Abstract

Parameter identification problems are frequently occurring within biomedical applications. These problems are often ill-posed, and thus challenging to solve numerically. Previously, it has been suggested that minimization of the Tikhonov functional using a time-adaptive finite element method could be useful for determining the drug efficacy for treatment of HIV infection. In this thesis, the suggested method was implemented and numerically tested in MATLAB. The results, however, suggested that the method might be unsuitable for these kinds of problems; instead, elementary methods were found to be more plausible.
The methods presented in the thesis can eventually be used by clinicians to determine the drug-response for each treated individual. The exact knowledge of the personal drug efficacy can aid in the determination of the most suitable drug as well as the most optimal dose for each individual, in the long run resulting in a personalized treatment with maximum efficacy and minimum adverse drug reactions.

Sammanfattning

Parameteridentifieringsproblem är vanligt förekommande inom biomedicinska tillämpningar. Dessa är ofta illaställda, och därmed svåra att lösa numeriskt. I ett tidigare arbete föreslogs att minimering av Tikhonovfunktionalen med hjälp av en tidsadaptiv finita elementmetod kunde användas för att identifiera läkemedelseffektiviteten vid behandling av HIV-infektion. I detta arbete har den föreslagna metoden implementerats och testats numeriskt i MATLAB. Resultaten antydde emellertid att metoden är olämplig för denna typ av problem; istället visade sig elementära metoder vara mer lämpade.
Metoderna som presenteras i denna uppsats skulle i framtiden kunna användas av läkare för att bestämma det individuella läkemedelssvaret för varje patient. Detta skulle i det långa loppet kunna leda till en personlig behandling med maximal effektivitet och minimal incidens av biverkningar.

Acknowledgments

First and foremost, I want to sincerely thank associate Professor Larisa Beilina, who has allowed me to do this project, and been wisely guiding me through it. She has been the most friendly and encouraging supervisor imaginable.
I also want to thank my family from the bottom of my heart, for the financial support that has made my studies possible - thank you so much for believing in me. And finally, my friends, and in particular my very special friend Johanna Hörberg, who has had patience with my absent-mindedness when writing my thesis, deserve my heartfelt gratitude.

Contents

Abbreviations

CGM = conjugate gradient method
FEM = finite element method
HIV = human immunodeficiency virus
LHS = left hand side
ODE = ordinary differential equation
PIP = parameter identification problem
RHS = right hand side
RT = reverse transcriptase
SVD = singular value decomposition

Index of notation

Since I do not strive to be possible to understand, but rather impossible to misunderstand, I think it is appropriate to clarify the exact meaning of certain symbols used in this thesis.

Symbol	Meaning
$\subseteq$	... is a subset of ...
$\subset$	... is a proper subset of ...
$\mathbb{N}$	$\{1,2,3, ...\}$.
$R(F)$	Range of F.
$D(F)$	Domain of F.
$B_\varepsilon[p]$	$\{x : \|\|x-p\|\| \leq \varepsilon\}$.
$\mathcal{C}(\Omega)$	Continuous functions defined on Ω.
$\mathcal{C}^k$	Space of k times continuously differentiable functions.
$\mathcal{C}_0^\infty$	Space of compactly supported smooth functions.
$\mathcal{P}^k$	Space of k'th degree polynomials.
$\mathcal{S}_k^r$	Spline space: $\{s \in \mathcal{C}^r : s\|_\tau \in \mathcal{P}^k\}$.
$\mathcal{S}^k$	Same as $\mathcal{S}_k^r$ with $r = k-1$.
$H^1(\Omega)$	$W_2^1(\Omega) = \{u \in L_2(\Omega) : \frac{\partial u}{\partial x_j} \in L_2(\Omega), j = 1, ..., n\}$, with corresponding Sobolev norm.
$L_2(\Omega)$	Space of square integrable functions on Ω, with scalar product $(f,g) = \int_\Omega f\bar{g}dx < \infty$.

Chapter 1

Introduction

Inverse problems are of great importance in many applications, but unfortunately, they are often ill-posed and therefore regularization methods are required for their numerical solution. To solve these problems efficiently, adaptive finite element methods (FEM) for coefficient inverse problems [2, 5] and parameter identification problems [13, 15, 16] have been developed. In this thesis we will test the adaptive FEM method suggested in [3, 4] to solve systems of initial value ODE problems, in particular for identifying the drug efficacy parameter in a model of HIV infection under treatment of a reverse trancriptase (RT) inhibitor.

The thesis is organized as follows. Chapter 2 gives the reader an introduction to ill-posed problems and Tikhonov regularization, and also a brief review of the curve fitting techniques that are used in this work. In Chapter 3, the model problem is presented, and it is argued that Tikhonov regularization is applicable to this problem; furthermore, the Lagrangian corresponding to the problem is defined and the adjoint problem is derived. In Chapter 4, the numerical methods used in the thesis are presented: Newtons method, for solving the forward and adjoint problems; the conjugate gradient method (CGM) for finding the minimum of the Tikhonov functional; and numerical differentiation, which is required for direct solution of the inverse problem. In Chapter 5 the computational results are presented, demonstrating the possibility of the reconstruction of drug efficacy for different number of observations in time, and different noise levels at measured data, using only elementary methods. Finally, in Chapter 6 we discuss the results and their utility in clinical practice. The appendices contain an introduction to the mechanism of HIV infection and the MATLAB programs used in our numerical simulations.

Chapter 2

Preliminaries

2.1 Inverse problems and ill-posedness

Let us consider the following problem:
Let B_1 and B_2 be Banach spaces. Let $G \subseteq B_1$ be an open set in B_1 and $F : G \to B_2$ an operator. Let $y \in B_2$ be given, and suppose we want to find $x \in G$ such that

$$F(x) = y. \tag{2.1}$$

Problems of this sort, when you want to identify x in (2.1), given observations, y, are called *inverse problems*. A special class of inverse problems are called parameter identification problems (PIP), i.e. x is some parameter of a differential equation, and $F(x)$ is the solution of the differential equation, with this parameter.

Definition 1. Problem (2.1) is said to be well-posed by Hadamard if it satisfies the following conditions [18]:

1. **Existence:** For each $y \in B_2$ there is an $x = x(y)$ such that $F(x) = y$.
2. **Uniqueness:** For each $y \in B_2$ there is not more than one $x = x(y)$ such that $F(x) = y$.
3. **Stability:** For each y such that a unique solution of (2.1) exists, the solution $x = x(y)$ is a continuous function of y [1].

Definition 2. Problem (2.1) is said to be ill-posed if it is not well-posed.

PIP and other inverse problems are often ill-posed. Ill-posedness means that it is difficult to solve (2.1) numerically, since measurement errors, or even errors induced by finite-precision computer arithmetic, can have disastrous consequences. Let y^* denote noiseless observations,

[1]We will call a problem *well-conditioned* if it is well-posed and such that a small change in data results in a small change in the solution. A problem may be well-posed but still ill-conditioned; even if $x(y)$ is continuous it might still be very sensitive to changes in y.

$\delta > 0$ be the noise level, and $B_\delta[y^*] = \{y : ||y - y^*||_{B_2} \leq \delta\}$. The solution to the slightly perturbed equation $F(x) = y_\delta$ (with $y_\delta \in B_\delta[y^*]$) could be entirely different from the solution to $F(x) = y^*$. Perhaps a solution to $F(x) = y_\delta$ does not even exist. No matter how small δ is. A generally ill-posed problem (2.1) can be well-posed if we consider the restriction of F in (2.1) to certain subsets of its domain.

Definition 3 (Conditional well-posedness)**.** Let B_1 and B_2 be Banach spaces. Suppose $G \subset B_1$ is the closure of an open subset in B_1. Let $F : G \to B_2$ be a continuous operator. Assume that $y^* \in F(G)$ is our ideal noiseless data, and pick a noise level $\delta > 0$. Suppose we want to solve

$$F(x) = y_\delta, \tag{2.2}$$

where $y_\delta \in B_\delta[y^*]$. This problem is called *conditionally well-posed on* G if it satisfies the following conditions [18]:

1. **Existence:** It is *a priori* known[2] that there exists an ideal solution $x^* = x^*(y^*) \in G$ for an ideal noiseless data y^*.

2. **Uniqueness:** The operator $F : G \to B_2$ is one-to-one.

3. **Stability:** The inverse operator F^{-1} is continuous on $F(G)$.

Definition 4. The set G in Definition 3 is called the *correctness set* of the problem (2.2).

Continuity of the inverse operator F^{-1} can be guaranteed if the domain of F is compact. Hence, any compact set with nonempty interior such that F is one-to-one is a correctness set. This suggests a method to solve (2.2) by choosing a suitable correctness set, G, and then finding a $x \in G$ such that $||F(x) - y_\delta||$ is as small as possible.

Theorem 1 (Tikhonov [29])**.** *Let B_1 and B_2 be Banach spaces, and $U \subset B_1$ a compact set. Let $F : U \to B_2$ be a continuous one-to-one operator and $V = F(U)$. Then $F^{-1} : V \to B_1$ is a continuous operator.*

For a proof of this important theorem see, for example [7, 29].

2.1.1 Quasi-solution

Let H_1 and H_2 be Hilbert spaces[3], and assume that $F : G \to H_2$ is a continuous mapping defined on a compact correctness set, $G \subset H_1$. Let $\delta > 0$ and assume, as before, that we want to solve

$$F(x) = y_\delta, \tag{2.3}$$

[2]The rationale behind this is the assumption that the problem to be solved is a model of some natural phenomenon. And since the phenomenon apparently exists, a solution of the equation describing it must also exist. At least if we assume that the natural phenomenon in question is exactly represented by the mathematical model.

[3]We require that the spaces are Hilbert spaces, rather than arbitrary Banach spaces in order for the closest point property to hold. However, it suffices that they are so-called reflexive Banach spaces, see [12].

with y_δ defined as before. We know that a solution exists for perfect data y^*, but in general (2.3) has no solution, since $y_\delta \notin F(G)$ (implying that we are dealing with an ill-posed problem). Our goal in this, and the following subsection is to sketch how to construct a family of approximate solutions $\{x_\delta\}$ in G that converges to x^* as $\delta \to 0$. Let us define

$$J_{y_\delta}(x) = ||F(x) - y_\delta||^2_{B_2}. \tag{2.4}$$

Since F is continuous, it takes compact sets to compact sets, thus $F(G)$ is compact in B_2. And since $F(G)$ is a compact subset of a Hilbert space, and therefore closed, a minimum of (2.4) exists (and if $F(G)$ also happens to be convex, this minimum is unique). Any $x \in G$, unique or not, that minimizes J_{y_δ} in (2.4) is called a *quasi-solution* to (2.3).
Since the inverse mapping, F^{-1}, is continuous by Theorem 1 and defined on a compact metric space, it admits a modulus of continuity, $\omega_{F^{-1}}$[4] From Theorem 1.5 in [7] it follows that, given $y_\delta \in B_2$, then for any $x^q_\delta \in \arg\min_{x \in G} J_{y_\delta}(x)$ the following error estimate holds:

$$||x^q_\delta - x^*||_{B_1} \leq \omega_{F^{-1}}(2\delta), \tag{2.6}$$

where $\omega_{F^{-1}}(z)$ is the modulus of continuity of the inverse operator F^{-1}. Thus $x^q_\delta \to x^*$ as $\delta \to 0$. Hence, we can take a sequence of quasi-solutions to be our desired family.
However, sometimes the set of all plausible solutions, to (2.3) does not form a compact set. In these cases, the inverse mapping F^{-1} need not be continuous on the image of the plausible solutions $F(G)$, and the minimum of (2.4) may not exist. Such problems are called essentially ill-posed.
Furthermore, even if the set of plausible solutions is compact, there is no guarantee that the minimum of (2.4) is unique (unless G is also convex), and even if the global minimum is unique, there might exist local minima, or regions where the gradient of the functional is very small, such that a minimization algorithm could get trapped. In the next subsection, we will discuss how a stable solution to essentially ill-posed problems could be obtained in practice.

2.1.2 The Tikhonov functional

The Tikhonov functional makes sure that when minimizing (2.4), we will stay in the neighbourhood of some point, x_0, which is *a priori* known to be close to the true solution, x^*. A general Tikhonov functional is given below

$$J_\gamma(x) = \frac{1}{2}||F(x) - y||^2_{B_2} + \frac{\gamma}{2}||x - x_0||^2_{B_1}. \tag{2.7}$$

[4]A modulus of continuity is a function $\omega : [0, \infty] \to [0, \infty]$ such that

$$\lim_{x \to 0^+} \omega(x) = \omega(0) = 0. \tag{2.5}$$

The function f admits ω as a modulus of continuity if and only if $||f(x) - f(y)|| \leq \omega(||x - y||)$. In particular, f has a modulus of continuity if and only if it is uniformly continuous.

The first term is essentially the same as in (2.4), the second term is the regularization term and $\gamma := \gamma(\delta)$ is the regularization parameter.
In this subsection, we will show that minimization of the Tikhonov functional is a powerful tool for solving many ill-posed problems. We will start by proving that there exists a minimizing sequence. The proposition below is conceptually equivalent to the construction of the minimizing sequence in Section 1.7 of [7], except that we do not require that the set Q is a proper dense subset of H_1. The proposition stays true with the same proof if H_1 and H_2 are Banach spaces, but let us stick to Hilbert spaces from now on. We say that a subspace $A \subseteq B$ is compactly embedded in B, if the embedding operator is a compact operator.

Proposition 1. *Let H_1 and H_2 be Hilbert spaces. Let $Q \subseteq H_1$ be a compactly embedded subspace of H_1 with nonempty interior, and assume that $G \subseteq Q$ is a closed set with nonempty interior and no isolated points. Let $F : G \to H_2$ be a one-to-one operator that is continuous in the norms of H_1 and H_2. Assume that $F(x^*) = y^*$ for some $x^* \in G$ and $y^* \in H_2$. Let $\{\delta_k\}_{k=1}^{\infty}$ be a sequence of noise levels such that $\delta_k > 0$ $\forall k$ and $\delta_k \to 0$ as $k \to \infty$. Let $\gamma : (0, \infty) \to (0, \infty)$ be a function such that*

$$\delta_k \to 0 \implies \gamma(\delta_k) \to 0 \wedge \frac{\delta_k^2}{\gamma(\delta_k)} \to 0. \tag{2.8}$$

Define the Tikhonov functional as

$$J_k(x) = \frac{1}{2}||F(x) - y_{\delta_k}||_{H_2}^2 + \frac{\gamma(\delta_k)}{2}||x - x_0||_Q^2, \tag{2.9}$$

where $||y_{\delta_k} - y^*|| < \delta_k$, and $x_0 \in G$. Then there exists a sequence, $\{x_k\}$, corresponding to $J_k(x)$, such that $x_k \to x^*$ as $k \to \infty$.

Proof. For y such that $||y - y^*|| < \delta_k$, we obtain

$$J_k(x^*) = \frac{1}{2}||y^* - y||_Q^2 + \frac{\gamma(\delta_k)}{2}||x^* - x_0||_Q^2 \leq \frac{\delta_k^2}{2} + \frac{\gamma(\delta_k)}{2}||x^* - x_0||_Q^2. \tag{2.10}$$

Let

$$m_k = \inf_{x \in G} J_k(x), \tag{2.11}$$

by (2.10)

$$m_k \leq J_k(x^*) \leq \frac{\delta_k^2}{2} + \frac{\gamma(\delta_k)}{2}||x^* - x_0||_Q^2, \tag{2.12}$$

then there exists an x_k such that

$$m_k \leq J_k(x_k) \leq \frac{\delta_k^2}{2} + \frac{\gamma(\delta_k)}{2}||x^* - x_0||_Q^2. \tag{2.13}$$

Using (2.13) in (2.9) for $x = x_k$ yields

$$\frac{1}{2}||F(x_k) - y^*||_{H_2}^2 + \frac{\gamma(\delta_k)}{2}||x_k - x_0||_Q^2 \leq \frac{\delta_k^2}{2} + \frac{\gamma(\delta_k)}{2}||x^* - x_0||_Q^2. \tag{2.14}$$

This implies that for all $\delta_k > 0$ such that $\lim_{k\to\infty} \gamma(\delta_k)$ and $\frac{\delta_k^2}{\gamma(\delta_k)} \to 0$ we have

$$||x_k - x_0||_Q^2 \le \frac{\delta_k^2}{\gamma(\delta_k)} + ||x^* - x_0||_Q^2 \to ||x^* - x_0||_Q^2. \tag{2.15}$$

Thus, $\{x_k\}$ is bounded in $G \subseteq Q$. Since Q is compactly embedded in H_1, there exists a subsequence to $\{x_k\}$ that converges in $(H_1, ||\cdot||_{H_1})$ (without loss of generality, and for ease of notation, we can assume that it is $\{x_k\}$ itself). And since all $x_k \in G$ and G is closed, it must actually converge in G. Assume that it converges to $\hat{x} \in G$. Then, since $y_{\delta_k} \to y^*$ and $\gamma(\delta_k) \to 0$ as $k \to \infty$

$$\lim_{k\to\infty} J_k(x_k) = \frac{1}{2}||F(\hat{x}) - y^*||_{H_2}^2, \tag{2.16}$$

but from (2.13) it follows that

$$\lim_{k\to\infty} J_k(x_k) = 0, \tag{2.17}$$

thus

$$\frac{1}{2}||F(\hat{x}) - y^*||_{H_2}^2 = 0, \tag{2.18}$$

and since we assumed that F was one-to-one we can conclude that $\hat{x} = x^*$. $\square$

The regularization parameter satisfying conditions (2.8) can be chosen as, for instance

$$\gamma(\delta) = \delta^{2\mu}, \tag{2.19}$$

where $\mu \in (0, 1)$. However, the proposition does not tell us how to find a minimizing sequence in practice. So, assume that we have a single noise level and our goal is to minimize (2.7). Assume the same conditions as in the previous proposition, and let γ be defined as above (2.19). Choose a $\xi \in (0, 1)$, then it can be proven [17] that there exists a δ_0 such that

$$\delta \in (0, \delta_0) \implies ||x_k - x^*|| \le \xi ||x_0 - x^*||, \tag{2.20}$$

in particular it follows that if (2.7) attains a minimum, any $x \in \arg\min J(x)$ would be a better approximation to x^* than x_0, if the noise level is small enough.
In general, the Tikhonov functional (2.7) might not actually attain its infimum; we can only guarantee the existence of the minimizing sequence, $\{x_k\}$. However, without loss of generality, we can assume that G is the closure of an open and bounded set containing the initial guess, x_0, the (bounded) minimizing sequence, $\{x_k\}$, and the exact solution, x^*. Hence, if we consider finite dimensional Hilbert spaces, the Tikhonov functional, defined on G, would have a minimum, since closed and bounded sets on finite dimensional Hilbert spaces are compact, and functionals defined on compact sets attain their infimum according to the Weierstrass' extreme value theorem. In numerical mathematics, we always work in finite dimensional spaces, so in practice (2.7) always has a minimum if the initial guess is contained in G.

Suppose now that G is convex and that (2.7) is Fréchet differentiable[5], with a Fréchet derivative that is uniformly bounded and Lipschitz continuous. Then one can prove [7, 8] that for given noise level and regularization parameter (2.7) is locally strongly convex in a neighbourhood of its minimum and that x^* is also contained in this neighbourhood if $||x^* - x_0||$ is small enough. Thus, if x_0 is chosen properly, the unique zero of the Fréchet derivative of (2.7) is its global minimum.
Thus, to sum up, under reasonable assumptions discussed above, a minimum of (2.7) exists and is a better approximation than the starting guess, x_0. And under reasonable assumptions, and if the initial guess is good enough, there is only one unique point that is the global minimum, and we do not need to worry about local minima. These facts explain why Tikhonov regularization is so useful for solving ill-posed problems.
To find the zero of the Fréchet derivative, one can use common minimization techniques, such as the conjugate gradient method (CGM) or the method of steepest descent.
Obviously, the minimum of the Tikhonov functional will not be exactly the same as the quasi-solution if the noise level and regularization parameter are constants. On the other hand, by letting γ decrease for each iteration of the minimization algorithm we will have a minimum of the Tikhonov functional that approaches the quasi-solution. In [1] it was suggested that γ can be updated as

$$\gamma_k = \frac{\gamma_0}{(k+1)^p}, \tag{2.22}$$

where $p \in (0, 1]$ and $k = 0, 1, 2, ...$
From what have been discussed, it is clear that a good first guess is essential for successful identification of the desired parameter. Of course, we do not in general have any idea at all what the solution to (2.1) might be, and therefore, in general, we need to devise some kind of globally convergent algorithm to solve ill-posed PIP. This is beyond the scope of this thesis, however. But we will discuss how to obtain a reasonable first guess for this particular problem in later chapters.

2.2 Curve fitting

In this section we will briefly review the curve fitting techniques used in the thesis.

2.2.1 Numerical solution of linear least squares problems

By the Stone-Weierstrass theorem, any continuous function defined on a closed interval of the reals, $[a, b]$ may be arbitrarily closely approximated by a polynomial. And by Taylor's

[5]A continuous linear operator between Banach spaces $A : B_1 \to B_2$ is called the Fréchet derivative of the operator $T : B_1 \to B_2$ at $x \in B_1$ if

$$\lim_{||h||\to 0} \frac{||T(x+h) - T(x) - Ah||}{||h||} = 0. \tag{2.21}$$

theorem any k times differentiable function can be approximated by a polynomial of degree $k-1$ or less in the neighbourhood of some point, a, with an error of at most $R_a(x) = \frac{f^k(\xi)}{k!}(x-a)^k$ with $\xi \in (\min\{x,a\}, \max\{x,a\})$ (these well-known facts can be found in any textbook on analysis, such as [25]).
These theoretical results suggest that polynomials are very versatile for fitting function graphs to sets of data, which they indeed are. The typical first step to fit a polynomial, $p(x) = c_1 + c_2 x + ... + c_n x^{n-1}$ to data, $y = (y_1, y_2, ..., y_m)^T$, by linear least squares, is to construct the so-called Vandermonde matrix

$$A = \begin{bmatrix} 1 & x_1 & x_1^2 & x_1^3 & \dots & x_1^{n-1} \\ 1 & x_2 & x_2^2 & x_2^3 & \dots & x_2^{n-1} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_m & x_m^2 & x_m^3 & \dots & x_m^{n-1} \end{bmatrix}, \tag{2.23}$$

of the time-mesh $x_1 < x_2 < ... < x_m$, and then consider finding $c = (c_1, c_2, ..., c_n)^T$ such that

$$Ac = y, \tag{2.24}$$

with $y = (y_1, y_2, ..., y_m)^T$, which in general has no solution, since the system is usually purposefully overdetermined in order to smooth out noise or inaccurate measurements. Therefore we want to find the solution c that is, in some sense, closest to solving (2.24); more precisely, we want to minimize:

$$f(c) = ||Ac - y||_2^2, \tag{2.25}$$

where $y = (y_1, y_2, ..., y_m)^T$ are the observations, and $c = (c_1, c_2, ..., c_n)^T$ are the coefficients of the fitting polynomial.
The most common methods, which can be used to minimize (2.25), are the method of normal equations, QR-factorization and singular value decomposition (SVD) [6, 11]. The most accurate, but also most computationally demanding method is SVD.
The problem with the solution of the minimization problem $\min_c ||Ac-y||_2^2$, using the Vandermonde matrix, is that as the degree of the polynomial increases, the Vandermonde matrix quickly becomes very ill-conditioned, and eventually so ill-conditioned that it is indistinguishable from a singular matrix within machine precision. As a rule-of-thumb one can fit polynomials up to degree 18 by using the Vandermonde matrix and standard techniques for solving linear least squares, such as QR factorization or SVD; beyond that, erratic results are obtained due to ill-conditioning of the Vandermonde matrix [6].
Various methods have been suggested to deal with matrices so severely ill-conditioned that even SVD fails, including methods based on Tikhonov regularization, or various iterative methods [30]. But since the Vandermonde matrix might be ill-conditioned beyond working precision, not even these sophisticated methods will do if you are required to fit a very high degree polynomial to obtain reasonable accuracy. Furthermore fitting a high degree polynomial to data frequently results in undesirable oscillating behaviour of the interpolant, known as Runge's phenomenon (Figure 2.1).

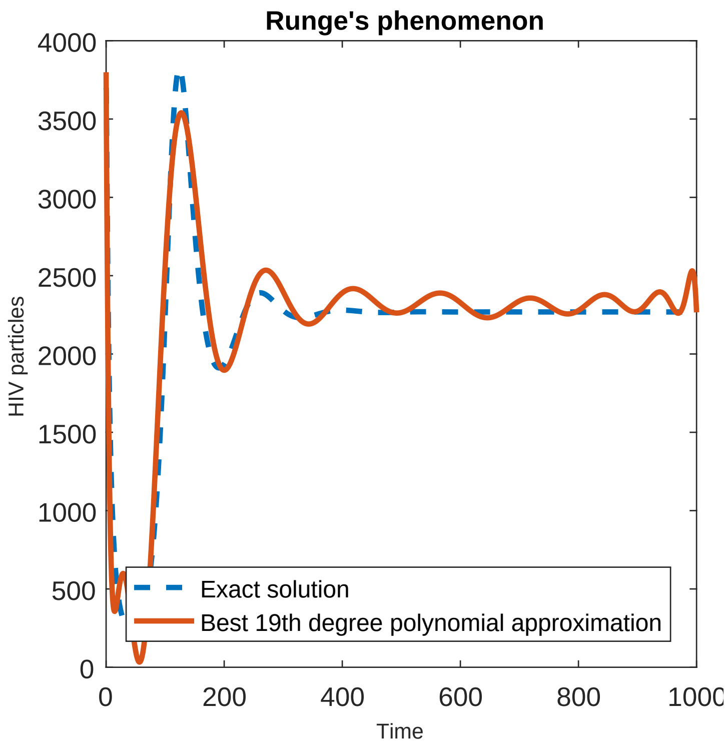


Figure 2.1: Runge's phenomenon exhibited by the model problem (3.1) considered in Chapter 3 of this thesis. The figure shows how the number of viruses changes over time under treatment of a RT-inhibitor with efficacy $\eta = 0.7$.

To overcome these pitfalls, it is often better to use piecewise polynomials of low degree, so-called spline functions, rather than a single high degree polynomial. Splines of degree n are functions in $\mathcal{C}^{n-1}$, which are also piecewise $\mathcal{P}^n$; we will denote the set of all spline functions as $\mathcal{S}^n$. The basis functions, B-splines, for $\mathcal{S}^n$ have bounded support, and hence unlike the Vandermonde matrix, the coefficient matrix of the B-splines is a (well-conditioned) band matrix. Another advantage with B-splines over the Vandermonde matrix is that, whereas a single outlier in the data could have potentially huge impact on the interpolant if the Vandermonde matrix is used, outliers will only affect points in the neighbourhood of the outlier if B-splines are used, due to their bounded support.

2.2.2 Interpolating splines and smoothing splines

The accuracy when fitting splines to data points increases as the number of B-splines used increases. If you have n data points, you can at most use $n+2$ cubic B-splines. If you use the maximum number of B-splines, the function will pass exactly through each of the data points. However, since the data obtained is typically contaminated with noise in most practical situations, a curve exactly fitted to the data points would actually not be a particularly good approximation to the true noiseless data; one would say that the curve is overfitted to the data. To avoid overfitting, we may use regularization techniques to add a penalty term that forces the interpolant to be a sufficiently smooth function.

Cubic smoothing splines

Assume that we have noisy observations, $y_1, y_2, ..., y_m$ of an unknown function, $f : \mathbb{R} \to \mathbb{R}$, at the points $x_1, x_2, ..., x_m$. The least squares problem is to find the $c = (c_1, c_2, ..., c_n)$ that best fits the data

$$\min_c \sum_{i=1}^{m} [y_i - \hat{f}(x_i, c)]^2, \tag{2.26}$$

where $\hat{f}(x, c) = c_1\phi_1(x) + c_1\phi_1(x) + ... + c_n\phi_n(x)$, and ϕ_j are cubic B-splines. That is, to find the $\hat{f} \in \mathcal{S}^3$ that is closest to f.
Cubic smoothing splines are regularized, such that the cubic spline function, $\hat{f}$, has a bounded second derivative (indicating that there are not too many abrupt changes in the curvature of the function graph). More formally, our goal is to minimize the functional $J : \mathcal{S}^3 \to \mathbb{R}$ defined as:

$$J(c) = \sum_{i=1}^{m} [y_i - \hat{f}(x_i, c)]^2 + \lambda \int_{x_1}^{x_m} \hat{f}''(x, c)^2 dx, \tag{2.27}$$

that is, find c_0 such that

$$c_0 = \arg\min_{c \in \mathbb{R}^n} J(c). \tag{2.28}$$

When the parameter $\lambda = 0$, we have just ordinary interpolating splines, and when $\lambda \to \infty$ we obtain linear regression. Typically, it is best when λ is a small number, just sufficient to avoid overfitting, but still obtaining a good fit to the data points.

Chapter 3

The model problem

Our model problem, which was developed in [27], is the system of ODE given by:

$$\begin{cases} \dot{u}_1 = s - ku_1u_4 - \mu u_1 + (\eta\alpha + b)u_2, \\ \dot{u}_2 = ku_1u_4 - (\mu_1 + \alpha + b)u_2, \\ \dot{u}_3 = (1-\eta)\alpha u_2 - \delta u_3, \\ \dot{u}_4 = N\delta u_3 - cu_4, \end{cases} \tag{3.1}$$

where u_1 represents uninfected T cells, u_2 – pre-RT infected T cells, u_3 – post-RT infected T cells, and u_4 virus (see Appendix A for explanation of the terms).

Parameter	Value	Units	Description
s	10	$mm^{-3}day^{-1}$	inflow rate of T cells
μ	0.01	day^{-1}	natural death rate of T cells
k	0.000024	$mm^{3}day^{-1}$	interaction-infection rate of T cells
μ_1	0.015	day^{-1}	death rate of infected cells
α	0.4	day^{-1}	transition rate from pre-RT infected T cells class to post-RT class
b	0.05	day^{-1}	reverting rate of infected cells return to uninfected class
δ	0.26	day^{-1}	death rate of actively infected cells
c	2.4	day^{-1}	clearance rate of virus
N	1000	$virions/cell$	total number of viral particles produced by an infected cell

Table 3.1: Table of parameters.

Let the time domain considered in our problem be denoted as

$$\Omega_T = [0, T] \tag{3.2}$$

Let us assume that the parameter η in (3.1), corresponding to the drug efficacy, is unknown, but contained in the set of admissible functions, M_η:

$$M_\eta = \{\eta \in \mathcal{C}(\Omega_T) : t \in \Omega_T \implies \eta(t) \in (0,1), t \notin \Omega_T \implies \eta(t) = 0\}, \tag{3.3}$$

furthermore, we will assume that all the other parameters, $\{s, \mu, k, \mu_1, \alpha, b, \delta, c, N\}$, are known and defined as in Table 3.1, and that the solution to (3.1) in a certain subset, Ω_{obs}, of the entire time domain, Ω_T, is known through noisy observations, g.

Inverse Problem 1. Assume that all parameters, $\{s, \mu, k, \mu_1, \alpha, b, \delta, c, N\}$, in the model problem (3.1) are known, and defined as in Table 3.1. Find $\eta(t) \in M_\eta$, satisfying (3.3), assuming that the following function is known

$$u(t) = g(t), t \in \Omega_{obs} \subseteq \Omega_T. \tag{3.4}$$

The function $g(t)$ represents observations of the function $u(t)$, which solves (3.1), inside the observation set Ω_{obs}.

Remark 1. It is easy to see that M_η is a convex set: take $\alpha \in [0,1]$ and $\eta_1, \eta_2 \in M_\eta$. Set $c(t) = \alpha\eta_1(t) + (1-\alpha)\eta_2(t)$, then for each $t \in \Omega_T$ it is obvious that $0 < \min\{\eta_1(t), \eta_2(t)\} \leq c(t) \leq \max\{\eta_1(t), \eta_2(t)\} < 1$. That is $c(t) \in M_\eta$.

3.1 Existence, uniqueness and stability of forward problem

According to the discussion in the previous chapter, Tikhonov regularization is useful for solving a PIP if the forward problem is well-posed by Hadamard, and the set of possible parameters is a convex set. In this section we will prove that the required conditions are fulfilled. But let us first review and elaborate on the results from [27].

3.1.1 Existence and Lyapunov stability of steady state

Let us now assume that the parameter η in system (3.1) is constant. That is, $\eta(t) \equiv c \in (0,1)$. Setting the LHS in (3.1) to zero and solving for u_1, u_2, u_3 and u_4 we can see that there are two possible steady states: an infected and an uninfected one [27].
The uninfected steady state is given by

$$\begin{cases} u_1 = \frac{s}{\mu}, \\ u_2 = 0, \\ u_3 = 0, \\ u_4 = 0, \end{cases} \tag{3.5}$$

and the infected steady state is achieved when

$$\begin{cases} u_1 = \frac{(\mu_1+\alpha+b)c}{N\alpha k(1-\eta)}, \\ u_2 = \frac{s-\mu u_1}{\mu_1+\alpha(1-\eta)}, \\ u_3 = \frac{\alpha(1-\eta)u_2}{\delta}, \\ u_4 = \frac{N\delta u_3}{c}. \end{cases} \tag{3.6}$$

In [27] was shown that the infected steady state can exist only when η is less than the following critical value

$$\eta_{crit} = 1 - \frac{\mu c(\mu_1 + \alpha + b)}{N\alpha k s}. \tag{3.7}$$

For our system of parameters, presented in Table 3.1, this critical value is $\eta_{crit} \approx 0.88375$. Whenever $\eta \geq \eta_{crit}$ only the uninfected steady state can exist.
Plugging in the values of Table 3.1 into (3.6) when $\eta < \eta_{crit}$, or (3.5) if $\eta \geq \eta_{crit}$, we obtain the numerical values for solutions $(u_1, u_2, u_3, u_4)^T$ of (3.1) presented in the Table 3.2.

η	u_1	u_2	u_3	u_4
0.0	116	21	33	3549
0.1	129	23	32	3483
0.2	145	26	31	3402
0.3	166	28	30	3298
0.4	194	32	29	3162
0.5	233	36	27	2975
0.6	291	41	25	2702
0.7	388	45	21	2269
0.8	581	44	14	1469
0.9	1000	0	0	0
1.0	1000	0	0	0

Table 3.2: Stable steady states for different values of η, while keeping the other parameters fixed.

Boundedness and stability of solutions

Let us define

$$\begin{cases} \mu_m = \min\{\mu, \mu_1\}, \\ \Xi = \frac{s}{\mu_m}, \\ \Phi := \Phi(\eta) = \frac{\alpha s(1-\eta)}{\mu_m \delta}, \\ \Psi := \Psi(\eta) = \frac{N\alpha s(1-\eta)}{\mu_m c}, \end{cases} \tag{3.8}$$

where μ, μ_1, s etc. are the parameters of (3.1). Consider the set

$$\Gamma(\eta) = \{(u_1, u_2, u_3, u_4) \in \mathbb{R}^4 : 0 \leq u_1 \leq \Xi, 0 \leq u_2 \leq \Xi, 0 \leq u_3 \leq \Phi, 0 \leq u_4 \leq \Psi\}. \tag{3.9}$$

It can be proven [27] that if $u(0) \in \Gamma(\eta)$, then the solution trajectories of (3.1) will stay inside $\Gamma(\eta)$ for all $t \in \Omega_T$.

Remark 2. It is not required that η is constant. As long as $\eta \in M_\eta$, we may allow $\eta(t)$ to vary with time.

For our parameters presented in Table 3.1, these bounds are quantitatively defined as

$$\begin{cases} 0 \leq u_1 \leq 1000, \\ 0 \leq u_2 \leq 1000, \\ 0 \leq u_3 \leq 1538.5(1-\eta), \\ 0 \leq u_4 \leq 166667(1-\eta). \end{cases} \tag{3.10}$$

For various values of η, the following upper limits apply for u_3 and u_4:

η	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
u_3	1538	1384	1230	1076	923	769	615	461	307	153
u_4	166666	150000	133333	116666	100000	83333	66666	50000	33333	16666

Table 3.3: Upper limit for the positive invariant set $\Gamma(\eta)$. The integer parts of fractional numbers is always reported as the upper bound.

It can furthermore be proven that if and only if $\eta \geq \eta_{crit}$ the uninfected state is globally asymptotically Lyapunov stable. On the other hand, if the steady state exists, then it is locally asymptotically Lyapunov stable whenever the following condition is satisfied [27]

$$\Delta C - A^2 D > 0, \tag{3.11}$$

where

$$\begin{aligned} &A = \mu + ku_4 + \mu_1 + \alpha + b + \delta + c, \\ &B = (c+\delta)(\alpha + \mu_1 + \mu + ku_4 + b) + c\delta + \mu(\mu_1 + \alpha + b) + ku_4(\mu_1 + (1-\eta)\alpha), \\ &C = c\delta(\mu + ku_4) + (c+\delta)(\mu\mu_1 + \mu\alpha + \mu b + \mu_1 ku_4 + (1-\eta)\alpha ku_4), \\ &D = c\delta ku_4(\mu_1 + \alpha(1-\eta)), \\ &\Delta = AB - C. \end{aligned} \tag{3.12}$$

We can calculate that, when η is constant and the other parameter values are chosen as in Table 3.1, then the infected steady state is locally asymptotically stable for all values of η such that η is less than the critical value, $\eta_{crit} \approx 0.88$ (Figure 3.1).

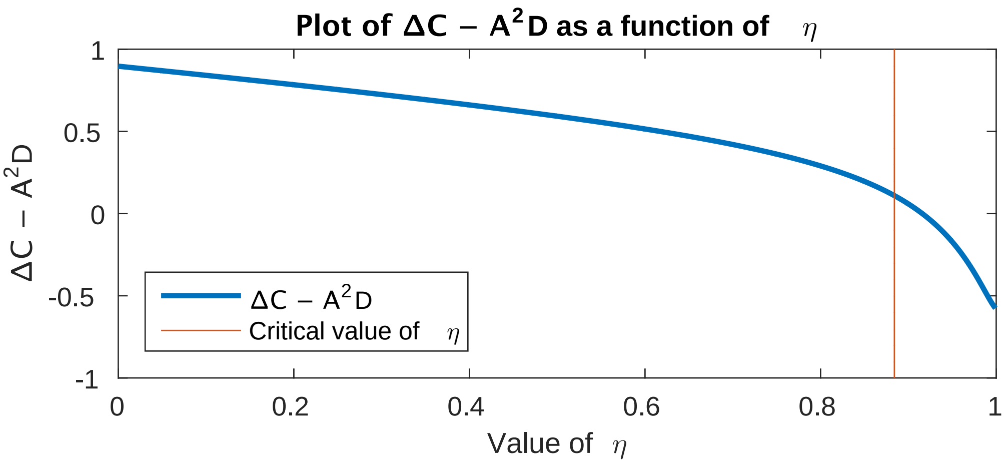


Figure 3.1: $\Delta C - A^2D$ plotted as a function of η. Note that $\Delta C - A^2D > 0 \ \forall \eta < 0.88$.

Thus, if η is constant, and less than the critical value $\eta_{crit} \approx 0.88$, it suffices to know the solution of (3.1) at steady state to deduce η.
Although it is often a reasonable assumption that the drug efficacy is constant for a given individual, viruses mutate readily, which can alter the efficacy of a RT-inhibitor. Thus, it is interesting to know how to determine $\eta(t)$ when it is not constant. So let us for the remainder of this thesis consider the case when $\eta(t)$ is not necessarily constant.

3.1.2 Well-posedness of the forward problem

Let us define $f = (f_1, f_2, f_3, f_4)^T$ as the RHS of (3.1):

$$\begin{cases} f_1 = s - ku_1u_4 - \mu u_1 + (\eta\alpha + b)u_2, \\ f_2 = ku_1u_4 - (\mu_1 + \alpha + b)u_2, \\ f_3 = (1-\eta)\alpha u_2 - \delta u_3, \\ f_4 = N\delta u_3 - cu_4. \end{cases} \tag{3.13}$$

Let $\eta_0 = \eta(0)$. Assume that we have an initial value $u(0) \in \Gamma(\eta_0)$, where $\Gamma(\eta)$ is defined as in (3.9) and $\Omega_T = [0, T]$. Then $f(u, t)$ is clearly Lipschitz continuous on the compact set $\Gamma(\eta) \times \Omega_T$ (and now η is allowed to vary with time). Thus, using the Picard-Lindelöf theorem (Theorem 2.2 in [28]) one can prove that, for given $u(0)$, (3.1) has a unique solution. Furthermore, the solution depends continuously on η in the following sense (Theorem 2.8 in [28]): if $||f(t, u, \eta_1) - f(t, u, \eta_2)|| < \varepsilon$, then

$$||u(t, \eta_1) - u(t, \eta_2)|| \leq ||u(0, \eta_1) - u(0, \eta_2)||e^{Lt} + \frac{\varepsilon}{L}\big(e^{Lt} - 1\big), \tag{3.14}$$

where L is the Lipschitz constant. If the initial values are equal, then on $\Omega_T = [0, T]$, we have

$$||u(t, \eta_1) - u(t, \eta_2)|| \leq \frac{\varepsilon}{L}\big(e^{LT} - 1\big) = C\varepsilon. \tag{3.15}$$

And since f is clearly continuous with respect to η it follows that the solution to (3.1) must be continuous with respect to η.
Hence, we can define a continuous operator $F(\eta) = u(\eta)$, such that the initial value is chosen in the set Γ. To see when F is one-to-one, suppose that there exists $\eta_1(t) \neq \eta_2(t)$ such that $u(t, \eta_1) = u(t, \eta_2)$, then it follows from (3.1) that $\dot{u}(t, \eta_1) \neq \dot{u}(t, \eta_2)$ unless $u_2(t, \eta_1) = u_2(t, \eta_2) = 0$. But if $u_2(t) = 0$ then it must also be the case that $u_3(t) = 0$ and $u_4(t) = 0$. In practical terms, this means that for our model problem, we can guarantee that F is one-to-one whenever a HIV infection is present (that is, if at least one of u_2, u_3 or u_4 is nonzero).
To sum up, we have shown that:

1. For each $\eta \in M_\eta$ there exists a unique solution, $u(\eta)$ to (3.1), such that we can define an operator $F : M_\eta \to \mathcal{C}^1$ as $F(\eta) = u(\eta)$.
2. F is one-to-one whenever the initial value of (3.1) is not of the type $(x, 0, 0, 0)$.
3. F is continuous with respect to η.
4. M_η is a convex set.

And from this we can conclude that it is theoretically possible to solve Inverse Problem 1 by the methods described in Chapter 2.

3.2 Tikhonov functional and Lagrangian

Let H be the Hilbert space of functions defined in Ω_T. It has previously been suggested [3, 4] that η could be determined by minimizing the following Tikhonov functional

$$J(\eta) = \frac{1}{2} \sum_{i=1}^{4} \int_{T_1}^{T_2} (u_i(t) - g_i(t))^2 z_\zeta(t) dt + \frac{1}{2} \gamma \int_0^T (\eta - \eta^0)^2 dt, \tag{3.16}$$

where $u(t)$ is the solution calculated from (3.1), $g(t)$ is the observed solution and η^0 is a first guess for the desired parameter η. The function $z_\zeta \in \mathcal{C}_0^\infty$ is a bump function introduced to make $J(\eta)$ continuous in the entire time interval [0,T]:

$$z_\zeta(t) = \begin{cases} 1, \text{for } t \in [T_1 + 2\zeta, T_2 - 2\zeta], \\ 0, \text{for } t \in [T_2 - \zeta, T_2] \cup [T_1, T_1 + \zeta], \\ \in (0, 1), \text{for } t \in (T_2 - 2\zeta, T_2 - \zeta) \cup (T_1 + \zeta, T_1 + 2\zeta), \end{cases} \tag{3.17}$$

where $\zeta > 0$ is some very small number (for a construction of such a function, see e.g. Chapter 2 of [19], Lemma 2.20, 2.21 and 2.22). It can be proven that (3.16) is Fréchet differentiable and locally strongly convex in a neighbourhood of the exact solution η^* (see [8] or Chapter 1.9 of [7]). This, together with the convexity of M_η, implies that, if η^0 is chosen to be close enough to η^*, a necessary and sufficient condition for a point to be the (unique)

global minimum of (3.16) in M_η is that it is a stationary point with respect to η.
Since we want to minimize (3.16) under the condition that u is a solution to (3.1), we will introduce the Lagrangian

$$L(\nu) = J(\eta) + \sum_{i=1}^{4} \int_0^T \lambda_i(\dot{u}_i - f_i)dt, \tag{3.18}$$

where λ is the Lagrange multiplier, $\nu = (\eta, u, \lambda)$ and f is defined as in (3.13).

3.2.1 Fréchet differential and derivation of the adjoint system

Let us define the following spaces

$$\begin{aligned} &H_u^1(\Omega_T) = \{f \in H^1(\Omega_T) : f(0) = 0\}, \\ &H_\lambda^1(\Omega_T) = \{f \in H^1(\Omega_T) : f(T) = 0\}, \\ &U = \mathcal{C}(\Omega_T) \times H_u^1(\Omega_T) \times H_\lambda^1(\Omega_T). \end{aligned} \tag{3.19}$$

We use the same "heuristic" approach to find the Fréchet differential as in [3, 4], where we assume that u, λ and η can be varied independently. The result is the same as for a rigorous calculation of the Fréchet differential. Let us consider the difference $L(\nu + \bar{\nu}) - L(\nu)$:

$$L(\eta + \bar{\eta}) - L(\eta) = \gamma \int_0^T (\eta\bar{\eta} + \eta^0\bar{\eta} + \frac{1}{2}\bar{\eta}^2)dt - \alpha \int_0^T u_2(\lambda_1 - \lambda_3)\bar{\eta}dt. \tag{3.20}$$

Neglecting $\frac{1}{2}\bar{\eta}^2$, we get

$$L'_\eta(\eta)(\bar{\eta}) = 0, \forall\bar{\eta} \in \mathcal{C}(\Omega_T) \implies \gamma \int_0^T (\eta + \eta^0)\bar{\eta}dt - \alpha \int_0^T u_2(\lambda_1 - \lambda_3)\bar{\eta}dt = 0,$$
$$\forall\bar{\eta} \in \mathcal{C}(\Omega_T). \tag{3.21}$$

We also have

$$L'_\lambda(\lambda)(\bar{\lambda}) = 0, \forall\bar{\lambda} \in H_\lambda^1(\Omega_T) \implies$$
$$\implies \begin{cases} \int_0^T (\dot{u}_1 - s + ku_1u_4 + \mu u_1 - (\eta\alpha + b)u_2)\bar{\lambda_1}dt = 0, \\ \int_0^T (\dot{u}_2 - ku_1u_4 + (\mu_1 + \alpha + b)u_2)\bar{\lambda_2}dt = 0, \\ \int_0^T (\dot{u}_3 - (1 - \eta)\alpha u_2 + \delta u_3)\bar{\lambda_3}dt = 0, \\ \int_0^T (\dot{u}_4 - N\delta u_3 + cu_4)\bar{\lambda_4}dt = 0, \end{cases}$$
$$\forall\bar{\lambda} \in H_\lambda^1(\Omega_T). \tag{3.22}$$

Finally, note that

$$\begin{aligned} \int_0^T \lambda_i(\dot{u}_i - f_i)dt &= \lambda_i(t)u_i(t)\Big|_{t=0}^T - \int_0^T \dot{\lambda}_i u_i - \int_0^T \lambda_i f_i dt = \\ &= C - \int_0^T \dot{\lambda}_i u_i - \int_0^T \lambda_i f_i dt, \end{aligned} \tag{3.23}$$

by partial integration. C is a constant that vanish when taking the difference $L(u+\bar{u})-L(u)$, thus neglecting nonlinear terms we get:

$$L'_u(u)(\bar{u}) = 0, \forall \bar{u} \in H^1_u(\Omega_T) \implies$$
$$\implies \begin{cases} \int_{T_1}^{T_2}(u_1 - g_1)z_\zeta \bar{u}_1 dt - \int_0^T(\dot{\lambda}_1 - \lambda_1 k u_4 - \lambda_1\mu + \lambda_2 k u_4)\bar{u}_1 dt = 0, \\ \int_{T_1}^{T_2}(u_2 - g_2)z_\zeta \bar{u}_2 dt - \int_0^T(\dot{\lambda}_2 - \lambda_2(\mu_1 + \alpha + b) + \lambda_1(\eta\alpha + b) + (1-\eta)\alpha\lambda_3)\bar{u}_2 dt = 0, \\ \int_{T_1}^{T_2}(u_3 - g_3)z_\zeta \bar{u}_3 dt - \int_0^T(\dot{\lambda}_3 - \lambda_3\delta + \lambda_4 N\delta)\bar{u}_3 dt = 0, \\ \int_{T_1}^{T_2}(u_4 - g_4)z_\zeta \bar{u}_4 dt - \int_0^T(\dot{\lambda}_4 - \lambda_4 c + \lambda_1 k u_1 + \lambda_2 k u_1)\bar{u}_4 dt = 0, \end{cases}$$
$$\forall \bar{\lambda} \in H^1_u(\Omega_T). \quad (3.24)$$

To find minimum of (3.18) we should have

$$L'(\eta, u, \lambda)(\bar{\eta}, \bar{u}, \bar{\lambda}) = 0, \forall (\bar{\eta}, \bar{u}, \bar{\lambda}) \in U. \quad (3.25)$$

From (3.22) it follows that we should solve the forward problem (3.1), and from (3.24) it follows that we also should solve the following adjoint problem

$$\begin{cases} \dot{\lambda}_1 = \lambda_1 k u_4 + \lambda_1\mu - \lambda_2 k u_4 + (u_1 - g_1), \\ \dot{\lambda}_2 = \lambda_2(\mu_1 + \alpha + b) - \lambda_1(\eta\alpha + b) - (1-\eta)\alpha\lambda_3 + (u_2 - g_2), \\ \dot{\lambda}_3 = \lambda_3\delta - \lambda_4 N\delta + (u_3 - g_3), \\ \dot{\lambda}_4 = \lambda_4 c + \lambda_1 k u_1 - \lambda_2 k u_1 + (u_4 - g_4), \\ \\ \lambda_i(T) = 0, \quad i = 1, \dots, 4. \end{cases} \quad (3.26)$$

which should be solved backwards in time, with $\lambda(T) = 0$. Existence and uniqueness is proved similarly to the forward problem.

What remains, in order to minimize (3.18) is thus to solve equation (3.21).

3.3 Finite element approximation of the model problem

We introduce the piecewise linear (for u and λ) and piecewise constant (for η) finite element spaces

$$\begin{aligned} W^u_\tau(\Omega_T) &= \{f \in H^1_u : f|_J \in P^1(J) \forall J_\tau\}, \\ W^\lambda_\tau(\Omega_T) &= \{f \in H^1_\lambda : f|_J \in P^1(J) \forall J_\tau\}, \\ W^\eta_\tau &= \{f \in L_2(\Omega_T) : f|_J \in P^0(J) \forall J_\tau\}. \end{aligned} \quad (3.27)$$

The norms are the ones induced from the spaces H^1 and L_2; since the FEM spaces are finite dimensional, the norms for all three spaces will be equivalent. We also define the space

$$U_\tau = W^u_\tau \times W^\lambda_\tau \times W^\eta_\tau. \quad (3.28)$$

The finite element method of (3.25) is: find $\nu_\tau \in U_\tau$ such that $\forall \bar{\nu} \in U_\tau$

$$L'(\nu_\tau)(\bar{\nu}) = 0. \tag{3.29}$$

This FEM formulation can be used to formulate a time-adaptive algorithm for the PIP [3, 4]. But since regularization was not found to be very useful for this particular problem, we will not do this in this thesis.

3.3.1 A posteriori error estimates

Refinement of the time mesh, if it was used, would be based on the following theorem.

Theorem 2 (A posteriori error estimate for the Tikhonov functional [4])**.** *Assume there exists a* $\eta \in H^1(\Omega_T)$ *such that* $\eta = \arg\min J(\eta)$*, where* J *is defined as (3.16). Let* $\eta_\tau \in W_\tau^q$ *be a finite element approximation of* η*. Then*

$$||J(\eta) - J(\eta_\tau)||_{L^2(\Omega_T)} \leq C_I C ||J'(\eta_\tau)||_{L^2(\Omega_T)} \max_{\tau_J} \tau_J^{-1} ||[\eta_\tau]||_{L^2(\Omega_T)} \tag{3.30}$$

with C *and* C_I *being positive constants,* $[\eta_\tau]$ *being the jump of* η_τ *over* $[t_{k-1}, t_k]$ *and* $[t_k, t_{k+1}]$*, and*

$$J'(\eta_\tau) = \gamma(\eta_\tau - \eta^0) - \alpha u_{2\tau}(\lambda_{1\tau} - \lambda_{3\tau}) \tag{3.31}$$

In [4] it was recommended that the time mesh is refined where (3.31) is largest.

Chapter 4

Numerical methods

4.1 Newton's method for forward and adjoint problems

The Newton-Raphson method, or just Newton's method, is a well-known iterative method of finding approximations to the zeroes of a real-valued function. It is used as the basis for our algorithms for solving the forward (3.1) and adjoint (3.26) problems.

4.1.1 Forward problem

Let f be chosen as in (3.13), and let us rewrite (3.1) as

$$\frac{\partial u}{\partial t} = f(u(t)). \tag{4.1}$$

First, we discretize (4.1) in time (0,T) as

$$\frac{u^{k+1} - u^k}{\tau} = f(u^{k+1}), \tag{4.2}$$

where u^{k+1}, u^k are discrete values of the function u at time moments $k+1$, k, respectively, and τ is the uniform time step, $\tau := t^{k+1} - t^k$. Now we extract u^{k+1} value from (4.2) to get

$$u^{k+1} = \tau f(u^{k+1}) + u^k. \tag{4.3}$$

The equation (4.3) can also be written as

$$u^{k+1} - \tau f(u^{k+1}) - u^k = 0. \tag{4.4}$$

To solve the nonlinear equation (4.4) we will use Newton's method. Let us introduce the new variable υ and define $\upsilon := u^{k+1}$. Then (4.4) can be written as

$$F(\upsilon) := \upsilon - \tau f(\upsilon) - u^k = 0. \tag{4.5}$$

Now we apply Newton's method to (4.5) for finding the function υ:

$$\upsilon^{n+1} = \upsilon^n - (F'(\upsilon^n)^{-1}) \cdot F(\upsilon^n). \tag{4.6}$$

Using definition of $F(\upsilon)$ we can find $F'(\upsilon^n)$ in (4.6) as:

$$F'(\upsilon^n) = I - \tau f'(\upsilon^n), \tag{4.7}$$

where I is the identity matrix and $f'(\upsilon^n)$ is the Jacobian of f at υ^n, or $J(\upsilon^n) = f'(\upsilon^n)$.

In the case of our function f given by (3.13), the Jacobian can be computed as

$$J(\upsilon^n) = \begin{bmatrix} \frac{\partial f_1}{\partial u_1} & \frac{\partial f_1}{\partial u_2} & \frac{\partial f_1}{\partial u_3} & \frac{\partial f_1}{\partial u_4} \\ \frac{\partial f_2}{\partial u_1} & \frac{\partial f_2}{\partial u_2} & \frac{\partial f_2}{\partial u_3} & \frac{\partial f_2}{\partial u_4} \\ \frac{\partial f_3}{\partial u_1} & \frac{\partial f_3}{\partial u_2} & \frac{\partial f_3}{\partial u_3} & \frac{\partial f_3}{\partial u_4} \\ \frac{\partial f_4}{\partial u_1} & \frac{\partial f_4}{\partial u_2} & \frac{\partial f_4}{\partial u_3} & \frac{\partial f_4}{\partial u_4} \end{bmatrix} (\upsilon^n), \tag{4.8}$$

or computing all partial derivatives in the above matrix, we can get the following expression of the Jacobian:

$$J(\upsilon^n) = \begin{bmatrix} -ku_4^{k+1} - \mu & (\eta\alpha + b) & 0 & -ku_1^{k+1} \\ ku_4^{k+1} & -(\mu_1 + \alpha + b) & 0 & ku_1^{k+1} \\ 0 & (1-\eta)\alpha & -\delta & 0 \\ 0 & 0 & N\delta & -c \end{bmatrix}. \tag{4.9}$$

4.1.2 Adjoint problem

We define RHS of the (3.26) as:

$$\begin{cases} y_1 = \lambda_1 k u_4 + \lambda_1 \mu - \lambda_2 k u_4 + (u_1 - g_1), \\ y_2 = \lambda_2(\mu_1 + \alpha + b) - \lambda_1(\eta\alpha + b) - (1-\eta)\alpha\lambda_3 + (u_2 - g_2), \\ y_3 = \lambda_3\delta - \lambda_4 N\delta + (u_3 - g_3), \\ y_4 = \lambda_4 c + \lambda_1 k u_1 - \lambda_2 k u_1 + (u_4 - g_4). \end{cases} \tag{4.10}$$

We can rewrite the system (3.26) in the form

$$\frac{\partial \lambda}{\partial t} = y(\lambda(t)). \tag{4.11}$$

First, to solve (4.11) numerically, we discretize (4.11) in time as

$$\frac{\lambda^{k+1} - \lambda^k}{\tau} = y(\lambda^k), \tag{4.12}$$

where λ^{k+1}, λ^k are discrete values of the function $\lambda(t)$ at time moments $k+1$, k, respectively, and τ is the uniform time step $\tau := t^{k+1} - t^k$. Since we solve the adjoint problem backwards in time from $t = T$ to $t = 0$, we extract λ^k from (4.12) for the already known values of λ^{k+1} to get

$$\lambda^k = \lambda^{k+1} - \tau y(\lambda^k), \tag{4.13}$$

which also can be written as

$$\lambda^k + \tau y(\lambda^k) - \lambda^{k+1} = 0. \tag{4.14}$$

The equation (4.14) is nonlinear, and we will use Newton's method to solve it. Let us introduce the new variable w and define

$$w := \lambda^k. \tag{4.15}$$

Then (4.14) can be written as

$$Q(w) := w + \tau y(w) - \lambda^{k+1} = 0. \tag{4.16}$$

The Newton's method applied to (4.16) for finding the function w will be:

$$w^{n+1} = w^n - (Q'(w^n)^{-1}) \cdot Q(w^n). \tag{4.17}$$

Using definition of $Q(w)$ in (4.16) we can get $Q'(w^n)$ in (4.17) as:

$$Q'(w^n) = I + \tau y'(w^n), \tag{4.18}$$

where I is the identity matrix and $y'(w^n)$ is the Jacobian of y at w^n, or $J(w^n) = y'(w^n)$. In the case when the function y is given by (4.10), the Jacobian can be computed as

$$J(w^n) = \begin{bmatrix} \frac{\partial y_1}{\partial \lambda_1} & \frac{\partial y_1}{\partial \lambda_2} & \frac{\partial y_1}{\partial \lambda_3} & \frac{\partial y_1}{\partial \lambda_4} \\ \frac{\partial y_2}{\partial \lambda_1} & \frac{\partial y_2}{\partial \lambda_2} & \frac{\partial y_2}{\partial \lambda_3} & \frac{\partial y_2}{\partial \lambda_4} \\ \frac{\partial y_3}{\partial \lambda_1} & \frac{\partial y_3}{\partial \lambda_2} & \frac{\partial y_3}{\partial \lambda_3} & \frac{\partial y_3}{\partial \lambda_4} \\ \frac{\partial y_4}{\partial \lambda_1} & \frac{\partial y_4}{\partial \lambda_2} & \frac{\partial y_4}{\partial \lambda_3} & \frac{\partial y_4}{\partial \lambda_4} \end{bmatrix} (w^n). \tag{4.19}$$

Taking into account (4.10) together with (4.15) the Jacobian in (4.19) can be explicitly computed as

$$J(w^n) = \begin{bmatrix} ku_4^k + \mu & -ku_4^k & 0 & 0 \\ -(\eta\alpha + b) & \mu_1 + \alpha + b & (\eta - 1)\alpha & 0 \\ 0 & 0 & \delta & -N\delta \\ ku_1^k & -ku_1^k & 0 & c \end{bmatrix}. \tag{4.20}$$

4.2 Minimization algorithms

4.2.1 The conjugate gradient algorithm

The Fréchet derivative of the Tikhonov functional is (3.31), so the gradient at the observation point t_i is

$$g^m(t_i) = \gamma(\eta^m(t_i) - \eta^0(t_i)) - \alpha u_2^m(t_i)(\lambda_1^m(t_i) - \lambda_3^m(t_i)), \tag{4.21}$$

where u^m and λ^m are obtained by Newton's method, as described above. The conjugate gradient method (CGM) is as follows:

Algorithm 1.

0. Choose a time partition, J_τ of Ω_T, and an initial guess η^0.
1. Compute the solutions to forward and adjoint problems corresponding to η^m on J_τ.
2. Compute g^m on J_τ according to (4.21).
3. Compute $\beta^m = \frac{||g^m||^2}{||g^{m-1}||^2}$.
4. Compute $d^m = -g^m + \beta^m d^{m-1}$ (or if $m = 0$ then $d^0(t_i) = -g^0(t_i)$).
5. Set $\eta^{m+1} = \eta^m + \sigma^m d^m$.
6. Stop computing new η if $||g^m|| < \theta$, where θ is the tolerance level, or if $||g^m||$ grows, which means that we have passed the minimum, or if $||\eta^m||$ is stabilized. Otherwise go to step 1.

The optimal step length, σ^m, can be calculated according to [24] as

$$\sigma^m = -\frac{(g^m, d^m)}{\gamma(d^m, d^m)}, \tag{4.22}$$

where γ is the regularization parameter, and $(.,.)$ denotes the L_2 scalar product.

4.2.2 Choice of regularization parameter γ

Lower bound for γ

We recall that the Lagrangian is explicitly

$$L(\nu) = \\ = \frac{1}{2}\sum_{i=1}^{4}\int_{T_1}^{T_2}(u_i(t) - g_i(t))^2 z_\zeta(t)dt + \frac{1}{2}\gamma\int_0^T(\eta_(t) - \eta^0(t))^2 dt + \sum_{i=1}^{4}\int_0^T \lambda_i(t)(\dot{u}_i(t) - f_i)dt. \tag{4.23}$$

In order to ensure that $\eta^m \in M_\eta$, i.e. the set of admissible functions (3.3), we must set the regularization parameter sufficiently large, such that the regularization term, $\frac{\gamma}{2}||\eta - \eta^0||_{L_2}$ is of about the same order of magnitude as the other terms in the Lagrangian (4.23). Otherwise the Tikhonov functional might be such that η is taken outside the set of admissible functions, M_η, during the CGM algorithm. In the following we will derive a lower bound for γ. Consider the gradient (4.21). From the first step of CGM (Algorithm 1) we will find that

$$\eta^1 = \eta^0 + \sigma^0 \alpha u_2(\lambda_1 - \lambda_3). \tag{4.24}$$

But if η^1 is supposed to be a better approximation of the exact solution, η^*, than η^0 was, then we must have:

$$\eta^0 \in \{\eta : ||\eta - \eta^*||_{L_\infty} < \varepsilon^0\} \implies \eta^1 \in \{\eta : ||\eta - \eta^*||_{L_\infty} < \varepsilon^0\}, \tag{4.25}$$

where $0 < \varepsilon^0 < \min\{1 - \eta^0, \eta^0\}$ is some error estimate of the initial guess. That is,

$$||\eta^1 - \eta^0||_{L_\infty} = ||\sigma^0 \alpha u_2(\lambda_1 - \lambda_3)||_{L_\infty} < 2\varepsilon^0. \tag{4.26}$$

From (4.22) it follows that

$$\sigma^0 = \frac{1}{\gamma}. \tag{4.27}$$

Thus, by combining (4.26) and (4.27), it follows that we must have for the first step of CGM (Algorithm 1)

$$\gamma^0 > \frac{\alpha}{2\varepsilon^0}||u_2(\lambda_1 - \lambda_3)||_{L_\infty}. \tag{4.28}$$

If γ^0 does not obey (4.28) the first step of the CGM would not improve the solution, but potentially take η^1 outside the set of admissible functions. The reason that we use the L_∞-norm here, is that we want γ^0 to be just sufficiently large to keep $\eta^1 \in M_\eta$. Other norms, such as the L_2-norm, would make γ^0 too large, and as a consequence the convergence rate of the algorithm would be very poor.

Recommendation for iteratively updated γ^i

We have seen that γ must not be too small. On the other hand, if γ is too big, minimization of (4.23) would do little to improve the first guess. Actually, if $\delta > 0$ is the noise level of the observations, we should have that $\gamma(\delta) \to 0$ as $\delta \to 0$. Therefore - assuming that each iteration of the CGM (Algorithm 1) will lead to a better approximation of η - we will choose a decreasing sequence, $\{\gamma^i\}$, of regularization parameters, according to [1], such that

$$\begin{aligned} \gamma^0 &= \tfrac{\alpha}{\delta_0}||u_{2\mathcal{T}}(\lambda_{1\mathcal{T}} - \lambda_{3\mathcal{T}})||_{L_\infty}, \\ \gamma^i &= \tfrac{\gamma^0}{\sqrt{i+1}}, i \in \mathbb{N}. \end{aligned} \tag{4.29}$$

4.3 Analytical determination of η at observation points

Suppose that the problem of identifying η is well-posed. Then, if one knows η at the entire time domain, Ω_T, one can simply calculate η explicitly from either the first or the third row of (3.1) as:

$$\eta = \Big(\frac{\dot{u}_1 - s + ku_1u_4 + \mu u_1}{u_2} - b\Big)/\alpha, \tag{4.30}$$

or as:

$$\eta = 1 - \frac{\dot{u}_3 + \delta u_3}{\alpha u_2}. \tag{4.31}$$

We will use the third row (4.31), since that expression involves less computations than (4.30). The derivatives, $\dot{u}_i$ in (4.31), can be approximated by central finite differences

$$\dot{u}(t) \approx \frac{u(t+\tau) - u(t-\tau)}{2\tau}. \tag{4.32}$$

However, we need to be careful, because numerical differentiation is one of the most well-known examples of potentially ill-posed problems. Thus, let us discuss (4.32) more carefully, especially since ill-posed problems is the main subject of this thesis.

4.3.1 Error estimates for central differences

There are two factors contributing to the error in the approximation (4.32). Firstly, it is error caused from the fact that (4.32) is only an approximation to the perfect derivative: as $\tau \to 0$ we approach the true derivative. So the smaller τ is, the smaller is the error. Secondly, it is round-off errors, which increases with decreasing τ; since we need to perform subtraction on two almost equally large quantities much accuracy is lost due to cancellation. Thus, the best results are obtained when τ is small, but not too small. Let us consider these errors more precisely in the following.

Truncation error of central differences

Let us in this section assume that $u \in \mathcal{C}^\infty(\Omega_T)$. Those functions are dense in $C^1(\Omega_T)$, so if the solution to (3.1) is not smooth we can just approximate it by a smooth function. Consider the Taylor series

$$u(t+\tau) = u(t) + \dot{u}(t)\tau + \frac{\ddot{u}(t)}{2!}\tau^2 + \frac{\dddot{u}(t)}{3!}\tau^3 + \frac{u^{(4)}(t)}{4!}\tau^4 + \mathcal{O}(\tau^5), \tag{4.33}$$

and

$$u(t-\tau) = u(t) - \dot{u}(t)\tau + \frac{\ddot{u}(t)}{2!}\tau^2 - \frac{\dddot{u}(t)}{3!}\tau^3 + \frac{u^{(4)}(t)}{4!}\tau^4 + \mathcal{O}(\tau^5), \tag{4.34}$$

thus, we have

$$\frac{u(t+\tau) - u(t-\tau)}{2\tau} = \dot{u}(t) + \frac{\dddot{u}(t)}{3!}\tau^2 + \mathcal{O}(\tau^4). \tag{4.35}$$

Hence the approximation error in the estimate (4.32), which we will call the *truncation error*, is defined as:

$$e(t) = \frac{\dddot{u}(t)\tau^2}{6} + \mathcal{O}(\tau^4). \tag{4.36}$$

Thus, the truncation error of (4.32) is $\mathcal{O}(\tau^2)$, and is minimized if τ and $\dddot{u}(t)$ are as small as possible. But there is another factor also that contributes to the error, which we need to consider.

Ill-posedness of numerical differentiation at small step sizes

Suppose that the function u is observed with noise, u_δ, and let $\delta(t)$ be the noise at the point t, so $u_\delta(t) = u(t) + \delta(t)$. Let $M_\delta = \sup_{t\in\Omega_T} |\delta(t)|$. Then, using estimate (4.36) and the definition of M_δ, we get

$$\begin{aligned} \big|e(t)\big| &= \big|\dot{u}(t) - \dot{u}_\delta(t)\big| = \left|\dot{u}(t) - \Big(\frac{u(t+\tau)-u(t-\tau)}{2\tau} + \frac{\delta(t+\tau)-\delta(t-\tau)}{2\tau}\Big)\right| \\ &\leq \left|\dot{u} - \frac{u(t+\tau)-u(t-\tau)}{2\tau}\right| + \left|\frac{\delta(t+\tau)-\delta(t-\tau)}{2\tau}\right| \leq \left|\frac{\dddot{u}(t)\tau^2}{6}\right| + \frac{M_\delta}{\tau} + \mathcal{O}(\tau^4). \end{aligned} \tag{4.37}$$

If the step size, τ, is too small, the second term in (4.37) will be very big. Thus, there exists an optimal step size for the problem. In particular we can conclude that a requirement for the problem to be well-posed is that $\frac{\delta}{\tau} \to 0$ as $\delta \to 0$, thus giving us the estimate $\tau(\delta) \geq \delta^\mu$ with $\mu \in (0,1)$. To find the optimal step-size, we set $e(t,\tau) = \frac{|\dddot{u}(t)|}{6}\tau^2 + \frac{M_\delta}{\tau}$ and differentiate with respect to τ to obtain

$$e'_\tau(t,\tau) = \frac{|\dddot{u}(t)|}{3}\tau - \frac{M_\delta}{\tau^2}, \tag{4.38}$$

which has a stationary point, $e'_\tau = 0$, when

$$\tau = \left(\frac{3M_\delta}{|\dddot{u}(t)|}\right)^{1/3}. \tag{4.39}$$

If we allow the step size to change with the third derivative, we can obtain the best possible accuracy of the derivatives. However, this requires that third derivatives are calculated first, which is of course even more difficult than first derivatives. For the error estimates in this study, we will simply content ourselves with using the following formula for the third derivative, which is analogous to (4.32).

$$\dddot{u}(t) \approx \frac{1}{2\tau^3}[u(t+2\tau) - 2u(t+\tau) + 2u(t-\tau) - u(t-2\tau)], \tag{4.40}$$

with truncation error $\mathcal{O}(\tau^2)$ and round-off error $\mathcal{O}(\tau^{-3})$.

Adaptivity

It would be interesting to compare the explicit calculation of η using (4.31) with results obtained by Tikhonov regularization (i.e. Algorithm 1). To do this properly, we will differentiate with near-optimal accuracy. More precisely, we will in general use a step size of 1, but refine with optimal step size where the truncation error is estimated to be high compared to the round-off error.
We will consider the truncation error to be high whenever

$$\frac{|\dddot{u}_3(t)|}{6}\tau^2 > \alpha, \tag{4.41}$$

at the same time as

$$\tau > \sqrt{\delta(t)}, \tag{4.42}$$

where $\alpha > 0$ is the error tolerance for the truncation error, which is set by the user. Note that we only consider the third component of u, since it is the derivative of this component that occurs in (4.31). So the following algorithm is suggested for explicit computation of η:

Algorithm 2.

0. Choose a uniform coarse time partition, J_τ, of Ω_T, and an error tolerance level, $\alpha > 0$.
1. Use (4.31) to compute η_τ. Derivatives can be computed using either the central finite difference formula (4.32), or Richardson extrapolation (4.43) described in next subsection.
2. Calculate the truncation error, $e = \frac{|\dddot{u}_3(t)|}{6}\tau^2$ (4.36) for each point of the time mesh. Use (4.40) to compute approximate third derivatives.
3. Refine the time mesh at those points where both (4.41) and (4.42) are true. We suggest that optimal step length is used for those points, which τ can be computed via (4.39).
4. Compute η using (4.31) on the refined time-mesh.

For Inverse Problem 1, refinement of the time mesh turned out to be especially useful at time moments where the parameter $\eta(t)$ changes fast, or where the solution of the forward problem (3.1) is far from equilibrium.

4.3.2 Improving accuracy of differentiation

Richardson extrapolation

The truncation error when performing numerical differentiation can be further decreased by using Richardson extrapolation. Richardson extrapolation means that you calculate the central differences for both τ and 2τ, and then subtract them in such a way that the $\mathcal{O}(\tau^2)$-term in their Taylor series cancels. Thus we will obtain the following estimate for the derivative:

$$\dot{u} \approx \frac{u(t-2\tau) + 8u(t+\tau) - 8u(t-\tau) - u(t+2\tau)}{12\tau}. \tag{4.43}$$

Here, the truncation error is $\mathcal{O}(\tau^4)$, but an analysis similar to the one above shows that the round-off error has increased to $\frac{3}{2}\frac{M_\delta}{\tau}$. Since Richardson extrapolation apparently is most useful in reducing errors when both the noise and step size are small, we will use this method to calculate the derivatives only when the noise level is smaller than 1%.

Smoothing noise

So far we, have only discussed how to reduce the truncation error. But if the noise level is large, which it often is in practice, it is typically the round-off errors that dominate. Then the step size should be large (actually the optimal step size for noisy data is often greater than 1), and we get little help from techniques such as Richardson extrapolation. So we need other tools than those mentioned above to deal with these kinds of problems. Many methods have been proposed for stable numerical differentiation of noisy data over the years, see for example [10, 14, 21, 22].
For this problem we will smooth the graphs using LOESS [9] or moving average filtering from MATLAB's curve fitting toolbox, in addition spikes will be removed using the Hampel filter [20] from MATLAB's signal processing toolbox. Numerical studies suggested that it was best to remove noise after η had been calculated analytically, rather than directly on the noisy observations. Otherwise there was the risk that the filters smoothed out the solution trajectories too much at the initial time steps. We used moving average smoothing for time-adaptive methods and LOESS smoothing otherwise.

Chapter 5

Computational results

In the computational studies we set $\Omega_T = [0, 500]$, and partition it into a uniform time mesh, $\mathcal{J}_\tau$, with step length $\tau = 1$. The forward and adjoint problems are solved using Newton's method on the time partition $\mathcal{J}_\tau$, with maximum number of iterations set to 1000.
Observations are simulated by defining an $\eta(\tau)$ on $\mathcal{J}_\tau$ and then computing the forward solution g, for this η_τ, additive uniform noise is simulated according to

$$\begin{aligned} &g_{\delta i} = g_i + \delta X_i g_i, \\ &i \in \{1, 2, 3, 4\}, \end{aligned} \tag{5.1}$$

where δ is the noise level, and X_i are uniformly distributed pseudo-random numbers in $[-1, 1]$.
The computations were performed on a computer with AMD E2-1800 APU (1.70 GHz), 64-bit system, 4.00 GB RAM.

5.1 Numerical study of the forward problem

An initial guess for η^0 can be obtained by formula (4.31), that is

$$\eta_\tau = 1 - \frac{\dot{u}_{3\tau} + \delta u_{3\tau}}{\alpha u_{2\tau}}, \tag{5.2}$$

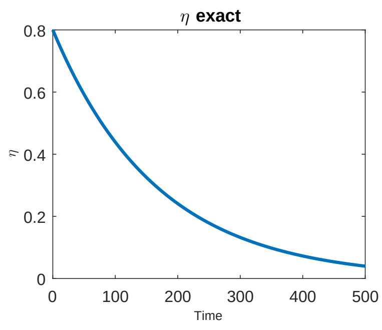
with derivatives computed according to formula (4.32). Then the forward solution can again be computed for this particular η_τ, to obtain the forward solution corresponding to the approximate η. We can calculate the relative errors as

$$e_2(x) = \frac{||x - x_\delta||_2}{||x||_2}, \tag{5.3}$$

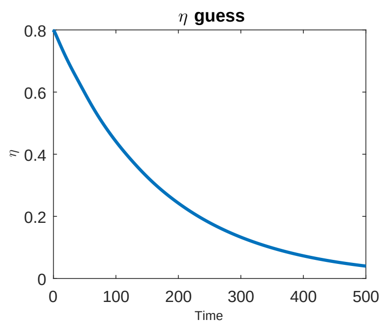
for the 2-norm, and similarly for the ∞-norm

$$e_\infty(x) = \frac{||x - x_\delta||_\infty}{||x||_\infty}. \tag{5.4}$$

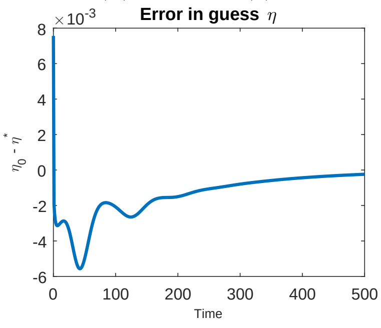
The relative errors suggest that the both the forward and inverse problems are fairly well-conditioned. However, the forward solution is huge compared to η, which means that we have to be very careful when choosing the regularization parameter for the Tikhonov functional (3.16). In fact we will see that the PIP (Inverse Problem 1) is difficult to solve by minimizing the Tikhonov functional due to the difficulty of choosing a suitable regularization parameter.



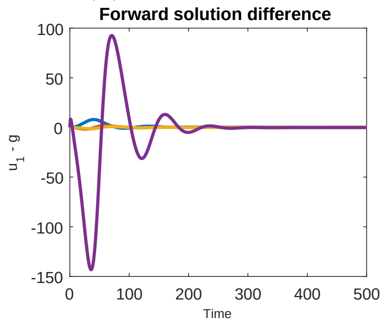
(a) Exact $\eta(t)$.



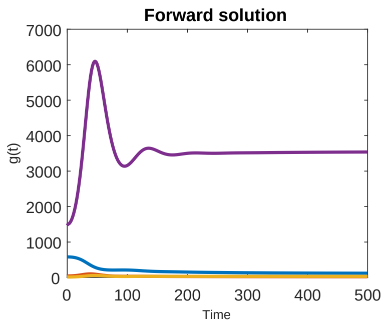
(b) Calculated η.



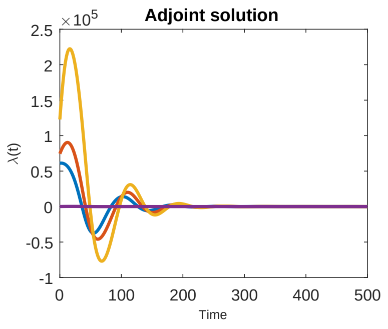
(c) Error in $\eta(t)$. Relative errors: $e_2 = 0.0067$, $e_\infty = 0.025$.



(d) Error in forward solution. Relative errors: $e_2 = 0.0077, e_\infty = 0.0039$.

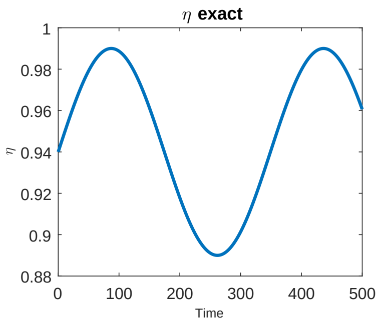


(e) Forward solution.

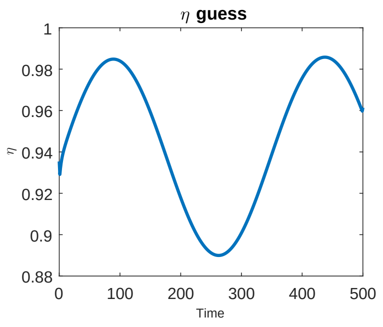


(f) Adjoint solution

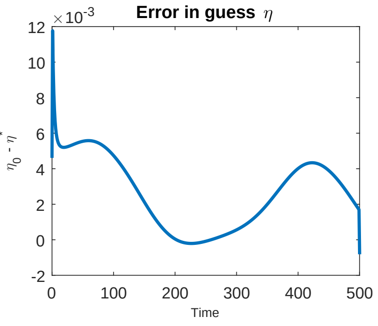
Figure 5.1: Computer simulations of PIP with $\eta(t) = 0.8e^{-3t/T}$.



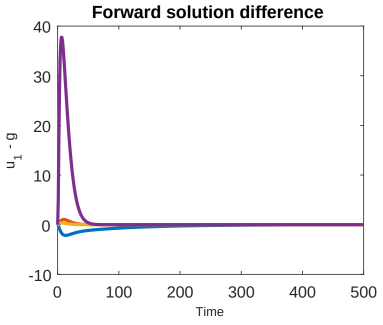
(a) Exact $\eta(t)$.



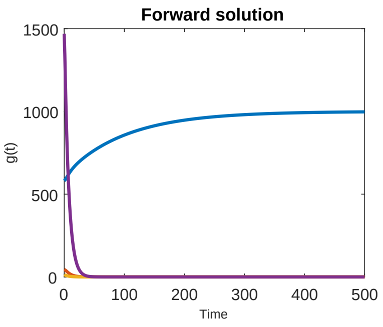
(b) Calculated η.



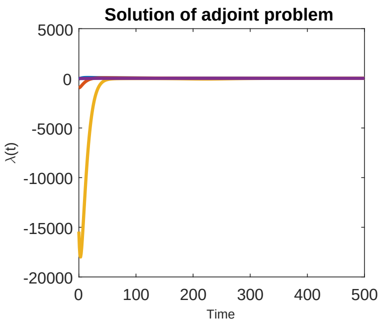
(c) Error in $\eta(t)$. Relative errors: $e_2 = 0.0076, e_\infty = 0.019$.



(d) Error in forward solution. Relative errors: $e_2 = 0.011, e_\infty = 0.0084$.

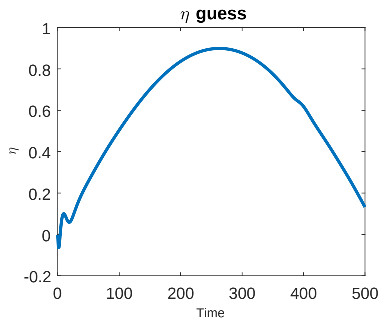


(e) Forward solution.

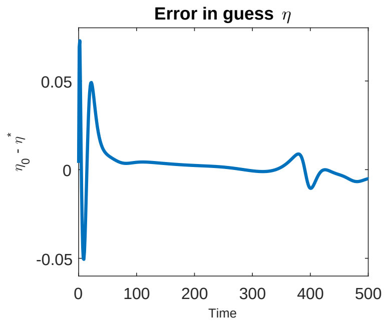


(f) Adjoint solution

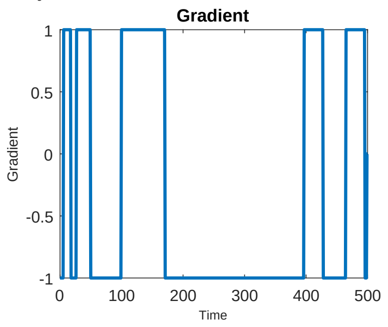
Figure 5.2: Computer simulations of PIP with $\eta(t) = 0.94 + 0.05\sin(9t/T)$.



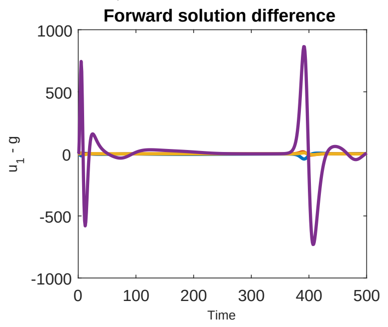
(a) Calculated $\eta(t)$. The exact one is very similar.



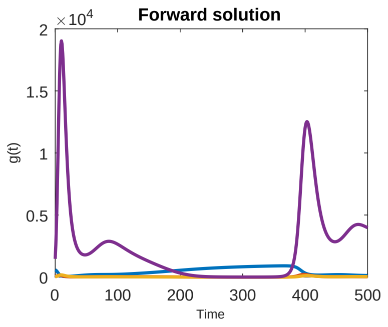
(b) Error in $\eta(t)$. Relative errors: $e_2 = 0.0076, e_\infty = 0.017$.



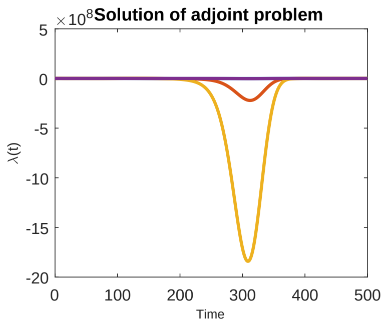
(c) Sign of gradient of the Tikhonov functional.



(d) Error in forward solution. Relative errors: $e_2 = 0.0048, e_\infty = 0.0028$.



(e) Forward solution.



(f) Adjoint solution

Figure 5.3: Computer simulations of PIP with $\eta(t) = 0.9\sin(3t/T)$.

5.2 Direct solution methods for the inverse problem

The inverse problem is, in fact, quite well-conditioned, and in the most cases it can easily be solved by elementary methods. Just as before, we solve the inverse problem simply by using (4.31) and (4.32).

5.2.1 Sparsely distributed observations

Suppose that we only have measurements at certain discrete and sparsely distributed time moments. This is a situation likely to occur in practice, since the physicians usually measure the concentrations of viruses and T cells only at certain times. Computer simulations showed that, by the use of smoothing splines one could interpolate to a finer time mesh with good accuracy, given that η stays below the critical value.

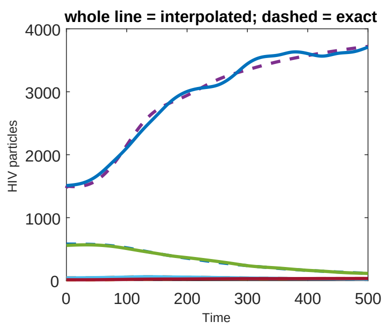
In the following examples, we will consider two different initial values for the model problem (3.1): the equilibrium values that would eventually have been obtained for constant $\eta \equiv 0.0$, i.e. no treatment, and the equilibrium values that would be obtained for $\eta \equiv 0.8$. For convenience, we will call these equilibrium 1 and equilibrium 2, respectively. See Table 5.1 for quantitative values. Furthermore, we will consider different $\eta(t)$, different noise levels and different number of observation points. The observations are always assumed to be equidistributed over the time domain, Ω_T.

Initial value	u_1	u_2	u_3	u_4
Equilibrium 1 (for $\eta \equiv 0.0$)	120	22	35	3700
Equilibrium 2 (for $\eta \equiv 0.8$)	581	44	14	1469

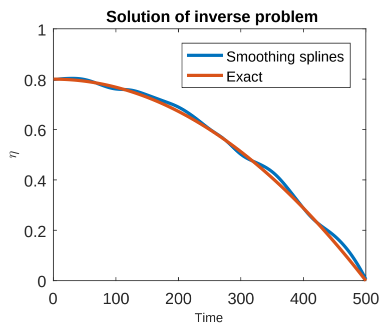
Table 5.1: Numerical values of the two equilibrium states considered.

In Figure 5.4, we see study reconstruction of $\eta(t) = 0.8\Big(1 - \big(\frac{3t}{T}\big)^2\Big)$. The reconstruction at 15% noise level is quite good, but reconstruction is poor at early time moments, if the initial value for (3.1) is such that $\eta(t)$ cannot be assumed to change continuously at $t = 0$. Equilibrium 2 corresponds to a smooth decay in drug efficacy over time, whereas equilibrium 1 corresponds to an abrupt change of drug efficacy from 0.0 to 0.8 at $t = 0$, and then a smooth decay to 0.

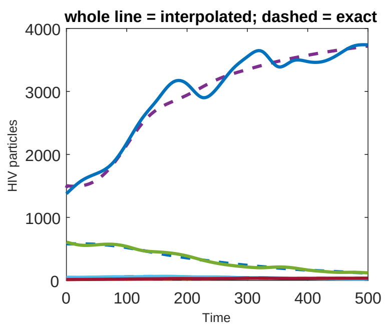
It is clear that the optimal number of observation points depend on the properties of η. For the exponential function (Figure 5.5), it seems that only 5 points on the time interval $\Omega_T = [0, 500]$ are quite sufficient for good accuracy. For the sine function observations at 10 different time moments is much better than at 5 time moments, and if the data are noisy, observations at 20 time moments are still better (Figures 5.6 and 5.7).
But noise and observation points have little effect on the results, compared to the smoothness of the solution trajectories. Figure 5.8 shows reconstruction of three different $\eta(t)$, with all other factors identical. In this simulation, the initial value of (3.1) equals the uninfected equilibrium, such that Figure 5.8 a) and b) shows a continuously increasing drug efficacy over time; Figure 5.8 c) and d) shows an abrupt change from 0.0 to 4.0 at $t = 0$ followed by constant efficacy and Figure 5.8 e) and f) shows an abrupt change from 0.0 to 0.8 followed by linear decay to 0.



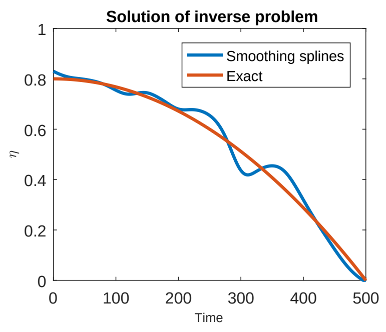
(a) Forward solution: 5% noise, 20 observations, initial value at equilibrium 2.



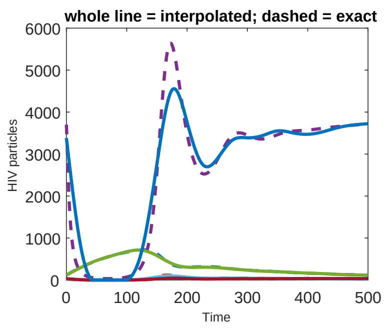
(b) Estimated η: 5% noise, 20 observations, initial value at equilibrium 2.



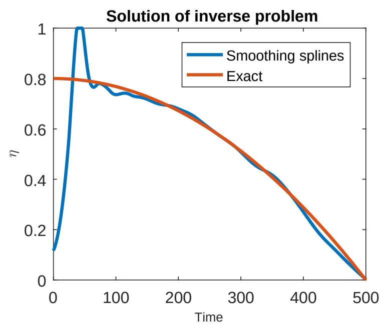
(c) Forward solution: 15% noise, 20 observations, initial value at equilibrium 2.



(d) Estimated η: 15% noise, 20 observations, initial value at equilibrium 2. Maximal error is 0.079

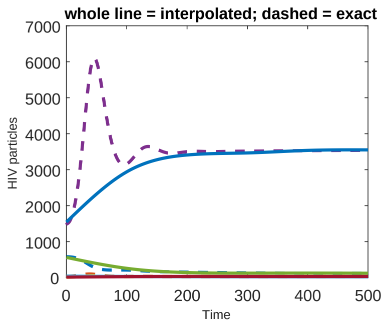


(e) Forward solution: 5% noise, 20 observations, initial value at equilibrium 1.

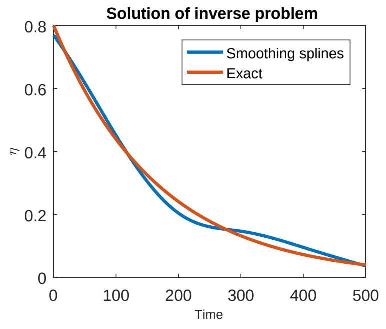


(f) Estimated η: 5% noise, 20 observations, initial value at equilibrium 1.

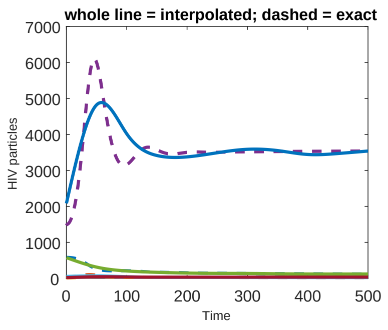
Figure 5.4: Computer simulations of PIP with $\eta(t) = 0.8(1 - \left(\frac{3t}{T}\right)^2)$.



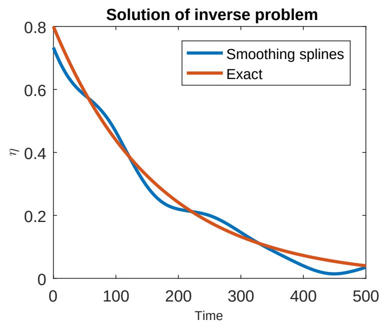
(a) Forward solution: 5% noise, 5 observations, initial value at equilibrium 2.



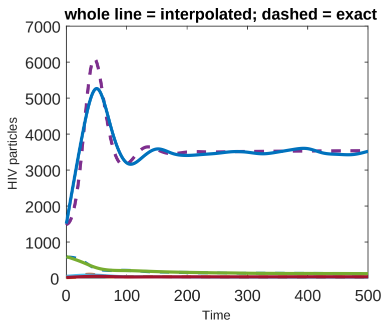
(b) Estimated η: 5% noise, 5 observations, initial value at equilibrium 2. Maximal error: 0.037



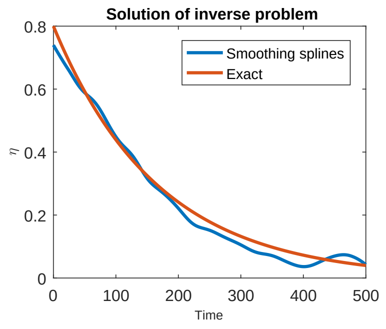
(c) Forward solution: 5% noise, 10 observations, initial value at equilibrium 2.



(d) Estimated η: 5% noise, 10 observations, initial value at equilibrium 2. Maximal error: 0.067

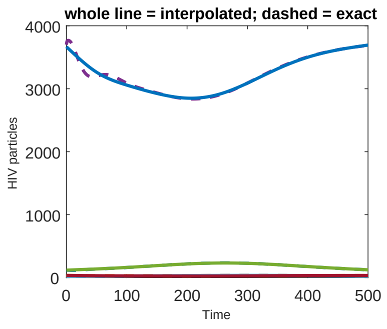


(e) Forward solution: 5% noise, 20 observations, initial value at equilibrium 2.

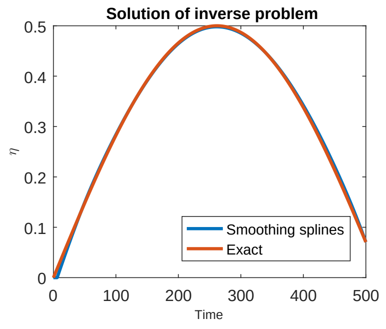


(f) Estimated η: 5% noise, 20 observations, initial value at equilibrium 2. Maximal error: 0.060

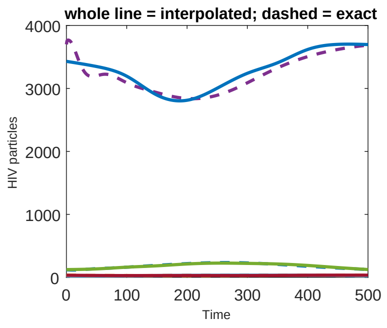
Figure 5.5: Computer simulations of PIP with $\eta(t) = 0.8e^{-3t/T}$. Despite considerable improvement in the approximation of the forward solution, with increasing number of observations, no significant improvement in the approximation of η occurred for this function.



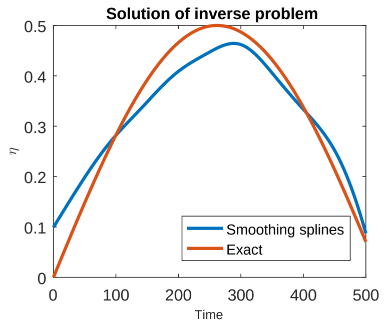
(a) Forward solution: no noise, 10 observations, initial value at equilibrium 1.



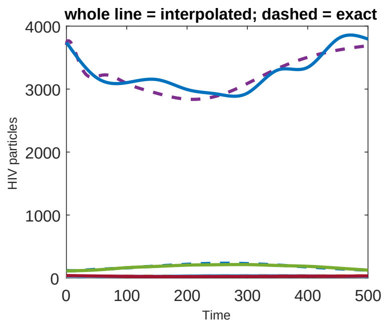
(b) Estimated η: no noise, 10 observations, initial value at equilibrium 1. Maximal error: 0.018



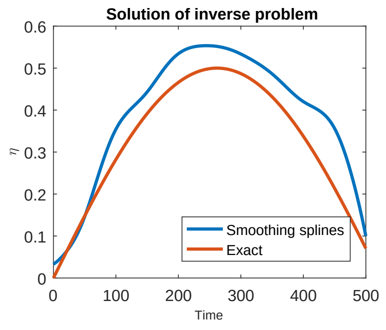
(c) Forward solution: 10% noise, 10 observations, initial value at equilibrium 1.



(d) Estimated η: 10% noise, 10 observations, initial value at equilibrium 1. Maximal error: 0.099

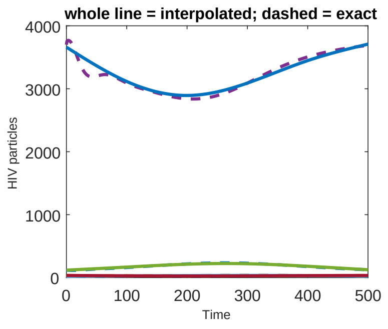


(e) Forward solution: 20% noise, 10 observations, initial value at equilibrium 1.

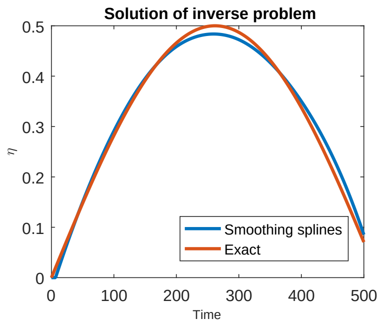


(f) Estimated η: 20% noise, 10 observations, initial value at equilibrium 1. Maximal error: 0.143

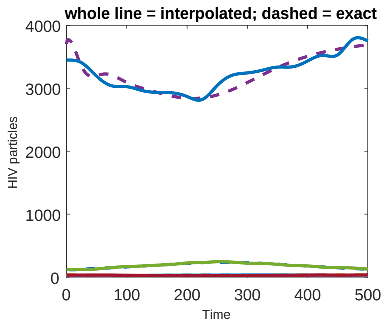
Figure 5.6: Computer simulations of PIP with $\eta(t) = 0.8\sin(3t/T)$. The accuracy in estimation of the parameter η deteriorates as the noise increases.



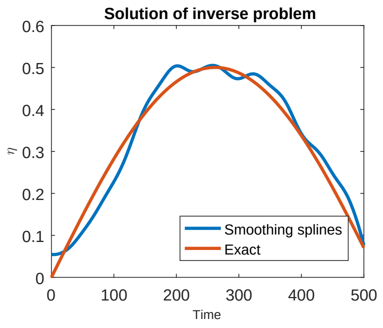
(a) Forward solution: no noise, 5 observations, initial value at equilibrium 1.



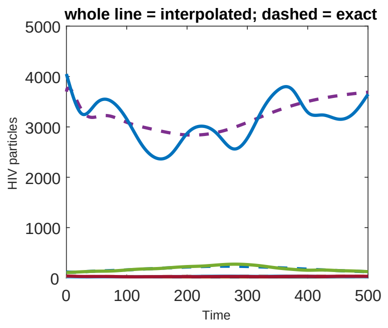
(b) Estimated η: no noise, 5 observations, initial value at equilibrium 1. Maximal error: 0.027



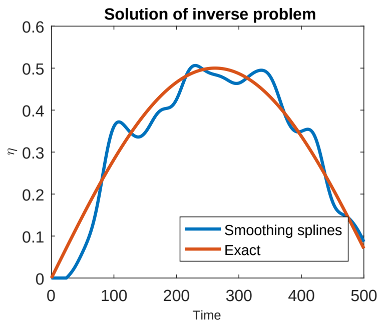
(c) Forward solution: 10% noise, 20 observations, initial value at equilibrium 1.



(d) Estimated η: 10% noise, 20 observations, initial value at equilibrium 1. Maximal error: 0.055

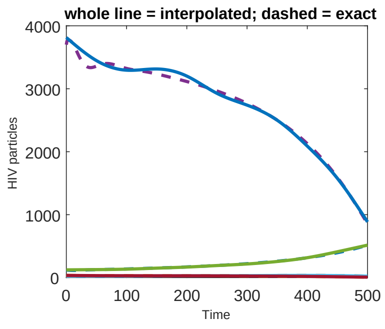


(e) Forward solution: 20% noise, 20 observations, initial value at equilibrium 1.

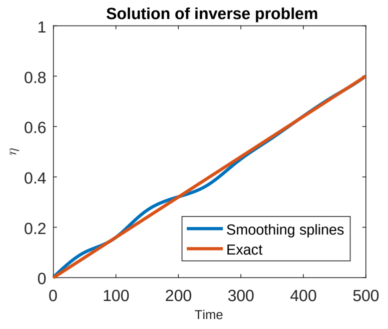


(f) Estimated η: 20% noise, 20 observations, initial value at equilibrium 1. Maximal error: 0.087

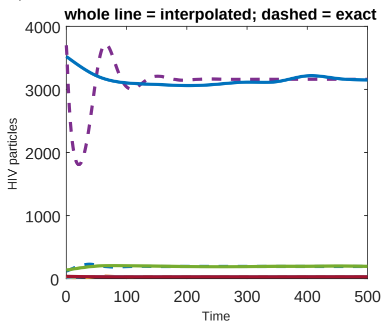
Figure 5.7: Computer simulations of PIP with $\eta(t) = 0.8\sin(3t/T)$. If the number of observation points are doubled, the accuracy in estimation of η increases considerably for noisy data. If the number of observation points are halved for the perfect data, the accuracy decreases, but is still good.



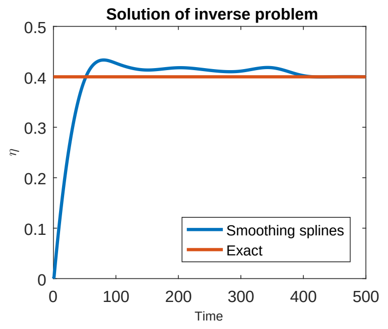
(a) Forward solution: $\eta(t) = 0.8t/T$.



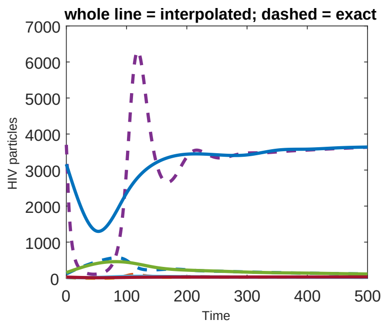
(b) Estimated η: $\eta(t) = 0.8t/T$. Maximal error: 0.029



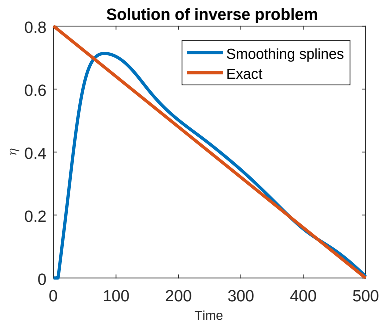
(c) Forward solution: $\eta(t) \equiv 0.4$.



(d) Estimated η: $\eta(t) \equiv 0.4$. Maximal error: 0.400



(e) Forward solution: $\eta(t) = 0.8(1 - t/T)$.



(f) Estimated η: $\eta(t) = 0.8(1 - t/T)$. Maximal error: 0.800

Figure 5.8: Computer simulations of PIP with various $\eta(t)$. Observation points: 10. Noise: 5%. Initial value at equilibrium 1.

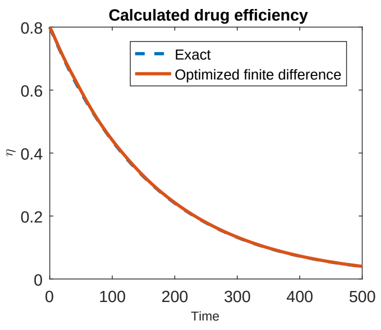
It is clear from these plots that the perhaps most important factor for accurate curve fitting is the initial value. If the efficacy of the drug changes smoothly, interpolation with sparsely located observations is sufficient to obtain a good reconstruction of the drug efficacy parameter η, even for large noise level, see Fig. 5.6 and 5.7. But if the efficacy has jumped abruptly, for instance if the drug has been recently introduced or if the dose has been changed, it is more difficult to reconstruct η accurately (see Fig. 5.8). In these cases measurement of virus and T-cell concentrations must be made at much smaller time intervals until the concentrations of viruses and T-cells stabilize.

5.2.2 Continuous observations at a time-subinterval

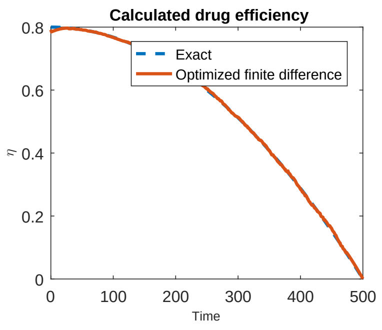
This situation is perhaps less likely to occur in practice, but since this was how the problem was originally formulated in [4], we will consider this case also. We will use the adaptive algorithm for calculating derivatives. We will first consider the case that we have observations on the entire domain, Ω_T, which should be easier than the previous problem (Fig. 5.9). And then we will consider observations on $I = [100, 400] \subset [0, 500] = \Omega_T$ (Fig. 5.10).
If we only have observations on the subinterval $I \subset \Omega_T$, we are using linear extrapolation to extend to the entire time domain, using 10% at each endpoint of the function graph for fitting a straight line. In this case, it is the time moments 101-131 and 370-400 that are used.

As one can see, the reconstruction of η is excellent, if sufficient observations are available. This applies even if the initial value is such that η cannot be assumed to change continuously at $t = 0$; see Fig. 5.11.

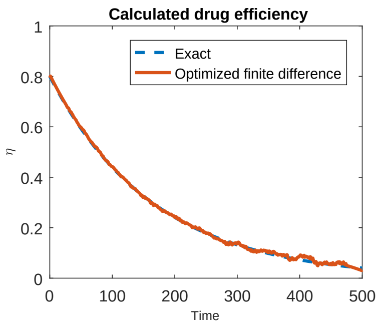
Observations at the entire time domain



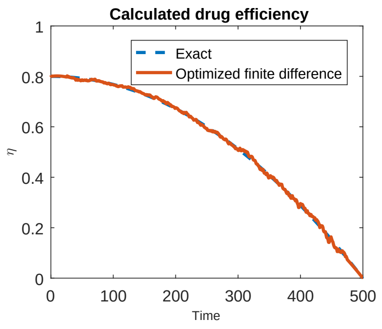
(a) $\eta(t) = 0.8e^{-3t/T}$. No noise. Maximal error: $5.56 \cdot 10^{-3}$.



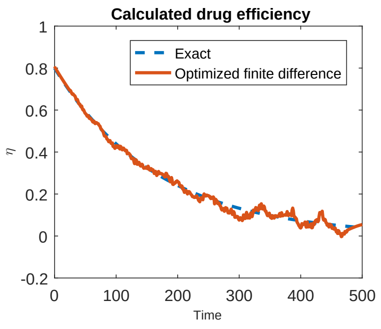
(b) $\eta(t) = 0.8(1 - t/T)$. No noise. Maximal error: $1.51 \cdot 10^{-2}$.



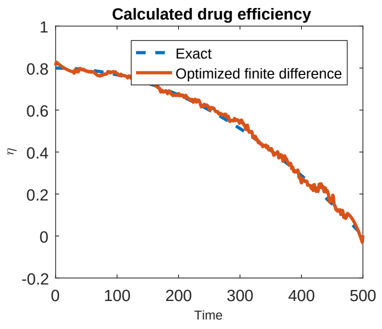
(c) $\eta(t) = 0.8e^{-3t/T}$. 5% noise. Maximal error: $2.10 \cdot 10^{-2}$.



(d) $\eta(t) = 0.8(1 - t/T)$. 10% noise. Maximal error: $2.42 \cdot 10^{-2}$.



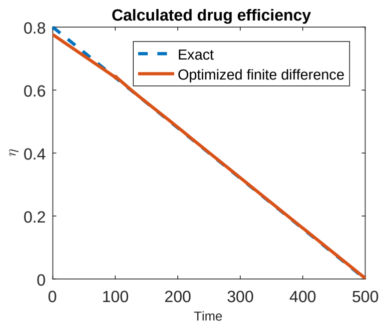
(e) $\eta(t) = 0.8e^{-3t/T}$. 15% noise. Maximal error: $5.94 \cdot 10^{-2}$.



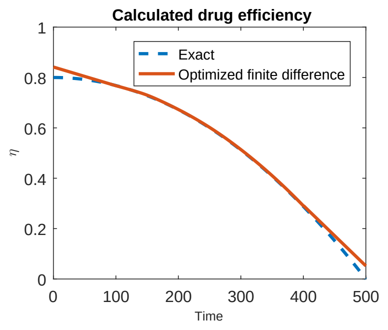
(f) $\eta(t) = 0.8(1 - t/T)$. 20% noise. Maximal error: $4.63 \cdot 10^{-2}$.

Figure 5.9: Computer simulations of PIP with $\eta(t)$ exponential (left) or quadratic (right) function. Initial value at equilibrium 2. Observations assumed to be known at the entire time interval.

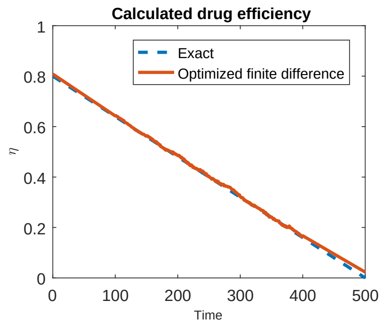
Observation at smaller intervals



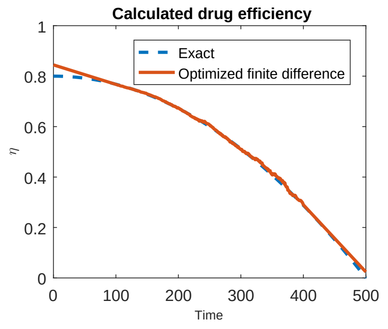
(a) $\eta(t) = -0.8t/T$. No noise. Maximal error: $5.38 \cdot 10^{-2}$.



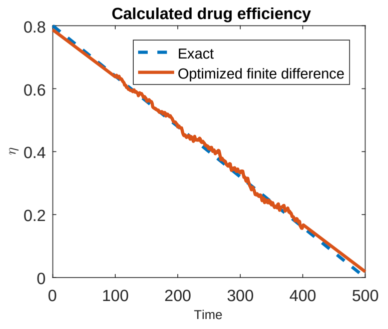
(b) $\eta(t) = 0.8(1 - t/T)$. No noise. Maximal error: $5.20 \cdot 10^{-2}$.



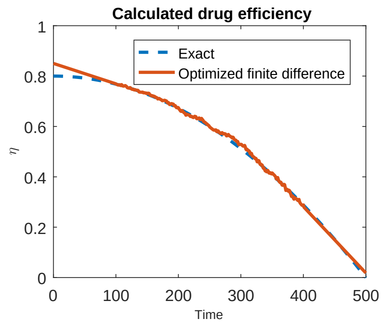
(c) $\eta(t) = -0.8t/T$. 3% noise. Maximal error: $2.31 \cdot 10^{-2}$.



(d) $\eta(t) = 0.8(1 - t/T)$. 3% noise. Maximal error: $4.47 \cdot 10^{-2}$.



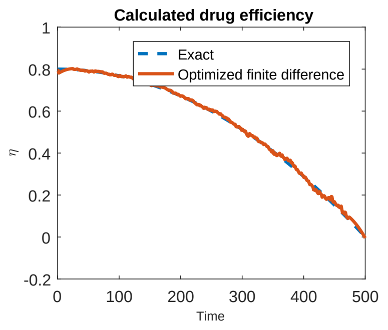
(e) $\eta(t) = -0.8t/T$. 10% noise. Maximal error: $2.82 \cdot 10^{-2}$.



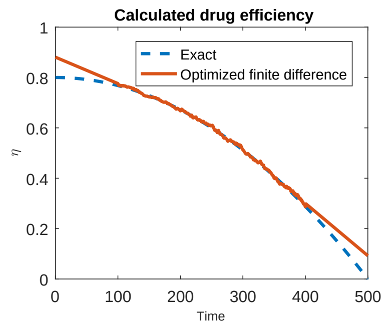
(f) $\eta(t) = 0.8(1 - t/T)$. 10% noise. Maximal error: $5.03 \cdot 10^{-2}$.

Figure 5.10: Computer simulations of PIP with $\eta(t)$ linear (left) or quadratic (right) function. Initial value at equilibrium 2. Observations assumed to be contained in the interval $[100, 400] \subset [0, 500]$.

Effect of initial values



(a) Observations in $[0, 500]$. Maximal error: $2.83 \cdot 10^{-2}$.



(b) Observations in $[100, 400]$. Maximal error: $9.24 \cdot 10^{-2}$.

Figure 5.11: Computer simulations of PIP with $\eta(t) = 0.8(1 - t/T)$. Initial value at equilibrium 1. 10% noise.

5.3 Improving the results

We will test the two methods presented in this thesis for improving the results. That is:

1. Algorithm 1: Minimization of the Lagrangian (3.18).

2. Algorithm 2: Refinement of the time mesh, such that the derivatives are computed with optimal accuracy.

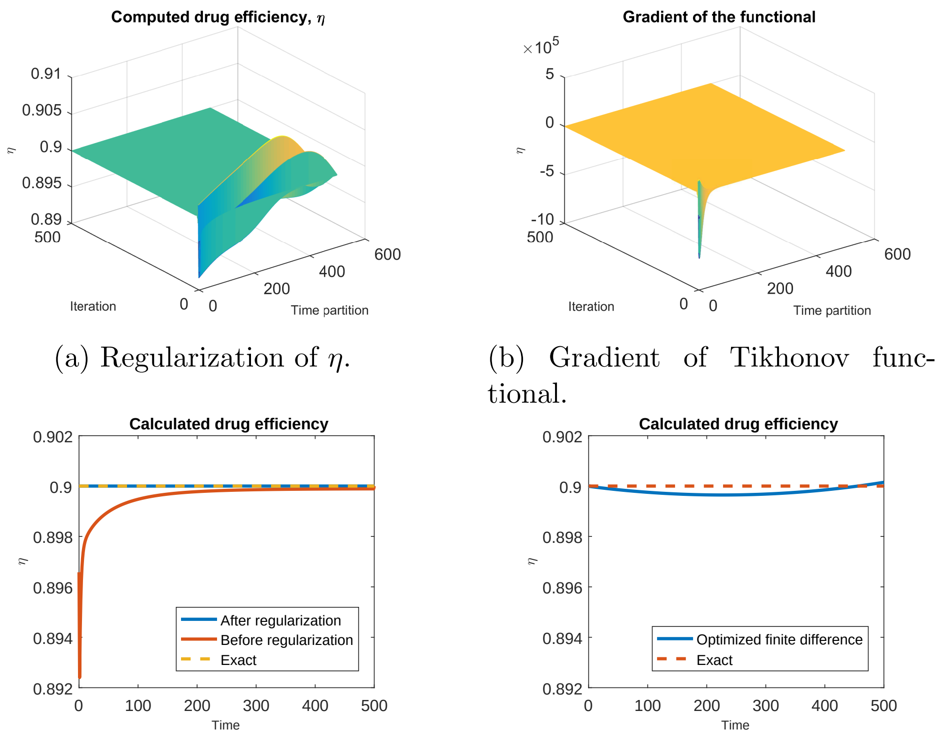
When solving inverse problems in general, the regularization parameter, γ, is typically a small number, but in this case it is not small. The previous results suggest that we already can determine η with an error of about $\varepsilon^0 \sim 10^{-2}$ for continuous observations, or $\varepsilon^0 \sim 0.1$ for discrete observations. The L_∞-norm of the gradient is typically of the order of magnitude $\sim 10^6$. So, according to (4.29) we should choose $\gamma_0 \sim 10^8$. More precisely, we will set

$$\gamma^0 = -\alpha||u_{\tau 2}(\lambda_{\tau 1} - \lambda_{\tau 3})||_{L_\infty} N, \tag{5.5}$$

where N is either 10, 100 or 1000.
Some improvements could be observed with these settings, but the Lagrangian never quite converged to the correct value, except for one case. In fact, it turned out to be extremely difficult to choose a proper regularization parameter: either it was too small and took η outside of M_η (leading to very peculiar results) or it was too large and did almost nothing to improve η.

5.3.1 Cases when solution can be improved

As an example to demonstrate that the Tikhonov functional has potential to be useful in some cases, consider Figure 5.12, where $\eta \equiv 0.9$. Regularization gives an almost perfect reconstruction of η, much better than the direct method (Fig. 5.12 c), even if the optimal time mesh is used (Fig. 5.12 d).



(a) Regularization of η.

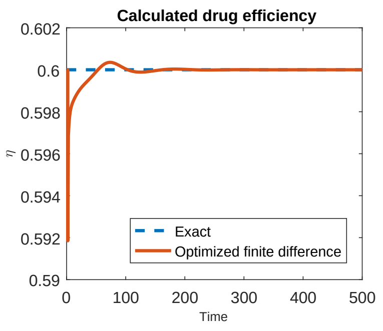
(b) Gradient of Tikhonov functional.

(c) Calculated η before and after regularization. Maximal error of regularized solution: $1.26 \cdot 10^{-7}$.

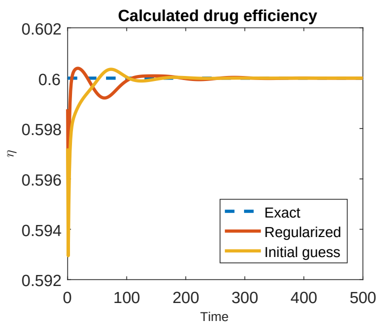
(d) Calculated η efter optimizing step lengt for finite differences. Maximal error: $3.54 \cdot 10^{-4}$.

Figure 5.12: Computer simulations of PIP with $\eta(t) \equiv 0.9$. Both Tikhonov regularization and optimal step-size for finite differences improves the solution, but Tikhonov regularization is better.

When considering the function $\eta(t) \equiv 0.7$, the good results demonstrated in Fig. 5.12 are not quite achieved, however, see Fig. 5.13. But even this time, the method based on Tikhonov regularization delivers a more accurate reconstruction of η than the direct solution on the adaptive time mesh. Furthermore, Algorithm 1 is several times faster than Algorithm 2.



(a) Calculated η after optimizing step length for finite differences. Computing time: 146.2 s.



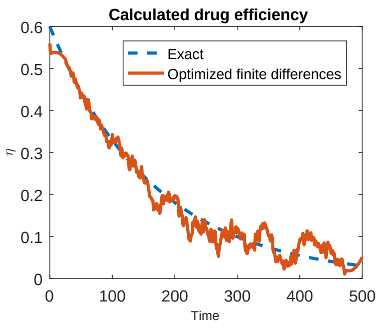
(b) Calculated η before and after regularization. Compuing time: 3.06 s.

Figure 5.13: Computer simulations of PIP with $\eta(t) \equiv 0.6$. Both Tikhonov regularization and optimal step-size for finite differences improves the solution, but Tikhonov regularization is better. No noise is present and the initial value of (3.1) is chosen as equilibrium 1.

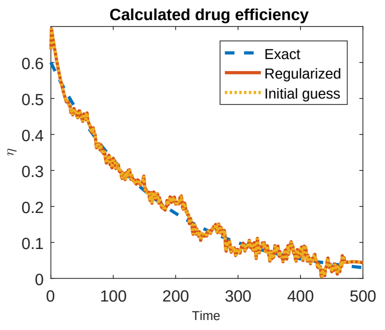
5.3.2 Cases when the solution cannot be improved

Noisy or sparse observations

When we consider observations of the solution to (3.1) with 10% noise, none of the methods are better than the direct solution on the coarse time partition, $\mathcal{J}_\tau$. See Figure 5.14) for an example, using $\eta(t) = 0.6e^{3t/T}$. Similar results occurs if we try to use Tikhonov regularization to improve the smoothing splines interpolated solution from sparse observations. For the adaptive mesh method, it is not surprising that no improvement can be seen, since the optimal step size is quite large for such noisy observations. However, there is almost no difference between the initial guess and the regularized solution either (Fig. 5.14 b).



(a) Calculated η efter optimizing step lengt for finite differences. $||e||_2 = 0.10$. $||e||_2 = 0.11$. Computing time: 0.96 s.

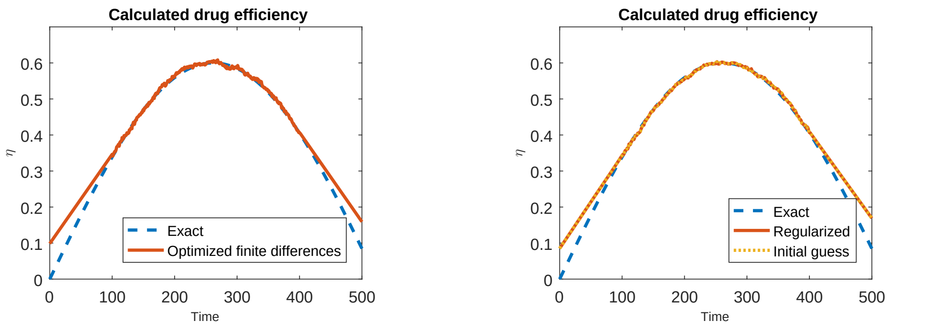


(b) Calculated η before and after regularization. Computing time: 1.95 s.

Figure 5.14: Computer simulations of PIP with $\eta(t) = 0.6e^{3t/T}$ at 10% noise level.

Extrapolation outside observation interval

If we try to apply Tikhonov regularization on an approximation where we have extended η beyond the observation interval, no improvement can be observed, see Figure 5.15. Here, it should be noted that the initial value of (3.1) is assumed known in the algorithms used in this study. If we only have observations in an interval $[T_1, T_2] \subset [0, T]$, then we obviously do not know the initial value. It is therefore likely that better results, in this case, could be achieved when also the initial value is considered an unknown parameter, which should be estimated.



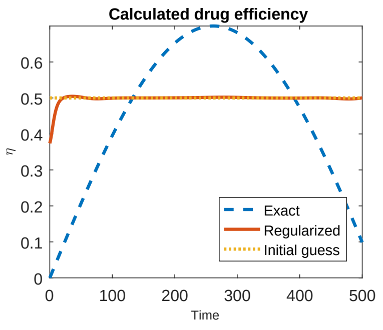
(a) Calculated η efter optimizing step lengt for finite differences. $||e||_2 = 0.056$. $||e||_\infty = 0.16$. Computing time: 1.16 s.

(b) Calculated η before and after regularization. $||e||_2 = 0.066$. $||e||_\infty = 0.14$. Computing time: 2.09 s.

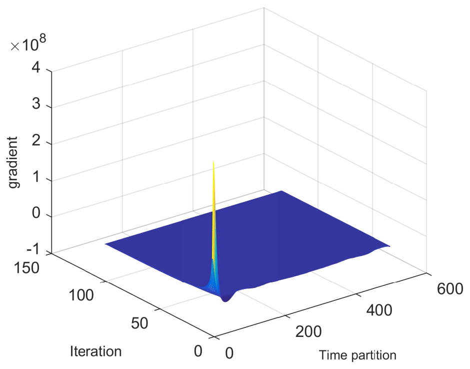
Figure 5.15: Computer simulations of PIP with $\eta(t) = 0.6\sin(3t/T)$ at 3% noise level, and observation interval $[100, 400]$.

Initial guess far from true solution

If we have not obtained a good first guess by solving for η by elementary methods, we will not be able to obtain the correct solution by the gradient method. For an example of this, see Figure 5.16 (a), where we have simulated Algorithm 1 with $\eta^0(t) = 0.5$, whereas the true $\eta(t) = 0.6\sin(3t/T)$. Although some steps are taken in the correct direction, the algorithm eventually terminates because η^n stops changing. An explanation is suggested if we look at Figure 5.16 (b): the gradient is initially huge at the first few time-steps, but soon levels out to a more moderate size ($\sim 10^5$), and since we were required to set the regularization parameter to suit this huge gradient, little change occurs to η for other gradient values.



(a) Calculated η before and after regularization. Computing time: 3.57 s.



(b) Evolution of gradient during simulation. The 2-norm of the gradient at last iteration is about 10^5.

Figure 5.16: Computer simulations of PIP with $\eta(t) = 0.6\sin(3t/T)$. No noise, $\eta^0(t) = 0.5$.

All in all, these results suggest that the usefulness of Tikhonov regularization for reconstruction of the drug efficacy parameter is very limited. In practice it is more advisable to use elementary techniques to get a rough estimate of the drug efficacy, combined with some signal processing technique, to remove noise.

Chapter 6

Discussion and conclusion

Our numerical tests show that Inverse Problem 1 can accurately be solved by elementary methods. Sparsely distributed observations can be interpolated to a finer time mesh by smoothing splines. Then, the parameter η can be reconstructed by simply solving equation (4.31) explicitly, using central finite differences (4.32). If desired, this can subsequently be followed by linear extrapolation to a wider time domain.

Methods based on minimizing the Tikhonov functional (3.16) has turned out to be challenging to implement, due to the importance of both an accurate choice of an exceptionally large regularization parameter, γ, and an accurate initial guess of the drug efficacy, η^0. For most choices of γ, the methods either take η outside the set of admissible functions, M_η, (i.e. too small regularization parameter) or fails to improve the initial guess (i.e. too large regularization parameter).

Perhaps the Tikhonov regularization procedure could better extrapolate the parameter η backwards in time if also the initial value, t_0, is considered an unknown parameter, that should be estimated. Actually, that is a more plausible situation, since we cannot assume that we know the initial values of (3.1), if we do not have any observations at $t = 0$.

Nevertheless, for low noise levels, Algorithm 1 improves the solutions obtained by the direct methods - although it does not improve the solutions from noisy observations, and it requires that a first guess, η_0, previously has been obtained by solving (4.31). Another algorithm, Algorithm 2, based on refining the time mesh where the truncation errors of finite differences is large, also improves the results, albeit at a higher computational cost and slightly lower accuracy than Algorithm 1. It is recommended that Algorithm 1 is used if higher accuracy is desired, but for most applications explicit solution of (4.31) on a coarse time partition is probably adequate. It is more likely that improved signal processing techniques, to remove noise are more useful here than improved regularization techniques.

Once the efficacy, η, has been determined, the drug dose can be optimized. In [26] a functional is suggested which might be used in this case. The authors of [26] simultaneously minimized the dose of chemotherapeutic and immunotherapeutic drugs at the same time as the drug response was maximized.

Drugs for HIV-treatment has to be taken for the rest of the life, and might cause side-effects, and it is therefore of great interest to reduce the drug-burden of the patient, just as it is for anti-cancer drugs. We propose that a functional analogous to functional (7) in [26] should be minimized for this task:

$$I(d) = \int_{\Omega_T} (w_1 d + w_2 V + w_3 \bar{T}_{pre-RT} + w_4 \bar{T}_{post-RT})^2 dx. \tag{6.1}$$

Here, w_i are weights, d is the drug dose and $V, \bar{T}_{pre-RT}$ and $\bar{T}_{post-RT}$ corresponds to viruses and pre- and post-RT infected T-cells, correspondingly. Thus, the dose will be minimized along with viral particles as well as infected cells. To use this in practice, a model linking dose and efficacy to the clinical response is required, such that suitable weights, w_1, w_2, w_3, w_4 can be determined.

Finally, it should be pointed out that the model of the dynamics of HIV infection under treatment of an RT-inhibitor, suggested in [27], is only for mono-therapy with a single drug. Usually, HIV-treatment consists of a combination of drugs to enhance the drug response and suppress development of drug resistance of HIV. This model ought thus to be completed with a model for several drugs simultaneously, which the authors of [27] claims that they will work on in the future. Perhaps, that model could also be subject for the techniques presented in this thesis.

Appendix A

Reverse transcriptase inhibitors

The central dogma of molecular biology states that the flow of genetic information in organisms goes from the storage in DNA, through mRNA (messenger-RNA) to protein synthesis in the ribosomes. The process of converting the genetic information from DNA to mRNA is called *transcription*.
Unlike non-viral organisms however, the genetic material of HIV - a virus targeting certain cells of the immune system called T cells (more specifically CD4$^+$ T cells) - is stored in RNA rather than DNA. From there, it is *reversely transcribed* into DNA, which is incorporated into the DNA of the infected cell. Once the genetic material of the HIV is incorporated into the infected T cell's genome, new viruses will be produced by the cell, together with the cell's own proteins [23].
To perform the reverse transcription of RNA into DNA, HIV carries an enzyme called *reverse transcriptase*, that catalyzes the reverse transcription. Drugs inhibiting this enzyme will prevent the virus from reproducing. Such drugs are called *reverse transcriptase inhibitors* and is one of two main classes of drugs that are usually combined when HIV infection is treated pharmacologically - the other class of drugs inhibit another viral enzyme known as *protease* [23].
In the ODE model discussed in this thesis, which was developed by [27], there are three kinds of CD4$^+$ T cells considered: completely healthy and uninfected T cells, T cells that are infected, but where reverse transcription has not yet occurred, and T cells where reverse transcription has already occurred.
The life-cycle of HIV, including reverse transcription, is illustrated in Figure A.1.

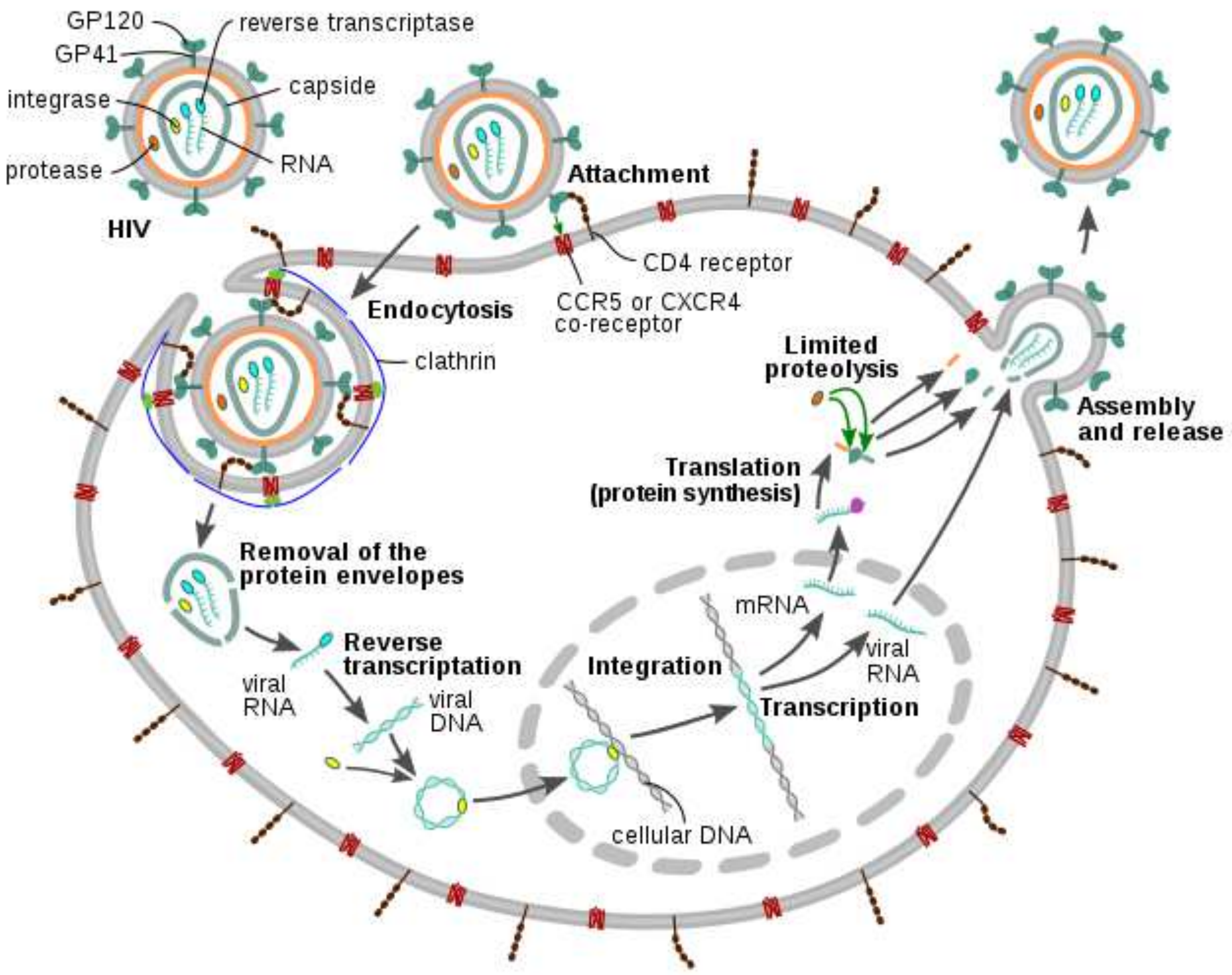


Figure A.1: Life cycle of HIV. The reverse transcription of viral RNA into DNA is catalysed by reverse transcriptase. The figure was obtained from: https://upload.wikimedia.org/wikipedia/commons/5/57/HIV-replication-cycle-en.svg

Appendix B

Matlab programs

There will be two different main programs, used for the two cases when observations are either sparsely distributed or continuous.

B.1 Programs used in both problems

B.1.1 η calculator

```
%Creates a vector of function values for various real−valued
    functions
%defined on some time mesh.
%ETA = ExactEta(SCALING,FLAG,TIME_MESH) returns a vector
    containing the
%desired function values given: SCALING = a scaling factor which
    should be
%between 0 and 1, TIME_MESH = the desired time mesh and FLAG = a
    number
%in {0,1,2,3,4,5,6} representing the following kinds of functions:
%0. const. 1. x 2. −x 3. c∗sin 4. c∗exp(−x) 5. c∗(1 − xˆ2) 6. 2c +
%c∗sin(x).
function ext_eta = ExactEta(scaling,flag,time_mesh)
%Input: 1. scaling factor
%2. function flag (0. const. 1. +linear 2. −linear 3. sin 4. exp(−
    x)
%5. quadratic 6. alternative sine)
%time = 0;
if scaling > 1 || scaling < 0
    error('Non−admissible values of \eta')
end
s = zeros(1,length(time_mesh));
```

```
for i = 1:length(time_mesh)
    s(i) = 3*time_mesh(i)/time_mesh(end);
end
    if flag == 1
        ext_eta = scaling*s/3;
    elseif flag == 2
        ext_eta = scaling*(1-s/3);
    elseif flag == 3
        ext_eta = scaling*sin(s);
    elseif flag == 4
        ext_eta = scaling*exp(-s);
    elseif flag == 5
        ext_eta = scaling*(1-(time_mesh/time_mesh(end)).^2);
    elseif flag == 6
        if scaling > 0.6666
            error('Some values of \eta are not admissible!')
        end
        ext_eta = scaling + (scaling/2)*sin(3*s);
    else
        ext_eta = scaling*ones(1,length(time_mesh));
    end
end
```

B.1.2 Forward problem solver

```
%Computes the exact solution of the model problem.
%[G,G_NOISE] = ExactNewton(ETA,TIME_MESH,RAND,SYST) returns both
    ideal noise-
%free data, G, and noisy data, G_NOISE. Required input is the
    exact value of
%the efficacy parameter, ETA, the time mesh, TIME_MESH, and the
    levels of
%random (RAND) and systematic (SYS) error.
function [g,g_brus] = ExactNewton(eta,time_mesh,noise_level,
    err_sys)

nodes = length(time_mesh); % Number of nodes in the time partition

ustart = [120;22;35;3700];    %Initial values for u
%ustart = [581;44;14;1469];
%ustart = [5;5;5;5];
%Equilibrium values without treatment are approximately
    [120;22;35;3700].
```

```
%Equilibrium values at eta = 0.8 is about [581;44;14;1469].

v=ustart;
MaxIter = 1000;  % maximal number of iterations in Newton method

u=zeros(4,nodes);
u(:,1) = ustart;

brus = zeros(4,nodes);
dt = zeros(1,length(time_mesh));
for i = 1:length(dt)-1
    dt(i) = time_mesh(i+1)-time_mesh(i);   %Time step
end

for i = 1:nodes-1
    tol=1;
    iter=0;
    while tol>10^(-5) && iter < MaxIter   %Newton iterations
        F= v-u(:,i)-dt(i)*Forwardfunc(v,eta(i));
        J=eye(length(ustart)) - dt(i)*ForwardfuncJac(v,eta(i));
        dv = -J\F;
        v=v+dv; %The Newton iteration
        iter = iter + 1 ;
        tol = norm(dv,inf);
    end
    if iter == MaxIter        %If the Newton meth. does not
       converge
        warning('No convergence in the Newton method at time step
           ')
        time_mesh(i)
        u(:,i+1) = u(:,i);
        continue
    end
    if v(1) < 0 || v(2) < 0 || v(3) < 0 || v(4) < 0
        error('Exact solution algorithm yields invalid solution.
           Try increasing the number of nodes.')
    end
    brus(1,i) = (2*rand-1)*noise_level; %Generate random noise.
    brus(2,i) = (2*rand-1)*noise_level;
    brus(3,i) = (2*rand-1)*noise_level;
    brus(4,i) = (2*rand-1)*noise_level;
    u(:,i+1) = v;                        %Update u.
```

```
end

g=u;
g_brus = g + g.*brus + g.*err_sys;        %Add random and systematic
    error.
end
```

B.2 Identification of η with sparsely distributed observations

B.2.1 Main program

```
%% Clear variables and windows
clear all
close all

%% Create observation values
T = 500;
time_mesh = 0:T;
function_flag = 4;
%0. const. 1. +linear 2. -linear 3. sin 4. exp(-x) 5. quadratic
%6. alternative sine
scaling = 0.8;                 %Scaling for function.
noise_level = 0;           %Random error in observations. Should be
    in [0,1).
err_sys = 0;                   %Systematic error. Should be in (-1,1)
    .
observation_points = 5;     %Number of observation points.

eta = ExactEta(scaling,function_flag,time_mesh);
[g,g_brus] = ExactNewton(eta,time_mesh,noise_level,err_sys);

step_size = floor(length(time_mesh)/observation_points);

g_obs = g_brus(:,1:step_size:end);
t_obs = time_mesh(1:step_size:end);

noise = floor(100*noise_level); %For output file names
%% Interpolate to get approximate observation values
%Interpolation and smoothing of data using smoothing splines
    ___________
[p_smooth1, P1] = csaps(t_obs,g_obs(1,:));
```

```
[p_smooth2, P2] = csaps(t_obs,g_obs(2,:));
[p_smooth3, P3] = csaps(t_obs,g_obs(3,:));
[p_smooth4, P4] = csaps(t_obs,g_obs(4,:));

Y = zeros(4,length(time_mesh));

Y(1,:) = ppval(p_smooth1,time_mesh);
Y(2,:) = ppval(p_smooth2,time_mesh);
Y(3,:) = ppval(p_smooth3,time_mesh);
Y(4,:) = ppval(p_smooth4,time_mesh);

%Force the interpolated solution to be nonnegative
for i = 1:T+1
    if Y(1,i) < 0
        Y(1,i) = 0;
    end
    if Y(2,i) < 0
        Y(2,i) = 0;
    end
    if Y(3,i) < 0
        Y(3,i) = 0;
    end
    if Y(4,i) < 0
        Y(4,i) = 0;
    end
end
%Visualization of the results
    ------------------------------------------------------------
%Parameters for pictures
    -------------------------------------------------------------------
width = 3.5;
height = 2.8;
alw = 0.75;     % AxesLineWidth
fsz = 8;        % Fontsize
lw = 1.5;       % LineWidth
msz = 8;        % MarkerSize
%
    ------------------------------------------------------------------------------------------------

figure(1)
pos = get(gcf, 'Position');
set(gcf, 'Position', [pos(1) pos(2) width*100, height*100]); %Set
```

```
    size
set(gca, 'FontSize', fsz, 'LineWidth', alw);            %Set
    properties

plot(time_mesh,g,'--','LineWidth',2)
hold on
plot(time_mesh,Y,'LineWidth',2)
title('whole line = interpolated; dashed = exact','FontSize',10)
xlabel('Time','FontSize',8)
ylabel('HIV particles','FontSize',8)

set(gcf,'InvertHardcopy','on');
set(gcf,'PaperUnits', 'inches');
papersize = get(gcf, 'PaperSize');
left = (papersize(1)- width)/2;
bottom = (papersize(2)- height)/2;
myfiguresize = [left, bottom, width, height];
set(gcf,'PaperPosition', myfiguresize);

str_title1 = ['for-f', num2str(function_flag),'n', num2str(noise),
    'o',num2str(observation_points)];
print(str_title1,'-depsc','-r500');
%% Deduce parameter Eta, by solving this system

delta = 0.26;                          %Value of parameter delta.
alfa=0.4;                              %Value of parameter alpha.
nodes = length(time_mesh);
time_step = zeros(1,nodes-1);
for i = 1:nodes-1
    time_step(i) = time_mesh(i+1)-time_mesh(i);
end                                    %Time step for discrete
    derivatives.
Y_prim = zeros(4,nodes);
eta_calc = zeros(1,nodes);

%Compute derivatives of g.
for i = 2:nodes-1
    Y_prim(:,i) = (Y(:,i+1)-Y(:,i-1))/(2*time_step(i));
end
Y_prim(:,1) = (Y(:,2)-Y(:,1))/time_step(1);
Y_prim(:,end) = (Y(:,end)-Y(:,end-1))/time_step(end);

```

```
%Compute analytic eta inside observation interval.
for i = 1:nodes
    eta_calc(i) = 1 - (delta*Y(3,i)+Y_prim(3,i))/(alfa*Y(2,i));
end

%Remove singularities by Hampel filtering
if min(Y(2,:)) < 0.1
    eta_filter = hampel(eta_calc,floor(0.1*nodes),2);
else
    eta_filter = eta_calc;
end

%Perform smoothing on data
smoothing_window = 2*round(T/20)+1;
eta_smooth = smooth(eta_filter,smoothing_window,'loess');
eta_filter = eta_smooth;

%Force eta to belong to [0,1]
for i = 1:nodes
    if eta_filter(i) < 0
        eta_filter(i) = 0;
    elseif eta_filter(i) > 1
        eta_filter(i) = 1;
    end
end

%% Visualize data

figure(2)
pos = get(gcf, 'Position');
set(gcf, 'Position', [pos(1) pos(2) width*100, height*100]); %Set
   size
set(gca, 'FontSize', fsz, 'LineWidth', alw);                %Set
   properties

plot(time_mesh,eta_filter,'LineWidth',2)
hold on
plot(time_mesh,eta,'LineWidth',2)
title('Solution of inverse problem','FontSize',10)
legend({'Smoothing splines','Exact'},'FontSize',8)
legend('Location','southeast')
xlabel('Time','FontSize',8)
```

```
ylabel('\eta','FontSize',9)

set(gcf,'InvertHardcopy','on');
set(gcf,'PaperUnits', 'inches');
papersize = get(gcf, 'PaperSize');
left = (papersize(1)- width)/2;
bottom = (papersize(2)- height)/2;
myfiguresize = [left, bottom, width, height];
set(gcf,'PaperPosition', myfiguresize);

str_title2 = ['inv-f', num2str(function_flag),'n', num2str(noise),
    'o',num2str(observation_points)];
print(str_title2,'-depsc','-r500');

err = norm(eta_filter - eta',Inf);
str_title3 = ['maximum error in PIP is ',num2str(err)];
disp(str_title3)
%End of file
```

B.3 Identification of η with continuous observations

B.3.1 Main program

```
%%This is the main program for solving the inverse problem in this
     thesis.
tic
%% Specify fixed parameter values.
clear all
%Time parameters
    ------------------------------------------------------------------
final_time = 500;
obs_start = 0;          %Starting time for observations of true
    solution.
obs_end = 500;          %End time for observations of true
    solution.
%Parameters from the model problem
    ----------------------------------------------
alfa=0.4;               %Value of parameter alpha.
%Optimization parameters
    ------------------------------------------------------------
%gamma_start = 10e+7;       %Initial value of regularization
    parameter.
```

```
optim_maxiter = 20;      %Maximum iterations of optimization alg.
%Noise

noise_level = 0.00;   %Noise level of observations.
err_sys = 0;          %Systematic error of measurements.
noise_flag = 0;       % 0 = no noise, 1 = random noise, 2 = additive noise.
method = 1;           % 1 = CGM; 2 = Adaptive mesh FD; other = basic.

%% Create initial time mesh, observations and starting guess for parameter eta.
time_mesh = [0:0.001:10,10.01:0.01:20,20.1:0.1:30,31:final_time]; % will be adaptively updated.
scaling_factor = 0.7;
function_flag = 3;       %eta(t) = scaling_factor*function
ext_eta = ExactEta(scaling_factor,function_flag,time_mesh); %Exact eta.
[eta_guess,g] = AnalyticEta_destroyed(ext_eta,time_mesh,obs_start,obs_end,noise_flag,noise_level,err_sys); %Observations and first guess for eta.
eta_guess = 0.5*ones(1,length(time_mesh));
if method == 1
    %Run algorithm
    %nodes = length(time_mesh);                          % Number of nodes in the present time mesh.
    eta = eta_guess;
    [u1,ffail] = ForwardNewton(eta,time_mesh);                    %Compute initial forward sol.
    [lambda1,lfail] = AdjointNewton(eta,u1,g,time_mesh,obs_start,obs_end);   %Compute adjoint sol.
    g0 = -alfa*u1(2,:).*(lambda1(1,:)-lambda1(3,:));          % Compute grad.
    gamma_start = norm(g0,Inf)*100;
    bm = 1/gamma_start;                                                   %For Conjugate Gradient algorithm.
    gn = -g0;
    dn = gn;
    eta = eta + bm*dn;
    eta_old = eta_guess;
    i = 0;
    %gamma_start = max(noise_level,10*eps);
    while norm(gn,2) > 0.01 && i < optim_maxiter && norm(eta-eta_old) > 0.0001
        i = i + 1;
        gamma = gamma_start/sqrt(i+1);
        u = ForwardNewton(eta,time_mesh);                    % Update forward and adjoint sol.
        lambda = AdjointNewton(eta,u,g,time_mesh,obs_start,obs_end);
        gm = gamma*(eta-eta_old) - alfa*u(2,:).*(lambda(1,:)-lambda(3,:));           %Update gradient.
        %gamma_start = norm(gm,Inf)*10;
        %gamma = gamma_start/sqrt(i+1);
        bs = (norm(gm)/norm(gn))^2;
        dm = -gm + bs.*dn;
        bm = -(gm*dm')/(gamma*(dm*dm'));                         %beta = -<g,d>/gamma<d,d>
        if norm(gm,2) > norm(gn,2)
            disp('Gradient grows')
            eta_new = (eta_old + eta)/2;
            eta_old = eta;
            eta = eta_new;
            continue
        else
            eta_new = eta + bm*dm;
        end
        eta_old = eta;
        eta = eta_new;
        dn = dm;
        gn = gm;
    end
elseif method == 2
    [ref_mesh,h_opt,diff_error] = mesh_refiner2(time_mesh,g,noise_level,obs_start,obs_end);
    ext_eta2 = ExactEta(scaling_factor,function_flag,ref_mesh);
    [eta,g2] = AnalyticEta_destroyed(ext_eta2,ref_mesh,obs_start,obs_end,noise_flag,noise_level);
else
    eta = eta_guess;
    return
end
```

```
toc
```

Identification of η using only elementary methods

```
%% Solves for parameter eta using elementary methods.
time_mesh = 0:500;
function_flag = 4;
scaling = 0.8;
obs_start = 0;
obs_end = 500;
noise_flag = 0;
noise_level = 0;
ext_eta1 = ExactEta(scaling,function_flag,time_mesh);
[eta_guess1,g_obs1] = AnalyticEta_destroyed(ext_eta1,time_mesh,obs_start,obs_end,noise_flag,noise_level);
[ref_mesh,h_opt,diff_error] = mesh_refiner2(time_mesh,g_obs1,noise_level,obs_start,obs_end);
ext_eta2 = ExactEta(scaling,function_flag,ref_mesh);
[eta_guess2,g_obs2] = AnalyticEta_destroyed(ext_eta2,ref_mesh,obs_start,obs_end,noise_flag,noise_level);
```

Gradient method

```
%%This is the main program for CGM simulations of the Tikhonov functional
%%of the model problem, considered in the thesis.

%% Specify fixed parameter values.
clear all
%Time parameters-------------------------------------------------------------
final_time = 500;
obs_start = 0;          %Starting time for observations of true solution.
obs_end = 500;          %End time for observations of true solution.
%Parameters from the model problem------------------------------------
alfa=0.4;               %Value of parameter alpha.
%Optimization parameters------------------------------------------------
%gamma_start = 10e+7;        %Initial value of regularization parameter.
optim_maxiter = 100;      %Maximum iterations of optimization alg.
```

```matlab
%Noise ------------------------------------------------------------
noise_level = 0.03;   %Noise level of observations.
noise_flag = 1;       % 0 = no noise, 1 = random noise, 2 = additive noise.

%% Create initial time mesh, observations and starting guess for parameter eta.
time_mesh = 0:final_time; % will be adaptively updated.
scaling_factor = 0.6;
function_flag = 3;       %eta(t) = scaling_factor*function
ext_eta = ExactEta(scaling_factor,function_flag,time_mesh); %Exact eta.
[eta_guess,g] = AnalyticEta(ext_eta,time_mesh,obs_start,obs_end,noise_flag,noise_level); %Observations and first guess for eta.
%eta_guess = 0.5*ones(1,length(time_mesh));
%Run algorithm
nodes = length(time_mesh);                                   % Number of nodes in the present time mesh.
eta = zeros(optim_maxiter+1,nodes);                           % Preallocation of eta (for use in for-loop).
eta(1,:) = eta_guess;
[u1,ffail] = ForwardNewton(eta(1,:),time_mesh);                          %Compute initial forward sol.
[lambda1,lfail] = AdjointNewton(eta(1,:),u1,g,time_mesh,obs_start,obs_end);   %Compute adjoint sol.
g0 = -alfa*u1(2,:).*(lambda1(1,:)-lambda1(3,:));             % Compute grad.
gamma_start = norm(g0,Inf)*1000;
grad_hist = zeros(optim_maxiter+1,nodes);
grad_hist(1,:) = g0;                                          %Save gradient for later plot.
bm = 1/gamma_start;                                           %For Conjugate Gradient algorithm.
gn = -g0;
dn = gn;
eta(2,:) = eta(1,:) + bm*dn;
nodes = length(time_mesh);
test_grad = zeros(1,nodes);
for i = 2:optim_maxiter
    gamma = gamma_start/sqrt(i);
    u = ForwardNewton(eta(i,:),time_mesh);                        %
       Update forward and adjoint sol.
        lambda = AdjointNewton(eta(i,:),u,g,time_mesh,obs_start,
           obs_end);
        gm = gamma*(eta(i,:)-eta(1,:)) - alfa*u(2,:).*(lambda(1,:)
           -lambda(3,:));           %Update gradient.
    gamma_start = norm(gm,Inf);
    bs = (norm(gm)/norm(gn))^2;
        dm = -gm + bs.*dn;
        bm = min(-(gm*dm')/(gamma*(dm*dm')),0.1);
                                   %beta = -<g,d>/gamma<d,d>
        grad_hist(i,:) = gm;
                                                      %Save gradient
           for later plot.
    if norm(gm,Inf) > norm(gn,Inf)
        disp('Gradient grows')
        eta(i+1,:) = (eta(i-1,:) + eta(i,:))/2;
        grad_hist(i+1,:) = grad_hist(i,:);
        %dn = dm;
        %gn = gm;
        continue
        elseif norm(gm,Inf) < 0.01
        disp('Algorithm has converged.')
        eta(i+1:end,:) = ones(length(i+1:optim_maxiter+1),1)*eta(i
           ,:);
        grad_hist(i+1:end,:) = ones(length(i+1:optim_maxiter+1),1)
           *grad_hist(i,:);
        break
    else
        eta(i+1,:) = eta(i,:) + bm*dm;
    end
        dn = dm;
        gn = gm;
end
```

While version of gradient method

```
%%This is the main program for CGM simulations of the Tikhonov
   functional
%%of the model problem, considered in the thesis.

%% Specify fixed parameter values.
clear all
```

```
%Time parameters --------------------------------------------------------------
final_time = 500;
obs_start = 0;            %Starting time for observations of true solution.
obs_end = 500;            %End time for observations of true solution.
%Parameters from the model problem ------------------------------------------
alfa=0.4;                 %Value of parameter alpha.
%Optimization parameters -----------------------------------------------------
%gamma_start = 10e+7;        %Initial value of regularization parameter.
optim_maxiter = 20;       %Maximum iterations of optimization alg.
%Noise ------------------------------------------------------------------------

noise_level = 0.05;   %Noise level of observations.
noise_flag = 0;       % 0 = no noise, 1 = random noise, 2 = additive noise.

%% Create initial time mesh, observations and starting guess for parameter eta.
time_mesh = 0:final_time; % will be adaptively updated.
scaling_factor = 0.6;
function_flag = 0;        %eta(t) = scaling_factor*function
ext_eta = ExactEta(scaling_factor,function_flag,time_mesh); %Exact eta.
[eta_guess,g] = AnalyticEta(ext_eta,time_mesh,obs_start,obs_end,noise_flag,noise_level); %Observations and first guess for eta.

%Run algorithm
nodes = length(time_mesh);                                   %Number of nodes in the present time mesh.
eta = eta_guess;
[u1,ffail] = ForwardNewton(eta,time_mesh);                     %Compute initial forward sol.
[lambda1,lfail] = AdjointNewton(eta,u1,g,time_mesh,obs_start,obs_end);   %Compute adjoint sol.
g0 = -alfa*u1(2,:).*(lambda1(1,:)-lambda1(3,:));             %Compute grad.
```

```
gamma_start = norm(g0,Inf)*1000;
bm = 1/gamma_start;     %For Conjugate Gradient algorithm.
gn = -g0;
dn = gn;
eta = eta + bm*dn;
eta_old = eta_guess;
i = 0;
while norm(gn,Inf) > 0.01 && norm(eta-eta_old) > 0.0001 && i < optim_maxiter
     i = i + 1;
    gamma = gamma_start/sqrt(i+1);
     u = ForwardNewton(eta,time_mesh);                          %Update forward and adjoint sol.
          lambda = AdjointNewton(eta,u,g,time_mesh,obs_start,obs_end);
          gm = gamma*(eta-eta_old) - alfa*u(2,:).*(lambda(1,:)-lambda(3,:));          %Update gradient.
     bs = (norm(gm)/norm(gn))^2;
          dm = -gm + bs.*dn;
          bm = min(-(gm*dm')/(gamma*(dm*dm')),0.1);                         %beta = -<g,d>/gamma<d,d>
     if norm(gm,Inf) > norm(gn,Inf)
          disp('Gradient grows')
          eta_new = (eta_old + eta)/2;
          eta_old = eta;
          eta = eta_new;
          continue
          elseif norm(gm,Inf) < 0.01
          disp('Algorithm has converged.')
          break
     else
          eta_new = eta + bm*dm;
     end
          eta_old = eta;
          eta = eta_new;
          dn = dm;
          gn = gm;
end
```

B.3.2 Subprograms

Forward solver

```
%Computes the forward solution of the model problem.
function [u,ffail] = ForwardNewton(eta,time_mesh)
ffail = 0;
t=0;  % initial time

nodes = length(time_mesh)-1;

%ustart = [120;22;35;3700];
%ustart = [233;36;28;2975];    %Initial values for u - write
    correct values
%ustart = [300;10;10;10];
ustart = [300;0;0;50];
%ustart = [581;44;14;1469];
%ustart = [0;0;0;0];
%ustart = [-10;-10;-10;-10];
%ustart = [300;0;0;2];

v=ustart;

%nodes = 100;
MaxIter = 1000;  % maximal number of iterations in Newton method

u = zeros(4,nodes+1);
u(:,1) = ustart;
%u_hist=[u];      %Save all u values for later plot

%final_time = 1000; % here we choose final time
dt = zeros(1,length(time_mesh));
for i = 1:length(dt)-1
     dt(i) = time_mesh(i+1)-time_mesh(i);   %Time step
end

for i = 1:nodes    % Here we define final time
     tol=1;
     iter=0;
     while tol>10^(-5) && iter < MaxIter   %Newton iterations
%     F= v-u(:,i)-dt*Forwardfunc(v,eta(i));
%     J=eye(length(ustart)) - dt.*ForwardfuncJac(v,eta(i));
     F= v-u(:,i)-dt(i)*Forwardfunc(v,eta(i));
     J=eye(length(ustart)) - dt(i)*ForwardfuncJac(v,eta(i));
     dv = -J\F;
      v=v+dv;                %The Newton iteration
      iter = iter +1;
      tol = norm(dv,inf);
     end
     if iter==MaxIter         %If the Newton meth. does not converge
         disp('No convergence in the Newton method')
         break
     end
    %   disp('Newton method converged at iteration:')
     %  iter
     if v(1) < 0 || v(2) < 0 || v(3) < 0 || v(4) < 0
         warning('Forward solution algorithm yields invalid
            solution. Try increasing the number of nodes in the
            time partition.')
         ffail = 1;
         return
     end
     u(:,i+1) =  v;
     t=t+dt;
    %u_hist=[u_hist,u];

 end

% to see only u2:
% plot(0:1/length(u(1,:)):(1-1/length(u(1,:))),u(2,:))

%here we define mesh for time
%time_mesh=0:dt:final_time;

%figure
%plot(time_mesh,u,'LineWidth',2)

%xlabel('time interval');
%ylabel('solution');

%legend('u_1','u_2','u_3','u_4');

%str_title = ['forward solution for eta=', num2str(eta)];
%title(str_title)

```

```
end
```

```
function [ f ] = func(u,eta)
%The function f in the problem: u'=f(u)

f=zeros(4,1);

s=10;
mu=0.01;
k=2.4e-5;
mu1=0.015;
alfa=0.4;
b=0.05;
delta=0.26;
c=2.4;
N=1000;


f(1)=s-k*u(1)*u(4)-mu*u(1)+eta*alfa*u(2)+b*u(2);
f(2)=k*u(1)*u(4)-mu1*u(2)-alfa*u(2)-b*u(2);
f(3)=alfa*u(2)-eta*alfa*u(2)-delta*u(3);
f(4)=N*delta*u(3)-c*u(4);
end
```

```
function [Jac ] = funcJac(u,eta)
% The Jacobian for our problem
% Input: u       (point)
% Output: Jac    (Jacobian in point u)

k=2.4e-5;
mu=0.01;
mu1=.015;
alpha=0.4;
b=0.05;
delta=0.26;
c=2.4;
N=1000;


Jac=[-k*u(4) - mu, eta*alpha + b, 0, -k*u(1);
        k*u(4), -(mu1 + alpha + b),0, k*u(1);
        0, (1- eta)*alpha, -delta, 0;
        0,0, N*delta, -c];

end
```

Adjoint problem solver

```
%Computes the adjoint solution of the model problem.
function [lambda,lfail] = AdjointNewton(eta, u, g, time_mesh,
   obs_start, obs_end)
lfail = 0;
final_time = time_mesh(end); % here we choose the final time
nodes = length(time_mesh);

t=final_time;  % final  time in adjoint solver, the same final
   time is in the forward problem

lambdastart = [0;0;0;0];    % values for lambda(T)= 0 at the final
    time

w = lambdastart;

MaxIter = 1000;  % maximal number of iterations in Newton's method

lambda = zeros(4,nodes);
lambda(:,nodes) = lambdastart;

%dt = final_time/nodes;   %Time step
dt = zeros(1,length(time_mesh));
for i = 1:length(dt)-1
    dt(i) = time_mesh(i+1)-time_mesh(i);   %Time step
end
dt(end) = time_mesh(end)-time_mesh(end-1);

%i = nodes + 1;
%c = ones(10,nodes);
for i = nodes:-1:2
    adjtol=1;
    adjiter=0;
    iter = 1;
    while adjtol>10^(-5) && adjiter < MaxIter   %Newton iterations
        %if low < i && high > i
```

```
%          adjF = w - lambda(:,i) + dt*adjfunc(w,u(:,i),g(:,i),eta(i));
          if t > obs_start && t < obs_end
              adjF = w - lambda(:,i) + dt(i)*adjfunc(w,u(:,i),g(:,i),eta(i));
          else
              adjF = w - lambda(:,i) + dt(i)*adj2func(w,u(:,i),eta(i));     %Do not include observations outside observation interval.
          end
%          adjJ = eye(length(lambdastart)) + dt.*adjfuncJac(u(:,i),eta(i));
          adjJ = eye(length(lambdastart)) + dt(i)*adjfuncJac(u(:,i),eta(i));
          %c(iter,i) = cond(adjJ);
          dw = -adjJ\adjF;
         w=w+dw;                  %The Newton iteration
         adjiter = adjiter + 1;
         adjtol = norm(dw,inf);
         iter = iter + 1;
     end
     if adjiter==MaxIter          %If the Newton meth. does not converge
          lfail = 1;
          disp('No convergence in the Newton method for adjoint problem')
          break
     end
       %disp('Newton method for adjoint problem converged at iteration:')
       %iter
     %i = i - 1;
     lambda(:,i-1) = w;
     t=t-dt(i);
    %lambda_hist=[lambda_hist,lambda];
    %time_hist = [t,time_hist];

end

%plot(time_hist,lambda,'LineWidth',2)

%xlabel('time interval');
```

```
%ylabel('solution');

%legend('lambda_1','lambda_2','lambda_3','lambda_4');

%str_title = ['adjoint solution for eta=', num2str(eta)];
%title(str_title)

end
```

```
function [ adjf ] = adjfunc(lambda,u,g,eta)
  %The rhs f(lambda) in the adjoint problem: lambda'=f(lambda)

adjf=zeros(4,1);

s=10;
mu=0.01;
k=2.4e-5;
mu1=0.015;
alfa=0.4;
b=0.05;
delta=0.26;
c=2.4;
N=1000;

adjf(1)= lambda(1)*k*u(4) +lambda(1)*mu - lambda(2)*k*u(4) + u(1)
    - g(1);
adjf(2)= lambda(2)*(mu1 + alfa + b) - lambda(1)*(eta*alfa + b)
    -(1-eta)*alfa*lambda(3) + u(2) -g(2);
adjf(3)= lambda(3)*delta -lambda(4)*N*delta + u(3) - g(3);
adjf(4)= lambda(4)*c + lambda(1)*k*u(1) -lambda(2)*k*u(1) + u(4)
    -g(4);

%adjf = adjf + [233;36;28;2975];

end
```

```
function adjf = adj2func(lambda,u,eta)
  %The rhs f(lambda) in the adjoint problem: lambda'=f(lambda)

adjf=zeros(4,1);

mu=0.01;
```

```
k=2.4e-5;
mu1=0.015;
alfa=0.4;
b=0.05;
delta=0.26;
c=2.4;
N=1000;


adjf(1)=  lambda(1)*k*u(4) +lambda(1)*mu - lambda(2)*k*u(4);
adjf(2)=  lambda(2)*(mu1 + alfa + b) - lambda(1)*(eta*alfa + b) -
    (1-eta)*alfa*lambda(3);
adjf(3)=  lambda(3)*delta - lambda(4)*N.*delta;
adjf(4)=  lambda(4)*c + lambda(1)*k*u(1) - lambda(2)*k*u(1);

%adjf = adjf + [233;36;28;2975];

end
```

```
function [adjJac] = adjfuncJac(u,eta)
% The Jacobian for the adjoint problem
% Input: u       (point)
% Output: adjJac    (Jacobian for the adjoint equation in point u)

k=2.4e-5;
mu=0.01;
mu1=.015;
alpha=0.4;
b=0.05;
delta=0.26;
c=2.4;
N=1000;


adjJac=[k*u(4) + mu,-k*u(4), 0, 0;
        -(eta*alpha + b), mu1 + alpha + b, (eta-1)*alpha, 0;
         0, 0, delta, -N*delta;
         k*u(1), -k*u(1), 0, c;];


end
```

Analytic η solver

```
%[eta,eta_guess,g_obs,g_prim]
function [eta_guess,g_obs] = AnalyticEta_destroyed(ext_eta,
    time_mesh,obs_start,obs_end,noise_flag,noise_level,err_sys)
    delta = 0.26;                           %Value of parameter delta.
    alfa=0.4;                               %Value of parameter alpha
    nodes = length(time_mesh)-1;
    time_step = zeros(1,length(time_mesh));
    for i = 2:length(time_step)-1
        time_step(i) = (time_mesh(i+1)-time_mesh(i-1))/2;
    end                                     %Time step for discrete
        derivatives.
    time_step(1) = time_mesh(2)-time_mesh(1);
    time_step(end) = time_mesh(end)-time_mesh(end-1);
    g_prim = zeros(4,nodes+1);
    eta = zeros(1,nodes+1);
    %Simulate the true values of the model problem, with and
        without noise.
    [g,g_brus] = ExactNewton(ext_eta,time_mesh,noise_level,err_sys
        );
    if noise_flag == 1
        g_obs = g_brus;
    %elseif noise_flag == 2
    %    g_obs = g_add;
    else
        g_obs = g;
    end

%Compute derivatives of g.
    for i = 2:nodes
        g_prim(:,i) = (g_obs(:,i+1)-g_obs(:,i-1))/(2*time_step(i))
            ;
    %g_prim(:,i) = (8*g_obs(:,i+1) - 8*g_obs(:,i-1) + g_obs(:,i-2)
        - g_obs(:,i+2))/(12*time_step(i));
    end
    g_prim(:,1) = (g_obs(:,2) - g_obs(:,1))/time_step(1);
    %g_prim(:,2) = (g_obs(:,3) - g_obs(:,1))/(2*time_step(2));
    g_prim(:,end) = (g_obs(:,end) - g_obs(:,end-1))/time_step(end)
        ;
    %g_prim(:,end-1) = (g_obs(:,end) - g_obs(:,end-2))/(2*
        time_step(end-1));

    %Compute analytic eta inside observation interval.
    for i = 1:nodes+1
        eta(i) = 1 - (delta*g_obs(3,i)+g_prim(3,i))/(alfa*g(2,i));
    end
%Smooth noise
    golayframe = round(length(time_mesh)/20)*2+1;
    eta_smooth = smooth(time_mesh,eta,golayframe,'sgolay');
    eta_hampel = hampel(eta_smooth,10);
    eta_smooth = eta_hampel;

    time = 0;
    start_ind = 1;
    while time <= obs_start
        time = time + time_step(start_ind);
        start_ind = start_ind + 1;
    end
    start_ind = start_ind - 1;
    time = 0;
    end_ind = 0;
    while time <= obs_end
        end_ind = end_ind + 1;
        time = time + time_step(end_ind);
    end

    if noise_flag ~= 0
        eta_obs = eta_smooth(1+start_ind:end_ind);
    else
        eta_obs = eta(1+start_ind:end_ind)';
    end

    Lin_ext = floor(nodes/10);

    V2 = ones(Lin_ext,2);
    V1 = V2;
    x = zeros(1,Lin_ext);
    y = x;
    x(end) = obs_end;
    y(1) = obs_start;
    for i = 1:Lin_ext-1
        x(Lin_ext-i) = x(Lin_ext+1-i) - time_step(end_ind - i);
        y(i+1) = y(i) + time_step(start_ind + i - 1);
    end
    V2(:,2) = x';
    V1(:,2) = y';
    l_ext2 = V2\eta_obs(end-(Lin_ext -1):end);
    l_ext1 = V1\eta_obs(1:Lin_ext);
    eta_guess = zeros(1,length(eta));
    for i = 1+start_ind:end_ind
        eta_guess(i) = eta_obs(i-start_ind);
    end

    l = 0;

    for i = 1:start_ind
        eta_guess(i) = l_ext1(1) + l_ext1(2)*l;
        l = l + time_step(i);
    end

    k = obs_end + time_step(end_ind);

    for i = 1+end_ind:nodes+1
        eta_guess(i) = l_ext2(1) + l_ext2(2)*k;
        k = k + time_step(i);
    end

end
```

Mesh refiner

```
function [refined,h_opt,diff_error] = mesh_refiner2(time_mesh,g,
    noise_level,obs_start,obs_end)
error_tolerance = max(noise_level,0.1);
%error_tolerance(1)
noise_level = max(noise_level,eps);
if length(time_mesh) ~= length(g)
    disp('ERROR: length of time mesh and input function must be
       equal!')
    return
end

time_step = zeros(1,length(time_mesh));
for i = 2:length(time_step)-1
        time_step(i) = (time_mesh(i+1)-time_mesh(i-1))/2;
end                                   %Time step for discrete
```

```
    derivatives.
time_step(1) = time_mesh(2)-time_mesh(1);
time_step(end) = time_mesh(end)-time_mesh(end-1);

time = 0;
start_ind = 1;
while time < obs_start
    time = time + time_step(start_ind);
    start_ind = start_ind + 1;
end

time = 0;
end_ind = 1;
while time < obs_end
    time = time + time_step(end_ind);
    end_ind = end_ind + 1;
end
%start_ind
%end_ind
%pause
time_mesh_o = time_mesh(start_ind:end_ind);
g_o = g(:,start_ind:end_ind);
time_step2 = zeros(1,length(time_mesh_o));
for i = 2:length(time_step2)-1
        time_step2(i) = (time_mesh_o(i+1)-time_mesh_o(i-1))/2;
end                                     %Time step for discrete
    derivatives.
time_step2(1) = time_mesh_o(2)-time_mesh_o(1);
time_step2(end) = time_mesh_o(end)-time_mesh_o(end-1);

third_d = zeros(4,length(time_mesh_o));
for i = 3:length(time_mesh_o)-2
    third_d(:,i) = (g_o(:,i+2) - g_o(:,i-2) + 2*g_o(:,i-1) - 2*g_o
        (:,i+1))/(2*time_step2(i)^3);
end
third_d(:,1) = (g_o(:,4) - g_o(:,1) + 2*g_o(:,2) - 2*g_o(:,3))/(2*
    time_step2(1)^3);
%third_d(:,1)
%pause
third_d(:,2) = (g_o(:,4) - g_o(:,1) + 2*g_o(:,2) - 2*g_o(:,3))/(2*
    time_step2(2)^3);
third_d(:,end-1) = (g_o(:,end) - g_o(:,end-3) + 2*g_o(:,end-2) -
    2*g_o(:,end-1))/(2*time_step2(end-1)^3);
third_d(:,end) = (g_o(:,end) - g_o(:,end-3) + 2*g_o(:,end-2) - 2*
    g_o(:,end-1))/(2*time_step2(end)^3);
third_d = abs(third_d);
%third_d(:,1)
%pause
diff_error = zeros(size(third_d));
diff_error(1,:) = third_d(1,:).*time_step2/3; %+ noise_level./
    time_mesh.^2;
diff_error(2,:) = third_d(2,:).*time_step2/3; %+ noise_level./
    time_mesh.^2;
diff_error(3,:) = third_d(3,:).*time_step2/3; %+ noise_level./
    time_mesh.^2;
diff_error(4,:) = third_d(4,:).*time_step2/3; %+ noise_level./
    time_mesh.^2;
large_error = zeros(4,length(time_mesh));
%diff_error(3,1)
%time_step2(1)
%error_tolerance(2)
%pause
if time_step2(1) > sqrt(noise_level*g(3,1)) && diff_error(3,1) >
    error_tolerance
    large_error(:,1) = ones(4,1);
end
%size(g)
%size(diff_error)
%size(error_tolerance)
%pause
if time_step2(2) > sqrt(noise_level*g(3,2)) && diff_error(3,2) >
    error_tolerance
    large_error(:,2) = ones(4,1);
end
%error_tolerance = error_tolerance/2;
for i = 3:length(time_mesh_o)
    if diff_error(1,i) > error_tolerance && time_step2(i) > sqrt(
        noise_level*g_o(1,i))
    large_error(1,i) = 1;
    end
    if diff_error(2,i) > error_tolerance && time_step2(i) > sqrt(
        noise_level*g_o(2,i))
    large_error(2,i) = 1;
    end
    if diff_error(3,i) > error_tolerance && time_step2(i) > sqrt(
        noise_level*g_o(3,i))
     large_error(3,i) = 1;
    end
    if diff_error(4,i) > error_tolerance && time_step2(i) > sqrt(
        noise_level*g_o(4,i))
     large_error(4,i) = 1;
    end
end
%large_error
%pause
h_opt = nthroot(6*noise_level./third_d,3);

refined = [];                   %Start refinement of time mesh ...
refining = [];
k = 1;
r = 1;
    while r < length(time_mesh_o)
        if large_error(3,r+1) == 1 && large_error(3,r) == 0
            refined = [refined,time_mesh_o(k:r)];
        end
        if large_error(3,r) == 0
            r = r + 1;
            if r == length(time_mesh_o)
                refined = [refined,time_mesh_o(k:r)];
            end
            continue
        else
            %j = i;
            while large_error(3,r) == 1 && r < length(time_mesh_o)
                ste = h_opt(r)+h_opt(r+1);
                refining = [refining,time_mesh_o(r):ste:
                    time_mesh_o(r+1)];
                r = r + 1;
                if r == length(time_mesh_o)
                    refining = [refining,time_mesh_o(r)];
                end
            end
            refined = [refined, refining];
            k = r;
        end
        refining = [];
    end

    refined = [time_mesh(1:start_ind-1),refined ,time_mesh(end_ind
        +1:end)];
```

Bibliography

[1] A. Bakushinsky, M. Y. Kokurin, and A. Smirnova, *Iterative methods for ill-posed problems*, Inverse and Ill-Posed Problems, vol. 54, De Gruyter, 2011.

[2] W. Bangerth and A. Joshi, *Adaptive finite element methods for the solution of inverse problems in optical tomography*, Inverse Problems **24** (2008), 034011.

[3] L. Beilina and I. Gainova, *Time-adaptive FEM for distributed parameter identification in biological models*, Applied Inverse Problems, Springer Proceedings in Mathematics & Statistics, vol. 48, Springer, 2013, pp. 37–50.

[4] ———, *Time-adaptive FEM for distributed parameter identification in mathematical model of HIV infection with drug therapy*, Inverse Problems and Applications, Springer Proceedings in Mathematics & Statistics, vol. 120, Springer, 2015, pp. 111–124.

[5] L. Beilina and C. Johnson, *A posteriori error estimation in computational inverse scattering*, Mathematical Models and Methods in Applied Sciences **15** (2013), 37–50.

[6] L. Beilina, E. Karchevskii, and M. Karchevskii, *Numerical linear algebra: Theory and applications*, Springer, New York, 2017.

[7] L. Beilina and M.V. Klibanov, *Approximate global convergence and adaptivity for coefficient inverse problems*, Springer, New York, 2012.

[8] L. Beilina, M.V. Klibanov, and M. Yu Kokurin, *Adaptivity with relaxation for ill-posed problems and global convergence for a coefficient inverse problem*, Journal of Mathematical Sciences **167** (2010), 279–325.

[9] W. S. Cleveland and S. J. Devlin, *Locally weighted regression: an approach to regression analysis by local fitting*, Journal of the American Statistical Association **83** (1988), 596–610.

[10] J. Cullum, *Numerical differentiation and regularization*, SIAM Journal on Numerical Analysis **8** (1971), 254–265.

[11] J. W. Demmel, *Applied numerical linear algebra*, Society of Industrial and Applied Mathematics, 1997.

[12] F. Deutsch, *Existence of best approximations*, Journal of Approximation Theory **28** (1980), 132–154.

[13] T. Feng, N Yan, and W Liu, *Adaptive finite element methods for the identification of distributed parameters in elliptic equation*, Advances in Computational Mathematics **29** (2008), 27–53.

[14] B. Fornberg, *Numerical differentiation of analytic functions*, ACM Transactions on Mathematical Software **7** (1981), 512–526.

[15] A. Griesbaum, B. Kaltenbacher, and B. Vexler, *Efficient computation of the tikhonov regularization parameter by goal-oriented adaptive discretization*, Inverse Problems **24** (2008), 025025.

[16] B. Kaltenbacher, A. Kirchner, and B. Vexler, *Adaptive discretizations for the choice of a tikhonov regularization parameter in nonlinear inverse problems*, Inverse Problems **27** (2011), 125008.

[17] M. V. Klibanov, A. B. Bakushinsky, and L. Beilina, *Why a minimizer of the Tikhonov functional is closer to the exact solution than the first guess*, Journal of Inverse and Ill-Posed Problems **19** (2011), 83–105.

[18] M. M. Lavrentiev, *Some improperly posed problems of mathematical physics*, Springer Tracts in Natural Philosophy, vol. 11, Springer Verlag, Berlin, 1967.

[19] J. M. Lee, *Introduction to smooth manifolds*, Graduate Texts in Mathematics, Springer, New York, 2012.

[20] H. Liu, S. Shah, and W. Jiang, *On-line outlier detection and data cleaning*, Computers and chemical engineering **28** (2004), 1635–1647.

[21] J. N. Lyness and C. B. Moler, *Numerical differentiation of analytic functions*, SIAM Journal on Numerical Analysis **4** (1967), 202–210.

[22] M. Mboup, C. Join, and M. Fliess, *Numerical differentiation with annihilators in noisy environment*, Numerical Algorithms **50** (2009), 439–467.

[23] G. L. Patrick, *An introduction to medicinal chemistry*, Oxford University Press, Oxford, 2013.

[24] C. Persson, *Iteratively regularized finite element method for conductivity reconstruction in a waveguide*, Master's thesis, Chalmers University of Technology, 2016.

[25] W. Rudin, *Principles of mathematical analysis*, 3 ed., McGraw-Hill, 1976.

[26] S. Sharma and G. P. Samanta, *Analysis of the dynamics of a tumor-immune system with chemotherapy and immunotherapy and quadratic optimal control*, Differential Equations and Dynamical Systems **24** (2016), 149–171.

[27] P.K. Srivastava, M. Banerjee, and P. Chandra, *Modelling the drug therapy for HIV infection*, Journal of Biological Systems **17** (2009), 213–223.

[28] G. Teschl, *Ordinary differential equations and dynamical systems*, Graduate Studies in Mathematics, vol. 140, American Mathematical Society, 2012.

[29] A. N. Tikhonov, *On the stability of inverse problems (in Russian)*, Doklady of the USSR Academy of Science **39** (1943), 195–198.

[30] D. Xingsheng, Y. Liangbo, P. Sichun, and D. Meiqing, *An iterative algorithm for solving ill-conditioned linear least squares problems*, Geodesy and Geodynamics **6** (2015), 453–459.