Java-based Modeling of the Actin Polymerization Cycle

by

Catherine Howell

Submitted to the Department of Mechanical Engineering in Partial Fulfillment of the Requirements for the Degree of

Bachelor of Science in Mechanical Engineering

at the

Massachusetts Institute of Technology
[June 2003]
May 15, 2003

Signature of Author.. [illegible]
Department of Mechanical Engineering
May 15, 2003

Certified by.. [illegible]
C. Forbes Dewey, Jr.
Professor of Mechanical Engineering and Bioengineering
Thesis Supervisor

Accepted by..
Ernest Cravalho
Professor of Mechanical Engineering
Chairman, Undergraduate Thesis Committee

JAVA-BASED MODELING OF THE ACTIN POLYMERIZATION CYCLE

by

CATHERINE HOWELL

Submitted to the Department of Mechanical Engineering on May 15, 2003 in Partial Fulfillment of the Requirements for the Degree of Bachelor of Science in Mechanical Engineering

ABSTRACT

This thesis project combines work on two aspects of a project entitled "A Mechanistic Model of the Actin Cycle" [1,2]. This project uses accumulated biochemical research to write a broad mathematical model that describes the effects of regulatory proteins (profilin, beta-thymosin, cofilin, and capping protein) on the steady-state actin cycle. The model necessitates the simultaneous solving of 90-330 differential equations and 51-171 additional equations. One object of this thesis was to prove that this model could be run on JSim, a modeling architecture developed at the University of Washington, to obtain the same results as McGrath et al calculated using Matlab. The theory behind using JSim was that it would be faster and more accessible, since JSim is Java-based and contains all the necessary software to run on various platforms. This project proved that the equations could be run quickly with JSim. It also highlighted some of the drawbacks of JSim, such as heavy demand on the processor. A second object of this thesis was to provide an independent evaluation of the mathematical model and correct mistakes in the original draft. Several errors were uncovered and the corrected results now appear in the paper submitted to The Biophysical Journal [2].

Thesis Supervisor: C. Forbes Dewey

Title: Professor of Mechanical Engineering and Bioengineering

Table of Contents

I. Actin Model

Actin monomers are continuously being polymerized into filaments and depolymerized off of filaments by an ATP-powered cycle. Studies of individual reactions in this cycle have identified the reactant proteins and rate constants. McGrath et al [1] combined these rate constants and concentration variables to write a set of homogeneous equations describing the actin cycle at steady-state. The equations were originally tested with Matlab, providing data that were subsequently used to test the JSim model. In the process, errors in the McGrath equations were found and corrected.

Actin filaments support and generate mechanical stresses at the cell periphery, determining the cell's shape. Proteins that contribute to actin filament polymerization bind actin are modulated by various cell signaling mechanisms. These modulating proteins, such as gelsolin, profiling, and cofilin, serve to modulate the length and location of the filaments and reconfigure them. McGrath et al [1] developed a model of the steady-state actin cycle that identifies the relationships between actin and its key modulating proteins. This model can be used to predict the key parameters describing the state of monomeric and polymeric actin, including filament length, polymer fraction, monomer flux, filament turnover, and a complete nucleotide profile of actin filaments, under the influence of various regulatory proteins. Actin monomers can be bound to ATP or ADP, and can associate themselves with 'barbed' and 'pointed' filament ends (see Fig. 1).

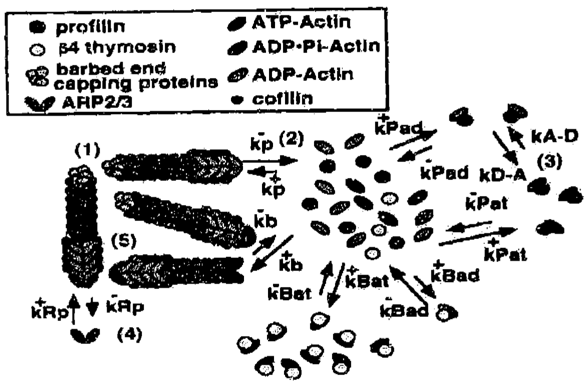


Figure 1: Diagram of Actin Cycle [1]

ATP hydrolizes quickly to ADP·Pi, (ADP plus inorganic phosphate (Pi)) which more slowly releases Pi to become ADP. Pi directly associates with filaments (F-actin) and is assumed not to bind to unpolymerized actin (G-actin), which can bind to ATP or ADP. Because of ATP consumption within filaments, polymerization proceeds to steady-state, at which point the dominant reactions are addition of ATP-bound actin to barbed ends and subtraction of ADP-bound actin from pointed ends. Subunits 'treadmill' from barbed ends to pointed ends. Disassembled ADP-bound actin monomers are recharged with ATP to complete the cycle. The chief regulatory proteins in the actin cycle are capping protein (CP), gelsolin, Arp2/3, cofilin, β-thymosin, and profilin. CP, gelsolin, and Arp2/3 nucleate new actin filaments and sever existing ones; their effects are represented in the equations by the fractions of uncapped filament ends, α and β. CP and gelsolin cap barbed ends, and Arp2/3 caps pointed ends. β-thymosin is the primary G-actin sequestering protein; it maintains G-actin at hundreds of times the critical concentration. Profilin is another G-actin sequestering protein; it accelerates the exchange of ADP for ATP 100 fold. β-thymosin and profilin are assigned the variables B and P in both the manuscript and the MML code. Profilin-bound-G-actin assembles at barbed ends, releasing unbound profilin. Cofilin binds to ADP-bound subunits near filament ends.

Cofilin destabilizes these ends by severing filaments and lowering subunit affinity for filaments, accelerating subunit disassembly at pointed ends. The activity of cofilin is inferred with the rate constant, k^{-}_{pD} (kpdm in MML), which can be varied to signify different concentrations of cofilin. A no-flux condition requires cofilin to similarly destabilize barbed ends, so the rate constant k^{-}_{bD} (kbdm) is varied in tandem with kpdm.

The number of filament subunits of interest varies. At some subunit within a filament the G-actin is entirely ADP bound, and each subsequent subunit is likewise bound until they approach the other end of the filament. The number of subunits necessary to observe is related to cofilin activity, barbed end capping, and filament concentration. The rule of thumb for determining this number 'len' is given in Table 1.

Table 1. Conditions that determine the model parameter 'len'. [1]

1. **If $k^{-}_{pD} < 1$ or $\alpha > 0.1$, then len=20 is sufficient.**
2. **If $k^{-}_{pD} \geq 1$ or $\alpha \leq 0.1$, and $n \leq 0.2$, then len = 50 is sufficient.**
3. **If $k^{-}_{pD} \geq 1$ or $\alpha \leq 0.1$, and $0.2 \leq n \leq 0.4$, then len = 80 is sufficient.**
4. **If $k^{-}_{pD} \geq 1$ or $\alpha \leq 0.1$, and $n > 0.4$, then the filaments are short and essentially have no core and the model would have to deal with solving things in a different way.**

II. JSim as a computational engine for complex biological modeling

This project is part of ongoing work on an information architecture for complex physiological models. JSim is a simulation program created at the University of Washington. It is based on an X-window based simulation interface for UNIX platforms called XSIM. The function of these interfaces is to take a configuration file provided by the user and solve simultaneous equations, providing simulations of scientific models. JSim is Java-based, eliminating the need for any software on the user end. The user downloads JSimStudio, a graphic user interface (GUI), which can be run on MSWindows, Linux, Solaris, or SGI IRIX. JSim software and documentation can be downloaded from the National Science Resource website at UWash (http://nsr.bioeng.washington.edu), along with other biological models written for JSim. With the use of JSim software, users can download and run existing models, or write their own models in Mathematical Modeling Language (MML), a high-level programming language that writes a set of mathematical variables and equations.

JSim uses a five-layer architecture to write, compile, solve, and display results of mathematical models. The layers and their communication structure are described in Table 2 below. The core of the architecture is MML. The user can write any number of standard mathematical equations, with a simple set of syntax rules (see [5] and Appendix B). The .mod code is converted to a Java object when it is loaded by a user interface. MML gives the option of unit conversion, which serves as a dimensional check for variables. The Biological Component Libraries contain models of biological components made in MML that can be referenced for higher-level modeling. The Numerical Planner chooses solvers for the MML equations. The solvers, such as Dopri5, Radau, Kutta-

Merson, Euler, and Runge-Kutta 4, constitute the Numerical Method Library. The user can choose a solver from a menu in the GUI or choose "Auto," allowing the NP to schedule solvers to suit the various needs of the algebraic, ordinary and partial differential equations. The user runs the solution of the model via one of the user interfaces.

Table 2. Major JSim software packages and standards [3]

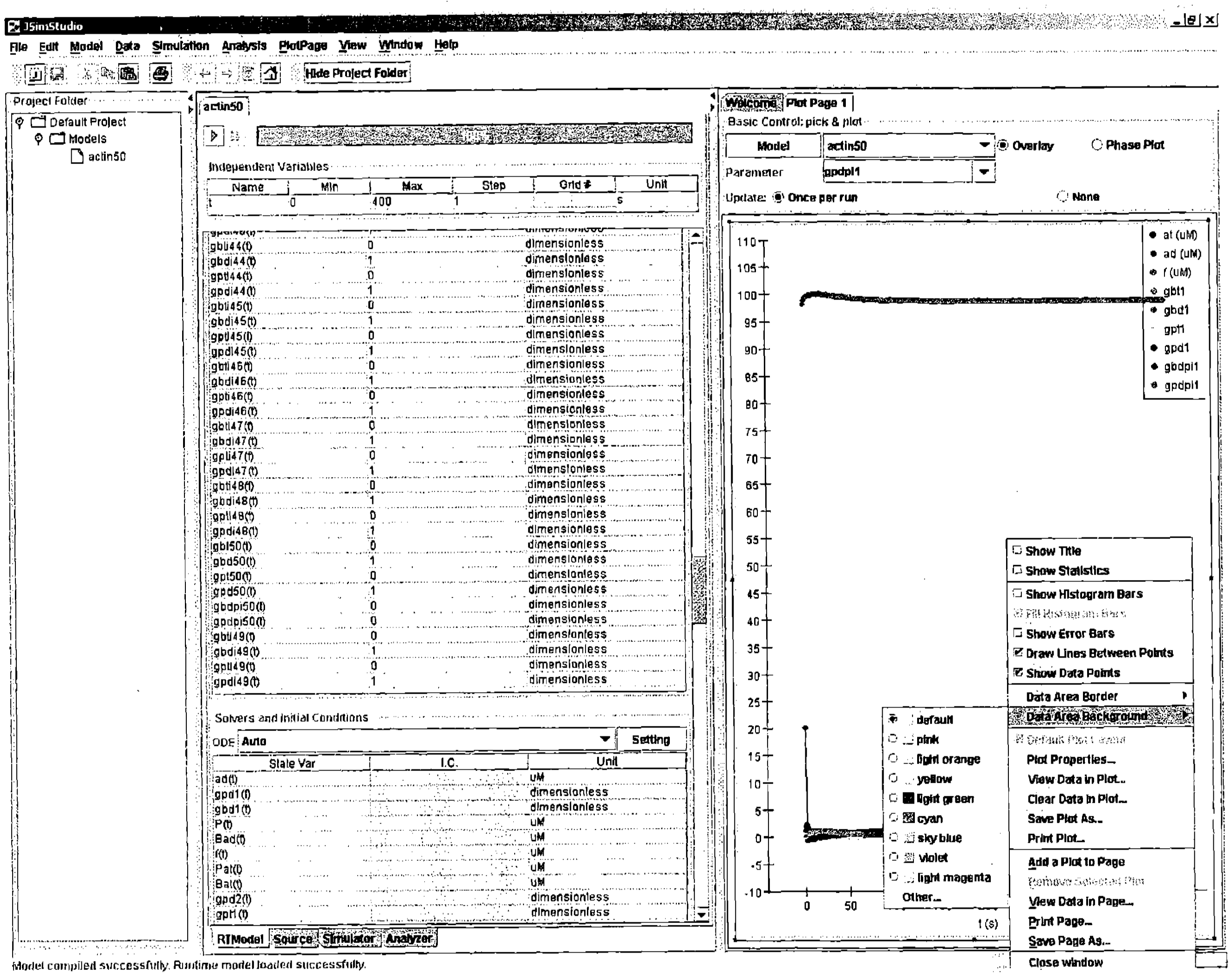
```
Graphical User Interface          (GUI)
  |   |
  | Biological Component Libraries (BCLs)
  |   |
Mathematical Modeling Language      (MML)
  |
Numerical Planner                (NP)
  |
Numerical Method Library           (NML)
```

Writing an MML model requires neither algorithmic programming skill nor familiarity with numerical routines (e.g. ODE solvers). Therefore using MML for scientific models is directly accessible to the scientists studying the physiological phenomena. However, for processes that cannot be written directly in MML, it is possible to write JSim models directly in Java, or to supplement MML models using Java procedures. There are two user interfaces, JSimStudio, the GUI, and jsbatch, a text-based interface for batch processing and debugging. Jsbatch, which writes Java source code from the .mod file and compiles it into a Java class file was not used for this thesis project because its capabilities were not needed, and it has no data analysis tools. [3]

JSim's Graphical User Interface (GUI), JSimStudio, was used to load the .mod file (Appendix C) of the actin model equations and manipulate the solutions. JSimStudio allows runtime control of input variable values, optimization, and function generation. Parameters to be displayed can be chosen before, during, or after runs, and can be displayed in various plots and spreadsheets. Multiple models can be loaded at once and toggled. Every operation, however, increases the work for the processor, and crashes are a common occurrence for models as large as this. More troublesome are the non-fatal errors that occasionally find their way into runs, possibly because of incomplete clearing of data.

III. Project Procedure

I proved that the actin cycle model could be written in JSim and run to obtain the desired results. The form of the homogeneous equations in MML is that of first-order ordinary differential equations; the output of the simulation is the solved variables vs. time. I also wrote another version of the model that uses an additional independent variable. The resulting output variables are given as solved variables vs. time and the independent variable, which can be any of the regulatory proteins. Running the model with more independent variables simultaneously is theoretically feasible, but too computationally intensive for the average single processor.



actin50

Independent Variables

Name	Min	Max	Step	Grid #	Unit
t	0	400	1		s

gbti44(t)	0	dimensionless
gbdi44(t)	1	dimensionless
gpti44(t)	0	dimensionless
gpdi44(t)	1	dimensionless
gbti45(t)	0	dimensionless
gbdi45(t)	1	dimensionless
gpti45(t)	0	dimensionless
gpdi45(t)	1	dimensionless
gbti46(t)	0	dimensionless
gbdi46(t)	1	dimensionless
gpti46(t)	0	dimensionless
gpdi46(t)	1	dimensionless
gbti47(t)	0	dimensionless
gbdi47(t)	1	dimensionless
gpti47(t)	0	dimensionless
gpdi47(t)	1	dimensionless
gbti48(t)	0	dimensionless
gbdi48(t)	1	dimensionless
gpti48(t)	0	dimensionless
gpdi48(t)	1	dimensionless
gbt50(t)	0	dimensionless
gbd50(t)	1	dimensionless
gpt50(t)	0	dimensionless
gpd50(t)	1	dimensionless
gbdpi50(t)	0	dimensionless
gpdpi50(t)	0	dimensionless
gbti49(t)	0	dimensionless
gbdi49(t)	1	dimensionless
gpti49(t)	0	dimensionless
gpdi49(t)	1	dimensionless

Solvers and Initial Conditions

ODE: Auto

State Var	I.C.	Unit
ad(t)		uM
gpd1(t)		dimensionless
gbd1(t)		dimensionless
P(t)		uM
Bad(t)		uM
f(t)		uM
Pat(t)		uM
Bat(t)		uM
gpd2(t)		dimensionless
gpt1(t)		dimensionless

Model: actin50 [✓ Overlay]

Parameter: gpdpl1

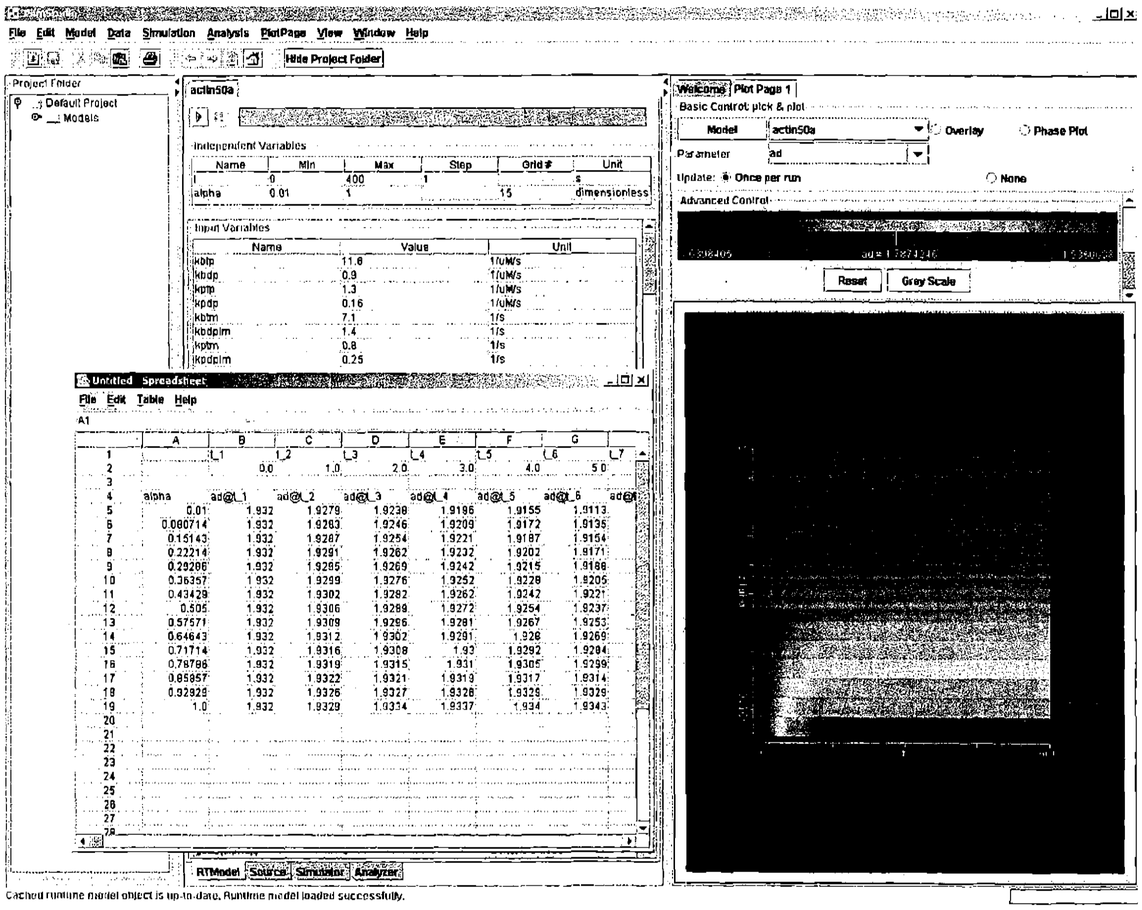
Update: [✓ Once per run]

Model compiled successfully. Runtime model loaded successfully.

Figure 2: JSimStudio running actin50.mod

Since it began as a UROP in June 2002, this project has used several versions of JSim, including Beta1.4.3 for Linux and the latest version, 1.5b5 for Windows, which is not yet available for UNIX systems.

For versions of the actin code utilizing two independent variables, or two-IV, JSimStudio outputs a two-dimensional color plot, which is more easily interpreted by its accompanying spreadsheet (see Figure 3). Each solved parameter reaches a steady-state value with regard to time.



Independent Variables

Name	Min	Max	Step	Grid #	Unit
t	0	400	1		s
alpha	0.01	1		15	dimensionless

Input Variables

Name	Value	Unit
kbtp	11.6	1/uM/s
kbdp	0.9	1/uM/s
kptp	1.3	1/uM/s
kpdp	0.16	1/uM/s
kbtm	7.1	1/s
kbdpim	1.4	1/s
kptm	0.8	1/s
kpdpim	0.25	1/s

Untitled Spreadsheet

File Edit Table Help

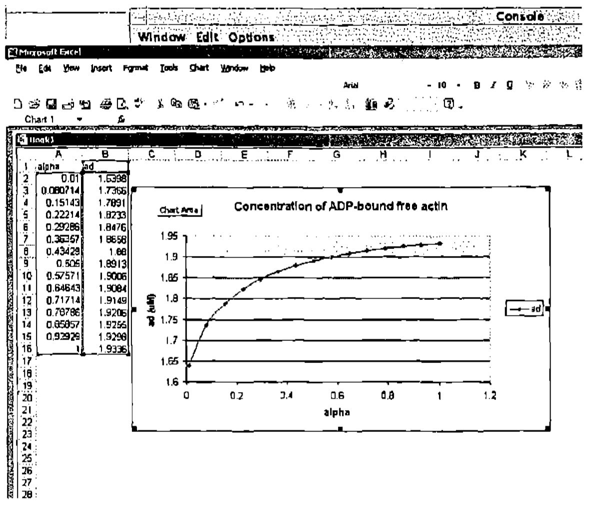
A1

	A	B	C	D	E	F	G
1		t_1	t_2	t_3	t_4	t_5	t_6
2		0.0	1.0	2.0	3.0	4.0	5.0
3							
4	alpha	ad@t_1	ad@t_2	ad@t_3	ad@t_4	ad@t_5	ad@t_6
5	0.01	1.932	1.9279	1.9238	1.9196	1.9155	1.9113
6	0.080714	1.932	1.9283	1.9246	1.9209	1.9172	1.9135
7	0.15143	1.932	1.9287	1.9254	1.9221	1.9187	1.9154
8	0.22214	1.932	1.9291	1.9262	1.9232	1.9202	1.9171
9	0.29286	1.932	1.9295	1.9269	1.9242	1.9215	1.9188
10	0.36357	1.932	1.9299	1.9276	1.9252	1.9228	1.9205
11	0.43429	1.932	1.9302	1.9282	1.9262	1.9242	1.9221
12	0.505	1.932	1.9306	1.9289	1.9272	1.9254	1.9237
13	0.57571	1.932	1.9309	1.9296	1.9281	1.9267	1.9253
14	0.64643	1.932	1.9312	1.9302	1.9291	1.928	1.9269
15	0.71714	1.932	1.9316	1.9308	1.93	1.9292	1.9284
16	0.78786	1.932	1.9319	1.9315	1.931	1.9305	1.9299
17	0.85857	1.932	1.9322	1.9321	1.9319	1.9317	1.9314
18	0.92929	1.932	1.9326	1.9327	1.9328	1.9329	1.9329
19	1.0	1.932	1.9329	1.9334	1.9337	1.934	1.9343

Cached runtime model object is up-to-date. Runtime model loaded successfully.

Figure 3: JSimStudio running actin50a.mod, with spreadsheet

To compare the JSim output to the figures in the original paper [1], I copied a column of data from this spreadsheet, at the steady-state values (usually at about t=400 seconds) and pasted it into a Microsoft Excel spreadsheet. So for instance, Figure 3 above becomes Figure 4, below.


Window Edit Options
Console
Microsoft Excel
File Edit View Insert Format Tools Chart Window Help
Chart 1
Book3
Concentration of ADP-bound free actin
ad (uM)
alpha
ad

	A	B
1	alpha	ad
2	0.01	1.6398
3	0.080714	1.7366
4	0.15143	1.7891
5	0.22214	1.8233
6	0.29286	1.8476
7	0.36357	1.8658
8	0.43429	1.88
9	0.505	1.8913
10	0.57571	1.9006
11	0.64643	1.9084
12	0.71714	1.9149
13	0.78786	1.9206
14	0.85857	1.9256
15	0.92929	1.9298
16	1	1.9336

Figure 4: JSim data copied to MSExcel

Now, the same run can be used to find other parameters. Clear the current data and select at (concentration of ATP-bound free actin) from the parameter menu, view the spreadsheet, and copy-and-paste again into the same Excel spreadsheet (see figure 5).

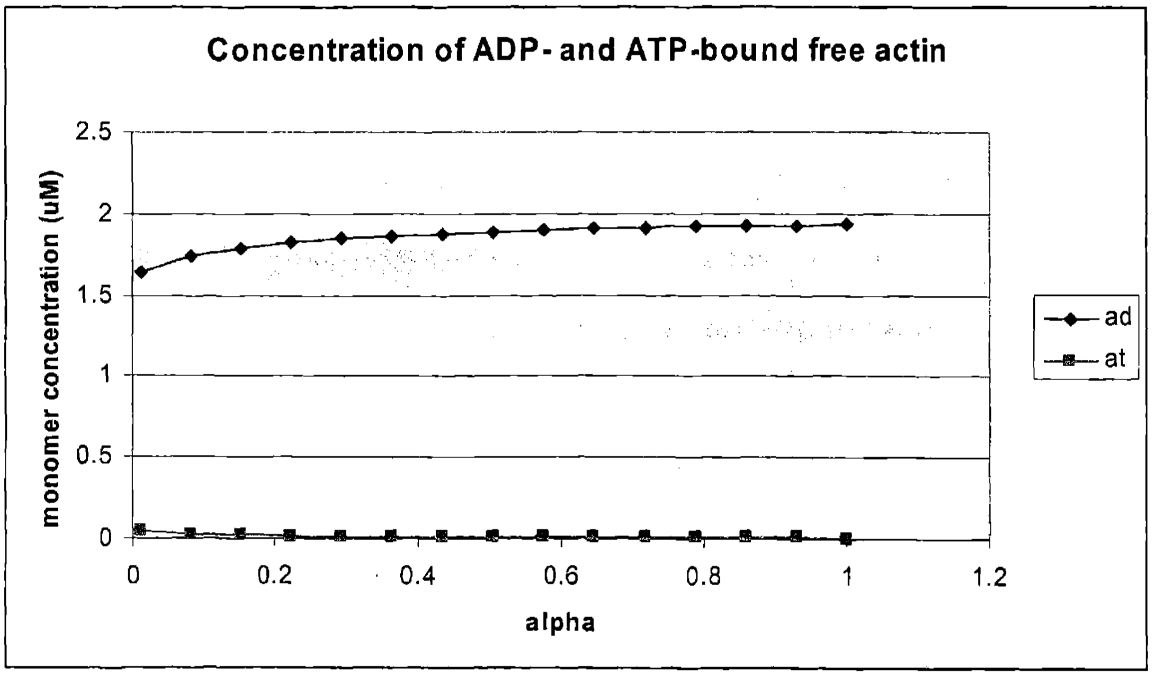


Figure 5: G-actin concentrations, from JSim

In a few minutes the simulation has been performed and compared to the paper, whose comparable figure is Figure 6A on page 52 of the original manuscript [2].

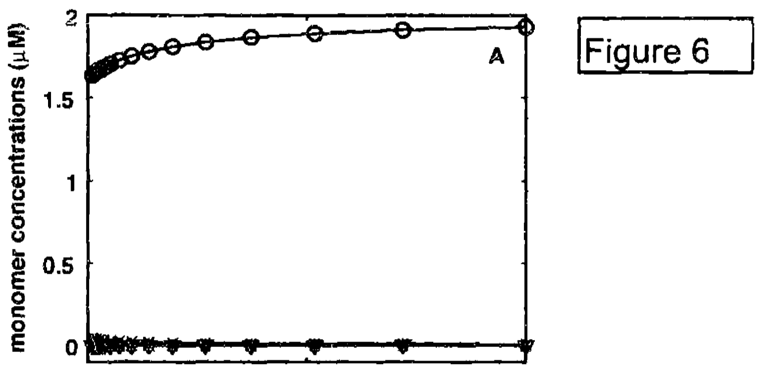


Figure 6: G-actin concentrations, from McGrath [2]

For the runs with a single independent variable, time, JSim treats the other independent variables, for instance alpha, as a single input variable. To reproduce the graphs, the user can plot and copy several output parameters at once, then change the input variable to a new value and repeat the process. The two-I.V. process is quicker but crashes the program much more frequently because several times more computations need to be done at run-time. Also, there are that many times more data points appearing simultaneously in the plot and spreadsheet. See the section on computational requirements.

The g-equations (M10-13 in Appendix B), representing the fraction of subunits at each position on the filaments, in the code (Appendix C) were generated by a Java equation writer program written by Shixin Zhang. The program writes out iterative expressions explicitly. The output of the program was inserted into actin20.mod (Appendix C). MML does not support iterative expressions (e.g. 'for,' 'do,' or 'while' loops).

For different values of the parameters kpdm, alpha, and n (respectively, effective cofilin, fraction of barbed ends capped, and filament concentration) different numbers of equations need to be run. The g-equations are the fractions of barbed and pointed ends of the actin filaments bound to ATP, ADP, and ADP*Pi molecules. The length of actin filament that needs to be considered varies between 20 and 80 subunits. The rule of thumb quoted in Reference 1 and reproduced in Table 1 determines an approximate point at which the filament subunits are entirely ADP-bound. Rather than writing a complex "if" statement, 3 versions of the code were written, for 20, 50, and 80 subunits of interest. The variable len can be used to check if this length is accurate. When in doubt, the user can err on the side of longer codes, but they are more computationally intensive.

A useful tool which is unavailable in the current version of JSim would be logarithmically spaced independent variable strings. Many of the graphs are similar to logarithmic graphs, but JSim only offers linear grids. The user is forced to choose between accuracy at the left part of the graph (small values of independent variables) and efficiency at the right part of the graph (large values of independent variables). Because JSim is intended for use with physiological models, a logarithmic option for independent variable grids makes sense.

IV. Computational Requirements

Running JSimStudio requires significant processing power. The work for this thesis was done on a Pentium 4 1.7 GHz processor with 523 MB of RAM memory. The hard disk used has a 37.2 GB capacity, 6.76 GB free. At the time JSim is launched, existing processes, including the operating system, fill 212 MB of memory, leaving 305 MB of RAM free. A typical one I.V. model run takes about 1 to 2 seconds of CPU time, or 5 to 10 seconds real time and 88 MB of memory(what does this mean?). A typical two I.V. model run takes about 2.2 seconds CPU time, 50-60 seconds real time, and 89 MB of memory. The computation overflows the physical memory and is written to disk as virtual memory. This memory isn't cleared after each calculation and plot is removed, making the system run slower until the user gives up or JSimStudio cancels the run. As long as JSim is running, the virtual memory keeps the old data while the overflowing physical memory slowly clears itself. When JSimStudio is terminated, all the memory is cleared.

V. Conclusions

These results represent a factor of roughly 10 in increased speed over the earlier Matlab program that was used to generate the figures in the original paper. They also independently verified the results of the model before those results were submitted for publication. In addition, the process of writing and running the equations exposed numerous errors, which have been corrected for subsequent versions of the paper. Some errors have not yet been corrected; the JSim results disagree with the paper's results concerning the behavior of profilin. However, the JSim results match very favorably for the effects of cofilin, barbed and pointed filament end exposure, and filament concentration. The actin model is becoming progressively faster and more convenient to manipulate.

Future work on the project involves parallelizing the program to support an extended model several times larger than the current one (~1300 equations, as opposed to ~300 in the current model). The new model will include more key mechanical processes in the cell structure, such as calcium pathways. Work on the actin project will extend for at least another 5 years.

References

1. McGrath, J.L., Bindschadler, M., Osborn, E.A., Hartwig, J.H., and Dewey, C.F. Jr. (July, 2002; revised August, 2002) . A Mechanistic Model of the Actin Cycle. In Preparation for Annal Biomed Eng.

2. Bindschadler, M., Osborn, E.A., Dewey, C.F. Jr, and McGrath, J.L. (April, 2003). A Mechanistic Model of the Actin Cycle. Submitted to Biophysical Journal .

3. JSim Documentation. http://nsr.bioeng.washington.edu/DOC/JSIMDOC; (visited 2003, May 9).

4. JSim Design Goals. (2002, May 31).
URL http://nsr.bioeng.washington.edu/Software/JSIM/JSIM.pdf; (visited 2003, May 9).

5. MML Manual. http://nsr.bioeng.washington.edu/DOC/JSIMDOC/MML_Manual; (visited 2003, May 13).

Appendix A: The Actin Model

Included here are Tables 1-3 from the corrected paper, listing variables, rate constants, and equations [2].

Tables

Table 1: Variable and Parameter Definitions

Variable	Description
at	concentration of ATP-bound free actin
ad	concentration of ADP-bound free actin
f	concentration of actin in filaments
b	concentration of barbed ends
p	concentration of pointed ends
n	concentration of filaments
Atot	total concentration of actin
Ptot	total concentration of profilin
Btot	total concentration of β-thymosin
B	concentration of free β-thymosin
Bat	concentration of ATP-bound monomer complexed with β-thymosin
Bad	concentration of ADP-bound monomer complexed with β-thymosin
P	concentration of free profilin
Pat	concentration of free ATP-bound monomer complexed with Profilin
Pad	concentration of free ADP-bound monomer complexed with Profilin
α	fraction of barbed ends capped ($\alpha = b/n$)
β	fraction of pointed ends capped ($\beta = p/n$)
$g_{bt\,(n)}$	fraction of subunits at position n near filament barbed ends that are bound to ATP
$g_{bdpi\,(n)}$	fraction of subunits at position n near filament barbed ends that are bound to ADP•Pi.
$g_{bd\,(n)}$	fraction of subunits at position n near filament barbed ends that are bound to ADP.
$g_{pt\,(n)}$	fraction of subunits at position n near filament pointed ends that are bound to ATP
$g_{pdpi\,(n)}$	fraction of subunits at position n near filament pointed ends that are bound to ADP•Pi
$g_{pd\,(n)}$	fraction of subunits at position n near filament pointed ends that are bound to ADP
Lavg	average length of filaments
τ	average lifetime of filaments
PF	fraction of actin polymerized
TR	turnover rate of filaments ($1/\tau$)

Table 2: Rate Constants and Literature Sources

Constant	Description	Value(s)	References
k^+_{bT}	rate of ATP-G-actin association at free barbed ends	11.6 (μM s)$^{-1}$	Pollard, 1986
k^+_{bD}	rate of ADP-G-actin association at free barbed ends	0.9 (μM s)$^{-1}$	Lal et al., 1984
k^+_{pT}	rate of ATP-G-actin association at pointed ends	1.3 (μM s)$^{-1}$	Pollard, 1986
k^+_{pD}	rate of ADP-G-actin association at pointed ends	0.16 (μM s)$^{-1}$	Pollard, 1986
k^-_{bT}	rate of ATP-G-actin disassociation at free barbed ends	7.1 s^{-1}	derived (see text)
$k^-_{bD \bullet Pi}$	rate of ADP•Pi subunit disassociation at barbed ends	1.4 s^{-1}	Pollard, 1986 (see text)
k^-_{bD}	rate of ADP subunit disassociation at barbed ends	1.8 - 45 s^{-1}	Lal *et al.*, 1984; range is derived (see text).
k^-_{pT}	rate of ATP subunit disassociation at pointed ends	0.8 s^{-1}	Pollard, 1986
$k^-_{pD \bullet Pi}$	rate of ADP•Pi subunit disassociation at pointed ends	0.250 s^{-1}	derived (see text)
k^-_{pD}	rate of ADP subunit disassociation at pointed ends	0.32 - 8 s^{-1}	nominal value: derived (see text); range: Carlier *et al.*, 1997
k^-_{PD}	rate of profilin-ADP actin dissociation	0.65 s^{-1}	based on equilibrium constant from Selden *et al.*, 1999
k^+_{PD}	association rate for profilin and ADP actin	1 (μM s)$^{-1}$	assigned
k^-_{PT}	disassociation rate for profilin and ATP actin	0.60 s^{-1}	based on equilibrium constant from Selden *et al.*, 1999
k^+_{PT}	association rate for profilin and ATP actin	1 s^{-1}	assigned
$k_{mD \to T}$	rate of ADP exchange for ATP on free monomer	2.66e-3 s^{-1}	derived from Kinosian *et al.*, 1993
$k_{mT \to D}$	rate of ADP exchange for ATP on free monomer	4.86e-5 s^{-1}	derived from Kinosian *et al.*, 1993
$k_{PD \to T}$	rate of ADP exchange for ATP on profilin	2.66 s^{-1}	derived from Kinosian et al., 1993 and Goldschmidt *et al.*, 1991
$k_{PT \to D}$	rate of ADP exchange for ATP on profilin	4.86 s^{-1}	derived from Kinosian et al., 1993 and Goldschmidt *et al.*, 1991
$k_{bT \to D \bullet Pi}$	rate of ATP hydrolysis near barbed ends	12.3 s^{-1}	Carlier *et al.*, 1987
$k_{pT \to D \bullet Pi}$	rate of ATP hydrolysis near pointed ends	1.3 s^{-1}	Carlier *et al.*, 1987
$k_{D \bullet Pi \to D}$	rate of ADP•Pi conversion to ADP on actin filaments	5.5 x 10^{-3}	Carlier and Pantaloni, 1986
k^-_{BD}	disassociation rate for β4 thymosin and ADP actin	100 (μM s)$^{-1}$	based on equilibrium constant from Carlier *et al.*, 1993
k^+_{BD}	association rate for β4 thymosin and ADP actin	1 s^{-1}	assigned
k^-_{BT}	disassociation rate for β4 thymosin and ATP actin	0.9 (μM s)$^{-1}$	based on equilibrium constant from Kang *et al.*, 1999
k^+_{BT}	association rate for β4 thymosin and ATP actin	1 s^{-1}	assigned

Table 3: Equations

Equation	Description
M1 $0 = \alpha[n](k^{+}_{bT}[Pat + at] - k^{-}_{bT} g_{bT(1)} - k^{-}_{bD\bullet Pi} g_{bD\bullet Pi(1)} + k^{+}_{bD}[ad] - k^{-}_{bD} g_{bD(1)}) + \beta(k^{+}_{pT}[at] - k^{-}_{pT} g_{pT(1)} - k^{-}_{pD\bullet Pi} g_{pD\bullet Pi(1)} - k^{+}_{pD}[ad + Pad] - k^{-}_{pD} g_{pD(1)})$	no net assembly of filaments
M2 $0 = k_{at \to d}[at] - k_{ad \to t}[ad] + \beta[n](k^{-}_{pD} g_{pD(1)} + k^{-}_{pD\bullet Pi} g_{pD\bullet Pi(1)} - k^{+}_{pD}[ad]) + \alpha[n](k^{-}_{bD} g_{bD(1)} + k^{-}_{bD\bullet Pi} g_{bD\bullet Pi(1)} - k^{+}_{bD}[ad]) + k^{-}_{PD}[Pad] - k^{+}_{PD}[P][ad] + k^{-}_{BD}[Bad] - k^{+}_{BD}[B][ad]$	no net formation of ADP-G-actin
M3 $A_{tot} = [at] + [ad] + [f] + [Pat] + [Pad] + [Bat] + [Bad]$	conservation of actin
M4 (barbed: $e = b$; $\gamma = \alpha$, $\zeta = 1$) M5 (pointed: $e = p$; $\gamma = \beta$; $\zeta = 0$) $0 = \gamma(k^{+}_{eT}[at + \zeta Pat](1 - g_{eT(1)}) - k^{-}_{eT} g_{eT(1)}(1 - g_{eT(2)}) - k^{+}_{eD}[ad + \zeta Pad] g_{eT(1)} + k^{-}_{eD} g_{eD(1)} g_{eT(2)} + k^{-}_{eD\bullet Pi} g_{eD\bullet Pi(1)} g_{eT(2)}) - k_{eT \to D\bullet Pi} g_{eT(1)}(1 - g_{eT(2)})$	no net formation of ATP-bound termini
M6 (barbed $e = b$; $\gamma = \alpha$; $\zeta = 1$) M7 (pointed $e = p$; $\gamma = \beta$; $\zeta = 0$) $0 = \gamma(k^{+}_{eD}[ad + \zeta Pad](1 - g_{eD(1)}) - k^{-}_{eD} g_{eD(1)}(1 - g_{eD(2)}) - k^{+}_{eT}[at + \zeta Pat] g_{eD(1)} + k^{-}_{eT} g_{eT(1)} g_{eD(2)} + k^{-}_{eD\bullet Pi} g_{eD\bullet Pi(1)} g_{eD(2)}) + k_{D\bullet Pi \to D} g_{eD\bullet Pi(1)}$	no net formation of ADP-bound termini
M8 (barbed $e = b$) M9 (pointed $e = p$) $1 = g_{eD\bullet Pi(1)} + g_{eD(1)} + g_{eT(1)}$	terminal subunits bound to 1 of 3 nucleotides
M10 (barbed $e = b$; $\gamma = \alpha$; $\zeta = 1$; $e' = p$) M11 (pointed $e = p$; $\gamma = \beta$; $\zeta = 0$, $e' = p$) $0 = \gamma(k^{-}_{eT} g_{eT(1)} + k^{-}_{eD} g_{eD(1)} + k^{-}_{eD\bullet Pi} g_{eD\bullet Pi(1)})(g_{eT(i+1)} - g_{eT(i)}) + \gamma(k^{+}_{eT}[at + \zeta Pat] + k^{+}_{eD}[ad + \zeta Pad])(g_{eT(i-1)} - g_{eT(i)}) - k_{eT \to D\bullet Pi} g_{eT(i)}(1 - g_{e'T(i+1)}) - k_{eT \to D\bullet Pi} g_{eT(i)}(1 - g_{eT(i-1)})$	no production of ATP at *i*th subunit
M12 (barbed $e = b$; $\gamma = \alpha$; $\zeta = 1$) M13 (pointed $e = p$; $\gamma = \beta$; $\zeta = 0$) $0 = \gamma(k^{-}_{eT} g_{eT(1)} + k^{-}_{eD} g_{eD(1)} + k^{-}_{eD\bullet Pi} g_{eD\bullet Pi(1)})(g_{eD(i+1)} - g_{eD(i)}) + \gamma(k^{+}_{eT}[at + \zeta Pat] + k^{+}_{eD}[ad + \zeta Pad])(g_{eD(i-1)} - g_{eD(i)}) + k_{D\bullet Pi \to D} g_{eD\bullet Pi(i)}$	no net production of ADP at *i*th subunit
M14 (barbed $e = b$) M15 (pointed $e = p$) $1 = g_{eD\bullet Pi(i)} + g_{eD(i)} + g_{eT(i)}$	the *i*th position subunits will be bound to 1 of 3 nucleotides
M16 M17 $g_{bD(i^*)} = g_{pD(i^*)}$; $g_{bD\bullet Pi(i^*)} = g_{pD\bullet Pi(i^*)}$	barbed/pointed continuity at i^*
M18 (barbed $e = b$) M19 (pointed $e = p$) $1 = g_{eD(i^*)} + g_{eT(i^*)} + g_{eD\bullet Pi(i^*)}$	i^* subunits bound to 1 of 3 nucleotides
M20 M21 $g_{bD(i^*)}$ and $g_{bD\bullet Pi(i^*)}$ lie on a cubic flanking i^*	profile shape continuity at i^*
M22 $0 = k^{+}_{PT}[at][P] - k^{-}_{PT}[Pat] - k_{P(T \to D)}[Pat] + k_{P(D \to T)}[Pad] - \alpha k^{+}_{bT}[Pat][n]$	no net formation of profilin/ATP-actincomplex
M23 $0 = \alpha k^{+}_{bT}[Pat][n] - k^{+}_{PD}[ad][P] - k^{+}_{PT}[at][P] + k^{-}_{PT}[Pat] + k^{-}_{PD}[Pad] + \alpha k^{+}_{bD}[Pad][n]$	no net formation of free profilin
M24 $P_{tot} = [P] + [Pat] + [Pad]$	conservation of profilin
M25 $0 = k^{-}_{BT}[Bat] - k^{+}_{BT}[at][B]$	β-thymosin and ATP-G-actin equilibrium
M26 $0 = k^{-}_{BD}[Bad] - k^{+}_{BD}[ad][B]$	β-thymosin ADP-G-actin equilibrium
M27 $B_{tot} = [Bat] + [Bad] + [B]$	β-thymosin conservation

Appendix B: MML Overview

The Mathematical Modeling Language (MML) used in JSim is a simple protocol for writing models into JSim. Each program declares a set of mathematical variables and the equations that constrain them. The salient features are domains, declared with

```
realDomain time s;
time.min=0; time.max=100; time.delta=1;
```

for instance. MML gives the option of declaring units for variables, and the additional option of whether to convert units during calculation, causing compile failures when the units do not align. Differential equations are written thus:

dBat/dt = kBtm*Bat – kBtp*at*B

becomes

```
Bat:t=kBtm*Bat – kBtp*at*B;
```

The order of equations is immaterial; an equation cannot reference a variable or domain that has not been declared above it in the code.

Declaration:

```
real ad(t,kpdm) uM
```

means that ad is a real variable whose domains are t and kpdm and whose unit is micromolar (concentration). MML can also write if and when expressions, the latter a necessity for writing initial conditions for differential equations. Upper and lower boundary conditions for differential equations were necessary in older versions of JSim, but are no longer necessary.

[5]

Appendix C: The MML code

This is actin20.mod, the input file taken by JSim to solve. Declaring a second domain, such as alpha, kpdm, or Ptot, would require all variables that depend on that independent variable to include that domain in their declarations, e.g. real gbt1(t,kpdm). Different versions include actin50.mod and actin80.mod, which calculate for 50 and 80 subunits, respectively.

```
JSim v1.1

import nsrunit;
unit conversion on;

math actin{
      realDomain t s; //independent variable, I think
      t.min = 0; t.max = 100; t.delta = 1;
      //realDomain kpdm 1/s; Use this for 2D version
      //kpdm.min = 0.32; kpdm.max = 8; kpdm.ct = 51;
      real len;


      real  kbtp = 11.6 1/uM/s,    //Table 2: Rate Constants p. 23
            kbdp = 0.9 1/uM/s,
            kptp = 1.3 1/uM/s,
            kpdp = 0.16 1/uM/s,
            kbtm = 7.1 1/s,
            kbdpim = 1.4 1/s,
            //kbdm = 1.8 1/s, //nominal value...
            kptm = 0.8 1/s,
            kpdpim = 0.250 1/s,
            //kpdm = 0.32 1/s, //nominal value...
            kBdm = 100 1/s, // page 24
            kBdp = 1 1/uM/s,
            kBtm = 0.9 1/s,
            kBtp= 1 1/uM/s,
            kPdm = 0.65 1/s,
            kPdp = 1 1/uM/s,
            kPtm = 0.60 1/s,
            kPtp  = 1 1/uM/s,
            kRpp = 0.02 uM,
            kmDT = 2.66*10^(-3) 1/s,
            kmTD = 4.86*10^(-5) 1/s,
            kPDT = 2.66 1/s, //page 25
            kPTD = 4.86 1/s,
            kbTDpi = 12.3 1/s,
            kpTDpi = 1.3 1/s,
            kdpid = 5.5*10^(-3) 1/s;
        real kpdm =0.32 1/s;
        real kbdm 1/s;
        kbdm = 5.63*kpdm;
        //variables, Table 1.
        real  at(t) uM, ad(t) uM, f(t) uM;//page 21

            real Ptot= 0 uM;
            real Btot = 0 uM;

        real B(t) uM, Bat(t) uM, Bad(t) uM,
            P(t) uM, Pat(t) uM, Pad(t) uM; //R(t) uM, Rp(t) uM,
            real Lavg(t), q(t) 1/s, tau(t) s, //22
            PF(t), TR(t) 1/s;
        real b = 0.1 uM, p = 0.1 uM, n = 0.1 uM, atot = 100 uM, alpha =
b/n,
        beta=p/n;

        //equations, Table 3
        len = if (kpdm < 1) 20//Rules of thumb
            else
            if (alpha > 0.1) 20
            else
            if (n <= 0.2) 50
            else 80;


real gbt1(t), gbd1(t), gpt1(t), gpd1(t), gbdpi1(t), gpdpi1(t);
real gbti1=0, gbdi1=1, gpti1=0, gpdi1=1;

real gbt2(t), gbd2(t), gpt2(t), gpd2(t), gbdpi2(t), gpdpi2(t);

//M4
gbt1:t = alpha*kbtp*(Pat+at)*(1-gbt1) - alpha*kbtm*gbt1*(1-gbt2)
      - alpha*kbdp*gbt1*(ad+Pad) + alpha*kbdm*gbd1*gbt2
      + alpha*kbdpim*gbdpi1*gbt2 - kbTDpi*gbt1*(1-gbt2);
//M6
gbd1:t = alpha*(kbdp*(ad+Pad)*(1-gbd1) - kbdm*gbd1*(1-gbd2)
      - kbtp*gbd1*(at+Pat) + kbtm*gbt1*gbd2
      +  kbdpim*gbdpi1*gbd2) + kdpid*gbdpi1;
//M8
gbdpi1 = 1 - gbt1 - gbd1;
//M5
gpt1:t = beta*kptp*at*(1-gpt1) - beta*kptm*gpt1*(1-gpt2)
      - beta*kpdp*gpt1*ad + beta*kpdm*gpd1*gpt2
      + beta*kpdpim*gpdpi1*gpt2 - kpTDpi*gpt1*(1-gbt2);
//M7
gpd1:t = beta*(kpdp*ad*(1-gpd1) - kpdm*gpd1*(1-gpd2)
      -kptp*gpd1*at + kptm*gpt1*gpd2
      + kpdpim*gpdpi1*gpd2) + kdpid*gpdpi1;
//M9
gpdpi1 = 1 - gpt1 - gpd1;
real Ab(t)= (kbtm*gbt1 + kbdm*gbd1 + kbdpim*gbdpi1),
Beb(t)=(kbtp*(at+Pat)+kbdp*(ad+Pad)),
      Ap(t)=(kptm*gpt1 + kpdm*gpd1 + kpdpim*gpdpi1),
Bep(t)=(kptp*at+kpdp*ad);
```

```
real gbt3(t), gbd3(t), gpt3(t), gpd3(t), gbdpi3(t), gpdpi3(t);
//M14
gbdpi2 = 1 - gbt2 - gbd2;
//M15
gpdpi2 = 1 - gpt2 - gpd2;
real gbti2(t)=0, gbdi2(t)=1, gpti2(t)=0, gpdi2(t)=1;
when (t=t.min){
         gbt2=gbti2; gbd2= gbdi2;  gpt2 = gpti2; gpd2= gpdi2;
 }
//M10
gbt2:t = alpha*Ab*(gbt3 - gbt2)
        + alpha*Beb*(gbt1 - gbt2)
         - kbTDpi*gbt2*(1-gbt3)
         - kpTDpi*gbt2*(1-gbt1);
//M12
gbd2:t = alpha*Ab*(gbd3 - gbd2) + alpha*(Beb)*(gbd1 - gbd2) +
kdpid*gbdpi2;
//M11
gpt2:t = beta*(Ap*(gpt3 - gpt2)
        + Bep*(gpt1 - gpt2))
         - kpTDpi*gpt2*(1-gpt3)
         - kbTDpi*gpt2*(1-gpt1);
//M13
gpd2:t = beta*(-gpd2*(kptp*at+kpdp*ad)*(1-gpd1)
         + gpd1*(kptp*at+kpdp*ad)*(1-gpd2)
         + gpd3*(kptm*gpt1+kpdm*gpd1 + kpdpim*gpdpi1)*(1-gpd2)
         - gpd2*(kptm*gpt1 + kpdm*gpd1+kpdpim*gpdpi1)*(1-gpd3))
         + kdpid*gpdpi2;
real gbt4(t), gbd4(t), gpt4(t), gpd4(t), gbdpi4(t), gpdpi4(t);
gbdpi3 = 1 - gbt3 - gbd3;
gpdpi3 = 1 - gpt3 - gpd3;
real gbti3(t)=0, gbdi3(t)=1, gpti3(t)=0, gpdi3(t)=1;
when (t=t.min){
         gbt3=gbti3; gbd3= gbdi3;  gpt3 = gpti3; gpd3= gpdi3;
 }
gbt3:t = alpha*Ab*(gbt4 - gbt3)
        + alpha*Beb*(gbt2 - gbt3)
         - kbTDpi*gbt3*(1-gbt4)
         - kpTDpi*gbt3*(1-gbt2);
gbd3:t = alpha*Ab*(gbd4 - gbd3) + alpha*(Beb)*(gbd2 - gbd3) +
kdpid*gbdpi3;
gpt3:t = beta*(Ap*(gpt4 - gpt3)
        + Bep*(gpt2 - gpt3))
         - kpTDpi*gpt3*(1-gpt4)
         - kbTDpi*gpt3*(1-gpt2);
gpd3:t = beta*(-gpd3*(kptp*at+kpdp*ad)*(1-gpd2)
         + gpd2*(kptp*at+kpdp*ad)*(1-gpd3)
         + gpd4*(kptm*gpt1+kpdm*gpd1 + kpdpim*gpdpi1)*(1-gpd3)
         - gpd3*(kptm*gpt1 + kpdm*gpd1+kpdpim*gpdpi1)*(1-gpd4))
         + kdpid*gpdpi3;
real gbt5(t), gbd5(t), gpt5(t), gpd5(t), gbdpi5(t), gpdpi5(t);
gbdpi4 = 1 - gbt4 - gbd4;
gpdpi4 = 1 - gpt4 - gpd4;
real gbti4(t)=0, gbdi4(t)=1, gpti4(t)=0, gpdi4(t)=1;
when (t=t.min){
         gbt4=gbti4; gbd4= gbdi4;  gpt4 = gpti4; gpd4= gpdi4;
 }
```

```
gbt4:t = alpha*Ab*(gbt5 - gbt4)
        + alpha*Beb*(gbt3 - gbt4)
         - kbTDpi*gbt4*(1-gbt5)
         - kpTDpi*gbt4*(1-gbt3);
gbd4:t = alpha*Ab*(gbd5 - gbd4) + alpha*(Beb)*(gbd3 - gbd4) +
kdpid*gbdpi4;
gpt4:t = beta*(Ap*(gpt5 - gpt4)
        + Bep*(gpt3 - gpt4))
         - kpTDpi*gpt4*(1-gpt5)
         - kbTDpi*gpt4*(1-gpt3);
gpd4:t = beta*(-gpd4*(kptp*at+kpdp*ad)*(1-gpd3)
         + gpd3*(kptp*at+kpdp*ad)*(1-gpd4)
         + gpd5*(kptm*gpt1+kpdm*gpd1 + kpdpim*gpdpi1)*(1-gpd4)
         - gpd4*(kptm*gpt1 + kpdm*gpd1+kpdpim*gpdpi1)*(1-gpd5))
         + kdpid*gpdpi4;
real gbt6(t), gbd6(t), gpt6(t), gpd6(t), gbdpi6(t), gpdpi6(t);
gbdpi5 = 1 - gbt5 - gbd5;
gpdpi5 = 1 - gpt5 - gpd5;
real gbti5(t)=0, gbdi5(t)=1, gpti5(t)=0, gpdi5(t)=1;
when (t=t.min){
         gbt5=gbti5; gbd5= gbdi5;  gpt5 = gpti5; gpd5= gpdi5;
 }
gbt5:t = alpha*Ab*(gbt6 - gbt5)
        + alpha*Beb*(gbt4 - gbt5)
         - kbTDpi*gbt5*(1-gbt6)
         - kpTDpi*gbt5*(1-gbt4);
gbd5:t = alpha*Ab*(gbd6 - gbd5) + alpha*(Beb)*(gbd4 - gbd5) +
kdpid*gbdpi5;
gpt5:t = beta*(Ap*(gpt6 - gpt5)
        + Bep*(gpt4 - gpt5))
         - kpTDpi*gpt5*(1-gpt6)
         - kbTDpi*gpt5*(1-gpt4);
gpd5:t = beta*(-gpd5*(kptp*at+kpdp*ad)*(1-gpd4)
         + gpd4*(kptp*at+kpdp*ad)*(1-gpd5)
         + gpd6*(kptm*gpt1+kpdm*gpd1 + kpdpim*gpdpi1)*(1-gpd5)
         - gpd5*(kptm*gpt1 + kpdm*gpd1+kpdpim*gpdpi1)*(1-gpd6))
         + kdpid*gpdpi5;
real gbt7(t), gbd7(t), gpt7(t), gpd7(t), gbdpi7(t), gpdpi7(t);
gbdpi6 = 1 - gbt6 - gbd6;
gpdpi6 = 1 - gpt6 - gpd6;
real gbti6(t)=0, gbdi6(t)=1, gpti6(t)=0, gpdi6(t)=1;
when (t=t.min){
         gbt6=gbti6; gbd6= gbdi6;  gpt6 = gpti6; gpd6= gpdi6;
 }
gbt6:t = alpha*Ab*(gbt7 - gbt6)
        + alpha*Beb*(gbt5 - gbt6)
         - kbTDpi*gbt6*(1-gbt7)
         - kpTDpi*gbt6*(1-gbt5);
gbd6:t = alpha*Ab*(gbd7 - gbd6) + alpha*(Beb)*(gbd5 - gbd6) +
kdpid*gbdpi6;
gpt6:t = beta*(Ap*(gpt7 - gpt6)
        + Bep*(gpt5 - gpt6))
         - kpTDpi*gpt6*(1-gpt7)
         - kbTDpi*gpt6*(1-gpt5);
gpd6:t = beta*(-gpd6*(kptp*at+kpdp*ad)*(1-gpd5)
         + gpd5*(kptp*at+kpdp*ad)*(1-gpd6)
         + gpd7*(kptm*gpt1+kpdm*gpd1 + kpdpim*gpdpi1)*(1-gpd6)
         - gpd6*(kptm*gpt1 + kpdm*gpd1+kpdpim*gpdpi1)*(1-gpd7))
         + kdpid*gpdpi6;
real gbt8(t), gbd8(t), gpt8(t), gpd8(t), gbdpi8(t), gpdpi8(t);
gbdpi7 = 1 - gbt7 - gbd7;
gpdpi7 = 1 - gpt7 - gpd7;
real gbti7(t)=0, gbdi7(t)=1, gpti7(t)=0, gpdi7(t)=1;
when (t=t.min){
         gbt7=gbti7; gbd7= gbdi7;  gpt7 = gpti7; gpd7= gpdi7;
 }
gbt7:t = alpha*Ab*(gbt8 - gbt7)
        + alpha*Beb*(gbt6 - gbt7)
         - kbTDpi*gbt7*(1-gbt8)
         - kpTDpi*gbt7*(1-gbt6);
gbd7:t = alpha*Ab*(gbd8 - gbd7) + alpha*(Beb)*(gbd6 - gbd7) +
kdpid*gbdpi7;
gpt7:t = beta*(Ap*(gpt8 - gpt7)
        + Bep*(gpt6 - gpt7))
         - kpTDpi*gpt7*(1-gpt8)
         - kbTDpi*gpt7*(1-gpt6);
gpd7:t = beta*(-gpd7*(kptp*at+kpdp*ad)*(1-gpd6)
         + gpd6*(kptp*at+kpdp*ad)*(1-gpd7)
         + gpd8*(kptm*gpt1+kpdm*gpd1 + kpdpim*gpdpi1)*(1-gpd7)
         - gpd7*(kptm*gpt1 + kpdm*gpd1+kpdpim*gpdpi1)*(1-gpd8))
         + kdpid*gpdpi7;
real gbt9(t), gbd9(t), gpt9(t), gpd9(t), gbdpi9(t), gpdpi9(t);
gbdpi8 = 1 - gbt8 - gbd8;
gpdpi8 = 1 - gpt8 - gpd8;
real gbti8(t)=0, gbdi8(t)=1, gpti8(t)=0, gpdi8(t)=1;
when (t=t.min){
         gbt8=gbti8; gbd8= gbdi8;  gpt8 = gpti8; gpd8= gpdi8;
 }
gbt8:t = alpha*Ab*(gbt9 - gbt8)
        + alpha*Beb*(gbt7 - gbt8)
         - kbTDpi*gbt8*(1-gbt9)
         - kpTDpi*gbt8*(1-gbt7);
gbd8:t = alpha*Ab*(gbd9 - gbd8) + alpha*(Beb)*(gbd7 - gbd8) +
kdpid*gbdpi8;
gpt8:t = beta*(Ap*(gpt9 - gpt8)
        + Bep*(gpt7 - gpt8))
         - kpTDpi*gpt8*(1-gpt9)
         - kbTDpi*gpt8*(1-gpt7);
gpd8:t = beta*(-gpd8*(kptp*at+kpdp*ad)*(1-gpd7)
         + gpd7*(kptp*at+kpdp*ad)*(1-gpd8)
         + gpd9*(kptm*gpt1+kpdm*gpd1 + kpdpim*gpdpi1)*(1-gpd8)
         - gpd8*(kptm*gpt1 + kpdm*gpd1+kpdpim*gpdpi1)*(1-gpd9))
         + kdpid*gpdpi8;
real gbt10(t), gbd10(t), gpt10(t), gpd10(t), gbdpi10(t), gpdpi10(t);
gbdpi9 = 1 - gbt9 - gbd9;
gpdpi9 = 1 - gpt9 - gpd9;
real gbti9(t)=0, gbdi9(t)=1, gpti9(t)=0, gpdi9(t)=1;
when (t=t.min){
         gbt9=gbti9; gbd9= gbdi9;  gpt9 = gpti9; gpd9= gpdi9;
 }
gbt9:t = alpha*Ab*(gbt10 - gbt9)
        + alpha*Beb*(gbt8 - gbt9)
         - kbTDpi*gbt9*(1-gbt10)
         - kpTDpi*gbt9*(1-gbt8);
```

```
gbd9:t = alpha*Ab*(gbd10 - gbd9) + alpha*(Beb)*(gbd8 - gbd9) +
kdpid*gbdpi9;
gpt9:t = beta*(Ap*(gpt10 - gpt9)
        + Bep*(gpt8 - gpt9))
         - kpTDpi*gpt9*(1-gpt10)
         - kbTDpi*gpt9*(1-gpt8);
gpd9:t = beta*(-gpd9*(kptp*at+kpdp*ad)*(1-gpd8)
         + gpd8*(kptp*at+kpdp*ad)*(1-gpd9)
         + gpd10*(kptm*gpt1+kpdm*gpd1 + kpdpim*gpdpi1)*(1-gpd9)
         - gpd9*(kptm*gpt1 + kpdm*gpd1+kpdpim*gpdpi1)*(1-gpd10))
         + kdpid*gpdpi9;
real gbt11(t), gbd11(t), gpt11(t), gpd11(t), gbdpi11(t), gpdpi11(t);
gbdpi10 = 1 - gbt10 - gbd10;
gpdpi10 = 1 - gpt10 - gpd10;
real gbti10(t)=0, gbdi10(t)=1, gpti10(t)=0, gpdi10(t)=1;
when (t=t.min){
         gbt10=gbti10; gbd10= gbdi10;  gpt10 = gpti10; gpd10= gpdi10;
 }
gbt10:t = alpha*Ab*(gbt11 - gbt10)
        + alpha*Beb*(gbt9 - gbt10)
         - kbTDpi*gbt10*(1-gbt11)
         - kpTDpi*gbt10*(1-gbt9);
gbd10:t = alpha*Ab*(gbd11 - gbd10) + alpha*(Beb)*(gbd9 - gbd10) +
kdpid*gbdpi10;
gpt10:t = beta*(Ap*(gpt11 - gpt10)
        + Bep*(gpt9 - gpt10))
         - kpTDpi*gpt10*(1-gpt11)
         - kbTDpi*gpt10*(1-gpt9);
gpd10:t = beta*(-gpd10*(kptp*at+kpdp*ad)*(1-gpd9)
         + gpd9*(kptp*at+kpdp*ad)*(1-gpd10)
         + gpd11*(kptm*gpt1+kpdm*gpd1 + kpdpim*gpdpi1)*(1-gpd10)
         - gpd10*(kptm*gpt1 + kpdm*gpd1+kpdpim*gpdpi1)*(1-gpd11))
         + kdpid*gpdpi10;
real gbt12(t), gbd12(t), gpt12(t), gpd12(t), gbdpi12(t), gpdpi12(t);
gbdpi11 = 1 - gbt11 - gbd11;
gpdpi11 = 1 - gpt11 - gpd11;
real gbti11(t)=0, gbdi11(t)=1, gpti11(t)=0, gpdi11(t)=1;
when (t=t.min){
         gbt11=gbti11; gbd11= gbdi11;  gpt11 = gpti11; gpd11= gpdi11;
 }
gbt11:t = alpha*Ab*(gbt12 - gbt11)
        + alpha*Beb*(gbt10 - gbt11)
         - kbTDpi*gbt11*(1-gbt12)
         - kpTDpi*gbt11*(1-gbt10);
gbd11:t = alpha*Ab*(gbd12 - gbd11) + alpha*(Beb)*(gbd10 - gbd11) +
kdpid*gbdpi11;
gpt11:t = beta*(Ap*(gpt12 - gpt11)
        + Bep*(gpt10 - gpt11))
         - kpTDpi*gpt11*(1-gpt12)
         - kbTDpi*gpt11*(1-gpt10);
gpd11:t = beta*(-gpd11*(kptp*at+kpdp*ad)*(1-gpd10)
         + gpd10*(kptp*at+kpdp*ad)*(1-gpd11)
         + gpd12*(kptm*gpt1+kpdm*gpd1 + kpdpim*gpdpi1)*(1-gpd11)
         - gpd11*(kptm*gpt1 + kpdm*gpd1+kpdpim*gpdpi1)*(1-gpd12))
         + kdpid*gpdpi11;
real gbt13(t), gbd13(t), gpt13(t), gpd13(t), gbdpi13(t), gpdpi13(t);
gbdpi12 = 1 - gbt12 - gbd12;
```

```
gpdpi12 = 1 - gpt12 - gpd12;
real gbti12(t)=0, gbdi12(t)=1, gpti12(t)=0, gpdi12(t)=1;
when (t=t.min){
         gbt12=gbti12; gbd12= gbdi12;  gpt12 = gpti12; gpd12= gpdi12;
 }
gbt12:t = alpha*Ab*(gbt13 - gbt12)
        + alpha*Beb*(gbt11 - gbt12)
         - kbTDpi*gbt12*(1-gbt13)
         - kpTDpi*gbt12*(1-gbt11);
gbd12:t = alpha*Ab*(gbd13 - gbd12) + alpha*(Beb)*(gbd11 - gbd12) +
kdpid*gbdpi12;
gpt12:t = beta*(Ap*(gpt13 - gpt12)
        + Bep*(gpt11 - gpt12))
         - kpTDpi*gpt12*(1-gpt13)
         - kbTDpi*gpt12*(1-gpt11);
gpd12:t = beta*(-gpd12*(kptp*at+kpdp*ad)*(1-gpd11)
         + gpd11*(kptp*at+kpdp*ad)*(1-gpd12)
         + gpd13*(kptm*gpt1+kpdm*gpd1 + kpdpim*gpdpi1)*(1-gpd12)
         - gpd12*(kptm*gpt1 + kpdm*gpd1+kpdpim*gpdpi1)*(1-gpd13))
         + kdpid*gpdpi12;
real gbt14(t), gbd14(t), gpt14(t), gpd14(t), gbdpi14(t), gpdpi14(t);
gbdpi13 = 1 - gbt13 - gbd13;
gpdpi13 = 1 - gpt13 - gpd13;
real gbti13(t)=0, gbdi13(t)=1, gpti13(t)=0, gpdi13(t)=1;
when (t=t.min){
         gbt13=gbti13; gbd13= gbdi13;  gpt13 = gpti13; gpd13= gpdi13;
 }
gbt13:t = alpha*Ab*(gbt14 - gbt13)
        + alpha*Beb*(gbt12 - gbt13)
         - kbTDpi*gbt13*(1-gbt14)
         - kpTDpi*gbt13*(1-gbt12);
gbd13:t = alpha*Ab*(gbd14 - gbd13) + alpha*(Beb)*(gbd12 - gbd13) +
kdpid*gbdpi13;
gpt13:t = beta*(Ap*(gpt14 - gpt13)
        + Bep*(gpt12 - gpt13))
         - kpTDpi*gpt13*(1-gpt14)
         - kbTDpi*gpt13*(1-gpt12);
gpd13:t = beta*(-gpd13*(kptp*at+kpdp*ad)*(1-gpd12)
         + gpd12*(kptp*at+kpdp*ad)*(1-gpd13)
         + gpd14*(kptm*gpt1+kpdm*gpd1 + kpdpim*gpdpi1)*(1-gpd13)
         - gpd13*(kptm*gpt1 + kpdm*gpd1+kpdpim*gpdpi1)*(1-gpd14))
         + kdpid*gpdpi13;
real gbt15(t), gbd15(t), gpt15(t), gpd15(t), gbdpi15(t), gpdpi15(t);
gbdpi14 = 1 - gbt14 - gbd14;
gpdpi14 = 1 - gpt14 - gpd14;
real gbti14(t)=0, gbdi14(t)=1, gpti14(t)=0, gpdi14(t)=1;
when (t=t.min){
         gbt14=gbti14; gbd14= gbdi14;  gpt14 = gpti14; gpd14= gpdi14;
 }
gbt14:t = alpha*Ab*(gbt15 - gbt14)
        + alpha*Beb*(gbt13 - gbt14)
         - kbTDpi*gbt14*(1-gbt15)
         - kpTDpi*gbt14*(1-gbt13);
gbd14:t = alpha*Ab*(gbd15 - gbd14) + alpha*(Beb)*(gbd13 - gbd14) +
kdpid*gbdpi14;
gpt14:t = beta*(Ap*(gpt15 - gpt14)
        + Bep*(gpt13 - gpt14))
```

```
         - kpTDpi*gpt14*(1-gpt15)
         - kbTDpi*gpt14*(1-gpt13);
gpd14:t = beta*(-gpd14*(kptp*at+kpdp*ad)*(1-gpd13)
         + gpd13*(kptp*at+kpdp*ad)*(1-gpd14)
         + gpd15*(kptm*gpt1+kpdm*gpd1 + kpdpim*gpdpi1)*(1-gpd14)
         - gpd14*(kptm*gpt1 + kpdm*gpd1+kpdpim*gpdpi1)*(1-gpd15))
         + kdpid*gpdpi14;
real gbt16(t), gbd16(t), gpt16(t), gpd16(t), gbdpi16(t), gpdpi16(t);
gbdpi15 = 1 - gbt15 - gbd15;
gpdpi15 = 1 - gpt15 - gpd15;
real gbti15(t)=0, gbdi15(t)=1, gpti15(t)=0, gpdi15(t)=1;
when (t=t.min){
         gbt15=gbti15; gbd15= gbdi15;  gpt15 = gpti15; gpd15= gpdi15;
 }
gbt15:t = alpha*Ab*(gbt16 - gbt15)
        + alpha*Beb*(gbt14 - gbt15)
         - kbTDpi*gbt15*(1-gbt16)
         - kpTDpi*gbt15*(1-gbt14);
gbd15:t = alpha*Ab*(gbd16 - gbd15) + alpha*(Beb)*(gbd14 - gbd15) +
kdpid*gbdpi15;
gpt15:t = beta*(Ap*(gpt16 - gpt15)
        + Bep*(gpt14 - gpt15))
         - kpTDpi*gpt15*(1-gpt16)
         - kbTDpi*gpt15*(1-gpt14);
gpd15:t = beta*(-gpd15*(kptp*at+kpdp*ad)*(1-gpd14)
         + gpd14*(kptp*at+kpdp*ad)*(1-gpd15)
         + gpd16*(kptm*gpt1+kpdm*gpd1 + kpdpim*gpdpi1)*(1-gpd15)
         - gpd15*(kptm*gpt1 + kpdm*gpd1+kpdpim*gpdpi1)*(1-gpd16))
         + kdpid*gpdpi15;
real gbt17(t), gbd17(t), gpt17(t), gpd17(t), gbdpi17(t), gpdpi17(t);
gbdpi16 = 1 - gbt16 - gbd16;
gpdpi16 = 1 - gpt16 - gpd16;
real gbti16(t)=0, gbdi16(t)=1, gpti16(t)=0, gpdi16(t)=1;
when (t=t.min){
         gbt16=gbti16; gbd16= gbdi16;  gpt16 = gpti16; gpd16= gpdi16;
 }
gbt16:t = alpha*Ab*(gbt17 - gbt16)
        + alpha*Beb*(gbt15 - gbt16)
         - kbTDpi*gbt16*(1-gbt17)
         - kpTDpi*gbt16*(1-gbt15);
gbd16:t = alpha*Ab*(gbd17 - gbd16) + alpha*(Beb)*(gbd15 - gbd16) +
kdpid*gbdpi16;
gpt16:t = beta*(Ap*(gpt17 - gpt16)
        + Bep*(gpt15 - gpt16))
         - kpTDpi*gpt16*(1-gpt17)
         - kbTDpi*gpt16*(1-gpt15);
gpd16:t = beta*(-gpd16*(kptp*at+kpdp*ad)*(1-gpd15)
         + gpd15*(kptp*at+kpdp*ad)*(1-gpd16)
         + gpd17*(kptm*gpt1+kpdm*gpd1 + kpdpim*gpdpi1)*(1-gpd16)
         - gpd16*(kptm*gpt1 + kpdm*gpd1+kpdpim*gpdpi1)*(1-gpd17))
         + kdpid*gpdpi16;
real gbt18(t), gbd18(t), gpt18(t), gpd18(t), gbdpi18(t), gpdpi18(t);
gbdpi17 = 1 - gbt17 - gbd17;
gpdpi17 = 1 - gpt17 - gpd17;
real gbti17(t)=0, gbdi17(t)=1, gpti17(t)=0, gpdi17(t)=1;
when (t=t.min){
         gbt17=gbti17; gbd17= gbdi17;  gpt17 = gpti17; gpd17= gpdi17;
 }
gbt17:t = alpha*Ab*(gbt18 - gbt17)
        + alpha*Beb*(gbt16 - gbt17)
         - kbTDpi*gbt17*(1-gbt18)
         - kpTDpi*gbt17*(1-gbt16);
gbd17:t = alpha*Ab*(gbd18 - gbd17) + alpha*(Beb)*(gbd16 - gbd17) +
kdpid*gbdpi17;
gpt17:t = beta*(Ap*(gpt18 - gpt17)
        + Bep*(gpt16 - gpt17))
         - kpTDpi*gpt17*(1-gpt18)
         - kbTDpi*gpt17*(1-gpt16);
gpd17:t = beta*(-gpd17*(kptp*at+kpdp*ad)*(1-gpd16)
         + gpd16*(kptp*at+kpdp*ad)*(1-gpd17)
         + gpd18*(kptm*gpt1+kpdm*gpd1 + kpdpim*gpdpi1)*(1-gpd17)
         - gpd17*(kptm*gpt1 + kpdm*gpd1+kpdpim*gpdpi1)*(1-gpd18))
         + kdpid*gpdpi17;
real gbt19(t), gbd19(t), gpt19(t), gpd19(t), gbdpi19(t), gpdpi19(t);
gbdpi18 = 1 - gbt18 - gbd18;
gpdpi18 = 1 - gpt18 - gpd18;
real gbti18(t)=0, gbdi18(t)=1, gpti18(t)=0, gpdi18(t)=1;
when (t=t.min){
         gbt18=gbti18; gbd18= gbdi18;  gpt18 = gpti18; gpd18= gpdi18;
 }
gbt18:t = alpha*Ab*(gbt19 - gbt18)
        + alpha*Beb*(gbt17 - gbt18)
         - kbTDpi*gbt18*(1-gbt19)
         - kpTDpi*gbt18*(1-gbt17);
gbd18:t = alpha*Ab*(gbd19 - gbd18) + alpha*(Beb)*(gbd17 - gbd18) +
kdpid*gbdpi18;
gpt18:t = beta*(Ap*(gpt19 - gpt18)
        + Bep*(gpt17 - gpt18))
         - kpTDpi*gpt18*(1-gpt19)
         - kbTDpi*gpt18*(1-gpt17);
gpd18:t = beta*(-gpd18*(kptp*at+kpdp*ad)*(1-gpd17)
         + gpd17*(kptp*at+kpdp*ad)*(1-gpd18)
         + gpd19*(kptm*gpt1+kpdm*gpd1 + kpdpim*gpdpi1)*(1-gpd18)
         - gpd18*(kptm*gpt1 + kpdm*gpd1+kpdpim*gpdpi1)*(1-gpd19))
         + kdpid*gpdpi18;
real gbt20(t), gbd20(t), gpt20(t), gpd20(t), gbdpi20(t), gpdpi20(t);
gbdpi19 = 1 - gbt19 - gbd19;
gpdpi19 = 1 - gpt19 - gpd19;
real gbti19(t)=0, gbdi19(t)=1, gpti19(t)=0, gpdi19(t)=1;
when (t=t.min){
         gbt19=gbti19; gbd19= gbdi19;  gpt19 = gpti19; gpd19= gpdi19;
 }
gbt19:t = alpha*Ab*(gbt20 - gbt19)
        + alpha*Beb*(gbt18 - gbt19)
         - kbTDpi*gbt19*(1-gbt20)
         - kpTDpi*gbt19*(1-gbt18);
gbd19:t = alpha*Ab*(gbd20 - gbd19) + alpha*(Beb)*(gbd18 - gbd19) +
kdpid*gbdpi19;
gpt19:t = beta*(Ap*(gpt20 - gpt19)
        + Bep*(gpt18 - gpt19))
         - kpTDpi*gpt19*(1-gpt20)
         - kbTDpi*gpt19*(1-gpt18);
gpd19:t = beta*(-gpd19*(kptp*at+kpdp*ad)*(1-gpd18)
         + gpd18*(kptp*at+kpdp*ad)*(1-gpd19)
```

```
          + gpd20*(kptm*gpt1+kpdm*gpd1 + kpdpim*gpdpi1)*(1-gpd19)
          - gpd19*(kptm*gpt1 + kpdm*gpd1+kpdpim*gpdpi1)*(1-gpd20))
          + kdpid*gpdpi19;

gbdpi20 = 0;
gpdpi20 = 0;
gpt20 = 0;
gpd20 =1;
gbt20 = 0;
gbd20 = 1;
real fin = 98.06 uM, atin = 0.006 uM, adin = 1.932 uM;
real Bin =0 uM, Pin =20 uM, Patin=0 uM, Batin =0 uM, Badin = 0 uM;
when (t=t.min){
f = fin;
ad = adin;
}
//M1
f:t=alpha*(kbtp*(Pat+at)*n - kbtm*n*gbt1 - kbdpim*n*gbdpi1 +
      kbdp*(ad+Pad)*n - kbdm*n*gbd1) + beta*(kptp*at*n - kptm*n*gpt1 -
      kpdpim*n*gpdpi1 + kpdp*(ad)*n - kpdm*n*gpd1);
//M3
at = atot - ad -f - Pat - Pad - Bat - Bad;
//M2
ad:t = kmTD*at - kmDT*ad + beta*n*(kpdm*gpd1 + kpdpim*gpdpi1
      - kpdp*ad) + alpha*n*(kbdm*gbd1 +kbdpim*gbdpi1 - kbdp*(ad+Pad)) +
      kPdm*Pad - kPdp*P*ad +kBdm*Bad - kBdp*B*ad;
//M23
P:t = alpha*kbtp*(Pat+at)*n - kPdp*ad*P - kPtp*at*P + kPtm*Pat +
kPdm*Pad + alpha*kbdp*(Pad+ad)*n;


when(t=t.min){
P = Pin;
Pat=Patin;
Bat=Batin;
Bad=Badin;
}
//M24
Pad = Ptot - Pat - P;
//M22
Pat:t=kPtp*at*P - kPtm*Pat-kPTD*Pat + kPDT*Pad - alpha*kbtp*(Pat+at)*n;
//M27
B=Btot-Bat-Bad;
//M25
Bat:t=kBtm*Bat - kBtp*at*B;
//M26
Bad:t=kBdm*Bad - kBdp*ad*B;
      Lavg = f/n;
      tau = Lavg/q;
      PF = f/atot;
real batp (t) 1/s, badp (t) 1/s, patp (t) 1/s, padp (t) 1/s;
batp = alpha*kbtp*(at + Pat) - alpha*kbtm*gbt1 - alpha*kbdpim*gbdpi1;
badp = alpha*kbdp*(ad + Pad) - alpha*kbdm*gbd1;
patp = beta*(kptp*at - kptm*gpt1 - kpdpim*gpdpi1);
padp = beta*(kpdp*ad - kpdm*gpd1);
q=batp+badp;
TR = q/Lavg;}
```