Comparison of Truncated SVD, Tikhonov regularization, and Preconditioners, and SOR for solving 1D Heat equation

Aaron Wubshet

1. Introduction

1.1. Background

Everyday life is filled with interesting and unexpected use cases for numerical methods to solve inverse problems efficiently. Inverse problems arise when two conditions are met:

(1) there is a high likelihood for noise or error to impact data measurements

(2) the dynamics of the system governing the solution are sensitive to that noise.

The sensitivity of system dynamics (represented as a matrix vector equation) can be measured by what is known as the condition number of a matrix. Given $Ax = b$, the condition number of A is a metric for how susceptible a system is to noise. It is defined as $\kappa(A) = ||A||||A^{-1}||$ and thus is only valid for square and invertible matrices. Some examples of systems susceptible to high conditions numbers and thus noise are image de-blurring and sub-fields of tomography, which is a general imaging technique used in fields such as seismology and radiology among others to create 2D image slices of the interior of a 3D object. Methods for dealing with the inverse problem vary based on the use case. For x-rays, the Radon transformation, $Rf = \int_L f(x)|dx|$ is used to transform the problem into "Radon space", solved, and then transformed back into normal space which alleviates the inverse issue. This is similar to taking the Fourier Transform to turn PDEs into ODEs, solving, and then inverting the transform back out of Fourier space

Another example of an inverse problem is determining the initial heat distribution of a system based on an observation of the heat distribution at time t. The first step is defining the heat equation and solving it. The heat

equation can be formulated as $\dot{u} = \alpha\nabla^2 u$. In words the equation can be described as the change in heat (temperature) with respect to time is equal to the difference of the value of the temperature (u) at a point and the average temperature of the nearby points multiplied by a constant representing the thermal conductivity, specific heat, and density of the material through which the heat distribution is changing. We will however, be considering the one dimensional case (thinking about a long rod that conducts heat) which can be expressed as:

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x}, \forall x \in \mathbb{R}, t > 0 \tag{1}$$

1.2. Fundamental Solution

While separation of variables can be used to solve the PDE, we will explore numerical techniques that expose (and deal with) the inverse nature of extracting the initial heat distribution from the solution. However, to do that, we must first arrive at the fundamental solution, or heat kernel. There are two important properties of the heat equation that we wish to exploit:

(1) the total amount of heat is conserved: $\frac{d}{dt}(\int u dx) = 0$

(2) if $u(x,t)$ solves the heat equation then a translated $u_1(x,t)$ and rescaled $u_2(x,t)$ version of of $u(x,t)$ also solves the heat equation. Specifically, $u_1 = u(x - x_0, t - t_0)$ and $u_2(x,t) = Au(\lambda x, \lambda^2 t)$

So, we can do a change of variables and choose solutions of the form $u(x,t) = \frac{1}{\sqrt{t}}F(\eta)$ where $\eta = \frac{x}{\sqrt{t}}$. Substituting into the heat equation (1), we are left with

$$0 = 2F'' + \eta F' + F \tag{2}$$

Now by normalizing the amount of heat in the system to be 1 and integrating $u(x,t)$ across the domain we arrive at the fundamental solution:

$$G(x,t) = \frac{1}{\sqrt{4\pi t}} \exp(-\frac{x^2}{4t}) \tag{3}$$

Now taking into account the boundary condition that as $x \to \infty, u \to 0$ and $u(x,0) = u_0(x)$ as well as our translation property we are left with the heat equation solution involving the heat kernel of:

$$u(x,t) = \int_{-\infty}^{\infty} G(x - x', t) u_0(x') dx' \tag{4}$$

2. Inverse Problem Solution Methods

2.1. Problem Setup

Given the solution from (4), we now wish to determine $u_0(x)$. First we need a way to discretely approximate the integral. Thus, we define a set of discrete points $x_j (j = 1..n)$ such that our integral can be written as the sum,

$$u(x_i, t) = \sum_{j=1}^{1} w_j G(x_i - x_j, t) u_0(x_j) \tag{5}$$

where the w_j are Gaussian quadrature weights for approximating integrals as sums. In this form, we can define column vectors $u(t)$ and u_0 so that $[u(t)]_i = u(x_i, t)$ and $[u_0]_i = u_0$. Leaving an n x n matrix $[A(t)]_{i,j} = w_j G(x_i - x_j, t)$ to be the factor relating them in our final scheme:

$$\mathbf{u}(t) = \mathbf{A}(t)\mathbf{u}_0 \tag{6}$$

Thus we aim to ascertain u_0 given some observation $u(t)$ and $A(t)$ (See Figure 1). The aforementioned condition number of $A(t)$ gives us an indication of how noise in our $u(t)$ measurement will impact the robustness of the numerical method we choose to determine u_0.

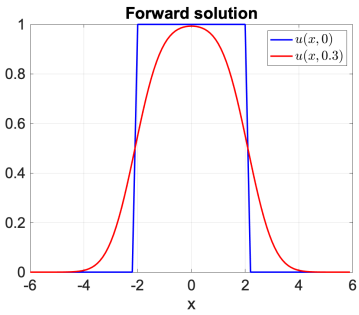

Figure 1: Aim: Recover blue from red

2.2. Singular Value Decomposition

The first method we will consider to achieve our aim is Truncated SVD. The following methods rely on the singular value decomposition, or SVD, which is a matrix decomposition of the form $M = U\Sigma V^T$ in which M is a m x n matrix with rank r,U and V are matrices with orthonormal columns (sizes m x r and n x r, respectively), and Σ which is a diagonal matrix containing the unitary values, σ_i inversely related to the eigenvalues of M. The columns of U are the eigenvectors of $A^T A$ while the columns of V are the eigenvectors of AA^T.

2.2.1. Truncated SVD

However, in order to use the SVD to solve inverse problems we can consider $Ax = b$ and $A = U\Sigma V^T$ with A invertible and n x n. We can write

$$x = \sum_{i=1}^{n} v_i \frac{(u_i \cdot b_i)}{\sigma_i} \tag{7}$$

With large $\kappa(A)$ our system will tend to not satisfy the Picard condition which says that $|u_i \cdot b_i|$ should go to 0 faster than σ_i goes to 0 as $i \to n$. To deal with this we truncate the SVD solution at k as in Figure 2. Picking a k to truncate at is a balance between a solution that is too noisy and a solution that is overly smooth. Thus k is usually chosen by hand, however, if we know the true solution, an error term can be minimized over possible k values. The truncated solution has the form:

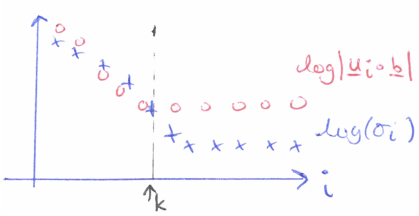

Figure 2: Approximate truncation at k

$$x_k = \sum_{i=1}^{k} v_i \frac{(u_i \cdot b_i)}{\sigma_i} \tag{8}$$

2.2.2. Tikhonov regularization

Another method of smoothing out noisy solutions involves introducing a regularization term $\lambda^2||x_\lambda||^2$ with $\lambda > 0$ such that our solution has the form:

$$x_\lambda = \arg\min_x [||Ax - b|| - \lambda^2||x_\lambda||^2] \tag{9}$$

Going back to the SVD of A and a bit of matrix algebra, we can reformulate our solution as:

$$x_\lambda = \sum_{i=1}^{n} v_i \frac{\sigma^2}{\sigma^2 + \lambda^2} \frac{(u_i \cdot b_i)}{\sigma_i} \tag{10}$$

where the middle term $\frac{\sigma^2}{\sigma^2+\lambda^2}$ determines the degree to which the solution is smoothed. The larger λ is relative to σ, the more smooth the solution will appear. Similar to truncated SVD, a residual or error term can be defined to determine suitable values for λ which is analogous to $\frac{1}{k}$.

2.3. Preconditioners

Preconditioning a matrix A provides a different way of handling ill conditioned (susceptible to noise) systems. The general idea being that if we can pick a matrix P to multiply both sides of our $Ax = b$ system by, and PA has a smaller condition number than A, we can subsequently solve our system, usually through iterative methods, namely pure iteration and conjugate gradient. Our new system that we aim to solve when using preconditioners is:

$$PAx = Pb \tag{11}$$

where PA is well conditioned.

Thus, the main issue becomes picking an appropriate preconditioner for each situation. Some common choices for P include a diagonal matrix of the eigenvalues of A, utilizing an incomplete Cholesky factorization and setting $P = L_{inc}$ or $P = L_{inc}U_{inc}$, or even a diagonal matrix with the entries set to $\sqrt{\sqrt{\lambda_{max} * \lambda_{min}}}$. This last method is based on a Stanford paper from Javadi and Takapoui who explore diagonal scaling as preconditioners. Each of the mentioned preconditioners were used in conjunction with two iterative methods, pure iteration and conjugate gradient.

2.3.1. Pure Iteration

The first iterative method attempted was the standard pure iteration. Starting again with $Ax = b$ and writing the expression as $0 = -Ax + b$ and then adding Px to both sides leaves us with $Px = (P - A)x + b$ which we can formulate as an iterative method with some initial guess x_0

$$Px_{k+1} = (P - A)x_k + b \tag{12}$$

The convergence of this method for a given error tolerance depends on how similar the preconditioner is to A. We can write the error by considering the true solution x_r and subtracting (12) from $Px_r = (P - A)x_r + b$ leaving us with $Pe_{k+1} = (P - A)e_k$ which can be written as $e_{k+1} = (I - P^{-1}A)e_k$ and setting $M = (I - P^{-1}A)$ and considering an expression for the error of the k^{th} iteration we are left with

$$e_k = M^k e_0 \tag{13}$$

where e_0 is the error of the initial guess. This allows us to precisely determine when convergence will occur: when the eigenvalues of M are all less than or equal to 1 in magnitude.

2.3.2. Conjugate Gradient (CG)

This method is more sophisticated and complicated than pure iteration and in turn tends to provide better convergence. Implementation of this method was done through Matlab's pcg function with allows the user to enter a preconditioner and automatically runs conjugate gradient. A rough sketch of the method begins with the assumption that A is symmetric and positive definite. The method then iterates on possible solutions according to $Az = b - Ax_0$ subject to the minimization of

$$f(x) = \frac{1}{2}x^T Ax - x^T b \tag{14}$$

2.4. Successive Over Relaxation (SOR)

The final method tested was Successive over relaxation, or SOR. This method involves a extrapolation that is varied to achieve faster convergence. Formulated by initially taking our A matrix and factoring it into a diagonal component, strictly upper triangular component, and strictly lower triangular component, D, U, and L, respectively. The system $Ax = b$ can be multiplied by our extrapolation factor ω and then rearranged since $A = L + D + U$ as $(D + \omega L)^{-1}x = \omega b - [\omega U + (\omega - 1)D]x$. From there an iterative method can be described as:

$$x_{k+1} = (D + \omega L)^{-1}(\omega b - [\omega U + (\omega - 1)D]x_k) = L_\omega x_k + c \tag{15}$$

The convergence of this method depends on the choice of ω and plays a similar role to the preconditioners of the above methods.

3. Solution Method Comparison Analysis

3.1. Truncated SVD

Implementing the truncated in SVD in Matlab begins by identifying the appropriate truncation point using the Picard condition and plotting the singular values. Notice the different approximate truncation value for the data with and without noise in Figure 3. Applying the different approximate truncation k values for a first pass and using the $L2$ norm as a minimizer to determine the exact k value results in Figure 4.

When there is no noise in our observation, we see a Gibb's Phenomenon like effect as the method attempts to resolves the large derivative corners.

However, when noise is introduced to the situation the phenomenon is reduced, but our solution grows less accurate. In the case with no noise, the optimal k value turns out to be 46, but in the case with noise, the k value that minimizes the $L2$ norm of the error drops all the way to 17. This follows from our intuition that a k that is small (i.e. truncating away a lot of the points) will result in a smoother solution. Thus when noise is added to our observation a lower k is required to minimize the error. The minimum achievable error also increases as we move from a noiseless situation to noisy situation. In this case, the error nearly doubles from .4 to over 1. Looking at Figure 5, we can see the error as a function of the truncation point, k, also has a strong exponential relationship near the optimal value that levels off as you increase k past it.

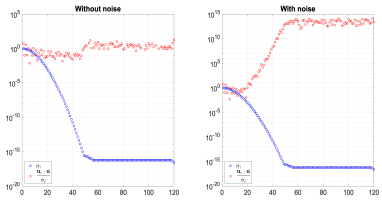

Figure 3: Approximate truncation at k

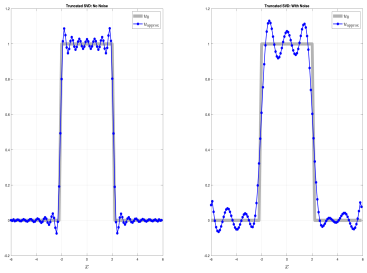

Figure 4: Noise and No Noise Truncated SVD Solution

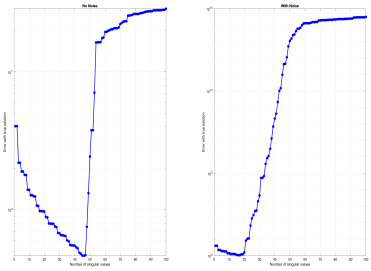

Figure 5: Error as a function of k

3.2. Tikhonov Regularization

Tikhonov, similar to truncated SVD, has a parameter that we vary to determine the optimal approximation. Unlike truncated SVD however, the parameter, λ is continuous, not discrete. It also has the opposite impact on the solution as k (increasing k is equivalent to decreasing λ). The solution produced from Tikhonov (Figure 6) is extremely similar to that of truncated SVD in both the noise and no noise situations. However, we can see that in the Tikhonov even when using the optimal λ the truncated SVD achieves a lower error when dealing with noisy observation by approximately 0.3, but Tikhonov has the edge in the no noise scenario by .002. Ultimately, we can see that the methods have extremely com-

parable results with λ values of .01 and 10^{-14} for noise and no noise, respectively. In Figure 7, an interesting note with Tikhonov can be seen as the fact that as you increase λ, the error drops pretty continuously when there is noise, but steadily climbs with λ without noise.

3.3. Preconditioner and Iterative Methods

When implementing the preconditioners with iterative methods, pure iteration and conjugate gradient, I restricted myself to 5 preconditioners to compare, ordered with increasing complexity (and hopefully performance). The most naive attempt was to simply extract the diagonal elements of A as a preconditioner. The next was to use a diagonal matrix of the eigenvalues of A. The next was to populate a diagonal matrix with the 4^{th} root of the product of the max and min eigenvalue. The final two precoditioners relied on the incomplete Cholesky factorization of A producing L and LU as our final two preconditioners. The final method, SOR, does not use preconditioners, but is iterative and for standard comparison, I used 10, 100, and 1000 iterations of all three iterative methods. I will compare the 1000 iteration scenario and include the images of 10 and 100 in the back.

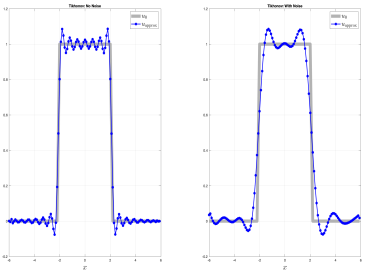

Figure 6: Tikhonov Solution with and without noise

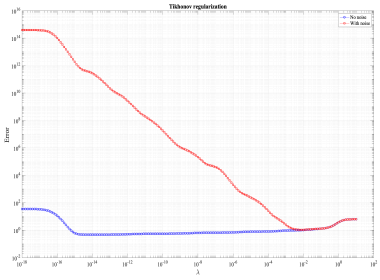

Figure 7: Tikhonov solution error as a function of λ

3.3.1. Pure Iteration

We first recall that in pure iteration each iteration is subject to the step defined by $Px_{k+1} = (P - A)x_k + b$. Implementing this with each of the preconditioners results in the plots shown below in Figure 8. The first thing we notice in the pure iteration approach is that when there is no noise, no solution is produced for the first three preconditioners, similarly with the noisey situation but only for the first preconditioner. While it may seem like no solution is produced, the issue lies in the fact that the first couple

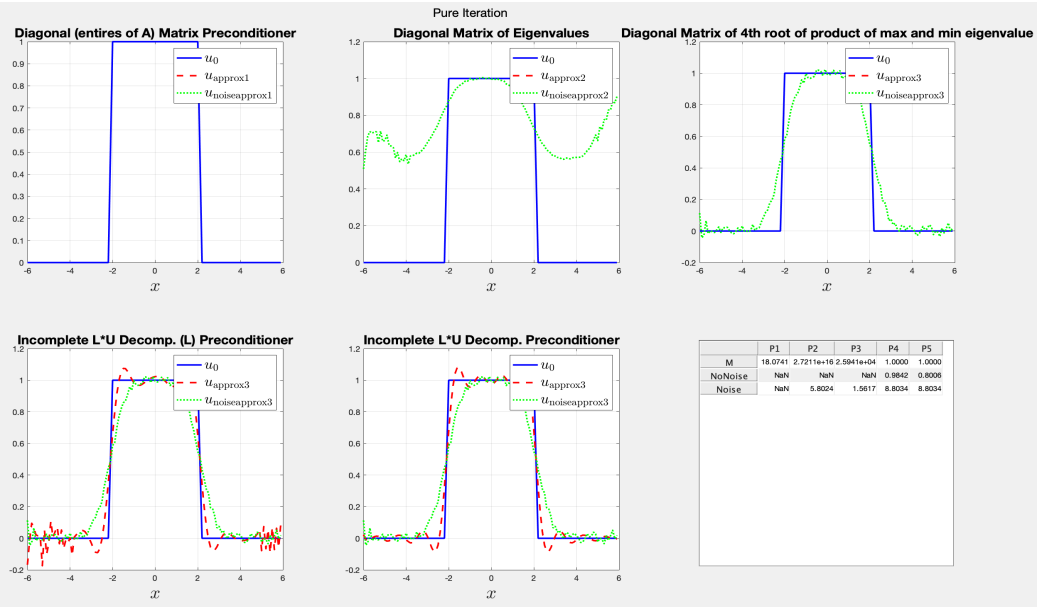

Figure 8: 1000 Iterations of Pure Iteration

preconditioners result in solutions that grow unboundedly. We can see in situations with fewer iterations we actually do get solutions, albeit very high in error. One thing we notice is that the approximations when noise is an issue tend to result in smoother solutions, due to the iterative nature of the methods. We can also recall that the matrix M we defined as $e_k = M^k e_0$ has maximum eigenvalues listed in the table part of Figure 8.

3.3.2. Conjugate Gradient

Matlab has a built in CG method with the option to include a precoditioner of our choice along with a minimum allowable error tolerance. The CG method is more robust and thus does not suffer from the same issue as pure iterations with unbounded solutions that cannot be plotted. However, we can note, that the diagonal matrix of the eigenvalues of A produces a bounded solution that oscillates rapidly due to what seems to be a scaling issue. Using the 4^{th} root of the product of the max and min eigenvalue of A however, provides a solution that actually minimizes the error. Looking at Figure 9, the preconditioners based on the incomplete Cholesky factorization have comparable but slightly worse performance with conjugate gradient and tend to have a lot of ringing when noise is added to the observation. It is

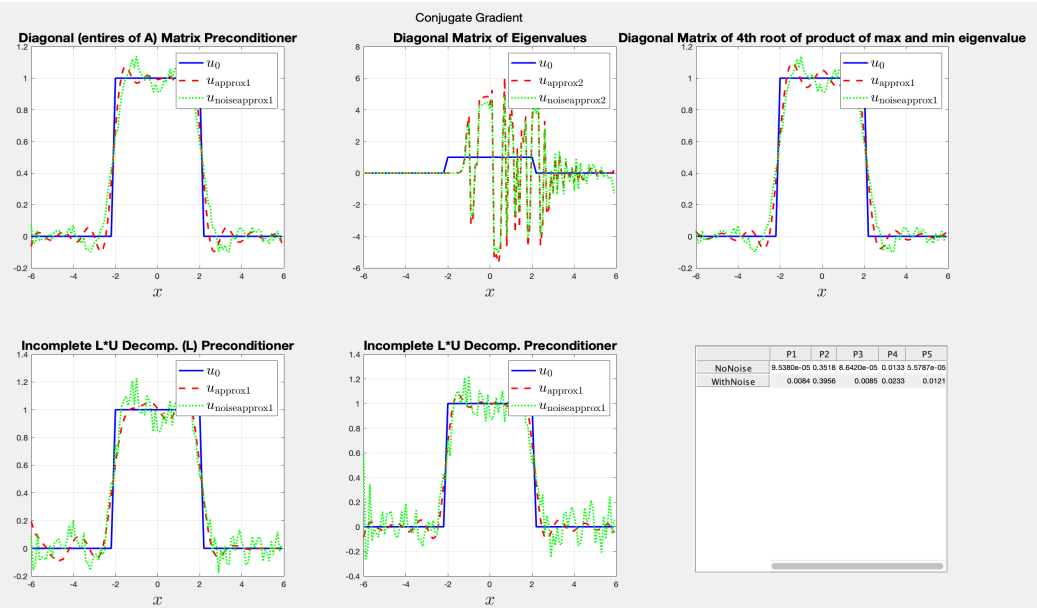

Figure 9: 1000 Iterations of Conjugate Gradient

worth noting that the performance is almost identical as the number of iterations is decreased to 100 and even 10. This would seem to indicate that conjugate gradient has very fast convergence for a given error tolerance.

3.4. Successive Over Relaxation

With successive over relaxation, we recall that we have a continuous parameter that we call the relaxation parameter and denote as ω. Even after 1000 iterations, however, as we can see with plot in Figure 10, the solution produced is offset and overly smooth compared to the other iterative and preconditioner based methods. While the ω can serve a similar role to the preconditioner, it is better thought of as a relaxation on how exactly we want our solution to match the true solution, to an extent.

Looking at the error terms, for both noisy and no noise, we can see that both are orders of magnitude higher than with the preconditioner methods. Similar to CG, however, we get similar performance with 10 and 100 iterations. The relaxation parameter ω has a quadratic relationship with the error, reaching a minimum around 1.4 for both the noisy and no noise scenarios. Increasing ω larger than 2 resulted in exponentially increasing error, corresponding to too much of a "relaxation" of an acceptable solution.

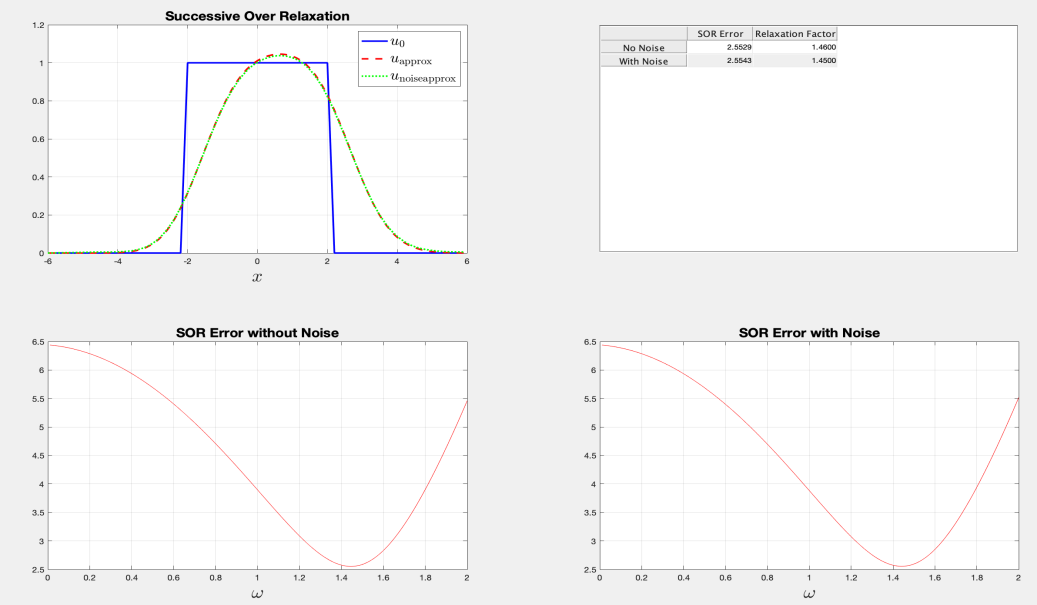

Figure 10: 1000 Iterations of SOR

4. Conclusion

We set out to solve an issue that comes up when solving a wide variety of problems from image de-blurring to x-ray imaging to understanding heat distribution. The canonical problem of determining the initial distribution of heat on a long one dimensional object was analyzed by first understanding the PDE governing the evolution of the heat and defining the fundamental solution. There after, our discrete scheme left us with a matrix vector equation of the form $Ax = b$ which we then set out to solve for x given a observation (possibly noisy) of b. This inverse problem arose as a result of the ill-conditioned nature of our evolution matrix A, in our case strongly tied to the quadrature weights and the fundamental solution we derived.

In order to tackle this ill-condition evolution matrix, we used a variety of methods including truncated SVD, Tikhonov regularization, pure iteration and conjugate gradient with preconditioners, and successive over relaxation. Each method has it's own merits in general, but for this specific problem a few key benefits became clear for conjugate gradient with preconditioners. The error for both the noise and no noise scenario was nearly the lowest of all the methods even when the number of iterations was decreased significantly.

The use of the 4^{th} root of the product of the maximum and minimum eigenvalue of A eliminated the need of doing the incomplete Cholesky factorization to obtain L and U. The success of using this approach is outlined in a paper by Takapoui and Javadi from Stanford and is unique to CG in it's sucess. As we can see from the pure iteration method results, this preconditioner causes the solution to grow unbounded if not paired with conjugate gradient.

Ultimately, the CG method seemed to provide the best results, however, applying SOR to any iterative method is a possibility to quicken convergence time, which is the main drawback for CG (higher computation time). In the future, I would be curious to attempt to incorprate SOR into a CG approach to solve an inverse problem similar to the heat equation explored here.

Bibliography

http://www.damtp.cam.ac.uk/user/dbs26/1BMethods/Heat.pdf
https://www.whitman.edu/Documents/Academics/Mathematics/mckenzie.pdf
https://en.wikipedia.org/wiki/Heat_kernel
https://en.wikipedia.org/wiki/Heat_equation
https://en.wikipedia.org/wiki/Gaussian_quadrature
https://en.wikipedia.org/wiki/Radon_transform
https://en.wikipedia.org/wiki/Inverse_problem
https://web.stanford.edu/~takapoui/preconditioning.pdf
https://en.wikipedia.org/wiki/Incomplete_Cholesky_factorization
https://en.wikipedia.org/wiki/Preconditioner#Other_preconditioners
https://www.youtube.com/watch?v=LtNVodIs1dI
https://en.wikipedia.org/wiki/Conjugate_gradient_method
http://mathworld.wolfram.com/SuccessiveOverrelaxationMethod.html

Appendix A. Images

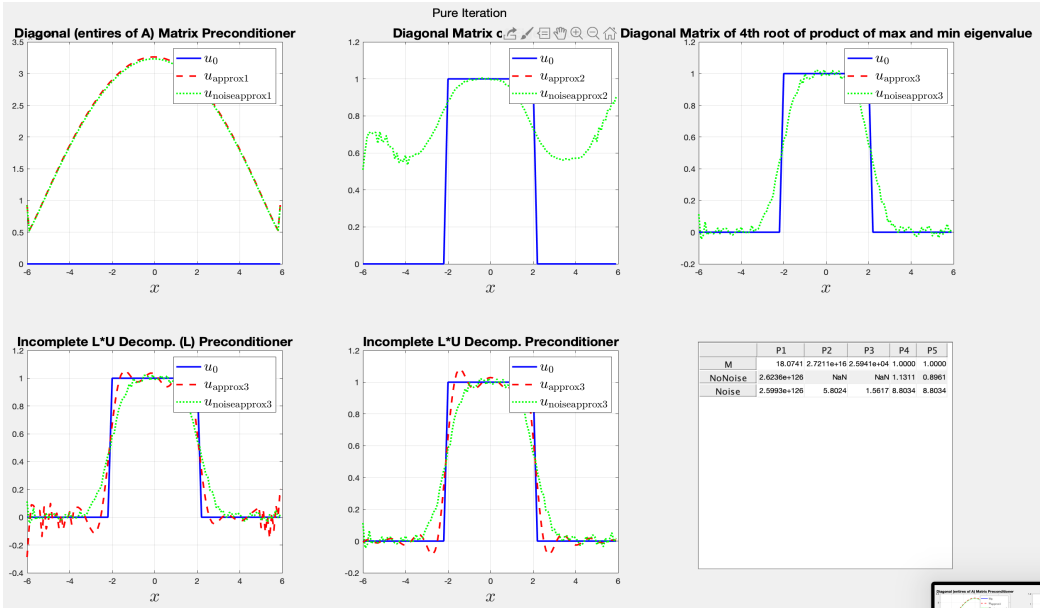

Figure A.11: 100 Iterations of Pure Iteration

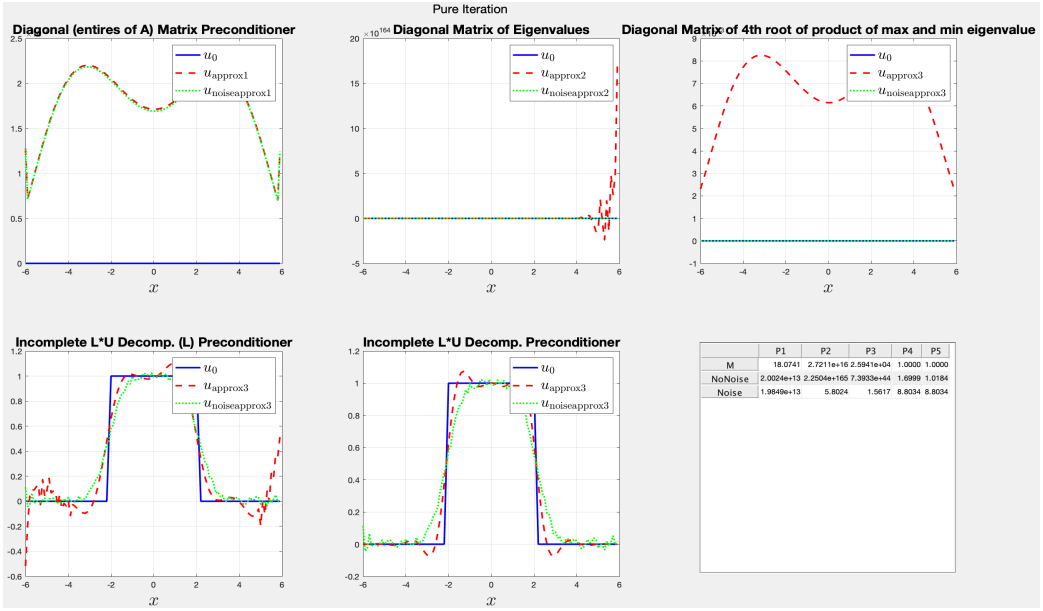

Figure A.12: 10 Iterations of Pure Iteration

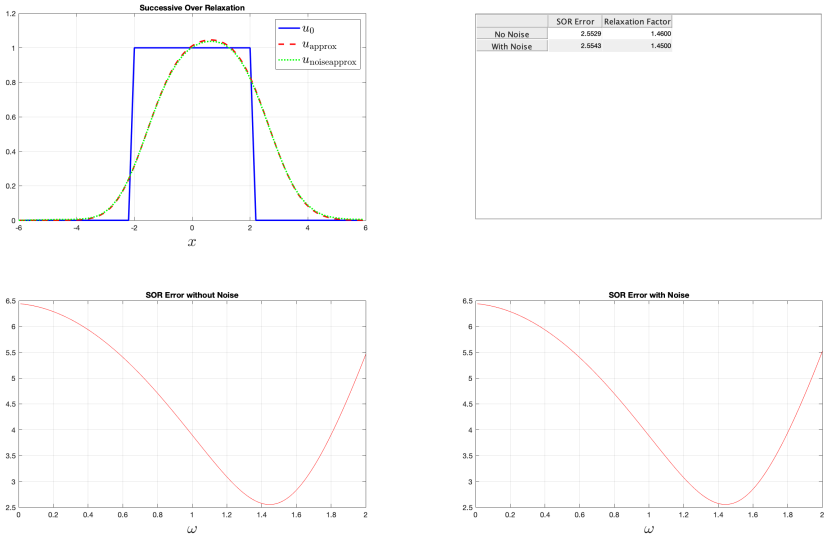

Figure A.13: 100 Iterations of SOR

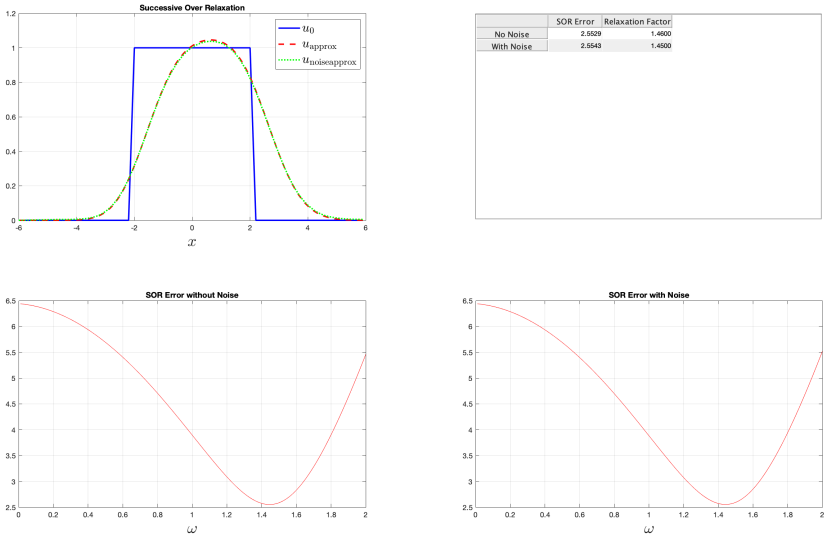

Figure A.14: 10 Iterations of SOR

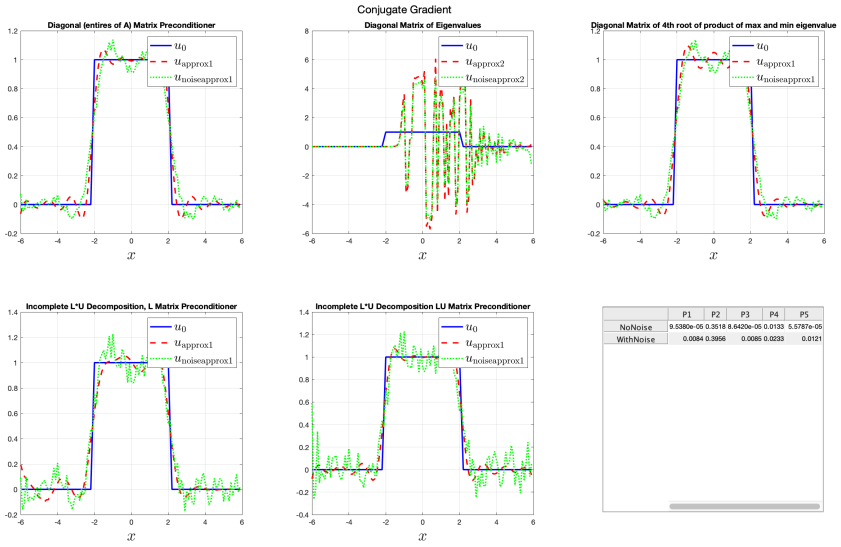

Figure A.15: 100 Iterations of CG

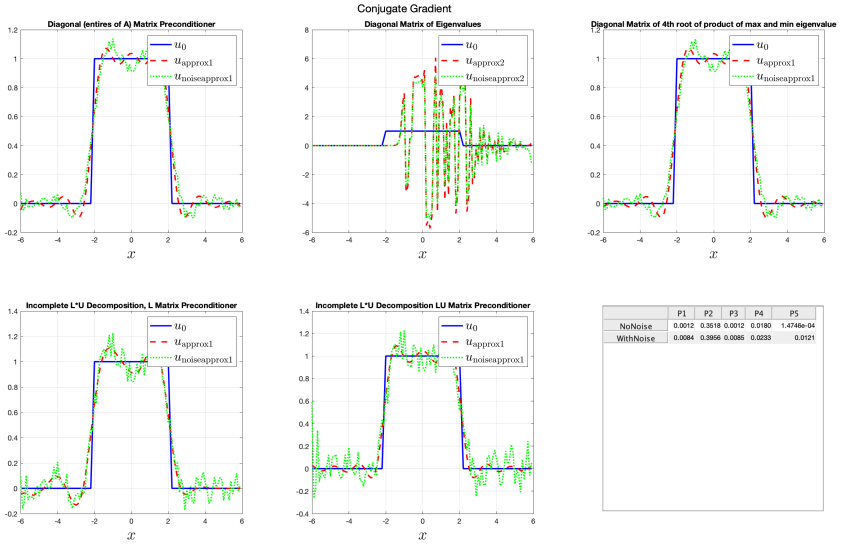

Figure A.16: 10 Iterations of CG