Blackbox Stencil Interpolation Method for Model Reduction

by

Han Chen

Submitted to the Department of Aeronautics and Astronautics
in partial fulfillment of the requirements for the degree of

Master of Science in Aeronautics and Astronautics

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2012

© Massachusetts Institute of Technology 2012. All rights reserved.

Author .....
Department of Aeronautics and Astronautics
August 10, 2012

Certified by .....
Qiqi Wang
Assistant Professor
Thesis Supervisor

Accepted by .....
Chairman
Chairman, Department Committee on Graduate Theses

Blackbox Stencil Interpolation Method for Model Reduction

by

Han Chen

Submitted to the Department of Aeronautics and Astronautics
on August 10, 2012, in partial fulfillment of the
requirements for the degree of
Master of Science in Aeronautics and Astronautics

Abstract

Model reduction often requires modifications to the simulation code. In many circumstances, developing and maintaining these modifications can be cumbersome. Non-intrusive methods that do not require modification to the source code are often preferred. This thesis proposed a new formulation of machine learning, Black-box Stencil Interpolation Method, for this purpose. It is a non-intrusive, data-oriented method to infer the underlying physics that governs a simulation, which can be combined with conventional intrusive model reduction techniques. This method is tested on several problems to investigate its accuracy, robustness, and applicabilities.

Thesis Supervisor: Qiqi Wang
Title: Assistant Professor

Acknowledgments

Firstly I am grateful to Professor Qiqi Wang, my academic advisor, who shows me to the door of the fantastic world of applied mathematics. His wisdom and kindness helps me a lot in my academic development. I also thank him for his detailed and assiduous revision of this thesis.

I also want to express my thankfulness to Hector Klie, the senior scientist in ConocoPhillips, and Karen Willcox, my another academic advisor in MIT. They provide many important advises to my research.

Finally I must thank my mother Hailing Xiong, for giving me priceless love. And I need to thank my girlfriend Huo Ran for her constant support.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Simulations based on partial differential equations (PDE) are heavily used to facilitate management, optimization, and risk assessment in engineering communities [29] [4]. In many scenarios, simulations based on accurate and complicated physics models are available. However, applications like optimization, uncertainty quantification and inverse design generally entails a large number of simulations, therefore using the PDE-based simulations directly for these applications may be inefficient [20]. For example, oil reservoir simulations are generally performed under uncertainties in the permeability field [25]. In order to assess the effect of the uncertainty to the oil productions, thousands of time-consuming simulations, under different realizations of permeabilities, may be required [30]. This may cost more than days or weeks of computation time even on large clusters [14] [22]. Consequently, there has been an increasing need to develop reduced models that could replace full-fledge simulations [34]. A reduced model, in a nutshell, captures the the behavior of a PDE-based simulator with a smaller and cheaper representation, and achieves computational acceleration over the "full" simulator.

1.1 Motivation of Model Reduction

The idea of model reduction is motivated by the trends in engineering communities towards larger scale simulations. Complex and time-consuming simulations based on sophisticated physics models and numerical schemes are used to obtain accurate simulation results. Despite rapid advances in computer hardware as well as the increasing efficiency of computational software, a single simulation run of PDE-based simulator can still be expensive; this said, in many scenarios just one simulation is far from enough. For example, in uncertainty quantifications, many simulations are required under different inputs or model realizations in order to effectively cover the uncertainty space and to assess the associated statistics. This is also true for other computational intensive tasks like optimization and inverse design. Consequently, if we use large-scale PDE-based simulations directly, these tasks can be too expensive to afford even with the aid of supercomputers. In order to reduce the computational burden, reduced order model (or ROM, model reduction) can be used. Model reduction is a technique of replacing a full model with a much "smaller" one which can still describe the full model behavior in some aspects. It has been successfully applied to many fields like oil reservoir engineering [8] [7] and electric circuit design [3] [13].

Depending on whether constructions of the reduced order model requires knowledge about the governing equations and numerical implementations of the full simulator, current model reduction techniques can be categorized into intrusive and non-intrusive methods [18]. Intrusive ROMs, such as the the Discrete Empirical Interpolation Method (DEIM), are appealing for linear and nonlinear model reductions. However, the scope of their application is limited due to their intrusive nature. For example, intrusive ROMs are not applicable when the source code is unavailable, which is the case for many commercial simulators.

Compared with intrusive ROMs, non-intrusive ROMs have the advantage that no knowledge on governing equations or physics is required. It builds a surrogate model to approximate the input-output response based on some pre-run simulation results known as samplings

or trainings. The samplings are interpolated or regressed to obtain an approximated model which can replace the full model in computational intensive tasks like uncertainty quantification and optimization. The surrogate should be cheaper and faster to run than the full model. However, since no physics is used in non-intrusive ROMs, their accuracy can be worse than intrusive ROMs (which we will discuss later in this chapter) in many cases. Also, non-intrusive methods requires the samplings to effectively cover the input parameter space, including control parameters and uncertain parameters. As the number of the inputs, or the dimensionality of the input space, increases, the number of trainings must increase dramatically to achieve a given accuracy. The so-called "curse of dimensionality" greatly undermines the applicability of non-intrusive ROMs in many real-world problems, where a lot of input parameters are involved.

In Section 1.2, we will explain intrusive model reduction techniques. We will illustrate this technique by a projection method, because it is one of the most popular intrusive model reduction techniques and is especially relevant to our method developed in this thesis. In Section 1.3, we will explain the non-intrusive model reduction by illustrating a technique developed recently in order to give a general flavor of non-intrusive ROMs.

1.2 Intrusive Model Reduction Techniques

We will focus on projection-based intrusive ROMs [21] [33] in this section. This kind of ROMs captures the dynamics of the full simulation by projecting the discretized PDE system, which is generally high-dimensional, to a lower dimensional subspace. For example, consider an nth order linear dynamic system

$$\begin{aligned} \dot{x} &= Ax + Bu \\ y &= Cx \end{aligned}, \tag{1.1}$$

where x is the state vector of size n, u is a force vector of size m, y is the system output vector of size p, and $\cdot$ indicates time derivative. Typically $n \gg m$ and $n \gg p$. A is an n-by-n square matrix, B is n-by-m, and C is p-by-n. The force vector often represents the controls in a control system, the external forces in a mechanical system and the electrical charge in a circuit. The state vector indicates the current status of the system. And the output vector represents the output quantities. Intrusive ROMs assume that Eqn (1.1) is available, i.e. A, B, and C are known matrices. To achieve dimensional reduction and subsequent computational acceleration, the state vector can be approximated in a subspace of $\mathbb{R}^n$

$$x \approx V x_r \,, \tag{1.2}$$

where V is an n-by-n_r (columnwise) orthonormal matrix with $n_r \ll n$. Therefore, Eqn (1.2) attempts to describe the full state vector x with a smaller state vector x_r . In addition to V, we use another orthornormal matrix $\tilde{V}$, which is also an n-by-n_r, to apply from the left to Eqn (1.1). So the reduced-order system reads

$$\begin{aligned} \dot{x}_r &= \tilde{V}^T A V x_r + \tilde{V}^T B u \\ y &= C V x_r \end{aligned} \tag{1.3}$$

or in a compact form

$$\begin{aligned} \dot{x}_r &= A_r x_r + B_r u \\ y &= C_r x_r \end{aligned} \,, \tag{1.4}$$

where A_r is an n_r-by-n_r matrix, B_r is n_r-by-m, and C_r is p-by-n_r. Because A_r, B_r, and C_r can be pre-computed, we can solve for the smaller dimensional system Eqn (1.4) instead of the high-dimensional (n-by-n) system Eqn (1.1) for the computational intensive tasks stated above.

There are several choices for the appropriate V and $\tilde{V}$ [21] [33] [36] [13]. Generally

speaking, we want the approximated state Vx_r, which lies in $range(V)$, a subspace of $\mathbb{R}^n$, to be close to a typical x. To this end, the Proper Orthorgonal Decomposition (POD) gives an "optimal" bases in the sense that [24]

$$\min_V \left\langle |x - Vx_r|_2^2 \right\rangle_x , \quad \text{where } x_r = V^T x \,. \tag{1.5}$$

Here $|\cdot|_2$ indicates the L_2 norm, and $\langle\cdot\rangle_x$ indicates the average over an empirical dataset of x, which is composed of x at different timesteps in various simulations under different controls u. Further, we can adopt the Galerkin projection approach, $V = \tilde{V}$ to construct Eqn (1.4). POD is the technique that we will use later to build our new method in this thesis.

To implement POD, we can stack x's into an empirical dataset matrix called the "snapshot" matrix.

$$S = \begin{pmatrix} \vdots & \vdots & \vdots & & \vdots & \vdots & \vdots \\ x_1^1 & \cdots & x_N^1 & \cdots & x_1^s & \cdots & x_N^s \\ \vdots & \vdots & \vdots & & \vdots & \vdots & \vdots \end{pmatrix} \tag{1.6}$$

Each column of the snapshot matrix is an instance of x, where x_j^i indicates the snapshot vecotor at t_j in the ith simulation. We assume there are N timesteps in each simulation. S is an n - by - $s \times N$ matrix, where n is the dimension of the vector x and $s \times N$ is the size of the empirical dataset. Singular value decompostion is subsequently applied to S

$$S = V_n \Sigma U_s \,, \tag{1.7}$$

where V_n is an n-by-n orthonormal matrix, Σ is an n-by-s diagonal matrix, and U_s is an s-by-s orthonormal matrix. The first n_r columns of V_n forms the optimal basis V [35]. Usually the singular values of the snapshot matrix, i.e. diagonal entries of Σ, decays rapidly. Thus only a small number of basis vectors are required to capture almost all of the energy contained in the state vectors, potentially enabling a significant dimensional reduction with a tolerable accuracy loss.

Another method to obtain V and $\tilde{V}$ is the balanced truncation method [27] [31]. We will only briefly introduce this method.

With any invertible linear transformation

$$x = Tx', \tag{1.8}$$

Eqn (1.1) is transformed into

$$\begin{aligned} \dot{x'} &= T^{-1}ATx' + T^{-1}Bu \\ y &= CTx' \end{aligned} . \tag{1.9}$$

Although looking differently, Eqn (1.1) and Eqn (1.9) describe the same dynamics. A balanced truncation is a method to construct V and $\tilde{V}$ that makes the reduced system Eqn (1.4) independent of any particular transformation T [36]. To build the appropriate projection matrices V and $\tilde{V}$, we must takes into account not only the matrices A and B that describe the state dynamics, but also matrix C that describes the output. For the linear system, we define two matrices

$$\begin{aligned} W_c &= \int_0^\infty e^{At}BB^*e^{A^*t}\,dt \\ W_o &= \int_0^\infty e^{A^*t}C^*Ce^{At}\,dt \end{aligned} \tag{1.10}$$

known as the controllability and the observability matrices. The balanced truncation requires a transformation T such that

$$T^{-1}W_cT^{-*} = T^*W_oT = \Sigma\,,$$

where Σ is a diagonal matrix whose diagonal entries are the Hankel singular values. Only the columns in T corresponding to large Hankel singular values are kept in the truncation,

i.e. we choose

$$V = T(:, 1 : n_r)\,, \tag{1.11}$$

where Matlab notation is used. Also, we adopt the Galerkin approach $\tilde{V} = V$. Similar to POD, we herein obtain a smaller system via Eqn (1.3).

1.3 Non-Intrusive Model Reduction Techniques

A non-intrusive model reduction technique does not require the knowledge of the governing equations or the source code. It views a simulation as a black-box and builds a reduced-order surrogate model to capture the input-output relationship instead of evolving the PDE-based dynamics. To build a surrogate, simulations are firstly performed under varying control and uncertain input parameters in order to explore the input parameter space. This process is known as "training", and the input parameters for training are named as the "cloud of design points" [1].

For example, in oil reservoir simulations, the inputs parameters can be the parameters describing the uncertain permeability field, while the output can be the time-dependent bottom hole pressure and the oil production rate. By performing simulations on a set of "design of experiment points", we may effectively explore the uncertainty of the permeability field and examine statistical quantities of the output, such as the expectation and the variance of the production rate at a given time. Design of experiment techniques including Latin hypercube sampling (LHS), minimum discrepancy sequences (quasi Monte Carlo), and adaptive sparse grid can be used to construct the design points.

A surrogate model interpolates or regresses on the existing design points for the response output. Given a new input, it should be able to deliver a similar response output to the PDE-based full model, with much less computational cost. This input-output relation is

shown as

$$\text{Surrogate}: \ \vec{\xi} \ \rightarrow \ u(\vec{x}, t)\,, \tag{1.12}$$

where ξ indicates the design parameters or the uncertain parameters, u indicates the spatial-temporal-dependent output, $\vec{x}$ indicates space, and t indicates time. When the dimensionality of the output is high, however, the interpolation or regression for the response can still be costly.

A method is recently proposed to reduce the dimension of the output by decomposing the output into the spatial and temporal principal components. Then it builds a surrogate for the decomposition coefficients [1] instead of for the full dimensional output. The key idea is an "ansatz" (1.13)

$$u(x, t; \xi) = p_{ref}(x, t) + \sum_{k} \sum_{m} \alpha_{k,m}(\xi)\phi_k(x)\lambda_m(t)\,. \tag{1.13}$$

The ouput u is split into two parts: $p_{ref}(x, t)$ is a reference output independent of the design parameters, it can be obtained by either solving an auxilliary PDE independent of ξ, or by averaging over the outputs u on the cloud of design points.

This method is demonstrated to work well when the input parameters have a small dimension (the examples' dimensions in reference [1] are either 2 or 5). However, for a fixed number of design of experiment points, the error of the surrogate increases dramatically as ξ's dimensionality increases. In other words, the number of design points required to cover an N-dimensional parameter space increases exponentially with N to achieve a given accuracy of the surrogate.

The challenges in both intrusive ROMs and non-intrusive ROMs motivate us to develop a new method. On one hand, it should be applicable when the source code of the PDE-based full model is unavailable; on the other hand, it should alleviate the suffering from the curse

of dimensionality in the input space.

Chapter 2

Nonlinear Model Reduction

In this section, we consider a nonlinear extension of Eqn (1.1); in other words, a nonlinear vector $F(x)$ is added into the state equation to give

$$\begin{aligned} \frac{d}{dt}x(t) &= Ax(t) + F(x(t)) \\ y &= Cx \end{aligned} \tag{2.1}$$

The direct application of projection-based intrusive ROMs to nonlinear problems does not yield dimensional reduction immediately, because the evaluation of the nonlinear vector F still requires a computational effort of $\mathcal{O}(n)$ [9]: Applying the same POD Galerkin projection as before we obtain a reduced model

$$\frac{d}{dt}x_r(t) = V^T AVx_r(t) + V^T F(Vx_r(t)) \tag{2.2}$$

where V is an n-by-n_r orthonormal matrix. The matrix $V^T AV$ of the linear term can be pre-computed. Therefore, in the equation of the reduced model, the evaluation of $V^T AVx_r(t)$ costs only $\mathcal{O}(n_r^2)$ computational work. However, the evaluation of the nonlinear term $V^T F(Vx_r(t))$ still requires $\mathcal{O}(n)$ nonlinear evaluations for the n entries of the vector F, and $\mathcal{O}(n_r n)$ floating point operations. Therefore the "reduced" model does not actually reduce the computational cost to $\mathcal{O}(n_r^2)$. To address this problem some nonlinear

model reduction methods are proposed. We are going to introduce the Trajectory Piecewise Linear Method (TPWL) and the Discrete Empirical Interpolation Method (DEIM). DEIM will be a building block for our Black-box Stencil Interpolation Method that we will develop in the next chapter.

2.1 Trajectory Piecewise Linear Method

To avoid the costly evaluation of F at every timestep, TPWL seeks to approximate F with a piecewise-linear interpolation based on pre-run simulations. Specifically, the nonlinear term in the discretized ODE can be linearized locally in the state space with the Taylor series expansion. For example, suppose we are solving a nonlinear system

$$\frac{d}{dt}x = F(x)\,, \tag{2.3}$$

where the state vector x has dimension n (without loss of generality we dropped the linear term Ax in Eqn (2.1)). Suppose $x_i \in \mathbb{R}^n$ indicates a possible state point. The linearized system in the neighborhood of x_i reads

$$\frac{d}{dt}x = F(x_i) + A_i(x - x_i)\,, \tag{2.4}$$

where

$$A_i = \left.\frac{\partial F}{\partial x}\right|_{x_i} \tag{2.5}$$

is the Jacobian of F at x_i, and x is the state vector. This linearization is only suitable to describe the system in the neighborhood of x_i. To capture the behavior of the system at a larger state domain, we need more state points $x_1, \cdots, x_i, \cdots, x_s$ to adequately cover the whole space. It is proposed in [32] that we can use a weighted combination of linearized models to approximate the full model Eqn (2.6)

$$\frac{d}{dt}x = \sum_{i=0}^{s} w_i(x)\left(F(x_i) + A_i(x - x_i)\right)\,, \tag{2.6}$$

where

$$\sum_{i=0}^{s} w_i = 1\,. \tag{2.7}$$

$w_i(x)$'s are state-dependent weights that determine the local linear approximation of the state equation. After applying a POD-based approximation

$$x \approx V x_r \tag{2.8}$$

to Eqn(2.6), where x_r is a size n_r vector, and applying the property that

$$V^T V = I_{n_r} \tag{2.9}$$

we get

$$\frac{d}{dt}x_r = \sum_{i=0}^{s} w_i(Vx_r)\left(V^T F(x_i) + V^T A_i V x_r - V^T A_i x_i\right)\,, \tag{2.10}$$

in which $V^T F(x_i)$, $V^T A_i V$, and $V^T A_i x_i$ can be pre-computed to avoid the corresponding run-time computational cost. Generally, a direct evaluation of $w_i(Vx_r)$ involves $\mathcal{O}(n)$ computation. To address this problem, TPWL proposes the weight to be

$$w_i \propto e^{-\beta d_i / \max_j(d_j)} \tag{2.11}$$

where

$$d_i \equiv |x - x_i|_2 = |x_r - x_{ri}|_2\,, \qquad x_{ri} \equiv V^T x_i \tag{2.12}$$

β is a parameter to be tuned. By Eqn (2.12) the evaluation of d_i in $\mathbb{R}^n$ is equivalent to its evaluation in $\mathbb{R}^{n_r}$, which enables the dimensional reduction.

In Eqn (2.10), the locations of x_i's are "designed" by the user and have a direct impact on the quality of the reduced model [32]. For this reason we call x_i's the design points. In each simulation, the evolution of state $x(t)$ will trace a trajectory in the state space. It is found that a good ROM quality can be achieved by choosing the design points x_i's

to spread on the trajectories of all the pre-run "training" simulations. After the training, the piecewise-linear reduced model Eqn (2.10) can be used. Clearly, when a state x is located close enough to a design point x_i, the piecewise-linear approximation will be accurate. However, when an x is far away from any of the design points, the linear assumption no longer holds and the reduced model's performance degrades. Besides, it can be demanding to compute the Jacobian A_i's, the N-by-N matrices, on every x_i. Also, to store $V^T F(x_i)$'s, $V^T A_i V$'s, and $V^T A_i x_i$'s for all the design points x_i's can be challenging for computer memory.

2.2 Discrete Empirical Interpolation Method

Discrete Empirical Interpolation Method, or DEIM, takes a different idea from TPWL: TPWL linearizes the full system and applies projection-based ROMs to the piecewise-linear system; DEIM, instead, retains the nonlinearity of the system but attempts to reduce the number of costly evaluations of the nonlinear vector F. In short, DEIM interpolates for the *entire* nonlinear field $F(x)$ on *every* spatial gridpoints based on the evaluation of $F(x)$ at only *some* (a subset) of the spatial points. Obviously, the subset of spatial points should be representative and informative of the entire spatial field being interpolated. To this end, [2] proposed an interpolation scheme, Empirical Interpolation Method (EIM) that combines POD with a greedy algorithm.

It is shown that EIM, in its discrete version, can be used to obtain dimensional reduction for nonlinear systems [9]. Consider a time dependent ODE resulted from the discretization of a PDE

$$\frac{d}{dt}x(t) = Ax(t) + F(x(t))\,. \tag{2.13}$$

Applying the POD model reduction procedure we get

$$\frac{d}{dt}x_r(t) = V^T AV x_r(t) + V^T F(V x_r(t))\,, \tag{2.14}$$

in which the computational cost still depends on the full dimension n due to the nonlinear term F. To reduce the dimensions involved in evaluating the nonlinear term, DEIM approximates F in a subspace of $\mathbb{R}^n$. Suppose we have some snapshots $\{F_1, \cdots, F_m\}$ of F, which are obtained by some training simulations, we can apply POD to this snapshot matrix of F (similar to Eqn (1.6), but replacing x with F) and get a set of truncated singular vectors

$$U = [u_1, \cdots, u_m], \tag{2.15}$$

where $u_i \in \mathbb{R}^n$ and $m \ll n$. Then a reasonable approximation of F is

$$F \approx Uc \tag{2.16}$$

where $c \in \mathbb{R}^m$. If the singular values of the snapshot matrix of F decays rapidly, then

$$c \approx U^T F \tag{2.17}$$

DEIM interpolates for the nonlinear vector F based on the evaluation of F on a subset of entries. To mathematically formulate this, an index matrix is proposed

$$P = [e_{\zeta_1}, \cdots, e_{\zeta_m}] \in \mathbb{R}^{n \times m}, \tag{2.18}$$

Each e_{ζ_i} is a column vector whose ζ_ith entry is 1 and all other entries are 0. Multiplying P^T to a vector effectively extracts m entries from the vector. We give an example index matrix as below

$$P = \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{pmatrix} \tag{2.19}$$

P in Eqn (2.19) can extract the 3rd and the 5th entries from a length 5 vector.
In order to compute c efficiently, we notice the following relation

$$Uc \approx F \quad \Rightarrow \quad P^T U c \approx P^T F \tag{2.20}$$

If $P^T U$, an m-by-m matrix, is well conditioned, we have

$$c \approx \underbrace{\left(P^T U\right)^{-1}}_{G^{-1}} \underbrace{P^T F}_{f_P}, \tag{2.21}$$

where f_P is a size m vector. The computation of c through Eqn (2.21) is efficient in the sense that it does not entail any $\mathcal{O}(n)$ work: firstly, G^{-1} can be pre-computed; secondly, the n dimensional matrix-vector multiplication $P^T F$ does not actually happen, because we only need to evaluate F at the gridpoints specified by the index matrix (DEIM points) to obtain $P^T F$.

However, Eqn (2.21) does not necessarily imply $Uc \approx F$ unless the interpolation points are carefully selected to minimize the approximation error. To this end, DEIM selects an appropriate subset of gridpoints to evaluate F, i.e. an appropriate index matrix P, such that

$$\max \left\{ \left| U(P^T U)^{-1} P^T F - F \right|_2 \right\} \tag{2.22}$$

is minimized. This is achieved approximatedly by a greedy algorithm shown below [9]

1. $\zeta_1 = arg\max\{|u_1|\}$
2. $U = [u_1], P = [e_{\zeta_1}], \vec{\zeta} = [\zeta_1]$
3. **for** $l = 2$ to m:
4. Solve $(P^T U)c = P^T u_l$ to get c
5. $r = u_l - Uc$

6. $\zeta_l = arg\max\{|r|\}$

7. $U \leftarrow [U,\ u_l],\ P \leftarrow [P,\ e_{\zeta_l}],\ \vec{\zeta} \leftarrow [\vec{\zeta},\ \zeta_l]$

in which u_1 is the first POD mode obtained from F's snapshot matrix, corresponding to the largest singular value. ζ_i is the index of the ith DEIM point. In the lth iteration we try to approximate U_{l+1} by an "optimal" vector in range $(U_{1:l})$ *only* on the DEIM points. The residual between U_{l+1} and the optimal approximated vector is a spatial field indicating the approximation error. At every iteration, the spatial point which has the largest error is selected and added to the DEIM point repository.

As stated in section 2.1, a system state x must lie in the neighborhood of a design point x_i, in order to ensure the quality of a TPWL-based reduced model. DEIM, however, does not require this; in other words, the system state can be away from any state in the training simulations. Besides, DEIM does not require the intensive storage needed by TPWL.

Although DEIM is an intrusive model reduction technique, it is used as a building block for our non-intrusive ROM based on the Black-box Stencil Interpolation Method, which will be discussed in the next chapter.

Chapter 3

Black-box Stencil Interpolation Method

3.1 Motivation

Although conventaional model reduction is demonstrated to be effective for many scenarios, there are two fundamental challenges still unresolved. First, lots of industrial simulators are legacy or proprietary code, which excludes the possibility of applying an intrusive ROM. Also, there are occasions where no simulator is available; only the data from historical records and experimental measurements is at hand. For these problems intrusive methods are therefore not applicable. Second, non-intrusive methods, though enjoying a wider range of applicability, generally suffer in accuracy when the number of the control or uncertain parameters is large.

However, there are many scenarios where the parameters are the same local quantities at different spatial locations and different time. For example, in an oil reservoir field, engineers control the water injection rates at hundreds of water injectors. Although the number of injectors can be large, the governing physics (described by a PDE) for the injectors are the same no matter where the injector is located. This property is named by *physics invariance*.

We will dive into this topic later.

To characterize a spatially varying field (e.g. permeability), we generally need many or even infinite number of parameters. For example, in a discretized PDE simulation where there are N gridpoints, we can use N parameters to describe the permeability field, in which each parameter is the permeability at a given spatial gridpoint. Clearly the reservoir state is be determined by all these N parameters. However, if we look at the discretized PDE at a given gridpoint, or stencil, *only* the parameters at *this* gridpoint and its neighboring points show explicitly in the PDE formulation. Changing the values of the parameters will only change the **local** residual of the governing equation. **This property is called *stencil locality*, and the corresponding parameters are called *local parameters*.**

We find it is possible to take advantage of these properties to fight the "curse of dimensionality" in a non-intrusive setting. Specifically, we propose a new, data-oriented approach: we infer the underlying *local* governing physics, where the data can be from either running simulations or historical records. The inferred physics is then used for model reduction and potentially optimal control under uncertainty.

3.2 New Concepts involved in BSIM

3.2.1 Taking Advantage of Stencil Locality

As we have mentioned before, many controls or uncertainties are *local*, because changing the values of the parameters will only change the local residual of the governing equation. Back to the oil reservoir example, we consider a simple 2D pressure equation derived from Darcy's law and conservation of mass:

$$\nabla \cdot (k(x)\nabla p(x)) = s(x)\,, \tag{3.1}$$

where p is pressure, k is permeability, and s is the spatially distributed injection rate. In oil reservoir s has non-zero values only at the location of injection and production wells, therefore we may write

$$s = \sum_{i=0}^{J} s_i \delta(x - x_i) \tag{3.2}$$

where J is the number of injection and production wells. $s_i,\ i = 0, \cdots, J$ stands for the injection or production rate at the ith well[1]. x_i is the location of the ith well. δ is the Dirac delta function. Suppose we use finite difference to discretize Eqn (3.1), and there are N spatial points in the discretization. To fully describes the uncertainty in $k(x)$ we need N parameters [2]; To explore the injection and production strategies we need J parameters. Therefore, a total of $N+J$ dimensional parameter is needed in order to determine the entire pressure field.

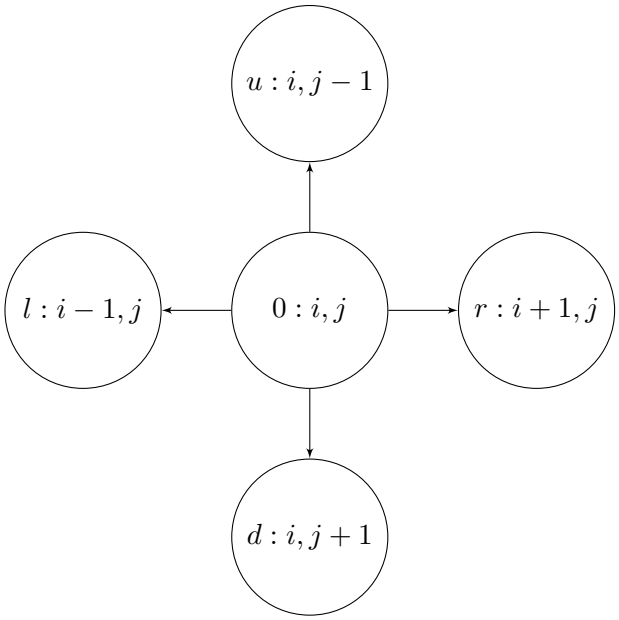
Instead of considering the relation between the $N + J$ dimensional parameter and the entire pressure field, we shift gear to a single spatial gridpoint. Applying a finite difference discretization to Eqn (3.1) we get

$$\frac{1}{\Delta x^2}\left\{\left(\frac{k_u + k_0}{2}\right)(p_u - p_0) + \left(\frac{k_d + k_0}{2}\right)(p_d - p_0) + \left(\frac{k_l + k_0}{2}\right)(p_l - p_0) + \left(\frac{k_r + k_0}{2}\right)(p_r - p_0)\right\} = s_0 \tag{3.3}$$

where $p_\alpha\,,\ \alpha = u, d, l, r, 0$ is the pressure at the corresponding gridpoint in the stencil, $k_\beta\,,\ \beta = u, d, l, r, 0$ is the permeability at the corresponding gridpoint in the stencil.

[1] for injection well $s_i > 0$, for production well $s_i < 0$

[2] By making certain assumptions, we can use Karhunen-Loeve expansion to reduce the number of parameters required to characterize the permeability field. However, the reduced number can still be large, making non-intrusive model reduction difficult [1]. We will not discuss this topic for simplicity.



A stencil in the finite difference discretization.

At each grid point, only 11 quantities are involved, i.e. $k_{u,d,l,r,0}$, $p_{u,d,l,r,0}$, s_0. Since Eqn (3.3) is valid throughout the spatial domain, the same relation between the 11 quantities and the local pressures $p_{u,d,l,r,0}$ holds everywhere inside the spatial domain. The relation between the local quantities is called the *local physics* model. Inferring this 11-dimensional parameter's local physics model from data is much easier than inferring the global relation between the $N + J$ dimensional input parameter and the pressure field.

Generally, after the discretization of a PDE, the numerical solution at a spatial gridpoint is updated only with the information from its stencil neighbors. This argument is valid for both time dependent PDEs solved by time marching and time independent PDEs solved by iterative methods. Therefore, the number of quantities involved in a local physics model can be much smaller than the number of paramters for the entire simulation. Conventionally, a non-intrusive ROM builds up a direct mapping (surrogate) from all the input parameters to the simulation output. We call this surrogate a "global surrogate" because it is used to approximate the behavior of the entire simulation. When the number of parameters is large, e.g. $N + J$ in the pressure equation example, the number of simulations required to build the

surrogate can be prohibitively large (see chapter 1). However, if we try to build a surrogate for the local governing physics that defines the relation between quantities on a stencil, the dimension of this surrogate's input can be significantly lower. Again in the pressure equation example, we can rewrite Eqn (3.3) as

$$p_0 = \frac{\Delta x^2}{\frac{k_u+k_d+k_r+k_r}{2} + 2k_0} \times \left(\frac{p_u}{\Delta x^2}\left(\frac{k_u+k_0}{2}\right) + \frac{p_d}{\Delta x^2}\left(\frac{k_d+k_0}{2}\right) + \frac{p_l}{\Delta x^2}\left(\frac{k_l+k_0}{2}\right) + \frac{p_r}{\Delta x^2}\left(\frac{k_r+k_0}{2}\right) - s_0 \right) \tag{3.4}$$

Eqn (3.4) describes the "local governing physics", it can be viewed as a predicting model for p_0 given inputs $p_{u,d,l,r}$, $k_{u,d,l,r,0}$, and s_0. Therefore, a surrogate to approximate the "exact" local physics Eqn (3.4),

$$p_0 = f(p_u, p_d, p_l, p_r, k_0, k_u, k_d, k_l, k_r, s_0) \tag{3.5}$$

will have 10 dimensional input and 1 dimensional output. Because $10 \ll N + J$, it requires fewer sample to build an accurate surrogate for the local physics than for the entire or global simulation. Besides, we notice that the complexity of the local physics model is independent of the spatial scale of the simulation: For example, the simulation can be performed for a small reservoir on a $100 - by - 100$ mesh, or for a large reservoir on a $1000 - by - 1000$ mesh. The number of parameters needed for describing the permeability field k, and the well injection / production rates, can be much more for the larger simulation, but the local surrogate Eqn (3.5) will have an input dimension of 10 in both cases.

3.2.2 Taking Advantage of Physics Invariance

In PDE-based simulations, there are some physics models associated with every control and uncertain parameter. For example, the Darcy's law and the conservation of mass relates the pressure field with the uncertain permeability. After the discretization of the governing PDE, we define *physics* as the relation beween local quantities in that equation, e.g. Eqn

(3.3) and Eqn (3.4).

Although the permeability field $k(x)$ and the injection rate $s(x)$ can vary with space, the governing physics for them does not change: For example, Eqn(3.1) can be valid throughout the spatial domain in our simulation. Similarly, after spatial discretization, the governing physics Eqn (3.4) is valid at different grid points (i, j) and (i', j'). Also, for a time dependent problem, we expect the physics to be invariant at different time. We call this property the "physics invariance", specifically the "physics invariance under spatial and temporal translation".

In some applications, however, several different physics, or equations, must be applied to different spatial or temporal domain. For example, in the simulation of flow-structure interaction, we need a set of fluid equations to describe the flow and a different set of equations to describe the solid structure. However, our statement of physics invariance is still valid within each subdomain.

In order to build an accurate surrogate, we need sufficient samples, or trainings, to effectively explore the input space of the surrogate. In conventional non-intrusive methods, the sample size is the number of simulations being performed; each simulation is preformed on a different set of input parameters. Therefore, each sample can be expensive as an entire simulation run is required. However, when we shift gear to build the surrogate for the local governing physics instead, we may get thousands of samples from a single simulation thanks to the physics invariance. Because the governing physics is invariant at every spatial and temporal location, every girdpoint at every timestep contributes a sample that helps to construct the physics surrogate. Generally, for a time-dependent simulation with N spatial gridpoints and T timesteps, each simulation may contribute almost $N \times T$ samples (this number is not accurate because of the spatial and temporal boundaries), in sharp contrast to only a single sample in the conventional non-intrusive method. By k simulations, we can

obtain $\mathcal{O}(kNT)$ number of stencil samples.

In fact, in some applications, the sample size $\mathcal{O}(kNT)$ can be overwhelming. When this happens, we may use the samples only from a selected subset of gridpoints and timesteps. The training data selection is a separate topic, and we will not discuss it until the next chapter.

3.3 Surrogate Techniques

3.3.1 Regression with Specific Assumptions

A surrogate model, also known as a response surface model, is a computationally efficient approximate model that mimics the input-output behavior of a system. In our context we are trying to approximate the exact physics model. For example, in the pressure equation example, we are trying to approximate the governing physics Eqn (3.4) with Eqn (3.5) by learning from a set of stencil samples

$$p_{u,d,l,r}\ ,\ K_{0,u,d,l,r}\ , s_0\ \rightarrow\ p_0 \tag{3.6}$$

Techniques for construction of non-intrusive surrogates are numerous [17]. For example, linear regression is a widely used surrogate technique

$$y = \beta^T x + \gamma \tag{3.7}$$

Given the input-output data (x, y) we adjust β and γ to best fit the data by mean square error minimization. However, linear regression is based on a linear assumption: the underlying model that generates the data is assumed to be linear. When the underlying model is nonlinear, linear regression can be a bad choice.

This problem is not only for linear regression, many regression techniques like the log-

normal regression that pre-assumes the form of the underlying model may perform badly when such their assumptions are not appropriate. Generally, for non-intrusive model reduction, we do not have enough knowledge of the physics model in PDE-based simulations, therefore these regression techniques are not suitable.

Another commonly used technique is high order polynomial regression. It does not make specific assumptions to the data and can approximate any smooth function up to arbitrary accuracy. It may be used to build the surrogate for our purpose. However, we did not test it in our work.

3.3.2 Kernel Methods

The previous techniques belong to parametric regression methods. In the linear regression example Eqn (3.7), the relation between the input x and the output y is parameterized by β and γ. During the "learning phase", a set of training data is used to adjust the parameters [5] for best data fitting. Future predictions are solely based on the fit parameters. There is a different family of techniques which keeps and uses the sampled data in every prediction: the kernel methods.

Radial Basis Function Interpolation

A popular kernel method is the Radial Basis Function (RBF) interpolation. An RBF is a symmetric bivariate function defined as

$$\phi(x, x') = \phi(|x - x'|)\,. \tag{3.8}$$

A commonly used RBF is the Gaussian kernel

$$\phi = \exp\left\{-\frac{|x - x'|^2}{2\sigma^2}\right\} \tag{3.9}$$

The weighted sum of multiple radial basis functions is typically used for function approximation. Suppose we have already evaluated a model at design points $x_1, \cdots, x_N$, then the interpolation reads

$$f(x^*) = \sum_{i=1}^{N} w_i \phi(x^*, x_i)\,, \tag{3.10}$$

where w_i are the weights and can be solved by

$$\begin{pmatrix} y(x_1) \\ \vdots \\ y(x_n) \end{pmatrix} = \begin{pmatrix} \phi(x_1, x_1) & \cdots & \phi(x_1, x_n) \\ \vdots & \vdots & \vdots \\ \phi(x_n, x_1) & \cdots & \phi(x_n, x_n) \end{pmatrix} \begin{pmatrix} w_1 \\ \vdots \\ w_n \end{pmatrix} \tag{3.11}$$

Kriging Interpolation

Kriging interpolation is a technique originated from the geostatistical problem of interpolating a random field with a known covariance function. Suppose we have already evaluated a realization of the random field $f(x)$ at some design points $x_1, \cdots, x_n$, then the approximation $\hat{f}(x^*)$ for $f(x^*)$ at a new point x^* will be a weighted combination of $f(x_1), \cdots, f(x_n)$,

$$\begin{aligned} \hat{f}(x^*) &= \mathbb{E}[f(x^*)|f(x_1), \cdots, f(x_n)] \\ &= \sum_{i=1}^{n} w_i(x_i, x^*) f(x_i) \end{aligned} \tag{3.12}$$

Kriging interpolation assumes that the covariance function is available:

$$c(x, x') \equiv Cov(f(x), f(x')) \tag{3.13}$$

In a simple Kriging method, the mean of the random field $\mathbb{E}[f] = 0$. The weights are computed by

$$\begin{pmatrix} c(x_1, x_1) & \cdots & c(x_1, x_n) \\ \vdots & \vdots & \vdots \\ c(x_n, x_1) & \cdots & c(x_n, x_n) \end{pmatrix} \begin{pmatrix} w_1 \\ \vdots \\ w_n \end{pmatrix} = \begin{pmatrix} c(x_1, x') \\ \vdots \\ c(x_n, x') \end{pmatrix} \tag{3.14}$$

It can be verified from Eqn (3.12) and Eqn (3.14) that

$$\hat{f}(x_i) = f(x_i) \qquad \text{for } \forall\ i = 1, \cdots, n \tag{3.15}$$

Therefore it is an interpolation method. However, the performance of simple Kriging depends strongly on the choice of the covariance function. For example, we may choose a Guassian covariance

$$c(x, x') = \exp\left\{-\frac{|x - x'|^2}{2\sigma^2}\right\} \tag{3.16}$$

If σ is too small, the interpolated $\hat{f}(x)$ between two adjacent design points will be close to zero; if σ is too large, Eqn (3.14) will be ill-conditioned. This is illustrated in Fig 3-1.

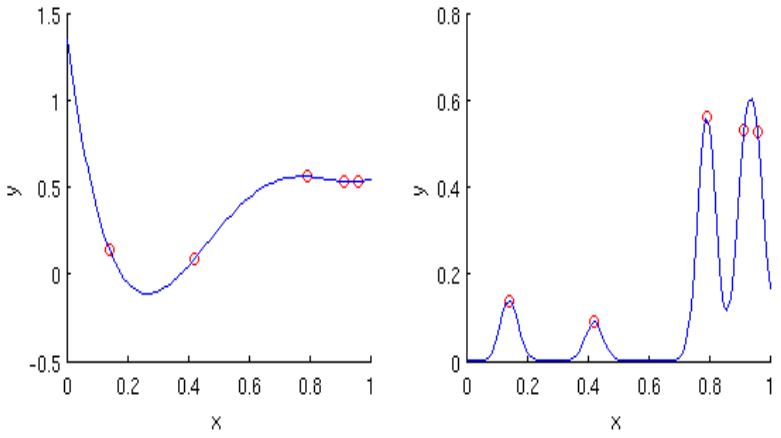


Figure 3-1: the performance of simple Kriging depends on the choice of the covariance function, for example, the correlation length. In the left figure σ can be too large; in the right figure σ can be too small.

Instead of assuming $\mathbb{E}[f] = 0$ as in the simple Kriging interpolation, the ordinary Kriging interpolation applies an unbiased condition by constraining the weights

$$\sum_{i=1}^{n} w_i = 1 \tag{3.17}$$

Combining Eqn (3.14) and Eqn (3.17), we have

$$\begin{pmatrix} c(x_1,x_1) & \cdots & c(x_1,x_n) & 1 \\ \vdots & \vdots & \vdots & 1 \\ c(x_n,x_1) & \cdots & c(x_n,x_n) & 1 \\ 1 & \cdots & 1 & 0 \end{pmatrix} \begin{pmatrix} w_1 \\ \vdots \\ w_n \\ \lambda \end{pmatrix} = \begin{pmatrix} c(x_1,x') \\ \vdots \\ c(x_n,x') \\ 1 \end{pmatrix} \tag{3.18}$$

The ordinary Kriging interpolation is not biased towards 0, however its performance still depends on the covariance, for example σ in the Gaussian kernel. This is shown in Fig 3-2.

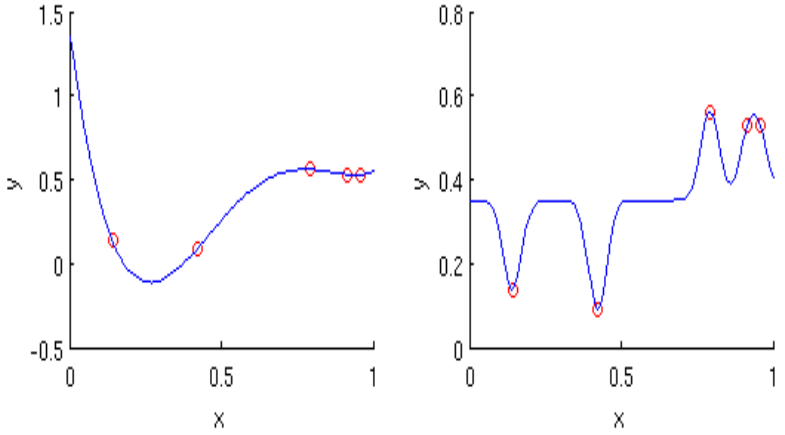


Figure 3-2: Ordinary Kriging is not biased, but the performance depends on the choice of the covariance, especially the correlation length.

There are methods, such as Maximum Likelihood Estimation [28], that can determine the optimal σ. We have not tested them in this research and will not discuss this topic.

Nearest Neighbor Interpolation

Nearest neighbor interpolation approximates $f(x^*)$, where $x^* \in \mathbb{R}^n$ [3], by linearly interpolating x^*'s $n+1$ nearest neighboring sample points. This requires the search for the nearest $n+1$ neighbors first. A naive search algorithm is:

[3] n is the dimension of x

1. Compute $d_i = |x^* - x_i|_2$ for $i = 1, \cdots, S$

2. Sort $d_{1 \cdots S}$ for the $n + 1$ smallest neighbors.

However, it requires $\mathcal{O}(S)$ operations, where S is the number of samples, or design points. Because the sample size is $\mathcal{O}(kNT)$ in our case, this naive neighbor approach algorithm is clearly not suitable. So an efficient algorithm of the nearest $n + 1$ neighbors searching is required for nearest neighbor interpolation.

K-d tree is an efficient algorithm for nearest neighbor search [26]. We do not give the details here, but the conclusion is that the k-d tree algorithm has a complexity of $\mathcal{O}(klogN)$ to search for the nearest k neighbors within N uniformly spaced design points. As we will see, however, in our application the size of the sample points can be huge. Even if the nearest neighbor search scales as $log(N)$ in complexity, N can be so large that the computational cost may still be unaffordable. We will not use this scheme in the development of our method. Another problem of k-d tree is that it requires a uniform distribution of the design points in order to achieve the $\mathcal{O}(klogN)$ complexity. However the stencil data generally do not distribute uniformly in our application. Therefore, $\mathcal{O}(klogN)$ complexity may not be achieved.

Kernel methods do not make specific assumptions to the data, so they have a wide applicability. However, it does not parameterize the function, therefore all samples must be stored and used in every prediction, a burden to both computer memory and computation speed when the number of points is huge.

3.3.3 Neural Networks

We need an interpolation or regression method that both parametric and does not make specific assumptions to the data, or *flexible*. This is because the number of design points in our application can be so large that we cannot store and use all of them for the interpolation

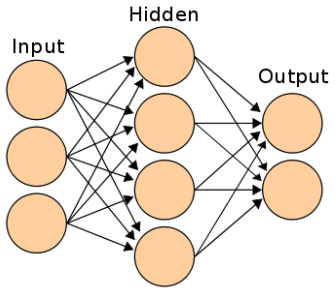


Figure 3-3: Neural network with one hidden layer.

at a new point. We thus need to parameterize the data and represent them in a compact way. Also, we hope that the method is flexible, i.e. can capture many different physical models accurately without losing generality. Therefore we refrain ourselves from making assumptions such as linearity.

A regression method that is especially suitable for our purpose is Artificial Neural Network, or neural network for short. Neural network extracts linear combinations of inputs (known as *features*), then models the output (known as *target*) as a nonlinear function of the *features*. A neural network consists of one input layer, several hidden layers, and one output layer. The structure of a single hidden layer neural network is shown in Fig 3-3.

Each neural network layer consists of several "units". For example, in Fig 3-3 there are 3 units in the input layer, 4 units in the hidden layer, and 2 units in the output layer. In our application we will consider only one output, namely the output is a scalar. A unit takes inputs from the previous layer and feeds its output to the next layer. For example, Fig 3-4 illustrates a typical unit of neural network, which admits 3 inputs x_1, x_2, x_3 and gives 1 output y. The procedure to obtain y from the inputs is as follows: Firstly, the inputs is rescaled to $\hat{x}_1, \hat{x}_2, \hat{x}_3$ in the range of $[-1, 1]$. Then we take a weighted linear combination of $\hat{x}_1, \hat{x}_2, \hat{x}_3$ together with a bias b, and feed the intermediate result to a nonlinear function f to get $\hat{y}$. Generally $\hat{y}$ is a normalized output within $[-1, 1]$. Finally $\hat{y}$ is rescaled to obtain

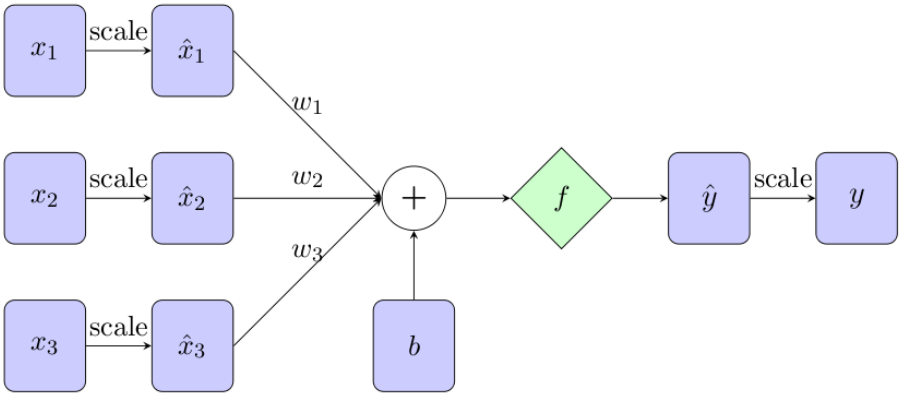


Figure 3-4: A unit in neural network.

the final output y.

A popular choice for the nonlinear function f is the sigmoid function.

$$f(x) = \frac{1}{1 + e^{-x}} \tag{3.19}$$

Sometimes a parameter σ is included in the sigmoid function to control the steepness as illustrated in Fig 3-5.

$$f(x) = \frac{1}{1 + e^{-\sigma x}} \tag{3.20}$$

The implementation of neural network can be broken down into two phases: the training phase and the approximation phase. In the training phase the weights w_i, bias b, and the steepness parameter σ in every unit of every layer is tuned in order to minimize the objective function defined as

$$R = \sum_{i=1}^{N} (y_i - g(x_i))^2 \tag{3.21}$$

where (x_i, y_i)'s are the "training data" consisted of inputs and outputs, and $g(x_i)$ is the neural network approximation for y_i. To find the optimal parameters w_i, b, σ , a technique

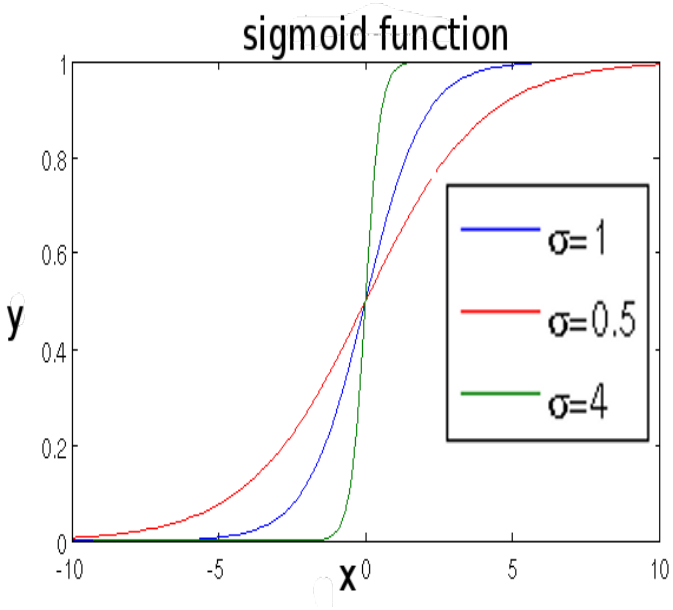


Figure 3-5: Sigmoid function under different σ.

called "backward propogation" can be used. We will not discuss this topic here.

Similar to other parametric regression techniques, neural network may suffer from over-fitting when there are too many units to tune for a given number of training samples. This can be suppressed by adding a regularization penalty for the "complexity" to the objective function, Eqn (3.22), known as the *weight decay*.

$$R = \sum_{i=1}^{N} (y_i - g(x_i))^2 + \underbrace{\sum_i w_i^2 + \sum b^2}_{\text{penalty}} \tag{3.22}$$

Because neural network is a *flexible* and *parametric* regression tool, We will use it for surrogate building in the following chapters.

3.4 Formulation

Now we introduce the Black-box Stencil Interpolation Method. Although the model problem in this section is time-dependent, the extension to time-independent problems is straightforward. Suppose a system is governed by a PDE

$$\frac{\partial y}{\partial t} = \mathcal{F}(y;\xi)\,, \tag{3.23}$$

in which

$$y = y(t,x,\xi) \tag{3.24}$$

We call Eqn (3.23) the *true physics*. y is the solution we are interested in. It has a dependence on time t, spatial location x, and the control or uncertain parameter ξ. $\mathcal{F}$ is a differential operator depending on the parameter ξ, indicating the true physics. Although the true physics $\mathcal{F}$ is unknown to us, we may have *some* knowledge about it and can give an educated guess $\mathcal{L}$ to approximate $\mathcal{F}$, i.e.

$$\mathcal{L}(y;\xi) \approx \mathcal{F}(y;\xi) \tag{3.25}$$

The corresponding approximate governing equation reads

$$\frac{\partial \hat{y}}{\partial t} = \mathcal{L}(\hat{y};\xi)\,,^{4} \tag{3.26}$$

In every simulation run of the exact model Eqn (3.23), a set of input-output data

$$\xi, t, x \;\rightarrow\; y\,, \tag{3.27}$$

i.e. the solution $y(t,x)$ for a realization of ξ, can be obtained. Clearly, the data, Eqn (3.27), which is generated by the true physics Eqn (3.23), does not satisfy the approximate physics,

[4] In the absence of any knowledge about $\mathcal{F}$, we set $\mathcal{L} = 0$. Notice this is just a special case of Eqn (3.25) and Eqn (3.26)

i.e.

$$\frac{\partial y}{\partial t} \neq \mathcal{L}(y;\xi) \tag{3.28}$$

To fix this discrepancy, a correction term can be added to Eqn (3.26). To this end, neural network can be used to approximate the residual $\frac{\partial y}{\partial t} - \mathcal{L}(y;\xi)$. The resultant residual can be used to correct Eqn (3.26) to approximate Eqn (3.23).

We first classify the parameters ξ's into 2 categories: *global parameters* ξ^g and *local parameters* ξ^l. Global parameters *are not* functions of x and t, whereas local parameters *are* functions of x or t. Take the heat equation on a 1-D stick for example. Suppose the stick is made of homogeneous material, and is heated by a spatially varying heating source. The governing equation is (3.29)

$$\frac{\partial}{\partial t} y = \xi^g \nabla^2 y + \xi^l(x,t)\,, \tag{3.29}$$

where y is the temperature. Because the material is homogeneous, ξ^g, the heat conductivity, is a fixed scalar parameter independent of x and t. Therefore, ξ^g is a global parameter. ξ^l, the heating source, is a variable dependent on x and t. Therefore, ξ^l is a local parameter.

In order to build a surrogate to correct the discrepancy between the true physics and the approximated physics, we plug the data, Eqn (3.27) into the simplified physics, Eqn (3.26), and compute the residual

$$R = \frac{\partial y}{\partial t} - \mathcal{L}(y;\xi)\,, \tag{3.30}$$

where $\frac{\partial y}{\partial t}$ is computed by finite difference of the data in time. Notice the residual depends on t, x, ξ^g, ξ^l at any location x, and also the solution y at the location x and its neighborhood. To write it explicitly,

$$R = R(t, x, \xi^g, \xi^l(x), y(\bar{x})) \tag{3.31}$$

where we use $\bar{x}$ to indicate the location x and its neighborhood. Although $\xi^l(x)$ can be varying spatially, R at a specific location x only depends on ξ^l locally.

Because the solution data, Eqn (3.27), satisfies both Eqn(3.30) and Eqn(3.23), the two equations must be identical

$$\text{Identical Physics} \begin{cases} \dfrac{dy}{dt} = \mathcal{F}(y;\xi) \\ \dfrac{dy}{dt} = \mathcal{L}(y;\xi) + R \end{cases} \tag{3.32}$$

Therefore, if an accurate surrogate for R can be built, we can non-intrusively recover the true physics by the corrected version of the simplified physics. We call the corrected version of the simplified physics to be *corrected simplified physics* or *corrected approximate physics*. In general, the surrogate should take the inputs of t, x, ξ^g, local $\xi^l(\bar{x})$, and $y(\bar{x})$. However, under the "physics invariance" assumption, we can remove the inputs t and x since R should be invariant under temporal and spatial translation. We build the surrogate by neural network, whose inputs and output are shown in Eqn (3.33)

$$\begin{aligned} \text{inputs}: &\quad \xi^g, \xi^l(\bar{x}), y(\bar{x}) \\ \text{output}: &\quad R \end{aligned} \tag{3.33}$$

$y(\bar{x})$ can be approximated locally by $y(x)$, $\nabla y(x)$, and $\nabla\nabla y(x)$ at x; therefore, we can use them to substitute $y(\bar{x})$ as inputs of the neural network. Besides, $y(x)$, $\nabla y(x)$, and $\nabla\nabla y(x)$ can be approximated by taking finite difference on the data. This is also true for $\xi(\bar{x})$.

Now we discuss the analogy of Eqn (3.29) through Eqn (3.33) (continuous in space) in a discretized setting. Suppose Eqn (3.23) and Eqn (3.26) are defined on a two-dimensional spatial domain and we discretize them by finite difference. The inputs and output of R will be: (we adopts the same notation as in Eqn(3.3))

$$\begin{aligned} \text{inputs}: &\quad \xi^g, \xi^l_0, \xi^l_l, \xi^l_r, \xi^l_u, \xi^l_d, y_0, y_l, y_r, y_u, y_d \\ \text{output}: &\quad R_0 \end{aligned} \tag{3.34}$$

Analogous to the continuous formulation, we can also use the alternative inputs

$$\begin{aligned} \text{inputs}: \quad & \xi^g, y_0, \frac{y_u - y_d}{2\Delta x}, \frac{y_r - y_l}{2\Delta x}, \frac{y_u + y_d - 2y_0}{\Delta x^2}, \frac{y_r + y_l - 2y_0}{\Delta x^2} \\ & \xi_0^l, \frac{\xi_u^l - \xi_d^l}{2\Delta x}, \frac{\xi_r^l - \xi_l^l}{2\Delta x}, \frac{\xi_u^l + \xi_d^l - 2\xi_0^l}{\Delta x^2}, \frac{\xi_r^l + \xi_l^l - 2\xi_0^l}{\Delta x^2} \\ \text{output}: \quad & R_0 \end{aligned} \tag{3.35}$$

If the data is defined on an unstructured finite element mesh, we will need to approximate the inputs in Eqn (3.35). This extension, however, is not attempted in this thesis and should be attempted in a future work.

After obtaining a surrogate $\tilde{R}$ to approximate R, the next step is to apply the Discrete Empirical Interpolation Method to the corrected approximation equation

$$\frac{d\tilde{y}}{dt} = \mathcal{L}(\tilde{y}; \xi) + \tilde{R}(\tilde{y}; \xi)\,, \tag{3.36}$$

where $\tilde{R}$ is the surrogate approximating the exact correction term R. Without loss of generality, we assume $\mathcal{L}$ is a linear operator. Notice even if $\mathcal{L}$ is linear, Eqn (3.36) is nonlinear because $\tilde{R}$ is nonlinear. Suppose, in its discrete version, $y \in \mathbb{R}^N$, $\mathcal{L}$ becomes an $N - by - N$ matrix A. Thus

$$\frac{d\tilde{y}}{dt} = A\tilde{y} + \tilde{R} \tag{3.37}$$

For notation cleanness, we denote $\tilde{y}$ as y and $\tilde{R}$ as R in the rest of this chapter. Given some training simulations, we obtain some snapshots on y, and can construct d POD modes: an $N - by - d$ matrix V. We can also obtain the snapshots for R by Eqn (3.30), and construct the POD modes U, an $N - by - m$ matrix, for R. Therefore, we can have the following approximation

$$\begin{aligned} y &\approx Vz \\ R &\approx Ur \end{aligned} \tag{3.38}$$

with appropriate z and r. Plug Eqn (3.38) to Eqn (3.37) and apply Galerkin projection, we have

$$\frac{dz}{dt} = V^T AVz + V^T Ur \tag{3.39}$$

To evaluate the nonlinear term $V^T Ur$, we apply the Discrete Empirical Interpolation Method (discussed in Chapter2). Eqn (3.39) can be rewritten as

$$\frac{dz}{dt} = V^T AVz + (V^T U)(U^T P)(\underbrace{P^T R(Vz)}_{r}), \tag{3.40}$$

in which P is an $m - by - N$ index matrix. Eqn (3.40) has an identical form as the formulation of the Discrete Empirical Interpolation Method given in Chapter 2, but we need to notice R is a surrogate that approximates the correction term.

If no knowledge about the true physics is known, the term of $\mathcal{L}$ is removed from Eqn (3.26), Eqn (3.30), Eqn (3.32). This is a special case of the discussion above.

To sum up, BSIM-based model reduction procedure can be arranged sequentially into three phases: *full model simulation phase*, *training phase*, *reduced model prediction phase*:

The procedure for the *full model simulation phase* is:

1. Run full model simulation, governed by Eqn (3.23), for k times, under different parameter realizations.
2. Collect stencil data, for example Eqn (3.35), on every gridpoint and at every timestep.
3. Compute $\frac{\partial y}{\partial t}$ by finite difference in time.
4. Compute R by Eqn (3.30) on every gridpoint and at every timestep.
5. Evaluate $\mathcal{L}(y;\xi)$ and $R(y;\xi)$ at every timestep [5], collect resultant snapshots for $\mathcal{L}$ and

[5] y and ξ at each timestep is plugged in $\mathcal{L}(y;\xi)$ to obtain a spatial field of $\mathcal{L}$ at that timestep. This spatial field is a snapshot of $\mathcal{L}$. The same statement also holds for R

R.

The procedure for the *training phase* is:

1. Train neural network by stencil data.[6] to obtain a surrogate $\tilde{R}$ for R, with inputs and output being Eqn (3.35).

2. Apply POD to the snapshots of $\mathcal{L}$ and R, obtain corresponding POD modes V and U [24].

3. Apply DEIM to U, obtain DEIM points and index matrix P [9].

4. Pre-compute reduced matrices V^TAV, $(V^TU)(U^TP)$, and CV [7]

The procedure for the *reduced model prediction phase* at each timestep[8] is:

1. Compute r in Eqn (3.40) from z^t by evaluating surrogate $\tilde{R}$ on DEIM points.

2. Time advance from z^t to z^{t+1} using Eqn (3.40).

3. Compute output y of the system in the formulation of z^t [9]

$$y^t = (CV)z^t \tag{3.41}$$

[6]The inputs of the neural network are the inputs of stencil data, e.g. inputs in Eqn (3.35) The output of the neural network is the output of stencil data, e.g. output in Eqn (3.35),

[7]C is the matrix related to the output of the system, see Eqn (1.3).

[8]We haven't discussed the time marching scheme for Eqn (3.40). This topic will be postponed to Chapter 4.

[9]C is the matrix related to the output of the system, see Eqn (1.3).

Chapter 4

Application and Validation of BSIM-based Model Reduction

In this chapter, we will show some examples on which we apply the BSIM-based model reduction method. The example problems include two 1-dimensional chemical reaction problems and a 2-dimensional oil-water porous media flow problem.

4.1 One Dimensional Chemical Reaction

4.1.1 A Steady State Problem

The first test problem we consider is a one-dimensional, time-independent problem illustrated by Fig 4-1. Fuel is injected from the left end of a tube. The physics in the tube is modelled by convection and chemical reaction. The fuel saturation lowers down gradually because of the chemical reaction, and the remnant fuel flows out of the tube from the right end of the tube.

This problem is modelled by Eqn (4.1) [19].

$$\frac{\partial}{\partial x}(us) - \frac{\partial}{\partial x}\left(\mu \frac{\partial}{\partial x} s\right) + g(s; A, E) = 0 \tag{4.1}$$

Figure 4-1: Illustration of 1-D time-independent chemical reaction problem.

Here, $0 \leq x \leq 1$. s is the fuel saturation we are solving for, which is a function of the spatial location x. We know that $0 \leq s \leq 1$ because s is the saturation. The reaction rate $g(s; A, E)$ is a nonlinear function defined by

$$g(s; A, E) = As(2c - s) \exp\left\{-\frac{E}{d - s}\right\}, \tag{4.2}$$

where $d = 0.24$ and $c = 0.1$. The parameters A (or equivalently $ln(A)$ [1]) and E are two scalar uncertain *global parameters* (see Chapter 3 for the definition of "global parameter") that determine the reaction rate. The flow velocity $u = 0.2$. The diffusivity μ is defined by

$$\mu(s) = \frac{\mu_0}{1 + e^{-10s}}, \tag{4.3}$$

where $\mu_0 = 10^{-5}$. [2]. The boundary conditions for Eqn (4.1) are

$$\begin{aligned} s(x = 0) &= c \\ \frac{\partial}{\partial x} s(x = 1) &= 0 \end{aligned}, \tag{4.4}$$

where $c = 0.1$ is the saturation at the left end. In this problem, the uncertain parameters A and E are global, and there are no local parameters. We use *nearest neighbor interpolation* for surrogate construction, because we were not aware of *neural networks* when this work was implemented.

[1] We use $ln(A)$ instead of A just to be consistent with [19]

[2] We choose values of parameters such as u and μ_0 to be consistent with [19]

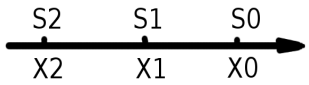


Figure 4-2: The notation of a stencil.

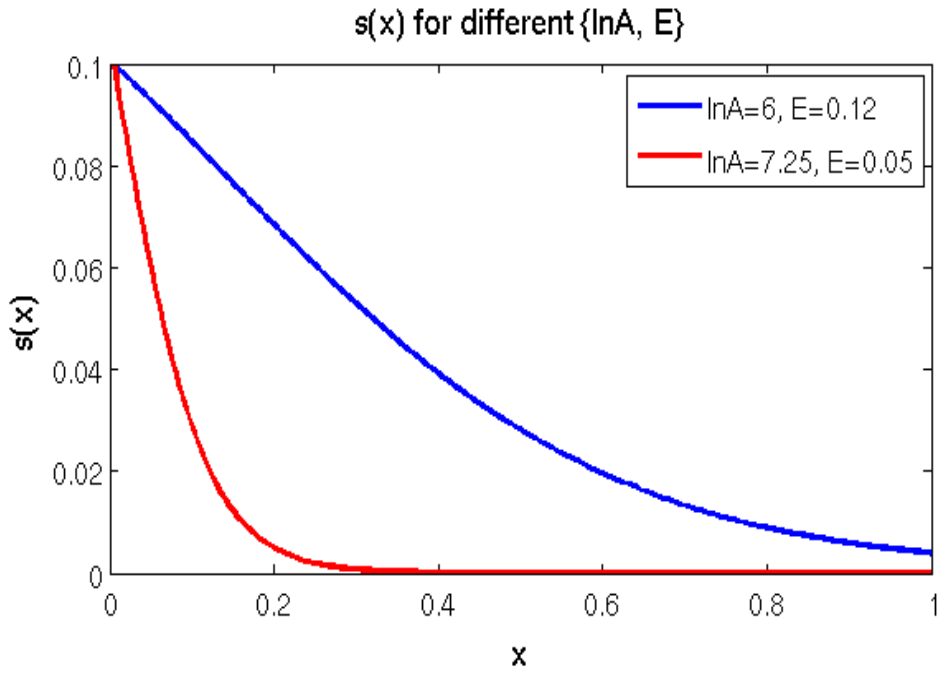


Figure 4-3: Using different realizations of $ln(A), E$, we obtain different solutions $s(x)$. For the red line $ln(A) = 6$, $E = 0.12$. For the blue line $ln(A) = 7.25$, $E = 0.05$.

Applying first-order upwind finite volume scheme [12] to Eqn (4.1), we get[3]

$$u\frac{s_0 - s_1}{\Delta x} - \frac{1}{\Delta x^2}\left(\mu(s_1)(s_0 - s_1) + \mu(s_2)(s_2 - s_1)\right) + As_1(2c - s_1)\exp\left\{-\frac{E}{d - s_1}\right\} = 0\,, \quad (4.5)$$

where the notation of a stencil is shown in Fig 4-2. At every gridpoint, we solve s_0 from s_1 and s_2. This procedure is sweeped from the left end, $x = 0$, to the right end, $x = 1$, in order to get the solution $s(x)$ at every gridpoint. Fig 4-3 shows the solution $s(x)$ for two different realizations of the uncertain parameters $ln(A)$ and E.

[3] $u > 0$, the fuel flows from left to right. So the upwind direction is left.

Assuming we have no knowledge about the chemical reaction term $g(s; A, E)$ and the diffusivity term $\mu(s)$, we use an approximate model:

$$\frac{\partial}{\partial x}(u\hat{s}) - \frac{\partial}{\partial x}(\bar{\mu}\frac{\partial}{\partial x}\hat{s}) = 0\,, \tag{4.6}$$

where $\bar{\mu}$ is an empirically chosen diffusivity that is independent of s. The discretized counterpart of Eqn (4.6) is given by

$$u\frac{\hat{s}_0 - \hat{s}_1}{\Delta x} - \frac{1}{\Delta x^2}\bar{\mu}\,(\hat{s}_0 + \hat{s}_2 - 2\hat{s}_1) = 0\,. \tag{4.7}$$

By adding a residual R to Eqn (4.7), we get

$$u\frac{s_0 - s_1}{\Delta x} - \frac{1}{\Delta x^2}\bar{\mu}\,(s_0 + s_2 - 2s_1) + R = 0\,, \tag{4.8}$$

which should be identical to the true model given by Eqn (4.5). Comparing Eqn (4.5) with Eqn (4.8), we can determine that the surrogate $\tilde{R}$ for R takes $s_1, s_2,$ [4] $ln(A)$, and E as inputs.

In our implementation, we choose an equivalent set of inputs $s_1, \frac{s_1 - s_2}{\Delta x}, ln(A)$ and E to avoid the strong correlation between inputs s_1 and s_2 [5]. By choosing the alternative input $\frac{s_1 - s_2}{\Delta x}$, the surrogate can be more sensitive to the derivative of the saturation, and potentially be more accurate.[6]

In the full model simulation phase, we set the range of $ln(A)$ to be $[5.00, 7.25]$ and the range of E to be $[0.05, 0.15]$, similar to in [19]. The full model simulations are performed using k uniformly generated samples in the parameter space $(ln(A), E)$.

[4] We solve for s from left to right by a single sweep, and s_0 is **what we are solving for** at each gridpoint. s_0 cannot be the input of $\tilde{R}$ because we want to use an explicit scheme. This is because explicit scheme is cheap in computation.

[5] At every gridpoint s_1 is generally close to s_2 because the fuel concentration is continuous in space.

[6] Based on our observation, the effect on accuracy is slight . This is because neural network will feed **linear combinations** of the inputs (See Chapter 3 3.3.3) to the nonlinear function. Neural network will automatically optimize the linear combination coefficients.

We discretize the spatial space x into $N = 200$ equidistant intervals. Therefore, from each training simulation, we obtain N samples, which we call *stencil samples*,

$$s_1, \frac{s_1 - s_2}{\Delta x}, ln(A), E \quad \longrightarrow \quad R\,, \tag{4.9}$$

where R is computed by plugging $s_0, s_1, s_2, ln(A), E$ into Eqn (4.8). So a total of $k \times 200$ stencil samples are obtained from k simulations [7].

If $\tilde{R}$ (the surrogate for R) is exact, $s_0, s_1, s_2, ln(A), E$ from the true physics will satisty

$$u\frac{s_0 - s_1}{\Delta x} - \frac{1}{\Delta x^2}\bar{\mu}\,(s_0 + s_2 - 2s_1) + \tilde{R} = 0 \tag{4.10}$$

exactly, and Eqn (4.10) can replace Eqn (4.5) without any error. But the sentence above is not true: surrogate $\tilde{R}$ cannot be exactly accurate and there will always be discrepancy between Eqn (4.10) and Eqn (4.5). Nearest neighbor interpolation is used to construct $\tilde{R}$ from the stencil samples.

The results are shown in Fig 4-4, Fig 4-5, and Fig 4-6, where we choose $ln(A) = 6.00$ and $E = 0.12$. In the training, we vary the values of k and μ_0 to analyze their effect on the quality of the surrogate and the solution of Eqn (4.10). In Fig 4-4 $k = 20$, $\mu_0 = 0.05$; In Fig 4-4 $k = 20$, $\mu_0 = 0.50$; In Fig 4-4 $k = 50$, $\mu_0 = 0.50$.

The quality of the surrogate for R depends on locations of the randomly chosen training parameters $(ln(A), E)$. We expect the approximate solution to be different everytime the surrogate is trainned on a different set of training parameters. Even if the numbers of training simulations are the same, the surrogate and the approximate solution can be different.

We first observe that the difference between the approximate solution and the true so-

[7] We will talk about the value of k later.

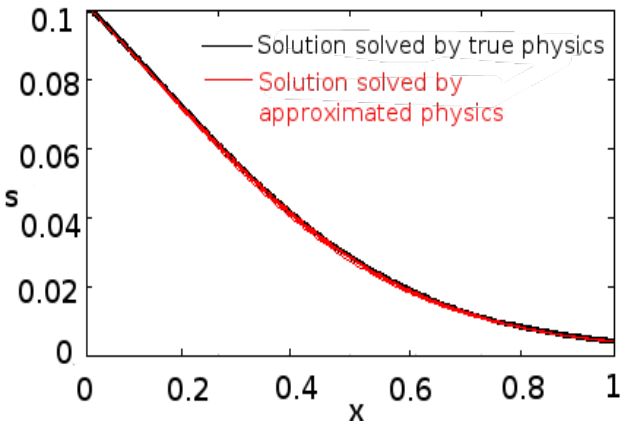


Figure 4-4: $k = 20$, $\mu_0 = 0.05$. The red lines are computed by the corrected simplified physics, while the black line is computed by the true physics.

lution decreases as k increases. This is because the number of the stencil samples is proportional to k, and surrogate construction is more accurate if the number of stencil samples increases. We also find that the error decreases as μ_0 decreases. This is because a smaller μ_0 makes the variation of μ less important, and makes the approximate physics more accurate. [8]

In conclusion, this test case validates the Black-box Stencil Interpolation Method in a simple scenario. However, 3 components are missing in the discussion above:

1. The uncertain parameters A and E are *global* parameters. No local parameters are involved in this problem.

2. We use nearest neighbor interpolation instead of neural network for surrogate construction.

3. We have not applied model reduction to the corrected approximate physics yet.

In the following sections we will add these components to fully demonstrate the BSIM-based model reduction method.

[8] We choose $\bar{\mu}$ to be not biased, i.e. we choose μ_0 as the average of μ at every gridpoint at every timestep.

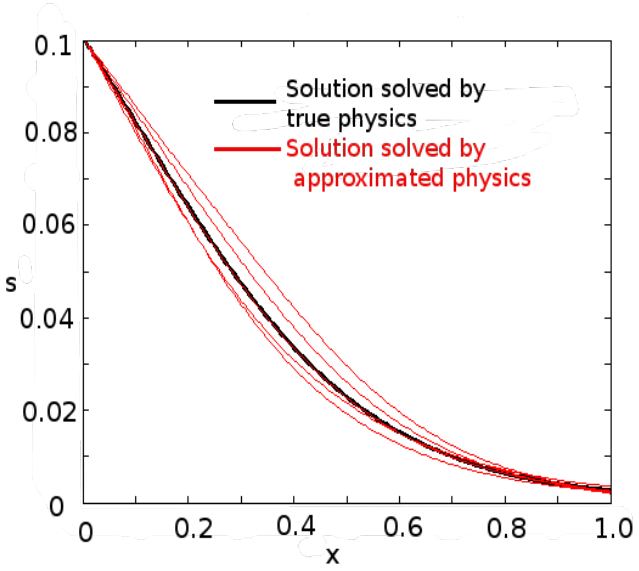


Figure 4-5: $k = 20$, $\mu_0 = 0.50$. As μ_0 increases, the discrepancy of the diffusion term between the simplified physics and the true physics increases, making the the approximation for R more difficult.

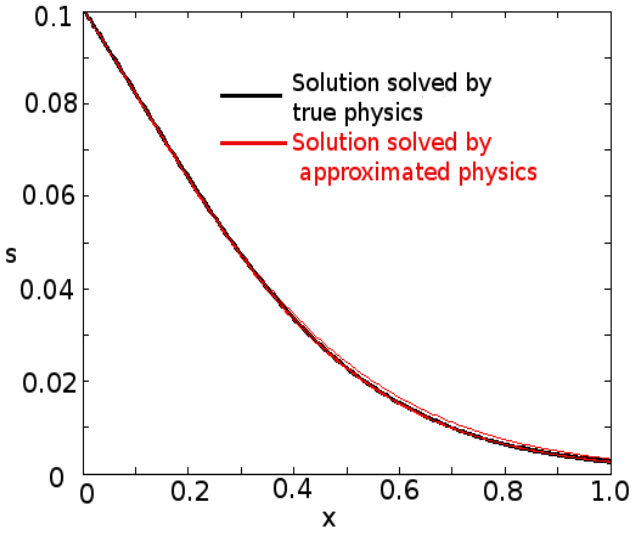


Figure 4-6: $k = 50$, $\mu_0 = 0.50$. The accuracy of the surrogate and the corrected simplified model increases as the number of training simulatons increases.

4.1.2 A Time Dependent Problem

As the second test case, we consider the 1-D chemical reaction problem described above, however now the saturation is a function of time and space, and we introduce spatially distributed fuel injection. This problem is modelled by a time-dependent hyperbolic PDE:

$$\frac{\partial s}{\partial t} + \frac{\partial}{\partial x}\left(us - \mu\frac{\partial s}{\partial x}\right) = J(t,x) - g(s)\,, \tag{4.11}$$

where $s = s(x,t)$ is the fuel saturation we are solving for, $x \in [0,1]$, $t \in [0,1]$, and $g(s)$ is the chemical reaction term given by Eqn (4.2). In this section we do not consider constants A and E as uncertain, but instead take them to be fixed constants. $J(t,x)$ is the spatially distributed, time dependent fuel injection rate given by

$$J(t,x) = \sum_{i=1}^{5} J_i(t) e^{-\frac{(x-x_i)^2}{2\sigma^2}}\,, \tag{4.12}$$

as shown in Fig 4-7. The velocity $u = 1$ and the diffusivity $\mu = 0.01$ are known scalar constants.

At the ith injector, the injection rate $J_i(t)$ is assumed to be a Gaussian random process independent of other injectors [10].[9] The mean of the Gaussian random process is 5. The covariance function is

$$cov(t_1,t_2) = \sigma^2 \exp\left\{-\frac{|t_1-t_2|^2}{2l^2}\right\}\,, \tag{4.13}$$

where $\sigma = 1$ and $l = 0.5$ [10]. If $J < 0$ happens in a given realization, we just discard this realization. A realization of the injection rates is shown in Fig 4-8.

As in the last problem we discretize x into N equidistant gridpoints. We denote the discretized saturation at time t as a vector $s^t \in \mathbb{R}^N$. A fully implicit time marching scheme

[9] Please refer to [10] for more information about Gaussian random process.

[10] These parameters are nothing special, they are just chosen to test our method.

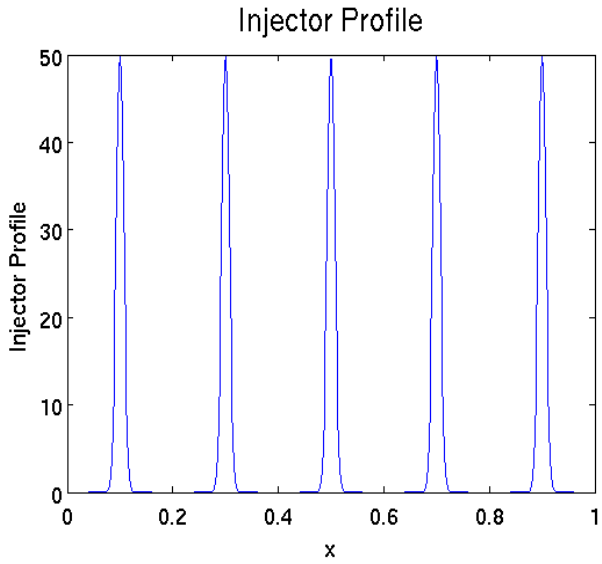


Figure 4-7: Injection rate is concentrated at 5 injectors. For illustrative purpose we set $J_i = 5,\ i = 1, \cdots, 5$

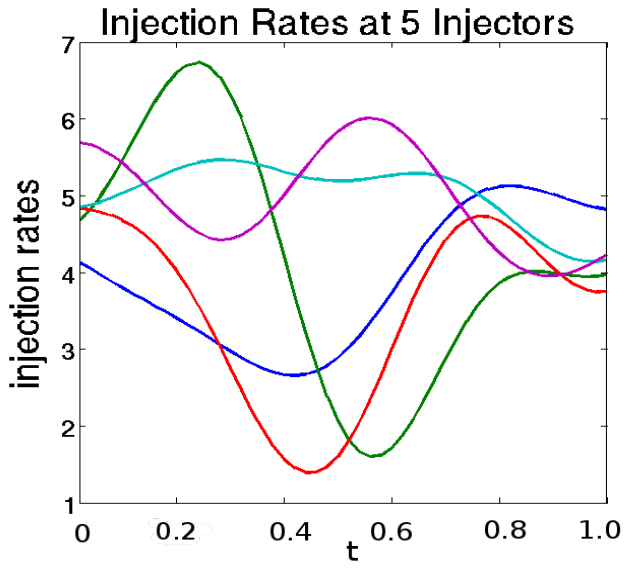


Figure 4-8: A realization of 5 i.i.d. Gaussian processes, which describes the time dependent injection rates at 5 injectors.

applied to Eqn (4.11) gives

$$\frac{1}{\Delta t}(s^{t+1}-s^t)+\frac{1}{\Delta x}uAs^{t+1}-\frac{1}{\Delta x^2}\mu Bs^{t+1}=J^{t+1}-g(s^{t+1})\,, \tag{4.14}$$

where

$$A=\begin{pmatrix} 1 & & & & \\ -1 & 1 & & & \\ & \cdots & & & \\ & -1 & 1 & & \\ & & -1 & 1 \end{pmatrix}_{N\times N} \qquad B=\begin{pmatrix} -2 & 1 & & & \\ 1 & -2 & 1 & & \\ & & \cdots & & \\ & & 1 & -2 & 1 \\ & & & 1 & -1 \end{pmatrix}_{N\times N} \tag{4.15}$$

Assuming we have no knowledge about the chemical reaction term, we have the simplified physics:

$$\frac{\partial \hat{s}}{\partial t}+\frac{\partial}{\partial x}\left(u\hat{s}-\mu\frac{\partial \hat{s}}{\partial x}\right)=J(t,x) \tag{4.16}$$

The corrected simplified model is

$$\frac{\partial s}{\partial t}+\frac{\partial}{\partial x}\left(us-\mu\frac{\partial s}{\partial x}\right)=J(t,x)+\tilde{R} \tag{4.17}$$

The surrogate $\tilde{R}$ takes $s_0^t,\ \frac{s_0^t-s_1^t}{\Delta x},\ \frac{s_0^t+s_2^t-2s_1^t}{\Delta x^2},\ J(t,x_0)$ as inputs, where we adopt the notation in Fig 4-2. Eqn (4.17) is discretized as

$$\underbrace{\frac{1}{\Delta t}(s^{t+1}-s^t)+\frac{1}{\Delta x}uAs^{t+1}-\frac{1}{\Delta x^2}\mu Bs^{t+1}=J^{t+1}}_{I}-\underbrace{\tilde{R}(s^t,\frac{1}{\Delta x}As^t,\frac{1}{\Delta x^2}Bs^t,J^t)}_{II}\,. \tag{4.18}$$

Notice, the true physics Eqn (4.14) is discretized implicitly; whereas the corrected simplified physics Eqn (4.18) is discretized semi-implicitly, in other words, part I is implicit and part II is explicit. This is because we want to avoid time marching Eqn (4.18) by expensive nonlinear Newton iteration.

We construct the surrogate for R by running k *true physics* simulations, with each sim-

ulation corresponding to a random realization of J. We will explain how we construct the surrogate later. We discretize x and t into N and T equidistant segments, respectively. The number of stencil samples [11] will therefore be $\mathcal{O}(kNT)$.

At each training simulation, we get s_0^t, s_1^t, s_2^t, J^t at every gridpoint at every timestep. They are plugged into Eqn (4.18) to compute R [12]. Therefore, at every timestep, we get a snapshot of R. We can arrange the snapshots into an N-by-kT snapshot matrix (see Chapter 3):

$$Q_R = \left(R_1^1 \cdots R_1^T \cdots R_k^1 \cdots R_k^T \right) , \tag{4.19}$$

where R_j^i indicates the snapshot of R at the ith timestep at the jth simulation.
Similarly, we get the snapshot matrix Q_s for s, i.e. we arrange snapshots of s into an N-by-kT snapshot matrix (see Chapter 3):

$$Q_s = \left(s_1^1 \cdots s_1^T \cdots s_k^1 \cdots s_k^T \right) , \tag{4.20}$$

where s_j^i indicates the snapshot of s at the ith timestep at the jth simulation.

In addition, we obtain the stencil sample data from training simulations:

$$s_0^t, \frac{s_0^t - s_1^t}{\Delta x}, \frac{s_0^t + s_2^t - 2s_1^t}{\Delta x^2}, J^t \longrightarrow R^t \tag{4.21}$$

We apply POD to Q_s and Q_R to get the principal modes for s and R, respectively. [13] We get

$$\begin{aligned} Q_s &\approx V \Sigma_s \tilde{V} \\ Q_R &\approx U \Sigma_R \tilde{U} \end{aligned} , \tag{4.22}$$

[11] Please see Section 4.1.1 for the definition of *stencil sample*.
[12] Here, we need to compute R, not $\tilde{R}$. This is because we want to get the snapshots for the exact correction, not the snapshots for the approximate correction.
[13] Please see Chapter 3 or read [36] for more detail about *POD* and *principal mode*.

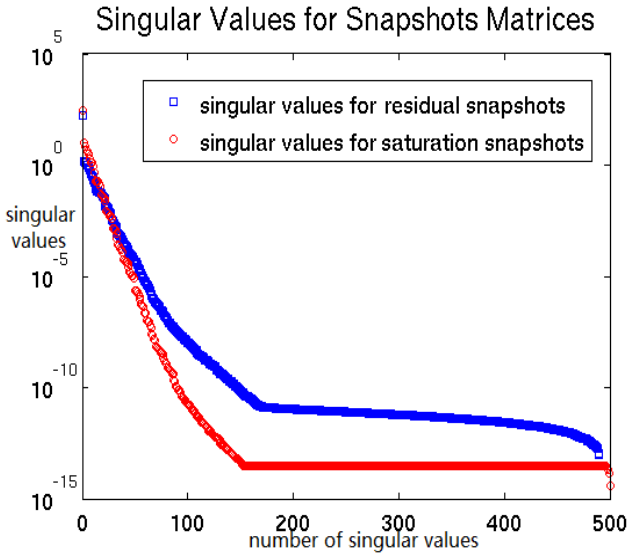


Figure 4-9: Singular values of Q_R and Q_s.

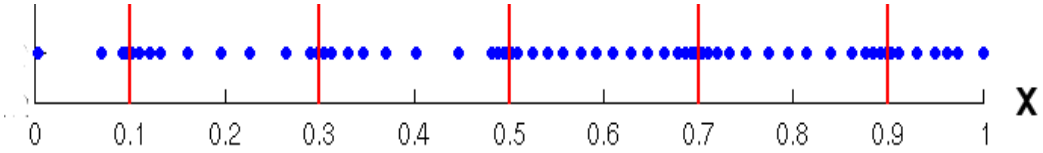


Figure 4-10: DEIM points for the residual.

where $V \in \mathbb{R}^{N\times d}$ and $U \in \mathbb{R}^{N\times m}$. The singular values $diag(\Sigma_s)$ and $diag(\Sigma_R)$ are shown in Fig 4-9. We find the singular values for the residual have a slower decay than for the saturation, which implies we need to keep $m > d$. In this example we choose $N = 500$, $d = 50$, and $m = 62$.

Next, we apply DEIM to U, using the method described in Chapter 2, Chapter 3 [9], to obtain DEIM points and index matrix P. We obtain the 62 DEIM points as shown in Fig 4-10 [14].

We use the stencil samples, Eqn (4.21), to train a neural network. The neural network

[14] Because we set $m = 62$, we can at most obtain 62 DEIM points. See Chapter 2, Chapter 3 or read [9].

has one hidden layer with 6 units, an input layer with 4 units, and an output layer with 1 unit. The neural network toolbox in MATLAB, especially the functions *fitnet* and *train*, is used for constructing and training the neural network.

The reduced model (see Chapter 3) for the corrected simplified physics Eqn (4.18) is

$$\frac{z^{t+1}}{\Delta t}+\frac{u}{\Delta x}(V^TAV)z^{t+1}-\frac{\mu}{\Delta x^2}(V^TBV)z^{t+1}=V^TJ^{t+1}-(V^TU)(P^TU)^{-1}(P^T\tilde{R})+\frac{z^t}{\Delta t}\,, \quad (4.23)$$

where $\tilde{R}$ is the surrogate for R, P is the N-by-m index matrix we just mentioned, z is a length-d state vector in the reduced model, and Vz^t gives an approximation to s^t. V^TAV, V^TBV are d-by-d matrices, and $(V^TU)(P^TU)^{-1}$ is a d-by-m matrix. V^TAV, V^TBV, and $(V^TU)(P^TU)^{-1}$ can be pre-computed. The computational procedure for the reduced model is as follows (x_0 indicates a DEIM point's location) [15]:

1. Pre-compute d-by-d matrix V^TAV and V^TBV

2. Pre-compute d-by-m matrix V^TU

3. Pre-compute m-by-m matrix $(P^TU)^{-1}$

4. **for** $t = 1 : T$, **do**

5. Evaluate approximately s_0^t, s_1^t, s_2^t by extracting corresponding entries of vector Vz^t [16].

6. Compute inputs of surrogate $\tilde{R}$: s_0^t, $\frac{s_0^t-s_1^t}{\Delta x}$, $\frac{s_0^t+s_2^t-2s_1^t}{\Delta x^2}$, $J^t(x_0)$

7. Evaluate neural network $\tilde{R}(s_0^t, \frac{s_0^t-s_1^t}{\Delta x}, \frac{s_0^t+s_2^t-2s_1^t}{\Delta x^2}, J^t(x_0))$, obtain $P^T\tilde{R}$

[15] We do not include the procedure of computing the system output y (See Chapter 1, Eqn (1.3)) though the procedure is straightforward. This is because we have not defined any system output in this section.

[16] $s^t \approx Vz^t$

8. Solve for z^{t+1} by [17]

$$z^{t+1} = \left(\frac{1}{\Delta t} I + \frac{u}{\Delta x} V^T A V - \frac{\mu}{\Delta x^2} V^T B V\right)^{-1} \left(V^T J^{t+1} - (V^T U)(P^T U)^{-1}(P^T \tilde{R}) + \frac{z^t}{\Delta t}\right) \tag{4.24}$$

9. **endfor**

We use 10 training simulations to build surrogate $\tilde{R}$ and snapshots matrices. Because $N = 500$ and $T = 50$, we can obtain 500×50 stencil samples from each simulation. Therefore we get a total of $250,000$ stencil samples from 10 simulations. This number of stencil samples is large, and can be a burden in training neural network. In this problem, we use stencil samples only on DEIM points to train neural network instead of using all stencil samples, in order to alleviate the computational burden [18]. Since we have 62 DEIM points in this problem, the number of stencil samples will be $62 \times 50 \times 10 = 31,000$.

Given a realization of injection rates, we compare the solution $s(t, x)$ of the true physics simulation, Eqn (4.14), with the solution reconstructed from the reduced dimensional corrected simplified physics simulation, Eqn (4.23). The reduced model performs satisfactorily as the two solutions are close. This is shown in Fig 4-11.

We also compared our method to two other reduced model techniques: parametric non-intrusive model reduction [1] (discussed in Chapter1) , and intrusive Discrete Empirical Interpolation Method [9] (discussed in Chapter2). In the parametric non-intrusive model reduction (Eqn (1.13)), we choose k[19] in Eqn (1.13) to be 50, and m[20] to be 50.

In our BSIM-based ROM, there are three sources of error that contributes to the difference between the solution of BSIM-based ROM and the true physics: POD, DEIM, and

[17] See Eqn (4.23).

[18] Here, we choose stencil samples only on 62 DEIM points. However, we do not have a solid evidence that this practice is better than choosing stencil samples on 62 random grid points.

[19] k in Eqn (1.13) does not mean "the number of simulations". Please refer to Chapter 2 for details.

[20] m in Eqn (1.13) does not mean "number of DEIM point". Please refer to Chapter 2 for details.

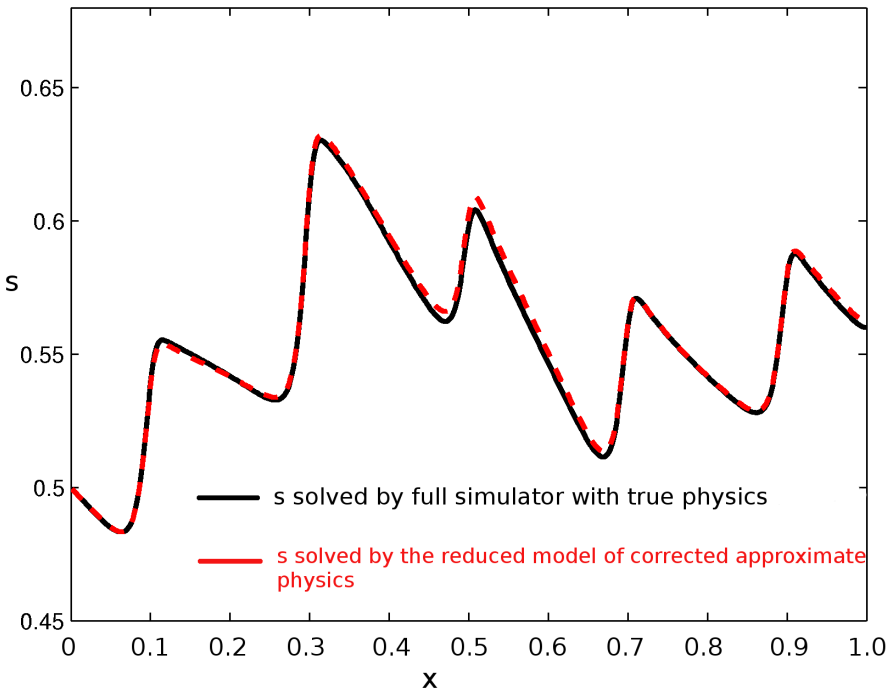


Figure 4-11: Solution $s(t = 1.0, x)$. The black line is solved by the full simulator with true physics. The red line is solved by the reduced model of the corrected approximate physics.

neural network surrogate: 1) POD restricts p to $range(V)$; 2) DEIM restricts $\tilde{R}$ (at every timestep) to $range(U)$, and interpolate $\tilde{R}$ by evaluations of $\tilde{R}$ only on DEIM points; 3) the surrogate $\tilde{R}$ is only an approximation for the exact R. In parametric non-intrusive ROM, there is only one source of error: POD. We do not want to compare the error introduced by POD in the two ROMs. We only want to see how our method improves the accuracy of parametric non-intrusive ROM based on the same number of POD modes and training simulations. The dimension of z in Eqn (4.23) is 50. To make the comparison fair, we should also set k in Eqn (1.13) to be 50. So we can make the error introduced by POD in both reduced models be equal.

In the intrusive Discrete Empirical Interpolation Method, we set $d = 50$ and $m = 62$. In intrusive DEIM ROM, there are only two sources of error: POD and DEIM. We focus on the error introduced by our surrogate $\tilde{R}$, which is the key difference between BSIM-based ROM and DEIM ROM. Because d and m are the same in both intrusive DEIM ROM and BSIM-based ROM, the error introduced by DEIM and by POD are similar. So the difference between the accuracy of two ROMs will be mostly due to the surrogate. The formulation of intrusive DEIM takes a similar form to Eqn (4.23), except for $\tilde{R}$ being replaced by R, and the equation is solved fully implicitly.

For all three methods, we use 10 simulations in the training. The solutions from the three reduced models (BSIM-based ROM, parametric non-intrusive ROM, intrusive DEIM ROM) are then compared against the solution from the true-physics full simulator. The comparison is shown in Fig 4-12, where the normalized error is defined as Eqn (4.25).

$$err(t) \equiv \sqrt{\int |s_{rm}(x,t) - s(x,t)|^2 \, \mathrm{d}x} \Big/ \sqrt{\int |s(x,t) - s_{ref}(x,t)|^2 \, \mathrm{d}x} \ , \tag{4.25}$$

where s_{rm} is the solution from a reduced model, s is the solution from the true-physics full simulator, and s_{ref} is the reference saturation, Eqn (4.26). s_{ref} refers to the average value

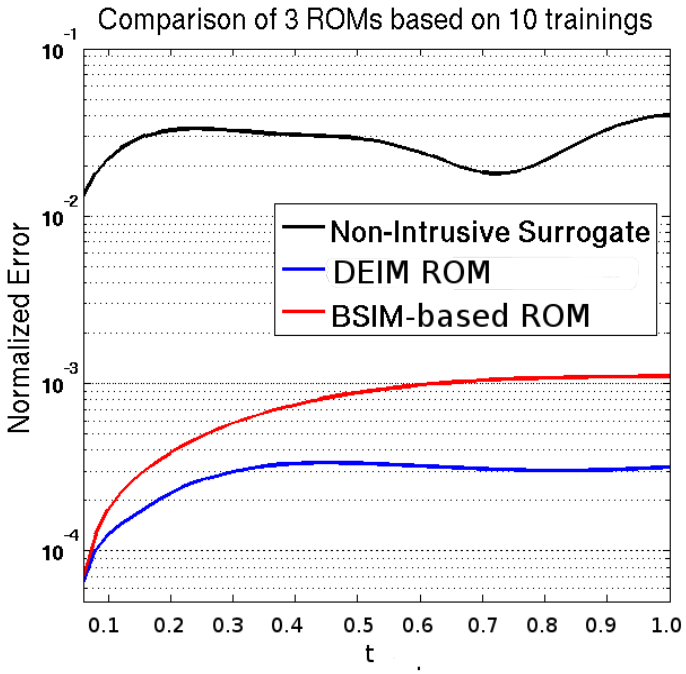


Figure 4-12: Compare the accuracy of the 3 reduced models.

of s over all the training simulations.

$$s_{ref}(x,t) = \frac{1}{k}\sum_{i=1}^{k} s_i(x,t) \tag{4.26}$$

We find that the accuracy of BSIM-based ROM is similar to that of the intrusive DEIM ROM, and is much better than parametric non-intrusive ROM. This is demonstrated in Fig 4-12.

We also tested the accuracy of our method using different neural networks. Specifically, we tried neural networks with 1, 3, and 6 hidden units in the hidden layer, respectively. The comparison is shown in Fig 4-13. We find that the neural network should be neither too complicated nor too simple, as the neural networks with 1 unit and with 6 units in the hidden layer perform worse than the neural network with 3 units in the hidden layer.

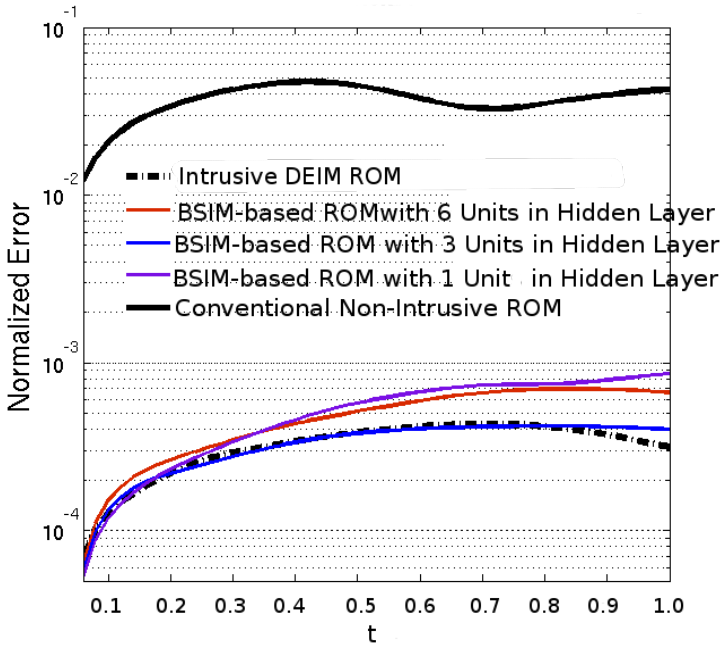


Figure 4-13: Compare the accuracy for different number of neural network hidden units.

In conclusion, this section demonstrates the applicability of BSIM-based model reduction in a 1D time-dependent problem. We will test a 2D problem in the next section.

4.2 A Two Dimensional Porous Media Flow Problem

In this section, we consider a 2-D nonlinear coupled PDE, Eqn (4.27), describing the water pressure p and oil saturation s in a porous media oil-water flow[15]. The problem is solved on a spatial domain $(x, y) \in [0, 1] \times [0, 1]$ (unit: 100 m), and time $t \in [0, 500]$ (unit: day).

$$\begin{aligned} -\nabla \cdot \left(\lambda_t K \nabla p + \lambda_n K \nabla p_c\right) &= 0 \\ -\phi \frac{\partial s}{\partial t} - \nabla \cdot \left(\lambda_w K \nabla p\right) &= 0 \end{aligned} \tag{4.27}$$

The permeability K is a spatially varying scalar random field [21]. We assume K to be an exponential Gaussian random field, which can be sampled by Karhunen-Loeve expansion [10]. The MATLAB code that generates K is attached in appendix A.1. An example realization of K is shown in Fig 4-14. λ_w and λ_n are mobilities of the wetting phase (water) and the non-wetting phase (oil)

$$\lambda_\alpha = \frac{k_{r_\alpha}}{\mu_\alpha}\,, \qquad \alpha = w, n\,, \tag{4.28}$$

in which μ_α stands for viscosity, $\mu_n = 10$, $\mu_w = 1$, and k_{r_α} stands for relative permeability. k_{r_α} is described by the Brooks-Corey model [6]

$$\begin{aligned} k_{r_w} &= (1 - s)^{\frac{2+3\theta}{\theta}}\,, \\ k_{r_n} &= s^2 \left(1 - (1 - s)^{\frac{2+\theta}{\theta}}\right)\,, \end{aligned} \tag{4.29}$$

where θ is the characteristic of the inhomogeneity of the medium [15]. In this thesis we set $\theta = 1.0$.

p_c is the capillary pressure

$$p_c(s) = p_d(1 - s)^{(-\frac{1}{\theta})}\,, \tag{4.30}$$

where $p_d = 10^{-3}$. ϕ is the porosity field. We assume $\phi = 0.2$ to be a constant scalar, independent of location.

[21] Generally K should be a non-scalar tensor. But for simplicity we assume K is just a scalar.

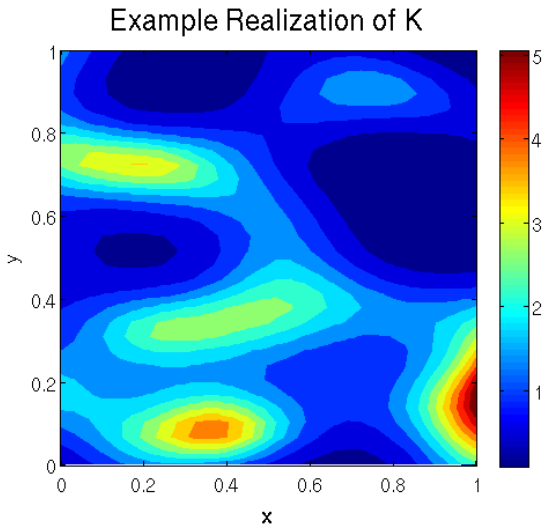


Figure 4-14: An example realization of K, the uncertain permeability field.

The boundary conditions for the simulator is

$$\begin{aligned} p &= 3 \quad \text{at } y = 0 \\ p &= 1 \quad \text{at } y = 1 \\ \hat{n} \cdot \nabla p &= 0 \quad \text{at } x = 0 \text{ and } x = 1 \\ s &= 0.15 \quad \text{at } y = 0 \end{aligned} \tag{4.31}$$

The initial condition for saturation is [22] [23]

$$s(t = 0) = 0.8 \tag{4.32}$$

Since the pressure at $y = 0$ (lower boundary) is larger than the pressure at $y = 1$ (upper boundary), the fluid will be pushed upwards. Because $s|_{y=0} = 0.15$ at the lower boundary, a mixture fluid of 85% water and 15% oil will be pushed in.

[22] We choose these boundary conditions and initial condition to be consistent with [15].

[23] We do not want to discuss the uniqueness or existence of the solution based on the boundary and initial conditions, because the author does not have expertise in this field. Please read [15] if interested.

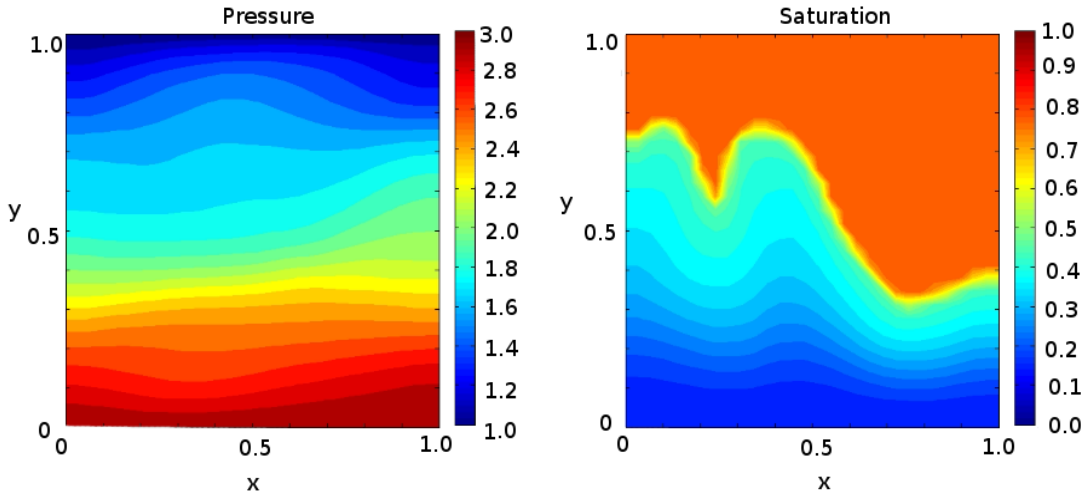


Figure 4-15: Example snapshot of pressure and saturation at $t = 400$

The coupled equation is solved by the *IMPES* [11] method, namely IMplicit in Pressure and Explicit in Saturation [24]. For the pressure equation, a 5-point stencil finite difference scheme is used. The saturation equation is solved by finite volume method with Van Leer flux limiter [23]. We do not dive into the details of the solver for the coupled equations[25]. An example snapshot of pressure and saturation at $t = 400$ days is given by Fig 4-15

In *IMPES*, the pressure solver costs most of the computation time, because pressure solves triggers a nonlinear Newton iteration at every timestep. Therefore we will consider **applying BSIM-based model reduction only to the pressure equation**. We assume the saturation solver in Eqn (4.27) is available, but neither the pressure solver nor the pressure equation is available. Our goal is to solve for the pressure at every timestep by BSIM-based model reduction, given the saturation s at the same timestep.

The simplified physics (sometimes we also call it "approximate physics") is assumed to

[24]In real reservoir simulations, *IMPES* is also a popular practice.

[25] If you are interested in the details, please send an email to the author requesting the MATLAB code.

be [26]

$$-\nabla^2 \hat{p} = 0 \tag{4.33}$$

Applying finite difference discretization, Eqn (4.33) can be written in a discrete 5-point stencil form

$$-\frac{1}{\Delta x^2}\big(\hat{p}_u^{t+1} + \hat{p}_d^{t+1} + \hat{p}_l^{t+1} + \hat{p}_r^{t+1} - 4\hat{p}_0^{t+1}\big) = 0 \tag{4.34}$$

where we adopt the same notation as in Eqn (3.3). Notice all p's are at time $t+1$, i.e. we adopts an implicit scheme in BSIM. This is necessary because the pressure equation is elliptic, so a change of pressure at a given spatial location will influence the pressure on the entire spatial field. In other words, the domain of influence is the entire field, so an implicit solving scheme is necessary [16]. We will omit superscript $t+1$ for notation cleanness. Eqn (4.34) can be re-written as

$$\hat{p}_0 = \frac{1}{4}\big(\hat{p}_u + \hat{p}_d + \hat{p}_l + \hat{p}_r\big) \tag{4.35}$$

A residual can be added to Eqn (4.35) to recover the true physics:

$$p_0 = \frac{1}{4}\big(p_u + p_d + p_l + p_r\big) + R \tag{4.36}$$

If we replace R with $\tilde{R}$ in Eqn (4.36), we obtain the corrected simplified physics [27].

The next step is to build $\tilde{R}$, a surrogate for R. First, we need to decide the inputs for $\tilde{R}$. The discretization of the pressure equation in Eqn (4.27) at the spatial grid point x_0 is

$$\begin{aligned} &\Phi_u p_u + \Phi_d p_d + \Phi_l p_l + \Phi_r p_r - \big(\Phi_u + \Phi_d + \Phi_l + \Phi_r\big)p_0 \\ =& -\Big(\Psi_u p_{cu} + \Psi_d p_{cd} + \Psi_l p_{cl} + \Psi_r p_{cr} - (\Psi_u + \Psi_r + \Psi_l + \Psi_r)p_{c0}\Big) \end{aligned}, \tag{4.37}$$

[26] We do not have a special reason to choose the simplified physics to be Eqn (4.33). We repeat that the simplified physics is up to the user's choice. But the user should choose it as close as the true physics based on his knowledge. Please see Chapter 3 for more details.

[27] See Chapter 33.4 for the definition of corrected simplified physics

where

$$\begin{aligned}\Phi_u &\equiv \frac{1}{4}\left(\lambda_t(s_u)+\lambda_t(s_0)\right)\left(K_u+K_0\right)\\ \Psi_u &\equiv \frac{1}{4}\left(\lambda_n(s_u)+\lambda_n(s_0)\right)\left(K_u+K_0\right)\end{aligned} \qquad \text{etc.} \tag{4.38}$$

p_c's are the capillary pressures, and λ_n, λ_w are the mobilities for the non-wetting phase and for the wetting phase respectively. Compare Eqn (4.36) with Eqn (4.37), we get

$$\begin{aligned}R = &\frac{1}{\Phi_u+\Phi_d+\Phi_l+\Phi_r}\\ &\left(\left(\Psi_u p_{cu}+\Psi_d p_{cd}+\Psi_l p_{cl}+\Psi_r p_{cr}-(\Psi_u+\Psi_r+\Psi_l+\Psi_r)p_{c0}\right)+\Phi_u p_u+\Phi_d p_d+\Phi_l p_l+\Phi_r p_r\right)\\ &-\frac{1}{4}\left(p_u+p_d+p_l+p_r\right)\end{aligned} \tag{4.39}$$

We can also expand Eqn (4.39) by plugging in Eqn (4.38) to Eqn (4.39) (this is straightforward and we will not do it in this thesis). Therefore, R and thus $\tilde{R}$ should have a dependence on $K_{u,d,l,r,0}$, $p_{u,d,l,r}$ and $s_{u,d,l,r,0}$.[28] The input dimensions for $\tilde{R}$ is 14 and output dimension is 1, which is illustrated in Fig 4-16.

The spatial domain is discretized into 30×30 grid points ($N = 900$), while the time domain is discretized into 500 segments, ($T = 500$). Therefore, from each simulation we can obtain $30 \times 30 \times 500$ stencil samples to build surrogate for R. The stencil samples are

$$K_{u,d,l,r,0}, p_{u,d,l,r}, s_{u,d,l,r,0} \longrightarrow R \tag{4.40}$$

where R is obtained by plugging $p_{u,d,l,r,0}$ into Eqn (4.36).

[28] However, in a non-intrusive environment, Eqn (4.39) is not available to determine the inputs for R and $\tilde{R}$. Nonetheless, we can select the inputs to be all the quantities on x_0 and its neighboring gridpoints. In this example, the quantities are $K_{u,d,l,r,0}$, $p_{u,d,l,r}$ and $s_{u,d,l,r,0}$ (p_0 is what we are solving for, so it should not be one of the inputs quantities of the surrogate.), which is the same conclusion as we get from Eqn (4.39).

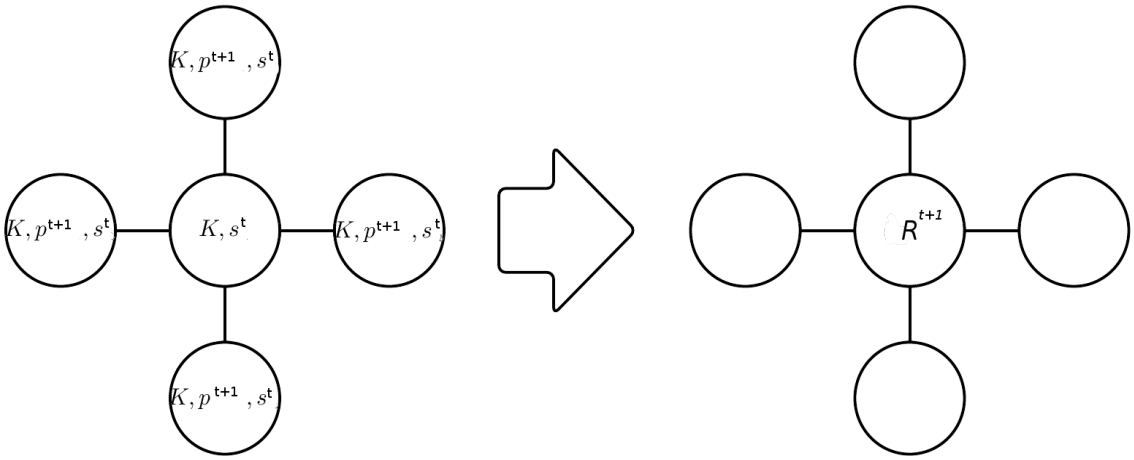


Figure 4-16: The inputs and output for R or $\tilde{R}$.

After training the surrogate, we can use

$$p_0 = \frac{1}{4}\big(p_u + p_d + p_l + p_r\big) + \tilde{R} \tag{4.41}$$

to evaluate p_0 from $K_{u,d,l,r,0}, p_{u,d,l,r}, s_{u,d,l,r,0}$ at a given gridpoint. As explained in Chapter 3, we conduct the following procedures (see Chapter 3 3.4):

1. Conduct k exact physics simulations with different realizations of K, obtain the snapshots for p, which is p at every timestep at every simulation.

2. Construct the snapshot matrix for p by the snapshots of p [29]

3. Apply POD to the snapshot matrix, obtain principal modes V, an N-by-d matrix [30]

4. At each timestep, compute

$$R_0 = p_0 - \frac{1}{4}(p_u + p_d + p_l + p_r) \tag{4.42}$$

[29] Please read Chapter 1 Eqn (1.6), or Chapter 4 Eqn (4.20) again if confused about how a snapshot matrix is build from snapshots.

[30] Please read Chapter 1, Chapter 3, and previous sections in Chapter 4 for how POD is applied to obtain V.

on every grid point, thus obtain a snapshot of R[31].

5. Construct the snapshot matrix for R by the snapshots of R.

6. Apply DEIM to the snapshot matrix of R, obtain principal modes U, DEIM points $\mathcal{U}$, and index matrix P, an N-by-m matrix [32].

Define

$$\begin{aligned} p'_{\mathcal{U}} &= P^T \left(\frac{1}{4}(p_u + p_d + p_l + p_r) + \tilde{R} \right), \\ p_{\mathcal{U}} &= P^T p \end{aligned} \tag{4.43}$$

where $p'_{\mathcal{U}}$ indicates the pressure computed by our approximate model on DEIM points. $p_{\mathcal{U}}$ indicates the pressure obtained by extracting the DEIM points' entries of a pressure snapshot.

Given a new realization of K, saturation s^{t-1}, and pressure p^{t-1}, the procedures for solving p^t by our BSIM-based ROM are

1. Take the initial guess: $p^t \Leftarrow p^{t-1}$, compute $z = V^T p^t$

2. Solve $z = \text{argmin}\|p'_{\mathcal{U}} - p_{\mathcal{U}}\|_2$, where $p'_{\mathcal{U}}$ and $p_{\mathcal{U}}$ is shown in Eqn (4.43).

3. Obtain result: $p^t = Vz$

In case the reader gets confused about how the second step is implemented, we explain it as follows:
Suppose $z = \text{argmin}\|p'_{\mathcal{U}} - p_{\mathcal{U}}\|_2$ is solved by an iterative method. At the ith iteration, we have p_i^t, the solution at time t of the current iteration step. At every DEIM point x_0, we gather the input quantities for the surrogate $\tilde{R}$, and gather pressures in x_0's neighboring

[31] A snapshot of R consists of R_0 at every grid point in the same time

[32] The procedures for DEIM is already explained in Chapter 2. We have also described how DEIM is used in our method in Chapter 3 and previous sections in Chapter 4. Please read them again if still confused about how we apply DEIM.

points. Evaluate $p'_{\mathcal{U}}$ in Eqn (4.43).
Also, we extract p_i^t's values on all the DEIM points, thus we obtain $p_{\mathcal{U}}$ by the second equation of Eqn (4.43). Then we can compute $\|p'_{\mathcal{U}} - p_{\mathcal{U}}\|_2$. We use Newton iteration to minimize $\|p'_{\mathcal{U}} - p_{\mathcal{U}}\|_2$.

We use 18 simulations to generate the snapshots, and 26 simulations to generate the stencil samples [33]. An example pressure solution solved by the true physics is plotted in Fig 4-17. We compare BSIM-based ROM with the full simulator. We use 30 principal modes for p, 36 principal modes for R, and 36 DEIM points ($\mathcal{U}$) to obtain the reduced model of the corrected simplified physics.[34] Fig 4-18 shows an example pressure solved by BSIM-based ROM, and Fig 4-19 shows its error against the pressure solved by the true physics.

The error shown in Fig 4-19 has three contributors: POD, DEIM, and surrogate: POD restricts p to $range(V)$; DEIM restricts $\tilde{R}$ (at every timestep) to $range(U)$, and interpolates $\tilde{R}$ by evaluations of $\tilde{R}$ on the DEIM points; the surrogate $\tilde{R}$ is only an approximation for the exact R.

To distinguish the three sources of error, we conduct the following experiment:
First, we compute the error only due to POD approximation of p, i.e.

$$\text{err}_{POD} = \|p - VV^T p\|_2 \,, \tag{4.44}$$

where p is the pressure solved by the true physics. The result is shown in Fig 4-20. we find the POD approximation error contributes about 20% of the total error in the BSIM-based ROM.
Second, we compute the error only due to surrogate, i.e. we set $V = I$, $U = I$, and evaluate

[33] 18 and 26 have no special meaning. At first the author conducted 8 simulations but deleted the snapshots by accident. Then he continued 18 simulation storing both snapshots and stencil samples.
[34] As we have mentioned, the singular values for the snapshot matrix of R has a slower decay than for the snapshot matrix of p, so we want to keep more principal modes for R than for p.

$\tilde{R}$ on all gridpoints (not just on DEIM points). [35]. With these assumptions the only error contributor will be the surrogate $\tilde{R}$, which cannot approximate R exactly. The result is shown in Fig 4-21.

We find the surrogate only contributes about 3% to the error of the BSIM-based ROM. Most of error are contributed from POD and DEIM. In conclusion, we have validated BSIM-based ROM by a 2D elliptic equation. Also, the error of BSIM-based ROM in this example is mostly due to POD and DEIM, whereas the error contributed from the surrogate is relatively small. Therefore, BSIM-based ROM method achieves a black-box model reduction with an accuracy similar to the intrusive POD-DEIM method.

[35] If $\tilde{R} = R$ holds exactly, Eqn (4.36) will be identical to Eqn (4.39), and the pressure obtained by BSIM-based ROM and true physics will be the same (assuming $V = I$, $U = I$, and we evaluate $\tilde{R}$ on all gridpoints).

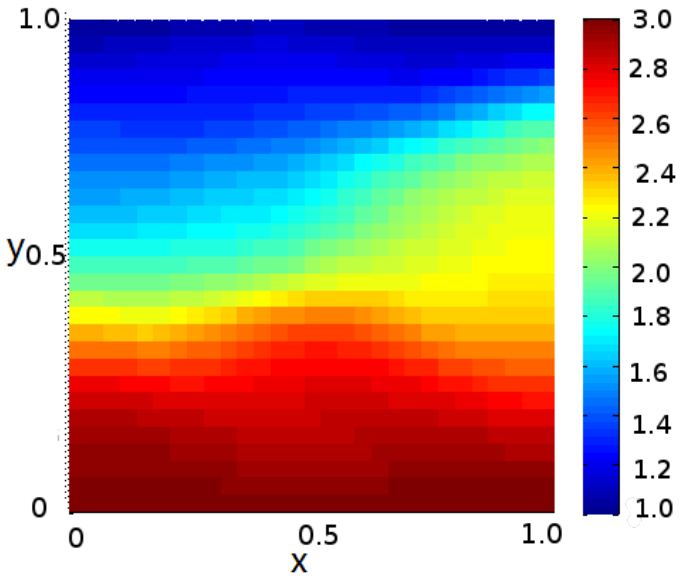


Figure 4-17: Pressure solved by true-physics simulation.

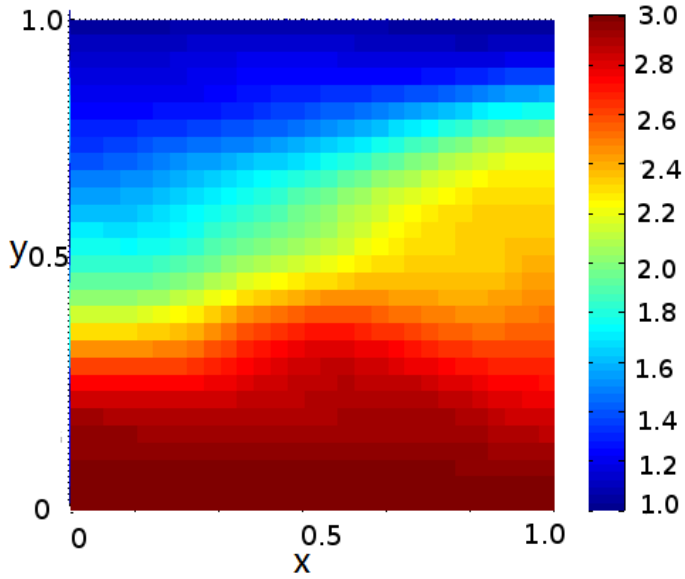


Figure 4-18: Pressure solved by BSIM-based ROM, with 30 principal modes for p, 36 principal modes for R, and 36 DEIM points.

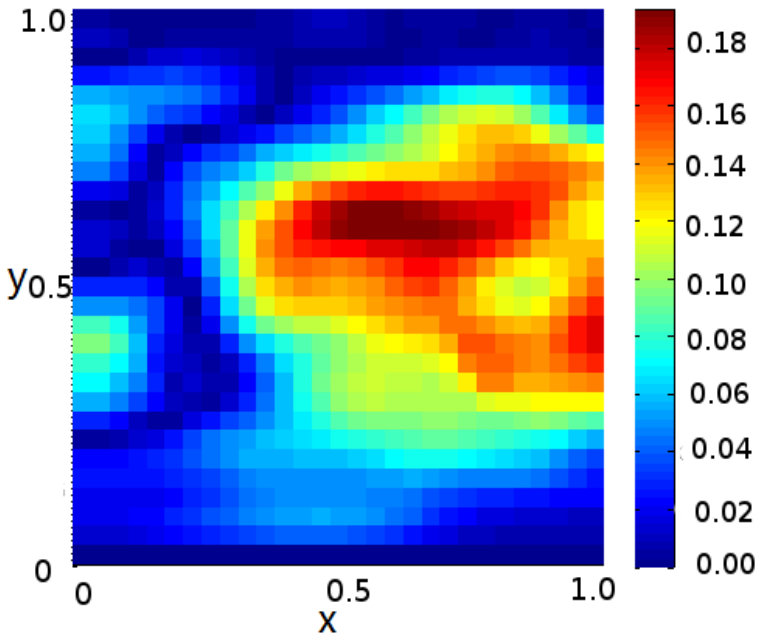


Figure 4-19: Error of the solution from BSIM-based ROM against the solution from true physics simulation.

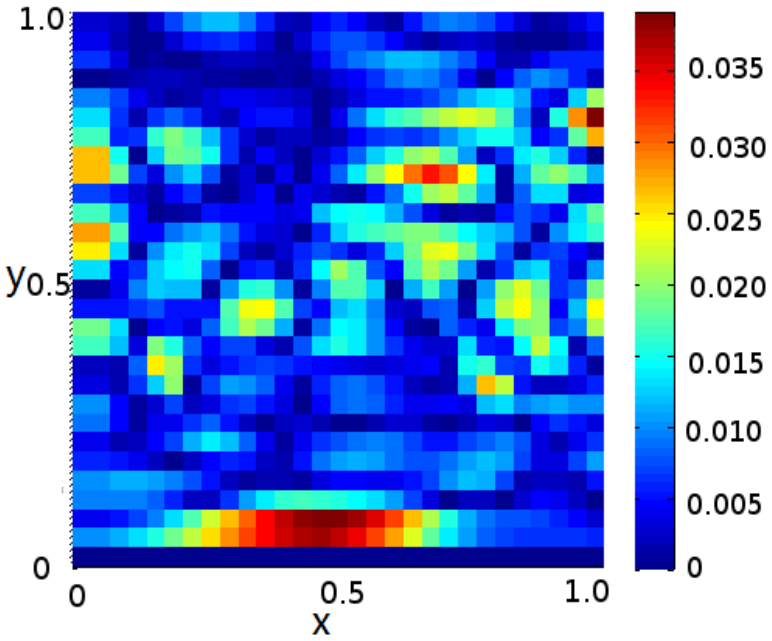


Figure 4-20: Error due to POD approximation.

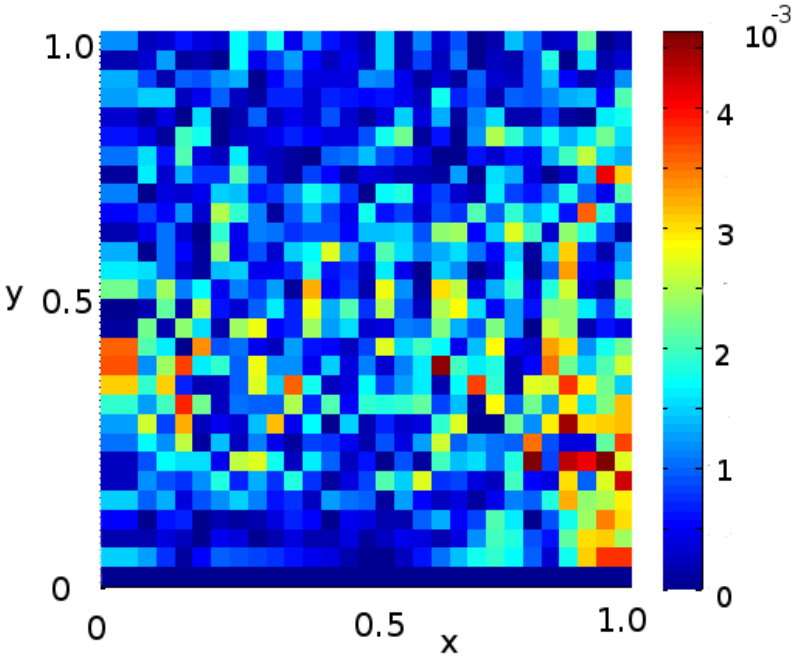


Figure 4-21: Error due to neural network surrogate.

Chapter 5

Conclusions

Model reduction is categorized into intrusive methods and non-intrusive methods. Intrusive model reduction requires modification to the simulation code, which can be cumbersome or even impossible in many circumstances. Non-intrusive model reduction has the advantage of not requiring modifications to the code. But they can suffer severely from the curse of dimensionality. This thesis developed the Black-box Stencil Interpolation Method for model reduction. In BSIM, we use machine learning techniques to infer the underlying physics that governs a simulation. The inferred physics can be combined with existing intrusive model reduction techniques, such as POD and DEIM, to obtain a non-intrusive ROM. Our framework is non-intrusive, but we have demonstrated that the BSIM-based ROM is able to achieve a higher accuracy than conventional non-intrusive ROMs.

The advantages of BSIM-based ROM are: First, BSIM-based ROM is non-intrusive, thus it has a wider applicability than intrusive methods; Second, our method can alleviate the curse of dimensionality suffered by conventional non-intrusive methods [1].

Note, the title of my thesis is Black-box Stencil Interpolation Method *for Model Reduc-*

[1] Our method is designed only to alleviate the suffering from the curse of dimensionality of *local parameters*. The dimensionality of *global parameters* can still be a serious threat to the accuracy of our method. Please read Chapter 3 if you get confused of *global parameters* or *local parameters*.

tion: BSIM is not a model reduction technique, but it enables intrusive model reduction techniques in a non-intrusive environment. It can work independently with POD, DEIM, or model reduction in general. When working independently, BSIM is a method to infer the underlying physics that governs a simulation. More generally, **BSIM is the application of machine learning to inferring the governing physics of a system**. The system can either be a PDE simulation, an experiment, or an observation. Even if no simulation is available, we can still use BSIM to infer the underlying physics that governs the dataset from an experiment.

Therefore, we can imagine BSIM to combine with other intrusive numerical techniques. Even if our working environment is non-intrusive, and even if we refrain ourselves from modifying simulation code, we can still use intrusive numerical techniques on the approximated physics obtained by BSIM. These intrusive techniques include POD and TPWL for model reduction, variational methods for data assimilation, and adjoint methods for optimization, etc [2]. One of the future work can be combining BSIM with these intrusive methods. It is also interesting to explore BSIM-based ROM for various applications like optimal control under uncertainty and inverse design. In conclusion, BSIM enables an array of intrusive methods for non-intrusive problems, providing a competitive alternative to conventional non-intrusive numerical methods.

[2] The author is not familiar with many of the methods, but this does not prevent the author to imagine their combination with BSIM.

Appendix A

Code

A.1 MATLAB Code for 2-D Permeability Generation

```
function K = genK(Nx, Ny, sigx, sigy)
    X = linspace(0,1,Nx);
    Y = linspace(0,1,Ny);
    CovX = exp(-(repmat(X,Nx,1) - repmat(X',1,Nx)).^2 ./ sigx^2);
    CovY = exp(-(repmat(Y,Ny,1) - repmat(Y',1,Ny)).^2 ./ sigy^2);
    [Ux,Dx] = eig(CovX);
    Ux = Ux(:,end:-1:1);  Dx = diag(Dx);  Dx = Dx(end:-1:1);  Dx = Dx(1:Nx);
    [Uy,Dy] = eig(CovY);
    Uy = Uy(:,end:-1:1);  Dy = diag(Dy);  Dy = Dy(end:-1:1);  Dy = Dy(1:Ny);
    K = zeros(Nx,Ny);
    for ii = 1:Nx,
        for jj = 1:Ny,
            K = K + Ux(:,ii) * sqrt( Dx(ii) * Dy(jj) ) * randn(1) * Uy(:,jj)';
        end
    end
    K = exp(0.8*K);
```

$Nx = 30$, $Ny = 30$ is the number of spatial discretization points in the x and y directions. $sigx = 0.15$, $sigy = 0.3$ is the correlation length of permeability in the x and y directions.

Bibliography

[1] C. Audouze, F. De. Vuyst, and P. B. Nair. Reduced-order modeling of parameterized pdes using time-space-parameter principal component analysis. *International Journal for Numerical Methods in Engineering*, 80(1):1025–1057, January 2009.

[2] M. Barrault, Y. Maday, N. C. Nguyen, and A. T. Patera. An 'empirical interpolation' method: application to efficient reduced-basis discretization of partial differential equations. *C.R.Acad.Sci.Paris, Ser.I*, (339), June 2004.

[3] P. Benner, M. Hinze, and E. Jan W. ter Maten. *Model reduction for circuit simulation*. Springer, 2011.

[4] L. T. Biegler, O. Ghattas, M. Heinkenschloss, and B. V. Bloemen Waanders. *Large-scale PDE constrained optimization: an introduction*, volume 30 of *Lecture Notes in Computational Science and Engineering*, pages 3–14. 2003.

[5] C. M. Bishop. *Pattern recognition and machine learning*, chapter 6. Springer, 2006.

[6] R. H. Brooks and A. T. Corey. Hydraulic properties of porous media. *Hydrology Paper*, 3, 1964.

[7] M. A. Cardoso. *Development and application of reduced-order modeling procedures for reservoir simulation*. PhD thesis, Stanford University, 2009.

[8] M. A. Cardoso and L. J. Durlofsky. Linearized reduced-order models for subsurface flow simulation. *Journal of Computational Physics*, 229:681–700, 2010.

[9] S. Chaturantabut and D. C. Sorensen. Discrete empirical interpolation for nonlinear model reduction. *Joint 48th Conference on Decision and Control*, December 2009.

[10] H. Chen, Q. Wang, R. Hu, and P. Constantine. Conditional sampling and experimental design for quantifying manufacturing error of transonic airfoil. *49th AIAA Aerospace Sciences Meeting*, 2011.

[11] K. H. Coats. A note on impes and some impes-based simulation models. *SPE Journal*, 5(3):245–251, 2000.

[12] R. Courant, E. Isaacson, and M. Rees. On the solution of nonlinear hyperbolic differential equations by finite differences. *Comm. Pure Appl. Math.*, 5:243–255, 1952.

[13] L. Daniel, C. S. Ong, S. C. Low, K. H. Lee, and J. White. A multiparameter moment matching model reduction approach for generating geometrically parameterized interconnect performance models. *IEEE transactions on Computer Aided Design of Integrated Circuits and Systems*, 23(5):678–693, May 2004.

[14] T. Desell, D. P. Anderson, M. Magdon-Ismail, H. Newberg, B. K. Szymanski, and C. A. Varela. An analysis of massively distributed evolutionary algorithms. *Proc. IEEE Congress on Evolutionary Computation*, 2010.

[15] Y. Epshteyn and B. Riviere. Fully implicit discontinuous finite element methods for two-phase flow. *Applied Numerical Mathematics*, 57:383–401, June 2006.

[16] G. C. Everstine. *Numerical Soluton of Partial Differential Equations*, chapter 3. lecture notes, 2010.

[17] A. Forrester, A. Sobester, and A. Keane. *Engineering design via surrogate modelling: a practical guide*. Wiley, 2008.

[18] M. Frangos, Y. Marzouk, and K. Willcox. *Surrogate and reduced-order modeling: a comparison of approaches for large-scale statistical inverse problems*. John Wiley and Sons, Ltd, 2001.

[19] D. Galbally, K. Fidkowski, K. Willcox, and O. Ghattas. Non-linear model reduction for uncertainty quantification in large-scale inverse problems. *International Journal for Numerical Methods in Engineering*, 81(12):1581–1608, March 2010.

[20] K. C. Giannakoglou, Th. I. Pappou, A. P. Giotis, and D. G. Kouboginnis. A parallel inverse-design algorithm in aeronautics based on genetic algorithms and the adjoint method. *European Congress on Computational Methods in Applied Sciences and Engineering*, 2000.

[21] Eric James Grimme. Krylov projection methods for model reduction. Technical report, 1997.

[22] K. Hvistendahi Karlsen, K. A. Lie, and N. H. Risebro. A fast marching method for reservoir simulation. *Computational Geosciences*, 4:185–206, 2000.

[23] B. V. Leer. Towards the ultimate conservative difference scheme, ii. monotonicity and conservation combined in a second order scheme. *J. Comp. Phys.*, 14:361–370, 1974.

[24] Y. C. Liang, W. Z. Lin, H. P. Lee, S. P. Lim, and K. H. Lee. Proper orthogonal decomposition and its applications - part ii: model reduction for mems dynamical analysis. *Journal of Sound and Vibration*, 256(3):515–532, 2002.

[25] L. Mohamed, M. Christie, V. Demyanov, and H. Watt. History matching and uncertainty quantification: multiobjective particle swarm optimisation approach. *Society of Petroleum Engineers*, pages 305–318, May 2011.

[26] A. W. Moore and T. Hall. *Efficient Memory-based Learning for Robot Control*. PhD thesis, University of Cambridge, 1991.

[27] B. C. Moore. Principal component analysis in linear systems, controllability, observability, and model reduction. *IEEE Transactions on Automatic Control*, 26(1), February 1981.

[28] I. J. Myung. Tutorial on maximum likelihood estimation. *Journal of Mathematical Psychology*, 47:90–100, 2003.

[29] D. Papadopoulos, M. Herty, V. Rath, and M. Behr. Identification of uncertainties in the shape of geophysical objects with level sets and the adjoint method. *Computational Geosciences*, 15:737–753, July 2011.

[30] S. Parker. *A component-based architecture for parallel multi-physics PDE simulations*, volume 2331 of *Lecture Notes in Computer Science*, pages 719–734. Springer Berlin / Heidelberg, 2002.

[31] L. Pernebo and L. Silverman. Model reduction via balanced state space representations. *IEEE transactions on Automatic Control*, 27:382–387, April 1982.

[32] M. J. Rewienski. *A trajectory piecewise linear approach to model order reduction of nonlinear dynamical systems*. PhD thesis, Massachusetts Institute of Technology, 2003.

[33] C. W. Rowley, T. Colonius, and R. M. Murray. Model reduction for compressible flows using pod and galerkin projection. *Physica D*, 189:115–129, May 2004.

[34] W. Schilder, H. V. D. Vorst, and J. Rommes. *Model order reduction: theory, research aspects, and applications*. springer, 2009.

[35] L. Sirovich. Turbulence and the dynamics of coherent structures, part i: coherent structures. *Quarterly of Applied Mathematics*, (3):561–571, October 1987.

[36] K. Willcox and J. Peraire. Balanced model reduction via the proper orthorgonal decomposition. *AIAA Journal*, 40(11), November 2002.