An Introduction to Parameter Estimation

David

May 13, 2008

Abstract

An abstract distributed parameter system is presented along with several techniques for the minimizing of cost functionals. The optimization loops attempts to find parameter values which bring the model system as close to the measured data as possible. Several examples are presented in the form of a diffusion equation, $u_t = c(x)u_{xx}$, where c may be allowed to vary, and where the input data is may have white noise with zero mean. We find that a hybrid BFGS-Steepest Descent optimization scheme produces spectral convergence in the noiseless problem, as well as in the white noise case, until the error reaches the level of the noise.

1 Introduction

Many times in practice one has a model to describe specific phenomena, but the model contains parameters which alter its behavior. In some cases these parameters can be measured directly, for example, measuring the size of some component of a machine. In other occasions the parameters of interest must be inferred from the data. This paper considers techniques the latter. We begin by making a (fairly weak) assumption regarding the nature of the model and its parameters, and explore several techniques for minimizing a least-squares cost functional. Several example cases of the diffusion equation are presented, along with a hybrid BFGS-Steepest Descent algorithm designed to circumvent some of the problems presented by the diffusion equation.

2 Theory

2.1 The Parameter System

The basic parameter fitting problem is introduced in Chapter 6 of [1], which this discussion follows. Our system is given by the model

$$A(c)u = f, \tag{1}$$

where $A(c)$ is a linear operator which depends on our set of parameters c, f is a known "forcing function" and u is a model prediction, which is to be computed and compared to some set of input data, y. In Section 3 I consider an initial value problem in which f is the initial condition and y is a potentially noisy reading of the solution at some time $t_{final} > 0$. In general, we introduce Hilbert spaces to represent the space of observations, $\mathcal{Y}$, the state space $\mathcal{U}$ (which is not necessarily the same as the observation space) and the parameter space $\mathcal{Q}$. For the purposes of this paper, the state and observations spaces will be identical. Then the observed data y can be expressed as

$$y = u + \eta$$

where u solves 1 and η is the noise inherent in the observations.

To estimate the parameters c of the system, we seek to minimize the regularized least-squares functional

$$\min_{c \in \mathcal{Q}} \|A^{-1}(c)f - y\|_2^2 + \gamma J_{reg}(c) \tag{2}$$

where J_{reg} is some function designed to incorporate *a-priori* information into the problem solution and γ is a parameter that adjusts the relative weight of the two objectives. In this paper I take $J_{reg} \equiv 0$ in the Examples section.

2.2 Optimization Methods

Information about optimization techniques comes from [2] and [3]. In order to minimize the cost function (2) one must implement appropriate optimization techniques. This project will focus on a subset of optimization techniques known as the line search. Line search techniques proceed iteratively using the algorithm

$$x_{k+1} = x_k + \alpha p_k,$$

where $\{x_k\}$ is the sequence of approximations of $\arg\min f(x)$, $\alpha > 0$ is a step size and p_k is the search direction. The search direction and step size is usually chosen so that $f(x_{k+1}) < f(x_k)$. If we assume that f is differentiable with gradient ∇f, then the search direction is chosen so that $p_k^T \nabla f < 0$ (ensuring that p_k is a direction of descent).

For the line search methods explored in this paper the iteration procedure proceeds as $p_k = -B_k^{-1}\nabla f_k$, where B_k is a symmetric, nonsingular matrix which may change as a function of x, and $\nabla f_k = \nabla f(x_k)$ is a convenient shorthand. One important fact is that we can ensure that p_k is a descent direction if B_k^{-1} is positive-definite, i.e., $\nabla f_k^T B_k^{-1} \nabla f_k > 0$, so $\nabla f_k^T p_k < 0$.

2.2.1 Wolfe Conditions

In order to make sufficient progress on the line search portion of the optimization, the output must be subject to some guidelines, which are given by the Wolfe Conditions. The first is that the function value ought to decrease proportional to it's derivative at that point at each iterations. This is codified in the inequality known as the *sufficient decrease condition*

$$f(x_k + \alpha p_k) \leq f(x_k) + c_1 \alpha \nabla f_k^T p_k, \tag{3}$$

for some constant $c_1 \in (0, 1)$. However, this condition is always satisfied for very small step sizes (since p_k is a descent direction), so we must have a second condition to prevent the algorithm from taking steps too small to get anywhere. This can be achieved by ensuring that the slope at the candidate location is larger than some multiple of the slope at the previous position. This is given in the *curvature condition*, and is represented mathematically as

$$\nabla f(x_k + \alpha p_k)^T p_k \geq c_2 \nabla f_k^T p_k \tag{4}$$

where $c_2 \in (c_1, 1)$ is constant. This condition is derived from the derivative of the function evaluated along the line, i.e., if $\phi_k(\alpha) = f(x_k + \alpha p_k)$, then (4) is the same as $\phi_k(\alpha)' \geq c_2 \phi_k(0)'$. Together (3) and (4) make up the Wolfe Conditions.

2.2.2 Steepest Descent

The simplest and most obvious example of a line search algorithm is the Steepest Descent method, so named because $B_k \equiv I$, and thus p_k follows the direction of the steepest local descent. If the objective function is a quadratic and the line search is exact the steepest descent descent algorithm gives a quadratic convergence. For general, twice differentiable objective functions the steepest descent method can converge very slowly, by the formula

$$f(x_{k+1}) - f(x^*) \leq r^2 \left(f(x_k) - f(x^*)\right)$$

where r is a scalar in the open interval

$$r \in \left(\frac{\lambda_n - \lambda_1}{\lambda_2 + \lambda_1}, 1\right)$$

where $\lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n$ are the eigenvalues of the exact Hessian at x^* [2].

2.2.3 Newton's Method

The standard version of Newton's Method used for root finding can easily be adapted to searching for extrema to functions, simply by finding the roots of the derivative. Newton's Method for optimization thus approximates f as a quadratic function, with iterations given by

$$x_{k+1} = x_k - \frac{f'(x_k)}{f''(x_k}.$$

In multiple dimensions we have the obvious extension

$$x_{k+1} = x_k - \left(\nabla^2 f(x_k)\right)^{-1} \nabla f(x_k)$$

where $\nabla^2 f_k$ is the Hessian matrix at x_k. Thus Newton's Method is also a line search method outlined above with $B_k = \nabla^2 f_k$ and a unit step parameter α.

Newton's Method for optimization inherits the problems and advantages of the original Newton's method. The principle problem is that the method simply might fail to converge unless you choose the initial guess x_0 sufficiently close to the true solution x^*. Unlike in the root finding technique, not all extrema are created equal. Newton's method will find maxima as easily as minima, so it is important to ensure that the Hessian matrices B_k are positive definite. Some discussion of this in the applied setting is found in 3.1.2. Like the original Newton's method, the optimization method converges quadratically.

2.2.4 Quasi-Newton Methods

In many cases computing the Hessian is computationally intensive, or there is not an analytic formula available for doing so. In cases such as these the Hessian is replaced by some approximation B_k. The BFGS method is the most popular such method, discovered by Broyden, Fletcher, Goldfarb and Shanno. The method begins similar to Newton's Method, with the iteration relation

$$x_{k+1} = x_k - \alpha_k B_k^{-1} \nabla f(x_k)$$

where the Hessian approximation is updated after each step by the knowledge gained from the previous step.

The recursion relation for B_k for the BFGS method is given by the formula

$$B_{k+1} = B_k - \frac{B_k s_k s_k^T B_k}{s_k^T B_k s_k} + \frac{y_k y_k^T}{y_k^T s_k}$$

where $y_k = \nabla f_{k+1} - \nabla f_k$ and $s_k = x_{k+1} - x_k$. The algorithm, however, used B_k^{-1} rather than B_k directly, so it is much more useful if one can compute the inverse directly. Fortunately this is possible, by the use of the Sherman-Morrison-Woodbury formula

$$B_{k+1}^{-1} = \left(I - \frac{s_k y_k^T}{y_k^T s_k}\right) B_k^{-1} \left(I - \frac{y_k s_k^T}{y_k^T s_k}\right) + \frac{s_k s_k^T}{y_k^T s_k}.$$

Then the search direction for the algorithm is defined. Now a step size is to be determined with satisfies the Wolfe Conditions.

3 Examples

3.1 A simple diffusion equation

We begin examining parameter estimation by choosing a very simple test case: the heat equation in one dimension. We take the equation

$$u_t(x,t) = c u_{xx}(x,t)$$

on the unit interval along with periodic boundary conditions

$$u(0,t) = u(1,t).$$

3.1.1 Implementation

The interval is broken up into distinct nodes such that $0 = x_1 < x_2 < x_3 < \cdots < x_N < x_{N+1} = 1$. The grid spacing $\Delta x_n = x_{n+1} - x_n$ was chosen to be constant, then $\Delta x = \frac{1}{N}$ and $x_j = (j-1)\Delta x$. The diffusion term is approximated by the central difference

$$u_{xx}(x_j,t) \approx \frac{u(x_{j+1},t) - 2u(x_j,t) + u(x_{j-1},t)}{\Delta x^2}.$$

At this stage the partial differential equation is turned into a system of ordinary differential equations given by

$$\frac{du_j(t)}{dt} = c\frac{u(x_{j+1},t) - 2u(x_j,t) + u(x_{j-1},t)}{\Delta x^2}.$$

for $j = 1, 2, \ldots, N$, where $u_j(t) = u(x_j, t)$. This can be restated as $u_t = cLu$, where L is the discrete Laplacian with periodic boundary conditions

$$L = \begin{pmatrix} -2 & 1 & 0 & 0 & \ldots & 1 \\ 1 & -2 & 1 & 0 & \ldots & 0 \\ 0 & 1 & -2 & 1 & \ldots & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ 1 & 0 & \ldots & 0 & 1 & -2 \end{pmatrix}. \tag{5}$$

The implicit Euler method is used to solve the equation in time, that is, $\frac{1}{\Delta t}(u^{n+1} - u^n) = \frac{c}{\Delta x^2} L u^{n+1}$, which simplifies to

$$A^n u^n = u^0 \tag{6}$$

where u^n is the vector $u(n\Delta t) = (u(x_0, n\Delta t), \ldots, u(x_N, n\Delta t))^T$, and A is the $N \times N$ matrix $I - c\frac{\Delta t}{\Delta x^2} L$.

For this simple case the cost function is taken to be simple a least-squares optimization

$$\min_{c>0} \|F(c) - d\|_2$$

where $F(c) = A(c)^{-n} u^0$ and d is the measured, potentially noisy data. A simple Newton's method optimization method was used. Five point numerical methods were used to compute both the first and second derivatives needed for the methods

$$\begin{aligned} F'(c) &\approx \frac{-F(c+2\Delta c) + 8F(c+\Delta c) - 8F(c-\Delta c) + F(c-2\Delta c)}{12\Delta c} \\ F''(c) &\approx \frac{-F(c+2\Delta c) + 16F(c+\Delta c) - 30F(c) + 16F(c-\Delta c) - F(c-2\Delta c)}{12\Delta c^2}. \end{aligned}$$

To begin the optimization procedure, the solution corresponding to c_{true} is calculated and an element of Gaussian noise is added. An example of the true computed solution and the noisy measurement is given in Figure 1.

3.1.2 Results

The analytic solution to the initial value problem can be given as a smooth periodic function, i.e., the function and all derivatives agree at the boundary points, as a summation

$$\sum_{m=0}^{\infty} a_m \cos(2\pi m x)$$

representing the initial data, where $\{a_m\}$ are Fourier coefficients. Then the solution is given by

$$\sum_{m=0}^{\infty} a_m e^{-4\pi^2 m^2 t} \cos(2\pi m x)$$

We first discuss the selection of an initial guess, which will be useful for any perturbation of the diffusion equation. A plot of the the cost function for various values of c (with $c_{true} = \frac{1}{2}$) is given in Figure 2. We see immediately that a choice of c too large will result in never finding the true solution (the function is concave for c too high, resulting in a parabolic approximation with a maximum, not a minimum). Then convergence can be achieved by choosing an initial guess c_0 to be small.

Plots showing the convergence of the solutions are given in Figure 3. The first thing that one notices (3a,3b) is that the convergence of c_k is only linear, as opposed to the quadratic rate that one generally expects with Newton's method. This is perhaps caused by using numerical approximations for the first and second derivatives rather than analytical formulas. In the case when the error standard deviation $\sigma = 0.1$ (3b,3d) we see an example of movement away from the true solution as a result of measurement noise.

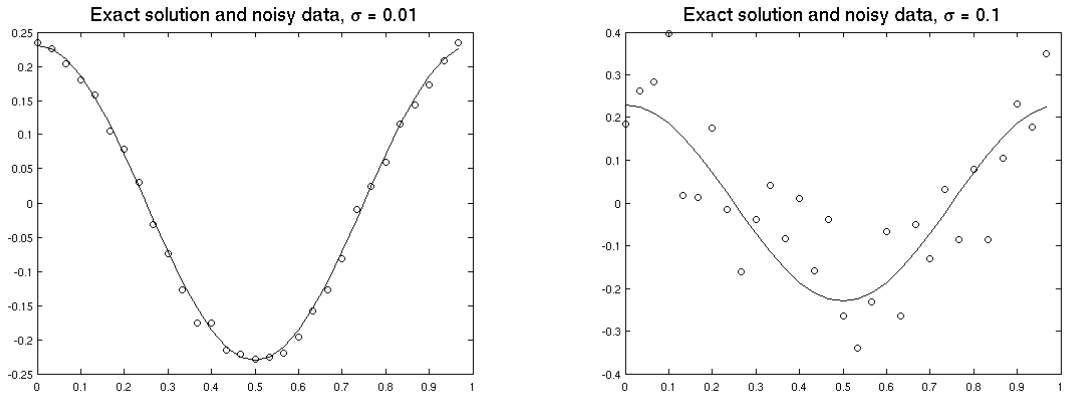

Figure 1: True solution to the simple initial value problem $u_t = cu_{xx}$ across a bar of unit length with boundary conditions $u(0,t) = u(1,t)$ and constant c. Gaussian noise is added to the true solution to generate the input data which the parameter fitting algorithm is to work with. The noise has zero mean in both cases, with standard deviation 0.01 (left) and 0.1 (right).

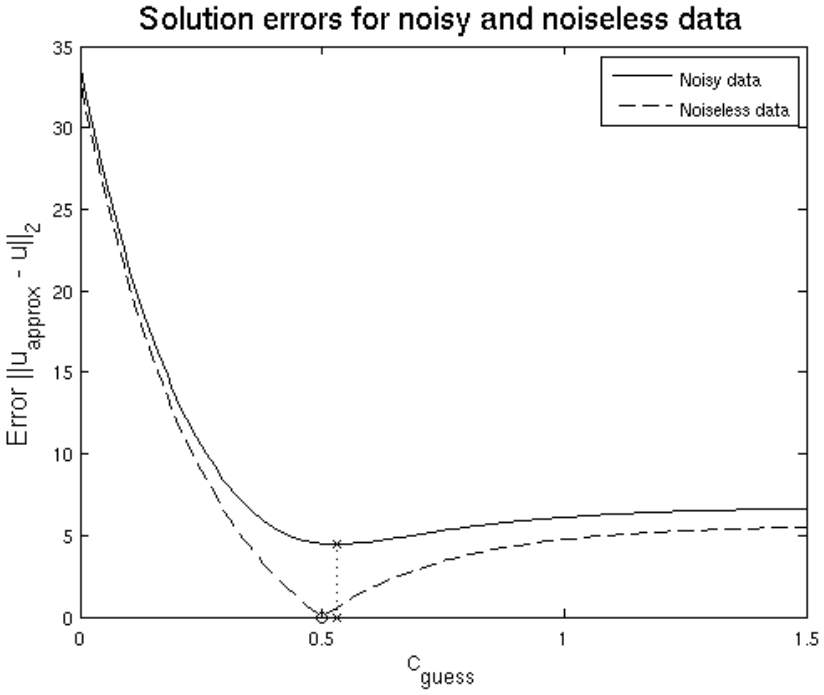

Figure 2: 2-norm errors in the solution to the initial value problem as a function of the initial guess for the diffusion constant, c_{guess}, with the true value $c_{true} = 0.5$. Notice that the global minimum generated by the noisy data may not be the same as for the noiseless data. Thus it may be impossible to reconstruct the diffusion constant exactly given noisy data. The curve has two areas of interest: a convex portion to the left of the minimizer and a concave portion to the right. When using a (quasi-)Newton method for optimization it is important that the function iterates stay in the convex portion, otherwise the algorithm will attempt to find a maxima rather than a minima.

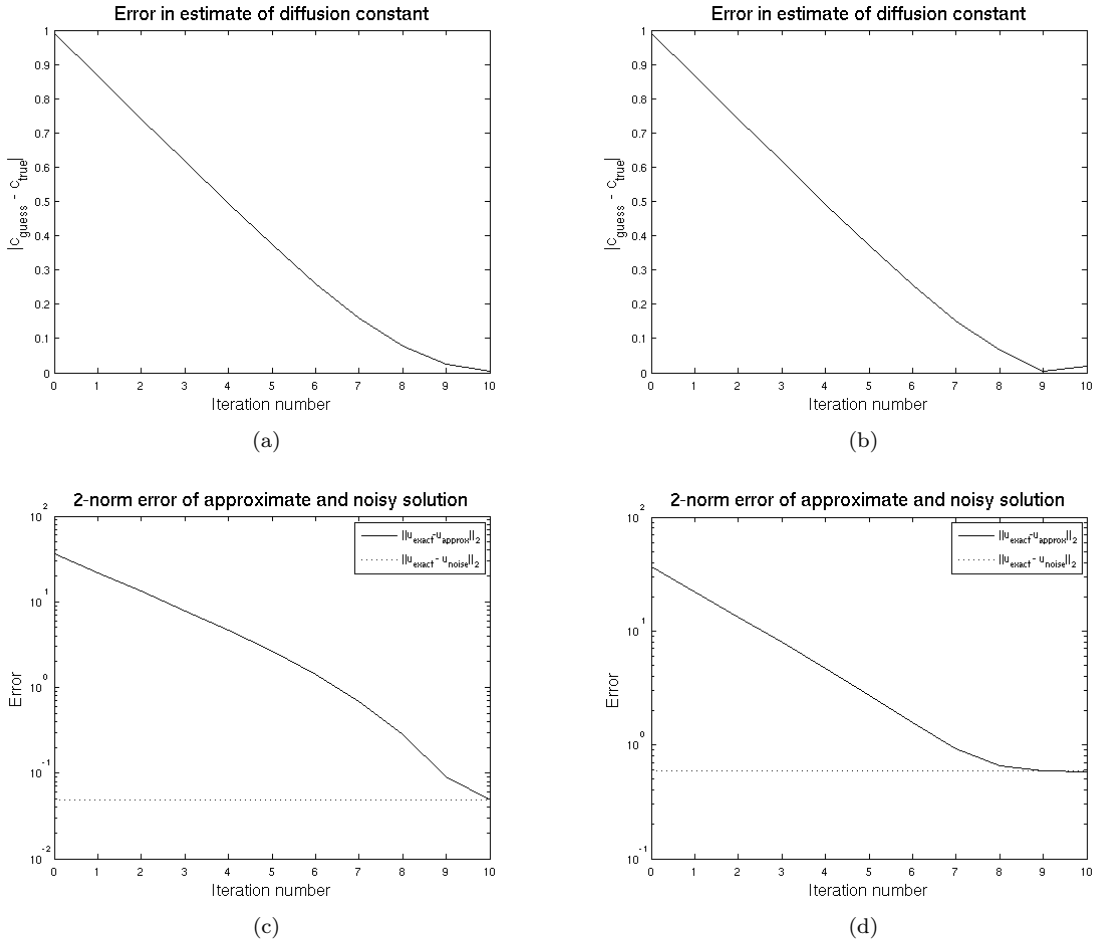

Figure 3: Convergence plots of two examples of the simple initial value problem $u_t = cu_{xx}$ with noisy data. The plots show the error in the estimated diffusion constant (top) and the 2-norm difference between the computed solution u and the noisy data (bottom). These solutions were taken for $\sigma = 0.01$ (left) and $\sigma = 0.1$ (right). We note that the convergence is only linear instead of quadratic that one generally expects with Newton's method. Perhaps this is the cost of dealing only with approximate first and second derivatives given by a 5-point stencil rather than analytic formulas.

3.2 A slightly more complicated diffusion equation

A somewhat more interesting example is the diffusion equation

$$u_t = c(x)u_{xx}$$

with periodic boundary conditions. To allow for computational simplicity, we assume that $c(x)$ is piecewise constant along intervals of roughly equal length. That is, if the parameter n_c describes the total number of piecewise constant sections, with the diffusion constants given by the n_c-dimensional vector $\mathbf{c}$, and the unit interval is broken into N nodes, then

$$c(x) = \begin{cases} c_1, & x \in \left[0, \frac{1}{n_c} - \frac{1}{2N}\right) \\ c_j, & x \in \left(\frac{j-1}{n_c} - \frac{1}{2N}, \frac{j}{n_c} - \frac{1}{2N}\right), j = 2, \ldots, n_c - 1 \\ c_{n_c}, & x \in \left(\frac{n_c - 1}{n_c} - \frac{1}{2N}, 1\right] \end{cases}$$

3.2.1 Implementation

The setup for solving the initial value problem is similar to the last example, with the obvious exception that the matrix L from (5) changes to take the non-constant c into account. If $n_c = N/2$, then

$$L = \begin{pmatrix} -2c_1 & c_1 & 0 & 0 & \ldots & c_1 \\ c_1 & -2c_1 & c_1 & 0 & \ldots & 0 \\ 0 & c_2 & -2c_2 & c_2 & \ldots & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ c_{N/2} & 0 & \ldots & 0 & c_{N/2} & -2c_{N/2} \end{pmatrix}. \tag{7}$$

The optimization algorithm itself does need to change. Similar to the one dimensional case, the multidimensional case presents us with two main areas in the objective function, defined by whether the function is concave or convex (See Figure 4 below for a two-dimensional example). Unfortunately, a Quasi-Newton approach by itself tends to land on the areas of concavity (i.e., areas where the Hessian approximation B_k is not at least positive semi-definite, it may even be negative-definite), resulting in movement away from the global minimizer.

To combat this problem I have opted to use a hybrid BFGS-Steepest Descent algorithm. The principle part of the algorithm is a standard BFGS method iteration. However, if the algorithm finds that the search direction p_k is not a descent direction a step of steepest descent is performed. The approximation of the inverse Hessian, here labeled $H_k = B_k^{-1}$ is updated with the standard BFGS during the steepest descent step.

To check that the search direction is a descent direction, recall that the Quasi-Newton methods set

$$p_k = -H_k \nabla f_k,$$

where $H_k = B_k^{-1}$. Then the computation only checks that

$$\nabla f_k^T p_k < 0.$$

Exit conditions of the algorithm are given in terms of the previous step size taken, ϵ_{step}, the function gradient at the current location, ϵ_{grad}, and the total number of iterations performed, I_{max}. The stop condition is

$$(\|c_k + 1 - c_k\|_\infty < \epsilon_{step} \text{ and } \|\nabla f_{k+1}\|_\infty < \epsilon_{grad}) \text{ or } k + 1 > I_{max}.$$

Values used are $\epsilon_{step} = 10^{-4}$, $\epsilon_{grad} = 10^{-3}$ and $I_{max} = 150$ or 300.

The line search for the Quasi-Newton step was implemented as a so-called Backtracking Line-Search. We begin with the step size $\alpha = 2$, and iterate as $\alpha \leftarrow \rho\alpha$ with $\rho = 0.9$ until an α is found such that the first Wolfe condition is satisfied.

The line search for the steepest descent step is executed slightly differently. The steepest descent portion of the algorithm is entered when one of the iterates lands on the concave portion of the objective function, so we would like the least squares estimate to place the next iterate on the convex portion to continue with the BFGS algorithm. Thus we want to slightly overshoot the line search. This is achieved by a bisection search technique. At each step, searching for $x_{k+1} = x_k - \alpha \nabla f_k$ we begin with a set of values for α, $\overline{\alpha}_j = (\alpha_j^1, \alpha_j^2, \alpha_j^3)$, and a corresponding set of function values $f_k^j = f(x_k - \alpha \nabla f_k)$, where j is the iteration counter for the line search. Let $J = \arg\min_j f_k^j$. Then the relation is

$$\overline{\alpha}_{j+1} = \begin{cases} \left(\alpha_j^1, \dfrac{\alpha_j^1 + \alpha_j^2}{2}, \alpha_j^2\right), & J = 1 \\ \left(\alpha_j^2, \dfrac{\alpha_j^2 + \alpha_j^3}{2}, \alpha_j^3\right), & J = 2, 3 \end{cases}$$

3.2.2 Results

We begin by presenting the case of $n_c = 2$ with noiseless data, as it will be instructive for examining other cases. A surface plot of the objective function is given in Figure 4. As in the simple case, the surface is split into distinct areas that are concave and convex. However, the extra dimension also allows for areas with a saddle-like shape. From this complexity arose the need to have a hybrid algorithm. Figure 4 also shows the path taken by the hybrid algorithm from the initial guess to the final resting place, and the type of iteration taken to get to that point.

Figure 5 gives the other interesting plots for this initial case. We can see the error of the approximate parameter system, measured either as $\|c_k(x) - c_{true}(x)\|_2$ (5a) or $\|u_k(x, t_{final}) - u_{true}(x, t_{final})\|_2$ (5c), decays as a slow exponential. Figure 5b shows the exact and estimated diffusion parameters in the system, and Figure 3.2.2 gives the true, measured and calculated final head distribution.

One final example shows the behavior of the algorithm with noisy data, shown in Figure 6. The level of noise sets an effective barrier to the optimization loop which takes considerable time to get around. The program still converges very fast to this noise level, but convergence becomes very slow once that level is reached. The hybrid nature of the algorithm seems to help it to avoid getting stuck at the level of noise. However, even with this advantage there is not a great deal more that can be done, and so convergence is very slow to a shallow local minimum, which approximates the true diffusion parameters.

4 Conclusion

We began with the problem of identifying a potentially non-constant diffusion parameter in a simple heat equation $u_t(x,t) = c(x)u_{xx}(x,t)$ on an interval of unit length with periodic boundary conditions $u(0,t) = u(1,t)$ and known initial condition $u(x,0) = f(x)$. The parameter fitting would use a (possibly noisy) measurement, $y(x)$ of the system at some time $t_{final} > 0$ and attempt to minimize the least-squares function $\|u_c(x, t_{final} - y(x)\|_2^2$, where u_c is the solution computed by solving the initial-boundary value problem with a parameter set c numerically. Because the objective function is concave in some area near the solution a straightforeward implementation of Newton's method, or the BFGS quasi-Newton method would fail. The optimization algorithm was modified to include a step of steepest descent method whenever a Newton step would produce a non-search direction.

We find that this method converges spectrally to the true solution, albeit with a fairly low constant. In the example given above, the convergence was approximately like $\|c^k - c_{true}\|_2^2 \leq K \exp(0.06k)\|c^0 - c_{true}\|_2^2$, where K is a positive constant, and c^k is the approximation to the parameter values after k iterations.

The inclusion of noise into the measurement creates an effective barrier to optimization, beyond which decreases in the objective function can be found only very slowly, if at all.

5 Acknowledgments

The author would like to thank Dr. Juan Restrepo for his guidance in the creation of this paper.

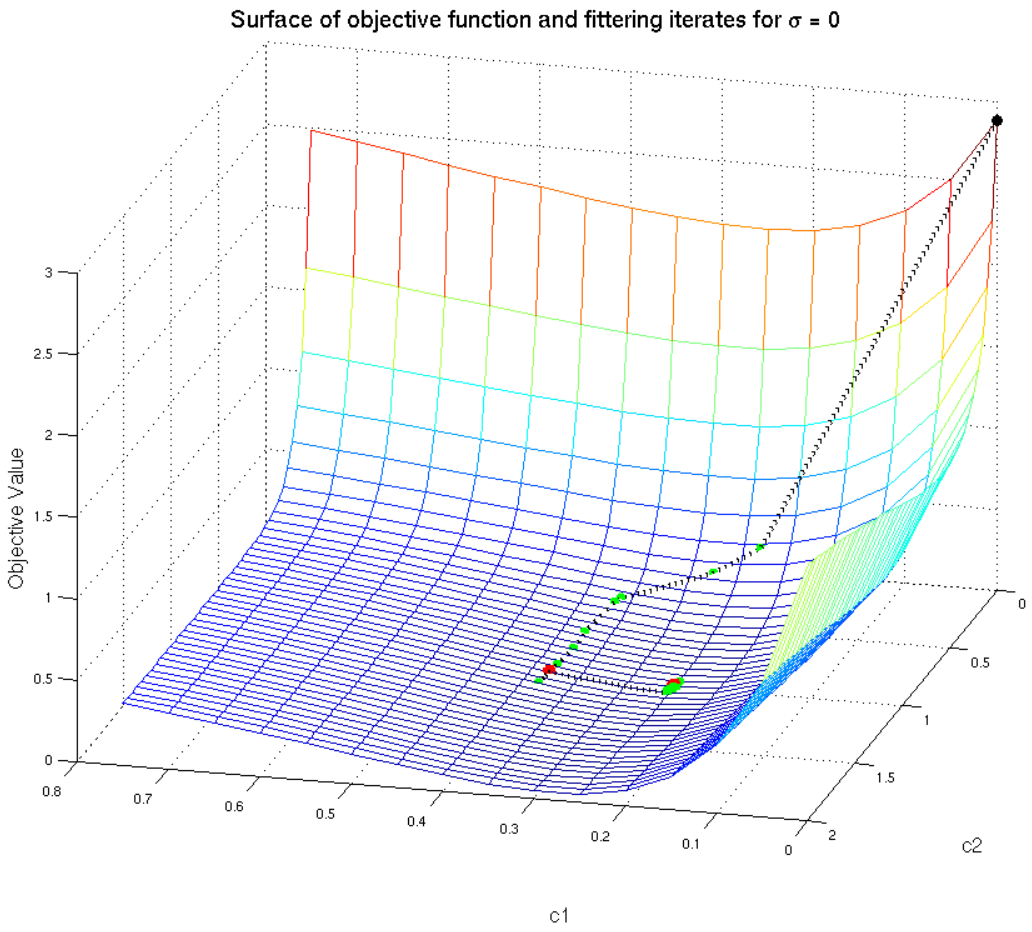

Figure 4: The surface of the objective function with noiseless data and $n_c = 2$, along with the path taken to reach the optimization point. The initial guess is shown as a black dot in the corner. After that, the dots are color-coded to represent the step taken to reach that point: green for a BFGS step and red for a steepest descent step. Steepest descent steps are taken either when the search direction for BFGS is a non-descent direction.

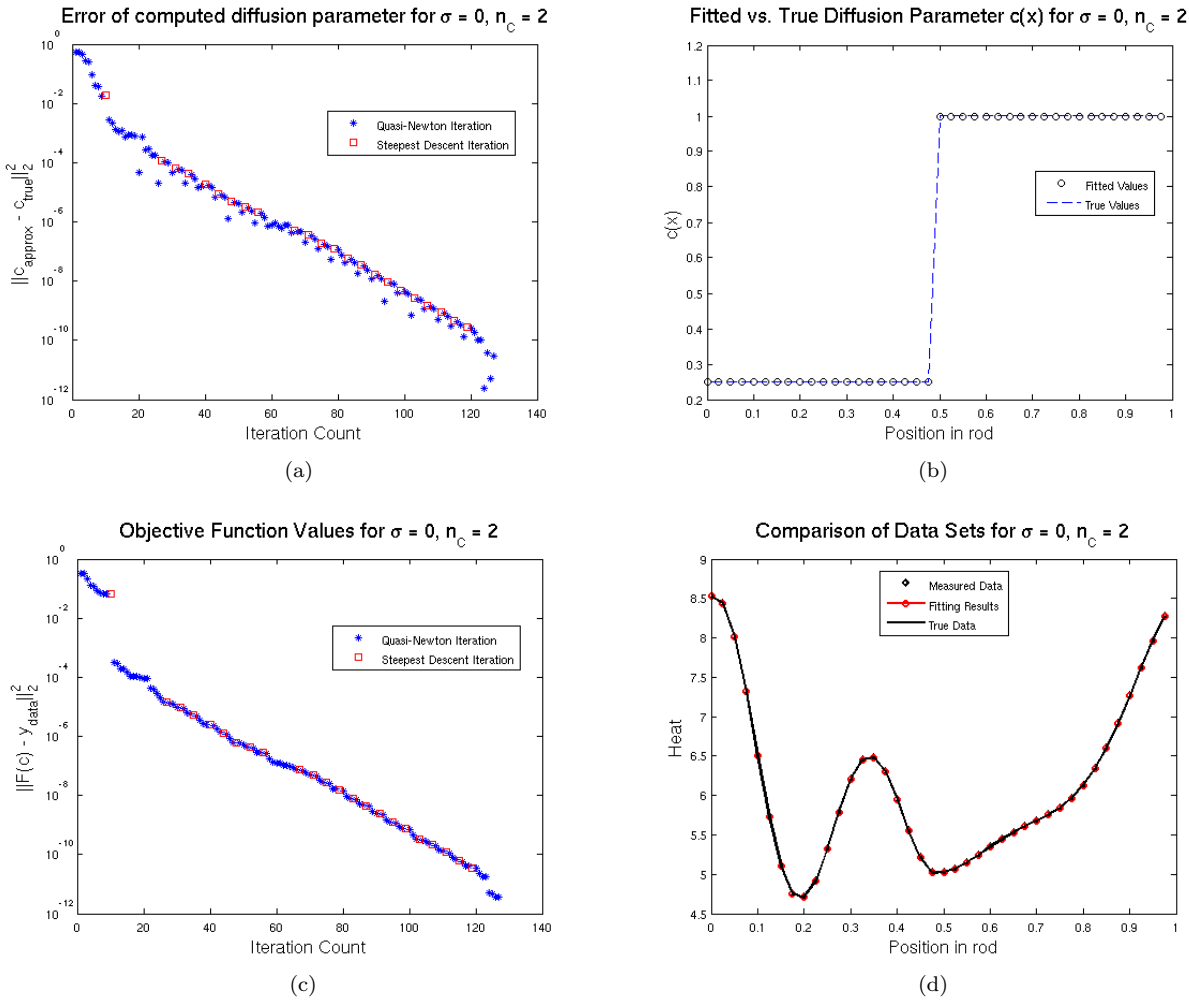

Figure 5: Important plots for the noiseless case of $n_c = 2$ of the diffusion equation $u_t = c(x)u_{xx}$. Iteration dependent information (left) shows how the 2-norm of the error in the diffusion parameter $c(x)$ (5a) and the quadratic objective function value (5c) change over time. Final state information (right) shows the true and calculated diffusion parameters (5b) and the true, measured and calculated values of u. The iteration-dependent plots are color coded to represent what method was used to reach that point. Both iteration-dependent plots demonstrate a slow exponential decay in the error of the estimate.

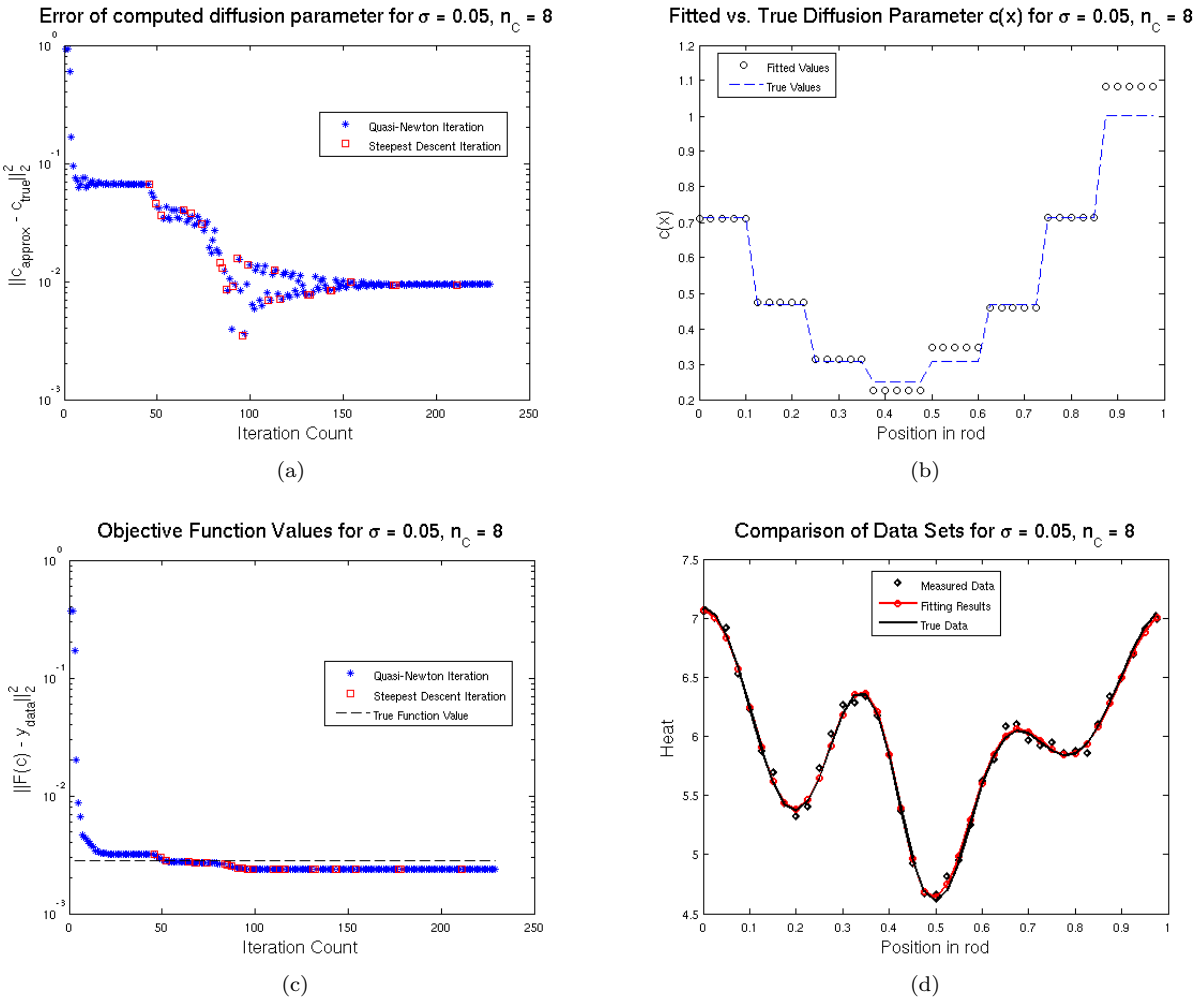

Figure 6: Plots for the case of $n_c = 8$, with Gaussian noise of standard deviation $\sigma = 0.05$. The first thing that we notice is that there is no longer a nice exponential convergence to the solution. The optimization begins by converging quickly, but is quickly halted once it reaches the level of the noise, shown by the dashed line in 6c. Convergence does not begin again until a steepest descent step is completed. The program begins converging again, though much more slowly and very jittery, until it settles down on some solution which is closest to the noisy data, and fairly close to the true parameter values.

References

[1] Curtis R. Vogel. *Computational Method for Inverse Problem.* SIAM, Philadelphia, PA, 2002.

[2] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization.* Springer, New York, NY, second edition, 2006.

[3] Dimitri P. Bertsekas. *Nonlinear Programming.* Athena Scientific, Belmont, Mass., second edition, 1999.