
		fault important		83	2.3%	
		fault neutral		3575	97.7%	
Method		Avg. Extent	Avg. Cycle Length	Avg. Cost per meter	Time	Cost
TBM w/ Concrete Lining fast		2597	20.5	11433	127	29,696,875
TBM w/ Concrete Lining mediu		2649	10.9	13067	242	34,615,371
TBM w/ Concrete Lining slow		628	6.1	17100	103	10,746,839
		5875			472	75,059,085
Tunnel Delays / Fix Costs					42	6,708,598
Total Time and Cost					514	81,767,683
Auxiliary Tunnel	remaining length	3659				
area ground classes:				Extent m	Extent Share	
		fault serious		0	0.0%	
		fault important		83	2.3%	
		fault neutral		3576	97.7%	
Method		Avg. Extent	Avg. Cycle Length	Avg. Cost per meter	Time	Cost
Small TBM w/ Shotcrete only		1607	21.0	3200	77	5,141,900
Small TBM w/ Bolts & Shotcrete		1536	15.1	7100	101	10,902,437
Small TBM w/ Steel Arches & S		517	7.8	10983	66	5,674,010
		3659			244	21,718,347
Tunnel Delays / Fix Costs					199	15,046,907
Total Time and Cost					443	36,765,253
Summary				Length		Cost
Shaft				0		18,598,000
Main Tunnel				14,915		273,163,759
Auxiliary Tunnel				10,319		112,446,524
Total Construction Time and Costs					986.7	404,208,283

Appendix B: Database Structure and Code

B.1 Database Structure and Naming Conventions

The Input and Updating Module of the DAT are coded on the basis of Microsoft® Access®. MS Access is a relational database. Appendix B assumes that the reader is familiar with coding in MS Access. The datatables that are used in the DAT are shown below.

ImpExp1	tActivities	tMethods
ImpExp2	tActResDemand	tParameters
ImpExpDelay	tAreas	tParamSets
ImpExpMemo	tBH_X_States	tParamStates
Messages	tBoreholes	tParam_X_Set
otAreaParamStat	tCGC_X_Meth_X_Geom	tRepositories
otChangeDelays	tChangeDelays	tResources
otChangeDelaysGraph	tCombGC	tSegProb
otCycleGraph	tCorrelations	tSemiDeterSegs
otCycleLengths	tCorrel_X_States	tStocks
otGCCGeom	tDeterSegs	tTime
otGenVariables	tGC	tTunDelayProb
otGenVariablesGraph	tGCCCombinations	tTunDelays
otMarkovGraph	tGCCComb_X_States	tTunnelArcs
otMethods	tGenVariables	tTunnelNodes
otTransitions	tGenVarProb	tTunnels
otTunDelays	tGeometries	tTunnel_X_Rep
otTunDelaysGraph	tHeadProb	tUpdatePhases
otVariables	tHeads	tUpDownChanges
otVariablesGraph	tLearnParam	tVariableProb
otZoneParamStat	tMarkovLengths	tVariables
rtTimeCost	tMarkovTransProb	tZoneProb
rtTimeCostGraph	tMethArcs	tZones
rtTimePosGraph	tMethNodes	

The following subsection explains naming conventions used in the database. The relations between these datatables are explained in subsequent sub-sections.

Naming Conventions

Access has various object types, six of them are used by the DAT: tables, queries, forms, reports, macros and modules. Only few macros and modules are used in the DAT, so that there are no naming conventions for them.

Table names have the prefixes “t”, “ot”, “rt” or have no prefix at all. Their meaning is:

- “t”: tables in general, that are not part of any of the following categories
- “ot”: tables that hold data from observations
- “rt”: tables that hold data from simulator results
- none: auxiliary tables that do not hold data which is relevant to the project which is loaded

Queries have the following prefixes and associated meanings:

- “aq”: query that adds data to the table with the same name as the remainder of the query name
- “cq”: query that is only used by the input data checker or observations checker
- “dq”: query that deletes the content of the table with the same name as the remainder of the query name
- “dco”: like “dq”, but for tables with prefix “ot”
- “e”: query that is only used for data export
- “oq”: query that queries data from tables with prefix “ot”
- “orq”: query that queries data for reports on observations
- “p”: query that queries only data from initial (prior) input
- “pr”: query that queries data for reports about initial input
- “q”: query that queries data from any update phase
- “qr”: query that queries data for updating reports
- “rq”: query that queries data from tables with prefix “rt”
- none: queries using auxiliary tables only

Forms have the following prefixes and associated meanings:

- “f”: form related to the updating process
- “of”: form related to observations
- “pf”: form related to initial (prior) input

“rf”: form that shows simulator results (from tables with prefix “rt”)

Reports have the following prefixes and associated meanings:

“or”: reports about observations

“pr”: reports about initial (prior) input

“r”: reports about the updating process

Update Phases

“tUpdatePhases” holds always (even if there is no project) three records, ID = 1 (“Initial Input”), ID = 10 (“Observed”) and ID=11 (“reserve”). Initial input is considered to be the first update phase and it is always present. Because it takes this special position, it can never become “current” (it is always there anyway). “Observed” and “reserve” are not update phases in the meaning of the DAT, in fact “reserve” is not yet used at all. They are just there because there may be data that is conveniently stored like an update phase. “Observed” is sometimes used to store observed data. “tUpdatePhases” is related to every other table which hold updateable data. Most tables that hold updateable data must at all times have a record for each update phase (i.e. if there is a record for “initial input” there must be also records for all other update phases, except “observed” and “reserve”).

These tables are: “tCGC_X_Meth_X_Geom”, “tChangeDelays”, “tGenVarProb”, “tHeadProb”, “tMarkovLengths”, “tMarkovTransProb”, “tSegProb”, “tStocks”. “tTunDelayProb”, “tVarProb”, “tZoneProb”,

The only table which contains updateable data but does not follow the above rule is “tDeterSegs”. It is not necessary that each “prior” segment has a counterpart in each update phase.

The field ‘path’ in “tUpdatePhases” holds the path to the directory to which the simulator input files should be written. It is usually different for each update phase (since old files are overwritten) and therefore convenient for the user that it is remembered.

Areas and Zones

The tables “otZoneParamStat” and “otAreaParamStat” hold the observed sequence of parameter states in zones and areas, respectively. “tZones” also hold observed zones (with update phase = 10).

Tunnels, Tunnel Network, Tunnel Delays

“otTunDelaysGraph” holds all data points for tunnel delay graphs, whereas “otTunDelays” holds all observed tunnel delays. “otMethods” hold the sequence of applied methods for each tunnel. Updates of fixcosts, beginpos and endpos of tunnels are stored directly in “tTunnels”, i.e. only the values for the current update phase (if any) are stored. Note that ‘value1’ and ‘value2’ in “tGeometries” are dummies and not used in the simulator. ‘PosX’ and ‘PosY’ in “tTunnelNodes” are the coordinate of the nodes on the graph and have nothing to do with the tunnel layout in reality.

Parameter Sets, Parameters and Parameter States

“otTransitions” holds the *number* of observed transitions in a particular parameter set from a particular parameter state. “tMarkovTransProb” also hold the observed transition probabilities (update phase = 10). “otMarkovGraph” holds the data points for the graph in the Markov length updating window. ‘DeterReference’ in “tParam_X_Set” defines whether the position in “tDeterSegs” and “tSegProb” are relative to the beginning of the zone or the area. Currently the value of this field cannot be changed in any input or update form. The default is the positions are always relative to the beginning of the area.

Methods, Cycle Lengths, Method Variables and Method Change Delays

General Variables

“otGenVariables” hold the observed general variable values and “otGenVariablesGraph” holds the graph for the updating window. Note that ‘unit’ in “tGenVariables” is a relict from older versions of the DAT and *is not used any longer*.

Resources and Repositories

Note that ‘Type’ in “tActResDemand” can only assume the values “use perm.”, “use temp.”, “produce” and “other”.

Others

ImpExp*** are used for Import and Export from or to text files. "tTime" has only one record which holds the construction start time of the current (active) update phase. It is adjusted at the time of creating new simulator input files. "rtTimeCost" holds the data points for the time-cost-scattergrams and "rtTimeCostGraph" holds the frequency plots for time and cost. "rtTimePosGraph" holds the min-max time-way-plot. "Messages" holds the error messages of the input data checker and observations checker.

B.2 Relevant Code

The code in the "DAT Input and Updating Module" database is in VBA (visual basic for applications). There are some modules which contain the bulk of the code which provides most of the functionality of the "DAT Input and Updating Module". In addition to this there is some code behind most forms. This code could be subdivided into three categories:

- code which performs basic program operations like navigation within the menu system, sorting and filtering of data which is presented in the form and the like
- code which performs basic database operations like deleting or adding a record. It thereby affects the content of the database but the code does not depend on the actual meaning of the data it changes.
- code which changes the contents of the database in a way which requires knowledge about the actual meaning of the contents. (Example: In order to delete a zone one must first delete all records in "tZoneProb" which are related to that zone, before it can be deleted in "tZones".)

This appendix shows the code of all modules and code behind forms which falls into the last category.

Module: Auxiliary

```
Option Compare Database
Option Explicit
```

```
Public Sub SetEndpos(Target As Recordset, ByVal EndPos As Double)
    Target!EndposProbMin = 0
    Target!EndposMin = EndPos
    Target!EndposMode = EndPos
    Target!EndposMax = EndPos
    Target!EndposProbMax = 0
End Sub
```

```
Public Sub SetEndposMin(Target As Recordset, ByVal EndPos As Double)
    If Target!EndposMin < EndPos Then
        Target!EndposMin = EndPos
        Target!EndposProbMin = 0
    End If
    If Target!EndposMode >= 0 And Target!EndposMode < EndPos Then _
        Target!EndposMode = EndPos
    If Target!EndposMax < EndPos Then
        Target!EndposMax = EndPos
        Target!EndposProbMax = 0
    End If
End Sub
```

```
Public Sub SetLength(Target As Recordset, ByVal EndPos As Double)
    Target!LengthProbMin = 0
    Target!LengthMin = EndPos
    Target!LengthMode = EndPos
    Target!LengthMax = EndPos
    Target!LengthProbMax = 0
End Sub
```

```
Public Sub SetLengthMin(Target As Recordset, ByVal Length As Double)
    If Target!LengthMin < Length Then
        Target!LengthMin = Length
        Target!LengthProbMin = 0
    End If
    If Target!LengthMode >= 0 And Target!LengthMode < Length Then _
        Target!LengthMode = Length
    If Target!LengthMax < Length Then
        Target!LengthMax = Length
        Target!LengthProbMax = 0
    End If
End Sub
```

```
Public Function MaxDb1(ByVal A As Double, ByVal B As Double) As Double
    MaxDb1 = B
    If A > B Then MaxDb1 = A
End Function
```

```
Public Function MinDb1(ByVal A As Double, ByVal B As Double) As Double
    MinDb1 = B
    If A < B Then MinDb1 = A
End Function
```

```
Public Sub CalculateSegmentFwd(Target As Recordset, BeginMin, BeginMode, BeginMax)
    Target.Edit
```



```
If Target!GenerationMode = "0" Then ' length
    Target!EndposProbMax = 0
    Target!EndposMin = BeginMin + Target!LengthMin
    Target!EndposMode = Target!LengthMode ' in case lengthmode = -1
    If Target!EndposMode >= 0 Then Target!EndposMode = BeginMode + Target!LengthMode
    Target!EndposMax = BeginMax + Target!LengthMax
    Target!EndposProbMax = 0
ElseIf Target!GenerationMode = "1" Then
    Target!EndposMin = Auxiliary.MaxDbl(BeginMin, Target!EndposMin)
    If Target!EndposMode >= 0 Then _
        Target!EndposMode = Auxiliary.MaxDbl(BeginMin, Target!EndposMode)
    Target!EndposMax = Auxiliary.MaxDbl(BeginMin, Target!EndposMax)
    Target!LengthProbMin = 0
    Target!LengthMin = Auxiliary.MaxDbl(0, Target!EndposMin - BeginMax)
    Target!LengthMode = Target!EndposMode ' in case endposmode = -1
    If Target!EndposMode >= 0 Then _
        Target!LengthMode = Auxiliary.MaxDbl(0, Target!EndposMode - BeginMode)
    Target!LengthMax = Auxiliary.MaxDbl(0, Target!EndposMax - BeginMin)
    Target!LengthProbMax = 0
End If
Target.Update
End Sub
```

Module: Checker

Option Compare Database

Option Explicit

' GENERAL NOTE: ONLY PRIOR INPUT AND INPUT OF THE CURRENT UPDATE PHASE ARE CHECKED

```
Public Sub CheckInputData()
    Dim DB As Database, Messages As Recordset, NoErrors As Boolean, UpdatePhase As Long
    Set DB = CurrentDb()
    Set Messages = DB.OpenRecordset("Messages", dbOpenTable)
    If IsNull(DLookup("ID", "tUpdatePhases", "Current = TRUE")) Then
        UpdatePhase = 1
    Else
        UpdatePhase = DLookup("ID", "tUpdatePhases", "Current = TRUE")
    End If
    NoErrors = True
    NoErrors = NoErrors And CheckTransProb(DB, Messages)
    NoErrors = NoErrors And CheckStartProb(DB, Messages)
    NoErrors = NoErrors And CheckBoreholes(DB, Messages)
    NoErrors = NoErrors And CheckAssocProb(DB, Messages)
    NoErrors = NoErrors And CheckParamSet(DB, Messages)
    NoErrors = NoErrors And CheckParameter(DB, Messages)
    NoErrors = NoErrors And CheckGeometry(DB, Messages)
    NoErrors = NoErrors And CheckMethod(DB, Messages)
    NoErrors = NoErrors And CheckDelays(DB, Messages)
    NoErrors = NoErrors And CheckZoneProb(DB, Messages, UpdatePhase)
    NoErrors = NoErrors And CheckSegProb(DB, Messages, UpdatePhase)
    NoErrors = NoErrors And CheckMarkovLength(DB, Messages, UpdatePhase)
    NoErrors = NoErrors And CheckVarProb(DB, Messages, UpdatePhase)
    NoErrors = NoErrors And CheckHeadProb(DB, Messages, UpdatePhase)
    NoErrors = NoErrors And CheckGenVarProb(DB, Messages, UpdatePhase)
    NoErrors = NoErrors And CheckEquations(DB, Messages)
    If NoErrors Then
        AddMessage Messages, "Input Data Checker: no errors found"
    End If
    Messages.Close
    DB.Close
End Sub
```

```
Public Sub CheckObservations()
    Dim DB As Database, Messages As Recordset, NoErrors As Boolean
    Set DB = CurrentDb()
    Set Messages = DB.OpenRecordset("Messages", dbOpenTable)
```

```
NoErrors = True
NoErrors = NoErrors And CheckGaps(DB, Messages)
NoErrors = NoErrors And CheckTunnelTimes(DB, Messages)
If NoErrors Then
    AddMessage Messages, "Observations Checker: no errors found"
End If
Messages.Close
DB.Close
End Sub

Public Function CheckTransProb(DB As Database, Messages As Recordset) As Boolean
    Dim TransProb As Recordset, Text As String
    Set TransProb = DB.OpenRecordset("cqMarkovTransProb", dbOpenDynaset, dbReadOnly)
    CheckTransProb = True
    TransProb.FindFirst "ProbSum <> 1.0"
    ' the probabilities are summed up in the query --> just search for those <> 1
    Do Until TransProb.NoMatch
        CheckTransProb = False
        Text = "Transition Probabilities: sum of probabilities from state '" & _
            TransProb!FromParamStatName & "' <> 1.0 (parameter set: '" & _
            CStr(TransProb!ParamSet) & "', parameter: '" & TransProb!ParamName
        If TransProb!Current Then
            Text = Text & ", current update phase)"
        Else
            Text = Text & ", initial [prior] input)"
        End If
        AddMessage Messages, Text
        TransProb.FindNext "ProbSum <> 1.0"
    Loop
End Function

Public Function CheckStartProb(DB As Database, Messages As Recordset) As Boolean
    Dim StartProb As Recordset, Text As String
    Set StartProb = DB.OpenRecordset("cqMarkovStartProb", dbOpenDynaset, dbReadOnly)
    CheckStartProb = True
    StartProb.FindFirst "ProbSum <> 1.0"
    ' same as with transition probabilities
    Do Until StartProb.NoMatch
        CheckStartProb = False
        Text = "Parameter Set '" & StartProb!ParamSet & "', Parameter '" & _
            StartProb!ParamName & "': sum of start probabilities <> 1.0 "
        If StartProb!Current Then
            Text = Text & "(current update phase)"
        Else
            Text = Text & "(initial [prior] input)"
        End If
        AddMessage Messages, Text
        StartProb.FindNext "ProbSum <> 1.0"
    Loop
End Function

Public Function CheckBoreholes(DB As Database, Messages As Recordset) As Boolean
    Dim Borehole As Recordset, Text As String
    Set Borehole = DB.OpenRecordset("cqBoreholes", dbOpenDynaset, dbReadOnly)
    CheckBoreholes = True
    Borehole.FindFirst "ProbSum <> 1.0"
    ' same as with transition probabilities
    Do Until Borehole.NoMatch
        CheckBoreholes = False
        Text = "Borehole '" & Borehole!Name & "', Parameter '" & _
            Borehole!ParamName & "': sum of param. state probabilities <> 1.0 "
        AddMessage Messages, Text
        Borehole.FindNext "ProbSum <> 1.0"
    Loop
End Function
```

```
Public Function CheckAssocProb(DB As Database, Messages As Recordset) As Boolean
    Dim AssocProb As Recordset, Text As String
    Set AssocProb = DB.OpenRecordset("cqAssociations", dbOpenDynaset, dbReadOnly)
    CheckAssocProb = True
    AssocProb.FindFirst "ProbSum <> 1.0"
    ' same as with transition probabilities
    Do Until AssocProb.NoMatch
        CheckAssocProb = False
        Text = "Association Geometry/GC/Methods: sum of probabilities from geometry " & _
            CStr(AssocProb!Geometry) & ", ground class '" & AssocProb!GCName & _
            "' <> 1.0 "
        If AssocProb!Current Then
            Text = Text & "(current update phase)"
        Else
            Text = Text & "(initial [prior] input)"
        End If
        AddMessage Messages, Text
        AssocProb.FindNext "ProbSum <> 1.0"
    Loop
End Function
```

```
Public Function CheckParamSet(DB As Database, Messages As Recordset) As Boolean
    Dim ParamSets As Recordset, Zones As Recordset, Areas As Recordset
    Set ParamSets = DB.OpenRecordset("tParamSets", dbOpenTable, dbReadOnly)
    Set Zones = DB.OpenRecordset("tZones", dbOpenTable, dbReadOnly)
    Zones.Index = "ParamSet"
    Set Areas = DB.OpenRecordset("tAreas", dbOpenTable, dbReadOnly)
    Areas.Index = "ParamSet"
    CheckParamSet = True
    If Not (ParamSets.EOF And ParamSets.BOF) Then
        ParamSets.MoveFirst
        Areas.Seek "=", ParamSets!ID
        Zones.Seek "=", ParamSets!ID
        If Areas.NoMatch And Zones.NoMatch Then
            ' neither an area nor a zone with this parameter set
            AddMessage Messages, "Parameter Set " & CStr(ParamSets!ID) & _
                " is not used by any zone or area!"
            CheckParamSet = False
        ElseIf Not (Areas.NoMatch Or Zones.NoMatch) Then
            ' a parameter set is used in both an area and a zone
            AddMessage Messages, "Parameter Set " & CStr(ParamSets!ID) & _
                " is used in both an area and a zone!"
            CheckParamSet = False
        End If
        ParamSets.MoveNext
    End If
End Function
```

```
Public Function CheckParameter(DB As Database, Messages As Recordset) As Boolean
    Dim Parameters As Recordset, ParamXSet As Recordset, DeterSegs As Recordset
    Dim SemiDeterSegs As Recordset, Areas As Recordset, Zones As Recordset
    Set Parameters = DB.OpenRecordset("tParameters", dbOpenTable, dbReadOnly)
    Set ParamXSet = DB.OpenRecordset("tParam_X_Set", dbOpenTable, dbReadOnly)
    Set DeterSegs = DB.OpenRecordset("eDeterSegs", dbOpenDynaset, dbReadOnly)
    Set SemiDeterSegs = DB.OpenRecordset("eSemiDeterSegs", dbOpenDynaset, dbReadOnly)
    Set Zones = DB.OpenRecordset("tZones", dbOpenSnapshot)
    Set Areas = DB.OpenRecordset("tAreas", dbOpenSnapshot)
    ParamXSet.Index = "Parameter"
    CheckParameter = True
    If Not (Parameters.EOF And Parameters.BOF) Then
        Parameters.MoveFirst
        ParamXSet.Seek "=", Parameters!ID
        If ParamXSet.NoMatch Then
            AddMessage Messages, "Parameter '" & Parameters!Name & _
                "' is not used in any parameter set!"
            CheckParameter = False
        End If
    End If
```

```
Parameters.MoveNext
End If
If (ParamXSet.EOF And ParamXSet.BOF) Then Exit Function
ParamXSet.MoveFirst
Do Until ParamXSet.EOF
  If ParamXSet!GenerationMode = "1" Then
    DeterSegs.FindLast "ParamSet = " & CStr(ParamXSet!ParamSet)
  ElseIf ParamXSet!GenerationMode = "2" Then
    SemiDeterSegs.FindLast "ParamSet = " & CStr(ParamXSet!ParamSet)
  End If
  If ParamXSet!GenerationMode <> "0" Then
    Zones.FindFirst "ParamSet = " & CStr(ParamXSet!ParamSet)
    If Not Zones.NoMatch Then
      Zones.FindNext "ParamSet = " & CStr(ParamXSet!ParamSet)
      If Not Zones.NoMatch Then
        AddMessage Messages, _
          "Do not twice use a parameter set which has other than " & _
          "Markov type parameters! (param. set: " & CStr(ParamXSet!ParamSet) & ")"
        CheckParameter = False
      End If
      Zones.FindFirst "ParamSet = " & CStr(ParamXSet!ParamSet)
      If ParamXSet!GenerationMode = "1" And Not DeterSegs.NoMatch Then
        If DeterSegs!EndPos <> Zones!EndposMin Or _
          (DeterSegs!EndPos <> Zones!EndposMode And Zones!EndposMode > 0) Or _
          DeterSegs!EndPos <> Zones!EndposMax Then
          AddMessage Messages, _
            "The endpos of the last deterministic segment in zone '" & _
            & CStr(Zones!Name) & "' does not coincide with the end of the zone."
          CheckParameter = False
        End If
      ElseIf ParamXSet!GenerationMode = "2" And Not SemiDeterSegs.NoMatch Then
        If SemiDeterSegs!EndposMin <> Zones!EndposMin Or _
          (SemiDeterSegs!EndposMode <> Zones!EndposMode _
            And Zones!EndposMode > 0 And SemiDeterSegs!EndposMode > 0) Or _
          SemiDeterSegs!EndposMax <> Zones!EndposMax Then
          AddMessage Messages, _
            "The endpos of the last semidetermin. segment in zone '" & _
            & CStr(Zones!Name) & "' does not coincide with the end of the zone."
          CheckParameter = False
        End If
      End If
    End If
    Areas.FindFirst "ParamSet = " & CStr(ParamXSet!ParamSet)
    If Not Areas.NoMatch Then
      Areas.FindNext "ParamSet = " & CStr(ParamXSet!ParamSet)
      If Not Areas.NoMatch Then
        AddMessage Messages, _
          "Do not twice use a parameter set which has other than " & _
          "Markov type parameters! (param. set: " & CStr(ParamXSet!ParamSet) & ")"
        CheckParameter = False
      End If
      Areas.FindFirst "ParamSet = " & CStr(ParamXSet!ParamSet)
      If ParamXSet!GenerationMode = "1" And Not DeterSegs.NoMatch Then
        If DeterSegs!EndPos <> Areas!Length Then
          AddMessage Messages, _
            "The endpos of the last deterministic segment in area '" & _
            CStr(Areas!Name) & "' does not coincide with the length of the area."
          CheckParameter = False
        End If
      ElseIf ParamXSet!GenerationMode = "2" And Not SemiDeterSegs.NoMatch Then
        If SemiDeterSegs!EndposMin <> Areas!Length Or _
          (SemiDeterSegs!EndposMode <> Areas!Length And _
            SemiDeterSegs!EndposMode > 0) Or _
          SemiDeterSegs!EndposMax <> Areas!Length Then
          AddMessage Messages, _
            "The endpos of the last semidetermin. segment in area '" & _
            CStr(Zones!Name) & "' does not coincide with the length of the area."
          CheckParameter = False
        End If
      End If
    End If
  End If
End Do
```

```
        End If
    End If
End If
    End If
    ParamXSet.MoveNext
Loop
DeterSegs.Close
SemiDeterSegs.Close
Areas.Close
Zones.Close
ParamXSet.Close
End Function

Public Function CheckGeometry(DB As Database, Messages As Recordset) As Boolean
    Dim Geometries As Recordset, Tunnels As Recordset
    Set Geometries = DB.OpenRecordset("tGeometries", dbOpenTable, dbReadOnly)
    Set Tunnels = DB.OpenRecordset("tTunnels", dbOpenTable, dbReadOnly)
    Tunnels.Index = "Geometry"
    CheckGeometry = True
    If Not (Geometries.EOF And Geometries.BOF) Then
        Geometries.MoveFirst
        Tunnels.Seek "=", Geometries!ID
        If Tunnels.NoMatch Then
            AddMessage Messages, "Geometry " & CStr(Geometries!ID) & _
                " is not used in any tunnel!"
            CheckGeometry = False
        End If
        Geometries.MoveNext
    End If
End Function

Public Function CheckMethod(DB As Database, Messages As Recordset) As Boolean
    Dim Methods As Recordset, Assoc As Recordset
    Set Methods = DB.OpenRecordset("tMethods", dbOpenTable, dbReadOnly)
    Set Assoc = DB.OpenRecordset("tCGC_X_Meth_X_Geom", dbOpenTable, dbReadOnly)
    Assoc.Index = "Method"
    CheckMethod = True
    If Not (Methods.EOF And Methods.BOF) Then
        Methods.MoveFirst
        Assoc.Seek "=", Methods!ID
        If Assoc.NoMatch Then
            AddMessage Messages, "Method '" & Methods!Name & _
                "' is not associated with any ground class/geometry combination!"
            CheckMethod = False
        End If
        Methods.MoveNext
    End If
End Function

Public Function CheckGaps(DB As Database, Messages As Recordset) As Boolean
    Dim ZoneStat As Recordset, AreaStat As Recordset, Zones As Recordset
    Dim Areas As Recordset, Bookmark As Variant, WrongPart As Boolean
    Dim SegBegin As Double, SegEnd As Double, Zone As Long, Area As Long
    Dim Parameter As Long, PartlyVisible As Boolean
    Set ZoneStat = DB.OpenRecordset("oqZoneParamStat", dbOpenDynaset, dbReadOnly)
    Set Zones = DB.OpenRecordset("oqZonesSeq", dbOpenDynaset, dbReadOnly)
    CheckGaps = True
    Zone = -1
    Parameter = -1
    If Not (ZoneStat.EOF And ZoneStat.BOF) Then
        Zones.MoveFirst
        ZoneStat.MoveFirst
        Do Until ZoneStat.EOF
            SegBegin = ZoneStat!EndPos - ZoneStat!Length
            If Zone <> ZoneStat!Zone Or ZoneStat!Parameter <> Parameter Then
                ' new zone or new parameter
                '--> checks concerning the begin and end of zones
            End If
        Loop
    End If
End Function
```

```
Parameter = ZoneStat!Parameter
Zone = ZoneStat!Zone
ZoneStat.MovePrevious
' go to previous segment which is last of previous zone and/or parameter
If ZoneStat.BOF Then ' this is the very first segment
    ZoneStat.MoveFirst
Else
    Zones.FindFirst "ID = " & CStr(ZoneStat!Zone)
    ' the recordset contains always all zones, i.e. there must be a match
    ' if there is no observation several fields are NULL, e.g. EndposMode
    If Not IsNull(Zones!EndposMode) Then
        If Not (ZoneStat!Part Or Zones!EndposMode = ZoneStat!EndPos) Then
            ' if last segment is not partly visible, its endpos must coincide
            ' with the endpos of the zone
            CheckGaps = False
            AddMessage Messages, "The last segment in zone " & _
                CStr(ZoneStat!Name) & " does not end where the " & _
                "zone ends or 'partly visible' is not checked."
        End If
    End If
    ZoneStat.MoveNext ' move back to segment we started this loop with
End If
Zones.FindFirst "ID = " & CStr(ZoneStat!Zone)
Area = Zones!Area
If IsNull(Zones!EndposMode) Then ' no observation about this zone
    CheckGaps = False
    AddMessage Messages, "No observation about zone " & CStr(ZoneStat!Name) & _
        " although parameter states were observed."
End If
If Not ZoneStat!Part Then
    ' the first observed segment is truly the first segment of the zone
    Zones.MovePrevious
    ' move to previous zone to check if endpos of that zone is correct
    If Zones.BOF Then
        Zones.MoveFirst
        ' this is the first zone of the area (they're sorted in the recordset)
        ' i.e. segment must begin with 0
        If SegBegin <> 0 Then
            CheckGaps = False
            AddMessage Messages, "Observations in zone " & CStr(ZoneStat!Name) & _
                " do not start with zero or 'partly visible' is not checked."
        End If
    ElseIf Zones!Area <> Area Then
        Zones.MoveNext ' move back to segment we started this loop with
        ' this is also the first zone of the area
        If SegBegin <> 0 Then
            CheckGaps = False
            AddMessage Messages, "Observations in zone " & CStr(ZoneStat!Name) & _
                " do not start with zero or 'partly visible' is not checked."
        End If
    ElseIf Zones!EndposMode <> SegBegin Then
        Zones.MoveNext ' move back to segment we started this loop with
        CheckGaps = False
        AddMessage Messages, "The first segment in zone " & _
            CStr(ZoneStat!Name) & " does not begin where the previous " & _
            "zone ends or 'partly visible' is not checked."
    Else
        Zones.MoveNext ' move back to segment we started this loop with
    End If
End If
SegEnd = SegBegin ' these are not defined because there is no previous
PartlyVisible = False ' segment in this zone, choose in order not to
                        ' trigger any action later in the code
End If
If Not IsNull(Zones!EndposMode) Then
    ' do not check this if observed zone is missing
    If ZoneStat!Part And Zones!Status <> "Current" Then
        ' partly visible segments can only occur in "current" zones
    End If
End If
```

```
        CheckGaps = False
        AddMessage Messages, "There is a partly visible segment in zone " & _
            CStr(ZoneStat!Name) & "but the zone's status is not 'Current'."
    End If
End If
If ZoneStat!Part And Not PartlyVisible Then
    WrongPart = True ' there is possibly a 'partly visible' that should not be
    If PartlyVisible And (ZoneStat!EndPos - ZoneStat!Length > SegEnd) Then _
        WrongPart = False
        ' 'partly visible' is ok if the previous segment is also 'partly visible'
        ' and there is a gap
        Bookmark = ZoneStat.Bookmark
        ZoneStat.MovePrevious
        If ZoneStat.BOF Then
            WrongPart = False 'partly visible' is ok if at the beginning of the sequence
        ElseIf ZoneStat!Zone <> Zone Or ZoneStat!Parameter <> Parameter Then _
            WrongPart = False 'partly visible' is ok if at the beginning of the sequence
        End If
        ZoneStat.Bookmark = Bookmark
        ZoneStat.MoveNext
        If ZoneStat.EOF Then
            WrongPart = False ' 'partly visible' is ok if a the end of the sequence
        ElseIf ZoneStat!Zone <> Zone Or ZoneStat!Parameter <> Parameter Then _
            WrongPart = False ' 'partly visible' is ok if a the end of the sequence
        ElseIf ZoneStat!Part Then
            WrongPart = False
            ' 'partly visible' is probably ok (will check in next loop if there is gap)
        End If
        ZoneStat.Bookmark = Bookmark
        If WrongPart Then
            CheckGaps = False
            AddMessage Messages, "An observed segment in zone " & CStr(ZoneStat!Name) _
                & " is erroneously marked 'partly visible' at position " & CStr(SegEnd)
        End If
    End If
    If Not ((ZoneStat!EndPos - ZoneStat!Length = SegEnd) Or _
        (PartlyVisible And ZoneStat!Part)) Then
        ' if there is a gap both the segment before and after must be partly visible
        CheckGaps = False
        AddMessage Messages, "There is a gap in the param. state sequence in zone " _
            & CStr(ZoneStat!Name) & ", at position " & CStr(SegEnd) & _
            " or 'partly visible' is not checked"
    End If
    SegEnd = ZoneStat!EndPos
    PartlyVisible = ZoneStat!Part
    ZoneStat.MoveNext
Loop
End If
ZoneStat.Close
Zones.Close
' same thing for areas
Set AreaStat = DB.OpenRecordset("oqAreaParamStat", dbOpenDynaset, dbReadOnly)
Set Areas = DB.OpenRecordset("tAreas", dbOpenTable, dbReadOnly)
Areas.Index = "PrimaryKey"
Area = -1
Parameter = -1
If Not (AreaStat.EOF And AreaStat.BOF) Then
    AreaStat.MoveFirst
    Do Until AreaStat.EOF
        SegBegin = AreaStat!EndPos - AreaStat!Length
        If Area <> AreaStat!Area Or AreaStat!Parameter <> Parameter Then
            Parameter = AreaStat!Parameter
            Area = AreaStat!Area
            AreaStat.MovePrevious
            If AreaStat.BOF Then
                AreaStat.MoveFirst
            Else
                Areas.Seek "=", AreaStat!Area
            End If
        End If
    Loop
End If
```

```
' the recordset contains all areas, i.e. there must be a match
' the endpos of the previous segment must coincide with
' the length of the previous area if it is not partly visible
If Not (AreaStat!Part Or Areas!Length = AreaStat!EndPos) Then
    CheckGaps = False
    AddMessage Messages, "The last segment in area " & _
        CStr(AreaStat!Name) & " does not end where the " & _
        "area ends or 'partly visible' is not checked."
End If
AreaStat.MoveNext
End If
Area = AreaStat!Area
Areas.Seek "=", AreaStat!Area
Area = Areas!ID
If Not (AreaStat!Part Or SegBegin = 0) Then
    ' the first observed segment must start with 0 or be partly visible
    CheckGaps = False
    AddMessage Messages, "Observations in area " & CStr(AreaStat!Name) & _
        " do not start with zero or 'partly visible' is not checked."
End If
SegEnd = SegBegin      ' see in the zones part above
PartlyVisible = False
End If
If Not ((AreaStat!EndPos - AreaStat!Length = SegEnd) Or _
    (PartlyVisible And AreaStat!Part)) Then
    ' if there is a gap, the segments before and after must be partly visible
    CheckGaps = False
    AddMessage Messages, "There is a gap in the param. state sequence in area " & _
        & CStr(AreaStat!Name) & ", at position " & CStr(SegEnd) & _
        " or 'partly visible' is not checked"
End If
SegEnd = AreaStat!EndPos
PartlyVisible = AreaStat!Part
AreaStat.MoveNext
Loop
End If
AreaStat.Close
Areas.Close
End Function

Public Function CheckDelays(DB As Database, Messages As Recordset) As Boolean
    Dim Delays As Recordset, ObsDelays As Recordset, Tunnels As Recordset, Tunnel As Long

    Set Delays = DB.OpenRecordset("qTunDelaysPost", dbOpenDynaset, dbReadOnly)
    Set ObsDelays = DB.OpenRecordset("oqTunDelays", dbOpenDynaset, dbReadOnly)
    Set Tunnels = DB.OpenRecordset("tTunnels", dbOpenTable, dbReadOnly)
    CheckDelays = True
    If Delays.BOF And Delays.EOF Then Exit Function
    Tunnels.Index = "PrimaryKey"
    ObsDelays.FindFirst "Part = TRUE"
    Do Until ObsDelays.NoMatch
        Tunnels.Seek "=", ObsDelays!Tunnel
        Tunnel = ObsDelays!Tunnel
        Delays.FindFirst "Tunnel = " & CStr(Tunnel) & " And Status = 'Current'"
        If Delays.NoMatch Then
            AddMessage Messages, "In tunnel '" & CStr(Tunnels!Name) & "' a delay " & _
                "was observed and marked 'still going on', but there is no 'current' " & _
                "delay in posterior input."
            CheckDelays = False
        Else
            Delays.FindNext "Tunnel = " & CStr(Tunnel) & " And Status = 'Current'"
            If Not Delays.NoMatch Then
                AddMessage Messages, "In tunnel '" & CStr(Tunnels!Name) & "' there are " & _
                    "two delays marked 'current' in posterior input."
                CheckDelays = False
            End If
        End If
        ObsDelays.FindNext "Part = TRUE"
```



```
Loop
Delays.FindFirst "Status = 'Current'"
Do Until Delays.NoMatch
    Tunnels.Seek "=", Delays!Tunnel
    Tunnel = Delays!Tunnel
    ObsDelays.FindFirst "Tunnel = " & CStr(Tunnel) & " And Part = TRUE"
    If ObsDelays.NoMatch Then
        AddMessage Messages, "In tunnel '" & CStr(Tunnels!Name) & "' a delay " & _
            "is marked 'current' in posterior input but there is no observed delay " & _
            "which is marked 'still going on'."
        CheckDelays = False
    Else
        ObsDelays.FindNext "Tunnel = " & CStr(Tunnel) & " And Part = TRUE"
        If Not ObsDelays.NoMatch Then
            AddMessage Messages, "In tunnel '" & CStr(Tunnels!Name) & "' there are " & _
                "two observed delays marked 'still going on'."
            CheckDelays = False
        End If
    End If
    Delays.FindNext "Status = 'Current'"
Loop
End Function

Public Function CheckZoneProb(DB As Database, Messages As Recordset, _
    UpdatePhase As Long) As Boolean
    Dim ZoneProb As Recordset, Zones As Recordset, Text As String, UPText As String
    Set ZoneProb = DB.OpenRecordset("tZoneProb", dbOpenTable, dbReadOnly)
    Set Zones = DB.OpenRecordset("tZones", dbOpenTable, dbReadOnly)
    Zones.Index = "PrimaryKey"
    CheckZoneProb = True
    If Not (ZoneProb.EOF And ZoneProb.BOF) Then
        ZoneProb.MoveFirst
        Do Until ZoneProb.EOF
            If ZoneProb!UpdatePhase = 1 Or ZoneProb!UpdatePhase = UpdatePhase Then
                If ZoneProb!UpdatePhase = 1 Then
                    UPText = " (initial [prior] input)"
                Else
                    UPText = " (current update phase)"
                End If
                If ZoneProb!GenerationMode <> "1" Then
                    ' genmode 'length' or 'both'
                    If ZoneProb!LengthProbMin + ZoneProb!LengthProbMax > 1# Then
                        Zones.Seek "=", ZoneProb!Zone
                        Text = "Length: prob. min + prob. max > 1.0 in zone '" & Zones!Name & "'"
                        AddMessage Messages, Text & UPText
                        CheckZoneProb = False
                    End If
                    If ZoneProb!LengthMode = -1 Then
                        If ZoneProb!LengthMin > ZoneProb!LengthMax Then
                            Zones.Seek "=", ZoneProb!Zone
                            Text = "Min length > max length in zone '" & Zones!Name & "'"
                            AddMessage Messages, Text & UPText
                            CheckZoneProb = False
                        End If
                    Else
                        If ZoneProb!LengthMin > ZoneProb!LengthMode Or _
                            ZoneProb!LengthMode > ZoneProb!LengthMax Then
                            Zones.Seek "=", ZoneProb!Zone
                            Text = "length must be min <= mode <= max in zone '" & Zones!Name & "'"
                            AddMessage Messages, Text & UPText
                            CheckZoneProb = False
                        End If
                    End If
                End If
            End If
        Loop
        If ZoneProb!GenerationMode <> "0" Then
            ' genmode 'endpos' or 'both'
            If ZoneProb!EndposProbMin + ZoneProb!EndposProbMax > 1# Then
                Zones.Seek "=", ZoneProb!Zone
```

```
Text = "End pos.: prob. min + prob. max > 1.0 in zone " & Zones!Name
AddMessage Messages, Text & UPText
CheckZoneProb = False
End If
If ZoneProb!EndposMode = -1 Then
    If ZoneProb!EndposMin > ZoneProb!EndposMax Then
        Zones.Seek "=", ZoneProb!Zone
        Text = "Min endpos > max endpos in zone " & Zones!Name
        AddMessage Messages, Text & UPText
        CheckZoneProb = False
    End If
Else
    If ZoneProb!EndposMin > ZoneProb!EndposMode Or _
        ZoneProb!EndposMode > ZoneProb!EndposMax Then
        Zones.Seek "=", ZoneProb!Zone
        Text = "Endpos must be min <= mode <= max in zone " & Zones!Name
        AddMessage Messages, Text & UPText
        CheckZoneProb = False
    End If
End If
End If
End If
ZoneProb.MoveNext
Loop
End If
Zones.Close
ZoneProb.Close
End Function
```

```
Public Function CheckSegProb(DB As Database, Messages As Recordset, _
    UpdatePhase As Long) As Boolean
    Dim SegProb As Recordset, Segs As Recordset, Text As String, UPText As String
    Set SegProb = DB.OpenRecordset("tSegProb", dbOpenTable, dbReadOnly)
    Set Segs = DB.OpenRecordset("tSemiDeterSegs", dbOpenTable, dbReadOnly)
    Segs.Index = "PrimaryKey"
    CheckSegProb = True
    If Not (SegProb.EOF And SegProb.BOF) Then
        SegProb.MoveFirst
        Do Until SegProb.EOF
            If SegProb!UpdatePhase = 1 Or SegProb!UpdatePhase = UpdatePhase Then
                If SegProb!UpdatePhase = 1 Then
                    UPText = " (initial [prior] input)"
                Else
                    UPText = " (current update phase)"
                End If
                If SegProb!GenerationMode <> "1" Then
                    ' genmode 'length' or 'both'
                    If SegProb!LengthProbMin + SegProb!LengthProbMax > 1# Then
                        Segs.Seek "=", SegProb!Segment
                        Text = "Length: prob. min + prob. max > 1.0 in segment " & _
                            CStr(Segs!Order) & ", parameter set " & CStr(Segs!ParamSet)
                        AddMessage Messages, Text & UPText
                        CheckSegProb = False
                    End If
                    If SegProb!LengthMode = -1 Then
                        If SegProb!LengthMin > SegProb!LengthMax Then
                            Segs.Seek "=", SegProb!Segment
                            Text = "Min length > max length in segment " & _
                                CStr(Segs!Order) & ", parameter set " & CStr(Segs!ParamSet)
                            AddMessage Messages, Text & UPText
                            CheckSegProb = False
                        End If
                    Else
                        If SegProb!LengthMin > SegProb!LengthMode Or _
                            SegProb!LengthMode > SegProb!LengthMax Then
                            Segs.Seek "=", SegProb!Segment
                            Text = "length must be min <= mode <= max in segment " & _
                                CStr(Segs!Order) & ", parameter set " & CStr(Segs!ParamSet)
                        End If
                    End If
                End If
            End If
        Loop
    End If
End Function
```

```

        AddMessage Messages, Text & UPText
        CheckSegProb = False
    End If
End If
End If
If SegProb!GenerationMode <> "0" Then
    ' genmode 'endpos' or 'both'
    If SegProb!EndposProbMin + SegProb!EndposProbMax > 1# Then
        Segs.Seek "=", SegProb!Segment
        Text = "End pos.: prob. min + prob. max > 1.0 in segment " & _
            CStr(Segs!Order) & ", parameter set " & CStr(Segs!ParamSet)
        AddMessage Messages, Text & UPText
        CheckSegProb = False
    End If
    If SegProb!EndposMode = -1 Then
        If SegProb!EndposMin > SegProb!EndposMax Then
            Segs.Seek "=", SegProb!Segment
            Text = "Min endpos > max endpos in segment " & _
                CStr(Segs!Order) & ", parameter set " & CStr(Segs!ParamSet)
            AddMessage Messages, Text & UPText
            CheckSegProb = False
        End If
    Else
        If SegProb!EndposMin > SegProb!EndposMode Or _
            SegProb!EndposMode > SegProb!EndposMax Then
            Segs.Seek "=", SegProb!Segment
            Text = "Endpos must be min <= mode <= max in segment " & _
                CStr(Segs!Order) & ", parameter set " & CStr(Segs!ParamSet)
            AddMessage Messages, Text & UPText
            CheckSegProb = False
        End If
    End If
End If
    SegProb.MoveNext
Loop
End If
Segs.Close
SegProb.Close
End Function

Public Function CheckMarkovLength(DB As Database, Messages As Recordset, _
    UpdatePhase As Long) As Boolean
    Dim MarkovLength As Recordset, ParamStates As Recordset, Text As String
    Dim UPText As String
    Set MarkovLength = DB.OpenRecordset("tMarkovLengths", dbOpenTable, dbReadOnly)
    Set ParamStates = DB.OpenRecordset("tParamStates", dbOpenTable, dbReadOnly)
    ParamStates.Index = "PrimaryKey"
    CheckMarkovLength = True
    If Not (MarkovLength.EOF And MarkovLength.BOF) Then
        MarkovLength.MoveFirst
        Do Until MarkovLength.EOF
            If MarkovLength!UpdatePhase = 1 Or MarkovLength!UpdatePhase = UpdatePhase Then
                If MarkovLength!UpdatePhase = 1 Then
                    UPText = " (initial [prior] input)"
                Else
                    UPText = " (current update phase)"
                End If
                If MarkovLength!LengthMode = -1 Then
                    If MarkovLength!LengthMin > MarkovLength!LengthMax Then
                        ParamStates.Seek "=", MarkovLength!ParamState
                        Text = "Markov length must be: min <= max for parameter state '" & _
                            ParamStates!Name & "', parameter set " & CStr(MarkovLength!ParamSet)
                        AddMessage Messages, Text & UPText
                        CheckMarkovLength = False
                    End If
                Else
                    If MarkovLength!LengthMin > MarkovLength!LengthMode Or _

```

```
MarkovLength!LengthMode > MarkovLength!LengthMax Then
ParamStates.Seek "=", MarkovLength!Segment
Text ="Markov length must be: min <= mode <= max for parameter state '" _
    & ParamStates!Name & "'", parameter set " & CStr(MarkovLength!ParamSet)
AddMessage Messages, Text & UPText
CheckMarkovLength = False
End If
End If
End If
MarkovLength.MoveNext
Loop
End If
ParamStates.Close
MarkovLength.Close
End Function

Public Function CheckVarProb(DB As Database, Messages As Recordset, _
    UpdatePhase As Long) As Boolean
Dim VarProb As Recordset, Vars As Recordset, Text As String, UPText As String
Dim Methods As Recordset
Set VarProb = DB.OpenRecordset("tVariableProb", dbOpenTable, dbReadOnly)
Set Vars = DB.OpenRecordset("tVariables", dbOpenTable, dbReadOnly)
Set Methods = DB.OpenRecordset("tMethods", dbOpenTable, dbReadOnly)
Vars.Index = "PrimaryKey"
Methods.Index = "PrimaryKey"
CheckVarProb = True
If Not (VarProb.EOF And VarProb.BOF) Then
VarProb.MoveFirst
Do Until VarProb.EOF
    If VarProb!UpdatePhase = 1 Or VarProb!UpdatePhase = UpdatePhase Then
        If VarProb!UpdatePhase = 1 Then
            UPText = " (initial [prior] input)"
        Else
            UPText = " (current update phase)"
        End If
        If VarProb!ProbMin + VarProb!ProbMax > 1# Then
            Vars.Seek "=", VarProb!Variable
            Methods.Seek "=", Vars!Method
            Text = "prob. min + prob. max > 1.0 in variable '" & Vars!Name & _
                "'", method " & Methods!Name & "'"
            AddMessage Messages, Text & UPText
            CheckVarProb = False
        End If
        If VarProb!Mode = -1 Then
            If VarProb!Min > VarProb!Max Then
                Vars.Seek "=", VarProb!Variable
                Methods.Seek "=", Vars!Method
                Text = "Min > max in variable '" & Vars!Name & _
                    "'", method " & Methods!Name & "'"
                AddMessage Messages, Text & UPText
                CheckVarProb = False
            End If
        Else
            If VarProb!Min > VarProb!Mode Or _
                VarProb!Mode > VarProb!Max Then
                Vars.Seek "=", VarProb!Variable
                Methods.Seek "=", Vars!Method
                Text = "values must be min <= mode <= max in variable '" & Vars!Name & _
                    "'", method " & Methods!Name & "'"
                AddMessage Messages, Text & UPText
                CheckVarProb = False
            End If
        End If
        VarProb.MoveNext
    Loop
End If
Vars.Close
```

```
VarProb.Close
Methods.Close
End Function
```

```
Public Function CheckHeadProb(DB As Database, Messages As Recordset, _
    UpdatePhase As Long) As Boolean
    Dim HeadProb As Recordset, Heads As Recordset, Text As String, UPText As String
    Dim Methods As Recordset
    Set HeadProb = DB.OpenRecordset("tHeadProb", dbOpenTable, dbReadOnly)
    Set Heads = DB.OpenRecordset("tHeads", dbOpenTable, dbReadOnly)
    Set Methods = DB.OpenRecordset("tMethods", dbOpenTable, dbReadOnly)
    Heads.Index = "PrimaryKey"
    Methods.Index = "PrimaryKey"
    CheckHeadProb = True
    If Not (HeadProb.EOF And HeadProb.BOF) Then
        HeadProb.MoveFirst
        Do Until HeadProb.EOF
            If HeadProb!UpdatePhase = 1 Or HeadProb!UpdatePhase = UpdatePhase Then
                If HeadProb!UpdatePhase = 1 Then
                    UPText = " (initial [prior] input)"
                Else
                    UPText = " (current update phase)"
                End If
                If HeadProb!CycleProbMin + HeadProb!CycleProbMax > 1# Then
                    Heads.Seek "=", HeadProb!Head
                    Methods.Seek "=", Heads!Method
                    Text = "cycle lengths: prob. min + prob. max > 1.0 in heading " & _
                        & CStr(Heads!Order) & ", method " & Methods!Name & ""
                    AddMessage Messages, Text & UPText
                    CheckHeadProb = False
                End If
                If HeadProb!CycleMode = -1 Then
                    If HeadProb!CycleMin > HeadProb!CycleMax Then
                        Heads.Seek "=", HeadProb!Head
                        Methods.Seek "=", Heads!Method
                        Text = "cycle lengths: min > max in heading " & CStr(Heads!Order) & _
                            ", method " & Methods!Name & ""
                        AddMessage Messages, Text & UPText
                        CheckHeadProb = False
                    End If
                Else
                    If HeadProb!CycleMin > HeadProb!CycleMode Or _
                        HeadProb!CycleMode > HeadProb!CycleMax Then
                        Heads.Seek "=", HeadProb!Head
                        Methods.Seek "=", Heads!Method
                        Text = "cycle lengths must be min <= mode <= max in heading " & _
                            CStr(Heads!Order) & ", method " & Methods!Name & ""
                        AddMessage Messages, Text & UPText
                        CheckHeadProb = False
                    End If
                End If
            End If
            HeadProb.MoveNext
        Loop
    End If
    Heads.Close
    HeadProb.Close
    Methods.Close
End Function
```

```
Public Function CheckGenVarProb(DB As Database, Messages As Recordset, _
    UpdatePhase As Long) As Boolean
    Dim GenVarProb As Recordset, GenVars As Recordset, Text As String, UPText As String
    Set GenVarProb = DB.OpenRecordset("tGenVarProb", dbOpenTable, dbReadOnly)
    Set GenVars = DB.OpenRecordset("tGenVariables", dbOpenTable, dbReadOnly)
    GenVars.Index = "PrimaryKey"
    CheckGenVarProb = True
```

```
If Not (GenVarProb.EOF And GenVarProb.BOF) Then
    GenVarProb.MoveFirst
    Do Until GenVarProb.EOF
        If GenVarProb!UpdatePhase = 1 Or GenVarProb!UpdatePhase = UpdatePhase Then
            If GenVarProb!UpdatePhase = 1 Then
                UPText = " (initial [prior] input)"
            Else
                UPText = " (current update phase)"
            End If
            If GenVarProb!ProbMin + GenVarProb!ProbMax > 1# Then
                GenVars.Seek "=", GenVarProb!GenVar
                Text = "prob. min + prob. max > 1.0 in general variable '" & GenVars!Name & "'"
                AddMessage Messages, Text & UPText
                CheckGenVarProb = False
            End If
            If GenVarProb!Mode = -1 Then
                If GenVarProb!Min > GenVarProb!Max Then
                    GenVars.Seek "=", GenVarProb!GenVar
                    Text = "Min > max in general variable '" & GenVars!Name & "'"
                    AddMessage Messages, Text & UPText
                    CheckGenVarProb = False
                End If
            Else
                If GenVarProb!Min > GenVarProb!Mode Or _
                    GenVarProb!Mode > GenVarProb!Max Then
                    GenVars.Seek "=", GenVarProb!GenVar
                    Text = "values must be min <= mode <= max in general variable '" & _
                        & GenVars!Name & "'"
                    AddMessage Messages, Text & UPText
                    CheckGenVarProb = False
                End If
            End If
            GenVarProb.MoveNext
        Loop
    End If
    GenVars.Close
    GenVarProb.Close
End Function

Public Function CheckEquations(DB As Database, Messages As Recordset)
    Dim Equations As Recordset, GenVar As Recordset
    CheckEquations = True
    Set Equations = DB.OpenRecordset("cqEquations", dbOpenDynaset, dbReadOnly)
    If Equations.EOF And Equations.BOF Then Exit Function
    Set GenVar = DB.OpenRecordset("tGenVariables", dbOpenTable, dbReadOnly)
    If Not (GenVar.EOF And GenVar.BOF) Then
        GenVar.MoveFirst
        Do Until GenVar.EOF
            Equations.MoveFirst
            Do Until Equations.EOF
                If Equations!UnifiedSimul <> GenVar!UnifiedSimul Then
                    If InStr(Equations!CostEquation, GenVar!Name) > 0 Or _
                        InStr(Equations!TimeEquation, GenVar!Name) > 0 Then
                        AddMessage Messages, "Method '" & Equations!MethName & _
                            "' uses gen. variable '" & GenVar!Name & "' in activity '" & _
                            Equations!ActName & "' but the positions of their 'one' & _
                            " value per simulation' switches are not the same."
                        CheckEquations = False
                    End If
                End If
            Equations.MoveNext
        Loop
        GenVar.MoveNext
    Loop
    End If
    Equations.Close
    Set Equations = DB.OpenRecordset("tActivities", dbOpenTable, dbReadOnly)
```

```
If Not (Equations.EOF And Equations.BOF) Then Equations.MoveFirst
Do Until Equations.EOF
    If InStr(Equations!TimeEquation, "round_length()") = 0 And _
        InStr(Equations!TimeEquation, "cycle_percentage()") = 0 Then
        AddMessage Messages, "It is suggested that you multiply " & _
            "the time equation in activity '" & Equations!Name & _
            "' with the function 'cycle_percentage()'"
        CheckEquations = False
    End If
    If InStr(Equations!CostEquation, "round_length()") = 0 And _
        InStr(Equations!CostEquation, "cycle_percentage()") = 0 Then
        AddMessage Messages, "It is suggested that you multiply " & _
            "the cost equation in activity '" & Equations!Name & _
            "' with the function 'cycle_percentage()'"
        CheckEquations = False
    End If
    Equations.MoveNext
Loop
Equations.Close
GenVar.Close
End Function

Public Function CheckTunnelTimes(DB As Database, Messages As Recordset)
    Dim Tunnels As Recordset, ObsDelays As Recordset, CurrentTime As Double
    CurrentTime = DLookup("Current", "tTime")
    CheckTunnelTimes = True
    Set Tunnels = DB.OpenRecordset("tTunnels", dbOpenTable, dbReadOnly)
    If Tunnels.EOF And Tunnels.BOF Then Exit Function
    Set ObsDelays = DB.OpenRecordset("oqTunDelays", dbOpenDynaset, dbReadOnly)
    Tunnels.MoveFirst
    Do Until Tunnels.EOF
        If Tunnels!ObsBeginTime < 0 Then
            ObsDelays.FindFirst "Tunnel = " & CStr(Tunnels!ID)
            If Not ObsDelays.NoMatch Then
                AddMessage Messages, "The observed construction start time of tunnel '" & _
                    CStr(Tunnels!Name) & "' is not specified although there are" & _
                    " observed delays."
                CheckTunnelTimes = False
            End If
        End If
        If Tunnels!ObsBeginTime > CurrentTime Then
            AddMessage Messages, "The observed construction start time of tunnel '" & _
                CStr(Tunnels!Name) & "' is in the future (after current time)."
            CheckTunnelTimes = False
        End If
        ObsDelays.FindFirst "Tunnel = " & CStr(Tunnels!ID) & " And Time + " & _
            CStr(Tunnels!ObsBeginTime) & " > " & CStr(CurrentTime)
        If Not ObsDelays.NoMatch Then
            AddMessage Messages, "There is an observed delay in tunnel '" & _
                CStr(Tunnels!Name) & "' which starts in the future (delay start " & _
                " + tunnel start > current time)."
            CheckTunnelTimes = False
        End If
        Tunnels.MoveNext
    Loop
    ObsDelays.Close
    Tunnels.Close
End Function

Public Sub AddMessage(Messages As Recordset, ByVal Text As String)
    Messages.addnew
    Messages!Message = Text
    Messages.Update
End Sub
```

Module: Export

```
Option Compare Database
Option Explicit
Dim DB As Database, FS As Object, RS As Recordset, MaxZone As Long
Dim AreaID() As Long, MaxArea As Integer, GCID() As Long, MaxGC As Integer
Dim ParamSetID() As Long, MaxParamSet As Integer, ZoneParamSet() As Long
Dim ImpExp1 As Recordset, ImpExp2 As Recordset, Path As String, UpdatePhase As Long

Public Sub Export(UP As Long)
    Dim Title As String
    UpdatePhase = UP
    Title = "EXPORT SIMULATOR INPUT DATA"
    Set DB = CurrentDb()
    Set FS = CreateObject("Scripting.FileSystemObject")
    Set RS = DB.OpenRecordset("tUpdatePhases", dbOpenTable)
    RS.Index = "PrimaryKey"
    RS.Seek "=", UP
    If IsNull(RS!Path) Then
        RS.MoveFirst
        If IsNull(RS!Path) Then
            RS.Edit
            RS!Path = " "
            RS.Update
        End If
    End If
    Do Until False
        Path = ""
        Path = InputBox("Export data to the following directory:", _
            Title, RS!Path)
        If Path = "" Then Exit Sub
        If Not FS.FolderExists(Path) Then
            MsgBox "The directory you have chosen does not exist.", vbOKOnly, Title
        Else
            Exit Do
        End If
    Loop
    RS.Seek "=", UP
    RS.Edit
    RS!Path = Path
    RS.Update
    RS.Close
    DoCmd.OpenForm "BePatient", acNormal
    DoCmd.RepaintObject acForm, "BePatient"
    DoCmd.Hourglass True
    DoCmd.Echo False, "Working hard...."
    DoCmd.SetWarnings False

    If UpdatePhase > 1 Then
        CreateTemporaryObjects
        CreateTemporaryDelays
    End If
    ExpGroundParam
    ExpConstrParam
    If UpdatePhase > 1 Then DeleteTemporaryObjects

    DB.Close
    DoCmd.SetWarnings True
    DoCmd.Echo True
    DoCmd.Hourglass False
    DoCmd.Close acForm, "BePatient", acSaveNo
End Sub

Private Sub ExpGroundParam()
    Dim ParamStatID() As Long, ParamID() As Long
    Dim MaxParam As Integer, MaxParamStat As Integer, Param As Recordset
    Set Param = DB.OpenRecordset("eParamStatCount", dbOpenSnapshot)
```



```
If Param.EOF And Param.BOF Then Exit Sub
Param.MoveLast
MaxParam = Param.RecordCount
MaxParamStat = 0
Param.MoveFirst
Do Until Param.EOF
    If Param!ParamStatNo > MaxParamStat Then MaxParamStat = Param!ParamStatNo
    Param.MoveNext
Loop
If MaxParamStat = 0 Then Exit Sub
ReDim ParamID(MaxParam - 1), ParamStatID(MaxParam - 1, MaxParamStat - 1)
Param.Close

ExpParameters ParamID, ParamStatID, MaxParam
ExpParamSets ParamID, ParamStatID, MaxParam, MaxParamStat
ExpGC ParamID, ParamStatID, MaxParam, MaxParamStat
ExpCorrel ParamID, ParamStatID, MaxParam, MaxParamStat
ExpBoreholes ParamID, ParamStatID, MaxParam
ExpCombGC
ExpAreaZones
End Sub

Private Sub ExpConstrParam()
    Dim ResID() As Long, MaxRes As Integer, RepID() As Long, MaxRep As Integer
    Dim GeomID() As Long, MaxGeom As Integer, MethID() As Long, MaxMeth As Integer
    Dim ActID() As Long, MaxAct As Integer

    ExpGeometries GeomID, MaxGeom
    ExpResources ResID, MaxRes
    ExpRepositories RepID, MaxRep, ResID, MaxRes
    ExpTunnels RepID, MaxRep, GeomID, MaxGeom
    ExpActivities ActID, MaxAct
    ExpActResDemands ResID, MaxRes
    ExpMethods MethID, MaxMeth, ActID, MaxAct
    ExpChangeDelays MethID, MaxMeth
    ExpGCGeomMeth MethID, MaxMeth, GeomID, MaxGeom
    ExpGenVariables
End Sub

Private Sub ExpParameters(ParamID() As Long, ParamStatID() As Long, MaxParam As Integer)
    Dim n As Integer, m As Integer, ParamStatNo As Integer

    Set RS = DB.OpenRecordset("eParamStates", dbOpenDynaset, dbReadOnly)
    DB.Execute ("ImpExplDel")
    Set ImpExpl = DB.OpenRecordset("ImpExpl", dbOpenTable, dbAppendOnly)
    ImpExpl.Index = "PrimaryKey"
    AddLine ImpExpl, MaxParam
    RS.MoveFirst
    For n = 0 To MaxParam - 1
        ParamID(n) = RS!ParamID
        AddLine ImpExpl, RS!ParamName
        AddLine ImpExpl, RS!ParamStatNo
        ParamStatNo = RS!ParamStatNo
        For m = 0 To ParamStatNo - 1
            ParamStatID(n, m) = RS!ParamStatID
            AddLine ImpExpl, RS!ParamStatName
            RS.MoveNext
        Next m
    Next n
    RS.Close
    ImpExpl.Close
    DoCmd.TransferText acExportDelim, _
        "Text ExpSpec", "Expl", Path & "\param_stat_save", 0
End Sub

Private Sub ExpParamSets(ParamID() As Long, ParamStatID() As Long, _
    MaxParam As Integer, MaxParamStat As Integer)
```

```
Dim n As Integer, m As Integer, l As Integer, ParamSet As Long, SegNo As Integer
Dim Param As Integer, ParamStatNo As Integer, Criterion As String
Dim MarkovLengths As Recordset, MarkovTransProb As Recordset
Dim SemiDeter As Recordset, DeterSegs As Recordset
Set RS = DB.OpenRecordset("eParamSetParam", dbOpenDynaset, dbReadOnly)
Set MarkovLengths = DB.OpenRecordset("tMarkovLengths", dbOpenTable, dbReadOnly)
Set MarkovTransProb = DB.OpenRecordset("tMarkovTransProb", dbOpenTable, dbReadOnly)
Set DeterSegs = DB.OpenRecordset("eDeterSegs", dbOpenDynaset, dbReadOnly)
Set SemiDeter = DB.OpenRecordset("eSemiDeterSegs", dbOpenDynaset, dbReadOnly)
MarkovLengths.Index = "PrimaryKey"
MarkovTransProb.Index = "PrimaryKey"
DB.Execute ("ImpExp1Del")
Set ImpExp1 = DB.OpenRecordset("ImpExp1", dbOpenTable, dbAppendOnly)
ImpExp1.Index = "PrimaryKey"
DB.Execute ("ImpExp2Del")
Set ImpExp2 = DB.OpenRecordset("ImpExp2", dbOpenTable, dbAppendOnly)
ImpExp2.Index = "PrimaryKey"

If RS.BOF And RS.EOF Then
    MaxParamSet = 0
    ReDim ParamSetID(0) As Long
    AddLine ImpExp1, 0
Else
    ParamSet = -1
    MaxParamSet = 0
    RS.MoveFirst
    Do Until RS.EOF ' count number of param sets
        If RS!ParamSet <> ParamSet Then
            ParamSet = RS!ParamSet
            MaxParamSet = MaxParamSet + 1
        End If
        RS.MoveNext
    Loop
    AddLine ImpExp1, MaxParamSet ' number of parameter sets
    ReDim ParamSetID(MaxParamSet - 1) As Long
    RS.MoveFirst
    ParamSet = -1
    l = 0
    Do Until RS.EOF
        If ParamSet <> RS!ParamSet Then
            AddLine ImpExp1, RS!ParamNo ' number of parameters in parameter set
            ParamSet = RS!ParamSet
            ParamSetID(l) = RS!ParamSet
            l = l + 1
        End If
        If RS!ParamNo > 0 Then
            Param = GiveArrayPosID(ParamID, MaxParam, RS!ParamID) ' parameter ID
            AddLine ImpExp1, Param
            AddLine ImpExp1, RS!ParamStatNo ' number of parameter states of parameter
            ParamStatNo = RS!ParamStatNo
            AddLine ImpExp1, RS!GenerationMode ' generation mode
            For n = 0 To ParamStatNo - 1
                MarkovLengths.Seek "=", ParamSet, ParamStatID(Param, n), UpdatePhase
                If MarkovLengths.NoMatch Then
                    AddLine ImpExp1, 0
                    AddLine ImpExp1, 0
                    AddLine ImpExp1, 0
                Else
                    AddLine ImpExp1, MarkovLengths!LengthMin
                    AddLine ImpExp1, MarkovLengths!LengthMode
                    AddLine ImpExp1, MarkovLengths!LengthMax
                End If
            Next n
            For m = 0 To ParamStatNo - 1
                MarkovTransProb.Seek "=", ParamSet, ParamStatID(Param, n), _
                    ParamStatID(Param, m), UpdatePhase
                If MarkovTransProb.NoMatch Then
```

```
        AddLine ImpExp1, 0
    Else
        AddLine ImpExp1, MarkovTransProb!Prob
    End If
Next m
Next n
For n = 0 To ParamStatNo - 1
    MarkovLengths.Seek "=", ParamSet, ParamStatID(Param, n), UpdatePhase
    If MarkovLengths.NoMatch Then
        AddLine ImpExp1, 0
    Else
        AddLine ImpExp1, MarkovLengths!StartProb
    End If
Next n
AddLine ImpExp1, RS!StartProbCorr
AddLine ImpExp1, RS!DeterReference
DeterSegs.FindFirst "ParamSet = " & CStr(ParamSet) & " And ParamID = " & _
    CStr(RS!ParamID) & " And UpdatePhase = " & CStr(UpdatePhase)
If DeterSegs.NoMatch Then
    AddLine ImpExp1, 0
Else
    n = 0
    Do Until DeterSegs.NoMatch ' count number of segments
        n = n + 1
        DeterSegs.FindNext "ParamSet = " & CStr(ParamSet) & " And ParamID = " & _
            CStr(RS!ParamID) & " And UpdatePhase = " & CStr(UpdatePhase)
    Loop
    AddLine ImpExp1, n
    DeterSegs.FindFirst "ParamSet = " & CStr(ParamSet) & " And ParamID = " & _
        CStr(RS!ParamID) & " And UpdatePhase = " & CStr(UpdatePhase)
    Do Until DeterSegs.NoMatch
        AddLine ImpExp1, GiveArrayPos2DM(ParamStatID(), MaxParam, MaxParamStat, _
            DeterSegs!ParamState)
        AddLine ImpExp1, DeterSegs!EndPos
        DeterSegs.FindNext "ParamSet = " & CStr(ParamSet) & " And ParamID = " & _
            CStr(RS!ParamID) & " And UpdatePhase = " & CStr(UpdatePhase)
    Loop
End If
Criterion = "UpdatePhase = " & CStr(UpdatePhase) & " And ParamSet = " & _
    CStr(RS!ParamSet) & " And Parameter = " & CStr(RS!ParamID)
SegNo = 0
SemiDeter.FindFirst Criterion
Do Until SemiDeter.NoMatch
    SegNo = SegNo + 1
    SemiDeter.FindNext Criterion
Loop
AddLine ImpExp2, RS!DeterReference
AddLine ImpExp2, SegNo
SemiDeter.FindFirst Criterion
Do Until SemiDeter.NoMatch
    AddLine ImpExp2, GiveArrayPos2DM(ParamStatID, MaxParam, _
        MaxParamStat, SemiDeter!ParamState)
    AddLine ImpExp2, SemiDeter!GenerationMode
    If SemiDeter!GenerationMode = "0" Then
        AddLine ImpExp2, SemiDeter!LengthMin
        AddLine ImpExp2, SemiDeter!LengthMode
        AddLine ImpExp2, SemiDeter!LengthMax
        AddLine ImpExp2, SemiDeter!LengthProbMin
        AddLine ImpExp2, SemiDeter!LengthProbMax
    Else
        AddLine ImpExp2, SemiDeter!EndposMin
        AddLine ImpExp2, SemiDeter!EndposMode
        AddLine ImpExp2, SemiDeter!EndposMax
        AddLine ImpExp2, SemiDeter!EndposProbMin
        AddLine ImpExp2, SemiDeter!EndposProbMax
    End If
    SemiDeter.FindNext Criterion
Loop
```

```
        End If
        RS.MoveNext
    Loop
End If
DeterSegs.Close
SemiDeter.Close
MarkovLengths.Close
MarkovTransProb.Close
RS.Close
ImpExp1.Close
DoCmd.TransferText acExportDelim, _
    "Text ExpSpec", "Exp1", Path & "\ground_save", 0
ImpExp2.Close
DoCmd.TransferText acExportDelim, _
    "Text ExpSpec", "Exp2", Path & "\deter_save", 0
End Sub

Private Sub ExpGC(ParamID() As Long, ParamStatID() As Long, _
    MaxParam As Integer, MaxParamStat As Integer)
    Dim ParamSets As Recordset, ParamSet As Long, n As Integer, BM As Variant

    Set RS = DB.OpenRecordset("eGroundClasses", dbOpenDynaset, dbReadOnly)
    Set ParamSets = DB.OpenRecordset("eGroundClassCount", dbOpenDynaset, dbReadOnly)
    DB.Execute ("ImpExp1Del")
    Set ImpExp1 = DB.OpenRecordset("ImpExp1", dbOpenTable, dbAppendOnly)
    ImpExp1.Index = "PrimaryKey"
    ParamSets.MoveFirst
    Do Until ParamSets.EOF
        AddLine ImpExp1, ParamSets!GCNo ' number of GC's in parameter set
        ParamSet = ParamSets!ParamSet
        RS.FindFirst "ParamSet = " & CStr(ParamSets!ParamSet)
        Do Until RS.NoMatch
            AddLine ImpExp1, RS!GCName
            BM = ParamSets.Bookmark
            Do Until ParamSets!ParamSet <> ParamSet
                If IsNull(RS(CStr(ParamSets!ParamID))) Then
                    AddLine ImpExp1, -1
                Else
                    AddLine ImpExp1, GiveArrayPos2DM(ParamStatID, MaxParam, MaxParamStat, _
                        RS(CStr(ParamSets!ParamID)))
                End If
                ParamSets.MoveNext
            If ParamSets.EOF Then Exit Do
        Loop
        ParamSets.Bookmark = BM
        RS.FindNext "ParamSet = " & CStr(ParamSets!ParamSet)
    Loop
    If ParamSets!ParamNo = 0 Then
        ParamSets.MoveNext
    Else
        ParamSets.Move (ParamSets!ParamNo)
    End If
Loop
RS.Close
ParamSets.Close
ImpExp1.Close
DoCmd.TransferText acExportDelim, _
    "Text ExpSpec", "Exp1", Path & "\gc_save", 0

Set RS = DB.OpenRecordset("eKeywords", dbOpenDynaset, dbReadOnly)
DB.Execute ("ImpExp1Del")
Set ImpExp1 = DB.OpenRecordset("ImpExp1", dbOpenTable, dbAppendOnly)
ImpExp1.Index = "PrimaryKey"
If RS.BOF And RS.EOF Then
    AddLine ImpExp1, 0
    ReDim GCID(0) As Long
Else
    RS.MoveLast
```

```
MaxGC = RS.RecordCount
AddLine ImpExpl, MaxGC
ReDim GCID(MaxGC - 1) As Long
RS.MoveFirst
For n = 0 To MaxGC - 1
    GCID(n) = RS!ID
    AddLine ImpExpl, RS!Name
    AddLine ImpExpl, RS!CombNo
    RS.MoveNext
Next n
End If
RS.Close
ImpExpl.Close
DoCmd.TransferText acExportDelim, _
    "Text ExpSpec", "Exp1", Path & "\gc_keyword_save", 0
End Sub

Private Sub ExpCorrel(ParamID() As Long, ParamStatID() As Long, _
    MaxParam As Integer, MaxParamStat As Integer)
    Dim ParamSets As Recordset, ParamSet As Long, BM As Variant

    Set RS = DB.OpenRecordset("eCorrelations", dbOpenDynaset, dbReadOnly)
    Set ParamSets = DB.OpenRecordset("eCorrelationsCount", dbOpenDynaset, dbReadOnly)
    DB.Execute ("ImpExplDel")
    Set ImpExpl = DB.OpenRecordset("ImpExpl", dbOpenTable, dbAppendOnly)
    ImpExpl.Index = "PrimaryKey"
    ParamSets.MoveFirst
    Do Until ParamSets.EOF
        AddLine ImpExpl, ParamSets!CorrelNo ' number of correlations in parameter set
        ParamSet = ParamSets!ParamSet
        RS.FindFirst "ParamSet = " & CStr(ParamSets!ParamSet)
        Do Until RS.NoMatch
            AddLine ImpExpl, RS!Value
            BM = ParamSets.Bookmark
            Do Until ParamSets!ParamSet <> ParamSet
                If IsNull(RS(CStr(ParamSets!ParamID))) Then
                    AddLine ImpExpl, 0
                Else
                    AddLine ImpExpl, GiveArrayPos2DM(ParamStatID, MaxParam, MaxParamStat, _
                        RS(CStr(ParamSets!ParamID)))
                End If
                ParamSets.MoveNext
            If ParamSets.EOF Then Exit Do
        Loop
        ParamSets.Bookmark = BM
        RS.FindNext "ParamSet = " & CStr(ParamSets!ParamSet)
    Loop
    If ParamSets!ParamNo = 0 Then
        ParamSets.MoveNext
    Else
        ParamSets.Move (ParamSets!ParamNo)
    End If
    Loop
    RS.Close
    ParamSets.Close
    ImpExpl.Close
    DoCmd.TransferText acExportDelim, _
        "Text ExpSpec", "Exp1", Path & "\gc_proba_save", 0
End Sub

Private Sub ExpBoreholes(ParamID() As Long, ParamStatID() As Long, MaxParam As Integer)
    Dim Parameters As Recordset, ParamSet As Long, Param As Integer, n As Integer
    Dim ParamStatNo As Integer, BM As Variant

    Set RS = DB.OpenRecordset("eBoreholes", dbOpenDynaset, dbReadOnly)
    Set Parameters = DB.OpenRecordset("eParamSetParam", dbOpenDynaset, dbReadOnly)
    DB.Execute ("ImpExplDel")
```

```
Set ImpExp1 = DB.OpenRecordset("ImpExp1", dbOpenTable, dbAppendOnly)
ImpExp1.Index = "PrimaryKey"
ParamSet = -1
Param = 0 ' temporary abuse of variable
Parameters.MoveFirst
Do Until Parameters.EOF ' count number of param sets
    If Parameters!ParamSet <> ParamSet Then
        ParamSet = Parameters!ParamSet
        Param = Param + 1
    End If
    Parameters.MoveNext
Loop
AddLine ImpExp1, Param ' number of parameter sets
Parameters.MoveFirst
Do Until Parameters.EOF
    RS.FindFirst "ParamSet = " & CStr(Parameters!ParamSet)
    If RS.NoMatch Then
        AddLine ImpExp1, 0
    Else
        AddLine ImpExp1, RS!BoreholeNo ' number of boreholes in parameter set
        ParamSet = Parameters!ParamSet
        BM = Parameters.Bookmark
        Do Until RS.NoMatch
            AddLine ImpExp1, RS!Name
            AddLine ImpExp1, RS!Pos
            If RS!Likelihood Then
                AddLine ImpExp1, 1
            Else
                AddLine ImpExp1, 0
            End If
            AddLine ImpExp1, Parameters!ParamNo
            Do Until Parameters!ParamSet <> ParamSet
                AddLine ImpExp1, Parameters!ParamStatNo
                Param = GiveArrayPosID(ParamID, MaxParam, Parameters!ParamID)
                ParamStatNo = Parameters!ParamStatNo
                For n = 0 To ParamStatNo - 1
                    On Error Resume Next
                    If IsNull(RS(CStr(ParamStatID(Param, n)))) Then
                        AddLine ImpExp1, 0
                    Else
                        AddLine ImpExp1, RS(CStr(ParamStatID(Param, n)))
                    End If
                    On Error GoTo 0
                Next n
                Parameters.MoveNext
            If Parameters.EOF Then Exit Do
        Loop
        Parameters.Bookmark = BM
        RS.FindNext "ParamSet = " & CStr(Parameters!ParamSet)
    Loop
End If
If Parameters!ParamNo = 0 Then
    Parameters.MoveNext
Else
    Parameters.Move (Parameters!ParamNo)
End If
Loop
RS.Close
Parameters.Close
ImpExp1.Close
DoCmd.TransferText acExportDelim, _
    "Text ExpSpec", "Exp1", Path & "\bh_save", 0
End Sub

Private Sub ExpCombGC()
    Set RS = DB.OpenRecordset("qCombGC", dbOpenDynaset, dbReadOnly)
    DB.Execute ("ImpExp1Del")
    Set ImpExp1 = DB.OpenRecordset("ImpExp1", dbOpenTable, dbAppendOnly)
```

```
ImpExp1.Index = "PrimaryKey"
If RS.EOF And RS.BOF Then
    AddLine ImpExp1, 0
Else
    RS.MoveLast
    AddLine ImpExp1, RS.RecordCount
    Do Until RS.BOF
        AddLine ImpExp1, RS!CombGCName
        If IsNull(RS!AreaGC) Then
            AddLine ImpExp1, ""
        Else
            AddLine ImpExp1, RS!AreaGCName
        End If
        If IsNull(RS!ZoneGC) Then
            AddLine ImpExp1, ""
        Else
            AddLine ImpExp1, RS!ZoneGCName
        End If
        RS.MovePrevious
    Loop
End If
RS.Close
ImpExp1.Close
DoCmd.TransferText acExportDelim, _
    "Text ExpSpec", "Exp1", Path & "\cb_save", 0
End Sub

Private Sub ExpAreaZones()
    Dim Area As Long, AreaNo As Integer, ZoneNo As Integer
    Set RS = DB.OpenRecordset("eAreaZones", dbOpenDynaset, dbReadOnly)
    DB.Execute ("ImpExp1Del")
    Set ImpExp1 = DB.OpenRecordset("ImpExp1", dbOpenTable, dbAppendOnly)
    ImpExp1.Index = "PrimaryKey"
    DB.Execute ("ImpExp2Del")
    Set ImpExp2 = DB.OpenRecordset("ImpExp2", dbOpenTable, dbAppendOnly)
    ImpExp2.Index = "PrimaryKey"
    If RS.EOF And RS.BOF Then
        ReDim AreaID(0) As Long
        AddLine ImpExp1, 0
        AddLine ImpExp2, 0
    Else
        Area = -1
        AreaNo = 0 ' temporary abuse of variable
        ZoneNo = 0 ' temporary abuse of variable
        RS.FindFirst "UpdatePhase = " & CStr(UpdatePhase)
        Do Until RS.NoMatch ' count number of areas
            If RS!AreaID <> Area Then
                Area = RS!AreaID
                AreaNo = AreaNo + 1
            End If
            ZoneNo = ZoneNo + 1
            RS.FindNext "UpdatePhase = " & CStr(UpdatePhase)
        Loop
        AddLine ImpExp1, AreaNo ' no of areas
        AddLine ImpExp2, ZoneNo ' no of zones
        MaxArea = AreaNo
        ReDim AreaID(MaxArea - 1)
        RS.FindFirst "UpdatePhase = " & CStr(UpdatePhase)
        Area = RS!AreaID
        AreaNo = 0
        ZoneNo = 0
        AreaID(AreaNo) = RS!AreaID
        AddLine ImpExp1, RS!AreaName
        AddLine ImpExp1, AreaNo
        AddLine ImpExp1, ZoneNo
        Do Until RS.NoMatch
            If RS!AreaID <> Area Then
                Area = RS!AreaID
```

```
AddLine ImpExp1, ZoneNo - 1 ' last zone
RS.FindPrevious "UpdatePhase = " & CStr(UpdatePhase)
AddLine ImpExp1, RS!AreaLength
If IsNull(RS!AreaParamSet) Then
    AddLine ImpExp1, -1
Else
    AddLine ImpExp1, GiveArrayPos1D(ParamSetID, MaxParamSet, _
        RS!AreaParamSet)
End If
RS.FindNext "UpdatePhase = " & CStr(UpdatePhase)
AreaNo = AreaNo + 1
AreaID(AreaNo) = RS!AreaID
AddLine ImpExp1, RS!AreaName
AddLine ImpExp1, AreaNo ' area index
AddLine ImpExp1, ZoneNo ' first zone
End If
AddLine ImpExp2, RS!ZoneName
AddLine ImpExp2, GiveArrayPos1D(ParamSetID, MaxParamSet, RS!ZoneParamSet)
AddLine ImpExp2, RS!GenerationMode
AddLine ImpExp2, RS!LengthMin
AddLine ImpExp2, RS!LengthMode
AddLine ImpExp2, RS!LengthMax
AddLine ImpExp2, RS!LengthProbMin
AddLine ImpExp2, RS!LengthProbMax
AddLine ImpExp2, RS!EndposMin
AddLine ImpExp2, RS!EndposMode
AddLine ImpExp2, RS!EndposMax
AddLine ImpExp2, RS!EndposProbMin
AddLine ImpExp2, RS!EndposProbMax
RS.FindNext "UpdatePhase = " & CStr(UpdatePhase)
ZoneNo = ZoneNo + 1
Loop
RS.FindLast "UpdatePhase = " & CStr(UpdatePhase)
AddLine ImpExp1, ZoneNo - 1 ' last zone
AddLine ImpExp1, RS!AreaLength
If IsNull(RS!AreaParamSet) Then
    AddLine ImpExp1, -1
Else
    AddLine ImpExp1, GiveArrayPos1D(ParamSetID, MaxParamSet, RS!AreaParamSet)
End If
End If
RS.Close

ImpExp1.Close
DoCmd.TransferText acExportDelim, _
    "Text ExpSpec", "Exp1", Path & "\area_save", 0
ImpExp2.Close
DoCmd.TransferText acExportDelim, _
    "Text ExpSpec", "Exp2", Path & "\zone_save", 0
End Sub

Private Sub ExpGeometries(GeomID() As Long, MaxGeom As Integer)
    Dim n As Integer
    Set RS = DB.OpenRecordset("tGeometries", dbOpenTable, dbReadOnly)
    RS.Index = "PrimaryKey"
    DB.Execute ("ImpExp1Del")
    Set ImpExp1 = DB.OpenRecordset("ImpExp1", dbOpenTable, dbAppendOnly)
    ImpExp1.Index = "PrimaryKey"
    MaxGeom = RS.RecordCount
    AddLine ImpExp1, MaxGeom
    If MaxGeom > 0 Then
        ReDim GeomID(MaxGeom - 1) As Long
        n = 0
        RS.MoveFirst
        Do Until RS.EOF
            GeomID(n) = RS!ID
            AddLine ImpExp1, RS!Value1
            AddLine ImpExp1, RS!Value2
        Loop
    End If
End Sub
```



```
        RS.MoveNext
        n = n + 1
    Loop
Else
    ReDim GeomID(0) As Long
End If
RS.Close
ImpExp1.Close
DoCmd.TransferText acExportDelim, _
    "Text ExpSpec", "Exp1", Path & "\geom_save", 0
End Sub

Private Sub ExpResources(ResID() As Long, MaxRes As Integer)
    Dim n As Integer

    Set RS = DB.OpenRecordset("tResources", dbOpenTable, dbReadOnly)
    RS.Index = "Name"
    DB.Execute ("ImpExp1Del")
    Set ImpExp1 = DB.OpenRecordset("ImpExp1", dbOpenTable, dbAppendOnly)
    ImpExp1.Index = "PrimaryKey"
    If RS.EOF And RS.BOF Then
        AddLine ImpExp1, 0
        MaxRes = 0
        ReDim ResID(0)
    Else
        MaxRes = RS.RecordCount
        ReDim ResID(MaxRes - 1)
        AddLine ImpExp1, MaxRes
        RS.MoveFirst
        For n = 0 To MaxRes - 1
            ResID(n) = RS!ID
            AddLine ImpExp1, RS!Name
            AddLine ImpExp1, 0 ' category currently not supported --> write 0
            RS.MoveNext
        Next n
    End If
    RS.Close
    ImpExp1.Close
    DoCmd.TransferText acExportDelim, _
        "Text ExpSpec", "Exp1", Path & "\res_name_save", 0
End Sub

Private Sub ExpRepositories(RepID() As Long, MaxRep As Integer, _
    ResID() As Long, MaxRes As Integer)
    Dim n As Integer, m As Integer, Stocks As Recordset

    Set RS = DB.OpenRecordset("tRepositories", dbOpenTable, dbReadOnly)
    RS.Index = "Name"
    Set Stocks = DB.OpenRecordset("tStocks", dbOpenTable, dbReadOnly)
    Stocks.Index = "Repository"
    DB.Execute ("ImpExp1Del")
    Set ImpExp1 = DB.OpenRecordset("ImpExp1", dbOpenTable, dbAppendOnly)
    ImpExp1.Index = "PrimaryKey"
    If RS.EOF And RS.BOF Then
        AddLine ImpExp1, 0
        MaxRep = 0
        ReDim RepID(0)
    Else
        MaxRep = RS.RecordCount
        ReDim RepID(MaxRep - 1)
        AddLine ImpExp1, MaxRep
        RS.MoveFirst
        For n = 0 To MaxRep - 1
            DB.Execute ("ImpExp2Del")
            Set ImpExp2 = DB.OpenRecordset("ImpExp2", dbOpenTable, dbAppendOnly)
            ImpExp2.Index = "PrimaryKey"
            Stocks.Seek "=", RS!ID, UpdatePhase
```

```
If Stocks.NoMatch Then
    AddLine ImpExp2, 0
Else
    m = 0
    Do Until Stocks!Repository <> RS!ID Or Stocks!UpdatePhase <> UpdatePhase
        Stocks.MoveNext
        m = m + 1
        If Stocks.EOF Then Exit Do
    Loop
    AddLine ImpExp2, m
    Stocks.Seek "=", RS!ID, UpdatePhase
    Do Until Stocks!Repository <> RS!ID Or Stocks!UpdatePhase <> UpdatePhase
        AddLine ImpExp2, GiveArrayPosID(ResID, MaxRes, Stocks!Resource)
        AddLine ImpExp2, Stocks!InitLevel
        AddLine ImpExp2, Stocks!Capacity
        Stocks.MoveNext
        If Stocks.EOF Then Exit Do
    Loop
End If
ImpExp2.Close
DoCmd.TransferText acExportDelim, "Text ExpSpec", "Exp2", _
    Path & "\stock_reposit_save_" & CStr(n + 1), 0

RepID(n) = RS!ID
AddLine ImpExp1, RS!Name
RS.MoveNext
Next n
End If
RS.Close
ImpExp1.Close
DoCmd.TransferText acExportDelim, _
    "Text ExpSpec", "Exp1", Path & "\reposit_save", 0
End Sub

Private Sub ExpTunnels(RepID() As Long, MaxRep As Integer, _
    GeomID() As Long, MaxGeom As Integer)
    Dim Rep As Recordset, UpDown As Recordset, Delays As Recordset
    Dim TunnelID() As Long, MaxTunnel As Integer, n As Integer, DataSetNo As Integer
    Dim NodeID() As Long, MaxNode As Integer, Nodes As Recordset, Arcs As Recordset
    Set RS = DB.OpenRecordset("eTunnels", dbOpenDynaset, dbReadOnly)
    Set Rep = DB.OpenRecordset("pTunnelRepositories", dbOpenDynaset, dbReadOnly)
    Set UpDown = DB.OpenRecordset("pTunnelUpDown", dbOpenDynaset, dbReadOnly)
    Set Delays = DB.OpenRecordset("eTunnelDelays", dbOpenDynaset, dbReadOnly)
    Set Arcs = DB.OpenRecordset("tTunnelArcs", dbOpenTable, dbReadOnly)
    Arcs.Index = "Tunnel"
    MaxTunnel = Arcs.RecordCount ' number of tunnels = number of arcs
    If MaxTunnel > 0 Then
        ReDim TunnelID(MaxTunnel - 1) As Long
        RS.MoveFirst
        n = 0
        Do Until RS.EOF
            Arcs.Seek "=", RS!ID ' do not save tunnels without arcs
            If Not Arcs.NoMatch Then
                TunnelID(n) = RS!ID
                n = n + 1
            End If
            RS.MoveNext
            If RS.EOF Then Exit Do
        Loop
        RS.MoveFirst
        n = 0
        Do Until RS.EOF
            Arcs.Seek "=", RS!ID ' do not save tunnels without arcs
            If Not Arcs.NoMatch Then
                DB.Execute ("ImpExp1Del")
                Set ImpExp1 = DB.OpenRecordset("ImpExp1", dbOpenTable, dbAppendOnly)
                ImpExp1.Index = "PrimaryKey"
                AddLine ImpExp1, RS!Name
            End If
        Loop
    End If
End Sub
```

```
AddLine ImpExpl, RS!Type
If IsNull(RS!OppTunnel) Then
    AddLine ImpExpl, -1
Else
    AddLine ImpExpl, GiveArrayPos1D(TunnelID, MaxTunnel, RS!OppTunnel)
End If
AddLine ImpExpl, GiveArrayPos1D(AreaID, MaxArea, RS!Area)
AddLine ImpExpl, GiveArrayPos1D(GeomID, MaxGeom, RS!Geometry)
If UpdatePhase = 1 Then
    AddLine ImpExpl, RS!BeginPos
    AddLine ImpExpl, RS!EndPos
    AddLine ImpExpl, RS!FixCostMin
    AddLine ImpExpl, RS!FixCostMode
    AddLine ImpExpl, RS!FixCostMax
Else
    If RS!UpdatedBeginPos = RS!UpdatedBeginPos Then
        If RS!BeginPos < RS!EndPos Then
            AddLine ImpExpl, RS!UpdatedBeginPos - 0.0001
        Else
            AddLine ImpExpl, RS!UpdatedBeginPos + 0.0001
        End If
    Else
        AddLine ImpExpl, RS!UpdatedBeginPos
    End If
    AddLine ImpExpl, RS!UpdatedEndPos
    AddLine ImpExpl, RS!UpdatedCostMin
    AddLine ImpExpl, RS!UpdatedCostMode
    AddLine ImpExpl, RS!UpdatedCostMax
End If
AddLine ImpExpl, RS!TimeCostMin
AddLine ImpExpl, RS!TimeCostMode
AddLine ImpExpl, RS!TimeCostMax
AddLine ImpExpl, RS!RepNo + 1
Rep.FindFirst "Tunnel = " & CStr(RS!ID)
Do Until Rep.NoMatch
    AddLine ImpExpl, GiveArrayPos1D(RepID, MaxRep, Rep!Repository)
    If UpdatePhase > 1 Then ' correct position for new tunnel beginpos
        AddLine ImpExpl, Auxiliary.MaxDbl(0, Rep!UpToPosition - _
            Abs(RS!BeginPos - RS!UpdatedBeginPos))
    Else
        AddLine ImpExpl, Rep!UpToPosition
    End If
    Rep.FindNext "Tunnel = " & CStr(RS!ID)
Loop
AddLine ImpExpl, -1
AddLine ImpExpl, 1000000001
DataSetNo = 0 ' count number of updown-changes
UpDown.FindFirst "Tunnel = " & CStr(RS!ID)
Do Until UpDown.NoMatch
    DataSetNo = DataSetNo + 1
    UpDown.FindNext "Tunnel = " & CStr(RS!ID)
Loop
AddLine ImpExpl, DataSetNo
UpDown.FindFirst "Tunnel = " & CStr(RS!ID)
Do Until UpDown.NoMatch
    If UpdatePhase > 1 Then
        AddLine ImpExpl, Auxiliary.MaxDbl(0, UpDown!Position - _
            Abs(RS!BeginPos - RS!UpdatedBeginPos))
    Else
        AddLine ImpExpl, UpDown!Position
    End If
    UpDown.FindNext "Tunnel = " & CStr(RS!ID)
Loop
Delays.FindFirst "Tunnel = " & CStr(RS!ID) & " And UpdatePhase = " & _
    CStr(UpdatePhase) & " And Type = 0"
DataSetNo = 0 ' count time-dependent delays
Do Until Delays.NoMatch
    DataSetNo = DataSetNo + 1
```

```
Delays.FindNext "Tunnel = " & CStr(RS!ID) & " And UpdatePhase = " & _
    CStr(UpdatePhase) & " And Type = 0"
Loop
AddLine ImpExp1, DataSetNo + 1
Delays.FindFirst "Tunnel = " & CStr(RS!ID) & " And UpdatePhase = " & _
    CStr(UpdatePhase) & " And Type = 0"
Do Until Delays.NoMatch
    If UpdatePhase > 1 And Delays!Status = "Posterior" And _
        RS!ObsBeginTime >= 0 Then
        ' correct start time for current time
        AddLine ImpExp1, Delays!Start - _
            (DLookup("Current", "tTime") - RS!ObsBeginTime)
    Else
        AddLine ImpExp1, Delays!Start
    End If
    AddLine ImpExp1, Delays!DurationMin
    AddLine ImpExp1, Delays!DurationMode
    AddLine ImpExp1, Delays!DurationMax
    Delays.FindNext "Tunnel = " & CStr(RS!ID) & " And UpdatePhase = " & _
        CStr(UpdatePhase) & " And Type = 0"
Loop
AddLine ImpExp1, 1000000001#
AddLine ImpExp1, -1
AddLine ImpExp1, -1
AddLine ImpExp1, -1
Delays.FindFirst "Tunnel = " & CStr(RS!ID) & " And UpdatePhase = " & _
    CStr(UpdatePhase) & " And Type = 1"
DataSetNo = 0 ' counts position-dependent delays
Do Until Delays.NoMatch
    DataSetNo = DataSetNo + 1
    Delays.FindNext "Tunnel = " & CStr(RS!ID) & " And UpdatePhase = " & _
        CStr(UpdatePhase) & " And Type = 1"
Loop
AddLine ImpExp1, DataSetNo + 1
Delays.FindFirst "Tunnel = " & CStr(RS!ID) & " And UpdatePhase = " & _
    CStr(UpdatePhase) & " And Type = 1"
Do Until Delays.NoMatch
    If UpdatePhase > 1 Then
        AddLine ImpExp1, Auxiliary.MaxDb1(0, Delays!Start - _
            Abs(RS!BeginPos - RS!UpdatedBeginPos))
    Else
        AddLine ImpExp1, Delays!Start
    End If
    AddLine ImpExp1, Delays!DurationMin
    AddLine ImpExp1, Delays!DurationMode
    AddLine ImpExp1, Delays!DurationMax
    Delays.FindNext "Tunnel = " & CStr(RS!ID) & " And UpdatePhase = " & _
        CStr(UpdatePhase) & " And Type = 1"
Loop
AddLine ImpExp1, 1000000001#
AddLine ImpExp1, -1
AddLine ImpExp1, -1
AddLine ImpExp1, -1
AddLine ImpExp1, RS!GraphicPos
ImpExp1.Close
DoCmd.TransferText acExportDelim, "Text ExpSpec", "Exp1", _
    Path & "\data_tun_save_" & CStr(n + 1), 0
n = n + 1
End If
RS.MoveNext
If RS.EOF Then Exit Do
Loop
End If
RS.Close
Delays.Close
UpDown.Close
Rep.Close
```

```
DB.Execute ("ImpExp1Del")
Set ImpExp1 = DB.OpenRecordset("ImpExp1", dbOpenTable, dbAppendOnly)
ImpExp1.Index = "PrimaryKey"
Set Nodes = DB.OpenRecordset("tTunnelNodes", dbOpenTable, dbReadOnly)
Nodes.Index = "Name"
MaxNode = Nodes.RecordCount
AddLine ImpExp1, MaxNode
If MaxNode > 0 Then
    ReDim NodeID(MaxNode - 1)
    Nodes.MoveFirst
    n = 0
    Do Until Nodes.EOF
        AddLine ImpExp1, Nodes!Name & "+"
        AddLine ImpExp1, Nodes!Type
        AddLine ImpExp1, n
        AddLine ImpExp1, Nodes!PosX
        AddLine ImpExp1, Nodes!PosY
        NodeID(n) = Nodes!ID
        Nodes.MoveNext
        n = n + 1
    Loop
End If
AddLine ImpExp1, Arcs.RecordCount
If Not (Arcs.EOF And Arcs.BOF) Then
    For n = 0 To MaxTunnel - 1
        Arcs.Seek "=", TunnelID(n)
        If Not Arcs.NoMatch Then
            AddLine ImpExp1, n
            AddLine ImpExp1, GiveArrayPos1D(NodeID, MaxNode, Arcs!StartNode)
            AddLine ImpExp1, GiveArrayPos1D(NodeID, MaxNode, Arcs!EndNode)
            AddLine ImpExp1, Arcs!Type
            Arcs.MoveNext
        End If
    Next n
End If
ImpExp1.Close
DoCmd.TransferText acExportDelim, "Text ExpSpec", "Exp1", _
    Path & "\net_tun_save", 0
Nodes.Close
Arcs.Close
End Sub

Private Sub ExpActivities(ActID() As Long, MaxAct As Integer)
    Dim n As Integer, ImpExpMemo As Recordset

    Set RS = DB.OpenRecordset("tActivities", dbOpenTable, dbReadOnly)
    RS.Index = "Name"
    DB.Execute ("ImpExp1Del")
    Set ImpExp1 = DB.OpenRecordset("ImpExp1", dbOpenTable, dbAppendOnly)
    ImpExp1.Index = "PrimaryKey"
    DB.Execute ("ImpExpMemoDel")
    Set ImpExpMemo = DB.OpenRecordset("ImpExpMemo", dbOpenTable, dbAppendOnly)
    ImpExpMemo.Index = "PrimaryKey"
    If RS.EOF And RS.BOF Then
        AddLine ImpExp1, 0
        MaxAct = 0
        ReDim ActID(0)
    Else
        MaxAct = RS.RecordCount
        ReDim ActID(MaxAct - 1)
        AddLine ImpExp1, MaxAct
        RS.MoveFirst
        For n = 0 To MaxAct - 1
            ActID(n) = RS!ID
            AddLine ImpExp1, RS!Name
            AddLine ImpExp1, n
            AddLine ImpExp1, RS!Priority
            AddLine ImpExpMemo, RS!TimeEquation & ";"
        Next n
    End If
End Sub
```

```
        AddLine ImpExpMemo, RS!CostEquation & ";"
        RS.MoveNext
    Next n
End If
RS.Close
ImpExp1.Close
DoCmd.TransferText acExportDelim, _
    "Text ExpSpec", "Exp1", Path & "\act_save", 0
ImpExpMemo.Close
DoCmd.TransferText acExportDelim, _
    "Text ExpSpec", "ExpMemo", Path & "\gen_eq_save", 0
End Sub

Private Sub ExpActResDemands(ResID() As Long, MaxRes As Integer)
    Dim Activities As Recordset, ResNeed As Recordset, TypeNo As Long
    Dim ResNo As Integer
    If MaxRes = 0 Then Exit Sub
    Set Activities = DB.OpenRecordset("tActivities", dbOpenTable, dbReadOnly)
    Activities.Index = "PrimaryKey"
    If Activities.EOF And Activities.BOF Then Exit Sub
    Activities.MoveFirst
    Set ResNeed = DB.OpenRecordset("tActResDemand", dbOpenTable, dbReadOnly)
    ResNeed.Index = "PrimaryKey"
    DB.Execute ("ImpExp1Del")
    Set ImpExp1 = DB.OpenRecordset("ImpExp1", dbOpenTable, dbAppendOnly)
    ImpExp1.Index = "PrimaryKey"
    AddLine ImpExp1, Activities.RecordCount
    Do Until Activities.EOF
        AddLine ImpExp1, Activities!Name
        For TypeNo = 0 To 3
            For ResNo = 0 To MaxRes - 1
                ResNeed.Seek "=", Activities!ID, ResID(ResNo), TypeNo
                If ResNeed.NoMatch Then
                    AddLine ImpExp1, 0
                Else
                    AddLine ImpExp1, ResNeed!Demand
                End If
            Next ResNo
        Next TypeNo
        Activities.MoveNext
    Loop
    Activities.Close
    ResNeed.Close
    ImpExp1.Close
    DoCmd.TransferText acExportDelim, _
        "Text ExpSpec", "Exp1", Path & "\ress_need_save", 0
End Sub

Private Sub ExpMethods(MethID() As Long, MaxMeth As Integer, ActID() As Long, _
    MaxAct As Integer)
    Dim n As Integer, m As Integer, l As Integer, Heads As Recordset, Learn As Recordset
    Dim NodeID() As Long, MaxNode As Integer, MaxArc As Integer, Variables As Recordset
    Dim NodeDef As QueryDef, ArcDef As QueryDef, Nodes As Recordset, Arcs As Recordset

    Set RS = DB.OpenRecordset("eMethods", dbOpenDynaset, dbReadOnly)
    Set Heads = DB.OpenRecordset("eHeads", dbOpenDynaset, dbReadOnly)
    Set ArcDef = DB.QueryDefs("eMethArcs")
    Set NodeDef = DB.QueryDefs("eMethNodes")
    DB.Execute ("ImpExp1Del")
    Set ImpExp1 = DB.OpenRecordset("ImpExp1", dbOpenTable, dbAppendOnly)
    ImpExp1.Index = "PrimaryKey"
    If RS.EOF And RS.BOF Then
        AddLine ImpExp1, 0
        MaxMeth = 0
        ReDim MethID(0)
    Else
        RS.MoveLast
    End If
End Sub
```

```
MaxMeth = RS.RecordCount
ReDim MethID(MaxMeth - 1)
AddLine ImpExp1, MaxMeth
RS.MoveFirst
For n = 0 To MaxMeth - 1
    MethID(n) = RS!ID
    AddLine ImpExp1, RS!Name
    AddLine ImpExp1, RS!CycleLength
    If RS!UnifiedSimul Then
        AddLine ImpExp1, 0
    Else
        AddLine ImpExp1, 1
    End If
    AddLine ImpExp1, RS!HeadNo
    If RS!UnifiedRound Then
        AddLine ImpExp1, 1
    Else
        AddLine ImpExp1, 0
    End If
    AddLine ImpExp1, RS!VarNo
    Heads.FindFirst "UpdatePhase = " & CStr(UpdatePhase) & _
        " And Method = " & CStr(RS!ID)
    For m = 0 To RS!HeadNo - 1
        DB.Execute ("ImpExp2Del")
        Set ImpExp2 = DB.OpenRecordset("ImpExp2", dbOpenTable, dbAppendOnly)
        ImpExp2.Index = "PrimaryKey"
        NodeDef.Parameters("HeadID") = Heads!ID
        Set Nodes = NodeDef.OpenRecordset(dbOpenDynaset, dbReadOnly)
        ArcDef.Parameters("HeadID") = Heads!ID
        Set Arcs = ArcDef.OpenRecordset(dbOpenDynaset, dbReadOnly)
        If Nodes.BOF And Nodes.EOF Then
            AddLine ImpExp2, 0
        Else
            Nodes.MoveLast
            MaxNode = Nodes.RecordCount
            AddLine ImpExp2, MaxNode
            ReDim NodeID(MaxNode - 1)
            Nodes.MoveFirst
            l = 0
            Do Until Nodes.EOF
                AddLine ImpExp2, Nodes!Name & "+"
                AddLine ImpExp2, Nodes!Type
                AddLine ImpExp2, 1
                AddLine ImpExp2, Nodes!PosX
                AddLine ImpExp2, Nodes!PosY
                NodeID(l) = Nodes!ID
                Nodes.MoveNext
                l = l + 1
            Loop
        End If
        If Arcs.BOF And Arcs.EOF Then
            AddLine ImpExp2, 0
        Else
            Arcs.MoveLast
            MaxArc = Arcs.RecordCount
            AddLine ImpExp2, MaxArc
            Arcs.MoveFirst
            For l = 0 To MaxArc - 1
                AddLine ImpExp1, GiveArrayPos1D(ActID, MaxAct, Arcs!Activity)
                AddLine ImpExp2, 1
                AddLine ImpExp2, GiveArrayPos1D(NodeID, MaxNode, Arcs!StartNode)
                AddLine ImpExp2, GiveArrayPos1D(NodeID, MaxNode, Arcs!EndNode)
                AddLine ImpExp2, Arcs!Type
                Arcs.MoveNext
            Next l
        End If
        ImpExp2.Close
        DoCmd.TransferText acExportDelim, "Text ExpSpec", "Exp2", _
```

```
        Path & "\net_act_save_" & CStr(n + 1) & "_" & CStr(m + 1), 0
    Heads.MoveNext
    Nodes.Close
    Arcs.Close
Next m
RS.MoveNext
Next n
End If
ImpExp1.Close
DoCmd.TransferText acExportDelim, _
    "Text ExpSpec", "Exp1", Path & "\new_method_save", 0

' learn parameters and variables
DB.Execute ("ImpExp1Del")
Set ImpExp1 = DB.OpenRecordset("ImpExp1", dbOpenTable, dbAppendOnly)
ImpExp1.Index = "PrimaryKey"
DB.Execute ("ImpExp2Del")
Set ImpExp2 = DB.OpenRecordset("ImpExp2", dbOpenTable, dbAppendOnly)
ImpExp2.Index = "PrimaryKey"
Set Learn = DB.OpenRecordset("tLearnParam", dbOpenTable, dbReadOnly)
Set Variables = DB.OpenRecordset("eVariables", dbOpenDynaset, dbReadOnly)
Learn.Index = "PrimaryKey"
If MaxMeth = 0 Then
    AddLine ImpExp1, 0
Else
    AddLine ImpExp1, MaxMeth
    RS.MoveFirst
    For n = 0 To MaxMeth - 1
        If UpdatePhase = 1 Then
            AddLine ImpExp1, RS!LearnNo
            Learn.Seek ">=", RS!ID
            For m = 0 To RS!LearnNo - 1
                AddLine ImpExp1, Learn!UpToCycle
                AddLine ImpExp1, Learn!Efficiency
                Learn.MoveNext
            Next m
        Else
            AddLine ImpExp1, 0 ' do not export learn parameters in update phases
        End If
        Variables.FindFirst "UpdatePhase = " & CStr(UpdatePhase) & _
            " And Method = " & CStr(RS!ID)
        For m = 0 To RS!VarNo - 1
            AddLine ImpExp2, Variables!Name
            AddLine ImpExp2, Variables!Min
            AddLine ImpExp2, Variables!Mode
            AddLine ImpExp2, Variables!Max
            AddLine ImpExp2, Variables!ProbMin
            AddLine ImpExp2, Variables!ProbMax
        Next m
        RS.MoveNext
    Next n
End If
ImpExp1.Close
DoCmd.TransferText acExportDelim, _
    "Text ExpSpec", "Exp1", Path & "\learn_save", 0
ImpExp2.Close
DoCmd.TransferText acExportDelim, "Text ExpSpec", "Exp2", _
    Path & "\new_method_var_save", 0
Learn.Close
Variables.Close

' head - bench information
DB.Execute ("ImpExp1Del")
Set ImpExp1 = DB.OpenRecordset("ImpExp1", dbOpenTable, dbAppendOnly)
ImpExp1.Index = "PrimaryKey"
If MaxMeth > 0 Then
    RS.MoveFirst
    For n = 0 To MaxMeth - 1
```



```
Heads.FindFirst "UpdatePhase = " & CStr(UpdatePhase) & _
                " And Method = " & CStr(RS!ID)
For m = 0 To RS!HeadNo - 1
    AddLine ImpExpl, Heads!CycleMin
    AddLine ImpExpl, Heads!CycleMode
    AddLine ImpExpl, Heads!CycleMax
    AddLine ImpExpl, Heads!CycleProbMin
    AddLine ImpExpl, Heads!CycleProbMax
    AddLine ImpExpl, Heads!DistMin
    AddLine ImpExpl, Heads!DistMax
Next m
RS.MoveNext
Next n
End If
ImpExpl.Close
DoCmd.TransferText acExportDelim, "Text ExpSpec", "Exp1", _
    Path & "\hb_save", 0
Heads.Close
RS.Close
End Sub

Private Sub ExpChangeDelays(MethID() As Long, MaxMeth As Integer)
    Set RS = DB.OpenRecordset("tChangeDelays", dbOpenTable, dbReadOnly)
    RS.Index = "PrimaryKey"
    DB.Execute ("ImpExplDel")
    Set ImpExpl = DB.OpenRecordset("ImpExpl", dbOpenTable, dbAppendOnly)
    ImpExpl.Index = "PrimaryKey"
    If Not (RS.EOF And RS.BOF) Then
        RS.MoveFirst
        Do Until RS.EOF
            If RS!UpdatePhase = UpdatePhase Then
                AddLine ImpExpl, GiveArrayPos1D(MethID(), MaxMeth, RS!FromMethod)
                AddLine ImpExpl, GiveArrayPos1D(MethID(), MaxMeth, RS!ToMethod)
                AddLine ImpExpl, RS!Delay
            End If
            RS.MoveNext
        Loop
    End If
    AddLine ImpExpl, -1
    RS.Close
    ImpExpl.Close
    DoCmd.TransferText acExportDelim, _
        "Text ExpSpec", "Exp1", Path & "\delay_save", 0
End Sub

Private Sub ExpGCCGeomMeth(MethID() As Long, MaxMeth As Integer, _
    GeomID() As Long, MaxGeom As Integer)
    Dim n As Integer, m As Integer, Criterion As String

    Set RS = DB.OpenRecordset("eAssoc", dbOpenDynaset, dbReadOnly)
    DB.Execute ("ImpExplDel")
    Set ImpExpl = DB.OpenRecordset("ImpExpl", dbOpenTable, dbAppendOnly)
    ImpExpl.Index = "PrimaryKey"
    AddLine ImpExpl, MaxGC
    AddLine ImpExpl, MaxGeom
    For n = 0 To MaxGC - 1
        For m = 0 To MaxGeom - 1
            Criterion = "CGC = " & CStr(GCID(n)) & " And Geometry = " & CStr(GeomID(m)) & _
                " And UpdatePhase = " & CStr(UpdatePhase)
            RS.FindFirst Criterion
            If RS.NoMatch Then
                AddLine ImpExpl, 0
            Else
                AddLine ImpExpl, RS!MethodNo
            End If
            Do Until RS.NoMatch
                AddLine ImpExpl, GiveArrayPos1D(MethID(), MaxMeth, RS!Method)
            Loop
        Next m
    Next n
End Sub
```

```
        AddLine ImpExp1, RS!Prob
        RS.FindNext Criterion
    Loop
Next m
Next n
RS.Close
ImpExp1.Close
DoCmd.TransferText acExportDelim, _
    "Text ExpSpec", "Exp1", Path & "\data_method_save", 0
End Sub

Private Sub ExpGenVariables()
    Dim Count As Recordset
    Set RS = DB.OpenRecordset("eGenVariables", dbOpenDynaset, dbReadOnly)
    Set Count = DB.OpenRecordset("tGenVariables", dbOpenTable, dbReadOnly)
    DB.Execute ("ImpExp1Del")
    Set ImpExp1 = DB.OpenRecordset("ImpExp1", dbOpenTable, dbAppendOnly)
    ImpExp1.Index = "PrimaryKey"
    AddLine ImpExp1, Count.RecordCount
    If Not (RS.EOF And RS.BOF) Then
        RS.MoveFirst
        Do Until RS.EOF
            If RS!UpdatePhase = UpdatePhase Then
                AddLine ImpExp1, RS!Name
                AddLine ImpExp1, RS!Description
                AddLine ImpExp1, RS!Unit
                AddLine ImpExp1, RS!Min
                AddLine ImpExp1, RS!Mode
                AddLine ImpExp1, RS!Max
                AddLine ImpExp1, RS!ProbMin
                AddLine ImpExp1, RS!ProbMax
            End If
            RS.MoveNext
        Loop
    End If
    RS.Close
    Count.Close
    ImpExp1.Close
    DoCmd.TransferText acExportDelim, _
        "Text ExpSpec", "Exp1", Path & "\general_var_save", 0
End Sub

Private Function GiveArrayPos1D(Structure() As Long, Size As Integer, _
    Value As Long) As Integer
    Static n As Integer
    GiveArrayPos1D = -1
    For n = 0 To Size - 1
        If Structure(n) = Value Then GiveArrayPos1D = n
    Next n
End Function

Private Function GiveArrayPos2DN(Structure() As Long, SizeN As Integer, _
    SizeM As Integer, Value As Long) As Integer
    Static n As Integer, m As Integer
    GiveArrayPos2DN = -1
    For n = 0 To SizeN - 1
        For m = 0 To SizeM - 1
            If Structure(n, m) = Value Then GiveArrayPos2DN = n
        Next m
    Next n
End Function

Private Function GiveArrayPos2DM(Structure() As Long, SizeN As Integer, _
    SizeM As Integer, Value As Long) As Integer
    Static n As Integer, m As Integer
    GiveArrayPos2DM = -1
    For n = 0 To SizeN - 1
```

```
For m = 0 To SizeM - 1
    If Structure(n, m) = Value Then GiveArrayPos2DM = m
Next m
Next n
End Function

Private Sub AddLine(RS As Recordset, Value As Variant)
    RS.addnew
    RS!F1 = CStr(Value)
    RS.Update
End Sub

Private Sub CreateTemporaryObjects()
    Dim ZoneStat As Recordset, ParamXSet As Recordset, Zones As Recordset
    Dim Bookmark As Variant, ParamSet As Long, Parameter As Long, ZoneNo As Integer
    Dim InKnownPart As Boolean, Order As Double, ZoneTab As Recordset
    Set ZoneStat = DB.OpenRecordset("oqZoneParamStat", dbOpenDynaset, dbReadOnly)
    Set Zones = DB.OpenRecordset("qZonesPost", dbOpenSnapshot)
    Set ZoneTab = DB.OpenRecordset("tZones", dbOpenTable)
    ZoneTab.Index = "PrimaryKey"
    Set ParamXSet = DB.OpenRecordset("tParam_X_Set", dbOpenTable)
    ParamXSet.Index = "PrimaryKey"

    MaxZone = Zones.RecordCount
    ReDim ZoneParamSet(MaxZone - 1, 1)
    If MaxZone = 0 Then Exit Sub
    Zones.MoveFirst
    ZoneNo = 0
    Do Until Zones.EOF
        ZoneParamSet(ZoneNo, 0) = Zones!Zone
        ZoneParamSet(ZoneNo, 1) = Zones!ParamSet
        If Zones!Status = "Observed" Then
            ParamSet = Zones!ParamSet
            Bookmark = Zones.Bookmark
            Zones.FindFirst "ParamSet = " & CStr(ParamSet) & _
                " And Zone <> " & CStr(ZoneParamSet(ZoneNo, 0))
            If Zones.NoMatch Then ' this zone is the only one using this parameter set
                Zones.Bookmark = Bookmark
                CopyDeterministicSequence ParamXSet, ZoneStat, ParamSet, Zones!Zone
            Else ' at least one other zone uses the same parameter set
                ' duplicate param set; param sets start with numbers above 1 billion
                Zones.Bookmark = Bookmark
                DuplicateParamSet Zones!ParamSet, ZoneNo + 1000000000, ParamXSet
                ZoneTab.Seek "=", Zones!Zone
                ZoneTab.Edit
                ZoneTab!ParamSet = ZoneNo + 1000000000
                ZoneTab.Update
                CopyDeterministicSequence ParamXSet, ZoneStat, ZoneNo + 1000000000, Zones!Zone
            End If
        ElseIf Zones!Status = "Current" Then
            InKnownPart = False
            ' temporary abuse of variable to find out if there are parameters with
            ' Markov generation mode
            ParamXSet.Seek ">=", Zones!ParamSet
            Do Until ParamXSet.EOF
                If ParamXSet!ParamSet <> Zones!ParamSet Then Exit Do
                If ParamXSet!GenerationMode = "0" Then InKnownPart = True
                ParamXSet.MoveNext
            Loop
            If InKnownPart Then ' there are parameters with Markov gen mode
                ' part of the zone is known, part is unknown. We must subdivide the zone into
                ' as many zones as there are parts, i.e. at least two if there is only an
                ' observed and an unobserved part.
                Order = Zones!Order
                ZoneStat.FindLast "Zone = " & CStr(Zones!Zone)
                ' start at the end and go backwards. We always keep the original zone as the
                ' last part of the subdivided zone. We can then leave the endpos as it is and
```

```
' work our way backwards through the other parts, decreasing 'order' by a small
' amount in every part
If Not ZoneStat.NoMatch Then
    Parameter = ZoneStat!Parameter
    ParamSet = ZoneNo + 1000000000
    DuplicateParamSet Zones!ParamSet, ParamSet, ParamXSet
    CopyDeterministicSequence ParamXSet, ZoneStat, ParamSet, Zones!Zone
    If ZoneStat!Part Then ' last part is unknown
        ' this last part of the zone just stays as it is
        InKnownPart = True ' will now enter known part
        ZoneStat.FindLast "Zone = " & CStr(Zones!Zone)
    Else ' last part is known --> use duplicate parameter set
        ZoneTab.Seek "=", Zones!Zone
        ZoneTab.Edit
        ZoneTab!ParamSet = ParamSet
        ZoneTab.Update
        ZoneStat.FindPrevious "Zone = " & CStr(Zones!Zone) & _
            " And Parameter = " & CStr(Parameter) & " And Part = TRUE"
        InKnownPart = False ' will now enter unknown part
    End If
End If
Do Until ZoneStat.NoMatch
    Order = Order - 0.000000001
    If InKnownPart Then ' this is the end of a known part
        DuplicateZone Zones, ZoneStat!EndPos, ParamSet, Order
        InKnownPart = False ' enter now unknown part
    Else
        DuplicateZone Zones, ZoneStat!EndPos - ZoneStat!Length, _
            Zones!ParamSet, Order
        InKnownPart = True
    End If
    ZoneStat.FindPrevious "Zone = " & CStr(Zones!Zone) & " And Parameter = " & _
        CStr(Parameter) & " And Part = TRUE"
Loop
End If
End If
Zones.MoveNext
ZoneNo = ZoneNo + 1
Loop
End Sub

Private Sub DeleteTemporaryObjects()
    Dim ParamXSet As Recordset, Zones As Recordset, ZoneProb As Recordset
    Dim Correl As Recordset, CorrelStat As Recordset, GC As Recordset, GCStat As Recordset
    Dim DeterSegs As Recordset, ParamSets As Recordset, ML As Recordset
    Dim Delays As Recordset, DelayProb As Recordset
    Dim Bookmark As Variant, ParamSet As Long, Parameter As Long, ZoneNo As Integer

    Set Zones = DB.OpenRecordset("tZones", dbOpenTable, dbAppendOnly)
    Set ZoneProb = DB.OpenRecordset("tZoneProb", dbOpenTable, dbAppendOnly)
    ZoneProb.Index = "PrimaryKey"
    If Zones.EOF And Zones.BOF Then Exit Sub
    Zones.MoveFirst
    Do Until Zones.EOF
        ZoneProb.Seek "=", Zones!ID, 1
        ZoneNo = 1
        If ZoneProb.NoMatch Then
            ZoneProb.Seek "=", Zones!ID, UpdatePhase
            ' delete temporary zones (those for which no prior input exists)
            If Not ZoneProb.NoMatch Then ZoneProb.delete
            Zones.delete
        Else
            Do Until ZoneNo >= MaxZone
                If ZoneParamSet(ZoneNo, 0) = Zones!ID Then ' reset to original parameter sets
                    Zones.Edit
                    Zones!ParamSet = ZoneParamSet(ZoneNo, 1)
                    Zones.Update
                Exit Do
            End Do
        End If
    Loop
End Sub
```

```
        End If
        ZoneNo = ZoneNo + 1
    Loop
End If
Zones.MoveNext
Loop
ZoneProb.Close
Zones.Close

' All temporary parameter sets have ID >= 1000000000
Set ParamXSet = DB.OpenRecordset("tParam_X_Set", dbOpenTable)
Set DeterSegs = DB.OpenRecordset("qNonMarkovDeterPost", dbOpenDynaset)
Set ML = DB.OpenRecordset("pMarkovLengths", dbOpenDynaset, dbReadOnly)
ParamXSet.MoveFirst
DeterSegs.MoveFirst
Do Until ParamXSet.EOF
    If ParamXSet!ParamSet >= 1000000000 Then
        DeterSegs.FindFirst "ParamSet = " & CStr(ParamXSet!ParamSet) & _
            " And Parameter = " & CStr(ParamXSet!Parameter)
        Do Until DeterSegs.NoMatch
            DeterSegs.delete
            If DeterSegs.EOF Then Exit Do
            DeterSegs.MoveFirst
            DeterSegs.FindFirst "ParamSet = " & CStr(ParamXSet!ParamSet) & _
                " And Parameter = " & CStr(ParamXSet!Parameter)
        Loop
        ParamXSet.delete
    ElseIf ParamXSet!GenerationMode = "1" Then
        ML.FindFirst "ParamSet = " & CStr(ParamXSet!ParamSet) & _
            " And Parameter = " & CStr(ParamXSet!Parameter)
        If Not ML.NoMatch Then
            ' the generation mode was only changed temporarily from Markov for the export
            ' and must be changed back. DeterSegs are only temporary too.
            ParamXSet.Edit
            ParamXSet!GenerationMode = "0"
            ParamXSet.Update
            DeterSegs.FindFirst "ParamSet = " & CStr(ParamXSet!ParamSet) & _
                " And Parameter = " & CStr(ParamXSet!Parameter)
            Do Until DeterSegs.NoMatch
                DeterSegs.delete
                If DeterSegs.EOF Then Exit Do
                DeterSegs.MoveFirst
                DeterSegs.FindFirst "ParamSet = " & CStr(ParamXSet!ParamSet) & _
                    " And Parameter = " & CStr(ParamXSet!Parameter)
            Loop
        End If
    End If
    ParamXSet.MoveNext
Loop
ParamXSet.Close
DeterSegs.Close

Set Correl = DB.OpenRecordset("tCorrelations", dbOpenTable)
If Not (Correl.EOF And Correl.BOF) Then
    Set CorrelStat = DB.OpenRecordset("tCorrel_X_States", dbOpenTable)
    CorrelStat.Index = "PrimaryKey"
    Correl.MoveFirst
    Do Until Correl.EOF
        If Correl!ParamSet >= 1000000000 Then
            CorrelStat.Seek "=", Correl!ID
            Do Until CorrelStat.NoMatch
                CorrelStat.delete
                CorrelStat.Seek "=", Correl!ID
            Loop
            Correl.delete
        End If
        Correl.MoveNext
    Loop
End If
```

```
End If
Correl.Close
CorrelStat.Close

Set GC = DB.OpenRecordset("tGCCCombinations", dbOpenTable)
If Not (GC.EOF And GC.BOF) Then
    Set GCStat = DB.OpenRecordset("tGCCComb_X_States", dbOpenTable)
    GCStat.Index = "PrimaryKey"
    GC.MoveFirst
    Do Until GC.EOF
        If GC!ParamSet >= 1000000000 Then
            GCStat.Seek "=", GC!ID
            Do Until GCStat.NoMatch
                GCStat.delete
                GCStat.Seek "=", GC!ID
            Loop
            GC.delete
        End If
        GC.MoveNext
    Loop
End If
GC.Close
GCStat.Close

Set ParamSets = DB.OpenRecordset("tParamSets", dbOpenTable)
ParamSets.MoveFirst
Do Until ParamSets.EOF
    If ParamSets!ID >= 1000000000 Then ParamSets.delete
    ParamSets.MoveNext
Loop
ParamSets.Close

Set Delays = DB.OpenRecordset("tTunDelays", dbOpenTable)
If Not (Delays.EOF And Delays.BOF) Then
    Set DelayProb = DB.OpenRecordset("tTunDelayProb", dbOpenTable)
    DelayProb.Index = "PrimaryKey"
    Delays.MoveFirst
    Do Until Delays.EOF
        If Delays!ExportStatus = "Inactive" Then
            Delays.Edit
            Delays!ExportStatus = "Active"
            Delays.Update
        ElseIf Delays!ExportStatus = "Temporary" Then
            DelayProb.Seek "=", Delays!ID, UpdatePhase
            If Not DelayProb.NoMatch Then DelayProb.delete
            Delays.delete
        End If
        Delays.MoveNext
    Loop
End If

End Sub

Private Sub DuplicateParamSet(Source As Long, Target As Long, ParamXSet As Recordset)
    Static ParamSets As Recordset, ParamXSetSnap As Recordset, Correl As Recordset
    Static CorrelSnap As Recordset, CorrelStat As Recordset, CorrelStatSnap As Recordset
    Static GC As Recordset, GCSnap As Recordset, GCStat As Recordset
    Static SourceID As Long, TargetID As Long, GCStatSnap As Recordset

    ' The snapshots are originals from which the copies are made in the table recordsets

    Set ParamSets = DB.OpenRecordset("tParamSets", dbOpenTable)
    Set ParamXSetSnap = DB.OpenRecordset("tParam_X_Set", dbOpenSnapshot)
    Set Correl = DB.OpenRecordset("tCorrelations", dbOpenTable)
    Set CorrelSnap = DB.OpenRecordset("tCorrelations", dbOpenSnapshot)
    Set CorrelStat = DB.OpenRecordset("tCorrel_X_States", dbOpenTable)
    Set CorrelStatSnap = DB.OpenRecordset("tCorrel_X_States", dbOpenSnapshot)
```

```
Set GC = DB.OpenRecordset("tGCCCombinations", dbOpenTable)
Set GCSnap = DB.OpenRecordset("tGCCCombinations", dbOpenSnapshot)
Set GCStat = DB.OpenRecordset("tGCCComb_X_States", dbOpenTable)
Set GCStatSnap = DB.OpenRecordset("tGCCComb_X_States", dbOpenSnapshot)

ParamSets.addnew
ParamSets!ID = Target
ParamSets.Update
ParamXSetSnap.FindFirst "ParamSet = " & CStr(Source)
Do Until ParamXSetSnap.NoMatch
    ParamXSet.addnew
    ParamXSet!ParamSet = Target
    ParamXSet!Parameter = ParamXSetSnap!Parameter
    ParamXSet!StartProbCorr = ParamXSetSnap!StartProbCorr
    ParamXSet!GenerationMode = "0"
    ParamXSet.Update
    ParamXSetSnap.FindNext "ParamSet = " & CStr(Source)
Loop
CorrelSnap.FindFirst "ParamSet = " & CStr(Source)
Do Until CorrelSnap.NoMatch
    Correl.addnew
    Correl!ParamSet = Target
    Correl!Value = CorrelSnap!Value
    TargetID = Correl!ID
    SourceID = CorrelSnap!ID
    Correl.Update
    CorrelStatSnap.FindFirst "Correlation = " & CStr(SourceID)
    Do Until CorrelStatSnap.NoMatch
        CorrelStat.addnew
        CorrelStat!Correlation = TargetID
        CorrelStat!ParamState = CorrelStatSnap!ParamState
        CorrelStat.Update
        CorrelStatSnap.FindNext "Correlation = " & CStr(SourceID)
    Loop
    CorrelSnap.FindNext "ParamSet = " & CStr(Source)
Loop
GCSnap.FindFirst "ParamSet = " & CStr(Source)
Do Until GCSnap.NoMatch
    GC.addnew
    GC!ParamSet = Target
    GC!GC = GCSnap!GC
    TargetID = GC!ID
    SourceID = GCSnap!ID
    GC.Update
    GCStatSnap.FindFirst "GCComb = " & CStr(SourceID)
    Do Until GCStatSnap.NoMatch
        GCStat.addnew
        GCStat!GCComb = TargetID
        GCStat!ParamState = GCStatSnap!ParamState
        GCStat.Update
        GCStatSnap.FindNext "GCComb = " & CStr(SourceID)
    Loop
    GCSnap.FindNext "ParamSet = " & CStr(Source)
Loop
End Sub

Private Sub CopyDeterministicSequence(ParamXSet As Recordset, ZoneStat As Recordset, _
    ParamSet As Long, Zone As Long)

    Static DeterSegs As Recordset
    Set DeterSegs = DB.OpenRecordset("tDeterSegs", dbOpenTable)

    ParamXSet.Seek ">=", ParamSet
    Do Until ParamXSet.EOF
        If ParamXSet!ParamSet <> ParamSet Then Exit Do
        If ParamXSet!GenerationMode = "0" Then
            ' change generation mode to deterministic
            ParamXSet.Edit
            ParamXSet!GenerationMode = "1"
        End If
    Loop
```

```
ParamXSet.Update
ZoneStat.FindFirst "Zone = " & CStr(Zone) & " And Parameter = " & _
    CStr(ParamXSet!Parameter)
' create deterministic segments from observations
Do Until ZoneStat.NoMatch
    DeterSegs.addnew
    DeterSegs!ParamSet = ParamSet
    DeterSegs!ParamState = ZoneStat!ParamState
    DeterSegs!EndPos = ZoneStat!EndPos
    DeterSegs!UpdatePhase = UpdatePhase
    DeterSegs.Update
    ZoneStat.FindNext "Zone = " & CStr(Zone) & " And Parameter = " & _
        & CStr(ParamXSet!Parameter)
Loop
End If
ParamXSet.MoveNext
Loop
DeterSegs.Close
End Sub

Private Sub DuplicateZone(Source As Recordset, EndPos As Double, _
    ParamSet As Long, Order As Double)
    Static Zones As Recordset, ZoneProb As Recordset, Zone As Long

    Set Zones = DB.OpenRecordset("tZones", dbOpenTable, dbAppendOnly)
    Set ZoneProb = DB.OpenRecordset("tZoneProb", dbOpenTable, dbAppendOnly)
    Zones.addnew
    Zones!ParamSet = ParamSet
    Zones!Name = Source!Name
    Zones!Area = Source!Area
    Zones!Order = Order
    Zone = Zones!ID
    Zones.Update
    ZoneProb.addnew
    ZoneProb!Zone = Zone
    ZoneProb!GenerationMode = "1"
    Auxiliary.SetEndpos ZoneProb, EndPos
    ZoneProb!UpdatePhase = UpdatePhase
    ZoneProb.Update
End Sub

Private Sub CreateTemporaryDelays()
    Dim Arcs As Recordset, Tunnels As Recordset, Delays As Recordset, ObsDelays As Recordset
    Dim DelayTab As Recordset, DelayProbTab As Recordset, Delay As Long
    Dim StartNode As Long, CurrentTime As Double, Bookmark As Variant

    Set Tunnels = DB.OpenRecordset("tTunnels", dbOpenTable, dbReadOnly)
    Set Arcs = DB.OpenRecordset("tTunnelArcs", dbOpenSnapshot)
    Set Delays = DB.OpenRecordset("qTunDelays", dbOpenSnapshot)
    Set DelayTab = DB.OpenRecordset("tTunDelays", dbOpenTable)
    DelayTab.Index = "PrimaryKey"
    Set DelayProbTab = DB.OpenRecordset("tTunDelayProb", dbOpenTable)
    Set ObsDelays = DB.OpenRecordset("otTunDelays", dbOpenSnapshot)
    CurrentTime = DLookup("Current", "tTime")

    If Delays.EOF And Delays.EOF Then Exit Sub
    Delays.MoveFirst
    Do Until Delays.EOF
        If Delays!Status <> "Posterior" Then
            DelayTab.Seek "=", Delays!Delay
            DelayTab.Edit
            DelayTab!ExportStatus = "Inactive" ' make observed and current delays inactive
            DelayTab.Update
        End If
        Delays.MoveNext
    Loop
```



```
If Tunnels.EOF And Tunnels.EOF Then Exit Sub
Tunnels.MoveFirst
Do Until Tunnels.EOF
    Arcs.FindFirst "Tunnel = " & CStr(Tunnels!ID)
    If Not Arcs.NoMatch Then
        StartNode = Arcs!StartNode
        Arcs.FindFirst "EndNode = " & CStr(StartNode)
        If Arcs.NoMatch Then
            ' the start node of the arc of this tunnel is not end node of any other arc
            ' --> it is a starting point of the tunnel network
            Delays.FindFirst "Tunnel = " & CStr(Tunnels!ID) & " And Status = 'Current'"
            If Delays.NoMatch Then
                DelayTab.addnew ' add a delay to make construction start at current time
                DelayTab!Tunnel = Tunnels!ID
                Delay = DelayTab!ID
                DelayTab!ExportStatus = "Temporary"
                DelayTab.Update
                DelayProbTab.addnew
                DelayProbTab!Delay = Delay
                DelayProbTab!Type = 0 ' time-dependent
                DelayProbTab!Status = "Prior"
                DelayProbTab!DurationMin = CurrentTime
                DelayProbTab!DurationMode = -1
                DelayProbTab!DurationMax = CurrentTime
                DelayProbTab!Start = 0
                DelayProbTab!UpdatePhase = UpdatePhase
                DelayProbTab.Update
            Else
                ObsDelays.FindFirst "Tunnel = " & CStr(Tunnels!ID) & " And Part = TRUE"
                ' recreate current delay such that it includes the time up to current
                RecreateDelay Delays, DelayTab, DelayProbTab, _
                    CurrentTime - ObsDelays!Duration
            End If
        Else
            ' the start node is not a starting point of the tunnel network
            Delays.FindFirst "Tunnel = " & CStr(Tunnels!ID) & " And Status = 'Current'"
            If Not Delays.NoMatch Then
                ObsDelays.FindFirst "Tunnel = " & CStr(Tunnels!ID) & " And Part = TRUE"
                ' recreate current delay with remaining duration time
                RecreateDelay Delays, DelayTab, DelayProbTab, -ObsDelays!Duration
            End If
        End If
    End If
    Tunnels.MoveNext
Loop
Delays.Close
DelayTab.Close
DelayProbTab.Close
Tunnels.Close
Arcs.Close
ObsDelays.Close
End Sub

Private Sub RecreateDelay(Delays As Recordset, DelayTab As Recordset, _
    DelayProbTab As Recordset, TimeAddition As Double)
    Static DurationMin As Double, DurationMode As Double, DurationMax As Double
    Static Delay As Long

    DurationMin = Delays!DurationMin
    DurationMode = Delays!DurationMode
    DurationMax = Delays!DurationMax
    DelayTab.addnew
    DelayTab!Tunnel = Delays!Tunnel
    Delay = DelayTab!ID
    DelayTab!ExportStatus = "Temporary"
    DelayTab.Update
    DelayProbTab.addnew
    DelayProbTab!Delay = Delay
```

```
DelayProbTab!Type = 0 ' time-dependent
DelayProbTab!Status = "Prior"
DelayProbTab!DurationMin = TimeAddition + DurationMin
If DurationMode >= 0 Then
    DelayProbTab!DurationMode = TimeAddition + DurationMode
Else
    DelayProbTab!DurationMode = -1
End If
DelayProbTab!DurationMax = TimeAddition + DurationMax
DelayProbTab!Start = 0
DelayProbTab!UpdatePhase = UpdatePhase
DelayProbTab.Update
End Sub
```

Module: Graphs

```
Option Compare Database
Option Explicit
```

```
Public Sub GraphsPriorPostCreate(DB As Database, SourceName As String, _
    TargetName As String, UpdatePhase As Long)
    Static Source As Recordset, Target As Recordset, Title As String
    If UpdatePhase = 1 Then
        Title = "Prior"
    Else
        Title = "Posterior"
    End If
    If Left$(SourceName, 1) = "t" Then
        ' source is a table
        Set Source = DB.OpenRecordset(SourceName, dbOpenTable, dbReadOnly)
    Else
        ' source is a query
        Set Source = DB.OpenRecordset(SourceName, dbOpenDynaset, dbReadOnly)
    End If
    Set Target = DB.OpenRecordset(TargetName, dbOpenTable)
    Target.Index = "PrimaryKey"
    Do Until Source.EOF
        If Source!UpdatePhase = UpdatePhase Then
            If SourceName = "tMarkovLengths" Then
                ' Markov graph
                GraphMarkovPriorPost Source, Target, Title
            Else
                ' Any other graph
                GraphVariablesPriorPost Source, Target, Title
            End If
        End If
        Source.MoveNext
    Loop
    Source.Close
    Target.Close
End Sub

Public Sub GraphMarkovPriorPost(Source As Recordset, Target As Recordset, Title As String)
    Static Order As Integer, IsNew As Boolean
    ' try to find existing records
    Target.Index = "PrimaryKey"
    Target.Seek "=", Source!ParamSet, Source!ParamState, Title, 0
    IsNew = Target.NoMatch
    ' IsNew is True when we have to create a new record to add a new point, i.e.
    ' when there is no more "unused" record available
    For Order = 0 To 3
        If (Order = 3) And (Source!LengthMode >= 0) Then
            ' triangular distribution, no fourth point necessary
            If Not IsNew Then Target.delete
        Else
            If IsNew Then
```

```
Target.addnew
Target!ParamSet = Source!ParamSet
Target!ParamState = Source!ParamState
Target!Title = Title
Else
    Target.Edit
End If
Target!Order = Order
If Source!LengthMode < 0 Then
    ' rectangular distribution
    If Order <= 1 Then
        Target!X = Source!LengthMin
    Else
        Target!X = Source!LengthMax
    End If
    If (Order = 0) Or (Order = 3) Then
        Target!Y = 0
    ElseIf Source!LengthMax = Source!LengthMin Then
        ' in this particular case Y would be infinite to make the area = 1
        Target!Y = 1
    Else
        Target!Y = 1 / (Source!LengthMax - Source!LengthMin)
        ' make area under curve = 1
    End If
Else
    ' triangular distribution (the case of order = 3 was intercepted before)
    Target!X = Source(Order + 3).Value
    If Order = 1 Then
        If Source!LengthMax = Source!LengthMin Then
            Target!Y = 1 ' would be infinite
        Else
            Target!Y = 2 / (Source!LengthMax - Source!LengthMin) ' area = 1
        End If
    Else
        Target!Y = 0
    End If
End If
Target.Update
If Not (IsNew Or (Order = 3)) Then
    Target.MoveNext
    If Target.EOF Then ' no more existing points
        IsNew = True
    ElseIf (Target!ParamSet <> Source!ParamSet) Or (Target!ParamState < _
        <> Source!ParamState) Or (Target!Title <> Title) Then
        IsNew = True ' no more existing points of this graph
    End If
End If
End If
Next Order
End Sub

Public Sub GraphVariablesPriorPost(Source As Recordset, Target As Recordset, _
    Title As String)
    Static Order As Integer, IsNew As Boolean, ModeY As Double, MinY As Double
    Static MaxY As Double
    ' try to find existing records
    Target.Index = "PrimaryKey"
    If Source!Max <= Source!Min Then
        ' area under curve would be zero or negative --> ModeY would be negative or infinite
        ModeY = 1
        MinY = 1
        MaxY = 1
    Else
        If Source!Mode < 0 Then
            ' rectangular distribution, make area under curve = 1
            ModeY = (1 - Source!ProbMax - Source!ProbMin) / (Source!Max - Source!Min)
        Else
            ' triangular distribution, make area under curve = 1
```

```
ModeY = 2 * (1 - Source!ProbMax - Source!ProbMin) / (Source!Max - Source!Min)
End If
' Y value of spikes (base of spike is (Max-Min) / 5)
MinY = 5 * Source!ProbMin / (Source!Max - Source!Min)
MaxY = 5 * Source!ProbMax / (Source!Max - Source!Min)
End If
Target.Seek "=", Source(0).Value, Title, 0
IsNew = Target.NoMatch
For Order = 0 To 11
    If IsNew Then
        Target.addnew
        Target!Variable = Source(0).Value
        Target!Title = Title
    Else
        Target.Edit
    End If
    Target!Order = Order
    Select Case Order
        Case 0, 1 ' left edge of prob min spike
            Target!X = 1.1 * Source!Min - 0.1 * Source!Max
        Case 2, 3 ' right edge of prob min spike
            Target!X = 0.9 * Source!Min + 0.1 * Source!Max
        Case 4 ' left corner of triangle / rectangle
            Target!X = Source!Min
        Case 5
            If Source!Mode < 0 Then
                Target!X = Source!Min ' left top corner
            Else
                Target!X = Source!Mode ' peak of triangle
            End If
        Case 6
            If Source!Mode < 0 Then
                Target!X = Source!Max ' right top corner
            Else
                Target!X = Source!Mode ' right corner of triangle
            End If
        Case 7
            Target!X = Source!Max ' right corner of triangle/rectangle
            ' note that for the triangle case this point is doubled
        Case 8, 9
            Target!X = 0.9 * Source!Max + 0.1 * Source!Min ' left edge of prob max spike
        Case 10, 11
            Target!X = 1.1 * Source!Max - 0.1 * Source!Min ' right edge of prob max spike
    End Select
    Select Case Order
        Case 0, 3, 4, 7, 8, 11
            Target!Y = 0
        Case 1, 2
            Target!Y = MinY ' top edge of prob min spike
        Case 5, 6
            Target!Y = ModeY ' top edge of rectangle, peak of triangle
        Case 9, 10
            Target!Y = MaxY ' top edge of prob max spike
    End Select
    Target.Update
    If Not (IsNew Or (Order = 11)) Then
        Target.MoveNext
        If Target.EOF Then
            IsNew = True
        ElseIf (Target!Variable <> Source(0).Value) Or (Target!Title <> Title) Then
            IsNew = True
        End If
    End If
Next Order
End Sub

Public Sub GraphMarkovLikelihood(Prior As Recordset, Target As Recordset, _
    X As Double, n As Integer)
```

```
Static LowerInfl As Double, Mode As Double, UpperInfl As Double, Area As Double
Static Order As Integer, IsNew As Boolean
'see procedures above for general structure of procedure

Target.Index = "PrimaryKey"
If n < 2 Then
    ' can't draw graph for n=1 because area under curve ---> infinity
    ' and therefore I can't normalize it
    ' instead delete all records of a possible earlier graph
    Target.Seek ">=", Prior!ParamSet, Prior!ParamState, "Likelihood"
    Do Until Target.NoMatch
        If Target!ParamSet = Prior!ParamSet And Target!ParamState = Prior!ParamState _
            And Target!Title = "Likelihood" Then
            Target.delete
            Target.Seek ">=", Prior!ParamSet, Prior!ParamState, "Likelihood"
        Else
            Exit Do
        End If
    Loop
    Exit Sub
End If
LowerInfl = X / n * (1 - Sqr(1 / (n + 1))) ' lower point of inflection
Mode = X / n
UpperInfl = X / n * (1 + Sqr(1 / (n + 1))) ' upper point of inflection
Area = Fact(n - 2, 1) * X ^ (1 - n) ' area under curve
Target.Seek "=", Prior!ParamSet, Prior!ParamState, "Likelihood", 0
IsNew = Target.NoMatch
For Order = 0 To 9
    If IsNew Then
        Target.addnew
        Target!ParamSet = Prior!ParamSet
        Target!ParamState = Prior!ParamState
        Target!Title = "Likelihood"
    Else
        Target.Edit
    End If
    Target!Order = Order
    Select Case Order
        Case 0
            Target!X = X * ((n + 2) * Sqr(n + 1) - 2 * (n + 1)) / (n * n * Sqr(n + 1))
            ' point close to strongest curvature at lower end of curve
        Case 1
            Target!X = LowerInfl
        Case 2
            Target!X = (2 * LowerInfl + Mode) / 3
        Case 3
            Target!X = (LowerInfl + 2 * Mode) / 3
        Case 4
            Target!X = Mode
        Case 5
            Target!X = (2 * Mode + UpperInfl) / 3
        Case 6
            Target!X = UpperInfl
        Case 7
            Target!X = 2 * UpperInfl - Mode
        Case 8
            Target!X = 4 * UpperInfl - 3 * Mode
        Case 9
            Target!X = 8 * UpperInfl - 7 * Mode
    End Select
    Target!Y = ((1 / Target!X) ^ n) * Exp(-X / Target!X) / Area
    Target.Update
    If Not (IsNew Or (Order = 9)) Then
        Target.MoveNext
        If Target.EOF Then
            IsNew = True
        ElseIf Not ((Target!ParamSet = Prior!ParamSet) And (Target!ParamState = _
            Prior!ParamState) And (Target!Title = "Likelihood")) Then
```

```
        IsNew = True
    End If
End If
Next Order
End Sub

Public Sub MarkovCalculatePosterior(Prior As Recordset, Length() As Double, _
    X As Double, n As Integer)
    Static a1 As Double, b1 As Double, c1 As Double, Em As Double, Em2 As Double
    Static u1 As Double, u2 As Double, C As Double, model As Double, mode2 As Double
    Static Sa(3) As Double, Sc(3) As Double
    a1 = Prior!LengthMin
    b1 = Prior!LengthMode
    c1 = Prior!LengthMax
    If (c1 = a1) Or (n <= 0) Then ' problem denaturates
        Length(0) = a1
        Length(1) = b1
        Length(2) = c1
        Exit Sub
    End If
    If b1 >= 0 Then
        ' triangular distribution
        If b1 <= a1 Then
            u1 = 0
            b1 = a1
        Else
            u1 = 2 / ((c1 - a1) * (b1 - a1))
        End If
        If c1 <= b1 Then
            u2 = 0
            c1 = b1
        Else
            u2 = 2 / ((c1 - a1) * (c1 - b1))
        End If
        Aux n, b1, c1, X, Sc() ' Sc() will hold the result of the auxiliary function
        ' Sc(i) = S(n-i, p, q) from thesis where p = b1 and q = c1
        Aux n, a1, b1, X, Sa()
        C = 1 / (u1 * (Sa(1) - a1 * Sa(0)) - u2 * (Sc(1) - c1 * Sc(0))) ' N in thesis
        Em = C * (u1 * (Sa(2) - a1 * Sa(1)) - u2 * (Sc(2) - c1 * Sc(1))) ' E(m)
        Em2 = C * (u1 * (Sa(3) - a1 * Sa(2)) - u2 * (Sc(3) - c1 * Sc(2))) ' E(m^2)
        If n = 1 Then
            Length(1) = b1 ' if n=1 we can't use the formula below. The posterior mode
                            ' is however most likely the same as the prior one
        Else
            model=((X + n * a1) + Sqr(X * (X - (2 * n - 4) * a1) + (n * a1) ^ 2))/(2 * n - 2)
            ' model: mode of left part of correct analytic posterior curve
            mode2=((X + n * c1) - Sqr(X * (X - (2 * n - 4) * c1) + (n * c1) ^ 2))/(2 * n - 2)
            If b1 > model Then
                Length(1) = model
            ElseIf b1 < mode2 Then
                Length(1) = mode2
            Else
                Length(1) = b1
            End If
        End If
    End If
Else
    ' rectangular distribution
    If c1 > a1 Then
        u1 = 1 / (c1 - a1)
    Else
        Length(0) = a1 ' problem denaturates
        Length(2) = c1
        Exit Sub
    End If
    Aux n, a1, c1, X, Sa()
    C = 1 / (u1 * Sa(0)) ' N in thesis
    Em = C * (u1 * Sa(1)) ' E(m)
    Em2 = C * (u1 * Sa(2)) ' E(m^2)
```

```
Length(1) = X / n      ' the posterior mode is at the peak of the likelihood function
If Length(1) < a1 Then ' peak of likelihood function is below range of prior
    Length(1) = a1      ' --> mode is lower bound of prior
ElseIf Length(1) > c1 Then ' same for upper bound
    Length(1) = c1
End If
End If
Length(0) = 0.5 * (3 * Em - Length(1) - _
                Sqr(24 * (Em2 - Em ^ 2) - 3 * (Length(1) - Em) ^ 2))
If Length(0) > 0 Then
    Length(2) = 0.5 * (3 * Em - Length(1) + _
                    Sqr(24 * (Em2 - Em ^ 2) - 3 * (Length(1) - Em) ^ 2))
Else ' special case where min would be smaller than zero
    Length(0) = 0
    Length(1) = 0.5 * (3 * Em - Sqr(24 * Em2 - 27 * Em ^ 2))
    Length(2) = 0.5 * (3 * Em + Sqr(24 * Em2 - 27 * Em ^ 2))
End If
' round all numbers to 3 significant digits
Length(0) = Round(Length(0), 2 - Int(Log(Length(0)) / Log(10)))
Length(1) = Round(Length(1), 2 - Int(Log(Length(1)) / Log(10)))
Length(2) = Round(Length(2), 2 - Int(Log(Length(2)) / Log(10)))
End Sub

Public Function Fact(ByVal Numerator As Integer, ByVal Denominator As Integer) As Double
    ' this procedure calculates the value of a natural fraction with a
    ' factorial product both in the numerator and denominator
    Fact = 1
    If Numerator < 0 Or Denominator < 0 Then Exit Function
    If Numerator > Denominator Then
        Do Until Numerator <= Denominator ' for Numerator <= Denominator they cancel out
            Fact = Fact * CDBl(Numerator)
            Numerator = Numerator - 1
        Loop
    Else
        Do Until Denominator <= Numerator ' for Denominator <= Numerator they cancel out
            Fact = Fact / CDBl(Denominator)
            Denominator = Denominator - 1
        Loop
    End If
End Function

Public Function Aux(n As Integer, ByVal P As Double, q As Double, _
                  X As Double, S() As Double)
    Static k As Integer, r As Integer
    ' S(k) holds S(n - k, p, q) as described in the thesis
    If P >= q Then
        For k = 0 To 3
            S(k) = 0
        Next k
        Exit Function
    End If
    ' for p = 0 many functions are not defined, but f(0) = 0, therefore the error
    ' of introducing p = 0.0001 is negligible
    If P = 0 Then P = 0.0001
    ' there are two different ways of calculating S(i, p, q) for i < 1 and i > 1
    If n >= 5 Then ' n - k (k = 1...3) is always > 1
        S(3) = AuxPos(n - 3, P, q, X)
        k = 2
        Do Until k < 0 ' S(2), S(1) and S(0) are calculated the quickest from S(3)
            S(k) = S(k + 1) * (n - k - 2) / X + _
                Exp(-X / q) / (X * q ^ (n - k - 2)) - _
                Exp(-X / P) / (X * P ^ (n - k - 2))
            k = k - 1
        Loop
    Else
        If n > 1 Then
            ' part of S() involves the procedure to calculate S(i,p,q) where i > 1
```

```

    k = n - 2
    S(k) = AuxPos(2, P, q, X)
    Do Until k <= 0
        k = k - 1
        S(k) = S(k + 1) * (n - k - 2) / X + _
            Exp(-X / q) / (X * q ^ (n - k - 2)) - _
            Exp(-X / P) / (X * P ^ (n - k - 2))
    Loop
End If
k = n - 1
' calculate S(1,p,q)
S(k) = AuxZero(P, q, X)
' calculate all S(i,p,q) where i < 1 based on S(1,p,q)
Do Until k >= 3
    k = k + 1
    S(k) = Exp(-X / q) * (q ^ (1 - n + k) / (1 - n + k)) - _
        Exp(-X / P) * (P ^ (1 - n + k) / (1 - n + k)) - _
        X / (1 - n + k) * S(k - 1)
Loop
End If
End Function

Private Function AuxPos(r As Integer, P As Double, q As Double, X As Double) As Double
    Static k As Integer, Increment As Double, ExpQ As Double, ExpP As Double
    If r < 2 Then ' function not defined
        AuxPos = 0
        Exit Function
    End If
    ExpQ = Exp(-X / q)
    ExpP = Exp(-X / P)
    k = r - 2
    Do Until k < 0 ' the series is monotonically decreasing
        Increment = (Fact(r - 2, k) / (X ^ (r - 1 - k))) * (ExpQ / q ^ k - ExpP / P ^ k)
        AuxPos = AuxPos + Increment
        If Abs(Increment) < Abs(0.001 * AuxPos) Then Exit Function
        k = k - 1
    Loop
End Function

Private Function AuxZero(P As Double, q As Double, X As Double) As Double
    Static m As Double, Lastm As Double, Stepm As Double
    Static f As Double, Lastf As Double, Steps(4) As Double
    ' there is a closed form solution for this but it is numerically very unreliable
    ' and expensive. Here a simple trapezoidal integration is used. The step size is
    ' generally regular but sometimes additional supports are used to optimally capture
    ' the shape of the curve.
    AuxZero = 0
    If q <= P Then Exit Function
    Steps(0) = X * (3 - 2 * Sqr(2)) ' point near strongest curvature
    Steps(1) = X * (1 - Sqr(2) / 2) ' point of inflection
    Steps(2) = X ' maximum
    Steps(3) = X * (1 + Sqr(2) / 2) ' point of inflection
    Stepm = (q - P) / 100 ' the regular step size
    Lastm = P
    Lastf = Exp(-X / P) / P
    m = P
    Do Until False
        m = m + Stepm
        If m > q Then
            m = q
        ElseIf Steps(0) > Lastm And Steps(0) < m Then
            m = Steps(0)
        ElseIf Steps(1) > Lastm And Steps(1) < m Then
            m = Steps(1)
        ElseIf Steps(2) > Lastm And Steps(2) < m Then
            m = Steps(2)
        ElseIf Steps(3) > Lastm And Steps(3) < m Then

```



```
        m = Steps(3)
    End If
    f = Exp(-X / m) / m
    AuxZero = AuxZero + (m - Lastm) * (f + Lastf) / 2
    Lastf = f
    Lastm = m
    If m >= q Then Exit Do
Loop
End Function

Public Sub GraphVariableObserved(Source As Recordset, Target As Recordset, _
    Variable As Long, Timelimit As Double)
    Static Min As Double, Max As Double, No(11) As Integer, IsNew As Boolean
    Static C As Double, k As Integer, Bookmark As Variant, StepNo As Integer
    Source.Index = "PrimaryKey"
    Source.Seek ">=", Variable, Timelimit
    If Source.NoMatch Then Exit Sub
    If Source.EOF Then Exit Sub
    If Source(0).Value <> Variable Then Exit Sub
    ' No(k) holds the number of observations in the k-th interval. There are 10 intervals
    ' and two end points with y = 0 (No(0) and No(11))
    For k = 0 To 11
        No(k) = 0
    Next k
    C = 1 ' C is the normalizing constant to make the area under the curve = 1
    If Not Source.NoMatch Then
        C = 0 ' temporarily C holds the total number of observed values
        Max = 0
        Min = 10000000000#
        Bookmark = Source.Bookmark
        Do Until Source(0).Value <> Variable
            If Source!Value > Max Then Max = Source!Value
            If Source!Value < Min Then Min = Source!Value
            C = C + 1
            Source.MoveNext
            If Source.EOF Then Exit Do
        Loop
        If Max = Min Then Exit Sub
        StepNo = 1 + Int(Sqr(C))
        If StepNo > 10 Then StepNo = 10
        C = StepNo / ((Max - Min) * C) ' C depends on the number of observed values
        If Source.EOF Then Source.MoveFirst
        Source.Bookmark = Bookmark
        Do Until Source(0).Value <> Variable ' Source is sorted by Variable
            ' determine in which interval this value lies
            k = 1 + Int(StepNo * (Source!Value - Min) / (Max - Min))
            ' the Max value only will yield StepNo + 1 but belongs to k = StepNo
            If k > StepNo Then k = StepNo
            No(k) = No(k) + 1
            Source.MoveNext
            If Source.EOF Then Exit Do
        Loop
    End If
    Target.Index = "PrimaryKey"
    ' look for existing records compare also to other procedures that create graphs
    Target.Seek "=", Variable, "Observed", 0
    If Target.NoMatch Then
        IsNew = True
    Else
        IsNew = False
    End If
    For k = 0 To 11
        If k > StepNo + 1 And Not IsNew Then
            Target.delete
        ElseIf k <= StepNo + 1 Then
            If IsNew Then
                Target.addnew
                Target!Variable = Variable
            End If
        End If
    Next k
End Sub
```

```
        Target!Title = "Observed"
    Else
        Target.Edit
    End If
    Target!Order = k
    Target!X = Min + (Max - Min) * (k - 0.5) / StepNo ' interval middle
    ' note that Min is the left BOUNDARY of the 1st interval, and Max the
    ' right boundary of the 10th interval
    Target!Y = C * No(k)
    Target.Update
End If
If Not (IsNew Or (k = 11)) Then
    Target.MoveNext
    If Target.EOF Then
        IsNew = True
    ElseIf Not (Target!Variable = Variable And Target!Title = "Observed") Then
        IsNew = True
    End If
End If
Next k
End Sub

Public Sub GraphsTunDelaySet(DB As Database, Prior As Boolean)
    Static Source As Recordset, Target As Recordset, Title As String
    Set Source = DB.OpenRecordset("qTunDelaysPriorPost", dbOpenDynaset, dbReadOnly)
    Set Target = DB.OpenRecordset("otTunDelaysGraph", dbOpenTable)
    Target.Index = "PrimaryKey"
    Do Until Source.EOF
        ' calculate graph only if the status of the record is as specified by Prior
        If (Source!Status = "Prior" And Prior) Or _
            (Source!Status <> "Prior" And Not Prior) Then _
            GraphTunDelayIndividual Source, Target
        Source.MoveNext
    Loop
    Source.Close
    Target.Close
End Sub

Public Sub GraphTunDelayIndividual(Source As Recordset, Target As Recordset)
    Static Order As Integer, IsNew As Boolean, OldStatus As String, Status As String
    ' try to find existing records, compare also to other procedures creating graphs
    Target.Index = "PrimaryKey"
    Status = Source!Status
    If Status = "Prior" Then
        Target.Seek "=", Source!Delay, "Prior", 0
    Else
        ' if the new status is not "Prior" it may replace an existing graph that has
        ' a different Status/Title, i.e. we look for records with any of the following
        ' titles
        Target.Seek "=", Source!Delay, "Observed", 0
        If Target.NoMatch Then Target.Seek "=", Source!Delay, "Current", 0
        If Target.NoMatch Then Target.Seek "=", Source!Delay, "Posterior", 0
    End If
    If Target.NoMatch Then
        OldStatus = Status
    Else
        OldStatus = Target!Title ' we have to remember the old Status to find the end of
        ' the existing set of records
    End If
    IsNew = Target.NoMatch
    For Order = 0 To 6
        ' the last point is always "empty" (X = Null, Y = Null) to separate the
        ' individual graphs
        If Source!Type = 0 And Status = "Observed" And Order > 5 Then
            ' rectangle --> 5 points (first = last to close box) plus empty point
            If Not IsNew Then Target.delete
            ElseIf Source!Type = 1 And Status = "Observed" And Order > 2 Then
```

```
' 2 point line --> 2 points plus empty point
If Not IsNew Then Target.delete
ElseIf Source!Type = 1 And Status <> "Observed" And Order > 4 Then
' 4 point line --> 4 points plus empty point
If Not IsNew Then Target.delete
Else
If IsNew Then
Target.addnew
Target!Delay = Source!Delay
Target!Title = Status
Else
Target.Edit
End If
Target!Type = Source!Type
Target!Title = Status
Target!Order = Order
If Source!Type = 0 Then
' time-dependent delay
Select Case Order
Case 0
Target!X = Source!Start
Target!Y = 1
Case 1
Target!X = Source!Start
Target!Y = 3
Case 2
Target!X = Source!Start + Source!DurationMin
Target!Y = 3
Case 3
If Status = "Observed" Then ' no middle point (DurationMode)
Target!X = Source!Start + Source!DurationMax
Target!Y = 1
Else
Target!X = Source!Start + Source!DurationMode
Target!Y = 2
End If
Case 4
If Status = "Observed" Then
Target!X = Source!Start
Target!Y = 1
Else
Target!X = Source!Start + Source!DurationMax
Target!Y = 1
End If
Case 5
If Status = "Observed" Then
Target!X = Null
Target!Y = Null
Else
Target!X = Source!Start
Target!Y = 1
End If
Case Else
Target!X = Null
Target!Y = Null
End Select
Else
' position-dependent delay
Target!X = Source!Start
Select Case Order
Case 0
Target!Y = 0
Case 1
Target!Y = Source!DurationMin
Case 2
If Status = "Observed" Then ' no min and max points
Target!X = Null
Target!Y = Null
```

```
        Else
            Target!Y = Source!DurationMode
        End If
    Case 3
        Target!Y = Source!DurationMax
    Case Else
        Target!X = Null
        Target!Y = Null
    End Select
End If
Target.Update
End If
If Not (IsNew Or (Order = 6)) Then
    Target.MoveNext
    If Target.EOF Then
        IsNew = True
    ElseIf (Target!Delay <> Source!Delay) Or (Target!Title <> OldStatus) Then
        IsNew = True
    End If
End If
Next Order
End Sub

Public Sub GraphChangeDelaysObserved(Source As Recordset, Target As Recordset, _
    FromMethod As Long, ToMethod As Long)
    Static Min As Double, Max As Double, No(11) As Integer, IsNew As Boolean
    Static C As Double, k As Integer, Bookmark As Variant, StepNo As Integer
    Source.Index = "PrimaryKey"
    Source.Seek ">=", FromMethod, ToMethod
    If Source.NoMatch Then Exit Sub
    If Source.EOF Then Exit Sub
    If Not (Source!FromMethod = FromMethod And Source!ToMethod = ToMethod) Then Exit Sub
    ' No(k) holds the number of observations in the k-th interval. There are 10 intervals
    ' and two end points with y = 0 (No(0) and No(11))
    For k = 0 To 11
        No(k) = 0
    Next k
    C = 1 ' C is the normalizing constant to make the area under the curve = 1
    If Not Source.NoMatch Then
        C = 0 ' temporarily C holds the total number of observed values
        Max = 0
        Min = 10000000000#
        Bookmark = Source.Bookmark
        Do Until Not (Source!FromMethod = FromMethod And Source!ToMethod = ToMethod)
            If Source!Delay > Max Then Max = Source!Delay
            If Source!Delay < Min Then Min = Source!Delay
            C = C + 1
            Source.MoveNext
            If Source.EOF Then Exit Do
        Loop
        If Max = Min Then
            Min = Max * 0.5
            Max = Max * 1.5
        End If
        ' number of steps should not be too large if there is little data
        StepNo = 1 + Int(Sqr(C))
        If StepNo > 10 Then StepNo = 10
        C = StepNo / ((Max - Min) * C) ' C depends on the number of observed values
        If Source.EOF Then Source.MoveFirst
        Source.Bookmark = Bookmark
        Do Until Not (Source!FromMethod = FromMethod And Source!ToMethod = ToMethod)
            ' determine in which interval this value lies
            k = 1 + Int(StepNo * (Source!Delay - Min) / (Max - Min))
            If k > StepNo Then k = StepNo 'the Max value only will yield 11 but belongs to k=10
            No(k) = No(k) + 1
            Source.MoveNext
            If Source.EOF Then Exit Do
        Loop
    End If
End Sub
```

```
End If
Target.Index = "PrimaryKey"
' look for existing records compare also to other procedures that create graphs
Target.Seek "=", FromMethod, ToMethod, 0
If Target.NoMatch Then
    IsNew = True
Else
    IsNew = False
End If
For k = 0 To 11
    If k > StepNo + 1 And Not IsNew Then
        Target.delete
    ElseIf k <= StepNo + 1 Then
        If IsNew Then
            Target.addnew
            Target!FromMethod = FromMethod
            Target!ToMethod = ToMethod
        Else
            Target.Edit
        End If
        Target!Order = k
        Target!X = Min + (Max - Min) * (k - 0.5) / StepNo ' interval middle
        ' note that Min is the left BOUNDARY of the 1st interval, and Max the
        ' right boundary of the 10th interval
        Target!Y = C * No(k)
        Target.Update
    End If
    If Not (IsNew Or (k = 11)) Then
        Target.MoveNext
        If Target.EOF Then
            IsNew = True
        ElseIf Not (Target!FromMethod = FromMethod And Target!ToMethod = ToMethod) Then
            IsNew = True
        End If
    End If
Next k
End Sub

Public Sub GraphTimeCostFrequency(UpdatePhase As Long, DB As Database)
    Dim MinTime As Double, MaxTime As Double, NoTime() As Integer, NoCost() As Integer
    Dim MinCost As Double, MaxCost As Double, k As Integer, n As Integer
    Dim Source As Recordset, Stat As Recordset, Target As Recordset, Bookmark As Variant

    Set Source = DB.OpenRecordset("rtTimeCost", dbOpenTable, dbReadOnly)
    Set Stat = DB.OpenRecordset("rqTimeCostStat", dbOpenSnapshot)
    Set Target = DB.OpenRecordset("rtTimeCostGraph", dbOpenTable)
    Stat.FindFirst "ID = " & CStr(UpdatePhase)
    If Stat.NoMatch Then Exit Sub
    If IsNull(Stat!Number) Then Exit Sub ' there are no data for this update phase
    Source.Index = "UpdatePhase"
    Target.Index = "PrimaryKey"
    Source.Seek ">=", UpdatePhase
    ' NoTime(k) and NoCost(k) hold the number of observations in the k-th interval.
    ' The number of intervals is chosen as the square root of the number of data points.
    ' Two more points are needed as end points with y = 0 (No(0) and No(n+1))
    n = Int(Sqr(Stat!Number))
    ReDim NoTime(n + 1)
    ReDim NoCost(n + 1)
    For k = 0 To n + 1
        NoTime(k) = 0
        NoCost(k) = 0
    Next k
    MaxTime = 0
    MaxCost = 0
    MinTime = 10000000000#
    MinCost = 10000000000#
    Bookmark = Source.Bookmark
    Do Until Source!UpdatePhase <> UpdatePhase
```

```
If Source!Time > MaxTime Then MaxTime = Source!Time
If Source!Time < MinTime Then MinTime = Source!Time
If Source!Cost > MaxCost Then MaxCost = Source!Cost
If Source!Cost < MinCost Then MinCost = Source!Cost
Source.MoveNext
If Source.EOF Then Exit Do
Loop
If Source.EOF Then Source.MoveFirst
Source.Bookmark = Bookmark
If MaxTime = MinTime Then
    MinTime = MaxTime * 0.5
    MaxTime = MaxTime * 1.5
End If
If MaxCost = MinCost Then
    MinCost = MaxCost * 0.5
    MaxCost = MaxCost * 1.5
End If
Do Until Source!UpdatePhase <> UpdatePhase
    ' determine in which interval this value lies
    k = 1 + Int(n * (Source!Time - MinTime) / (MaxTime - MinTime))
    If k > n Then k = n ' the Max value only will yield n+1 but belongs to k = n
    NoTime(k) = NoTime(k) + 1
    k = 1 + Int(n * (Source!Cost - MinCost) / (MaxCost - MinCost))
    If k > n Then k = n
    NoCost(k) = NoCost(k) + 1
    Source.MoveNext
    If Source.EOF Then Exit Do
Loop
For k = 0 To n + 1
    Target.addnew
    Target!UpdatePhase = UpdatePhase
    Target!Title = "Time"
    Target!X = MinTime + (MaxTime - MinTime) * (k - 0.5) / n ' interval middle
    ' note that Min is the left BOUNDARY of the 1st interval, and Max the
    ' right boundary of the n-th interval
    Target!Y = NoTime(k)
    Target.Update
    Target.addnew
    Target!UpdatePhase = UpdatePhase
    Target!Title = "Cost"
    Target!X = MinCost + (MaxCost - MinCost) * (k - 0.5) / n
    Target!Y = NoCost(k)
    Target.Update
Next k
Source.Close
Stat.Close
Target.Close
End Sub
```

Module: Import

```
Option Compare Database
Option Explicit
Dim Path As String
Dim DB As Database, FS As Object
Dim ImpExp1 As Recordset, ImpExp2 As Recordset
Dim pointer1 As Long, pointer2 As Long, pointer3 As Long, pointer4 As Long
Dim pointerArr1() As Long

Public Sub Import()
    Dim Title As String, UP As Recordset
    Title = "IMPORT SIMULATOR INPUT DATA"
    If MsgBox("This action will delete ALL data of this project!" & Chr(13) & _
        "Do you want to continue?", vbYesNo + vbQuestion, Title) _
        = vbNo Then Exit Sub
    Set DB = CurrentDb()
    Set FS = CreateObject("Scripting.FileSystemObject")
```

```
Set UP = DB.OpenRecordset("tUpdatePhases", dbOpenTable)
UP.Index = "PrimaryKey"
UP.Seek "=", 1
If IsNull(UP!Path) Then
    UP.Edit
    UP!Path = " "
    UP.Update
End If
Do Until False
    Path = ""
    Path = InputBox("Import data from the following directory:", _
        Title, UP!Path)
    If Path = "" Then Exit Sub
    If Not FS.FolderExists(Path) Then
        MsgBox "The directory you have chosen does not exist.", vbOKOnly, Title
    Else
        Exit Do
    End If
Loop
UP.Edit
UP!Path = Path
UP.Update
UP.Close
DoCmd.OpenForm "BePatient", acNormal
DoCmd.RepaintObject acForm, "BePatient"
DoCmd.Hourglass True
DoCmd.Echo False, "Working hard...."
DoCmd.SetWarnings False
DoCmd.RunMacro "Delete All Data"
ImpGroundParam
ImpTunnel
ImpConstrParam
DB.Close
DoCmd.SetWarnings True
DoCmd.Echo True
DoCmd.Hourglass False
DoCmd.Close acForm, "BePatient", acSaveNo
End Sub

Private Sub ImpGroundParam()
    ImpParameters
    ImpParamSets
    ImpGC
    ImpCorrel
    ImpBoreholes
    ImpCGC
    ImpGeom
    ImpAreaZone
End Sub

Private Sub ImpTunnel()
    ImpResources
    ImpRepositories
    ImpTunnelGraph
End Sub

Private Sub ImpConstrParam()
    ImpMethod
    ImpChangeDelays
    ImpLearnParam
    ImpGCGeomMeth
    ImpGenVar
End Sub

Private Sub ImpParameters()
    Dim Parameters As Recordset, ParamStates As Recordset
    Dim ParamNo As Integer, ParamStatNo As Integer, ParamID As Long
```

```
Dim MaxParamNo As Integer, MaxParamStatNo As Integer
DB.Execute ("ImpExplDel")
DoCmd.TransferText acImportFixed, "Text ImpSpec", "ImpExpl", _
    Path & "\param_stat_save", 0
Set ImpExpl = DB.OpenRecordset("ImpExpl", dbOpenTable, dbReadOnly)
ImpExpl.Index = "PrimaryKey"
Set Parameters = DB.OpenRecordset("tParameters", dbOpenTable, dbAppendOnly)
Set ParamStates = DB.OpenRecordset("tParamStates", dbOpenTable, dbAppendOnly)
ReDim pointerArr1(CInt(ImpExpl!F1)) As Long
' pointerArr1 remembers the ID of parameter states once they are created in
' the database
MaxParamNo = CInt(ImpExpl!F1)
ImpExpl.MoveNext
For ParamNo = 0 To MaxParamNo - 1
    Parameters.addnew
    Parameters!Name = ImpExpl!F1
    ImpExpl.MoveNext
    ParamID = Parameters!ID
    Parameters.Update
    ' pointer1 = ID of first parameter record
    If ParamNo = 0 Then pointer1 = ParamID
    MaxParamStatNo = CInt(ImpExpl!F1)
    ImpExpl.MoveNext
    For ParamStatNo = 0 To MaxParamStatNo - 1
        ParamStates.addnew
        ParamStates!Name = ImpExpl!F1
        ImpExpl.MoveNext
        ParamStates!Parameter = ParamID
        ' remember ID of first parameter state of every parameter
        If ParamStatNo = 0 Then pointerArr1(ParamNo) = ParamStates!ID
        ParamStates.Update
    Next ParamStatNo
Next ParamNo
If Not ImpExpl.EOF Then
    MsgBox ("file param_stat_save is longer than expected")
End If
Parameters.Close
ParamStates.Close
ImpExpl.Close
End Sub

Private Sub ImpParamSets()
    Dim ParamStates As Recordset, ParamSets As Recordset, ParamXSet As Recordset
    Dim MarkovLengths As Recordset, MarkovTransProb As Recordset, ParamStatNo As Integer
    Dim DeterSegs As Recordset, SemiDeterSegs As Recordset, SegProb As Recordset
    Dim ParamNo As Integer, ParamSetNo As Integer, ParamCounter As Integer
    Dim MaxParamNo As Integer, MaxParamStatNo As Integer, MaxParamSetNo As Integer
    Dim FromParamStatNo As Integer, MaxFromParamStatNo As Integer
    Dim DeterSegNo As Integer, MaxDeterSegNo As Integer, GenMode As String
    Dim SemiDeterSegNo As Integer, MaxSemiDeterSegNo As Integer, SDSID As Long
    DB.Execute ("ImpExplDel")
    DoCmd.TransferText acImportFixed, "Text ImpSpec", "ImpExpl", Path & "\ground_save", 0
    Set ImpExpl = DB.OpenRecordset("ImpExpl", dbOpenTable, dbReadOnly)
    ImpExpl.Index = "PrimaryKey"
    Set ParamSets = DB.OpenRecordset("tParamSets", dbOpenTable, dbAppendOnly)
    Set ParamXSet = DB.OpenRecordset("tParam_X_Set", dbOpenTable)
    Set MarkovLengths = DB.OpenRecordset("tMarkovLengths", dbOpenTable)
    Set MarkovTransProb = DB.OpenRecordset("tMarkovTransProb", dbOpenTable, dbAppendOnly)
    Set DeterSegs = DB.OpenRecordset("tDeterSegs", dbOpenTable, dbAppendOnly)
    MaxParamSetNo = CInt(ImpExpl!F1)
    ImpExpl.MoveNext
    For ParamSetNo = 0 To MaxParamSetNo - 1
        ParamSets.addnew
        ParamSets!ID = CLng(ParamSetNo)
        ParamSets.Update
        MaxParamNo = CInt(ImpExpl!F1)
        ImpExpl.MoveNext
        For ParamCounter = 0 To MaxParamNo - 1
```



```
ParamNo = CInt(ImpExp1!F1)
ImpExp1.MoveNext
ParamXSet.addnew
ParamXSet!ParamSet = ParamSetNo
ParamXSet!Parameter = pointer1 + ParamNo
MaxParamStatNo = CInt(ImpExp1!F1)
MaxFromParamStatNo = MaxParamStatNo
ImpExp1.MoveNext
GenMode = ImpExp1!F1
ParamXSet!GenerationMode = GenMode
ImpExp1.MoveNext
If GenMode = "0" Then ' Markov Generation Mode
    For ParamStatNo = 0 To MaxParamStatNo - 1
        MarkovLengths.addnew
        MarkovLengths!ParamSet = ParamSetNo
        MarkovLengths!ParamState = pointerArr1(ParamNo) + CLng(ParamStatNo)
        MarkovLengths!LengthMin = CDb1(ImpExp1!F1)
        ImpExp1.MoveNext
        MarkovLengths!LengthMode = CDb1(ImpExp1!F1)
        ImpExp1.MoveNext
        MarkovLengths!LengthMax = CDb1(ImpExp1!F1)
        ImpExp1.MoveNext
        MarkovLengths.Update
    Next ParamStatNo
    For FromParamStatNo = 0 To MaxFromParamStatNo - 1
        For ParamStatNo = 0 To MaxParamStatNo - 1
            MarkovTransProb.addnew
            MarkovTransProb!ParamSet = ParamSetNo
            MarkovTransProb!FromParamState = pointerArr1(ParamNo) _
                + CLng(FromParamStatNo)
            MarkovTransProb!ToParamState = pointerArr1(ParamNo) + CLng(ParamStatNo)
            MarkovTransProb!Prob = CDb1(ImpExp1!F1)
            MarkovTransProb.Update
        Next ParamStatNo
    Next FromParamStatNo
    For ParamStatNo = 0 To MaxParamStatNo - 1
        MarkovLengths.Index = "PrimaryKey"
        MarkovLengths.Seek ">=", ParamSetNo, pointerArr1(ParamNo) _
            + CLng(ParamStatNo)
        'pointerArr1(ParamNo) is the ID of the first param state of param. ParamNo
        MarkovLengths.Edit
        MarkovLengths!StartProb = CDb1(ImpExp1!F1)
        ImpExp1.MoveNext
        MarkovLengths.Update
    Next ParamStatNo
Else
    ImpExp1.Move (MaxParamStatNo * (MaxParamStatNo + 4))
End If
ParamXSet!StartProbCorr = ImpExp1!F1
ImpExp1.MoveNext
ParamXSet!DeterReference = ImpExp1!F1
ImpExp1.MoveNext
ParamXSet.Update
MaxDeterSegNo = CInt(ImpExp1!F1)
ImpExp1.MoveNext
For DeterSegNo = 0 To MaxDeterSegNo - 1
    DeterSegs.addnew
    DeterSegs!ParamSet = ParamSetNo
    DeterSegs!ParamState = pointerArr1(ParamNo) + CLng(ImpExp1!F1)
    ImpExp1.MoveNext
    DeterSegs!EndPos = CDb1(ImpExp1!F1)
    ImpExp1.MoveNext
    DeterSegs.Update
Next DeterSegNo
Next ParamCounter
Next ParamSetNo
If Not ImpExp1.EOF Then
    MsgBox ("file ground_save is longer than expected")
```

```
End If
ParamSets.Close
MarkovLengths.Close
MarkovTransProb.Close
DeterSegs.Close
ImpExpl.Close
DB.Execute ("ImpExplDel")
DoCmd.TransferText acImportFixed, "Text ImpSpec", "ImpExpl", Path & "\deter_save", 0
Set ImpExpl = DB.OpenRecordset("ImpExpl", dbOpenTable, dbReadOnly)
ImpExpl.Index = "PrimaryKey"
Set SemiDeterSegs = DB.OpenRecordset("tSemiDeterSegs", dbOpenTable, dbAppendOnly)
Set SegProb = DB.OpenRecordset("tSegProb", dbOpenTable, dbAppendOnly)
Set ParamStates = DB.OpenRecordset("tParamStates", dbOpenSnapshot)
ParamXSet.Index = "PrimaryKey"
For ParamSetNo = 0 To MaxParamSetNo - 1
    ParamXSet.Seek ">=", ParamSetNo
    If Not ParamXSet.EOF Then
        Do While ParamXSet!ParamSet = ParamSetNo
            MaxSemiDeterSegNo = CInt(ImpExpl!F1)
            ImpExpl.MoveNext
            For SemiDeterSegNo = 0 To MaxSemiDeterSegNo - 1
                SemiDeterSegs.addnew
                SemiDeterSegs!ParamSet = ParamSetNo
                ParamStates.FindFirst "Parameter = " & CStr(ParamXSet!Parameter)
                ParamStates.Move (CLng(ImpExpl!F1))
                ImpExpl.MoveNext
                SemiDeterSegs!ParamState = ParamStates!ID
                SemiDeterSegs!Order = CDBl(SemiDeterSegNo)
                SDSID = SemiDeterSegs!ID
                SemiDeterSegs.Update
                SegProb.addnew
                SegProb!Segment = SDSID
                SegProb!GenerationMode = ImpExpl!F1
                ImpExpl.MoveNext
                If SegProb!GenerationMode = "0" Then
                    SegProb!LengthMin = CDBl(ImpExpl!F1)
                    ImpExpl.MoveNext
                    SegProb!LengthMode = CDBl(ImpExpl!F1)
                    ImpExpl.MoveNext
                    SegProb!LengthMax = CDBl(ImpExpl!F1)
                    ImpExpl.MoveNext
                    SegProb!LengthProbMin = CDBl(ImpExpl!F1)
                    ImpExpl.MoveNext
                    SegProb!LengthProbMax = CDBl(ImpExpl!F1)
                    ImpExpl.MoveNext
                Else
                    SegProb!EndposMin = CDBl(ImpExpl!F1)
                    ImpExpl.MoveNext
                    SegProb!EndposMode = CDBl(ImpExpl!F1)
                    ImpExpl.MoveNext
                    SegProb!EndposMax = CDBl(ImpExpl!F1)
                    ImpExpl.MoveNext
                    SegProb!EndposProbMin = CDBl(ImpExpl!F1)
                    ImpExpl.MoveNext
                    SegProb!EndposProbMax = CDBl(ImpExpl!F1)
                    ImpExpl.MoveNext
                End If
                SegProb.Update
            Next SemiDeterSegNo
            ParamXSet.MoveNext
            If ParamXSet.EOF Then Exit Do
        Loop
    End If
Next ParamSetNo
If Not ImpExpl.EOF Then
    MsgBox ("file deter_save is longer than expected")
End If
SemiDeterSegs.Close
```

```
SegProb.Close
ParamStates.Close
ParamXSet.Close
ImpExpl.Close
End Sub

Private Sub ImpGC()
    Dim ParamXSet As Recordset, GC As Recordset, GCCombXStates As Recordset
    Dim GCCombNo As Integer, MaxGCCombNo As Integer, ParamCounter As Long, GCID As Long
    Dim CurrentParamSet As Long, GCCombID As Long, GCCombinations As Recordset
    Set GC = DB.OpenRecordset("tGC", dbOpenTable, dbAppendOnly)
    Set GCCombinations = DB.OpenRecordset("tGCCombinations", dbOpenTable)
    Set GCCombXStates = DB.OpenRecordset("tGCComb_X_States", dbOpenTable, dbAppendOnly)
    Set ParamXSet = DB.OpenRecordset("eParamSetParam", dbOpenDynaset, dbReadOnly)

    DB.Execute ("ImpExplDel")
    DoCmd.TransferText acImportFixed, "Text ImpSpec", "ImpExpl", Path & "\gc_save", 0
    Set ImpExpl = DB.OpenRecordset("ImpExpl", dbOpenTable, dbReadOnly)
    ImpExpl.Index = "PrimaryKey"
    ParamXSet.MoveFirst
    Do Until ParamXSet.EOF
        MaxGCCombNo = CInt(ImpExpl!F1)
        ImpExpl.MoveNext
        If MaxGCCombNo = 0 Then
            ParamCounter = ParamXSet!ParamSet
            Do Until ParamXSet!ParamSet <> ParamCounter
                ParamXSet.MoveNext
                If ParamXSet.EOF Then Exit Do
            Loop
        End If
        For GCCombNo = 0 To MaxGCCombNo - 1
            CurrentParamSet = ParamXSet!ParamSet
            GC.Index = "Name"
            GC.Seek "=", ImpExpl!F1
            If GC.NoMatch Then
                GC.addnew
                GC!Name = ImpExpl!F1
                GCID = GC!ID
                GC.Update
            Else
                GCID = GC!ID
            End If
            ImpExpl.MoveNext
            GCCombinations.addnew
            GCCombinations!ParamSet = CurrentParamSet
            GCCombinations!GC = GCID
            GCCombID = GCCombinations!ID
            GCCombinations.Update
            ParamCounter = 0
            Do While (ParamXSet!ParamSet = CurrentParamSet)
                ' we scan through every parameter of this parameter set and read which
                ' parameter state constitutes this GC. if param state < 0 then it does
                ' not matter. the param state is identified by its number relative to this
                ' parameter. they are not numbered continuously like in the database
                If (CLng(ImpExpl!F1) >= 0) Then
                    GCCombXStates.addnew
                    GCCombXStates!GCComb = GCCombID
                    ' pointerArr1(k) = ID of 1st parameter state of k-th parameter
                    GCCombXStates!ParamState = pointerArr1(ParamXSet!ParamID - pointer1) _
                        + CLng(ImpExpl!F1)
                    GCCombXStates.Update
                End If
                ImpExpl.MoveNext
                ParamXSet.MoveNext
                ParamCounter = ParamCounter + 1
                If ParamXSet.EOF Then Exit Do
            Loop
            ' go back to first parameter of parameter set
        Next
    Loop
End Sub
```

```
        If Not GCCombNo = MaxGCCombNo - 1 Then ParamXSet.Move (-ParamCounter)
    Next GCCombNo
Loop
If Not ImpExpl.EOF Then
    MsgBox ("file gc_save is longer than expected")
End If
ParamXSet.Close
GC.Close
GCCombinations.Close
GCCombXStates.Close
ImpExpl.Close
End Sub

Private Sub ImpCorrel()
    Dim ParamXSet As Recordset, Correlations As Recordset, CorrelXStates As Recordset
    Dim CorrelNo As Integer, MaxCorrelNo As Integer, ParamCounter As Long
    Dim CurrentParamSet As Long, CorrelID As Long
    Set Correlations = DB.OpenRecordset("tCorrelations", dbOpenTable, dbAppendOnly)
    Set CorrelXStates = DB.OpenRecordset("tCorrel_X_States", dbOpenTable, dbAppendOnly)
    Set ParamXSet = DB.OpenRecordset("eParamSetParam", dbOpenDynaset, dbReadOnly)
    DB.Execute ("ImpExplDel")
    DoCmd.TransferText acImportFixed, "Text ImpSpec", "ImpExpl", Path & "\gc_proba_save", 0
    Set ImpExpl = DB.OpenRecordset("ImpExpl", dbOpenTable, dbReadOnly)
    ImpExpl.Index = "PrimaryKey"
    ParamXSet.MoveFirst
    Do Until ParamXSet.EOF
        MaxCorrelNo = CInt(ImpExpl!F1)
        ImpExpl.MoveNext
        If MaxCorrelNo = 0 Then
            ParamCounter = ParamXSet!ParamSet
            Do Until ParamXSet!ParamSet <> ParamCounter
                ParamXSet.MoveNext
            If ParamXSet.EOF Then Exit Do
        Loop
    End If
    For CorrelNo = 0 To MaxCorrelNo - 1
        CurrentParamSet = ParamXSet!ParamSet
        Correlations.addnew
        Correlations!ParamSet = CurrentParamSet
        Correlations!Value = CDbl(ImpExpl!F1)
        CorrelID = Correlations!ID
        ImpExpl.MoveNext
        Correlations.Update
        ParamCounter = 0
        Do While (ParamXSet!ParamSet = CurrentParamSet)
            If (CLng(ImpExpl!F1) >= 0) Then
                CorrelXStates.addnew
                CorrelXStates!Correlation = CorrelID
                ' pointerArr1(k) = ID of 1st parameter state of k-th parameter
                CorrelXStates!ParamState = pointerArr1(ParamXSet!ParamID - pointer1) _
                    + CLng(ImpExpl!F1)
                CorrelXStates.Update
            End If
            ImpExpl.MoveNext
            ParamXSet.MoveNext
            ParamCounter = ParamCounter + 1
            If ParamXSet.EOF Then Exit Do
        Loop
        ' go back to 1st parameter of this parameter set
        If Not CorrelNo = MaxCorrelNo - 1 Then ParamXSet.Move (-ParamCounter)
    Next CorrelNo
Loop
If Not ImpExpl.EOF Then
    MsgBox ("file gc_proba_save is longer than expected")
End If
ParamXSet.Close
Correlations.Close
CorrelXStates.Close
```

```
ImpExpl.Close
End Sub

Private Sub ImpBoreholes()
    Dim ParamXSet As Recordset, Boreholes As Recordset, BHXStates As Recordset
    Dim BHNo As Integer, MaxBHNo As Integer, ParamStateNo As Integer
    Dim MaxParamStateNo As Integer, ParamCounter As Long
    Dim CurrentParamSet As Long, BHID As Long
    Set Boreholes = DB.OpenRecordset("tBoreholes", dbOpenTable, dbAppendOnly)
    Set BHXStates = DB.OpenRecordset("tBH_X_States", dbOpenTable, dbAppendOnly)
    Set ParamXSet = DB.OpenRecordset("eParamSetParam", dbOpenDynaset, dbReadOnly)
    DB.Execute ("ImpExplDel")
    DoCmd.TransferText acImportFixed, "Text ImpSpec", "ImpExpl", Path & "\bh_save", 0
    Set ImpExpl = DB.OpenRecordset("ImpExpl", dbOpenTable, dbReadOnly)
    ImpExpl.Index = "PrimaryKey"
    ParamXSet.MoveFirst
    ImpExpl.MoveNext
    Do Until ParamXSet.EOF
        MaxBHNo = CInt(ImpExpl!F1)
        ImpExpl.MoveNext
        If MaxBHNo = 0 Then
            ParamCounter = ParamXSet!ParamSet
            ' move to first parameter of next parameter set
            Do Until ParamXSet!ParamSet <> ParamCounter
                ParamXSet.MoveNext
            If ParamXSet.EOF Then Exit Do
        Loop
    End If
    For BHNo = 0 To MaxBHNo - 1
        CurrentParamSet = ParamXSet!ParamSet
        Boreholes.addnew
        Boreholes!ParamSet = CurrentParamSet
        Boreholes!Name = ImpExpl!F1
        ImpExpl.MoveNext
        Boreholes!Pos = CDBl(ImpExpl!F1)
        ImpExpl.MoveNext
        Boreholes!Likelihood = CBool(ImpExpl!F1)
        ImpExpl.Move (2)
        BHID = Boreholes!ID
        Boreholes.Update
        ParamCounter = 0
        Do While (ParamXSet!ParamSet = CurrentParamSet)
            MaxParamStateNo = CInt(ImpExpl!F1)
            ImpExpl.MoveNext
            For ParamStateNo = 0 To MaxParamStateNo - 1
                BHXStates.addnew
                BHXStates!Borehole = BHID
                ' pointerArr1(k) = ID of 1st parameter state of k-th parameter
                BHXStates!ParamState = pointerArr1(ParamXSet!ParamID - pointer1) _
                    + CLng(ParamStateNo)
                BHXStates!Prob = CDBl(ImpExpl!F1)
                ImpExpl.MoveNext
                BHXStates.Update
            Next ParamStateNo
            ParamXSet.MoveNext
            ParamCounter = ParamCounter + 1
            If ParamXSet.EOF Then Exit Do
        Loop
        ' go back to first parameter of this parameter set
        If Not BHNo = MaxBHNo - 1 Then ParamXSet.Move (-ParamCounter)
    Next BHNo
Loop
If Not ImpExpl.EOF Then
    MsgBox ("file bh_save is longer than expected")
End If
ParamXSet.Close
Boreholes.Close
BHXStates.Close
```

```
ImpExp1.Close  
End Sub
```

```
Private Sub ImpCGC()  
    Dim CGC As Recordset, GC As Recordset  
    Dim CGCNo As Integer, MaxCGCNo As Integer  
    Set CGC = DB.OpenRecordset("tCombGC", dbOpenTable, dbAppendOnly)  
    Set GC = DB.OpenRecordset("tGC", dbOpenSnapshot)  
    DB.Execute ("ImpExp1Del")  
    DoCmd.TransferText acImportFixed, "Text ImpSpec", "ImpExp1", Path & "\cb_save", 0  
    Set ImpExp1 = DB.OpenRecordset("ImpExp1", dbOpenTable, dbReadOnly)  
    ImpExp1.Index = "PrimaryKey"  
    MaxCGCNo = CInt(ImpExp1!F1)  
    ImpExp1.MoveNext  
    For CGCNo = 0 To MaxCGCNo - 1  
        CGC.addnew  
        GC.FindFirst "Name = '" & ImpExp1!F1 & "'"   
        If GC.NoMatch Then  
            MsgBox ("Cannot find combined groundclass " & ImpExp1!F1)  
        End If  
        CGC!CombGC = GC!ID  
        ImpExp1.MoveNext  
        If ImpExp1!F1 <> "" Then  
            GC.FindFirst "Name = '" & ImpExp1!F1 & "'"   
            If GC.NoMatch Then  
                MsgBox ("Cannot find groundclass " & ImpExp1!F1)  
            End If  
            CGC!AreaGC = GC!ID  
        End If  
        ImpExp1.MoveNext  
        If ImpExp1!F1 <> "" Then  
            GC.FindFirst "Name = '" & ImpExp1!F1 & "'"   
            If GC.NoMatch Then  
                MsgBox ("Cannot find groundclass " & ImpExp1!F1)  
            End If  
            CGC!ZoneGC = GC!ID  
        End If  
        ImpExp1.MoveNext  
        CGC.Update  
    Next CGCNo  
    If Not ImpExp1.EOF Then  
        MsgBox ("file cb_save is longer than expected")  
    End If  
    GC.Close  
    CGC.Close  
    ImpExp1.Close  
End Sub
```

```
Private Sub ImpGeom()  
    Dim Geometries As Recordset  
    Dim GeomNo As Integer, MaxGeomNo As Integer  
    Set Geometries = DB.OpenRecordset("tGeometries", dbOpenTable, dbAppendOnly)  
    DB.Execute ("ImpExp1Del")  
    DoCmd.TransferText acImportFixed, "Text ImpSpec", "ImpExp1", Path & "\geom_save", 0  
    Set ImpExp1 = DB.OpenRecordset("ImpExp1", dbOpenTable, dbReadOnly)  
    ImpExp1.Index = "PrimaryKey"  
    MaxGeomNo = CInt(ImpExp1!F1)  
    ImpExp1.MoveNext  
    For GeomNo = 0 To MaxGeomNo - 1  
        Geometries.addnew  
        Geometries!ID = CLng(GeomNo)  
        Geometries!Value1 = ImpExp1!F1  
        ImpExp1.MoveNext  
        Geometries!Value2 = ImpExp1!F1  
        ImpExp1.MoveNext  
        Geometries.Update  
    Next GeomNo
```

```
If Not ImpExpl.EOF Then
    MsgBox ("file geom_save is longer than expected")
End If
ImpExpl.Close
Geometries.Close
End Sub

Private Sub ImpAreaZone()
    Dim Areas As Recordset, Zones As Recordset, ZoneProb As Recordset
    Dim FirstZone() As Integer, LastZone() As Integer, MaxAreaNo As Integer
    Dim AreaNo As Integer, ZoneNo As Integer, MaxZoneNo As Integer, ZoneID As Long
    Set Areas = DB.OpenRecordset("tAreas", dbOpenTable, dbAppendOnly)
    Set Zones = DB.OpenRecordset("tZones", dbOpenTable, dbAppendOnly)
    Set ZoneProb = DB.OpenRecordset("tZoneProb", dbOpenTable, dbAppendOnly)
    DB.Execute ("ImpExplDel")
    DoCmd.TransferText acImportFixed, "Text ImpSpec", "ImpExpl", Path & "\area_save", 0
    Set ImpExpl = DB.OpenRecordset("ImpExpl", dbOpenTable, dbReadOnly)
    ImpExpl.Index = "PrimaryKey"
    MaxAreaNo = CInt(ImpExpl!F1)
    ImpExpl.MoveNext
    ReDim FirstZone(MaxAreaNo), LastZone(MaxAreaNo)
    For AreaNo = 0 To MaxAreaNo - 1
        Areas.addnew
        ' from now on pointer1 = ID of first area
        If AreaNo = 0 Then pointer1 = Areas!ID
        Areas!Name = ImpExpl!F1
        ImpExpl.Move (2)
        FirstZone(AreaNo) = CInt(ImpExpl!F1)
        ImpExpl.MoveNext
        LastZone(AreaNo) = CInt(ImpExpl!F1)
        ImpExpl.MoveNext
        Areas!Length = CDbl(ImpExpl!F1)
        ImpExpl.MoveNext
        If CLng(ImpExpl!F1) >= 0 Then
            Areas!ParamSet = CLng(ImpExpl!F1)
        End If
        ImpExpl.MoveNext
        Areas.Update
    Next AreaNo
    If Not ImpExpl.EOF Then
        MsgBox ("file area_save is longer than expected")
    End If
    Areas.Close
    ImpExpl.Close
    DB.Execute ("ImpExplDel")
    DoCmd.TransferText acImportFixed, "Text ImpSpec", "ImpExpl", Path & "\zone_save", 0
    Set ImpExpl = DB.OpenRecordset("ImpExpl", dbOpenTable, dbReadOnly)
    ImpExpl.Index = "PrimaryKey"
    MaxZoneNo = CInt(ImpExpl!F1)
    ImpExpl.MoveNext
    AreaNo = 0
    For ZoneNo = 0 To MaxZoneNo - 1
        If (ZoneNo < FirstZone(AreaNo) Or ZoneNo > LastZone(AreaNo)) Then
            AreaNo = 0
            Do Until (ZoneNo <= LastZone(AreaNo) And ZoneNo >= FirstZone(AreaNo)) Or _
                (AreaNo = MaxAreaNo)
                AreaNo = AreaNo + 1
            Loop
            If AreaNo = MaxAreaNo Then
                MsgBox ("no area contains zone " & CStr(ZoneNo))
            End If
        End If
        Zones.addnew
        Zones!Name = ImpExpl!F1
        ImpExpl.MoveNext
        Zones!ParamSet = CLng(ImpExpl!F1)
        ImpExpl.MoveNext
        Zones!Order = CDbl(ZoneNo)
    Next ZoneNo
End Sub
```

```
Zones!Area = pointer1 + CLng(AreaNo)
ZoneID = Zones!ID
Zones.Update
ZoneProb.addnew
ZoneProb!Zone = ZoneID
ZoneProb!GenerationMode = ImpExp1!F1
ImpExp1.MoveNext
ZoneProb!LengthMin = CDb1(ImpExp1!F1)
ImpExp1.MoveNext
ZoneProb!LengthMode = CDb1(ImpExp1!F1)
ImpExp1.MoveNext
ZoneProb!LengthMax = CDb1(ImpExp1!F1)
ImpExp1.MoveNext
ZoneProb!LengthProbMin = CDb1(ImpExp1!F1)
ImpExp1.MoveNext
ZoneProb!LengthProbMax = CDb1(ImpExp1!F1)
ImpExp1.MoveNext
ZoneProb!EndposMin = CDb1(ImpExp1!F1)
ImpExp1.MoveNext
ZoneProb!EndposMode = CDb1(ImpExp1!F1)
ImpExp1.MoveNext
ZoneProb!EndposMax = CDb1(ImpExp1!F1)
ImpExp1.MoveNext
ZoneProb!EndposProbMin = CDb1(ImpExp1!F1)
ImpExp1.MoveNext
ZoneProb!EndposProbMax = CDb1(ImpExp1!F1)
ImpExp1.MoveNext
ZoneProb.Update
Next ZoneNo
If Not ImpExp1.EOF Then
    MsgBox ("file zone_save is longer than expected")
End If
Zones.Close
ZoneProb.Close
ImpExp1.Close
End Sub

Private Sub ImpResources()
    Dim Resources As Recordset
    Dim ResNo As Integer, MaxResNo As Integer
    Set Resources = DB.OpenRecordset("tResources", dbOpenTable, dbAppendOnly)
    DB.Execute ("ImpExp2Del")
    DoCmd.TransferText acImportFixed, "Text ImpSpec", "ImpExp2", Path & "\res_name_save", 0
    Set ImpExp2 = DB.OpenRecordset("ImpExp2", dbOpenTable, dbReadOnly)
    ImpExp2.Index = "PrimaryKey"
    MaxResNo = CInt(ImpExp2!F1)
    pointer2 = 0
    ImpExp2.MoveNext
    For ResNo = 0 To MaxResNo - 1
        Resources.addnew
        ' pointer4 = ID of first resource
        If ResNo = 0 Then pointer4 = Resources!ID
        Resources!Name = ImpExp2!F1
        ImpExp2.MoveNext
        'Resources!Category = ImpExp2!F1 (category currently not supported)
        ImpExp2.MoveNext
        Resources.Update
    Next ResNo
    If Not ImpExp2.EOF Then
        MsgBox ("file res_name_save is longer than expected")
    End If
    Resources.Close
    ImpExp2.Close
End Sub

Private Sub ImpRepositories()
    Dim Repositories As Recordset, Stocks As Recordset
```



```
Dim ReposNo As Integer, MaxReposNo As Integer, ResNo As Integer, MaxResNo As Integer
Set Repositories = DB.OpenRecordset("tRepositories", dbOpenTable, dbAppendOnly)
Set Stocks = DB.OpenRecordset("tStocks", dbOpenTable, dbAppendOnly)
DB.Execute ("ImpExp2Del")
DoCmd.TransferText acImportFixed, "Text ImpSpec", "ImpExp2", Path & "\reposit_save", 0
Set ImpExp2 = DB.OpenRecordset("ImpExp2", dbOpenTable, dbReadOnly)
ImpExp2.Index = "PrimaryKey"
MaxReposNo = CInt(ImpExp2!F1)
ImpExp2.MoveNext
For ReposNo = 0 To MaxReposNo - 1
    Repositories.addnew
    ' pointer3 = ID of first repository
    If ReposNo = 0 Then pointer3 = Repositories!ID
    Repositories!Name = ImpExp2!F1
    ImpExp2.MoveNext
    Repositories.Update
    If pointer2 > 0 Then
        DB.Execute ("ImpExp1Del")
        DoCmd.TransferText acImportFixed, "Text ImpSpec", "ImpExp1", _
            Path & "\stock_reposit_save_" & CStr(ReposNo + 1), 0
        Set ImpExp1 = DB.OpenRecordset("ImpExp1", dbOpenTable, dbReadOnly)
        ImpExp1.Index = "PrimaryKey"
        MaxResNo = CInt(ImpExp1!F1)
        ImpExp1.MoveNext
        For ResNo = 0 To MaxResNo - 1
            Stocks.addnew
            Stocks!Repository = pointer3 + CLng(ReposNo)
            Stocks!Resource = pointer4 + CLng(ImpExp1!F1)
            ImpExp1.MoveNext
            Stocks!InitLevel = CDbl(ImpExp1!F1)
            ImpExp1.MoveNext
            Stocks!Capacity = CDbl(ImpExp1!F1)
            ImpExp1.MoveNext
            Stocks.Update
        Next ResNo
        If Not ImpExp1.EOF Then
            MsgBox ("file stock_reposit_save_" & CStr(ReposNo + 1) & _
                " is longer than expected")
        End If
        ImpExp1.Close
    End If
Next ReposNo
If Not ImpExp2.EOF Then
    MsgBox ("file reposit_save is longer than expected")
End If
Repositories.Close
ImpExp2.Close
Stocks.Close
End Sub

Private Sub ImpTunnelGraph()
    Dim Nodes As Recordset, Edges As Recordset, NodeNo As Integer, MaxNodeNo As Integer
    Dim EdgeNo As Integer, MaxEdgeNo As Integer, NodeNum() As Integer
    DB.Execute ("ImpExp1Del")
    DoCmd.TransferText acImportFixed, "Text ImpSpec", "ImpExp1", _
        Path & "\net_tun_save", 0
    Set ImpExp1 = DB.OpenRecordset("ImpExp1", dbOpenTable, dbReadOnly)
    ImpExp1.Index = "PrimaryKey"
    Set Nodes = DB.OpenRecordset("tTunnelNodes", dbOpenTable, dbAppendOnly)
    MaxNodeNo = CInt(ImpExp1!F1)
    ImpExp1.MoveNext
    ReDim NodeNum(MaxNodeNo) As Integer
    For NodeNo = 0 To MaxNodeNo - 1
        Nodes.addnew
        ' now pointer2 = ID of first node
        If NodeNo = 0 Then pointer2 = Nodes!ID
        Nodes!Name = Left(ImpExp1!F1, Len(ImpExp1!F1) - 1)
        ImpExp1.MoveNext
    Next NodeNo
End Sub
```

```
Nodes!Type = ImpExp1!F1
ImpExp1.MoveNext
NodeNum(NodeNo) = Cint(ImpExp1!F1)
ImpExp1.MoveNext
Nodes!PosX = CDBl(ImpExp1!F1)
ImpExp1.MoveNext
Nodes!PosY = CDBl(ImpExp1!F1)
ImpExp1.MoveNext
Nodes.Update
Next NodeNo
Nodes.Close

Set Edges = DB.OpenRecordset("tTunnelArcs", dbOpenTable, dbAppendOnly)
MaxEdgeNo = Cint(ImpExp1!F1)
ImpExp1.MoveNext
ImpTunnelData (MaxEdgeNo)

For EdgeNo = 0 To MaxEdgeNo - 1
    Edges.addnew
    Edges!ArcNo = Cint(ImpExp1!F1)
    ImpExp1.MoveNext
    Edges!StartNode = -1
    For NodeNo = 0 To MaxNodeNo - 1
        If NodeNum(NodeNo) = Cint(ImpExp1!F1) Then _
            Edges!StartNode = pointer2 + CLng(NodeNo)
    Next NodeNo
    If Edges!StartNode = -1 Then
        MsgBox ("cannot find node in net_tun_save")
        Edges!StartNode = pointer2
    End If
    ImpExp1.MoveNext
    Edges!EndNode = -1
    For NodeNo = 0 To MaxNodeNo - 1
        If NodeNum(NodeNo) = Cint(ImpExp1!F1) Then Edges!EndNode = pointer2 + CLng(NodeNo)
    Next NodeNo
    If Edges!EndNode = -1 Then
        MsgBox ("cannot find node in in net_tun_save")
        Edges!EndNode = pointer2
    End If
    ImpExp1.MoveNext
    Edges!Type = ImpExp1!F1
    ImpExp1.MoveNext
    Edges!Tunnel = pointer1 + CLng(EdgeNo)
    Edges.Update
Next EdgeNo
If Not ImpExp1.EOF Then
    MsgBox ("file net_tun_save is longer than expected")
End If
Edges.Close
ImpExp1.Close
End Sub

Private Sub ImpTunnelData(MaxTunnelNo As Integer)
    Dim Tunnels As Recordset, UpDown As Recordset, Delays As Recordset
    Dim DelayProb As Recordset, TunXRep As Recordset, FirstArea As Long, RepNo As Integer
    Dim TunnelNo As Integer, UpDownNo As Integer, DelayNo As Integer, DelayID As Integer
    Dim MaxUpDownNo As Integer, MaxDelayNo As Integer, MaxRepNo As Integer
    FirstArea = pointer1
    Set Tunnels = DB.OpenRecordset("tTunnels", dbOpenTable)
    Set UpDown = DB.OpenRecordset("tUpDownChanges", dbOpenTable, dbAppendOnly)
    Set Delays = DB.OpenRecordset("tTunDelays", dbOpenTable, dbAppendOnly)
    Set DelayProb = DB.OpenRecordset("tTunDelayProb", dbOpenTable, dbAppendOnly)
    Set TunXRep = DB.OpenRecordset("tTunnel_X_Rep", dbOpenTable, dbAppendOnly)
    ' because on of the fields is "opposite tunnel" we first have to create the records
    ' for all tunnels. otherwise we can't add opposite tunnels which are defined later
    ' in the file.
    For TunnelNo = 0 To MaxTunnelNo - 1
        Tunnels.addnew
```

```
' now pointer1 = ID of first tunnel
If TunnelNo = 0 Then pointer1 = Tunnels!ID
Tunnels!Name = "dummy"
Tunnels!Area = FirstArea
Tunnels!Geometry = 0
Tunnels.Update
Next TunnelNo
Tunnels.Index = "PrimaryKey"
Tunnels.MoveFirst
TunnelNo = 0
Do Until Tunnels.EOF
    DB.Execute ("ImpExp2Del")
    DoCmd.TransferText acImportFixed, "Text ImpSpec", "ImpExp2", _
        Path & "\data_tun_save_" & CStr(TunnelNo + 1), 0
    Set ImpExp2 = DB.OpenRecordset("ImpExp2", dbOpenTable, dbReadOnly)
    ImpExp2.Index = "PrimaryKey"
    Tunnels.Edit
    Tunnels!Name = ImpExp2!F1
    ImpExp2.MoveNext
    Tunnels!Type = CInt(ImpExp2!F1)
    ImpExp2.MoveNext
    Tunnels!OppTunnel = pointer1 + CLng(ImpExp2!F1)
    ImpExp2.MoveNext
    Tunnels!Area = FirstArea + CLng(ImpExp2!F1)
    ImpExp2.MoveNext
    Tunnels!Geometry = CLng(ImpExp2!F1)
    ImpExp2.MoveNext
    Tunnels!BeginPos = CDb1(ImpExp2!F1)
    ImpExp2.MoveNext
    Tunnels!EndPos = CDb1(ImpExp2!F1)
    ImpExp2.MoveNext
    Tunnels!FixCostMin = CDb1(ImpExp2!F1)
    ImpExp2.MoveNext
    Tunnels!FixCostMode = CDb1(ImpExp2!F1)
    ImpExp2.MoveNext
    Tunnels!FixCostMax = CDb1(ImpExp2!F1)
    ImpExp2.MoveNext
    Tunnels!TimeCostMin = CDb1(ImpExp2!F1)
    ImpExp2.MoveNext
    Tunnels!TimeCostMode = CDb1(ImpExp2!F1)
    ImpExp2.MoveNext
    Tunnels!TimeCostMax = CDb1(ImpExp2!F1)
    ImpExp2.MoveNext
    MaxRepNo = CInt(ImpExp2!F1)
    ImpExp2.MoveNext
    For RepNo = 0 To MaxRepNo - 1
        If ImpExp2!F1 = "-1" Then
            ImpExp2.Move (2)
        Else
            TunXRep.addnew
            TunXRep!Tunnel = Tunnels!ID
            TunXRep!Repository = pointer3 + CLng(ImpExp2!F1)
            ImpExp2.MoveNext
            TunXRep!UpToPosition = CDb1(ImpExp2!F1)
            ImpExp2.MoveNext
            TunXRep.Update
        End If
    Next RepNo
    MaxUpDownNo = CInt(ImpExp2!F1)
    ImpExp2.MoveNext
    For UpDownNo = 0 To MaxUpDownNo - 1
        UpDown.addnew
        UpDown!Tunnel = Tunnels!ID
        UpDown!Position = CDb1(ImpExp2!F1)
        ImpExp2.MoveNext
        UpDown.Update
    Next UpDownNo
    MaxDelayNo = CInt(ImpExp2!F1)
```

```
ImpExp2.MoveNext
' time-dependent delays
For DelayNo = 0 To MaxDelayNo - 1
    ' a delay with start --> infinity is added in order to have
    ' at least one delay.
    If CDBl(ImpExp2!F1) > 1000000000 Then
        ImpExp2.Move (4)
    Else
        Delays.addnew
        Delays!Tunnel = Tunnels!ID
        DelayID = Delays!ID
        Delays.Update
        DelayProb.addnew
        DelayProb!Delay = DelayID
        DelayProb!Type = 0
        DelayProb!Start = CDBl(ImpExp2!F1)
        ImpExp2.MoveNext
        DelayProb!DurationMin = CDBl(ImpExp2!F1)
        ImpExp2.MoveNext
        DelayProb!DurationMode = CDBl(ImpExp2!F1)
        ImpExp2.MoveNext
        DelayProb!DurationMax = CDBl(ImpExp2!F1)
        ImpExp2.MoveNext
        DelayProb.Update
    End If
Next DelayNo
' position-dependent delays
MaxDelayNo = CInt(ImpExp2!F1)
ImpExp2.MoveNext
For DelayNo = 0 To MaxDelayNo - 1
    If CDBl(ImpExp2!F1) > 1000000000 Then
        ImpExp2.Move (4)
    Else
        Delays.addnew
        Delays!Tunnel = Tunnels!ID
        DelayID = Delays!ID
        Delays.Update
        DelayProb.addnew
        DelayProb!Delay = DelayID
        DelayProb!Type = 1
        DelayProb!Start = CDBl(ImpExp2!F1)
        ImpExp2.MoveNext
        DelayProb!DurationMin = CDBl(ImpExp2!F1)
        ImpExp2.MoveNext
        DelayProb!DurationMode = CDBl(ImpExp2!F1)
        ImpExp2.MoveNext
        DelayProb!DurationMax = CDBl(ImpExp2!F1)
        ImpExp2.MoveNext
        DelayProb.Update
    End If
Next DelayNo
Tunnels!GraphicPos = CDBl(ImpExp2!F1)
ImpExp2.MoveNext
If Not ImpExp2.EOF Then
    MsgBox ("file data_tun_save_" & CStr(TunnelNo + 1) & " is longer than expected")
End If
ImpExp2.Close
Tunnels.Update
Tunnels.MoveNext
TunnelNo = TunnelNo + 1
Loop
End Sub

Private Sub ImpMethod()
    Dim Methods As Recordset, Heads As Recordset, HeadProb As Recordset
    Dim Variables As Recordset, VarProb As Recordset, ResNeed As Recordset
    Dim Activities As Recordset, ImpExpMemo As Recordset
    Dim MethodNo As Integer, MaxMethodNo As Integer, HeadNo As Integer
```

```
Dim MaxHeadNo As Integer, VarNo As Integer, MaxVarNo() As Integer
Dim ActNo As Integer, MaxActNo As Integer, ActIdentif() As Integer
Dim MethodID As Long, HeadID As Long, VariableID As Long
Dim Resources As Recordset, ResNo As Long, MaxResNo As Long, TypeNo As Integer
DB.Execute ("ImpExp2Del")
DoCmd.TransferText acImportFixed, "Text ImpSpec", "ImpExp2", Path & "\act_save", 0
Set ImpExp2 = DB.OpenRecordset("ImpExp2", dbOpenTable, dbReadOnly)
ImpExp2.Index = "PrimaryKey"
Set Activities = DB.OpenRecordset("tActivities", dbOpenTable)
MaxActNo = CInt(ImpExp2!F1)
ReDim ActIdentif(MaxActNo) As Integer
ImpExp2.MoveNext
For ActNo = 0 To MaxActNo - 1
    Activities.addnew
    ' now pointer1 = ID of first activity
    If ActNo = 0 Then pointer1 = Activities!ID
    Activities!Name = ImpExp2!F1
    ImpExp2.MoveNext
    ActIdentif(ActNo) = CInt(ImpExp2!F1)
    ImpExp2.MoveNext
    Activities!Priority = CInt(ImpExp2!F1)
    ImpExp2.MoveNext
    ' read equations later
    Activities!CostEquation = "0"
    Activities!TimeEquation = "0"
    Activities.Update
Next ActNo
If Not ImpExp2.EOF Then
    MsgBox ("file act_save is longer than expected")
End If
ImpExp2.Close
DB.Execute ("ImpExpMemoDel")
DoCmd.TransferText acImportDelim, "Memo ImpSpec", "ImpExpMemo", Path & "\gen_eq_save", 0
Set ImpExpMemo = DB.OpenRecordset("ImpExpMemo", dbOpenTable, dbReadOnly)
ImpExpMemo.Index = "PrimaryKey"
Activities.MoveFirst
For ActNo = 0 To MaxActNo - 1
    Activities.Edit
    ' read equations without ';' at the end
    Activities!TimeEquation = Left(ImpExpMemo!F1, Len(ImpExpMemo!F1) - 1)
    ImpExpMemo.MoveNext
    Activities!CostEquation = Left(ImpExpMemo!F1, Len(ImpExpMemo!F1) - 1)
    ImpExpMemo.MoveNext
    Activities.Update
Next ActNo
If Not ImpExpMemo.EOF Then
    MsgBox ("file gen_eq_save is longer than expected")
End If
ImpExpMemo.Close

Set Resources = DB.OpenRecordset("tResources", dbOpenTable, dbReadOnly)
If Not Resources.EOF And Resources.BOF Then
    MaxResNo = Resources.RecordCount
    Resources.Close
    Activities.Index = "Name"
    DB.Execute ("ImpExp1Del")
    DoCmd.TransferText acImportFixed, "Text ImpSpec", "ImpExp1", _
        Path & "\ress_need_save", 0
    Set ImpExp1 = DB.OpenRecordset("ImpExp1", dbOpenTable, dbReadOnly)
    ImpExp1.Index = "PrimaryKey"
    Set ResNeed = DB.OpenRecordset("tActResDemand", dbOpenTable)
    MaxActNo = CInt(ImpExp1!F1)
    ImpExp1.MoveNext
    For ActNo = 0 To MaxActNo - 1
        Activities.Seek "=", ImpExp1!F1
        ImpExp1.MoveNext
        If Activities.NoMatch Then
            ImpExp1.Move (4 * MaxResNo)
```

```
Else
  For TypeNo = 0 To 3
    For ResNo = 0 To MaxResNo - 1
      If CDBl(ImpExp1!F1) <> 0 Then ' only load demands <> 0
        ResNeed.addnew
        ResNeed!Activity = Activities!ID
        ResNeed!Resource = pointer4 + ResNo
        ResNeed!Type = TypeNo
        ResNeed!Demand = CDBl(ImpExp1!F1)
        ResNeed.Update
      End If
      ImpExp1.MoveNext
    Next ResNo
  Next TypeNo
End If
Next ActNo
If Not ImpExp1.EOF Then
  MsgBox ("file ress_need_save is longer than expected")
End If
ImpExp1.Close
ResNeed.Close
End If
Activities.Close

Set Methods = DB.OpenRecordset("tMethods", dbOpenTable, dbAppendOnly)
Methods.Index = "PrimaryKey"
Set Heads = DB.OpenRecordset("tHeads", dbOpenTable)
DB.Execute ("ImpExp1Del")
DoCmd.TransferText acImportFixed, "Text ImpSpec", "ImpExp1", Path & "\new_method_save", 0
Set ImpExp1 = DB.OpenRecordset("ImpExp1", dbOpenTable, dbReadOnly)
ImpExp1.Index = "PrimaryKey"
MaxMethodNo = CInt(ImpExp1!F1)
ImpExp1.MoveNext
ReDim MaxVarNo(MaxMethodNo) As Integer
For MethodNo = 0 To MaxMethodNo - 1
  Methods.addnew
  ' pointer3 = ID of first method
  If MethodNo = 0 Then pointer3 = Methods!ID
  Methods!Name = ImpExp1!F1
  ImpExp1.MoveNext
  Methods!CycleLength = CDBl(ImpExp1!F1)
  ImpExp1.MoveNext
  Methods!UnifiedSimul = Not CBool(ImpExp1!F1)
  ImpExp1.MoveNext
  MaxHeadNo = CInt(ImpExp1!F1)
  ImpExp1.MoveNext
  Methods!UnifiedRound = CBool(ImpExp1!F1)
  ImpExp1.MoveNext
  MethodID = Methods!ID
  Methods.Update
  MaxVarNo(MethodNo) = ImpExp1!F1
  ImpExp1.MoveNext
  For HeadNo = 0 To MaxHeadNo - 1
    Heads.addnew
    Heads!Method = MethodID
    Heads!Order = HeadNo
    HeadID = Heads!ID
    Heads.Update
    ImpHeadGraph MethodNo, HeadNo, HeadID, ActIdentif(), MaxActNo
  Next HeadNo
Next MethodNo
If Not ImpExp1.EOF Then
  MsgBox ("file new_method_save is longer than expected")
End If
ImpExp1.Close

DB.Execute ("ImpExp1Del")
DoCmd.TransferText acImportFixed, "Text ImpSpec", "ImpExp1", Path & "\hb_save", 0
```

```
Set ImpExpl = DB.OpenRecordset("ImpExpl", dbOpenTable, dbReadOnly)
ImpExpl.Index = "PrimaryKey"
Set HeadProb = DB.OpenRecordset("tHeadProb", dbOpenTable, dbAppendOnly)
Heads.MoveFirst
Do Until Heads.EOF
    Methods.Seek "=", Heads!Method
    HeadProb.addnew
    If Methods!CycleLength <= 0 Then
        HeadProb!CycleMin = CDb1(ImpExpl!F1)
        ImpExpl.MoveNext
        HeadProb!CycleMode = CDb1(ImpExpl!F1)
        ImpExpl.MoveNext
        HeadProb!CycleMax = CDb1(ImpExpl!F1)
        ImpExpl.MoveNext
        HeadProb!CycleProbMin = CDb1(ImpExpl!F1)
        ImpExpl.MoveNext
        HeadProb!CycleProbMax = CDb1(ImpExpl!F1)
        ImpExpl.MoveNext
    Else
        HeadProb!CycleProbMin = 0
        HeadProb!CycleMin = Methods!CycleLength
        HeadProb!CycleMode = Methods!CycleLength
        HeadProb!CycleMax = Methods!CycleLength
        HeadProb!CycleProbMax = 0
        ImpExpl.Move (5)
    End If
    HeadProb!Head = Heads!ID
    HeadProb.Update
    Heads.Edit
    Heads!DistMin = CDb1(ImpExpl!F1)
    ImpExpl.MoveNext
    Heads!DistMax = CDb1(ImpExpl!F1)
    ImpExpl.MoveNext
    Heads.Update
    Heads.MoveNext
Loop
If Not ImpExpl.EOF Then
    MsgBox ("file hb_save is longer than expected")
End If
ImpExpl.Close
Heads.Close
HeadProb.Close
Methods.Close

DB.Execute ("ImpExplDel")
DoCmd.TransferText acImportFixed, "Text ImpSpec", "ImpExpl", _
    Path & "\new_method_var_save", 0
Set ImpExpl = DB.OpenRecordset("ImpExpl", dbOpenTable, dbReadOnly)
ImpExpl.Index = "PrimaryKey"
Set Variables = DB.OpenRecordset("tVariables", dbOpenTable, dbAppendOnly)
Set VarProb = DB.OpenRecordset("tVariableProb", dbOpenTable, dbAppendOnly)
For MethodNo = 0 To MaxMethodNo - 1
    For VarNo = 0 To MaxVarNo(MethodNo) - 1
        Variables.addnew
        Variables!Method = pointer3 + CLng(MethodNo)
        Variables!Name = ImpExpl!F1
        ImpExpl.MoveNext
        VariableID = Variables!ID
        Variables.Update
        VarProb.addnew
        VarProb!Variable = VariableID
        VarProb!Min = CDb1(ImpExpl!F1)
        ImpExpl.MoveNext
        VarProb!Mode = CDb1(ImpExpl!F1)
        ImpExpl.MoveNext
        VarProb!Max = CDb1(ImpExpl!F1)
        ImpExpl.MoveNext
        VarProb!ProbMin = CDb1(ImpExpl!F1)
```

```
        ImpExp1.MoveNext
        VarProb.ProbMax = CDbl(ImpExp1!F1)
        ImpExp1.MoveNext
        VarProb.Update
    Next VarNo
Next MethodNo
If Not ImpExp1.EOF Then
    MsgBox ("file new_method_var_save is longer than expected")
End If
ImpExp1.Close
Variables.Close
VarProb.Close
End Sub

Private Sub ImpHeadGraph(Method As Integer, Head As Integer, HeadID As Long, _
    ActIdentif() As Integer, MaxActNo As Integer)
    Dim Nodes As Recordset, Edges As Recordset, NodeNo As Integer
    Dim MaxNodeNo As Integer, EdgeNo As Integer, MaxEdgeNo As Integer, NodeNum() As Integer
    Dim ActNo As Integer
    DB.Execute ("ImpExp2Del")
    DoCmd.TransferText acImportFixed, "Text ImpSpec", "ImpExp2", _
        Path & "\net_act_save_" & CStr(Method + 1) & "_" & CStr(Head + 1), 0
    Set ImpExp2 = DB.OpenRecordset("ImpExp2", dbOpenTable, dbReadOnly)
    ImpExp2.Index = "PrimaryKey"
    Set Nodes = DB.OpenRecordset("tMethNodes", dbOpenTable, dbAppendOnly)
    MaxNodeNo = CInt(ImpExp2!F1)
    ImpExp2.MoveNext
    ReDim NodeNum(MaxNodeNo) As Integer
    For NodeNo = 0 To MaxNodeNo - 1
        Nodes.addnew
        ' pointer2 = ID of first node of this method
        If NodeNo = 0 Then pointer2 = Nodes!ID
        Nodes!Head = HeadID
        Nodes!Name = Left(ImpExp2!F1, Len(ImpExp2!F1) - 1)
        ImpExp2.MoveNext
        Nodes!Type = ImpExp2!F1
        ImpExp2.MoveNext
        NodeNum(NodeNo) = CInt(ImpExp2!F1)
        ImpExp2.MoveNext
        Nodes!PosX = CDbl(ImpExp2!F1)
        ImpExp2.MoveNext
        Nodes!PosY = CDbl(ImpExp2!F1)
        ImpExp2.MoveNext
        Nodes.Update
    Next NodeNo
    Nodes.Close

    Set Edges = DB.OpenRecordset("tMethArcs", dbOpenTable, dbAppendOnly)
    MaxEdgeNo = CInt(ImpExp2!F1)
    ImpExp2.MoveNext
    For EdgeNo = 0 To MaxEdgeNo - 1
        Edges.addnew
        Edges!ArcNo = CInt(ImpExp2!F1)
        ImpExp2.MoveNext
        Edges!StartNode = -1
        For NodeNo = 0 To MaxNodeNo - 1
            If NodeNum(NodeNo) = CInt(ImpExp2!F1) Then _
                Edges!StartNode = pointer2 + CLng(NodeNo)
        Next NodeNo
        If Edges!StartNode = -1 Then
            MsgBox ("cannot find node in head " & CStr(HeadID))
            Edges!StartNode = pointer2
        End If
        ImpExp2.MoveNext
        Edges!EndNode = -1
        For NodeNo = 0 To MaxNodeNo - 1
            If NodeNum(NodeNo) = CInt(ImpExp2!F1) Then Edges!EndNode = pointer2 + CLng(NodeNo)
        Next NodeNo
    Next EdgeNo
    Edges.Close
End Sub
```



```
If Edges!EndNode = -1 Then
    MsgBox ("cannot find node in head " & CStr(HeadID))
    Edges!EndNode = pointer2
End If
ImpExp2.MoveNext
Edges!Type = ImpExp2!F1
ImpExp2.MoveNext
Edges!Activity = -1
For ActNo = 0 To MaxActNo - 1
    If ActIdentif(ActNo) = CInt(ImpExp1!F1) Then _
        Edges!Activity = pointer1 + CLng(ActNo)
Next ActNo
If Edges!Activity = -1 Then
    MsgBox ("cannot find activity in head " & CStr(HeadID))
    Edges!Activity = pointer1
End If
ImpExp1.MoveNext
Edges.Update
Next EdgeNo
If Not ImpExp2.EOF Then
    MsgBox ("file net_act_save_" & CStr(Method + 1) & "_" & CStr(Head + 1) & _
        "is longer than expected")
End If
Edges.Close
ImpExp2.Close
End Sub

Private Sub ImpChangeDelays()
    Dim ChangeDelays As Recordset, ImpExpDelay As Recordset
    DB.Execute ("ImpExpDelayDel")
    DoCmd.TransferText acImportFixed, "Delay ImpSpec", "ImpExpDelay", Path & "\delay_save", 0
    Set ImpExpDelay = DB.OpenRecordset("ImpExpDelay", dbOpenForwardOnly)
    Set ChangeDelays = DB.OpenRecordset("tChangeDelays", dbOpenTable, dbAppendOnly)
    Do While ImpExpDelay!F1 > -1
        ChangeDelays.addnew
        ChangeDelays!FromMethod = pointer3 + ImpExpDelay!F1
        ChangeDelays!ToMethod = pointer3 + ImpExpDelay!F2
        ChangeDelays!Delay = ImpExpDelay!F3
        ImpExpDelay.MoveNext
        ChangeDelays.Update
    Loop
    ChangeDelays.Close
    ImpExpDelay.Close
End Sub

Private Sub ImpLearnParam()
    Dim LearnParam As Recordset, LearnParamNo As Integer
    Dim MaxLearnParamNo As Integer, MethodNo As Integer, MaxMethodNo As Integer
    Set LearnParam = DB.OpenRecordset("tLearnParam", dbOpenTable, dbAppendOnly)
    DB.Execute ("ImpExp1Del")
    DoCmd.TransferText acImportFixed, "Text ImpSpec", "ImpExp1", Path & "\learn_save", 0
    Set ImpExp1 = DB.OpenRecordset("ImpExp1", dbOpenTable, dbReadOnly)
    ImpExp1.Index = "PrimaryKey"
    MaxMethodNo = CInt(ImpExp1!F1)
    ImpExp1.MoveNext
    For MethodNo = 0 To MaxMethodNo - 1
        MaxLearnParamNo = CInt(ImpExp1!F1)
        ImpExp1.MoveNext
        For LearnParamNo = 0 To MaxLearnParamNo - 1
            LearnParam.addnew
            LearnParam!Method = pointer3 + CLng(MethodNo)
            LearnParam!UpToCycle = CInt(ImpExp1!F1)
            ImpExp1.MoveNext
            LearnParam!Efficiency = CDbl(ImpExp1!F1)
            ImpExp1.MoveNext
            LearnParam.Update
        Next LearnParamNo
    Next MethodNo
End Sub
```

```
Next MethodNo
If Not ImpExp1.EOF Then
    MsgBox ("file learn_save is longer than expected")
End If
ImpExp1.Close
LearnParam.Close
End Sub

Private Sub ImpGCCGeomMeth()
    Dim Geometries As Recordset, GC As Recordset, GCXMethXGeom As Recordset
    Dim MaxGeomNo As Integer, MaxGCNo As Integer, MethNo As Integer, MaxMethNo As Integer
    Set Geometries = DB.OpenRecordset("tGeometries", dbOpenTable, dbReadOnly)
    Set GC = DB.OpenRecordset("tGC", dbOpenTable, dbAppendOnly)
    Set GCXMethXGeom = DB.OpenRecordset("tCGC_X_Meth_X_Geom", dbOpenTable, dbAppendOnly)
    DB.Execute ("ImpExp1Del")
    DoCmd.TransferText acImportFixed, "Text ImpSpec", "ImpExp1", _
        Path & "\gc_keyword_save", 0
    Set ImpExp1 = DB.OpenRecordset("ImpExp1", dbOpenTable, dbReadOnly)
    ImpExp1.Index = "PrimaryKey"
    DB.Execute ("ImpExp2Del")
    DoCmd.TransferText acImportFixed, "Text ImpSpec", "ImpExp2", _
        Path & "\data_method_save", 0
    Set ImpExp2 = DB.OpenRecordset("ImpExp2", dbOpenTable, dbReadOnly)
    ImpExp2.Index = "PrimaryKey"
    MaxGCNo = CInt(ImpExp2!F1)
    ImpExp2.MoveNext
    If MaxGCNo <> CInt(ImpExp1!F1) Then
        MsgBox "The number of ground classes does not equal the number of keywords."
    End If
    ImpExp1.MoveNext
    MaxGeomNo = CInt(ImpExp2!F1) * MaxGCNo
    ImpExp2.MoveNext
    GC.Index = "Name"
    Geometries.Index = "PrimaryKey"
    Do Until ImpExp1.EOF
        MaxGCNo = MaxGCNo - 1
        GC.Seek "=", ImpExp1!F1
        If GC.NoMatch Then
            GC.addnew
            GC!Name = ImpExp1!F1
            GC.Update
            GC.Seek "=", ImpExp1!F1
        End If
        ImpExp1.Move (2)
        Geometries.MoveFirst
        Do Until Geometries.EOF
            MaxGeomNo = MaxGeomNo - 1
            MaxMethNo = CInt(ImpExp2!F1)
            ImpExp2.MoveNext
            For MethNo = 0 To MaxMethNo - 1
                GCXMethXGeom.addnew
                GCXMethXGeom!CGC = GC!ID
                GCXMethXGeom!Geometry = Geometries!ID
                GCXMethXGeom!Method = pointer3 + CLng(ImpExp2!F1)
                ImpExp2.MoveNext
                GCXMethXGeom!Prob = CDBl(ImpExp2!F1)
                ImpExp2.MoveNext
                On Error GoTo Repair
                GCXMethXGeom.Update
            Next MethNo
            Geometries.MoveNext
        Loop
        GC.MoveNext
    Loop
    If MaxGCNo <> 0 Then
        MsgBox ("wrong number of groundclasses in data_method_save")
    Else
        If MaxGeomNo <> 0 Then
```

```
        MsgBox ("wrong number of geometries in data_method_save")
    End If
End If
If Not ImpExp2.EOF Then
    MsgBox ("file data_method_save is longer than expected")
End If
ImpExp2.Close
ImpExp1.Close
Geometries.Close
GC.Close
GCXMethXGeom.Close
Exit Sub
Repair:
If Err.Number = 3022 Then
    GCXMethXGeom.MoveLast
    Resume Next
Else
    MsgBox "Error importing file data_method_save"
    Resume Next
End If
End Sub

Private Sub ImpGenVar()
    Dim GenVariables As Recordset, GenVarProb As Recordset
    Dim MaxVarNo As Integer, VarNo As Integer, GenVarID As Long
    DB.Execute ("ImpExp1Del")
    DoCmd.TransferText acImportFixed, "Text ImpSpec", _
        "ImpExp1", Path & "\general_var_save", 0
    Set ImpExp1 = DB.OpenRecordset("ImpExp1", dbOpenTable, dbReadOnly)
    ImpExp1.Index = "PrimaryKey"
    Set GenVariables = DB.OpenRecordset("tGenVariables", dbOpenTable, dbAppendOnly)
    Set GenVarProb = DB.OpenRecordset("tGenVarProb", dbOpenTable, dbAppendOnly)
    MaxVarNo = CInt(ImpExp1!F1)
    ImpExp1.MoveNext
    For VarNo = 0 To MaxVarNo - 1
        GenVariables.addnew
        GenVariables!Name = ImpExp1!F1
        ImpExp1.MoveNext
        GenVariables!Description = ImpExp1!F1
        ImpExp1.MoveNext
        GenVariables!Unit = CInt(ImpExp1!F1)
        ImpExp1.MoveNext
        GenVarID = GenVariables!ID
        GenVariables.Update
        GenVarProb.addnew
        GenVarProb!Variable = GenVarID
        GenVarProb!Min = CDbl(ImpExp1!F1)
        ImpExp1.MoveNext
        GenVarProb!Mode = CDbl(ImpExp1!F1)
        ImpExp1.MoveNext
        GenVarProb!Max = CDbl(ImpExp1!F1)
        ImpExp1.MoveNext
        GenVarProb!ProbMin = CDbl(ImpExp1!F1)
        ImpExp1.MoveNext
        GenVarProb!ProbMax = CDbl(ImpExp1!F1)
        ImpExp1.MoveNext
        GenVarProb.Update
    Next VarNo
    If Not ImpExp1.EOF Then
        MsgBox "file general_var_save is longer than expected"
    End If
    ImpExp1.Close
    GenVariables.Close
    GenVarProb.Close
End Sub
```

Module: ManageUpdatePhase

Option Compare Database
Option Explicit

```
Public Sub StartNewPhase()  
    Dim DB As Database, Phases As Recordset, Name As String  
    Dim PhaseFrom As Long, PhaseTo As Long, Title As String  
    Title = "START NEW UPDATE PHASE"  
    Name = ""  
    Name = InputBox("Which title would you like to give" & Chr(13) & _  
        "to this new update phase?", Title)  
    If Name = "" Then Exit Sub  
    DoCmd.OpenForm "BePatient", acNormal  
    DoCmd.RepaintObject acForm, "BePatient"  
    DoCmd.Hourglass True  
    DoCmd.Echo False, "Working hard..."  
    DoCmd.SetWarnings False  
    Set DB = CurrentDb()  
    Set Phases = DB.OpenRecordset("tUpdatePhases", dbOpenTable)  
    Phases.Index = "Current"  
    Phases.Seek "=", True  
    If Phases.NoMatch Then ' no active phase yet  
        PhaseFrom = 1  
    Else  
        PhaseFrom = Phases!ID  
        Phases.Edit  
        Phases!Current = False  
        Phases.Update  
    End If  
    Phases.addnew  
    Phases!Name = Name  
    Phases!Current = True  
    PhaseTo = Phases!ID  
    Phases.Update  
    Phases.Close  
    ' copy records of last active phase to new update phase  
    AppendPhase DB, PhaseFrom, PhaseTo, "CGC_X_Meth_X_Geom"  
    AppendPhase DB, PhaseFrom, PhaseTo, "ChangeDelays"  
    AppendPhase DB, PhaseFrom, PhaseTo, "DeterSegs"  
    AppendPhase DB, PhaseFrom, PhaseTo, "GenVarProb"  
    AppendPhase DB, PhaseFrom, PhaseTo, "HeadProb"  
    AppendPhase DB, PhaseFrom, PhaseTo, "MarkovLengths"  
    AppendPhase DB, PhaseFrom, PhaseTo, "MarkovTransProb"  
    AppendPhase DB, PhaseFrom, PhaseTo, "SegProb"  
    AppendPhase DB, PhaseFrom, PhaseTo, "Stocks"  
    AppendPhase DB, PhaseFrom, PhaseTo, "TunDelayProb"  
    AppendPhase DB, PhaseFrom, PhaseTo, "VariableProb"  
    AppendPhase DB, PhaseFrom, PhaseTo, "ZoneProb"  
    If PhaseFrom = 1 Then ' among other things there are no graphs yet (see thesis too)  
        ' this creates the records of phase 10  
        AppendPhase DB, PhaseFrom, 10, "MarkovLengths"  
        ' this updates all input, among other things it calculates Markov length distrib.  
        Observations.Digest  
        ' this copys the calculated values to "posterior"  
        MarkovCopyCalculated DB, PhaseTo  
        ' create prior graphs  
        Graphs.GraphsPriorPostCreate DB, "tMarkovLengths", "otMarkovGraph", 1  
        Graphs.GraphsPriorPostCreate DB, "tVariableProb", "otVariablesGraph", 1  
        Graphs.GraphsPriorPostCreate DB, "tGenVarProb", "otGenVariablesGraph", 1  
        Graphs.GraphsPriorPostCreate DB, "qHeadProb", "otCycleGraph", 1  
        Graphs.GraphsTunDelaySet DB, True  
        ' create posterior graphs  
        Graphs.GraphsPriorPostCreate DB, "tMarkovLengths", "otMarkovGraph", PhaseTo  
        Graphs.GraphsPriorPostCreate DB, "tVariableProb", "otVariablesGraph", PhaseTo  
        Graphs.GraphsPriorPostCreate DB, "tGenVarProb", "otGenVariablesGraph", PhaseTo  
        Graphs.GraphsPriorPostCreate DB, "qHeadProb", "otCycleGraph", PhaseTo
```

```
End If
' note that except for the special case above, observed data are not processed.
' it is not necessary to calculate posterior graphs because the underlying data has
' not changed (since it was copied from the last active update phase)
DB.Close
DoCmd.SetWarnings True
DoCmd.Echo True
DoCmd.Hourglass False
DoCmd.Close acForm, "BePatient", acSaveNo
End Sub

Public Sub AppendPhase(DB As Database, PhaseFrom As Long, PhaseTo As Long, Name As String)
    Static Count As Recordset, Append As QueryDef
    Set Count = DB.OpenRecordset("t" & Name, dbOpenTable, dbForwardOnly)
    ' if # of records = 0, the query would generate an error
    If Not (Count.BOF And Count.EOF) Then
        Set Append = DB.QueryDefs("aq" & Name)
        Append.Parameters!PhaseFrom = PhaseFrom
        Append.Parameters!PhaseTo = PhaseTo
        Append.Execute
    End If
    Count.Close
End Sub

Private Sub MarkovCopyCalculated(DB As Database, PhaseTo As Long)
    Dim Lengths As Recordset, Graph As Recordset, ParamSet As Long, ParamState As Long
    Dim Min As Double, Mode As Double, Max As Double
    Set Lengths = DB.OpenRecordset("tMarkovLengths", dbOpenTable)
    If Lengths.BOF And Lengths.EOF Then Exit Sub
    Set Graph = DB.OpenRecordset("otMarkovGraph", dbOpenTable)
    Lengths.Index = "PrimaryKey"
    Lengths.MoveFirst
    Do Until Lengths.EOF
        If Lengths!UpdatePhase = 10 Then
            ' just remember values
            ParamSet = Lengths!ParamSet
            ParamState = Lengths!ParamState
            Min = Lengths!LengthMin
            Mode = Lengths!LengthMode
            Max = Lengths!LengthMax
        ElseIf Lengths!UpdatePhase = PhaseTo Then
            ' every new update phase has an ID > 10 and every record which corresponds to
            ' a new phase appears later in the recordset than the record corr. to phase 10
            If ParamSet = Lengths!ParamSet And ParamState = Lengths!ParamState Then
                Lengths.Edit
                Lengths!LengthMin = Min
                Lengths!LengthMode = Mode
                Lengths!LengthMax = Max
                Lengths.Update
            End If
        End If
        Lengths.MoveNext
    Loop
End Sub

Public Sub RemoveUpdatePhase(UpdatePhase As Long, Current As Boolean, DB As Database)
    ' delete all probabilistic data of this update phase
    ExecuteDeleteQuery "ChangeDelays", UpdatePhase
    ExecuteDeleteQuery "CGC_X_Meth_X_Geom", UpdatePhase
    ExecuteDeleteQuery "GenVarProb", UpdatePhase
    ExecuteDeleteQuery "DeterSegs", UpdatePhase
    ExecuteDeleteQuery "HeadProb", UpdatePhase
    ExecuteDeleteQuery "MarkovLengths", UpdatePhase
    ExecuteDeleteQuery "MarkovTransProb", UpdatePhase
    ExecuteDeleteQuery "SegProb", UpdatePhase
    ExecuteDeleteQuery "Stocks", UpdatePhase
    ExecuteDeleteQuery "TunDelayProb", UpdatePhase
End Sub
```

```
ExecuteDeleteQuery "VariableProb", UpdatePhase
ExecuteDeleteQuery "ZoneProb", UpdatePhase
' delete results of this update phase
ExecuteDeleteQuery "rTimeCost", UpdatePhase
ExecuteDeleteQuery "rTimeCostGraph", UpdatePhase
ExecuteDeleteQuery "rTimePosGraph", UpdatePhase
If Current Then
  ' delete all graphs
  DB.Execute ("dgoMarkovGraph")
  DB.Execute ("dgoVariablesGraph")
  DB.Execute ("dgoGenVariablesGraph")
  DB.Execute ("dgoCycleGraph")
  DB.Execute ("dgoTunDelaysGraph")
  DB.Execute ("dgoChangeDelaysGraph")
  ' delete calculated or observed values
  ExecuteDeleteQuery "MarkovLengths", 10
  ExecuteDeleteQuery "MarkovTransProb", 10
  ExecuteDeleteQuery "CGC_X_Meth_X_Geom", 10
  'ExecuteDeleteQuery "ZoneProb", 10
  ' delete all tunnel delays with duration = 0
  RemoveZeroDelays
  ' delete all semi-deterministic segments with length = 0
  RemoveZeroSegments
  ' delete all associations with probability = 0
  RemoveZeroAssociations
End If
End Sub

Public Sub RemoveAllUpdatePhases(DB As Database)
  ' delete all graphs
  DB.Execute ("dgoMarkovGraph")
  DB.Execute ("dgoVariablesGraph")
  DB.Execute ("dgoGenVariablesGraph")
  DB.Execute ("dgoCycleGraph")
  DB.Execute ("dgoTunDelaysGraph")
  DB.Execute ("dgoChangeDelaysGraph")
  ' delete all probabilistic data except prior and calculated / observed data
  ExecuteDeleteQuery "ChangeDelays", 0
  ExecuteDeleteQuery "CGC_X_Meth_X_Geom", 0
  ExecuteDeleteQuery "DeterSegs", 0
  ExecuteDeleteQuery "GenVarProb", 0
  ExecuteDeleteQuery "HeadProb", 0
  ExecuteDeleteQuery "MarkovLengths", 0
  ExecuteDeleteQuery "MarkovTransProb", 0
  ExecuteDeleteQuery "SegProb", 0
  ExecuteDeleteQuery "Stocks", 0
  ExecuteDeleteQuery "TunDelayProb", 0
  ExecuteDeleteQuery "VariableProb", 0
  ExecuteDeleteQuery "ZoneProb", 0
  ' delete calculated or observed data
  ExecuteDeleteQuery "MarkovLengths", 10
  ExecuteDeleteQuery "MarkovTransProb", 10
  ExecuteDeleteQuery "CGC_X_Meth_X_Geom", 10
  ' delete the results of all update phases except prior input
  ExecuteDeleteQuery "rTimeCost", 0
  ExecuteDeleteQuery "rTimeCostGraph", 0
  ExecuteDeleteQuery "rTimePosGraph", 0
  'delete all Tunnel Delays with duration = 0
  RemoveZeroDelays
  ' delete all semi-deterministic segments with length = 0
  RemoveZeroSegments
  ' delete all associations with probability = 0
  RemoveZeroAssociations
  DB.QueryDefs("dqUpdatePhases").Execute
End Sub
```

```
Public Sub RemoveObservations(DB As Database)
    ' delete all graphs
    DB.Execute ("dgoMarkovGraph")
    DB.Execute ("dgoVariablesGraph")
    DB.Execute ("dgoGenVariablesGraph")
    DB.Execute ("dgoCycleGraph")
    DB.Execute ("dgoTunDelaysGraph")
    DB.Execute ("dgoChangeDelaysGraph")
    ' delete calculated or observed data
    ExecuteDeleteQuery "MarkovLengths", 10
    ExecuteDeleteQuery "MarkovTransProb", 10
    ExecuteDeleteQuery "CGC_X_Meth_X_Geom", 10
    ExecuteDeleteQuery "ZoneProb", 10
    DB.Execute ("dgoAreaParamStat")
    DB.Execute ("dgoChangeDelays")
    DB.Execute ("dgoCycleLengths")
    DB.Execute ("dgoGCGeom")
    DB.Execute ("dgoGenVariables")
    DB.Execute ("dgoMethods")
    DB.Execute ("dgoTransitions")
    DB.Execute ("dgoTunDelays")
    DB.Execute ("dgoVariables")
    DB.Execute ("dgoZoneParamStat")
End Sub

Public Sub ExecutedDeleteQuery(Name As String, UpdatePhase As Long)
    Static QDef As QueryDef, SQLString As String, OldSQL As String
    Set QDef = CurrentDb().QueryDefs("dq" & Name)
    SQLString = Left$(QDef.SQL, Len(QDef.SQL) - 3)
    OldSQL = SQLString
    If UpdatePhase = 0 Then ' delete all phases (except 1, 10 and 11)
        SQLString = SQLString & _
            " WHERE Not(UpdatePhase = 1 Or UpdatePhase = 10 Or UpdatePhase = 11)"
    Else ' delete specified phase
        SQLString = SQLString & " WHERE UpdatePhase = " & CStr(UpdatePhase)
    End If
    QDef.SQL = SQLString
    QDef.Execute
    QDef.SQL = OldSQL
End Sub

Public Sub RemoveZeroDelays()
    Static Delays As Recordset, Delay As Long, DelayProb As Recordset
    ' delete all Tunnel Delays with duration = 0
    Set Delays = CurrentDb().OpenRecordset("tTunDelays", dbOpenTable)
    If Delays.EOF And Delays.BOF Then Exit Sub
    Set DelayProb = CurrentDb().OpenRecordset("tTunDelayProb", dbOpenTable)
    Delays.Index = "PrimaryKey"
    With DelayProb
        .MoveFirst
        Do Until .EOF
            If !DurationMin = 0 And !DurationMode = 0 And !DurationMax = 0 Then
                Delay = !Delay
                .delete
                Delays.Seek "=", Delay
                Delays.delete
            End If
            .MoveNext
        Loop
    End With
    Delays.Close
    DelayProb.Close
End Sub

Public Sub RemoveZeroZones()
    Static Zones As Recordset, Zone As Long, ZoneProb As Recordset
    ' delete all zones with length = 0
```

```
Set Zones = CurrentDb().OpenRecordset("tZones", dbOpenTable)
If Zones.EOF And Zones.BOF Then Exit Sub
Set ZoneProb = CurrentDb().OpenRecordset("tZoneProb", dbOpenTable)
Zones.Index = "PrimaryKey"
With ZoneProb
    .MoveFirst
    Do Until .EOF
        If !LengthMin = 0 And !LengthMode <= 0 And _
            !LengthMax = 0 And !GenerationMode = "0" Then
            Zone = !Zone
            .delete
            Zones.Seek "=", Zone
            Zones.delete
        End If
        .MoveNext
    Loop
End With
Zones.Close
ZoneProb.Close
End Sub

Public Sub RemoveZeroSegments()
    Static Segs As Recordset, Seg As Long, SegProb As Recordset
    ' delete all segments with length = 0
    Set Segs = CurrentDb().OpenRecordset("tSemiDeterSegs", dbOpenTable)
    If Segs.EOF And Segs.BOF Then Exit Sub
    Set SegProb = CurrentDb().OpenRecordset("tSegProb", dbOpenTable)
    Segs.Index = "PrimaryKey"
    With SegProb
        .MoveFirst
        Do Until .EOF
            If !LengthMin = 0 And !LengthMode <= 0 And _
                !LengthMax = 0 And !GenerationMode = "0" Then
                Seg = !Segment
                .delete
                Segs.Seek "=", Seg
                Segs.delete
            End If
            .MoveNext
        Loop
    End With
    Segs.Close
    SegProb.Close
End Sub

Public Sub RemoveZeroAssociations()
    Static Assoc As Recordset, AssocSum As Recordset
    Set Assoc = CurrentDb().OpenRecordset("tCGC_X_Meth_X_Geom", dbOpenTable)
    Assoc.Index = "PrimaryKey"
    Set AssocSum = CurrentDb().OpenRecordset("qAssocProbSum", dbOpenDynaset, dbReadOnly)
    If AssocSum.EOF And AssocSum.BOF Then Exit Sub
    AssocSum.FindFirst "SumOfProb = 0"
    Do Until AssocSum.NoMatch
        Assoc.Seek ">=", AssocSum!Geometry, AssocSum!CGC, AssocSum!Method
        Do Until Assoc.NoMatch
            If Assoc!Geometry = AssocSum!Geometry And Assoc!CGC = AssocSum!CGC _
                And Assoc!Method = AssocSum!Method Then
                Assoc.delete
                Assoc.Seek ">=", AssocSum!Geometry, AssocSum!CGC, AssocSum!Method
            Else
                Exit Do
            End If
        Loop
        AssocSum.Requery
        AssocSum.FindFirst "SumOfProb = 0"
    Loop
    Assoc.Close
```



```
AssocSum.Close
End Sub
```

Module: Observations

```
Option Compare Database
Option Explicit
Dim DB As Database, GCGeomMeth As Recordset, GCGeom As Recordset
Dim GCComb As Recordset, GCCombStat As Recordset, CGC As Recordset
Dim Trans As Recordset, TransProb As Recordset
Dim GCID() As Long, GCEndpos() As Double
Dim MaxNoParams As Integer, MaxGCSEg As Integer, GCSEg As Integer

Public Sub Digest()
    Dim UpdatePhase As Long, UpdatePhases As Recordset, LastUpdatePhase As Long
    Set DB = CurrentDb()
    If IsNull(DLookup("ID", "tUpdatePhases", "[Current] = True")) Then
        Exit Sub ' there is no active update phase
    Else
        UpdatePhase = DLookup("ID", "tUpdatePhases", "[Current] = True")
    End If
    Set GCGeomMeth = DB.OpenRecordset("tCGC_X_Meth_X_Geom", dbOpenTable)
    GCGeomMeth.Index = "PrimaryKey"
    Set GCGeom = DB.OpenRecordset("otGCGeom", dbOpenTable)
    GCGeom.Index = "PrimaryKey"
    Set GCComb = DB.OpenRecordset("tGCCCombinations", dbOpenTable)
    GCComb.Index = "GC"
    Set GCCombStat = DB.OpenRecordset("tGCComb_X_States", dbOpenTable)
    GCCombStat.Index = "PrimaryKey"
    Set CGC = DB.OpenRecordset("tCombGC", dbOpenTable, dbReadOnly)
    CGC.Index = "GC"
    AssocUpdate ' find observed geometry/GC/method association probabilities
    GCGeomMeth.Close
    GCGeom.Close
    GCComb.Close
    GCCombStat.Close
    CGC.Close

    ' start matching process with data from previous updatephase and current posterior
    ' data. These data may be messed up by the user or because the last
    ' time observations were processed there was an error.
    Set UpdatePhases = DB.OpenRecordset("tUpdatePhases", dbOpenTable, dbReadOnly)
    UpdatePhases.Index = "PrimaryKey"
    UpdatePhases.MoveFirst
    Do Until UpdatePhases!Current
        UpdatePhases.MoveNext
    Loop
    UpdatePhases.MovePrevious
    ' update phases are ordered chronologically because ID is assigned automatically
    LastUpdatePhase = UpdatePhases!ID
    If LastUpdatePhase <= 11 Then LastUpdatePhase = 1
    UpdatePhases.Close
    ' delete old posterior segments/zones/delays
    ManageUpdatePhases.ExecuteDeleteQuery "DeterSegs", UpdatePhase
    ManageUpdatePhases.ExecuteDeleteQuery "SegProb", UpdatePhase
    ManageUpdatePhases.ExecuteDeleteQuery "TunDelayProb", UpdatePhase
    ManageUpdatePhases.ExecuteDeleteQuery "ZoneProb", UpdatePhase
    If LastUpdatePhase = 1 Then
        ' delete delays with duration = 0 (must delete graphs first)
        DB.Execute ("dgoTunDelaysGraph")
        ManageUpdatePhases.RemoveZeroDelays
        ' delete all semi-deterministic segments with length = 0
        ManageUpdatePhases.RemoveZeroSegments
        ' delete all associations with probability = 0
        ManageUpdatePhases.RemoveZeroAssociations
    End If
    ' create new posterior segments/zones/delays (copy form last update phase)
```

```
ManageUpdatePhases.AppendPhase DB, LastUpdatePhase, UpdatePhase, "DeterSegs"
ManageUpdatePhases.AppendPhase DB, LastUpdatePhase, UpdatePhase, "SegProb"
ManageUpdatePhases.AppendPhase DB, LastUpdatePhase, UpdatePhase, "TunDelayProb"
ManageUpdatePhases.AppendPhase DB, LastUpdatePhase, UpdatePhase, "ZoneProb"

ZoneUpdate DB          ' match observed zone sequence with prior input
SegSemiDeterUpdate DB  ' match observed parameter state sequence with prior input
SegDeterUpdate DB      ' match observed parameter state sequence with prior input
TunDelayUpdate DB      ' match observed tunnel delays with prior input
TunnelInfoUpdate DB    ' find current positions of tunnels and current time

Set Trans = DB.OpenRecordset("otTransitions", dbOpenTable)
Trans.Index = "PrimaryKey"
Set TransProb = DB.OpenRecordset("tMarkovTransProb", dbOpenTable)
TransProb.Index = "PrimaryKey"
TransProbUpdate DB ' find observed transition probabilities
Trans.Close
TransProb.Close

MarkovUpdate ' calculate likelihood graph and analytically correct posterior
VariablesUpdate DB, "Variables"          ' (re)calculate observed frequency graph
VariablesUpdate DB, "GenVariables"
VariablesUpdate DB, "Heads"
ChangeDelaysUpdate DB
Graphs.GraphsTunDelaySet DB, False ' (re)calculate observed/posterior graph
' (re)calculate prior delay graphs (in case of added delays)
Graphs.GraphsTunDelaySet DB, True
DB.Close
End Sub

Private Sub AssocUpdate()
Dim ZoneDef As QueryDef, AreaDef As QueryDef, MethodDef As QueryDef, Areas As Recordset
Dim Zone As Recordset, Area As Recordset, Method As Recordset, ParamSets As Recordset
Dim MaxParamNo As Integer, CurrentAreaParam As Integer, CurrentZoneParam As Integer
Dim ZoneParamNo As Integer, AreaParamNo As Integer, Param As Long, Tunnel As Long
Dim ZoneParamID() As Long, AreaParamID() As Long, ZoneParamComb() As Long
Dim AreaParamComb() As Long, AreaEndpos As Double
Dim ZoneBeginpos() As Double, AreaBeginpos() As Double, MaxBeginpos As Double
Dim ZoneParamSet As Long, AreaParamSet As Long, ZoneEndpos As Double
Dim ZoneGC As Long, AreaGC As Long, Geom As Long, MethodBeginpos As Double
Dim MethodEndpos As Double, SegBegin As Double, MethodNo As Long, GCNo As Long

'NOTE: Most comments are only given for the context of areas. The treatement of
'       zones is almost identical and comments are omitted.
'       Also note that all recordsets are sorted by decreasing endpos!

AssocInitialize
' Sets "total observed length" of each GC/geometry combination and
' probabilities of all associations to 0
' Whenever a complete set of GC/geometry/method is found, procedure AssocAdd will
' add the length over which this set was observed to the probability of this assoc.
' and to the length of this GC/Geom combination. Note that during the procedure
' AssocUpdate the 'probability' of a GC/geometry/method set is in fact the total
' length over which it was observed. Only at the end, procedure AssocCalculate will
' divide the 'probability' (total length) of the GC/geom/meth set by the total length
' of the GC/geometry set, thereby calculating the probability of the GC/geom/meth set.

Set ParamSets = DB.OpenRecordset("qParam_X_Set", dbOpenDynaset, dbReadOnly)
ParamSets.MoveFirst
' Determine max number of parameters per parameter set, in order to ReDim arrays
MaxParamNo = 0
Do Until ParamSets.EOF
    If ParamSets!NoParams > MaxParamNo Then MaxParamNo = ParamSets!NoParams
    ParamSets.MoveNext
Loop
ReDim ZoneParamID(MaxParamNo), AreaParamID(MaxParamNo)
ReDim ZoneParamComb(MaxParamNo), AreaParamComb(MaxParamNo)
ReDim ZoneBeginpos(MaxParamNo), AreaBeginpos(MaxParamNo)
```

```
Set Areas = DB.OpenRecordset("tAreas", dbOpenTable, dbReadOnly)
Set ZoneDef = DB.QueryDefs("oqZoneParamComb")
Set AreaDef = DB.QueryDefs("oqAreaParamComb")
Set MethodDef = DB.QueryDefs("oqMethod")
Areas.MoveFirst
Do Until Areas.EOF
    ZoneDef.Parameters("AreaID") = Areas!ID
    AreaDef.Parameters("AreaID") = Areas!ID
    MethodDef.Parameters("AreaID") = Areas!ID
    Set Zone = ZoneDef.OpenRecordset(dbOpenDynaset, dbReadOnly)
    Set Area = AreaDef.OpenRecordset(dbOpenDynaset, dbReadOnly)
    Set Method = MethodDef.OpenRecordset(dbOpenDynaset, dbReadOnly)
    If Area.EOF Then
        AreaEndpos = -1
        ' --> AreaEndpos will always be < ZoneEndpos, i.e. later in the code area GC
        ' will never change because there is no change in area parameter states
    Else
        Area.MoveFirst
        AreaEndpos = Area!EndPos
    End If
    If IsNull(Areas!ParamSet) Then
        AreaParamNo = 0
        AreaGC = -1
        ' -1 means that it is not relevant (i.e. there is no area parameter set)
        ' as opposed to 0, which means that there is an area parameter set, but the
        ' area GC is unknown because one or multiple parameter states are unknown
    Else
        AreaParamSet = Areas!ParamSet
        ParamSets.FindFirst "ParamSet = " & CStr(AreaParamSet)
        AreaParamNo = ParamSets!NoParams
        For Param = 0 To AreaParamNo - 1
            AreaParamID(Param) = ParamSets!ID
            'AreaParamID(i-1) holds the record ID of the i-th parameter
            AreaParamComb(Param) = 0
            'AreaParamComb(i-1) holds the record ID of the state of the i-th parameter
            ParamSets.MoveNext
        Next Param
        AreaGC = 0
    End If
    MaxGCSEg = Zone.RecordCount + Area.RecordCount
    GCSEg = 0
    ReDim GCID(MaxGCSEg), GCEndpos(MaxGCSEg)
    ZoneParamSet = -1
    'if there are no zone param states or no methods, no associations will result
    If (Not Zone.EOF) And (Not Method.EOF) Then
        Zone.MoveFirst
        ZoneEndpos = Zone!EndPos
        ZoneGC = 0
        Do Until Zone.EOF And Area.EOF
            ' Zone!Endpos holds the position of next change in zone parameter state
            ' according to the recordset.
            ' ZoneEndpos hold the true pos of the next change. It takes into account,
            ' that the beginpos of an observed parameter state may be greater than the
            ' endpos of the previous observed state of the same parameter, thus creating
            ' a gap where the parameter state is unknown. The endpos of this gap
            ' (= beginpos of the said observed param state) is then the true pos of the
            ' next change. If Zone!Endpos < ZoneEndpos there is a gap. (its minimal
            ' length being ZoneEndpos - Zone!Endpos). In case of a gap the next record
            ' of the recordset must not yet be considered.

            ' If the parameter set changes (i.e. entering a different zone), all
            ' states of the new parameters will be unknown
            If AreaEndpos > ZoneEndpos Then
                ' the next change in param state (and therefore GC) occurs in
                ' area param set
                If Area.EOF Then
                    ' Arriving at this point means that there can be no more valid GC
```

```
' beyond this point, because at least CurrentAreaParam will always
' be unknown.
AreaParamComb(CurrentAreaParam) = 0
AreaBeginpos(CurrentAreaParam) = 0
AreaGC = 0
GCID(GCSeg) = 0
ElseIf AreaEndpos <= Area!EndPos Then      'check for gap
  For Param = 0 To AreaParamNo - 1      'no gap
    'check to see which parameter changes its state
    If AreaParamID(Param) = Area!Parameter Then
      AreaParamComb(Param) = Area!ParamState
      ' AreaBeginpos(i-1) defines up to which position the
      ' i-th parameter is currently known
      AreaBeginpos(Param) = Area!BeginPos
    End If
  Next Param
  AreaGC = AssocFindGC(AreaParamSet, AreaParamNo, AreaParamComb())
  GCID(GCSeg) = AssocFindCGC(AreaGC, ZoneGC)
  Area.MoveNext
Else
  AreaParamComb(CurrentAreaParam) = 0
  ' CurrentAreaParam is the index of the parameter which has just
  ' entered a gap
  AreaGC = 0
  GCID(GCSeg) = 0
End If
' the endpos of the new GC is the endpos of the latest change in
' parameter state
GCEndpos(GCSeg) = AreaEndpos
If GCSeg > 0 Then
  If (GCEndpos(GCSeg - 1) = GCEndpos(GCSeg)) Then
    ' eliminate GC segments with length = 0
    ' note that the previous GC has length 0, not the new one
    GCID(GCSeg - 1) = GCID(GCSeg)
    GCSeg = GCSeg - 1
  ElseIf GCID(GCSeg - 1) = GCID(GCSeg) Then
    ' eliminate fictitious changes in GC
    GCSeg = GCSeg - 1
  End If
End If
GCSeg = GCSeg + 1
' MaxBeginpos is a temporary variable used to determine whether there
' are parameters with beginpos between AreaEndpos and Area!Endpos.
' If this is the case the one with the largest beginpos will become the
' current one and its state will later be changed to 0 because after
' its beginpos there will be a gap. Its beginpos will
' become AreaEndpos, being the true next change in a parameter state.
' If Area.EOF, Area!Endpos is undefined and therefore taken as 0.
If Area.EOF Then
  MaxBeginpos = 0
Else
  MaxBeginpos = Area!EndPos
End If
For Param = 0 To AreaParamNo - 1
  If (AreaBeginpos(Param) < AreaEndpos) And _
    (AreaBeginpos(Param) > MaxBeginpos) Then
    MaxBeginpos = AreaBeginpos(Param)
    CurrentAreaParam = Param
  End If
Next Param
AreaEndpos = MaxBeginpos
Else
  ' the next change in param state (and therefore GC) occurs in
  ' zone param set
  If Zone.EOF Then
    ZoneParamComb(CurrentZoneParam) = 0
    ZoneBeginpos(CurrentZoneParam) = 0
    ZoneGC = 0
```

```
GCID(GCseg) = 0
ElseIf Zone!ParamSet <> ZoneParamSet Then ' enter new zone
' Arriving at this point means, that the next change in AreaParamSet
' will be before (in coordinates) beginpos of the last param state
' segment of the previous zone.
' Everything between that beginpos and the endpos of the first zone
' param state segment will have an invalid ZoneGC.
ZoneParamSet = Zone!ParamSet
ParamSets.FindFirst "ParamSet = " & CStr(ZoneParamSet)
ZoneParamNo = ParamSets!NoParams
For Param = 0 To ZoneParamNo - 1
    ZoneParamID(Param) = ParamSets!ID
    ZoneParamComb(Param) = 0
    ZoneBeginpos(Param) = 0
    ParamSets.MoveNext
Next Param
ZoneGC = 0
GCID(GCseg) = 0
ElseIf ZoneEndpos <= Zone!EndPos Then
    For Param = 0 To ZoneParamNo - 1
        If ZoneParamID(Param) = Zone!Parameter Then
            ZoneParamComb(Param) = Zone!ParamState
            ZoneBeginpos(Param) = Zone!BeginPos
        End If
    Next Param
    ZoneGC = AssocFindGC(ZoneParamSet, ZoneParamNo, ZoneParamComb())
    GCID(GCseg) = AssocFindCGC(AreaGC, ZoneGC)
    Zone.MoveNext
Else
    ZoneParamComb(CurrentZoneParam) = 0
    ZoneGC = 0
    GCID(GCseg) = 0
End If
GCEndpos(GCseg) = ZoneEndpos
If Not GCseg = 0 Then
    If (GCEndpos(GCseg - 1) = GCEndpos(GCseg)) Then
        GCID(GCseg - 1) = GCID(GCseg)
        GCseg = GCseg - 1
    ElseIf GCID(GCseg - 1) = GCID(GCseg) Then
        GCseg = GCseg - 1
    End If
End If
GCseg = GCseg + 1
If Zone.EOF Then
    MaxBeginpos = 0
ElseIf Zone!ParamSet <> ZoneParamSet Then
    MaxBeginpos = 0
Else
    MaxBeginpos = Zone!EndPos
End If
For Param = 0 To ZoneParamNo - 1
    If (ZoneBeginpos(Param) < ZoneEndpos) And _
        (ZoneBeginpos(Param) > MaxBeginpos) Then
        MaxBeginpos = ZoneBeginpos(Param)
        CurrentZoneParam = Param
    End If
Next Param
ZoneEndpos = MaxBeginpos
End If
Loop
MaxGCseg = GCseg + 1
Method.MoveFirst
' Unlike for GC's Tunnel and Geom will be -1, when they are not known.
' This is because Geom may be 0 and because in order to have a
' set of GC/geometry/method all of the variables must be known, i.e. all
' are relevant.
Tunnel = -1
MethodNo = -1
```

```
Method.MoveFirst
GCSeg = 0
AreaEndpos = GCEndpos(GCSeg)
MethodEndpos = AreaEndpos + 1 ' always start with method
Do Until (Method.EOF) And (GCSeg >= MaxGCSeg)
  If MethodEndpos > AreaEndpos Then ' change of method first
    If Method.EOF Then
      ' No more valid sets will occur in this area
      MethodNo = -1
      GCSeg = MaxGCSeg
    ElseIf Method!Tunnel <> Tunnel Then
      ' If we enter a new tunnel we restart at begin (end in coordinate
      ' system) of the GC profile.
      Tunnel = Method!Tunnel
      Geom = Method!Geometry
      MethodNo = -1
      GCSeg = 0
      AreaEndpos = GCEndpos(GCSeg)
      MethodEndpos = Method!EndPos
      SegBegin = MethodEndpos
    ElseIf MethodEndpos <= Method!EndPos Then 'no gap
      MethodNo = Method!Method
      SegBegin = Method!EndPos
      MethodEndpos = Method!BeginPos
      Method.MoveNext
      If Method.EOF Then
        'no changes
      ElseIf Method!Tunnel <> Tunnel Then
        'no changes
      ElseIf MethodEndpos <= Method!EndPos Then
        ' in case the beginpos of the previous segment is smaller than
        ' the endpos of this segment (by error)
        MethodEndpos = Method!EndPos
      End If
    Else
      ' there is a gap between MethodEndpos and Method!Endpos
      MethodNo = -1
      MethodEndpos = Method!EndPos
    End If
  Else ' change in GC first
    GCNo = GCID(GCSeg)
    SegBegin = GCEndpos(GCSeg)
    GCSeg = GCSeg + 1
    If GCSeg = MaxGCSeg Then
      AreaEndpos = -1
    Else
      AreaEndpos = GCEndpos(GCSeg)
    End If
  End If
  If AreaEndpos > MethodEndpos Then
    AssocAdd Geom, GCNo, MethodNo, SegBegin - AreaEndpos
  Else
    AssocAdd Geom, GCNo, MethodNo, SegBegin - MethodEndpos
  End If
Loop
Zone.Close
Area.Close
Method.Close
End If
Areas.MoveNext
Loop
ZoneDef.Close
AreaDef.Close
MethodDef.Close
Areas.Close
ParamSets.Close
AssocCalculate
End Sub
```

```
Private Sub AssocAdd(Geometry As Long, GC As Long, Method As Long, Length As Double)
    Static UP As Recordset
    If IsNull(Geometry) Or GC = 0 Or Method < 0 Or (Length = 0) Then Exit Sub
    Set UP = DB.OpenRecordset("tUpdatePhases", dbOpenDynaset, dbReadOnly)
    GCGeomMeth.Seek "=", Geometry, GC, Method, 10
    If GCGeomMeth.NoMatch Then
        ' this combination has not been predicted and must therefore be added
        ' in all update phases
        UP.MoveFirst
        Do Until UP.EOF
            If UP!ID <> 11 Then
                GCGeomMeth.Seek "=", Geometry, GC, Method, UP!ID
                ' this is just a security check. The seek method should always report NoMatch
                ' because a certain assoc should exist either for all phases or for none, and
                ' it is already clear that it does not exist for phase 10.
                If GCGeomMeth.NoMatch Then
                    GCGeomMeth.addnew
                    GCGeomMeth!CGC = GC
                    GCGeomMeth!Geometry = Geometry
                    GCGeomMeth!Method = Method
                    GCGeomMeth!UpdatePhase = UP!ID
                    If UP!ID = 10 Then
                        GCGeomMeth!Prob = Length
                        ' length over which this combination was observed. the probability
                        ' will be calculated later (see intro comment)
                    Else
                        GCGeomMeth!Prob = 0
                    End If
                    GCGeomMeth.Update
                End If
            End If
            UP.MoveNext
        Loop
    Else
        ' add length of this segment to total length this combination was observed
        GCGeomMeth.Edit
        GCGeomMeth!Prob = GCGeomMeth!Prob + Length
        GCGeomMeth.Update
    End If
    GCGeom.Seek "=", Geometry, GC
    If GCGeom.NoMatch Then
        GCGeom.addnew
        GCGeom!Geometry = Geometry
        GCGeom!GC = GC
        GCGeom!Length = Length
        GCGeom.Update
    Else
        GCGeom.Edit
        GCGeom!Length = GCGeom!Length + Length
        GCGeom.Update
    End If
End Sub

Private Sub AssocCalculate()
    Dim GCNo As Long, Geom As Long
    ' Calculate probabilities from length of geom/gc/meth combination and total length
    ' of geom/gc combination
    If Not (GCGeomMeth.EOF And GCGeomMeth.BOF) Then
        GCGeomMeth.MoveFirst
        GCNo = -1
        Geom = -1
        Do Until GCGeomMeth.EOF
            If GCGeomMeth!UpdatePhase = 10 Then
                If GCNo <> GCGeomMeth!CGC Or Geom <> GCGeomMeth!Geometry Then
                    GCNo = GCGeomMeth!CGC
                    Geom = GCGeomMeth!Geometry
                    GCGeom.Seek "=", Geom, GCNo
                End If
            End If
        Loop
    End If
End Sub
```

```
GCGeomMeth.Edit
If GCGeom.NoMatch Then
    ' the first two cases should logically never happen since the same
    ' procedure creates both geom/gc and geom/gc/meth combinations
    GCGeomMeth!Prob = 0
ElseIf GCGeom!Length <= 0 Then
    GCGeomMeth!Prob = 0
Else
    GCGeomMeth!Prob = GCGeomMeth!Prob / GCGeom!Length
End If
GCGeomMeth.Update
End If
GCGeomMeth.MoveNext
Loop
End If
End Sub

Private Sub AssocInitialize()
    ' set all observed probabilities and lengths to 0
    If Not (GCGeomMeth.BOF And GCGeomMeth.EOF) Then _
        GCGeomMeth.MoveFirst
    Do Until GCGeomMeth.EOF
        If GCGeomMeth!UpdatePhase = 10 Then
            GCGeomMeth.Edit
            GCGeomMeth!Prob = 0
            GCGeomMeth.Update
        End If
        GCGeomMeth.MoveNext
    Loop
    If Not (GCGeom.BOF And GCGeom.EOF) Then GCGeom.MoveFirst
    Do Until GCGeom.EOF
        GCGeom.Edit
        GCGeom!Length = 0
        GCGeom.Update
        GCGeom.MoveNext
    Loop
End Sub

Private Function AssocFindGC(ParamSet As Long, ParamNo As Integer, _
    ParamStat() As Long) As Long
    Static CombID As Long, Hits As Integer, MaxHits As Integer, Counter As Integer
    Static ValidComb As Boolean
    For Counter = 0 To ParamNo - 1
        If ParamStat(Counter) = 0 Then
            ' GC is undefined if any of the param states is undefined
            AssocFindGC = 0
            Exit Function
        End If
    Next Counter
    MaxHits = 0
    ' correct GC is the one which most closely resembles the given param state combination
    ' i.e. the one for which the most parameter states match exactly
    ' Hits counts how many param state could be matched for the current GC combination
    ' MaxHits remembers the maximum reached so far
    AssocFindGC = 0
    GCComb.Seek ">=", ParamSet
    Do Until GCComb!ParamSet <> ParamSet
        ValidComb = True
        CombID = GCComb!ID
        GCCombStat.Seek ">=", CombID
        ' Check whether all param states in the gc comb match one in the observed comb
        If Not GCCombStat.NoMatch Then
            Do Until GCCombStat!GCComb <> CombID ' i.e. for all states in the combination
                Hits = -1 'temporary abuse of variable
                For Counter = 0 To ParamNo - 1
                    ' one of the observed param states must match the one in the gc comb
                    If GCCombStat!ParamState = ParamStat(Counter) Then Hits = 0
                Next Counter
            Loop
        End If
    Loop
End Function
```



```
Next Counter
' if only one param state in the gc comb can NOT be matched, this is not
' the right gc combination
If Hits = -1 Then ValidComb = False
GCCombStat.MoveNext
If GCCombStat.EOF Then Exit Do
Loop
If ValidComb Then
' Check how many observed param states match this gc combination
Hits = 0
For Counter = 0 To ParamNo - 1
GCCombStat.Seek "=", CombID, ParamStat(Counter)
If Not GCCombStat.NoMatch Then Hits = Hits + 1
Next Counter
If Hits > MaxHits Then
AssocFindGC = GCComb!GC
MaxHits = Hits
End If
End If
End If
GCComb.MoveNext
If GCComb.EOF Then Exit Do
Loop
End Function

Private Function AssocFindCGC(AreaGC As Long, ZoneGC As Long) As Long
AssocFindCGC = 0
If AreaGC = 0 Or ZoneGC = 0 Then Exit Function ' neither of them can be unknown
CGC.Seek "=", AreaGC, ZoneGC
If Not CGC.NoMatch Then
AssocFindCGC = CGC!CombGC
Exit Function
End If
CGC.Seek "=", AreaGC, Null
If Not CGC.NoMatch Then
AssocFindCGC = CGC!CombGC
Exit Function
End If
CGC.Seek "=", Null, ZoneGC
If Not CGC.NoMatch Then AssocFindCGC = CGC!CombGC
End Function

Public Sub ZoneUpdate(DB As Database)
Dim Zones As Recordset, Length As Double, EndPos As Double, Status As String
Dim Area As Long, BeginMin As Double, BeginMode As Double, BeginMax As Double
Dim ObsBegin As Double, ZoneStat As Recordset
If IsNull(DLookup("ID", "tUpdatePhases", "[Current] = True")) Then Exit Sub
Set Zones = DB.OpenRecordset("qZoneSeq", dbOpenDynaset)
' Zones has all posterior and observed zones, w/ increasing order and observed first
Set ZoneStat = DB.OpenRecordset("oqZoneParamStat", dbOpenDynaset, dbReadOnly)
Zones.MoveFirst
Area = -1
Do Until Zones.EOF
If Area <> Zones!Area Then
' we are entering a new area
Area = Zones!Area
BeginMin = 0
BeginMode = 0
BeginMax = 0
End If
If Zones!UpdatePhase = 10 Then
' This zone has been observed (at least partially). The subsequent recordset
' will contain the posterior data of the current update phase
Status = Zones!Status
EndPos = Zones!EndposMode
' in case the previous zone had endpos > endpos of the observed zone
```

```
BeginMax = Auxiliary.MinDbl(BeginMax, EndPos)
BeginMode = Auxiliary.MinDbl(BeginMode, EndPos)
BeginMin = Auxiliary.MinDbl(BeginMin, EndPos)
Zones.MoveNext
Else
    Status = "Prior"
End If
' calculate min mode max of parameters not specified by the user input
Auxiliary.CalculateSegmentFwd Zones, BeginMin, BeginMode, BeginMax
If Status <> "Prior" Then ' there was an observed zone
    Zones.Edit
    If Status = "Current" Then
        Zones!GenerationMode = "1"
        ZoneStat.FindLast "Zone = " & CStr(Zones!Zone) ' find last segment
        If ZoneStat.NoMatch Then
            ' we don't know whether this is the zone's true end or only
            ' the tunnel face, in any case endpos is a minimum value of true endpos
            Auxiliary.SetEndposMin Zones, EndPos
            ' the distance from beginmax to endpos is the minimum for length
            Auxiliary.SetLengthMin Zones, EndPos - BeginMax
        ElseIf ZoneStat.Part Then
            ' the observed endpos is the tunnel face (because the last segment
            ' is only partly visible) --> endpos is a minimum value of true endpos
            Auxiliary.SetEndposMin Zones, EndPos
            ' the distance from beginmax to endpos is the absolute minimum for length
            Auxiliary.SetLengthMin Zones, EndPos - BeginMax
        Else
            ' the observed endpos is the true endpos of the zone (because the last
            ' segment was completely observed) --> endpos is exact end of zone
            Auxiliary.SetEndpos Zones, EndPos
            Auxiliary.SetLength Zones, 0
        End If
    Else
        Auxiliary.SetEndpos Zones, EndPos
        Auxiliary.SetLength Zones, 0
    End If
    Zones!Status = Status
    Zones.Update
End If
BeginMin = Zones!EndposMin
BeginMax = Zones!EndposMax
BeginMode = Zones!EndposMode
If BeginMode < 0 Then BeginMode = (BeginMin + BeginMax) / 2
Zones.MoveNext
Loop
Zones.Close
End Sub

Public Sub SegSemiDeterUpdate(DB As Database)
    Dim Parameters As Recordset, Zones As Recordset, Post As Recordset, Obs As Recordset
    Dim Criterion As String, Match As Boolean, Area As Long, Bookmark As Variant
    Dim Order As Double, BeginMin As Double, BeginMode As Double, BeginMax As Double
    Dim UnknownBefore As Boolean, PreviousObsState As Long
    If IsNull(DLookup("ID", "tUpdatePhases", "[Current] = True")) Then Exit Sub
    ' If there is no active update phase, we can't match observed data to posterior one
    ' Also the basic assumption for the rest of the procedure is, that the same parameter
    ' set with a semi-deterministic parameter is only used once in the whole project.

    Set Parameters = DB.OpenRecordset("SELECT [ParamSet], [Parameter] FROM " & _
        "tParam_X_Set WHERE [GenerationMode] = '2'", dbOpenDynaset, dbReadOnly)
    Set Zones = DB.OpenRecordset("SELECT * FROM qZoneSeq WHERE [UpdatePhase] <> 10", _
        dbOpenDynaset, dbReadOnly)
    Set Post = DB.OpenRecordset("qSegParamStateSemi", dbOpenDynaset)
    Set Obs = DB.OpenRecordset("qSegParamStateObs", dbOpenDynaset, dbReadOnly)
    Parameters.MoveFirst
    Do Until Parameters.EOF
        Criterion = "ParamSet = " & CStr(Parameters!ParamSet) & " And Parameter = " & _
            CStr(Parameters!Parameter)
```

```
Post.FindFirst Criterion
Obs.FindFirst Criterion
' find beginpos of this zone/area
If Not Obs.NoMatch Then
    BeginMin = 0
    BeginMode = 0
    BeginMax = 0
    If InStr("Z", Obs!No) Then
        ' i.e. this is a zone parameter set and we have to find beginpos of
        ' the zone. If it is an area param set, beginpos is always 0.
        Zones.FindFirst "ParamSet = " & CStr(Parameters!ParamSet)
        If Zones.NoMatch Then
            MsgBox "Can't find zone which has parameter set " & _
                CStr(Parameters!ParamSet), , "Error preparing segment table"
        Else
            Area = Zones!Area
            Zones.MovePrevious
            If Zones.BOF Then
                ' no changes
            ElseIf Zones!Area = Area Then
                BeginMin = Zones!EndposMin
                BeginMode = Zones!EndposMode
                BeginMax = Zones!EndposMode
            End If
        End If
    End If
End If
SegSemiDeterRecalculatePost Post, BeginMin, BeginMode, BeginMax
Do Until Obs.NoMatch

' first find a match between observation and posterior
If Obs!Part Then
    BeginMode = Obs!EndPos - Obs!Length 'temporary abuse of variable
    ' check if there is an unknown part before this observed segment
    Obs.FindPrevious Criterion
    UnknownBefore = True
    If Obs.NoMatch Then
        If BeginMode = 0 Then UnknownBefore = False
        ' the unknown part is not before the active segment
        Obs.FindFirst Criterion
    ElseIf Obs!EndPos = BeginMode Then
        ' the unknown part is not before the active segment
        UnknownBefore = False
        Obs.FindNext Criterion
    Else
        ' the unknown part is before the active segment
        Obs.FindNext Criterion
    End If
End If
Match = SegSemiDeterFindPost(Post, Obs!ParamState, UnknownBefore, _
    Obs!EndPos, Obs!Length, Obs!Part, Criterion)
If Not Match Then
    ' A new segment must be added before the first posterior segment that
    ' has EndposMax > (Endpos of the observed segment)
    BeginMode = -1 ' temporary abuse of variable
    Do Until Post!EndposMax > Obs!EndPos
        If Post.NoMatch Then
            Post.FindLast Criterion ' will add new segment at the end
            BeginMode = Post!Order + 1
            Exit Do
        End If
        Post.FindNext Criterion
    Loop
    If BeginMode = -1 Then
        BeginMode = Post!Order
        Post.FindPrevious Criterion
        If Post.NoMatch Then
            BeginMode = BeginMode - 1
        End If
    End If
End If
```

```
Else
  BeginMode = (BeginMode + Post!Order) / 2
End If
End If
SegSemiDeterAdd Obs, BeginMode
Post.Requery
' the inserted segment becomes the active one
Post.FindFirst Criterion & " And Order = " & CStr(BeginMode)
End If
If Not (Obs!Part And UnknownBefore) Then
  ' --> otherwise there may be zones that have not yet been observed
  Bookmark = Post.Bookmark
  Post.FindPrevious Criterion
  If Not Post.NoMatch Then
    ' All segments between the last observed one and the currently
    ' observed one do obviously not exist and must have 0 length
    Do Until Post!Order <= Order
      SegSemiDeterSetToZero Post
      Post.FindPrevious Criterion
      If Post.NoMatch Then Exit Do
    Loop
  End If
  Post.Bookmark = Bookmark
End If

' update posterior segment with observed segment
If Obs!Part Then
  If UnknownBefore Then
    BeginMin = Obs!EndPos ' beginmin of next segment!
    Obs.FindNext Criterion
    If Obs.NoMatch Then
      ' endpos may be precise, maybe not
      Obs.FindLast Criterion
      SegSemiDeterCopy Obs, Post, True, True
      BeginMin = Post!EndposMin
      BeginMode = Post!EndposMode
      BeginMax = Post!EndposMax
    ElseIf (Obs!EndPos - Obs!Length = BeginMin) Then
      ' endpos is precise
      Obs.FindPrevious Criterion
      SegSemiDeterCopy Obs, Post, True, False
      BeginMin = Obs!EndPos
      BeginMode = Obs!EndPos
      BeginMax = Obs!EndPos
    Else
      ' endpos is not precise, i.e. both before and after is unknown
      Obs.FindPrevious Criterion
      SegSemiDeterCopy Obs, Post, True, True
      BeginMin = Post!EndposMin
      BeginMode = Post!EndposMode
      BeginMax = Post!EndposMax
    End If
  Else
    ' the unknown part is after the segment
    ' --> the precise endpos of the segment is not yet known.
    SegSemiDeterCopy Obs, Post, False, True
    BeginMin = Post!EndposMin
    BeginMode = Post!EndposMode
    BeginMax = Post!EndposMax
  End If
Else
  ' we know the begin and end pos precisely
  SegSemiDeterCopy Obs, Post, False, False
  BeginMin = Obs!EndPos
  BeginMode = Obs!EndPos
  BeginMax = Obs!EndPos
End If
```

```
If BeginMode < 0 Then BeginMode = (BeginMin + BeginMax) / 2

Order = Post!Order
' remember the order (identification) of the last matched segment
PreviousObsState = Obs!ParamState ' remember param state of this segment
UnknownBefore = Obs!Part ' remember 'partly visible' of this segment
Obs.FindNext Criterion
If Obs.NoMatch Then
    Post.FindNext Criterion
    If Not Post.NoMatch Then _
        SegSemiDeterRecalculatePost Post, BeginMin, BeginMode, BeginMax
    ElseIf PreviousObsState = Obs!ParamState And UnknownBefore _
        And Obs!Part Then
        ' stay with same post recordset because the previous and current
        ' observation may be of the same predicted segment
    Else
        Post.FindNext Criterion
        If Not Post.NoMatch Then _
            SegSemiDeterRecalculatePost Post, BeginMin, BeginMode, BeginMax
    End If
Loop
End If
Parameters.MoveNext
Loop
Obs.Close
Post.Close
Parameters.Close
Zones.Close
End Sub

Private Sub SegSemiDeterCopy(Obs As Recordset, Post As Recordset, _
    UnknownBefore As Boolean, UnknownAfter As Boolean)
    Post.Edit
    If UnknownAfter Then
        ' the observed length and endpos are a minimum of the true length and endpos
        Post!Status = "Current"
        Auxiliary.SetEndposMin Post, Obs!EndPos
        Auxiliary.SetLengthMin Post, Obs!Length
    Else
        If UnknownBefore Then
            ' we know the precise endpos, the length is a minimum
            Post!Status = "Current"
            Post!GenerationMode = "1"
            Auxiliary.SetEndpos Post, Obs!EndPos
            Auxiliary.SetLength Post, 0
        Else
            ' we know both length and endpos
            Post!Status = "Observed"
            Post!GenerationMode = "1" 'there is not generation mode "both" for segments
            Auxiliary.SetEndpos Post, Obs!EndPos
            Auxiliary.SetLength Post, Obs!Length
        End If
    End If
    Post.UpdateEnd Sub

Private Sub SegSemiDeterAdd(Obs As Recordset, Order As Double)
    Static UP As Recordset, SDS As Recordset, SP As Recordset, SegID As Long
    Set UP = DB.OpenRecordset("tUpdatePhases", dbOpenSnapshot)
    Set SDS = DB.OpenRecordset("tSemiDeterSegs", dbOpenTable, dbAppendOnly)
    Set SP = DB.OpenRecordset("tSegProb", dbOpenTable)
    ' This procedure only adds the new segment. Later in SegSemiDeterUpdate the
    ' procedure SegSemiDeterCopy will be called to give the observed length and
    ' endpos to the posterior recordset.
    SP.Index = "PrimaryKey"
    SDS.addnew
    SDS!ParamSet = Obs!ParamSet
    SDS!ParamState = Obs!ParamState
```

```
SDS!Order = Order
SegID = SDS!ID
SDS.Update
UP.MoveFirst
Do Until UP.EOF
    If UP!ID <> 10 And UP!ID <> 11 Then ' do not add in these two update phases
        SP.addnew
        SP!Segment = SegID
        SP!UpdatePhase = UP!ID
        If UP!ID = 1 Then
            SP!Status = "Prior"
        Else
            SP!Status = "Posterior"
        End If
        SP.Update
    End If
    UP.MoveNext
Loop
UP.Close
SDS.Close
SP.Close
End Sub

Private Sub SegSemiDeterSetToZero(Post As Recordset)
    Post.Edit
    Post!Status = "Observed"
    Post!GenerationMode = "0"
    Auxiliary.SetEndpos Post, 0
    Auxiliary.SetLength Post, 0
    Post.Update
End Sub

Private Function SegSemiDeterFindPost(Post As Recordset, ParamState As Long, _
    UnknownBefore As Boolean, EndPos As Double, Length As Double, _
    Part As Boolean, Criterion As String) As Boolean
    Static Bookmark As Variant
    Bookmark = Post.Bookmark
    Do Until Post.NoMatch
        If Post!ParamState = ParamState Then
            If Part Then
                ' endpos is precise, observed length is a minimum
                If UnknownBefore And _
                    Post!EndposMax + 0.2 * Length >= EndPos And _
                    Post!EndposMin - 0.2 * Length <= EndPos And _
                    Post!LengthMax * 1.2 >= Length Then
                    SegSemiDeterFindPost = True
                    Exit Function
                ' observed endpos and length are a minimum of true values
            ElseIf (Not UnknownBefore) And _
                Post!EndposMax + 0.2 * Length >= EndPos And _
                Post!LengthMax * 1.2 >= Length Then
                SegSemiDeterFindPost = True
                Exit Function
            End If
        Else
            If Post!EndposMax + 0.2 * Length >= EndPos And _
                Post!EndposMin - 0.2 * Length <= EndPos And _
                Post!LengthMax * 1.2 >= Length And _
                Post!LengthMin * 0.8 <= Length Then
                SegSemiDeterFindPost = True
                Exit Function
            End If
        End If
        Post.FindNext Criterion
    Loop
    SegSemiDeterFindPost = False
End Function
```

```
Post.Bookmark = Bookmark
End Function

Private Sub SegSemiDeterRecalculatePost(Post As Recordset, ByVal BeginMin As Double, _
    ByVal BeginMode As Double, ByVal BeginMax As Double)
    Static Bookmark As Variant, EndposMin As Double, EndposMax As Double
    Bookmark = Post.Bookmark
    Do Until Post.EOF
        Auxiliary.CalculateSegmentFwd Post, BeginMin, BeginMode, BeginMax
        BeginMin = Post!EndposMin
        BeginMode = Post!EndposMode
        BeginMax = Post!EndposMax
        If BeginMode < 0 Then BeginMode = (BeginMin + BeginMax) / 2
        Post.MoveNext
    Loop
    Post.Bookmark = Bookmark
End Sub

Public Sub SegDeterUpdate(DB As Database)
    Dim Parameters As Recordset, Zones As Recordset, Post As Recordset, Obs As Recordset
    Dim Criterion As String, Match As Boolean, EndPos As Double, Bookmark As Variant
    If IsNull(DLookup("ID", "tUpdatePhases", "[Current] = True")) Then Exit Sub
    Set Parameters = DB.OpenRecordset("SELECT [ParamSet], [Parameter] FROM tParam_X_Set " & _
        & "WHERE [GenerationMode] = '1'", dbOpenDynaset, dbReadOnly)
    Set Zones = DB.OpenRecordset("qZoneSeq", dbOpenDynaset, dbReadOnly)
    Set Post = DB.OpenRecordset("qSegParamStateDeter", dbOpenDynaset)
    Set Obs = DB.OpenRecordset("qSegParamStateObs", dbOpenDynaset, dbReadOnly)
    If Parameters.EOF And Parameters.BOF Then Exit Sub
    Parameters.MoveFirst
    Do Until Parameters.EOF
        Criterion = "ParamSet = " & CStr(Parameters!ParamSet) & " And Parameter = " & _
            CStr(Parameters!Parameter)
        Post.FindFirst Criterion
        Obs.FindFirst Criterion
        If Not Obs.NoMatch Then
            ' Endpos is the position of the previous segment
            If Obs!Part Then
                ' we can't be sure that this partially known segment is at the beginning
                ' of the zone, 0 is a minimum
                EndPos = 0
            Else
                ' if the segment is fully observed, any previous one has endpos as follows
                EndPos = Obs!EndPos - Obs!Length
            End If
        End If
        Do Until Obs.NoMatch
            If Post.NoMatch Then
                SegDeterAdd Obs, Criterion
                Post.Requery
                Post.FindFirst Criterion & " And Endpos = " & CStr(Obs!EndPos)
                ' check if there is an unknown part before this observed segment
            ElseIf EndPos = Obs!EndPos - Obs!Length Then
                ' the unknown part is not before the active segment
                If Post!ParamState = Obs!ParamState Then
                    Post.Edit
                    Post!EndPos = Obs!EndPos
                    If Obs!Part Then
                        Post!Status = "Current"
                    Else
                        Post!Status = "Observed"
                    End If
                    Post.Update
                Else
                    ' must remember the last segment that was matched to observations
                    EndPos = Post!EndPos 'temporary abuse of variable
                    Match = SegDeterFindPost(Post, Obs!ParamState, Obs!EndPos, Criterion)
                    If Match Then
```

```
      ' All segments between the last observed one and the currently
      ' observed one do obviously not exist and are deleted
      Bookmark = Post.Bookmark
      Post.FindPrevious Criterion
      Do Until Post!EndPos <= EndPos
        Post.delete
        Post.FindPrevious Criterion
      Loop
      Post.Bookmark = Bookmark
      Post.Edit
      Post!EndPos = Obs!EndPos
      If Obs!Part Then
        Post!Status = "Current"
      Else
        Post!Status = "Observed"
      End If
      Post.Update
    Else
      SegDeterAdd Obs, Criterion
      Post.Requery
      Post.FindFirst Criterion & " And Endpos = " & CStr(Obs!EndPos)
    End If
  End If
Else
  ' there is an unknown part before the active segment
  Match = SegDeterFindPost(Post, Obs!ParamState, Obs!EndPos, Criterion)
  If Match Then
    Post.Edit
    Post!EndPos = Obs!EndPos
    If Obs!Part Then
      Post!Status = "Current"
    Else
      Post!Status = "Observed"
    End If
    Post.Update
  Else
    SegDeterAdd Obs, Criterion
    Post.Requery
    Post.FindFirst Criterion & " And Endpos = " & CStr(Obs!EndPos)
  End If
End If
Bookmark = Post.Bookmark
Post.FindNext Criterion
' all subsequent endpos must be >= endpos of the active observed segment
Do Until Post.NoMatch
  If Post!EndPos < Obs!EndPos Then
    Post.Edit
    Post!EndPos = Obs!EndPos
    Post.Update
  Else
    Exit Do
  End If
  Post.FindNext Criterion
Loop
Post.Bookmark = Bookmark
EndPos = Obs!EndPos
Post.FindNext Criterion
Obs.FindNext Criterion
Loop
Parameters.MoveNext
Loop
Obs.Close
Post.Close
Parameters.Close
Zones.Close
End Sub
```



```
Private Sub SegDeterAdd(Obs As Recordset, Criterion As String)
    Static UP As Recordset, DS As Recordset
    Set UP = DB.OpenRecordset("tUpdatePhases", dbOpenSnapshot)
    Set DS = DB.OpenRecordset("tDeterSegs", dbOpenDynaset)
    UP.FindFirst "Current = True"
    ' only add a segment to the active update phase (unlike semi deter segments or zones)
    If Not UP.NoMatch Then
        DS.addnew
        DS!ParamSet = Obs!ParamSet
        DS!ParamState = Obs!ParamState
        DS!EndPos = Obs!EndPos
        DS!UpdatePhase = UP!ID
        If Obs!Part Then
            DS!Status = "Current"
        Else
            DS!Status = "Observed"
        End If
        DS.Update
    End If
End Sub
```

```
Private Function SegDeterFindPost(Post As Recordset, ParamState As Long, _
    EndPos As Double, Criterion As String) As Boolean
    Static Bookmark As Variant, Begin As Double, Length As Double
    Bookmark = Post.Bookmark
    Post.FindPrevious Criterion
    If Post.NoMatch Then
        Begin = 0 ' this yields an upper limit to the true length since zone beginpos > 0
    Else
        Begin = Post!EndPos
    End If
    Post.FindNext Criterion
    Do Until Post.NoMatch
        Length = Post!EndPos - Begin
        Begin = Post!EndPos
        If Post!ParamState = ParamState And _
            Post!EndPos >= EndPos - 0.2 * Length And _
            Post!EndPos <= EndPos + 0.2 * Length Then
            SegDeterFindPost = True
            Exit Function
        End If
        Post.FindNext Criterion
    Loop
    SegDeterFindPost = False
    Post.Bookmark = Bookmark
End Function
```

```
Private Sub TransProbUpdate(DB As Database)
    Dim RS As Recordset, Section As Long, Parameter As Long, FromParamState As Long
    Dim QueryName As String, EndPos As Double
    ' set all probabilities and total counts to 0. First simply transitions are
    ' counted and only at the end probabilities are calculated.
    ' (same system as geom/gc/meth assoc)
    TransProbInitialize
    QueryName = "oqZoneParamStat"
    Do Until QueryName = ""
        Set RS = DB.OpenRecordset(QueryName, dbOpenDynaset, dbReadOnly)
        Section = -1
        Parameter = -1
        FromParamState = -1
        EndPos = 0
        Do Until RS.EOF
            If (Section <> RS(1)) Or (Parameter <> RS!Parameter) Then
                Section = RS(1)
                Parameter = RS!Parameter
            Else
                ' transition counts only if there is no gap between the two segments
            End If
        Loop
    Loop
End Sub
```

```
        If EndPos = RS!EndPos - RS!Length Then _
            TransProbAdd RS!ParamSet, FromParamState, RS!ParamState
        End If
        FromParamState = RS!ParamState
        EndPos = RS!EndPos
        RS.MoveNext
    Loop
    RS.Close
    If QueryName = "oqZoneParamStat" Then
        ' do the same thing for area param state sequence
        QueryName = "oqAreaParamStat"
    Else
        QueryName = ""
    End If
Loop
TransProbCalculate
End Sub

Private Sub TransProbAdd(ParamSet As Long, FromParamState As Long, ToParamState As Long)
    If (ParamSet < 0) Or (FromParamState < 0) Or (ToParamState < 0) Then Exit Sub
    TransProb.Seek "=", ParamSet, FromParamState, ToParamState, 10
    If TransProb.NoMatch Then
        TransProb.addnew
        TransProb!ParamSet = ParamSet
        TransProb!FromParamState = FromParamState
        TransProb!ToParamState = ToParamState
        TransProb!UpdatePhase = 10
        TransProb!Prob = 1
        TransProb.Update
    Else
        TransProb.Edit
        ' increase the number this transition was observed by 1
        TransProb!Prob = TransProb!Prob + 1
        TransProb.Update
    End If
    Trans.Seek "=", ParamSet, FromParamState
    If Trans.NoMatch Then
        Trans.addnew
        Trans!ParamSet = ParamSet
        Trans!ParamState = FromParamState
        Trans!Number = 1
        Trans.Update
    Else
        Trans.Edit
        ' increase the total number of transitions from this param state by 1
        Trans!Number = Trans!Number + 1
        Trans.Update
    End If
End Sub

Private Sub TransProbCalculate()
    Dim ParamSet As Long, ParamState As Long
    If Not Trans.EOF And Trans.BOF Then
        Trans.MoveFirst
        Do Until Trans.EOF
            If Trans!Number = 0 Then Trans.delete
            Trans.MoveNext
        Loop
    End If
    TransProb.MoveFirst
    ParamSet = -1
    ParamState = -1
    Do Until TransProb.EOF
        If TransProb!Prob = 0 Then
            TransProb.delete
        ElseIf TransProb!UpdatePhase = 10 Then
            If ParamSet <> TransProb!ParamSet Or _
```

```
ParamState <> TransProb!FromParamState Then
ParamSet = TransProb!ParamSet
ParamState = TransProb!FromParamState
Trans.Seek "=", ParamSet, ParamState
End If
If Trans.NoMatch Then
TransProb!Prob = 0
ElseIf Trans!Number = 0 Then
TransProb!Prob = 0
Else
TransProb.Edit
TransProb!Prob = TransProb!Prob / Trans!Number
TransProb.Update
End If
End If
TransProb.MoveNext
Loop
End Sub

Private Sub TransProbInitialize()
If Not (TransProb.EOF And TransProb.BOF) Then TransProb.MoveFirst
Do Until TransProb.EOF
If TransProb!UpdatePhase = 10 Then
TransProb.Edit
TransProb!Prob = 0
TransProb.Update
End If
TransProb.MoveNext
Loop
If Not (Trans.EOF And Trans.BOF) Then Trans.MoveFirst
Do Until Trans.EOF
Trans.Edit
Trans!Number = 0
Trans.Update
Trans.MoveNext
Loop
End Sub

Private Sub TunDelayCopy(Obs As Recordset, Post As Recordset)
Post.Edit
If Obs!Part Then ' observed duration is a minimum
Post!Status = "Current"
If Post!DurationMin < Obs!Duration Then Post!DurationMin = Obs!Duration
If (Post!DurationMode >= 0) And (Post!DurationMode < Obs!Duration) Then _
Post!DurationMode = Obs!Duration
If Post!DurationMax < Obs!Duration Then Post!DurationMax = Obs!Duration
Else ' observed duration is precise
Post!Status = "Observed"
Post!DurationMin = Obs!Duration
Post!DurationMode = Obs!Duration
Post!DurationMax = Obs!Duration
End If
If Post!Type = 0 Then ' time-dependent
Post!Start = Obs!Time
Else ' position-dependent
Post!Start = Obs!Position
End If
Post.Update
End Sub

Private Sub TunDelayAdd(Obs As Recordset)
Static UP As Recordset, TunDelay As Recordset, DelayProb As Recordset, Delay As Long
Set UP = DB.OpenRecordset("tUpdatePhases", dbOpenSnapshot)
Set TunDelay = DB.OpenRecordset("tTunDelays", dbOpenTable, dbAppendOnly)
Set DelayProb = DB.OpenRecordset("tTunDelayProb", dbOpenTable)
DelayProb.Index = "PrimaryKey"
TunDelay.addnew
```

```
TunDelay!Tunnel = Obs!Tunnel
Delay = TunDelay!ID
TunDelay.Update
UP.MoveFirst
Do Until UP.EOF
    If UP!ID <> 10 And UP!ID <> 11 Then
        DelayProb.addnew
        DelayProb!Delay = Delay
        DelayProb!UpdatePhase = UP!ID
        ' any new delay is added as time-dependent
        DelayProb!Type = 0
        DelayProb!Start = Obs!Time
        If UP!ID = 1 Then
            DelayProb!Status = "Prior" ' by default durations are 0
        ElseIf UP!Current Then
            If Obs!Part Then
                DelayProb!Status = "Current"
            Else
                DelayProb!Status = "Observed"
            End If
            DelayProb!DurationMin = Obs!Duration
            DelayProb!DurationMode = Obs!Duration
            DelayProb!DurationMax = Obs!Duration
        Else
            DelayProb!Status = "Posterior" ' by default durations are 0
        End If
        DelayProb.Update
        DelayProb.Seek "=", Delay, UP!ID
    End If
    UP.MoveNext
Loop
UP.Close
TunDelay.Close
DelayProb.Close
End Sub

Private Sub TunDelaySetToZero(Post As Recordset)
    Post.Edit
    Post!Status = "Observed"
    Post!DurationMin = 0
    Post!DurationMode = 0
    Post!DurationMax = 0
    Post.Update
End Sub

Private Function TunDelayFindPost(Post As Recordset, Time As Double, _
    Position As Double, Duration As Double, Part As Boolean, _
    Criterion As String) As Boolean
    Static Bookmark As Variant
    Post.FindFirst Criterion
    If Post.NoMatch Then
        TunDelayFindPost = False
        Exit Function
    End If
    Do Until Post.NoMatch
        If (Part Or Post!DurationMin * 0.8 <= Duration) And _
            Post!DurationMax * 1.2 >= Duration Then
            If (Post!Type = 0 And _
                Post!Start >= Time - Duration And _
                Post!Start <= Time + Duration) Or _
                (Post!Type = 1 And _
                Post!Start >= 0.75 * Position And _
                Post!Start <= 1.25 * Position) Then
                TunDelayFindPost = True
                Exit Function
            End If
        End If
    End If
End Function
```

```
Post.FindNext Criterion
Loop
TunDelayFindPost = False
End Function

Public Sub TunDelayUpdate(DB As Database)
Dim Tunnels As Recordset, Post As Recordset, Obs As Recordset
Dim Criterion As String, Match As Boolean, Tunnel As Long, Bookmark As Variant
If IsNull(DLookup("ID", "tUpdatePhases", "[Current] = True")) Then Exit Sub
Set Tunnels = DB.OpenRecordset("tTunnels", dbOpenTable, dbReadOnly)
Set Post = DB.OpenRecordset("qTunDelays", dbOpenDynaset)
Set Obs = DB.OpenRecordset("oqTunDelays", dbOpenDynaset, dbReadOnly)
Do Until Tunnels.EOF
Criterion = "Tunnel = " & CStr(Tunnels!ID)
Obs.FindFirst Criterion
Do Until Obs.NoMatch
Match = TunDelayFindPost(Post, Obs!Time, Obs!Position, Obs!Duration, _
Obs!Part, Criterion)

If Match Then
TunDelayCopy Obs, Post
Else
TunDelayAdd Obs
End If
Post.Requery
Obs.FindNext Criterion
Loop
Obs.FindLast Criterion
If Not Obs.NoMatch Then
Post.FindFirst Criterion
Do Until Post.NoMatch
If Not Post!Status = "Observed" Then
' all delays with predicted time or position before the last
' observed one obviously did not occur and are set to zero
' note that "position before" depends on the advance direction of the
' tunnel which is determined by comparing Beginpos and Endpos
If (Post!Type = 0 And Post!Start < Obs!Time) Or (Post!Type = 1 And _
Tunnels!BeginPos < Tunnels!EndPos And Post!Start < Obs!Position) Or _
(Post!Type = 1 And Tunnels!BeginPos > Tunnels!EndPos And _
Post!Start > Obs!Position) Then
TunDelaySetToZero Post
End If
End If
Post.FindNext Criterion
Loop
End If
Tunnels.MoveNext
Loop
Obs.Close
Post.Close
Tunnels.Close
End Sub

Private Sub MarkovUpdate()
Dim LengthsPrior As Recordset, Lengths As Recordset
Dim ZoneSeq As Recordset, AreaSeq As Recordset, Graph As Recordset
Dim Criterion As String, LengthTot As Double, LengthNo As Integer
Dim ParamSet As Long, ParamState As Long, Post(2) As Double
Set Lengths = DB.OpenRecordset("tMarkovLengths", dbOpenTable)
Lengths.Index = "PrimaryKey"
Set LengthsPrior = DB.OpenRecordset( _
"SELECT * FROM tMarkovLengths WHERE UpdatePhase = 1", dbOpenDynaset, dbReadOnly)
If LengthsPrior.BOF And LengthsPrior.EOF Then
LengthsPrior.Close
Exit Sub
End If
Set Graph = DB.OpenRecordset("otMarkovGraph", dbOpenTable)
Set ZoneSeq = DB.OpenRecordset("oqZoneParamStat", dbOpenDynaset, dbReadOnly)
```

```
Set AreaSeq = DB.OpenRecordset("oqAreaParamStat", dbOpenDynaset, dbReadOnly)
LengthsPrior.MoveFirst
Do Until LengthsPrior.EOF
    ParamSet = LengthsPrior!ParamSet
    ParamState = LengthsPrior!ParamState
    LengthTot = 0
    LengthNo = 0
    Criterion = "ParamSet = " & CStr(ParamSet) & " And ParamState = " & _
        CStr(ParamState) & " And Part = False"
    ZoneSeq.FindFirst Criterion
    ' find the total length over which this state has been observed and how many times
    Do Until ZoneSeq.NoMatch
        LengthNo = LengthNo + 1
        LengthTot = LengthTot + ZoneSeq!Length
        ZoneSeq.FindNext Criterion
    Loop
    AreaSeq.FindFirst Criterion
    ' same for area
    Do Until AreaSeq.NoMatch
        LengthNo = LengthNo + 1
        LengthTot = LengthTot + AreaSeq!Length
        AreaSeq.FindNext Criterion
    Loop
    ' calculate likelihood graph
    Graphs.GraphMarkovLikelihood LengthsPrior, Graph, LengthTot, LengthNo
    ' calculate posterior distribution analytically
    Graphs.MarkovCalculatePosterior LengthsPrior, Post(), LengthTot, LengthNo
    ' store calculated distribution under update phase 10
    Lengths.Seek "=", ParamSet, ParamState, 10
    If Not Lengths.NoMatch Then
        Lengths.Edit
        Lengths!LengthMin = Post(0)
        Lengths!LengthMode = Post(1)
        Lengths!LengthMax = Post(2)
        Lengths.Update
    End If
    LengthsPrior.MoveNext
Loop
LengthsPrior.Close
Lengths.Close
AreaSeq.Close
ZoneSeq.Close
Graph.Close
End Sub

Private Sub VariablesUpdate(DB As Database, SourceName As String)
    Static Variables As Recordset, Observations As Recordset, Graph As Recordset
    Set Variables = DB.OpenRecordset("t" & SourceName, dbOpenTable, dbReadOnly)
    If Variables.BOF And Variables.EOF Then Exit Sub
    ' this procedure is used for cycle lengths, method variables and general variables
    Select Case SourceName
        Case "Variables"
            Set Observations = DB.OpenRecordset("otVariables", dbOpenTable, dbReadOnly)
            Set Graph = DB.OpenRecordset("otVariablesGraph", dbOpenTable)
        Case "GenVariables"
            Set Observations = DB.OpenRecordset("otGenVariables", dbOpenTable, dbReadOnly)
            Set Graph = DB.OpenRecordset("otGenVariablesGraph", dbOpenTable)
        Case "Heads"
            Set Observations = DB.OpenRecordset("otCycleLengths", dbOpenTable, dbReadOnly)
            Set Graph = DB.OpenRecordset("otCycleGraph", dbOpenTable)
    End Select
    Variables.MoveFirst
    Do Until Variables.EOF
        ' For every variable/cycle length the graph of observed frequencies is calculated
        Graphs.GraphVariableObserved Observations, Graph, Variables!ID, _
            Variables!Timelimit
        Variables.MoveNext
    Loop
```

```
Variables.Close
Observations.Close
Graph.Close
End Sub

Private Sub ChangeDelaysUpdate(DB As Database)
    Static Delays As Recordset, Observations As Recordset, Graph As Recordset
    Static UP As Recordset
    Set Delays = DB.OpenRecordset("tChangeDelays", dbOpenTable)
    Delays.Index = "PrimaryKey"
    Set UP = DB.OpenRecordset("tUpdatePhases", dbOpenTable, dbReadOnly)
    Set Observations = DB.OpenRecordset("otChangeDelays", dbOpenTable, dbReadOnly)
    ' check if delays were observed for which no prior input exists
    If Not Observations.EOF And Observations.EOF Then
        Observations.MoveFirst
        Do Until Observations.EOF
            Delays.Seek "=", Observations.FromMethod, Observations.ToMethod, 1
            If Delays.NoMatch Then
                ' create delay in all update phases except 10 and 11
                UP.MoveFirst
                Do Until UP.EOF
                    If UP.ID <> 10 And UP.ID <> 11 Then
                        Delays.addnew
                        Delays.FromMethod = Observations.FromMethod
                        Delays.ToMethod = Observations.ToMethod
                        Delays.UpdatePhase = UP.ID
                        Delays.Delay = 0
                        Delays.Update
                    End If
                    UP.MoveNext
                Loop
            End If
            Observations.MoveNext
        Loop
    End If
    Set Graph = DB.OpenRecordset("otChangeDelaysGraph", dbOpenTable)
    If Delays.EOF And Delays.EOF Then Exit Sub
    Delays.MoveFirst
    Do Until Delays.EOF
        ' For every change delay the graph of observed frequencies is calculated
        Graphs.GraphChangeDelaysObserved Observations, Graph, Delays.FromMethod, _
            Delays.ToMethod
        Delays.MoveNext
    Loop
    Delays.Close
    Observations.Close
    Graph.Close
End Sub

Private Sub TunnelInfoUpdate(DB As Database)
    Dim ObsMethodsDef As QueryDef, ObsMethods As Recordset, Tunnels As Recordset
    Dim Areas As Recordset, Bookmark As Variant, OppBegin As Double

    Set Areas = DB.OpenRecordset("tAreas", dbOpenTable, dbReadOnly)
    Set Tunnels = DB.OpenRecordset("tTunnels", dbOpenTable)
    Tunnels.Index = "Area"
    Set ObsMethodsDef = DB.QueryDefs("oqMethod")

    If Areas.EOF And Areas.EOF Then Exit Sub
    If Tunnels.EOF And Tunnels.EOF Then Exit Sub
    Areas.MoveFirst
    Do Until Areas.EOF
        ObsMethodsDef.Parameters("AreaID") = Areas.ID
        Set ObsMethods = ObsMethodsDef.OpenRecordset
        Tunnels.Seek "=", Areas.ID
        Do Until Tunnels.NoMatch
            ' the records are sorted by decreasing endpos --> the first endpos is the current
```

```
' position for a tunnel advancing in positive direction, the last beginpos
' is the current position for a tunnel advancing in negative direction
If Tunnels!BeginPos < Tunnels!EndPos Then
    ObsMethods.FindFirst "Tunnel = " & CStr(Tunnels!ID)
    Tunnels.Edit
    Tunnels!UpdatedBeginPos = ObsMethods!EndPos
    Tunnels.Update
Else
    ObsMethods.FindLast "Tunnel = " & CStr(Tunnels!ID)
    Tunnels.Edit
    Tunnels!UpdatedBeginPos = ObsMethods!BeginPos
    Tunnels.Update
End If
Tunnels.MoveNext
If Tunnels.EOF Then Exit Do
If Tunnels!Area <> Areas!ID Then Exit Do
Loop
ObsMethods.Close
Areas.MoveNext
Loop
Areas.Close

Tunnels.Index = "PrimaryKey"
Tunnels.MoveFirst
Do Until Tunnels.EOF
    ' the updated endpos of a tunnel is the updated begin pos of its opposite tunnel
    ' or the initial endpos if there is no opposite tunnel
    If IsNull(Tunnels!OppTunnel) Then
        OppBegin = Tunnels!EndPos
    Else
        Bookmark = Tunnels.Bookmark
        Tunnels.Seek "=", Tunnels!OppTunnel
        OppBegin = Tunnels!UpdatedBeginPos
        Tunnels.Bookmark = Bookmark
    End If
    Tunnels.Edit
    Tunnels!UpdatedEndPos = OppBegin
    Tunnels.Update
    Tunnels.MoveNext
Loop
Tunnels.Close
End Sub
```

Module: Results

```
Option Compare Database
Option Explicit
Dim Path As String, DB As Database, FS As Object, ImpExp1 As Recordset

Public Sub ImportTimeCost(UpdatePhase As Long)
    Dim TimeCost As Recordset
    Path = ImportPreparations(UpdatePhase)
    If Path = "" Then
        ImportFinish
        Exit Sub
    End If
    ManageUpdatePhases.ExecuteDeleteQuery "rTimeCost", UpdatePhase
    ManageUpdatePhases.ExecuteDeleteQuery "rTimeCostGraph", UpdatePhase
    Set TimeCost = DB.OpenRecordset("rtTimeCost", dbOpenTable)
    DB.Execute ("ImpExp1Del")
    On Error GoTo Repair
    DoCmd.TransferText acImportFixed, "Text ImpSpec", "ImpExp1", _
        Path & "\time_cost.out", 0
    On Error GoTo 0
    Set ImpExp1 = DB.OpenRecordset("ImpExp1", dbOpenTable, dbReadOnly)
    ImpExp1.Index = "PrimaryKey"
    If ImpExp1.EOF And ImpExp1.BOF Then Exit Sub
```



```
ImpExpl.MoveFirst
Do Until ImpExpl.EOF
    TimeCost.addnew
    TimeCost!UpdatePhase = UpdatePhase
    TimeCost!Time = ImpExpl!F1
    ImpExpl.MoveNext
    TimeCost!Cost = ImpExpl!F1
    ImpExpl.MoveNext
    TimeCost.Update
Loop
TimeCost.Close
DB.Execute ("ImpExplDel")
Graphs.GraphTimeCostFrequency UpdatePhase, DB
ImportFinish
Exit Sub
Repair:
    ImportFinish
    MsgBox "Can't find the file 'time_cost.out'", , "IMPORT TIME-COST RESULTS"
End Sub

Public Sub ImportTimePos(UpdatePhase As Long)
    Dim TimePos As Recordset, Areas As Recordset, StepSize As Double, TunnelNo As Long
    Dim k As Integer, Position As Double, Time As Double, Stat As Recordset
    Dim TimePosGraph As Recordset, LastMin As Double, LastMax As Double
    Dim Bookmark As Variant, MaxMin As Double, MaxMinPos As Double
    Path = ImportPreparations(UpdatePhase)
    If Path = "" Then
        ImportFinish
        Exit Sub
    End If
    Set Areas = DB.OpenRecordset("tAreas", dbOpenTable, dbReadOnly)
    If Areas.EOF And Areas.BOF Then
        ImportFinish
        Exit Sub
    End If
    Areas.MoveFirst
    Position = 0 'temporary abuse of variable
    Do Until Areas.EOF
        If Areas!Length > Position Then Position = Areas!Length
        Areas.MoveNext
    Loop
    Areas.Close
    StepSize = Position / 200
    ManageUpdatePhases.ExecuteDeleteQuery "rTimePosGraph", UpdatePhase
    Set TimePos = DB.OpenRecordset("rtTimePosGraph", dbOpenTable)
    TimePos.Index = "PrimaryKey"
    DB.Execute ("ImpExplDel")
    On Error GoTo Repair
    DoCmd.TransferText acImportFixed, "Text ImpSpec", "ImpExpl", _
        Path & "\min_max.out", 0
    On Error GoTo 0
    Set ImpExpl = DB.OpenRecordset("ImpExpl", dbOpenTable, dbReadOnly)
    ImpExpl.Index = "PrimaryKey"
    If ImpExpl.EOF And ImpExpl.BOF Then Exit Sub
    ImpExpl.MoveFirst
    Do Until ImpExpl.EOF
        Time = CDBl(Mid(ImpExpl!F1, InStr(ImpExpl!F1, "-") + 1))
        ImpExpl.MoveNext
        TunnelNo = 0
        Do Until InStr(ImpExpl!F1, "nit") > 0
            If ImpExpl!F1 = "0" Then
                ImpExpl.Move (6)
            Else
                For k = 1 To 2
                    ImpExpl.Move (2)
                    Position = (Int(CDBl(ImpExpl!F1) / StepSize + 0.5)) * StepSize
                    TimePos.Seek "=", TunnelNo, UpdatePhase, Position, "min"
                    If TimePos.NoMatch Then
```

```
        TimePos.addnew
        TimePos!Tunnel = TunnelNo
        TimePos!UpdatePhase = UpdatePhase
        TimePos!Position = Position
        TimePos!Title = "min"
        TimePos!Time = Time
        TimePos.Update
    ElseIf TimePos!Time > Time Then
        TimePos.Edit
        TimePos!Time = Time
        TimePos.Update
    End If
    TimePos.Seek "=", TunnelNo, UpdatePhase, Position, "max"
    If TimePos.NoMatch Then
        TimePos.addnew
        TimePos!Tunnel = TunnelNo
        TimePos!UpdatePhase = UpdatePhase
        TimePos!Position = Position
        TimePos!Title = "max"
        TimePos!Time = Time
        TimePos.Update
    ElseIf TimePos!Time < Time Then
        TimePos.Edit
        TimePos!Time = Time
        TimePos.Update
    End If
Next k
ImpExpl.Move (2)
End If
TunnelNo = TunnelNo + 1
If ImpExpl.EOF Then Exit Do
Loop
Loop
Set TimePosGraph = DB.OpenRecordset("rqTimePosGraph", dbOpenDynaset)
For k = 0 To TunnelNo - 1
    TimePosGraph.FindFirst "Tunnel = " & CStr(k) & _
        " And UpdatePhase = " & CStr(UpdatePhase)
    If Not TimePosGraph.NoMatch Then
        Time = TimePosGraph!Min ' temporary abuse of variable
        TimePosGraph.FindLast "Tunnel = " & CStr(k) & _
            " And UpdatePhase = " & CStr(UpdatePhase)
        If Time < TimePosGraph!Min Then
            Time = 1 ' tunnel advances in positive direction
        Else
            Time = -1 ' tunnel advances backwards
        End If
        If Time > 0 Then TimePosGraph.FindFirst "Tunnel = " & CStr(k) & _
            " And UpdatePhase = " & CStr(UpdatePhase)
        ' this makes sure that min and max start at the same time at the beginning
        ' of each tunnel
        LastMin = TimePosGraph!Min
        LastMax = LastMin
        TimePos.Seek "=", k, UpdatePhase, TimePosGraph!Position, "max"
        TimePos.Edit
        TimePos!Time = LastMin
        TimePos.Update
        MaxMin = -1
        MaxMinPos = -1
        Do Until TimePosGraph!Tunnel <> k Or TimePosGraph!UpdatePhase <> UpdatePhase
            Bookmark = TimePosGraph.Bookmark
            ' this eliminates the part of the plot after the peak and also downward
            ' jumps of the curve due to missing real max values
            If TimePosGraph!Max <= LastMax Then
                TimePos.Seek "=", k, UpdatePhase, TimePosGraph!Position, "max"
                TimePosGraph.FindFirst "Tunnel = " & CStr(k) & " And UpdatePhase = " & _
                    & CStr(UpdatePhase) & " And max > " & CStr(LastMax)
                If TimePosGraph.NoMatch Then
                    TimePos.delete
                End If
            End If
        Loop
    End If
Next k
```

```
        TimePosGraph.MoveFirst
    Else
        TimePos.Edit
        TimePos!Time = LastMax
        TimePos.Update
    End If
    TimePosGraph.Bookmark = Bookmark
Else
    LastMax = TimePosGraph!Max
End If
' if the new "min" point is too close to the maximum this means that
' there is no real min value at this position and that it must be
' neglected when determining the greatest min value
If MaxMin < TimePosGraph!Min Then
    TimePos.Seek "=", k, UpdatePhase, TimePosGraph!Position, "max"
    If TimePosGraph!Min < 0.95 * LastMax Then
        MaxMin = TimePosGraph!Min
        MaxMinPos = TimePosGraph!Position
    End If
End If
TimePosGraph.Move (Time)
LastMin = TimePosGraph!Min
If TimePosGraph.BOF Or TimePosGraph.EOF Then Exit Do
Loop
TimePosGraph.Close
Set TimePosGraph = DB.OpenRecordset("rqTimePosGraph", dbOpenDynaset)
' position at the end of tunnel and go backwards
If Time > 0 Then
    TimePosGraph.FindLast "Tunnel = " & CStr(k) & _
        " And UpdatePhase = " & CStr(UpdatePhase)
Else
    TimePosGraph.FindFirst "Tunnel = " & CStr(k) & _
        " And UpdatePhase = " & CStr(UpdatePhase)
End If
LastMin = MaxMin
Do Until TimePosGraph!Tunnel <> k Or TimePosGraph!UpdatePhase <> UpdatePhase
    TimePos.Seek "=", k, UpdatePhase, TimePosGraph!Position, "min"
    If ((Time > 0 And TimePosGraph!Position > MaxMinPos) Or _
        (Time < 0 And TimePosGraph!Position < MaxMinPos)) And _
        TimePosGraph!Min < MaxMin Then
        ' we are in the falling branch after the peak --> delete these points
        TimePos.delete
    ElseIf TimePosGraph!Min > LastMin Then
        ' make curve monotonously falling
        TimePos.Edit
        TimePos!Time = LastMin
        TimePos.Update
    Else
        LastMin = TimePosGraph!Min
    End If
    TimePosGraph.Move (-Time)
    If TimePosGraph.BOF Or TimePosGraph.EOF Then Exit Do
Loop
End If
Next k
TimePosGraph.Close
Set Stat = DB.OpenRecordset("rqTimePosStat", dbOpenSnapshot)
Stat.FindFirst "UpdatePhase = " & CStr(UpdatePhase)
For k = 0 To TunnelNo - 1
    TimePos.Seek ">=", k, UpdatePhase
    If TimePos.NoMatch Then
        ' do nothing because there are no data points for this tunnel
    ElseIf TimePos.EOF Then
        ' do nothing because there are no data points for this tunnel
    ElseIf TimePos!Tunnel = k And TimePos!UpdatePhase = UpdatePhase Then
        TimePos.addnew
        TimePos!Tunnel = k
        TimePos!UpdatePhase = UpdatePhase
    End If
Next k
```

```
        TimePos!Position = Stat(CStr(k)) + StepSize
        TimePos!Title = "min"
        TimePos!Time = Null
        TimePos.Update
    End If
Next k
Stat.Close
TimePos.Close
DB.Execute ("ImpExp1Del")
ImportFinish
Exit Sub

Repair:
    ImportFinish
    MsgBox "Can't find the file 'min_max.out'", , "IMPORT TIME-POSITION RESULTS"
End Sub

Private Function ImportPreparations(UpdatePhase As Long) As String
    Dim Title As String, UP As Recordset
    Title = "IMPORT SIMULATOR RESULTS"
    Set DB = CurrentDb()
    Set FS = CreateObject("Scripting.FileSystemObject")
    Set UP = DB.OpenRecordset("tUpdatePhases", dbOpenTable)
    UP.Index = "PrimaryKey"
    UP.Seek "=", UpdatePhase
    If IsNull(UP!Path) Then
        UP.MoveFirst
        If IsNull(UP!Path) Then
            UP.Edit
            UP!Path = " "
            UP.Update
        End If
    End If
End If
Do Until False
    ImportPreparations = ""
    ImportPreparations=InputBox("Import results from the following project directory:", _
        Title, UP!Path)
    If ImportPreparations = "" Then Exit Function
    If Not FS.FolderExists(ImportPreparations) Then
        MsgBox "The directory you have chosen does not exist.", vbOKOnly, Title
    Else
        If Not FS.FolderExists(ImportPreparations & "\tmp.dir") Then
            MsgBox "The directory you have chosen does not contain" & _
                "a 'tmp.dir' directory.", vbOKOnly, Title
        Else
            Exit Do
        End If
    End If
End If
Loop
UP.Seek "=", UpdatePhase
UP.Edit
UP!Path = ImportPreparations
UP.Update
UP.Close
ImportPreparations = ImportPreparations & "\tmp.dir"
DoCmd.OpenForm "BePatient", acNormal
DoCmd.RepaintObject acForm, "BePatient"
DoCmd.Hourglass True
DoCmd.Echo False, "Working hard...."
DoCmd.SetWarnings False
End Function

Private Sub ImportFinish()
    DB.Close
    DoCmd.SetWarnings True
    DoCmd.Echo True
    DoCmd.Hourglass False
    DoCmd.Close acForm, "BePatient", acSaveNo
```

End Sub

Module: Transfer

This module transfers a whole project (including all update phases) from an other database to the currently open database.

```
Option Compare Database
Option Explicit
    Dim SourceDB As Database, TargetDB As Database

Public Function Transfer() As Boolean
    Set SourceDB = TransferPreparations
    On Error GoTo Repair:
    If SourceDB.Name <> " " Then
        On Error GoTo 0
        Set TargetDB = CurrentDb()
        DoCmd.RunMacro "Delete All Data"
        TransferData
        SourceDB.Close
        TargetDB.Close
    End If
    TransferFinish
    Exit Function
Repair:
    TransferFinish
    Exit Function
End Function

Private Function TransferPreparations() As Database
    Dim FS As Object, Title As String, Path As String
    Title = "TRANSFER DATA FROM OTHER DATABASE"
    If MsgBox("This action will delete ALL data of this project!" & Chr(13) & _
        "Do you want to continue?", vbYesNo + vbQuestion, Title) _
        = vbNo Then Exit Function
    Set FS = CreateObject("Scripting.FileSystemObject")
    Do Until False
        Path = ""
        Path = InputBox("Transfer project from the following database:", _
            Title, Path)
        If Path = "" Then Exit Function
        If FS.FileExists(Path) Then
            Set TransferPreparations = OpenDatabase(Path, True, True)
            Exit Do
        Else
            MsgBox "The file you have chosen does not exist.", vbOKOnly, Title
        End If
    Loop
    DoCmd.OpenForm "BePatient", acNormal
    DoCmd.RepaintObject acForm, "BePatient"
    DoCmd.Hourglass True
    DoCmd.Echo False, "Working hard...."
    DoCmd.SetWarnings False
End Function

Private Sub TransferFinish()
    DoCmd.SetWarnings True
    DoCmd.Echo True
    DoCmd.Hourglass False
    DoCmd.Close acForm, "BePatient", acSaveNo
End Sub

Private Sub CopyTable(Name As String)
    Static SourceRS As Recordset, TargetRS As Recordset
```

```
Static FieldNo As Integer, MaxFieldNo As Integer
Set SourceRS = SourceDB.OpenRecordset(Name, dbOpenTable, dbReadOnly)
Set TargetRS = TargetDB.OpenRecordset(Name, dbOpenTable, dbAppendOnly)
If SourceRS.EOF And SourceRS.BOF Then Exit Sub
On Error Resume Next
SourceRS.Index = "PrimaryKey"
SourceRS.MoveFirst
MaxFieldNo = TargetRS.Fields.Count
Do Until SourceRS.EOF
    TargetRS.addnew
    For FieldNo = 0 To MaxFieldNo - 1
        ' it is safer to copy the fields by name than by number because
        ' someone might add a field or change their order. If a field doesn't
        ' exist in the source table it is simply ignored. One must make sure that
        ' the default value of such a new field is always a valid value, otherwise
        ' the recordset may not be added.
        TargetRS(FieldNo).Value = SourceRS(TargetRS(FieldNo).Name).Value
    Next FieldNo
    TargetRS.Update
    SourceRS.MoveNext
Loop
SourceRS.Close
TargetRS.Close
End Sub

Private Sub TransferData()
    ' it is important to transfer the tables in this order because otherwise
    ' links between tables inhibit adding recordsets if the table on the 1-side of
    ' a link has not yet been transferred.
    CopyUpdatePhases
    CopyTable "tParamSets"
    CopyTable "tParameters"
    CopyTable "tParam_X_Set"
    CopyTable "tParamStates"
    CopyTable "tDeterSegs"
    CopyTable "tSemiDeterSegs"
    CopyTable "tSegProb"
    CopyTable "tMarkovLengths"
    CopyTable "otTransitions"
    CopyTable "otMarkovGraph"
    CopyTable "tMarkovTransProb"
    CopyTable "tBoreholes"
    CopyTable "tBH_X_States"
    CopyTable "tGC"
    CopyTable "tGCCCombinations"
    CopyTable "tGCCComb_X_States"
    CopyTable "tCombGC"
    CopyTable "tCorrelations"
    CopyTable "tCorrel_X_States"
    CopyTable "tAreas"
    CopyTable "otAreaParamStat"
    CopyTable "tZones"
    CopyTable "tZoneProb"
    CopyTable "otZoneParamStat"
    CopyTable "tGeometries"
    CopyTable "otGCCGeom"
    CopyTunnels
    CopyTable "otTunDelays"
    CopyTable "tUpDownChanges"
    CopyTable "tTunDelays"
    CopyTable "tTunDelayProb"
    CopyTable "otTunDelaysGraph"
    CopyTable "tTunnelNodes"
    CopyTable "tTunnelArcs"
    CopyTable "tRepositories"
    CopyTable "tTunnel_X_Rep"
    CopyTable "tMethods"
    CopyTable "tCGC_X_Meth_X_Geom"
```

```
CopyTable "tChangeDelays"
CopyTable "otChangeDelays"
CopyTable "otChangeDelaysGraph"
CopyTable "tLearnParam"
CopyTable "otMethods"
CopyTable "tResources"
CopyTable "tStocks"
CopyTable "tActivities"
CopyTable "tActResDemand"
CopyTable "tHeads"
CopyTable "tHeadProb"
CopyTable "otCycleLengths"
CopyTable "otCycleGraph"
CopyTable "tMethNodes"
CopyTable "tMethArcs"
CopyTable "tVariables"
CopyTable "tVariableProb"
CopyTable "otVariables"
CopyTable "otVariablesGraph"
CopyTable "tGenVariables"
CopyTable "tGenVarProb"
CopyTable "otGenVariables"
CopyTable "otGenVariablesGraph"
CopyTable "rtTimeCost"
CopyTable "rtTimeCostGraph"
CopyTable "rtTimePosGraph"
CopyTime
End Sub

Private Sub CopyUpdatePhases()
    Static SourceRS As Recordset, TargetRS As Recordset
    Static FieldNo As Integer, MaxFieldNo As Integer
    Set SourceRS = SourceDB.OpenRecordset("tUpdatePhases", dbOpenTable, dbReadOnly)
    Set TargetRS = TargetDB.OpenRecordset("tUpdatePhases", dbOpenTable, dbAppendOnly)
    If SourceRS.EOF And SourceRS.BOF Then Exit Sub
    On Error Resume Next
    SourceRS.Index = "PrimaryKey"
    SourceRS.MoveFirst
    SourceRS.Move (3) ' records with ID=1, 10 and 11 exist already
    MaxFieldNo = TargetRS.Fields.Count
    Do Until SourceRS.EOF
        TargetRS.addnew
        For FieldNo = 0 To MaxFieldNo - 1
            TargetRS(FieldNo).Value = SourceRS(TargetRS(FieldNo).Name).Value
        Next FieldNo
        TargetRS.Update
        SourceRS.MoveNext
    Loop
    SourceRS.Close
    TargetRS.Close
End Sub

Private Sub CopyTunnels()
    Static SourceRS As Recordset, TargetRS As Recordset
    Static FieldNo As Integer, MaxFieldNo As Integer
    Set SourceRS = SourceDB.OpenRecordset("tTunnels", dbOpenTable, dbReadOnly)
    Set TargetRS = TargetDB.OpenRecordset("tTunnels", dbOpenTable, dbAppendOnly)
    If SourceRS.EOF And SourceRS.BOF Then Exit Sub
    On Error Resume Next
    SourceRS.Index = "PrimaryKey"
    SourceRS.MoveFirst
    MaxFieldNo = TargetRS.Fields.Count
    ' must first create all tunnels because of field "opposite tunnel"
    Do Until SourceRS.EOF
        TargetRS.addnew
        For FieldNo = 0 To MaxFieldNo - 1
            ' must add opp tunnel later when all tunnels are created
        Next FieldNo
    Loop
End Sub
```

```
        If Not TargetRS(FieldNo).Name = "OppTunnel" Then _
            TargetRS(FieldNo).Value = SourceRS(TargetRS(FieldNo).Name).Value
    Next FieldNo
    TargetRS!ID = SourceRS!ID
    TargetRS.Update
    SourceRS.MoveNext
Loop
TargetRS.MoveFirst
Do Until TargetRS.EOF
    SourceRS.Seek "=", TargetRS!ID
    TargetRS.Edit
    TargetRS!OppTunnel = SourceRS!OppTunnel
    TargetRS.Update
    TargetRS.MoveNext
Loop
SourceRS.Close
TargetRS.Close
End Sub

Private Sub CopyTime()
    Dim SourceRS As Recordset, TargetRS As Recordset
    Set SourceRS = SourceDB.OpenRecordset("tTime", dbOpenTable, dbReadOnly)
    Set TargetRS = TargetDB.OpenRecordset("tTime", dbOpenTable)
    SourceRS.MoveFirst
    TargetRS.MoveFirst
    TargetRS.Edit
    TargetRS!Current = SourceRS!Current
    TargetRS.Update
    TargetRS.Close
    SourceRS.Close
End Sub
```

Form_fAssocAdd

In this form one can add new geometry/gc/method association during the updating process.

```
Private Sub AddAssoc_Click()
    Static Assoc As Recordset, UP As Recordset, FM As Form
    If IsNull(Me!Geometry) Or IsNull(Me!Method) Or IsNull(Me!GC) Then Exit Sub
    Set UP = CurrentDb().OpenRecordset("tUpdatePhases", dbOpenTable, dbReadOnly)
    Set Assoc = CurrentDb().OpenRecordset("tCGC_X_Meth_X_Geom", dbOpenTable)
    Assoc.Index = "PrimaryKey"
    Assoc.Seek "=", Me!Geometry, Me!GC, Me!Method, 1
    If Not Assoc.NoMatch Then
        MsgBox "This association exists already!", vbOKOnly, "ERROR ADDING ASSOCIATION"
        UP.Close
        Assoc.Close
        Exit Sub
    End If
    UP.MoveFirst
    Do Until UP.EOF
        If UP!ID <> 11 Then ' add for all update phases except 11
            Assoc.addnew
            Assoc!Geometry = Me!Geometry
            Assoc!CGC = Me!GC
            Assoc!Method = Me!Method
            Assoc!UpdatePhase = UP!ID
            Assoc!Prob = 0
            Assoc.Update
        End If
        UP.MoveNext
    Loop
    UP.Close
```



```
Assoc.Close
Set FM = Forms!fAssocMain 'adjust height of form
FM.Detail.Height = FM.Detail.Height + 341
FM.fAssocMethods.Height = FM.fAssocMethods.Height + 341
FM.fAssocPost.Height = FM.fAssocPost.Height + 341
FM!fAssocPost.Requery
FM!fAssocMethods.Requery
End Sub
```

Form_fAssocPost

This (sub)form contains the “remove” button to remove an association.

```
Private Sub remove_Click()
    Dim RS As Recordset, Assoc As Recordset, FM As Form
    If MsgBox("Do you really want to delete this association?" & Chr(13) & Chr(13) & _
        "It will be removed from all update phases, including prior input.", _
        vbYesNo, "DELETE ASSOCIATION") = vbYes Then
        Set RS = Me.Recordset.Clone
        RS.Bookmark = Me.Bookmark
        Set Assoc = CurrentDb().OpenRecordset("tCGC_X_Meth_X_Geom", dbOpenTable)
        Assoc.Index = "PrimaryKey"
        Assoc.Seek ">=", RS!Geometry, RS!CGC, RS!Method
        Do Until Assoc.NoMatch
            On Error GoTo Repair
            If Assoc!Geometry <> RS!Geometry Or Assoc!CGC <> RS!CGC Or _
                Assoc!Method <> RS!Method Then Exit Do
            Assoc.delete
            Assoc.Seek ">=", RS!Geometry, RS!CGC, RS!Method
        Loop
        Set FM = Forms!fAssocMain 'adjust height of form
        FM.Detail.Height = FM.Detail.Height - 341
        FM.fAssocMethods.Height = FM.fAssocMethods.Height - 341
        FM.fAssocPost.Height = FM.fAssocPost.Height - 341
        Me.Requery
        FM!fAssocMethods.Requery
    End If
    Exit Sub
Repair:
    Set FM = Forms!fAssocMain
    FM.Detail.Height = FM.Detail.Height - 341
    FM.fAssocMethods.Height = FM.fAssocMethods.Height - 341
    FM.fAssocPost.Height = FM.fAssocPost.Height - 341
    Me.Requery
    FM!fAssocMethods.Requery
    Exit Sub
End Sub
```

Form_fChangeDelaysAdd

This form is used to add a method change delay during the updating process.

```
Private Sub AddChangeDelay_Click()
    Static ChangeDelay As Recordset, UP As Recordset, FM As SubForm
    If IsNull(Me!FromMethod) Or IsNull(Me!ToMethod) Then Exit Sub
    If Me!FromMethod = Me!ToMethod Then
        MsgBox "From and To method must not be identical", vbOKOnly, _
            "ERROR ADDING METHOD CHANGE DELAY"
        Exit Sub
    End If
    Set UP = CurrentDb().OpenRecordset("tUpdatePhases", dbOpenTable, dbReadOnly)
    Set ChangeDelay = CurrentDb().OpenRecordset("tChangeDelays", dbOpenTable)
    ChangeDelay.Index = "PrimaryKey"
```

```
ChangeDelay.Seek "=", Me!FromMethod, Me!ToMethod, 1
If Not ChangeDelay.NoMatch Then
    MsgBox "This method change delay exists already!", vbOKOnly, _
        "ERROR ADDING CHANGE DELAY"
    UP.Close
    ChangeDelay.Close
    Exit Sub
End If
UP.MoveFirst
Do Until UP.EOF
    If UP!ID <> 11 And UP!ID <> 10 Then
        ChangeDelay.addnew
        ChangeDelay!FromMethod = Me!FromMethod
        ChangeDelay!ToMethod = Me!ToMethod
        ChangeDelay!UpdatePhase = UP!ID
        ChangeDelay.Update
    End If
    UP.MoveNext
Loop
UP.Close
ChangeDelay.Close
Set FM = Forms!fChangeDelaysMain!fChangeDelaysSub!fChangeDelaysPrior
FM.Height = FM.Height + 341
FM.Requery
Set FM = Forms!fChangeDelaysMain!fChangeDelaysSub!fChangeDelaysPost
FM.Height = FM.Height + 341
FM.Requery
End Sub
```

Form_fChangeDelaysPost

This (sub)form contains the button to remove a method change delay while updating.

```
Private Sub remove_Click()
    Dim RS As Recordset, ChangeDelay As Recordset, FM As Form
    Dim FromMethod As Long, ToMethod As Long, Name As String
    If MsgBox("Do you really want to delete this method change delay?" & Chr(13) & _
        Chr(13) & "It will be removed from all update phases, prior input" & Chr(13) & _
        "and observed data.", vbYesNo, "DELETE METHOD CHANGE DELAY") = vbYes Then
        Set RS = Me.Recordset.Clone
        RS.Bookmark = Me.Bookmark
        FromMethod = RS!FromMethod
        ToMethod = RS!ToMethod
        Name = "tChangeDelays" ' change name in each loop
        Do Until Name = "finish"
            Set ChangeDelay = CurrentDb().OpenRecordset(Name, dbOpenTable)
            ChangeDelay.Index = "PrimaryKey"
            ChangeDelay.Seek ">=", FromMethod, ToMethod
            Do Until ChangeDelay.NoMatch
                If ChangeDelay!FromMethod <> FromMethod Or _
                    ChangeDelay!ToMethod <> ToMethod Then Exit Do
                ChangeDelay.delete
                ChangeDelay.Seek ">=", FromMethod, ToMethod
            Loop
            ChangeDelay.Close
            If Name = "tChangeDelays" Then
                Name = "otChangeDelaysGraph" ' do the same again for graph table
            ElseIf Name = "otChangeDelaysGraph" Then
                Name = "otChangeDelays" ' do the same again for observed values
            Else
                Name = "finish"
            End If
        Loop
        Me.Requery
        Set FM = Me.Parent
    End Sub
```

```
        FM!fChangeDelaysPrior.Requery
    End If
End Sub
```

Form_fCycleMain

This form is used to update cycle lengths.

```
Private Sub AddHead_Click()
    Dim RS As Recordset, UP As Recordset, HeadProb As Recordset
    Dim MaxOrder As Double, Head As Long
    Set RS = Me.Recordset.Clone
    Set UP = CurrentDb().OpenRecordset("tUpdatePhases", dbOpenTable, dbReadOnly)
    Set HeadProb = CurrentDb().OpenRecordset("tHeadProb", dbOpenTable)
    MaxOrder = 0
    RS.FindFirst ("Method = " & CStr(Me!Method))
    Do Until RS.NoMatch
        ' look for highest Order value
        If RS!Order > MaxOrder Then MaxOrder = RS!Order
        RS.FindNext ("Method = " & CStr(Me!Method))
    Loop
    RS.addnew
    RS!Method = Me!Method
    RS!Order = MaxOrder + 1
    Head = RS!ID
    RS.Update
    UP.MoveFirst
    Do Until UP.EOF
        If UP!ID <> 10 And UP!ID <> 11 Then
            HeadProb.addnew
            HeadProb!Head = Head
            ' cycle lengths are zero by default
            HeadProb!UpdatePhase = UP!ID
            HeadProb.Update
        End If
        UP.MoveNext
    Loop
    UP.Close
    HeadProb.Close
    Me.Requery
    RS.Requery
    RS.FindFirst "ID = " & CStr(Head)
    Me.Bookmark = RS.Bookmark
    Me.Head.Requery
    Me.Head = RS!ID
    Me.fCyclePost.Requery
    Me.fCyclePrior.Requery
    RS.Close
End Sub

Private Sub remove_Click()
    Dim Heads As Recordset, HeadProb As Recordset, Nodes As Recordset, Arcs As Recordset
    Dim Name As String, Head As Long
    If MsgBox("Do you really want to delete this heading?" & Chr(13) & Chr(13) & _
        "It will be removed from all update phases, prior input" & Chr(13) & _
        "(including activity network) and observed data.", _
        vbYesNo, "DELETE HEADING") = vbYes Then
        Head = Me!Head
        Name = "tHeadProb" ' change name in each loop
        Do Until Name = "finish"
            Set HeadProb = CurrentDb().OpenRecordset(Name, dbOpenTable)
            HeadProb.Index = "PrimaryKey"
            HeadProb.Seek ">=", Head
            Do Until HeadProb.NoMatch
                ' in otCycleGraph the field is called 'Variable' rather than 'Head'
            Loop
        Loop
    End If
End Sub
```

```
' ---> refer to it as 1st field in the recordset
If HeadProb.Fields(0).Value <> Head Then Exit Do
HeadProb.delete
HeadProb.Seek ">=", Head
Loop
HeadProb.Close
If Name = "tHeadProb" Then
    Name = "otCycleGraph" ' do the same again for graph table
ElseIf Name = "otCycleGraph" Then
    Name = "otCycleLengths" ' do the same again for observed values
Else
    Name = "finish"
End If
Loop
' delete nodes and arcs of this head
Set Nodes = CurrentDb().OpenRecordset("tMethNodes", dbOpenTable)
Set Arcs = CurrentDb().OpenRecordset("tMethArcs", dbOpenTable)
Nodes.Index = "Head"
Nodes.Seek "=", Head ' look for nodes of this head
Do Until Nodes.NoMatch
    Arcs.Index = "StartNode"
    Arcs.Seek "=", Nodes!ID
    ' look for arcs with this node (cannot look directly for arcs because they do not
    ' directly belong to a head, only by their relation to nodes), must delete arcs first
    Do Until Arcs.NoMatch
        Arcs.delete
        Arcs.Seek "=", Nodes!ID
    Loop
    Arcs.Index = "EndNode"
    Arcs.Seek "=", Nodes!ID
    Do Until Arcs.NoMatch
        Arcs.delete
        Arcs.Seek "=", Nodes!ID
    Loop
    Nodes.delete
    Nodes.Seek "=", Head
Loop
Nodes.Close
Arcs.Close
Set Heads = CurrentDb().OpenRecordset("tHeads", dbOpenTable)
Heads.Index = "PrimaryKey"
Heads.Seek "=", Head
Heads.delete
Heads.Close
MethodID_AfterUpdate
End If
End Sub

Private Sub Timelimit_AfterUpdate()
    Static Graph As Recordset, Source As Recordset, CT As Control
    ' recalculate and redraw graph because data with time < timelimit are discarded
    RunCommand acCmdSaveRecord
    Set Graph = CurrentDb().OpenRecordset("otCycleGraph", dbOpenTable)
    Set Source = CurrentDb().OpenRecordset("otCycleLengths", dbOpenTable, dbReadOnly)
    Graphs.GraphVariableObserved Source, Graph, Me!Head, Me!Timelimit
    Set CT = Forms!fCycleMain!GraphTime
    CT.Requery
    Set CT = Forms!fCycleMain!Graph
    CT.Requery
    Set CT = Forms!fCycleMain!ObsMean
    CT.Requery
End Sub
```

Form_fGenVariablesMain

This form is used to update general variables.

```
Private Sub AddGenVar_Click()
    Dim RS As Recordset, UP As Recordset, GenVarProb As Recordset
    Dim Name As String, Variable As Long
    Name = ""
    Name = InputBox("Which name would you like to give" & Chr(13) & _
        "to this new variable?", "ADD GENERAL VARIABLE")
    If Name = "" Then Exit Sub
    Set RS = Me.Recordset.Clone
    Set UP = CurrentDb().OpenRecordset("tUpdatePhases", dbOpenTable, dbReadOnly)
    Set GenVarProb = CurrentDb().OpenRecordset("tGenVarProb", dbOpenTable)
    RS.addnew
    RS!Name = Name
    Name = ""
    Name = InputBox("Which description would you like to give" & Chr(13) & _
        "to this new variable?", "ADD GENERAL VARIABLE")
    If Name <> "" Then RS!Description = Name
    Variable = RS!ID
    RS.Update
    UP.MoveFirst
    Do Until UP.EOF
        If UP!ID <> 10 And UP!ID <> 11 Then
            'add probabilities for all update phases except 10 and 11
            GenVarProb.addnew
            GenVarProb!Variable = Variable
            GenVarProb!UpdatePhase = UP!ID
            GenVarProb.Update
        End If
        UP.MoveNext
    Loop
    UP.Close
    GenVarProb.Close
    Me.Requery
    RS.Requery
    RS.FindFirst "ID = " & CStr(Variable)
    Me.Bookmark = RS.Bookmark
    Me.Variable.Requery
    Me.Variable = RS!ID
    Me.fGenVariablesPost.Requery
    Me.fGenVariablesPrior.Requery
    RS.Close
End Sub

Private Sub remove_Click()
    Dim GenVariables As Recordset, GenVarProb As Recordset
    Dim Name As String, Variable As Long
    If MsgBox("Do you really want to delete this variable?" & Chr(13) & Chr(13) & _
        "It will be removed from all update phases, prior input" & Chr(13) & _
        "and observed data.", vbYesNo, "DELETE VARIABLE") = vbYes Then
        Variable = Me!Variable
        Name = "tGenVarProb" ' change name in each loop
        Do Until Name = "finish"
            Set GenVarProb = CurrentDb().OpenRecordset(Name, dbOpenTable)
            GenVarProb.Index = "PrimaryKey"
            GenVarProb.Seek ">=", Variable
            Do Until GenVarProb.NoMatch
                If GenVarProb.Fields(0).Value <> Variable Then Exit Do
                GenVarProb.delete
                GenVarProb.Seek ">=", Variable
            Loop
            GenVarProb.Close
            If Name = "tGenVarProb" Then
                Name = "otGenVariablesGraph" ' do the same again for graph table
            ElseIf Name = "otGenVariablesGraph" Then
                Name = "otGenVariables" ' do the same again for observed values
            Else
                Name = "finish"
            End If
        Loop
    End If
End Sub
```

```
Set GenVariables = Me.Recordset.Clone
GenVariables.FindFirst "ID = " & CStr(Variable)
GenVariables.delete
Me.Requery
If Not (GenVariables.EOF And GenVariables.EOF) Then
    GenVariables.MoveFirst
    Me!Variable = GenVariables!ID ' go to first record in recordset
    Me.Bookmark = GenVariables.Bookmark
End If
End If
End Sub

Private Sub Timelimit_AfterUpdate()
    Static Graph As Recordset, Source As Recordset, CT As Control
    RunCommand acCmdSaveRecord
    ' recalculate and redraw graph because data with time < timelimit are discarded
    Set Graph = CurrentDb().OpenRecordset("otGenVariablesGraph", dbOpenTable)
    Set Source = CurrentDb().OpenRecordset("otGenVariables", dbOpenTable, dbReadOnly)
    Graphs.GraphVariableObserved Source, Graph, Me!Variable, Me!Timelimit
    Set CT = Forms!fGenVariablesMain!GraphTime
    CT.Requery
    Set CT = Forms!fGenVariablesMain!Graph
    CT.Requery
    Set CT = Forms!fGenVariablesMain!ObsMean
    CT.Requery
End Sub
```

Form_fMarkovMain

This form is used to update Markov lengths.

```
Private Sub Calculate_Click()
    Static ML As Recordset, Min As Double, Mode As Double, Max As Double
    Static CT As Control, Graph As Recordset
    Set ML = CurrentDb().OpenRecordset("tMarkovLengths", dbOpenTable)
    ML.Index = "PrimaryKey"
    ML.Seek "=", Me!ParamSet, Me!ParameterState, 10 ' calculated values are stored in UP=10
    If ML.NoMatch Then ' this should never happen
        Min = 0
        Mode = 0
        Max = 0
    Else
        Min = ML!LengthMin
        Mode = ML!LengthMode
        Max = ML!LengthMax
    End If
    ML.Seek "=", Me!ParamSet, Me!ParameterState, _
        DLookup("ID", "tUpdatePhases", "Current = True") ' look for current UP
    If Not ML.NoMatch Then
        ML.Edit
        ML!LengthMin = Min
        ML!LengthMode = Mode
        ML!LengthMax = Max
        ML.Update
        Me.fMarkovProbPost.Requery
        Set Graph = CurrentDb().OpenRecordset("otMarkovGraph", dbOpenTable)
        ' recalculate graph
        Graphs.GraphMarkovPriorPost ML, Graph, "Posterior"
        Set CT = Forms!fMarkovMain!Graph
        CT.Requery
        ML.Close
    End If
End Sub
```

Form_fNonMarkovSemiDeterPrior

This (sub)form has the buttons to add and remove semi-deterministic segment during the updating process.

```
Private Sub insert_Click()  
    Static SegProb As Recordset, SDS As Recordset, UpdatePhases As Recordset  
    Static RS As Recordset  
    Static Segment As Long, Order As Double, ParamState As Long, ParamSet As Long  
    Set SegProb = CurrentDb().OpenRecordset("tSegProb", dbOpenTable, dbAppendOnly)  
    Set SDS = CurrentDb().OpenRecordset("tSemiDeterSegs", dbOpenTable, dbAppendOnly)  
    Set UpdatePhases = CurrentDb().OpenRecordset("tUpdatePhases", dbOpenTable, dbReadOnly)  
    Set RS = Me.RecordsetClone  
    Order = Me!Order  
    RS.Bookmark = Me.Bookmark  
    ParamState = RS!ParamState  
    ParamSet = RS!ParamSet  
    RS.MovePrevious  
    If RS.BOF Then  
        Order = Order - 1  
    Else  
        ' order is between the orders of the current and the previous segments  
        Order = (Order + RS!Order) / 2  
    End If  
    SDS.addnew  
    SDS!ParamSet = ParamSet  
    SDS!ParamState = ParamState  
    SDS!Order = Order  
    Segment = SDS!ID  
    SDS.Update  
    Do Until UpdatePhases.EOF  
        If UpdatePhases!ID <> 10 And UpdatePhases!ID <> 11 Then  
            ' add probabilities in all update phases except for 10 and 11  
            SegProb.addnew  
            SegProb!Segment = Segment  
            SegProb!UpdatePhase = UpdatePhases!ID  
            SegProb!GenerationMode = "0"  
            If UpdatePhases!ID = 1 Then  
                SegProb!Status = "Prior"  
            Else  
                SegProb!Status = "Posterior"  
            End If  
            SegProb.Update  
        End If  
        UpdatePhases.MoveNext  
    Loop  
    SDS.Close  
    SegProb.Close  
    UpdatePhases.Close  
    RS.Requery  
    Me.Requery  
    Forms!fNonMarkovSemiDeterMain!fNonMarkovSemiDeterPost.Requery  
    RS.FindFirst "Order = " & CStr(Order)  
    Me.Bookmark = RS.Bookmark  
End Sub
```

```
Private Sub remove_Click()  
    Static SegProb As Recordset, SDS As Recordset, Segment As Long, Order As Double  
    If MsgBox("Do you really want to delete this segment?" & Chr(13) & Chr(13) & _  
        "It will be removed from all update phases, including prior input.", _  
        vbYesNo, "DELETE SEGMENT") = vbYes Then  
        Segment = Me!Segment  
        Order = Me!Order  
        Set SegProb = CurrentDb().OpenRecordset("tSegProb", dbOpenTable)
```

```
Set SDS = CurrentDb().OpenRecordset("tSemiDeterSegs", dbOpenTable)
SegProb.Index = "PrimaryKey"
SDS.Index = "PrimaryKey"
SegProb.Seek ">=", Segment
Do Until SegProb.NoMatch
    If SegProb!Segment = Segment Then
        SegProb.delete
        SegProb.Seek ">=", Segment
    Else
        Exit Do
    End If
Loop
SegProb.Close
SDS.Seek "=", Segment
SDS.delete
SDS.Close

Me.Requery
Set SDS = Me.Recordset.Clone
SDS.FindFirst "Order >= " & CStr(Order)
' go to segment after the deleted one
If Not SDS.NoMatch Then Me.Bookmark = SDS.Bookmark
SDS.Close
Forms!fNonMarkovSemiDeterMain!fNonMarkovSemiDeterPost.Requery
End If
End Sub
```

Form_fStockAdd

This form is used to add new stocks in repositories during the updating process.

```
Private Sub AddStock_Click()
    Static Stock As Recordset, UP As Recordset, FM As Form
    If IsNull(Me!Repository) Or IsNull(Me!Resource) Then Exit Sub
    Set UP = CurrentDb().OpenRecordset("tUpdatePhases", dbOpenTable, dbReadOnly)
    Set Stock = CurrentDb().OpenRecordset("tStocks", dbOpenTable)
    Stock.Index = "PrimaryKey"
    Stock.Seek "=", Me!Repository, Me!Resource, 1
    If Not Stock.NoMatch Then
        MsgBox "This stock exists already!", vbOKOnly, "ERROR ADDING STOCK"
        UP.Close
        Stock.Close
        Exit Sub
    End If
    UP.MoveFirst
    Do Until UP.EOF
        If UP!ID <> 11 And UP!ID <> 10 Then
            Stock.addnew
            Stock!Repository = Me!Repository
            Stock!Resource = Me!Resource
            Stock!UpdatePhase = UP!ID
            Stock.Update
        End If
        UP.MoveNext
    Loop
    UP.Close
    Stock.Close
    Set FM = Forms!fStocksMain
    FM.fStocksPrior.Height = FM.fStocksPrior.Height + 341
    FM.fStocksPost.Height = FM.fStocksPost.Height + 341
    FM!fStocksPrior.Requery
    FM!fStocksPost.Requery
End Sub
```


Form_fStocksPost

This (sub)form contains the button to remove stocks during the updating process.

```
Private Sub remove_Click()  
    Static Stocks As Recordset, Resource As Long, Repository As Long, FM As Form  
    If MsgBox("Do you really want to delete this stock?" & Chr(13) & Chr(13) & _  
        "It will be removed from all update phases, including prior input.", _  
        vbYesNo, "DELETE STOCK") = vbYes Then  
        Repository = Me!Repository  
        Resource = Me!Resource  
        Set Stocks = CurrentDb().OpenRecordset("tStocks", dbOpenTable)  
        Stocks.Index = "PrimaryKey"  
        Stocks.Seek ">=", Repository, Resource  
        Do Until Stocks.NoMatch  
            If Stocks!Repository <> Repository Or Stocks!Resource <> Resource Then Exit Do  
            Stocks.delete  
            Stocks.Seek ">=", Repository, Resource  
        Loop  
        Stocks.Close  
        Set FM = Forms!fStocksMain  
        FM.Detail.Height = FM.Detail.Height - 341  
        FM.fStocksPrior.Height = FM.fStocksPrior.Height - 341  
        FM.fStocksPost.Height = FM.fStocksPost.Height - 341  
        FM!fStocksPrior.Requery  
        Me.Requery  
    End If  
End Sub
```

Form_fTransMain

This form is used to update transition probabilities.

```
Private Sub transitionmatrix_Click()  
    Static Criterion As String  
    ' cross-table queries cannot be parametrized. therefore we have to change the  
    ' SQL statement directly to limit the table to the actual parameter and parameter set  
    Criterion = "TRANSFORM First(qTransOrderedList.Prob) AS FirstOfProb " & _  
        "SELECT IIf([qTransOrderedList].[UpdatePhase]=1,[qTransOrderedList].[FromName]" & _  
        ",Null) AS FromName, IIf([qTransOrderedList].[UpdatePhase]=1,'Prior',IIf([qTra" & _  
        "nsOrderedList].[UpdatePhase]=10,'Observed','Posterior')) AS UpdatePhase From " & _  
        "qTransOrderedList WHERE ((qTransOrderedList.ParamSet) = " & CStr(Me!ParamSet) & _  
        ") And ((qTransOrderedList.Parameter) = " & CStr(Me!Parameter) & ") GROUP BY " & _  
        "qTransOrderedList.FromName, qTransOrderedList.UpdatePhase, IIf([qTransOrdered" & _  
        "List].[UpdatePhase]=1,[qTransOrderedList].[FromName],Null), IIf([qTransOrdere" & _  
        "dList].[UpdatePhase]=1,'Prior',IIf([qTransOrderedList].[UpdatePhase]=10,'Obse" & _  
        "rved','Posterior')) ORDER BY qTransOrderedList.FromName, qTransOrderedList.Up" & _  
        "datePhase PIVOT qTransOrderedList.ToName;"  
    CurrentDb().QueryDefs("qTransMatrix").Properties("SQL") = Criterion  
    DoCmd.OpenQuery "qTransMatrix", , acReadOnly  
    DoCmd.OpenForm "fTransMatrixClose"  
End Sub
```

Form_fTunDelaysMain

This form is used to update tunnel delays.

```
Private Sub AddDelay_Click()  
    Dim Delays As Recordset, DelayProb As Recordset, UP As Recordset, Graph As Recordset  
    Dim Delay As Long
```

```
Set Delays = CurrentDb().OpenRecordset("tTunDelays", dbOpenTable)
Set DelayProb = CurrentDb().OpenRecordset("tTunDelayProb", dbOpenTable)
Set UP = CurrentDb().OpenRecordset("tUpdatePhases", dbOpenSnapshot)
Set Graph = CurrentDb().OpenRecordset("otTunDelaysGraph", dbOpenTable)
Delays.addnew
Delays!Tunnel = Me!Tunnel
Delay = Delays!ID
Delays.Update
DelayProb.Index = "PrimaryKey"
UP.MoveFirst
Do Until UP.EOF
    ' add it in all update phases except 10 and 11
    If UP!ID <> 10 And UP!ID <> 11 Then
        DelayProb.addnew
        DelayProb!Delay = Delay
        DelayProb!Type = 0 'make it time-dependent
        DelayProb!UpdatePhase = UP!ID
        If UP!ID = 1 Then
            DelayProb!Status = "Prior"
        Else
            DelayProb!Status = "Posterior"
        End If
        DelayProb.Update
        DelayProb.Seek "=", Delay, UP!ID
    End If
    UP.MoveNext
Loop
Me.fTunDelaysPost.Requery
Me.fTunDelaysPrior.Requery
Me.GraphTime.Requery
UP.Close
Delays.Close
DelayProb.Close
' recalculate height of subforms
Me.fTunDelaysPrior.Height = Me.fTunDelaysPrior.Height + 341
Me.fTunDelaysPost.Height = Me.fTunDelaysPost.Height + 341
End Sub
```

Form_fTunDelaysPost

This (sub)form contains the button to remove a tunnel delay during updating.

```
Private Sub remove_Click()
    Dim DelayProb As Recordset, Delays As Recordset, Delay As Long
    Dim CT As Control, Name As String
    If MsgBox("Do you really want to delete this delay?" & Chr(13) & Chr(13) & _
        "It will be removed from all update phases, including prior input.", _
        vbYesNo, "DELETE DELAY") = vbYes Then
        Delay = Me!Delay
        Set Delays = CurrentDb().OpenRecordset("tTunDelays", dbOpenTable)
        Delays.Index = "PrimaryKey"
        Name = "tTunDelayProb" ' after first loop change name to delete in graph table
        Do Until Name = "finish"
            Set DelayProb = CurrentDb().OpenRecordset(Name, dbOpenTable)
            DelayProb.Index = "PrimaryKey"
            DelayProb.Seek ">=", Delay
            Do Until DelayProb.NoMatch
                If DelayProb!Delay <> Delay Then Exit Do
                DelayProb.delete
                DelayProb.Seek ">=", Delay
            Loop
            DelayProb.Close
            If Name = "tTunDelayProb" Then
                Name = "otTunDelaysGraph" ' do the same again for graph table
            Else

```

```
        Name = "finish"
    End If
    Loop
    Delays.Seek "=", Delay
    Delays.delete
    Delays.Close
    Me.Requery
    Set CT = Forms!fTunDelaysMain!GraphTime
    CT.Requery
    Set CT = Forms!fTunDelaysMain!GraphPosition
    CT.Requery
    Set CT = Forms!fTunDelaysMain!fTunDelaysPrior
    CT.Requery
End If
End Sub
```

Form_fUpdatePhases

This form is used to manage update phases.

```
Private Sub makeactive_Click()
    Static DB As Database
    If Me!Current Then Exit Sub
    Set RS = Me.Recordset.Clone
    Set DB = CurrentDb()
    RS.FindFirst "Current = True"
    If Not RS.NoMatch Then
        RS.Edit
        RS!Current = False
        RS.Update
    End If
    RS.Bookmark = Me.Bookmark
    RS.Edit
    RS!Current = True
    RS.Update
    RS.Requery
    Me.Requery
    RS.FindFirst "Current = True"
    ' (re)calculate posterior graphs
    Graphs.GraphsPriorPostCreate DB, "tMarkovLengths", "otMarkovGraph", RS!ID
    Graphs.GraphsPriorPostCreate DB, "tVariableProb", "otVariablesGraph", RS!ID
    Graphs.GraphsPriorPostCreate DB, "tGenVarProb", "otGenVariablesGraph", RS!ID
    Graphs.GraphsPriorPostCreate DB, "qHeadProb", "otCycleGraph", RS!ID
    Graphs.GraphsTunDelaySet DB, False
    Me.Bookmark = RS.Bookmark
End Sub

Private Sub remove_Click()
    Static QDef As QueryDef, RS As Recordset
    Set RS = Me.Recordset.Clone
    If Me!Current And RS.RecordCount > 1 Then
        MsgBox "You can't delete an active update phases unless it is the only one.", _
            , "DELETE UPDATE PHASE"
        Exit Sub
    End If
    If MsgBox("Are you sure that you want to delete all" & Chr(13) & _
        "data related to this update phase?", vbYesNo, _
        "DELETE UPDATE PHASE") = vbYes Then
        DoCmd.SetWarnings False
        DoCmd.Echo False
        DoCmd.Hourglass True
        ManageUpdatePhases.RemoveUpdatePhase Me!ID, Me!Current, CurrentDb()
        DoCmd.SetWarnings True
        DoCmd.Echo True
        DoCmd.Hourglass False
    End If
End Sub
```

```
        RunCommand acCmdDeleteRecord
        Me.Requery
    End If
    If RS.RecordCount < 1 Then mainmenu_Click
End Sub

Private Sub removeall_Click()
    If MsgBox("Are you sure that you want to delete all" & Chr(13) & _
        "data related to ANY update phase?", vbYesNo, _
        "DELETE ALL UPDATE PHASES") = vbYes Then
        DoCmd.SetWarnings False
        DoCmd.Echo False
        DoCmd.Hourglass True
        ManageUpdatePhases.RemoveAllUpdatePhases CurrentDb()
        DoCmd.SetWarnings True
        DoCmd.Echo True
        DoCmd.Hourglass False
        mainmenu_Click
    End If
End Sub

Private Sub removeobs_Click()
    If MsgBox("Are you sure that you want to delete all" & Chr(13) & _
        "observations ?", vbYesNo, _
        "DELETE ALL OBSERVATIONS") = vbYes Then
        DoCmd.SetWarnings False
        DoCmd.Echo False
        DoCmd.Hourglass True
        ManageUpdatePhases.RemoveObservations CurrentDb()
        DoCmd.SetWarnings True
        DoCmd.Echo True
        DoCmd.Hourglass False
        mainmenu_Click
    End If
End Sub

Private Sub removeproject_Click()
    ' remove whole project incl. prior data
    If MsgBox("This action will delete ALL data of this project!" & Chr(13) & _
        "Do you want to continue?", vbYesNo + vbQuestion, "DELETE ALL DATA") _
        = vbYes Then
        DoCmd.OpenForm "BePatient", acNormal
        DoCmd.RepaintObject acForm, "BePatient"
        DoCmd.Hourglass True
        DoCmd.Echo False
        DoCmd.SetWarnings False
        DoCmd.RunMacro "Delete All Data"
        DoCmd.Hourglass False
        DoCmd.Echo True
        DoCmd.SetWarnings True
        DoCmd.Close acForm, "BePatient", acSaveNo
    End If
End Sub
```

Form_fVariablesMain

This form is used to update variables.

```
Private Sub AddVariable_Click()
    Dim RS As Recordset, UP As Recordset, VariableProb As Recordset
    Dim Name As String, Variable As Long
    Name = ""
    Name = InputBox("Which name would you like to give" & Chr(13) & _
        "to this new variable?", "ADD METHOD VARIABLE")
```

```
If Name = "" Then Exit Sub
Set RS = Me.Recordset.Clone
Set UP = CurrentDb().OpenRecordset("tUpdatePhases", dbOpenTable, dbReadOnly)
Set VariableProb = CurrentDb().OpenRecordset("tVariableProb", dbOpenTable)
RS.addnew
RS!Method = Me!Method
RS!Name = Name
Variable = RS!ID
RS.Update
UP.MoveFirst
Do Until UP.EOF
    If UP!ID <> 10 And UP!ID <> 11 Then
        VariableProb.addnew
        VariableProb!Variable = Variable
        VariableProb!UpdatePhase = UP!ID
        VariableProb.Update
    End If
    UP.MoveNext
Loop
UP.Close
VariableProb.Close
Me.Requery
RS.Requery
RS.FindFirst "ID = " & CStr(Variable)
Me.Bookmark = RS.Bookmark
Me.Variable.Requery
Me.Variable = RS!ID
Me.fVariablesPost.Requery
Me.fVariablesPrior.Requery
RS.Close
End Sub

Private Sub remove_Click()
Dim Variables As Recordset, VariableProb As Recordset
Dim Name As String, Variable As Long
If MsgBox("Do you really want to delete this variable?" & Chr(13) & Chr(13) & _
    "It will be removed from all update phases, prior input" & Chr(13) & _
    "and observed data.", vbYesNo, "DELETE VARIABLE") = vbYes Then
    Variable = Me!Variable
    Name = "tVariableProb" ' change name in each loop
    Do Until Name = "finish"
        Set VariableProb = CurrentDb().OpenRecordset(Name, dbOpenTable)
        VariableProb.Index = "PrimaryKey"
        VariableProb.Seek ">=", Variable
        Do Until VariableProb.NoMatch
            If VariableProb!Variable <> Variable Then Exit Do
            VariableProb.delete
            VariableProb.Seek ">=", Variable
        Loop
        VariableProb.Close
        If Name = "tVariableProb" Then
            Name = "otVariablesGraph" ' do the same again for graph table
        ElseIf Name = "otVariablesGraph" Then
            Name = "otVariables" ' do the same again for observed values
        Else
            Name = "finish"
        End If
    Loop
    Set Variables = CurrentDb().OpenRecordset("tVariables", dbOpenTable)
    Variables.Index = "PrimaryKey"
    Variables.Seek "=", Variable
    Variables.delete
    Variables.Close
    MethodID_AfterUpdate
End If
End Sub
```

```
Private Sub Timelimit_AfterUpdate()  
    Static Graph As Recordset, Source As Recordset, CT As Control  
    RunCommand acCmdSaveRecord  
    Set Graph = CurrentDb().OpenRecordset("otVariablesGraph", dbOpenTable)  
    Set Source = CurrentDb().OpenRecordset("otVariables", dbOpenTable, dbReadOnly)  
    Graphs.GraphVariableObserved Source, Graph, Me!Variable, Me!Timelimit  
    Set CT = Forms!fVariablesMain!GraphTime  
    CT.Requery  
    Set CT = Forms!fVariablesMain!Graph  
    CT.Requery  
    Set CT = Forms!fVariablesMain!ObsMean  
    CT.Requery  
End Sub
```

Form_fZonesPrior

This (sub)form contains the buttons to add and remove zones while updating.

```
Private Sub insert_Click()  
    Static ZoneProb As Recordset, Zones As Recordset, UpdatePhases As Recordset  
    Static RS As Recordset  
    Static Zone As Long, Order As Double, Area As Long, ParamSet As Long  
    Set ZoneProb = CurrentDb().OpenRecordset("tZoneProb", dbOpenTable, dbAppendOnly)  
    Set Zones = CurrentDb().OpenRecordset("tZones", dbOpenTable, dbAppendOnly)  
    Set UpdatePhases = CurrentDb().OpenRecordset("tUpdatePhases", dbOpenTable, dbReadOnly)  
    Set RS = Me.RecordsetClone  
    Order = Me!Order  
    RS.Bookmark = Me.Bookmark  
    Area = RS!Area  
    ParamSet = RS!ParamSet  
    RS.MovePrevious  
    If RS.BOF Then  
        Order = Order - 1  
    Else  
        Order = (Order + RS!Order) / 2  
    End If  
    Zones.addnew  
    Zones!Area = Area  
    Zones!ParamSet = ParamSet  
    Zones!Name = "<< new zone >>"  
    Zones!Order = Order  
    Zone = Zones!ID  
    Zones.Update  
    Do Until UpdatePhases.EOF  
        If UpdatePhases!ID <> 10 And UpdatePhases!ID <> 11 Then  
            ZoneProb.addnew  
            ZoneProb!Zone = Zone  
            ZoneProb!UpdatePhase = UpdatePhases!ID  
            ZoneProb!GenerationMode = "0"  
            If UpdatePhases!ID = 1 Then  
                ZoneProb!Status = "Prior"  
            Else  
                ZoneProb!Status = "Posterior"  
            End If  
            ZoneProb.Update  
        End If  
        UpdatePhases.MoveNext  
    Loop  
    Zones.Close  
    ZoneProb.Close  
    UpdatePhases.Close  
    RS.Requery  
    Me.Requery  
    Forms!fZonesMain!fZonesPost.Requery  
    RS.FindFirst "Order = " & CStr(Order)
```

```
Me.Bookmark = RS.Bookmark
Forms!fZonesMain.Area_AfterUpdate
End Sub

Private Sub remove_Click()
    Static ZoneProb As Recordset, Zones As Recordset, Zone As Long, Order As Double
    If MsgBox("Do you really want to delete this zone?" & Chr(13) & Chr(13) & _
        "It will be removed from all update phases, including prior input.", _
        vbYesNo, "DELETE ZONE") = vbYes Then
        Zone = Me!Zone
        Order = Me!Order
        Set ZoneProb = CurrentDb().OpenRecordset("tZoneProb", dbOpenTable)
        Set Zones = CurrentDb().OpenRecordset("tZones", dbOpenTable)
        ZoneProb.Index = "PrimaryKey"
        Zones.Index = "PrimaryKey"
        ZoneProb.Seek ">=", Zone
        Do Until ZoneProb.NoMatch
            If ZoneProb!Zone = Zone Then
                ZoneProb.delete
                ZoneProb.Seek ">=", Zone
            Else
                Exit Do
            End If
        Loop
        ZoneProb.Close
        Zones.Seek "=", Zone
        Zones.delete
        Zones.Close
        Me.Requery
        Set Zones = Me.Recordset.Clone
        Zones.FindFirst "Order >= " & CStr(Order)
        If Not Zones.NoMatch Then Me.Bookmark = Zones.Bookmark
        Zones.Close
        Forms!fZonesMain!fZonesPost.Requery
    End If
End Sub
```

Form_ofZonesSeq

This (sub)form is used to input observations about zones, which includes the possibility of adding and removing zones.

```
Private Sub Endpos_Enter()
    Static PS As Long
    PS = Me!ParamSet
    Set FM = Me.Parent
    ' set fields in parent to same values as in this form, because the subform with the
    ' observed parameter states changes with the fields in the parent form
    FM!Name = Me!Name
    FM!ID = Me!Zone
    FM!Order = Me!Order
    Set CT = Forms![ofZonesMain]!Parameter
    CT.RowSource = "SELECT [ID], [Name] FROM qParam_X_Set WHERE _
        [ParamSet]=[Forms]![ofZonesMain]![ofZonesSeq]!ParamSet;"
    FM!Parameter = DLookup("ID", "qParam_X_Set", "ParamSet = " & Str(PS))
    Set CT = Forms![ofZonesMain]![ofZonesParamStates]!ParamState
    CT.Requery
End Sub

Private Sub insertzone_Click()
    Dim Area As Long, Order As Double, Name As String, PS As Long
    Dim InputString As String, ParamSet As Recordset
    Area = Me!Area
    Order = Me!Order
```

```
Set RS = Me.Recordset.Clone
On Error Resume Next
RS.Bookmark = Me.Bookmark
RS.MovePrevious
On Error GoTo 0
If RS.BOF Then
    Order = Order - 1
Else
    Order = (Order + RS!Order) / 2
End If
Name = "<< new zone >>"
InputString = ""
InputString = InputBox("Parameter set no:", "NEW ZONE")
If InputString = "" Then
    RS.Close ' user cancelled operation
    Exit Sub
End If
On Error GoTo Repair:
PS = CLng(InputString)
On Error GoTo 0
Set ParamSet = CurrentDb().OpenRecordset("tParamSets", dbOpenDynaset)
ParamSet.FindFirst "ID = " & InputString
If ParamSet.NoMatch Then
    If MsgBox("This parameter set does not exist!. Do you want to create it?", _
        vbYesNo, "ERROR") = vbYes Then
        ParamSet.addnew
        ParamSet!ID = PS
        ParamSet.Update
    Else
        RS.Close
        ParamSet.Close
        Exit Sub
    End If
End If
ParamSet.Close
NewZone Area, Order, Name, PS
Me.Requery
RS.Requery
RS.FindFirst "Order = " & CStr(Order)
Me.Bookmark = RS.Bookmark
RS.Close
Exit Sub
Repair:
MsgBox "Parameter Set has to be a number!", , "NEW ZONE"
RS.Close
Exit Sub
End Sub

Public Sub NewZone(Area As Long, Order As Double, Name As String, ParamSet As Long)
    Dim DB As Database, Zones As Recordset, ZoneProb As Recordset
    Dim UP As Recordset, ZoneID As Long
    Set DB = CurrentDb()
    Set Zones = DB.OpenRecordset("tZones", dbOpenTable)
    Set ZoneProb = DB.OpenRecordset("tZoneProb", dbOpenTable)
    Set UP = DB.OpenRecordset("tUpdatePhases", dbOpenSnapshot)
    Zones.addnew
    Zones!Name = Name
    Zones!Order = Order
    Zones!Area = Area
    Zones!ParamSet = ParamSet
    ZoneID = Zones!ID
    Zones.Update
    UP.MoveFirst
    Do Until UP.EOF
        If UP!ID <> 10 And UP!ID <> 11 Then
            ZoneProb.addnew
            ZoneProb!Zone = ZoneID
            ZoneProb!UpdatePhase = UP!ID
        End If
    Loop
End Sub
```



```
        ZoneProb!GenerationMode = "0" ' by default the lengths are 0
        ZoneProb.Update
    End If
    UF.MoveNext
Loop
End Sub
```

```
Private Sub remove_Click()
    Static Zone As Long, Order As Double
    If IsNull(Me!Zone) Then Exit Sub
    Zone = Me!Zone
    Order = Me!Order
    Set RS = CurrentDb().OpenRecordset("tZoneProb", dbOpenTable)
    RS.Index = "PrimaryKey"
    RS.Seek "=", Zone, 10
    If Not RS.NoMatch Then RS.delete
    RS.Close
    Set RS = Me.Recordset.Clone
    Me.Requery
    RS.FindFirst ("Order = " & CStr(Order))
    Me.Bookmark = RS.Bookmark
    RS.Close
End Sub
```

Form_pfAreasMain

This form is used to define initial input for areas.

```
Private Sub ParamSet_NotInList(NewData As String, Response As Integer)
    Dim ParamSets As Recordset
    Response = acDataErrAdded
    If MsgBox("This parameter set does not yet exist!" & Chr(13) & _
        "Do you want to create it?", vbYesNo, "PARAMETER SET") = vbYes Then
        On Error GoTo Repair
        If CLng(NewData) > 10000000000 Then
            MsgBox "Parameter set numbers must be < 1'000'000'000", , "PARAMETER SET"
            SendKeys "{ESC}{ESC}"
        Else
            Set ParamSets = CurrentDb().OpenRecordset("tParamSets", dbOpenTable)
            ParamSets.addnew
            ParamSets!ID = CLng(NewData)
            ParamSets.Update
            ParamSets.Close
        End If
    Else
        SendKeys "{ESC}{ESC}"
    End If
    Exit Sub
Repair:
    MsgBox "Parameter set must be a number.", , "PARAMETER SET"
    SendKeys "{ESC}{ESC}"
    Exit Sub
End Sub
```

Form_pfAreasZones

This (sub)form is used to define zones initially.

```
Private Sub PSet_NotInList(NewData As String, Response As Integer)
    Dim ParamSets As Recordset
    Response = acDataErrAdded
    If MsgBox("This parameter set does not yet exist!" & Chr(13) & _
```

```
        "Do you want to create it?", vbYesNo, "PARAMETER SET") = vbYes Then
    On Error GoTo Repair
    If CLng(NewData) > 1000000000 Then
        MsgBox "Parameter set numbers must be < 1'000'000'000", , "PARAMETER SET"
        SendKeys "{ESC}{ESC}"
    Else
        Set ParamSets = CurrentDb().OpenRecordset("tParamSets", dbOpenTable)
        ParamSets.addnew
        ParamSets!ID = CLng(NewData)
        ParamSets.Update
        ParamSets.Close
    End If
Else
    SendKeys "{ESC}{ESC}"
End If
Exit Sub

Repair:
    MsgBox "Parameter set must be a number.", , "PARAMETER SET"
    SendKeys "{ESC}{ESC}"
    Exit Sub
End Sub

Private Sub remove_Click()
    Dim Zones As Recordset, Zone As Long, RS As Recordset, NextZone As Long
    Dim ZoneProb As Recordset
    On Error GoTo Repair1:
    'if this causes an error, it means that Me!Zone is NULL. Note that first the record for
    'the zone is created and only later the record with the zone probabilities, when a value
    'is entered in the endpos or length fields. Also the zone record is not yet confirmed,
    'so in order to delete it we only have to send two ESC keys (see Repair1)
    Zone = Me!Zone
    On Error GoTo 0
    Set ZoneProb = CurrentDb().OpenRecordset("tZoneProb", dbOpenTable)
    ZoneProb.Index = "PrimaryKey"
    ZoneProb.Seek "=", Zone, 10
    If Not ZoneProb.NoMatch Then
        MsgBox "This zone has already been observed and therefore" & Chr(13) & _
            "can't be deleted.", , "ERROR DELETING ZONE"
        ZoneProb.Close
        Exit Sub
    End If
    Set RS = Me.Recordset.Clone
    RS.Bookmark = Me.Bookmark
    RunCommand acCmdDeleteRecord
    RS.MoveNext
    ' find ID of next record to position cursor after requery
    If RS.EOF Then
        NextZone = 0
    Else
        NextZone = Me!Zone
    End If
    Set Zones = CurrentDb().OpenRecordset("tZones", dbOpenTable)
    Zones.Index = "PrimaryKey"
    Zones.Seek "=", Zone
    ' if the record for this zone has not yet been created, we can just send ESC keys
    If Zones.NoMatch Then
        SendKeys "{ESC}{ESC}{ESC}"
        Exit Sub
    End If
    On Error GoTo Repair2:
    Zones.delete
    On Error GoTo 0
    Zones.Close
    Me.Requery
    RS.Requery
    ' go to record after the deleted one
    If NextZone = 0 Then
        DoCmd.GoToRecord , , acNewRec
```

```
Else
    RS.FindFirst "Zone = " & CStr(NextZone)
    Me.Bookmark = RS.Bookmark
End If
Exit Sub
Repair1:
SendKeys "{ESC}{ESC}"
Exit Sub

Repair2:
MsgBox "This zone has already been observed and therefore" & Chr(13) & _
    "can't be deleted.", , "ERROR DELETING ZONE"
Zones.Close
RS.Close
Exit Sub
End Sub
```

Form_pfCorrelMain

This form is used to input correlation numbers. It contains an embedded Excel sheet.

```
Option Compare Database
Option Explicit
Dim OB As ObjectFrame, WS As Worksheet, ParamXSet As Recordset
Dim Correl As Recordset, DB As Database, FMKey As Form
Dim ParamNo As Long, MaxParamNo As Long, CurrentRow As Integer, CurrentCol As Integer
Dim FieldIndex() As Integer, Index As Integer, Criterion As String

Private Sub Edit_Click()
    Set OB = Me!ExcelSheet
    Set WS = OB.Worksheets("Correlations")
    Set DB = CurrentDb()
    ' correlations require at least 2 parameters
    Set ParamXSet = DB.OpenRecordset("SELECT * FROM qParam_X_Set WHERE " & _
        "NoParams > 1", dbOpenSnapshot)
    Set Correl = DB.OpenRecordset("pCorrelations", dbOpenDynaset)
    If ParamXSet.BOF And ParamXSet.EOF Then
        MsgBox "There aren't any parameter sets with more than one parameter!"
        Exit Sub
    End If
    DoCmd.Echo False
    ' The remaining part of the procedure opens the embedded Excel sheet and fills in the
    ' appropriate fields with the data of the database
    OB.Verb = -1 ' read help file to understand
    OB.Action = acOLEActivate
    WS.Activate
    Me.Submit.Visible = True ' make submit-button visible
    Me.printpreview.Visible = False
    Me.Edit.Visible = False
    WS.Unprotect
    ' the Excel workbook has two sheets, the template and the actual sheet which is visible
    Set WS = OB.Worksheets("Correl_Template")
    WS.Activate
    WS.Range("Everything").Copy
    Set WS = OB.Worksheets("Correlations")
    WS.Activate
    WS.Cells(1, 1).PasteSpecial xlPasteFormats ' paste all formats and values from template
    WS.Cells(1, 1).PasteSpecial xlPasteValues
    ' unhide row 1 which has the formats for the param state rows
    WS.Rows(1).Hidden = False
    CurrentRow = 2
    Do Until ParamXSet.EOF
        Criterion = "ParamSet = " & CStr(ParamXSet!ParamSet)
        MaxParamNo = ParamXSet!NoParams
        ReDim FieldIndex(MaxParamNo) As Integer
        WS.Cells(CurrentRow + 1, 1).Value = "Parameter Set: " & CStr(ParamXSet!ParamSet)
```

```
WS.Cells(CurrentRow + 3, 1).Value = "Correlation Number:"
WS.Cells(CurrentRow + 4, 1).Value = "Param:"
If CurrentRow <> 2 Then
    ' includes everything up to but not including the first parameter row
    WS.Range("2:5").Copy
    ' copy table formats for first param set to every subsequent parameter set
    WS.Cells(CurrentRow, 1).PasteSpecial xlPasteFormats
End If
WS.Range("Pattern").Copy ' this is the first row
' copy formats to parameter rows
WS.Range(CStr(CurrentRow + 4) & ":" & CStr(CurrentRow + 3 + MaxParamNo)).PasteSpecial xlPasteFormats
For ParamNo = 0 To (MaxParamNo - 1)
    For Index = 2 To (Correl.Fields.Count - 1)
        ' Correl is a cross-table query with param names in column headings
        If Correl.Fields(Index).Name = ParamXSet!Name Then _
            FieldIndex(ParamNo) = Index
    Next Index
    ' set parameter name at beginning of row
    WS.Cells(CurrentRow + 4 + ParamNo, 2).Value = ParamXSet!Name
    ParamXSet.MoveNext
Next ParamNo
CurrentCol = 3
Correl.FindFirst Criterion
Do Until Correl.NoMatch
    WS.Cells(CurrentRow + 3, CurrentCol).Value = Correl!Value ' correl number
    For ParamNo = 0 To (MaxParamNo - 1)
        ' fill in param state name if it is not left blank
        If Not IsNull(Correl.Fields(FieldIndex(ParamNo)).Value) Then _
            WS.Cells(CurrentRow + 4 + ParamNo, CurrentCol).Value = _
                Correl.Fields(FieldIndex(ParamNo)).Value
    Next ParamNo
    Correl.FindNext Criterion
    CurrentCol = CurrentCol + 1
Loop
CurrentRow = CurrentRow + 4 + MaxParamNo
Loop
WS.Rows(1).Hidden = True
WS.Cells(5, 3).Activate
WS.Protect
Correl.Close
DoCmd.Echo True
End Sub

Private Sub Form_Deactivate()
    ParamXSet.Close
    DB.Close
    Me.Edit.Visible = True
    Me.printpreview.Visible = True
    DoCmd.GoToControl "Edit"
    Me.Submit.Visible = False
End Sub

Private Sub Form_Load()
    Set FMKey = New Form_fKeyParamStates
    FMKey.Visible = True
End Sub

Private Sub mainmenu_Click()
    If Me.Submit.Visible = True Then
        If MsgBox("Do you want to quit this form without submitting the information you " & _
            "provided on the sheet?", vbYesNo) = vbNo Then
            Edit_Click
            Exit Sub
        Else
            ParamXSet.Close
            DB.Close
        End If
    End If
End Sub
```

```
End If
End If
DoCmd.Close acForm, "fKeyParamStates"
DoCmd.OpenForm "pfMenuInitial"
DoCmd.Close acForm, "pfCorrelMain"
End Sub

Private Sub Submit_Click()
    Dim ParamStates As Recordset, Correlations As Recordset
    Dim CorrelXStates As Recordset, NameErrors As Boolean, CorrelID As Long
    Set Correlations = DB.OpenRecordset("tCorrelations", dbOpenTable)
    Set CorrelXStates = DB.OpenRecordset("tCorrel_X_States", dbOpenTable)
    Set ParamStates = DB.OpenRecordset("qParameterStates", dbOpenDynaset, dbReadOnly)
    WS.Activate
    CurrentRow = 2
    NameErrors = False
    'one pass-through only to check parameter state names
    ParamXSet.MoveFirst
    Do Until ParamXSet.EOF
        MaxParamNo = ParamXSet!NoParams
        CurrentCol = 3
        Do Until IsEmpty(WS.Cells(CurrentRow + 3, CurrentCol)) ' empty value cell
            For ParamNo = 0 To (MaxParamNo - 1)
                If Not IsEmpty(WS.Cells(CurrentRow + 4 + ParamNo, CurrentCol)) Then
                    Criterion = WS.Cells(CurrentRow + 4 + ParamNo, 2).Value & " " _
                        & WS.Cells(CurrentRow + 4 + ParamNo, CurrentCol).Value
                    If FindParamState(ParamStates, Criterion) = -1 Then
                        NameErrors = True
                        With WS.Cells(CurrentRow + 4 + ParamNo, CurrentCol)
                            .Value = ">>" & .Value
                        End With
                    End If
                End If
            Next ParamNo
            CurrentCol = CurrentCol + 1
        Loop
        ParamXSet.Move (MaxParamNo)
        CurrentRow = CurrentRow + 4 + MaxParamNo
    Loop
    If NameErrors Then
        Correlations.Close
        CorrelXStates.Close
        MsgBox "There are parameter states which were not found in the database." & _
            Chr(13) & "Please check again those names which are marked with '>>'.", _
            , "ERROR READING PARAMETER STATES"
        OB.Verb = -1
        OB.Action = acOLEActivate
        WS.Activate
        WS.Cells(5, 3).Activate
        Exit Sub
    End If

    ' delete all correlations
    DB.Execute ("dqCorrel_X_States")
    DB.Execute ("dqCorrelations")
    CurrentRow = 2
    ParamXSet.MoveFirst
    Do Until ParamXSet.EOF
        MaxParamNo = ParamXSet!NoParams
        CurrentCol = 3
        ' read each correlation from left to right
        Do Until IsEmpty(WS.Cells(CurrentRow + 3, CurrentCol))
            Correlations.addnew
            Correlations!ParamSet = ParamXSet!ParamSet
            Correlations!Value = WS.Cells(CurrentRow + 3, CurrentCol).Value
            CorrelID = Correlations!ID
            Correlations.Update
        For ParamNo = 0 To (MaxParamNo - 1)
```

```
        If Not IsEmpty(WS.Cells(CurrentRow + 4 + ParamNo, CurrentCol)) Then
            Criterion = WS.Cells(CurrentRow + 4 + ParamNo, 2).Value & " " _
                & WS.Cells(CurrentRow + 4 + ParamNo, CurrentCol).Value
            CorrelXStates.addnew
            CorrelXStates!Correlation = CorrelID
            CorrelXStates!ParamState = FindParamState(ParamStates, Criterion)
            CorrelXStates.Update
        End If
    Next ParamNo
    CurrentCol = CurrentCol + 1
Loop
ParamXSet.Move (MaxParamNo)
CurrentRow = CurrentRow + 4 + MaxParamNo
Loop
Correlations.Close
CorrelXStates.Close
ParamStates.Close
ParamXSet.Close
DB.Close
Me.Edit.Visible = True
Me.printpreview.Visible = True
DoCmd.GoToControl "Edit"
Me.Submit.Visible = False
End Sub

Private Function FindParamState(ParamStates As Recordset, Name As String) As Long
    ' fullname is name of parameter & name of param state separated by a blank and
    ' param state name in ''.
    ParamStates.FindFirst "FullName = '" & Name & "'"
    If ParamStates.NoMatch Then
        FindParamState = -1
    Else
        FindParamState = ParamStates!ID
    End If
End Function
```

Form_pfGCMain

This form is used to input ground classes and also contains an embedded Excel sheet.

```
Option Compare Database
Option Explicit
Dim OB As ObjectFrame, WS As Worksheet, ParamXSet As Recordset
Dim GCComb As Recordset, DB As Database, FMKey As Form, FMGC As Form
Dim ParamNo As Long, MaxParamNo As Long, CurrentRow As Integer, CurrentCol As Integer
Dim FieldIndex() As Integer, Index As Integer, Criterion As String

Private Sub Edit_Click()
    ' read in pfCorrelMain for comments
    Set OB = Me!ExcelSheet
    Set WS = OB.Worksheets("GC")
    Set DB = CurrentDb()
    Set ParamXSet = DB.OpenRecordset("qParam_X_Set", dbOpenSnapshot)
    If ParamXSet.RecordCount >= 10 Then _
        MsgBox "This action may take several minutes." & Chr(13) & "Please be patient." _
            & Chr(13) & Chr(13) & "Press OK to continue.", vbOKOnly, "EDIT GROUND CLASSES"
    Set GCComb = DB.OpenRecordset("pGCCCombinations", dbOpenDynaset)
    DoCmd.Echo False
    OB.Verb = -1
    OB.Action = acOLEActivate
    WS.Activate
    Me.Submit.Visible = True
    Me.Edit.Visible = False
    Me.EditNames.Visible = False
    Me.printpreview.Visible = False
    On Error Resume Next
```

```
' can't edit GC names while excel sheet is active
DoCmd.Close acForm, "fKeyGC", acSaveNo
WS.Unprotect
Set WS = OB.Worksheets("GC_Template")
WS.Activate
WS.Range("Everything").Copy
Set WS = OB.Worksheets("GC")
WS.Activate
WS.Cells(1, 1).PasteSpecial xlPasteFormats
WS.Cells(1, 1).PasteSpecial xlPasteValues
WS.Rows(1).Hidden = False
CurrentRow = 2
Do Until ParamXSet.EOF
    Criterion = "ParamSet = " & CStr(ParamXSet!ParamSet)
    MaxParamNo = ParamXSet!NoParams
    ReDim FieldIndex(MaxParamNo) As Integer
    WS.Cells(CurrentRow + 1, 1).Value = "Parameter Set: " & CStr(ParamXSet!ParamSet)
    WS.Cells(CurrentRow + 3, 1).Value = "Ground Class Name"
    WS.Cells(CurrentRow + 4, 1).Value = "Param:"
    If CurrentRow <> 2 Then
        WS.Range("2:5").Copy
        WS.Cells(CurrentRow, 1).PasteSpecial xlPasteFormats
    End If
    WS.Range("Pattern").Copy
    WS.Range(CStr(CurrentRow + 4) & ":" & _
        CStr(CurrentRow + 3 + MaxParamNo)).PasteSpecial xlPasteFormats
    For ParamNo = 0 To (MaxParamNo - 1)
        For Index = 2 To (GCComb.Fields.Count - 1)
            If GCComb.Fields(Index).Name = ParamXSet!Name Then FieldIndex(ParamNo) = Index
        Next Index
        WS.Cells(CurrentRow + 4 + ParamNo, 2).Value = ParamXSet!Name
        ParamXSet.MoveNext
    Next ParamNo
    CurrentCol = 3
    GCComb.FindFirst Criterion
    Do Until GCComb.NoMatch
        WS.Cells(CurrentRow + 3, CurrentCol).Value = GCComb!GCName
        For ParamNo = 0 To (MaxParamNo - 1)
            If Not IsNull(GCComb.Fields(FieldIndex(ParamNo)).Value) Then _
                WS.Cells(CurrentRow + 4 + ParamNo, CurrentCol).Value = _
                    GCComb.Fields(FieldIndex(ParamNo)).Value
        Next ParamNo
        GCComb.FindNext Criterion
        CurrentCol = CurrentCol + 1
    Loop
    CurrentRow = CurrentRow + 4 + MaxParamNo
Loop
WS.Rows(1).Hidden = True
WS.Cells(5, 3).Activate
WS.Protect
GCComb.Close
DoCmd.Echo True
End Sub

Private Sub EditNames_Click()
    Set FMGC = New Form_fKeyGC
    FMGC.Visible = True
End Sub

Private Sub Form_Deactivate()
    ParamXSet.Close
    DB.Close
    Me.Edit.Visible = True
    Me.EditNames.Visible = True
    Me.printpreview.Visible = True
    DoCmd.GoToControl "Edit"
    Me.Submit.Visible = False
```

```
End Sub

Private Sub Form_Load()
    Set FMKey = New Form_fKeyParamStates
    FMKey.Visible = True
End Sub

Private Sub mainmenu_Click()
    If Me.Submit.Visible = True Then
        If MsgBox("Do you want to quit this form without submitting the information " & _
            "you provided on the sheet?", vbYesNo) = vbNo Then
            Submit_Click
            Exit Sub
        Else
            ParamXSet.Close
            DB.Close
        End If
    End If
    DoCmd.OpenForm "pfMenuInitial"
    DoCmd.Close acForm, "pfGCMain"
End Sub

Private Sub Submit_Click()
    Dim ParamStates As Recordset, GC As Recordset, GCCombinations As Recordset
    Dim GCCombXStates As Recordset, NameErrors As Boolean, CombID As Long
    Set DB = CurrentDb()
    Set GC = DB.OpenRecordset("tGC", dbOpenTable)
    Set GCCombinations = DB.OpenRecordset("tGCCCombinations", dbOpenTable)
    Set GCCombXStates = DB.OpenRecordset("tGCCComb_X_States", dbOpenTable)
    Set ParamStates = DB.OpenRecordset("qParameterStates", dbOpenDynaset, dbReadOnly)
    WS.Activate
    CurrentRow = 2
    NameErrors = False
    'one pass-through only to check parameter state names
    ParamXSet.MoveFirst
    Do Until ParamXSet.EOF
        MaxParamNo = ParamXSet.NoParams
        CurrentCol = 3
        Do Until IsEmpty(WS.Cells(CurrentRow + 3, CurrentCol))
            For ParamNo = 0 To (MaxParamNo - 1)
                If Not IsEmpty(WS.Cells(CurrentRow + 4 + ParamNo, CurrentCol)) Then
                    Criterion = WS.Cells(CurrentRow + 4 + ParamNo, 2).Value & " " _
                        & WS.Cells(CurrentRow + 4 + ParamNo, CurrentCol).Value
                    If FindParamState(ParamStates, Criterion) = -1 Then
                        NameErrors = True
                        With WS.Cells(CurrentRow + 4 + ParamNo, CurrentCol)
                            .Value = ">>" & .Value
                        End With
                    End If
                End If
            Next ParamNo
            CurrentCol = CurrentCol + 1
        Loop
        ParamXSet.Move (MaxParamNo)
        CurrentRow = CurrentRow + 4 + MaxParamNo
    Loop
    If NameErrors Then
        GC.Close
        GCCombinations.Close
        GCCombXStates.Close
        MsgBox "There are parameter states which were not found in the database." & _
            Chr(13) & "Please check again those names which are marked with '>>'.", _
            , "ERROR READING PARAMETER STATES"
        OB.Verb = -1
        OB.Action = acOLEActivate
        WS.Activate
        WS.Cells(5, 3).Activate
    End If
End Sub
```



```
Exit Sub
End If
DB.Execute ("dqGCComb_X_States")
DB.Execute ("dqGCCombinations")
CurrentRow = 2
ParamXSet.MoveFirst
Do Until ParamXSet.EOF
    MaxParamNo = ParamXSet!NoParams
    CurrentCol = 3
    Do Until IsEmpty(WS.Cells(CurrentRow + 3, CurrentCol))
        GCCombinations.addnew
        GCCombinations!ParamSet = ParamXSet!ParamSet
        GCCombinations!GC = FindGC(GC, WS.Cells(CurrentRow + 3, CurrentCol).Value)
        CombID = GCCombinations!ID
        GCCombinations.Update
        For ParamNo = 0 To (MaxParamNo - 1)
            If Not IsEmpty(WS.Cells(CurrentRow + 4 + ParamNo, CurrentCol)) Then
                Criterion = WS.Cells(CurrentRow + 4 + ParamNo, 2).Value & " " _
                    & WS.Cells(CurrentRow + 4 + ParamNo, CurrentCol).Value
                GCCombXStates.addnew
                GCCombXStates!GCComb = CombID
                GCCombXStates!ParamState = FindParamState(ParamStates, Criterion)
                GCCombXStates.Update
            End If
        Next ParamNo
        CurrentCol = CurrentCol + 1
    Loop
    ParamXSet.Move (MaxParamNo)
    CurrentRow = CurrentRow + 4 + MaxParamNo
Loop
If Not (GC.EOF And GC.BOF) Then
    GC.MoveFirst
    Do Until GC.EOF
        On Error Resume Next
        GC.delete
        GC.MoveNext
    Loop
End If
GC.Close
GCCombinations.Close
GCCombXStates.Close
ParamStates.Close
ParamXSet.Close
DB.Close
Me.Edit.Visible = True
Me.EditNames.Visible = True
Me.printpreview.Visible = True
DoCmd.GoToControl "Edit"
Me.Submit.Visible = False
End Sub

Private Function FindGC(GC As Recordset, Name As String) As Long
    GC.Index = "Name"
    GC.Seek "=", Name
    If GC.NoMatch Then
        GC.addnew
        GC!Name = Name
        FindGC = GC!ID
        GC.Update
    Else
        FindGC = GC!ID
    End If
End Function

Private Function FindParamState(ParamStates As Recordset, Name As String) As Long
    ParamStates.FindFirst "FullName = '" & Name & "'"
    If ParamStates.NoMatch Then
```

```
        FindParamState = -1
    Else
        FindParamState = ParamStates!ID
    End If
End Function
```

Form_pfGenVariablesMain

This (sub)form is used to initially input general variables.

```
Private Sub remove_Click()
    Dim Var As Recordset, Variable As Long, RS As Recordset, NextVariable As Long
    SendKeys "{ESC}{ESC}"
    If IsNull(Me!ID) Then Exit Sub
    Variable = Me!ID
    RunCommand acCmdDeleteRecord
    Set RS = Me.Recordset.Clone
    If RS.BOF And RS.EOF Then
        Set Var = CurrentDb().OpenRecordset("tGenVariables", dbOpenTable)
        Var.Index = "PrimaryKey"
        Var.Seek "=", Variable
        Var.delete
        Var.Close
        Exit Sub
    End If
    RS.Bookmark = Me.Bookmark
    RS.MoveNext
    ' find ID of next record to position cursor after requery
    If RS.EOF Then
        NextVariable = 0
    Else
        NextVariable = Me!Variable
    End If
    Set Var = CurrentDb().OpenRecordset("tGenVariables", dbOpenTable)
    Var.Index = "PrimaryKey"
    Var.Seek "=", Variable
    Var.delete
    Var.Close
    Me.Requery
    RS.Requery
    If NextVariable = 0 Then
        DoCmd.GoToRecord , , acNewRec
    Else
        RS.FindFirst "Variable = " & CStr(NextVariable)
    End If
    Me.Bookmark = RS.Bookmark
End Sub

Private Sub unifysimulall_Click()
    Dim RS As Recordset ' check all "one value per simulation" boxes
    Set RS = Me.Recordset.Clone
    If RS.EOF And RS.BOF Then Exit Sub
    RS.MoveFirst
    Do Until RS.EOF
        RS.Edit
        RS!UnifiedSimul = True
        RS.Update
        RS.MoveNext
    Loop
    RS.Close
    Me.Requery
End Sub

Private Sub unifysimulall_Db1Click(Cancel As Integer)
    Dim RS As Recordset ' uncheck all "one value per simulation" boxes
```

```
Set RS = Me.Recordset.Clone
If RS.EOF And RS.BOF Then Exit Sub
RS.MoveFirst
Do Until RS.EOF
    RS.Edit
    RS!UnifiedSimul = False
    RS.Update
    RS.MoveNext
Loop
RS.Close
Me.Requery
End Sub
```

Form_pfGeometires

```
Private Sub remove_Click()
    Dim Tunnels As Recordset, GCGeom As Recordset, Assoc As Recordset
    Set Tunnels = CurrentDb().OpenRecordset("tTunnels", dbOpenTable, dbReadOnly)
    Tunnels.Index = "Geometry"
    Tunnels.Seek "=", Me!ID
    If Tunnels.NoMatch Then
        If MsgBox("Do you really want to delete this geometry and all " & Chr(13) & _
            "geometry/ground class/method associations referring to it?", vbYesNo, _
            "DELETE GEOMETRY") = vbYes Then
            Set GCGeom = CurrentDb().OpenRecordset("otGCGeom", dbOpenTable)
            GCGeom.Index = "PrimaryKey"
            GCGeom.Seek ">=", Me!ID
            Do Until GCGeom.NoMatch
                If GCGeom.EOF Then Exit Do
                If GCGeom.Geometry = Me!ID Then GCGeom.delete
                GCGeom.Seek ">=", Me!ID
            Loop
            GCGeom.Close
            Set Assoc = CurrentDb().OpenRecordset("tCGC_X_Meth_X_Geom", dbOpenTable)
            Assoc.Index = "PrimaryKey"
            Assoc.Seek ">=", Me!ID
            Do Until Assoc.NoMatch
                If Assoc.EOF Then Exit Do
                If Assoc.Geometry = Me!ID Then Assoc.delete
                Assoc.Seek ">=", Me!ID
            Loop
            Assoc.Close
            SendKeys "{ESC}{ESC}"
            RunCommand acCmdDeleteRecord
        End If
    Else
        MsgBox "You can't delete this geometry because there is at least" & Chr(13) & _
            "one tunnel referring to it!", vbOKOnly, "ERROR DELETING GEOMETRY"
    End If
    Tunnels.Close
End Sub
```

Form_pfMarkovMainForm

This form is used to initially input Markov parameters.

```
Private Sub transitionmatrix_Click()
    Static Criterion As String
    ' cross-table queries cannot be parametrized. therefore we have to change the
    ' SQL statement directly to limit the table to the actual parameter and parameter set
    Criterion = "TRANSFORM First(tMarkovTransProb.Prob) AS FirstOfProb " & _
        "SELECT tParamStates.Name FROM tParamStates INNER JOIN " & _
        "(tMarkovTransProb INNER JOIN tParamStates AS tParamStates_1 ON " & _
        "tMarkovTransProb.ToParamState = tParamStates_1.ID) " & _
```

```
"ON tParamStates.ID = tMarkovTransProb.FromParamState " & _
"WHERE (((tMarkovTransProb.UpdatePhase) = 1) And ((tMarkovTransProb.ParamSet) = " & _
CStr(Me!ParamSet)&")And((tParamStates.Parameter) = " & CStr(Me!Parameter) & ")) " & _
" GROUP BY tParamStates.Name, tMarkovTransProb.UpdatePhase, " & _
"tMarkovTransProb.ParamSet, tParamStates.Parameter PIVOT tParamStates_1.Name;"
CurrentDb().QueryDefs("pMarkovTransProbMatrix").Properties("SQL") = Criterion
DoCmd.OpenQuery "pMarkovTransProbMatrix", , acReadOnly
DoCmd.OpenForm "pfMarkovTransProbMatrixClose"
End Sub
```

Form_pfMethodHeadsMain

This form is used to initially input headings and cycle lengths.

```
Private Sub changeview_Click()
    Dim CT As ComboBox
    DoCmd.GoToControl "pfMethodsHeadsSub"
    RunCommand acCmdSubformDatasheet
    Set CT = Me!pfMethodsHeadsSub!Method
    ' in form view only headings of the active method (method on method form) can be
    ' edited, i.e. the method field is locked. in datasheet view all headings can be
    ' edited, i.e. the method field is active. also the form is wider.
    With Me.pfMethodsHeadsSub
        If .LinkChildFields = "" Then
            .LinkChildFields = "Method"
            .LinkMasterFields = "ID"
            .Width = 7083
            Me.InsideWidth = 7660
            CT.TabStop = False
            Me!pfMethodsHeadsSub!Order.SetFocus
            CT.Enabled = False
            CT.Locked = True
            CT.DefaultValue = Me!ID
        Else
            .LinkChildFields = ""
            .LinkMasterFields = ""
            .Width = 10920
            Me.InsideWidth = 11470
            CT.TabStop = True
            CT.Enabled = True
            CT.Locked = False
            CT.DefaultValue = ""
            DoCmd.GoToControl "Method"
        End If
    End With
End Sub
```

Form_pfMethodHeadsSub

```
Private Sub remove_Click()
    Dim Nodes As Recordset, Arcs As Recordset, Prob As Recordset, Heads As Recordset
    Dim Head As Long, Criterion1 As String, Criterion2 As String
    ' we can only delete the Heading if we delete the activity network first
    Me.Requery
    SendKeys "{ESC}{ESC}{ESC}"
    If IsNull(Me!ID) Then Exit Sub
    If MsgBox("Do you want to delete this heading and all pertaining nodes and arcs?" _
        , vbYesNo, "ERROR DELETING HEAD") = vbNo Then Exit Sub
    Set Nodes = CurrentDb().OpenRecordset("tMethNodes", dbOpenDynaset)
    Set Arcs = CurrentDb().OpenRecordset("tMethArcs", dbOpenDynaset)
    Set Prob = CurrentDb().OpenRecordset("tHeadProb", dbOpenTable)
    Head = Me!ID
    Criterion1 = "Head =" & CStr(Head)
    Nodes.FindFirst Criterion1
```

```
Do Until Nodes.NoMatch
    Criterion2 = "StartNode = " & CStr(Nodes!ID) & " Or EndNode = " & CStr(Nodes!ID)
    Arcs.FindFirst Criterion2
    Do Until Arcs.NoMatch
        Arcs.delete
        Arcs.FindNext Criterion2
    Loop
    Nodes.delete
    Nodes.FindNext Criterion1
Loop
Prob.Index = "PrimaryKey"
Prob.Seek ">=", Head
Do Until Prob.NoMatch
    If Prob!Head <> Head Then Exit Do
    Prob.delete
    Prob.Seek ">=", Me!ID
Loop
Nodes.Close
Arcs.Close
Prob.Close
Set Heads = CurrentDb().OpenRecordset("tHeads", dbOpenDynaset)
Heads.FindFirst "ID =" & CStr(Head)
If Not Heads.NoMatch Then Heads.delete
Heads.Close

On Error GoTo Repair
DoCmd.GoToRecord , , acPrevious
Me.Requery
Exit Sub
Repair:
DoCmd.GoToRecord , , acFirst
Resume Next
End Sub
```

Form_pfMethodsLearnMain

```
Private Sub changeview_Click()
    Dim CT As ComboBox
    DoCmd.GoToControl "pfMethodsLearnSub"
    RunCommand acCmdSubformDatasheet
    Set CT = Me!pfMethodsLearnSub!Method
    ' see explanations in heading form
    With Me.pfMethodsLearnSub
        If .LinkChildFields = "" Then
            .LinkChildFields = "Method"
            .LinkMasterFields = "ID"
            CT.TabStop = False
            Me!pfMethodsLearnSub!Efficiency.SetFocus
            CT.Enabled = False
            CT.Locked = True
            CT.DefaultValue = Me!ID
        Else
            .LinkChildFields = ""
            .LinkMasterFields = ""
            CT.TabStop = True
            CT.Enabled = True
            CT.Locked = False
            CT.DefaultValue = ""
            DoCmd.GoToControl "Method"
        End If
    End With
End Sub
```

Form_pfMethodsSub

```
Private Sub Form_BeforeUpdate(Cancel As Integer)
    Dim Heads As Recordset, HeadProb As Recordset
    If Me!CycleLength <= 0 Then
        Me!CycleLength = -1 ' not deterministic cycle length
    Else ' deterministic cycle length
        ' --> change in headprob for initial input in all headings
        Set Heads = CurrentDb().OpenRecordset("tHeads", dbOpenTable, dbReadOnly)
        Heads.Index = "Method"
        Set HeadProb = CurrentDb().OpenRecordset("tHeadProb", dbOpenTable)
        HeadProb.Index = "PrimaryKey"
        Heads.Seek ">=", Me!ID
        If Heads.NoMatch Then Exit Sub
        Do While Heads!Method = Me!ID
            HeadProb.Seek "=", Heads!ID, 1
            If Not HeadProb.NoMatch Then
                HeadProb.Edit
                HeadProb!CycleProbMin = 0
                HeadProb!CycleMin = Me!CycleLength
                HeadProb!CycleMode = Me!CycleLength
                HeadProb!CycleMax = Me!CycleLength
                HeadProb!CycleProbMax = 0
                HeadProb.Update
            End If
            Heads.MoveNext
            If Heads.EOF Then Exit Do
        Loop
        Heads.Close
        HeadProb.Close
    End If
End Sub
```

Form_pfMethodVariablesMain

```
Private Sub changeview_Click()
    Dim CT As ComboBox
    DoCmd.GoToControl "pfMethodsVariablesSub"
    RunCommand acCmdSubformDatasheet
    Set CT = Me!pfMethodsVariablesSub!Method
    ' see explanations in heading form
    With Me.pfMethodsVariablesSub
        If .LinkChildFields = "" Then
            .LinkChildFields = "Method"
            .LinkMasterFields = "ID"
            .Width = 7083
            Me.InsideWidth = 7660
            CT.TabStop = False
            Me!pfMethodsVariablesSub!Name.SetFocus
            CT.Enabled = False
            CT.Locked = True
            CT.DefaultValue = Me!ID
        Else
            .LinkChildFields = ""
            .LinkMasterFields = ""
            .Width = 10010
            Me.InsideWidth = 10570
            CT.TabStop = True
            CT.Enabled = True
            CT.Locked = False
            CT.DefaultValue = ""
            DoCmd.GoToControl "Method"
        End If
    End With
End Sub
```

Form_pfMethodVariablesSub

```
Private Sub remove_Click()  
    Dim Var As Recordset, Variable As Long, RS As Recordset, NextVariable As Long  
    SendKeys "{ESC}{ESC}"  
    If IsNull(Me!ID) Then Exit Sub  
    Variable = Me!ID  
    RunCommand acCmdDeleteRecord  
    Set RS = Me.Recordset.Clone  
    If Me.Recordset.RecordCount > 0 Then  
        RS.Bookmark = Me.Bookmark  
        RS.MoveNext  
        ' find ID of next record to position cursor after requery  
        If RS.EOF Then  
            NextVariable = 0  
        Else  
            NextVariable = Me!Variable  
        End If  
    End If  
    Set Var = CurrentDb().OpenRecordset("tVariables", dbOpenTable)  
    Var.Index = "PrimaryKey"  
    Var.Seek "=", Variable  
    If Not Var.NoMatch Then Var.delete  
    Var.Close  
    On Error Resume Next  
    Me.Requery  
    RS.Requery  
    If NextVariable = 0 Then  
        DoCmd.GoToRecord , , acNewRec  
    Else  
        RS.FindFirst "Variable = " & CStr(NextVariable)  
        Me.Bookmark = RS.Bookmark  
    End If  
End Sub
```

Form_pfNonMarkovMain

This form is used to input deterministic and semi-deterministic parameters.

```
Dim opened As Boolean  
' note that both the deterministic and semi-deterministic subform are on this main form  
' depending on the generation mode one of the two has to be made visible.
```

```
Private Sub changeview_Click()  
On Error GoTo Repair  
    If Me.pfNonMarkovDeter.Visible = True Then  
        DoCmd.GoToControl "pfNonMarkovDeter"  
    ElseIf Me.pfNonMarkovSemiDeter.Visible = True Then  
        DoCmd.GoToControl "pfNonMarkovSemiDeter"  
    End If  
    RunCommand acCmdSubformDatasheet  
    Exit Sub  
Repair:  
    Resume Next  
End Sub
```

```
Private Sub Parameter_AfterUpdate()  
    Dim CT As Control, GenerationMode As String  
    If Not (IsNull(Me!ParamSet) Or IsNull(Me!Parameter)) Then  
        GenerationMode = DLookup("GenerationMode", "pNonMarkovMain", _  
            "ParamSet = " & Str(Me!ParamSet) & " And Parameter = " & Str(Me!Parameter))  
        If GenerationMode = "1" Then  
            Me.pfNonMarkovDeter.Requery  
            If opened Then  
                Set CT = Me!pfNonMarkovDeter!ParamState
```

```
        CT.Requery
    End If
    Me.pfNonMarkovDeter.Visible = True
    Me.pfNonMarkovSemiDeter.Visible = False
    Me.Title.Caption = "Deterministic Segments"
    Me.warning.Visible = False
ElseIf GenerationMode = "2" Then
    Me.pfNonMarkovSemiDeter.Requery
    If opened Then
        Set CT = Me!pfNonMarkovSemiDeter!ParamState
        CT.Requery
    End If
    Me.pfNonMarkovSemiDeter.Visible = True
    Me.pfNonMarkovDeter.Visible = False
    Me.Title.Caption = "Semi-Deterministic Segments"
    If IsNull(DLookup("ID", "tUpdatePhases", "Current = True")) Then
        Me.warning.Visible = False
    Else
        Me.warning.Visible = True
    End If
End If
Else
    Me.pfNonMarkovSemiDeter.Visible = False
    Me.pfNonMarkovDeter.Visible = False
    Me.Title.Caption = "No Parameters with (Semi-)Deterministic Generation Mode " & _
        in this Parameter Set"
End If
opened = True
End Sub
```

Form_pfParamMain

This form is used to input geotechnical parameters, parameter states and parameter sets.

```
Private Sub Form_Close()
    Dim ML As Recordset, ParamXSet As Recordset, ParamStates As Recordset
    Dim UP As Recordset, Graph As Recordset, MTP As Recordset, ParamStates2 As Recordset
    Dim Current As Long
    Set ML = CurrentDb().OpenRecordset("tMarkovLengths", dbOpenTable)
    Set MTP = CurrentDb().OpenRecordset("tMarkovTransProb", dbOpenTable)
    ML.Index = "PrimaryKey"
    MTP.Index = "PrimaryKey"
    Set ParamXSet = CurrentDb().OpenRecordset("tParam_X_Set", dbOpenTable, dbReadOnly)
    Set ParamStates = CurrentDb().OpenRecordset("tParamStates", dbOpenSnapshot)
    Set ParamStates2 = CurrentDb().OpenRecordset("tParamStates", dbOpenSnapshot)
    Set UP = CurrentDb().OpenRecordset("tUpdatePhases", dbOpenTable)
    Set Graph = CurrentDb().OpenRecordset("otMarkovGraph", dbOpenTable)
    Graph.Index = "PrimaryKey"

    ' this procedure is in case a parameter state has been added to a parameter
    ' which has been used with Markov gen mode in a parameter set. In this case
    ' we must create the MarkovLength record for this state as well as the
    ' Transition probability records. This procedure checks the existence of each
    ' record that should exist and creates a new one if it doesn't.

    If ParamXSet.EOF And ParamXSet.EOF Then Exit Sub
    UP.Index = "Current"
    UP.Seek "=", True
    If UP.NoMatch Then
        Current = 1
    Else
        Current = UP!ID
    End If
    ParamXSet.MoveFirst
    Do Until ParamXSet.EOF
```



```
If ParamXSet!GenerationMode = "0" Then
    ParamStates.FindFirst "Parameter = " & CStr(ParamXSet!Parameter)
    Do Until ParamStates.NoMatch
        UP.MoveFirst
        Do Until UP.EOF
            If UP!ID <> 11 And Not (UP!ID = 10 And Current = 1) Then
                ' do not add phase 10 record if there is no current update phase
                ML.Seek "=", ParamXSet!ParamSet, ParamStates!ID, UP!ID
                If ML.NoMatch Then
                    ML.addnew
                    ML!ParamSet = ParamXSet!ParamSet
                    ML!ParamState = ParamStates!ID
                    ML!UpdatePhase = UP!ID
                    ML.Update
                    ML.Seek "=", ParamXSet!ParamSet, ParamStates!ID, UP!ID
                    ' only create posterior graph (prior will be a singular spike)
                    If UP!ID = Current And Current > 1 Then _
                        Graphs.GraphMarkovPriorPost ML, Graph, "Posterior"
                End If
            End If
            UP.MoveNext
        Loop
        ' do second loop for transition probabilities (ToParamState)
        ParamStates2.FindFirst "Parameter = " & CStr(ParamXSet!Parameter) & _
            " And ID <> " & CStr(ParamStates!ID) ' exclude Tij where i=j
        Do Until ParamStates2.NoMatch
            UP.MoveFirst
            Do Until UP.EOF
                If UP!ID <> 11 And Not (UP!ID = 10 And Current = 1) Then
                    ' do not add phase 10 record if there is no current update phase
                    MTP.Seek "=", ParamXSet!ParamSet, ParamStates!ID, _
                        ParamStates2!ID, UP!ID
                    If MTP.NoMatch Then
                        MTP.addnew
                        MTP!ParamSet = ParamXSet!ParamSet
                        MTP!FromParamState = ParamStates!ID
                        MTP!ToParamState = ParamStates2!ID
                        MTP!UpdatePhase = UP!ID
                        MTP.Update
                    End If
                End If
                UP.MoveNext
            Loop
            ParamStates2.FindNext "Parameter = " & CStr(ParamXSet!Parameter) & _
                " And ID <> " & CStr(ParamStates!ID)
        Loop
        ParamStates.FindNext "Parameter = " & CStr(ParamXSet!Parameter)
    Loop
End If
ParamXSet.MoveNext
Loop
ParamStates.Close
ParamXSet.Close
ML.Close
End Sub
```

Form_pfParamSet

This is a sub-form of the previous one.

```
Private Sub ParamSet_NotInList(NewData As String, Response As Integer)
    Static ParamSets As Recordset
    Response = acDataErrAdded
    If MsgBox("This parameter set does not yet exist!" & Chr(13) & _
        "Do you want to create it?", vbYesNo, "PARAMETER SET") = vbYes Then
```

```
On Error GoTo Repair
If CLng(NewData) > 1000000000 Then
    MsgBox "Parameter set numbers must be < 1'000'000'000", , "PARAMETER SET"
    Me!ParamSet = Null
Else
    Set ParamSets = CurrentDb().OpenRecordset("tParamSets", dbOpenTable)
    ParamSets.addnew
    ParamSets!ID = CLng(NewData)
    ParamSets.Update
    ParamSets.Close
End If
Else
    Me!ParamSet = Null
End If
Exit Sub
Repair:
MsgBox "Parameter set must be a number.", , "PARAMETER SET"
Me!ParamSet = Null
Exit Sub
End Sub

Private Sub removeparamset_Click()
    Static ParamSets As Recordset
    Set ParamSets = CurrentDb().OpenRecordset("tParamSets", dbOpenTable)
    ParamSets.Index = "PrimaryKey"
    ParamSets.Seek "=", Me!ParamSet
    On Error GoTo Repair
    ParamSets.delete
    ParamSets.MoveNext
    If ParamSets.EOF Then
        ParamSets.MoveLast
        If ParamSets.EOF Then
            Me!ParamSet = Null
        Else
            Me!ParamSet = ParamSets!ID
        End If
    Else
        Me!ParamSet = ParamSets!ID
    End If
    ParamSets.Close
    Me.ParamSet.Requery
    Exit Sub
Repair:
MsgBox "Can't delete this parameter set because there is" _
    & " other data which refers to it.", , "ERROR DELETING PARAMETER SET"
    ParamSets.Close
    Exit Sub
End Sub
```

Form_pfParamSetParam

And this is even a sub-form of the previous one.

```
Private Sub remove_Click()
    Dim DB As Database, ParamStates As Recordset, MTP As Recordset, GCComb As Recordset
    Dim GCStat As Recordset, Correl As Recordset, CorrelStat As Recordset
    Dim BH As Recordset, BHStat As Recordset, AreaObs As Recordset, Areas As Recordset
    Dim ZoneObs As Recordset, Zones As Recordset, LastID As Long
    If Not IsNull(Me!Parameter) Then
        If MsgBox("If you remove this parameter from this parameter set, any related" & _
            Chr(13) & "data will also be deleted." & Chr(13) & Chr(13) & _
            "Do you want remove this parameter from this parameter set?", vbYesNo, _
            "WARNING:") = vbNo Then
            Exit Sub
        Else
            Exit Sub
        End If
    End If
```

```
' change gen mode to "" simply deletes the existing records of Markov lengths,
' transition prob, deterministic and semideterministic segments
ChangeGenMode Me!ParamSet, Me!Parameter, Me!GenerationMode, ""
Set DB = CurrentDb()
Set ParamStates = DB.OpenRecordset("tParamStates", dbOpenSnapshot)
Set MTP = DB.OpenRecordset("tMarkovTransProb", dbOpenTable)
Set GCComb = DB.OpenRecordset("tGCCCombinations", dbOpenTable)
Set GCStat = DB.OpenRecordset("tGCCComb_X_States", dbOpenTable)
Set Correl = DB.OpenRecordset("tCorrelations", dbOpenTable)
Set CorrelStat = DB.OpenRecordset("tCorrel_X_States", dbOpenTable)
Set BH = DB.OpenRecordset("tBoreholes", dbOpenTable)
Set BHStat = DB.OpenRecordset("tBH_X_States", dbOpenTable)
GCComb.Index = "PrimaryKey"
GCStat.Index = "ParamState"
Correl.Index = "PrimaryKey"
CorrelStat.Index = "ParamState"
BH.Index = "PrimaryKey"
BHStat.Index = "ParamState"
Set Areas = DB.OpenRecordset("tAreas", dbOpenTable)
Areas.Index = "ParamSet"
Areas.Seek "=", Me!ParamSet
If Not Areas.NoMatch Then
    Set AreaObs = DB.OpenRecordset("otAreaParamStat", dbOpenTable)
    AreaObs.Index = "ParamState"
End If
Set Zones = DB.OpenRecordset("tZones", dbOpenTable)
Zones.Index = "ParamSet"
Zones.Seek "=", Me!ParamSet
If Not Zones.NoMatch Then
    Set ZoneObs = DB.OpenRecordset("otZoneParamStat", dbOpenTable)
    ZoneObs.Index = "ParamState"
End If
ParamStates.FindFirst "Parameter = " & CStr(Me!Parameter)
Do Until ParamStates.NoMatch
    ' trans prob are observed for all gen modes --> not deleted in
    ' ChangeGenMode procedure if gen mode is (semi-)deterministic
    MTP.Index = "PrimaryKey"
    MTP.Seek ">=", ParamSet, ParamStates!ID
    Do Until MTP.NoMatch
        If MTP.EOF Then Exit Do
        If MTP!ParamSet <> ParamSet Or _
            MTP!FromParamState <> ParamStates!ID Then Exit Do
        MTP.delete ' Markov transition probabilities
        MTP.Seek ">=", ParamSet, ParamStates!ID
    Loop
    ' find a GC-State-Combination with this parameter state
    GCStat.Seek "=", ParamStates!ID
    Do Until GCStat.NoMatch
        ' check if the found combination belongs to this parameter set
        GCComb.Seek "=", GCStat!GCComb
        If GCComb!ParamSet = ParamSet Then
            GCStat.delete
            ' delete GC combination if this was the only parameter state
            GCStat.Index = "GCComb"
            GCStat.Seek "=", GCComb!ID
            If GCStat.NoMatch Then GCComb.delete
            GCStat.Index = "ParamState"
        End If
        GCStat.Seek "=", ParamStates!ID
    Loop
    ' find a Correlation-State-Combination with this parameter state
    CorrelStat.Seek "=", ParamStates!ID
    Do Until CorrelStat.NoMatch
        ' check if the found combination belongs to this parameter set
        Correl.Seek "=", CorrelStat!Correlation
        If Correl!ParamSet = ParamSet Then CorrelStat.delete
        CorrelStat.Seek "=", ParamStates!ID
    Loop
```

```
' find a Borehole-State-Combination with this parameter state
BHStat.Seek "=", ParamStates!ID
Do Until BHStat.NoMatch
    ' check if the found borehole belongs to this parameter set
    BH.Seek "=", BHStat!Borehole
    If BH!ParamSet = ParamSet Then BHStat.delete
    BHStat.Seek "=", ParamStates!ID
Loop
If Not (Areas.EOF And Areas.BOF) Then Areas.MoveFirst
Do Until Areas.EOF
    If Areas!ParamSet = Me!ParamSet Then
        AreaObs.Seek "=", ParamStates!ID, Areas!ID
        Do Until AreaObs.NoMatch
            AreaObs.delete
            AreaObs.Seek "=", ParamStates!ID, Areas!ID
        Loop
    End If
    Areas.MoveNext
Loop
If Not (Zones.EOF And Zones.BOF) Then Zones.MoveFirst
Do Until Zones.EOF
    If Zones!ParamSet = Me!ParamSet Then
        ZoneObs.Seek "=", ParamStates!ID, Zones!ID
        Do Until ZoneObs.NoMatch
            ZoneObs.delete
            ZoneObs.Seek "=", ParamStates!ID, Zones!ID
        Loop
    End If
    Zones.MoveNext
Loop
ParamStates.FindNext "Parameter = " & CStr(Me!Parameter)
Loop
CorrelStat.Index = "Correlation"
If Not Correl.EOF And Correl.BOF Then Correl.MoveFirst
Do Until Correl.EOF
    ' don't waste time on searching in other parameter sets
    If Correl!ParamSet = ParamSet Then
        CorrelStat.Seek "=", Correl!ID
        If CorrelStat.NoMatch Then
            LastID = Correl!ID
            Correl.delete
            Correl.Seek ">=", LastID
            If Correl.NoMatch Then Exit Do
            If Correl.EOF Then Exit Do
        Else
            CorrelStat.MoveNext
            ' must have at least two states for every correlation
            If CorrelStat.EOF Then
                CorrelStat.MoveLast
                CorrelStat.delete
                Correl.delete
            ElseIf CorrelStat!Correlation <> Correl!ID Then
                Correl.MovePrevious
                CorrelStat.delete
                Correl.delete
            End If
        End If
    End If
    Correl.MoveNext
Loop
End If
On Error Resume Next
RunCommand acCmdDeleteRecord
End If
End Sub

Private Sub GenerationMode_AfterUpdate()
    If IsNull(Me!Parameter) Then
```

```
' user has to choose a generation mode
SendKeys "{ESC}"
DoCmd.GoToControl ("Parameter")
Exit Sub
End If
If GenMode = "" Then
    ChangeGenMode ParamSet, Parameter, GenMode, Me!GenerationMode
    Exit Sub
End If
If Me!GenerationMode <> GenMode Then
    If MsgBox("If you change the generation mode any input data related to the" & _
        Chr(13) & "old generation mode will be lost." & Chr(13) & Chr(13) & _
        "Do you want change generation mode? ", vbYesNo, "WARNING:") = vbNo Then
        Me!GenerationMode = GenMode
    Else
        ChangeGenMode ParamSet, Parameter, GenMode, Me!GenerationMode
    End If
End If
End Sub

Private Sub GenerationMode_GotFocus()
    ' remember genmode before change
    GenMode = Me!GenerationMode
End Sub

Private Sub ChangeGenMode(ParamSet As Long, Parameter As Long, _
    OldGenMode As String, NewGenMode As String)
    Dim ParamStates As Recordset, ML As Recordset, MTP As Recordset, SDS As Recordset
    Dim SDSP As Recordset, Graph As Recordset, OTrans As Recordset
    Dim DS As Recordset, DB As Database, PStat As Long, UP As Recordset
    Dim NoParamStat As Double, Bookmark As Variant, FirstPStat As Long, Current As Long
    Set DB = CurrentDb()
    Set ParamStates = DB.OpenRecordset("tParamStates", dbOpenSnapshot)
    FirstPStat = -1
    NoParamStat = 0
    ParamStates.FindFirst "Parameter = " & CStr(Parameter)
    Do Until ParamStates.NoMatch
        If FirstPStat = -1 Then FirstPStat = ParamStates!ID
        NoParamStat = NoParamStat + 1
        ParamStates.FindNext "Parameter = " & CStr(Parameter)
    Loop
    Set UP = DB.OpenRecordset("tUpdatePhases", dbOpenTable, dbReadOnly)
    UP.Index = "Current"
    UP.Seek "=", True
    If UP.NoMatch Then
        Current = 1
    Else
        Current = UP!ID
    End If
    UP.Index = "PrimaryKey"
    If OldGenMode = "0" Or NewGenMode = "0" Then ' Markov
        Set ML = DB.OpenRecordset("tMarkovLengths", dbOpenTable)
        Set MTP = DB.OpenRecordset("tMarkovTransProb", dbOpenTable)
        Set Graph = DB.OpenRecordset("otMarkovGraph", dbOpenTable)
        Set OTrans = DB.OpenRecordset("otTransitions", dbOpenTable)
        Graph.Index = "PrimaryKey"
    End If
    If OldGenMode = "1" Or NewGenMode = "1" Then ' Deterministic
        Set DS = DB.OpenRecordset("tDeterSegs", dbOpenTable)
    End If
    If OldGenMode = "2" Or NewGenMode = "2" Then ' Semi-Deterministic
        Set SDS = DB.OpenRecordset("tSemiDeterSegs", dbOpenTable)
        Set SDSP = DB.OpenRecordset("tSegProb", dbOpenTable)
    End If
    ParamStates.FindFirst "Parameter = " & CStr(Parameter)
    Do Until ParamStates.NoMatch
        PStat = ParamStates!ID
```

```
Select Case OldGenMode
Case "0"
  ML.Index = "PrimaryKey"
  ML.Seek ">=", ParamSet, PStat
  Do Until ML.NoMatch
    If ML!ParamSet <> ParamSet Or ML!ParamState <> PStat Then Exit Do
    ML.delete ' Markov lengths
    ML.Seek ">=", ParamSet, PStat
  Loop
  MTP.Index = "PrimaryKey"
  MTP.Seek ">=", ParamSet, PStat
  Do Until MTP.NoMatch
    If MTP!ParamSet <> ParamSet Or MTP!FromParamState <> PStat Then Exit Do
    MTP.delete ' Markov transition probabilities
    MTP.Seek ">=", ParamSet, PStat
  Loop
  Graph.Index = "PrimaryKey"
  Graph.Seek ">=", ParamSet, PStat
  Do Until Graph.NoMatch
    If Graph!ParamSet <> ParamSet Or Graph!ParamState <> PStat Then Exit Do
    Graph.delete ' Markov graph
    Graph.Seek ">=", ParamSet, PStat
  Loop
  OTrans.Index = "PrimaryKey"
  OTrans.Seek "=", ParamSet, PStat
  If Not OTrans.NoMatch Then _
    OTrans.delete ' Number of observed transitions
Case "1"
  DS.Index = "ParamSet"
  DS.Seek ">=", ParamSet, PStat
  Do Until DS.NoMatch
    If DS!ParamSet <> ParamSet Or DS!ParamState <> PStat Then Exit Do
    DS.delete ' deterministic segments
    DS.Seek ">=", ParamSet, PStat
  Loop
Case "2"
  SDS.Index = "ParamSet"
  SDSP.Index = "PrimaryKey"
  SDS.Seek "=", ParamSet, PStat
  Do Until SDS.NoMatch
    SDSP.Seek ">=", SDS!ID ' semi-deter segment
    Do Until SDSP.NoMatch
      If SDSP!Segment <> SDS!ID Then Exit Do
      SDSP.delete ' semi-deter segment probability
      SDSP.Seek ">=", SDS!ID
    Loop
    SDS.delete
    SDS.Seek "=", ParamSet, PStat
  Loop
End Select
UP.MoveFirst
Do Until UP.EOF
  If NewGenMode = "0" Then
    If UP!ID <> 11 And Not (UP!ID = 10 And Current = 1) Then
      ' do not add phase 10 record if there is no current update phase
      ML.addnew
      ML!ParamSet = ParamSet
      ML!ParamState = PStat
      ML!UpdatePhase = UP!ID
      ' first state of each parameter has start prob = 1, others start prob = 0
      ML!StartProb = 1 / NoParamStat
      ML.Update
    End If
    If UP!ID <> 11 And Not (UP!ID = 10 And Current = 1) Then
      ' add all transition probabilities except between same state
      Bookmark = ParamStates.Bookmark
      ParamStates.FindFirst "Parameter = " & CStr(Parameter)
      Do Until ParamStates.NoMatch
```

```

    If PStat <> ParamStates!ID Then
        MTP.addnew
        MTP!ParamSet = ParamSet
        MTP!FromParamState = PStat
        MTP!ToParamState = ParamStates!ID
        MTP!UpdatePhase = UP!ID
        MTP!Prob = 1 / (NoParamStat - 1)
        MTP.Update
    End If
    ParamStates.FindNext "Parameter = " & CStr(Parameter)
    Loop
    ParamStates.Bookmark = Bookmark
End If
If (Current > 1) And (UP!ID = Current) Then
    ML.Index = "PrimaryKey"
    ML.Seek "=", ParamSet, PStat, Current
    ' only create posterior graph (prior will be a singular spike)
    If Not ML.NoMatch Then _
        Graphs.GraphMarkovPriorPost ML, Graph, "Posterior"
End If
ElseIf NewGenMode = "1" And PStat = FirstPStat _
    And UP!ID <> 10 And UP!ID <> 11 Then
    DS.addnew ' add one segment (with first param state)
    DS!ParamSet = ParamSet
    DS!ParamState = PStat
    DS!UpdatePhase = UP!ID
    DS!Endpos = 0
    If UP!ID > 1 Then DS!Status = "Posterior"
    DS.Update
ElseIf NewGenMode = "2" And PStat = FirstPStat _
    And UP!ID <> 10 And UP!ID <> 11 Then
    ' add one segment (with first param state)
    If UP!ID = 1 Then
        SDS.addnew
        SDS!ParamSet = ParamSet
        SDS!ParamState = PStat
        SDS!Order = 1
        SDS.Update
        SDS.Index = "ParamSet"
        SDS.Seek "=", ParamSet, PStat
    End If
    SDSP.addnew
    SDSP!Segment = SDS!ID
    SDSP!UpdatePhase = UP!ID
    If UP!ID > 1 Then SDSP!Status = "Posterior"
    SDSP.Update
End If
UP.MoveNext
Loop
ParamStates.FindNext "Parameter = " & CStr(Parameter)
Loop
If OldGenMode = "0" Or NewGenMode = "0" Then
    ML.Close
    MTP.Close
    Graph.Close
End If
If OldGenMode = "1" Or NewGenMode = "1" Then
    DS.Close
ElseIf OldGenMode = "2" Or NewGenMode = "2" Then
    SDS.Close
    SDSP.Close
End If
UP.Close
DB.Close
End Sub

Private Sub Parameter_GotFocus()
    If Not IsNull(Me!Parameter) Then Me.GenerationMode.SetFocus
End Sub
```

End Sub

Form_pfTunnelDetailsMain

This form is used to input tunnel detail information.

```
Private Sub changeview_Click()  
' see comments in headings form  
Dim CT As ComboBox  
DoCmd.GoToControl "pfTunnelDetailsDelays"  
RunCommand acCmdSubformDdatasheet  
Set CT = Me!pfTunnelDetailsDelays!Tunnel  
With Me.pfTunnelDetailsDelays  
    If .LinkChildFields = "" Then  
        .LinkChildFields = "Tunnel"  
        .LinkMasterFields = "ID"  
        CT.TabStop = False  
        Me!pfTunnelDetailsDelays!Type.SetFocus  
        CT.Enabled = False  
        CT.Locked = True  
        CT.DefaultValue = Me!ID  
    Else  
        .LinkChildFields = ""  
        .LinkMasterFields = ""  
        CT.TabStop = True  
        CT.Enabled = True  
        CT.Locked = False  
        CT.DefaultValue = ""  
        Me!pfTunnelDetailsDelays!Tunnel.SetFocus  
    End If  
End With  
  
DoCmd.GoToControl "pfTunnelDetailsRepositories"  
RunCommand acCmdSubformDdatasheet  
Set CT = Me!pfTunnelDetailsRepositories!Tunnel  
With Me.pfTunnelDetailsRepositories  
    If .LinkChildFields = "" Then  
        .LinkChildFields = "Tunnel"  
        .LinkMasterFields = "ID"  
        CT.TabStop = False  
        Me!pfTunnelDetailsRepositories!Repository.SetFocus  
        CT.Enabled = False  
        CT.Locked = True  
        CT.DefaultValue = Me!ID  
    Else  
        .LinkChildFields = ""  
        .LinkMasterFields = ""  
        CT.TabStop = True  
        CT.Enabled = True  
        CT.Locked = False  
        CT.DefaultValue = ""  
        Me!pfTunnelDetailsRepositories!Tunnel.SetFocus  
    End If  
End With  
  
DoCmd.GoToControl "pfTunnelDetailsUpDown"  
RunCommand acCmdSubformDdatasheet  
Set CT = Me!pfTunnelDetailsUpDown!Tunnel  
With Me.pfTunnelDetailsUpDown  
    If .LinkChildFields = "" Then  
        .LinkChildFields = "Tunnel"  
        .LinkMasterFields = "ID"  
        CT.TabStop = False  
        Me!pfTunnelDetailsUpDown!Position.SetFocus  
        CT.Enabled = False  
    End If  
End With
```



```
        CT.Locked = True
        CT.DefaultValue = Me!ID
    Else
        .LinkChildFields = ""
        .LinkMasterFields = ""
        CT.TabStop = True
        CT.Enabled = True
        CT.Locked = False
        CT.DefaultValue = ""
        Me!pfTunnelDetailsUpDown!Tunnel.SetFocus
    End If
End With
End Sub
```

Form_pfTunnelNetTunnels

This (sub)form is used to input tunnels.

```
Private Sub Area_NotInList(NewData As String, Response As Integer)
    Dim Areas As Recordset
    Response = acDataErrAdded
    If MsgBox("This area does not yet exist!" & Chr(13) & "Do you want to create it?", _
        vbYesNo, "AREA") = vbYes Then
        Set Areas = CurrentDb().OpenRecordset("tAreas", dbOpenTable)
        Areas.addnew
        Areas!Name = CStr(NewData)
        Areas.Update
        Areas.Close
    Else
        Me!Area = Null
    End If
End Sub

Private Sub Geometry_NotInList(NewData As String, Response As Integer)
    Dim Geometries As Recordset
    Response = acDataErrAdded
    If MsgBox("This geometry does not yet exist!" & Chr(13) & "Do you want to create it?", _
        vbYesNo, "GEOMETRY") = vbYes Then
        Set Geometries = CurrentDb().OpenRecordset("tGeometries", dbOpenTable)
        Geometries.addnew
        Geometries!ID = CStr(NewData)
        Geometries.Update
        Geometries.Close
    Else
        Me!Geometry = Null
    End If
End Sub
```

2976-32