A Variational Approach to Signal Denoising

Name: Edward Rusu
RIN: 660878003

18 February 2016

Abstract

We derive the Tikhonov model for signal denoising using Lagrange Multipliers. We minimize the resulting functional in two ways. First, we use gradient descent to design a sequence of energy-decreasing functions that leads to the correct minimizer. Second, we solve the resulting elliptic partial differential equation using Gauss-Seidel and Conjugate Gradient schemes on matrix equations derived from Finite Differences. We show that the Tikhonov model denoises the signal but with the undesired side-effect of blurring the discontinuities. Finally, we suggest a better model that retains the discontinuities while denoising the signal.

1 Problems

Denoising signals is a desirable process in many areas of science and industry. For example, a camera manufacturer will want an algorithm that successfully denoises images without harming the actual content. In this report, we will take a mathematical approach to solve this problem.

Consider a signal that is a function from some domain Ω to real numbers $\mathbb{R}$.

$$u : \Omega \to \mathbb{R} \tag{1}$$

and let $S = \{u : \Omega \to \mathbb{R}\}$. We are not given the original function u; instead, we are provided with a function u_0 that has been corrupted by some noising operator Γ, such as the signal and image shown in Figures (1) and (2). We want to construct some denoising operator Γ^{-1} so that given u_0,

$$u = \Gamma^{-1}(u_0) \tag{2}$$

To better understand this problem, we make the following assumptions:

- We assume the corruption is additive

$$u_0 = u + n. \tag{3}$$

  where n is noise given from a Gaussian distribution with mean $\mu = 0$ and standard deviation σ.

- We assume the original signal is smooth

$$\int_\Omega |\nabla u|^2 \, dx \text{ is bounded.} \tag{4}$$

- We assume the signal and the noise have equal means

$$\int_\Omega u \, dx = \int_\Omega u_0 \, dx \tag{5}$$

- We define the standard deviation σ to be

$$\frac{1}{2}\int_\Omega (u_0 - u)^2 \, dx = \sigma^2 \tag{6}$$

- We impose no flux on the boundaries

$$\frac{\partial u}{\partial \vec{n}} = 0. \tag{7}$$

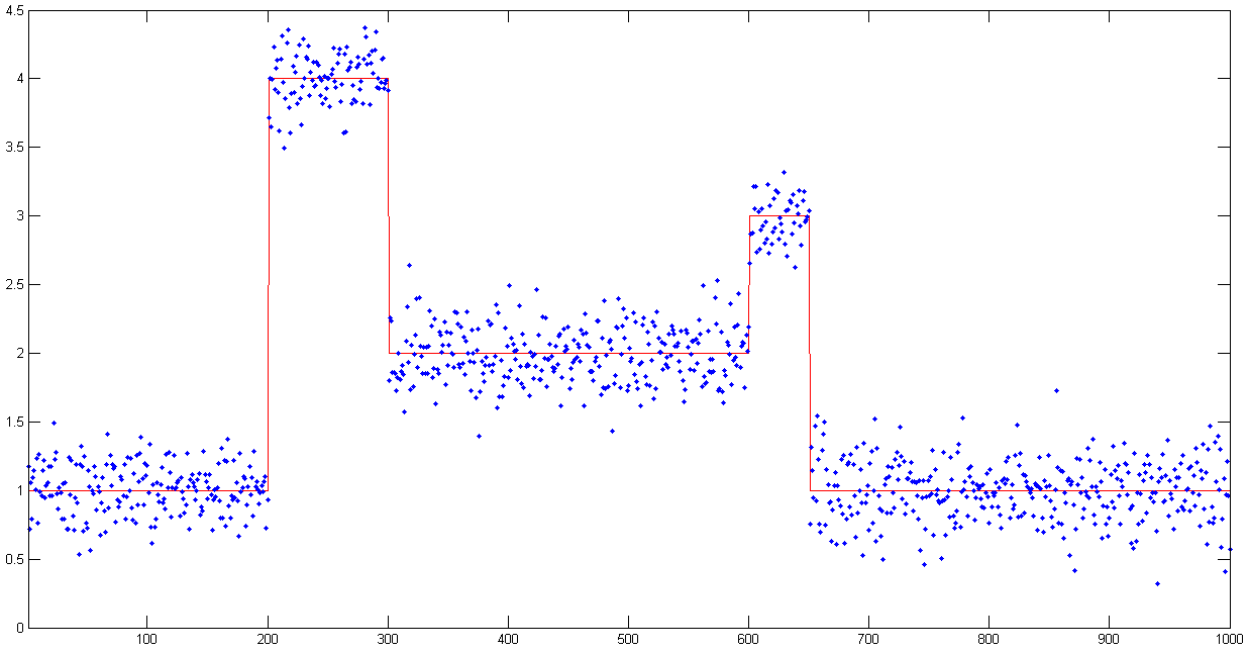

Figure 1: A signal u shown in red has been corrupted into a noisy signal u_0 shown in blue.

Figure 2: An image u shown on the left has been corrupted into a noisy image u_0 on the right.

The fundamental idea of the Tikhonov model is to seek a function u that satisfies Equations (4 - 7). We can find such a function by implementing Lagrange Multipliers. Thus, we have the optimization problem

$$\min_{u \in S} \mathcal{F} = \int_\Omega |\nabla u|^2 + \frac{\lambda}{2}(u - u_0)^2 \, dx. \tag{8}$$

1.1 Gâteaux Derivative

From calculus, we have a certain understanding that we should be able to find the minimum of a function by setting its derivative to zero. We will use the same idea here, but first we must define a derivative for functionals.

Definition 1. *Gâteaux Derivative*
Given $\mathcal{F} : L^1(\Omega) \to \mathbb{R}$, the derivative of $\mathcal{F}$ at u along η is

$$\lim_{\epsilon \to 0} \frac{\mathcal{F}(u + \epsilon\eta) - \mathcal{F}(u)}{\epsilon} = \frac{d}{d\epsilon}\mathcal{F}(u + \epsilon\eta) \, \Big|_{\epsilon=0}. \tag{9}$$

Furthermore, if there exists $\hat{u} \in (L^1(\Omega))'$ such that Equation (9) $= \hat{u}(\eta)$, then $\hat{u}$ is the Gâteaux derivative of $\mathcal{F}$ at u, and we denote it

$$\hat{u} = \frac{\delta \mathcal{F}}{\delta u}(u) \tag{10}$$

We also need the following theorem.

Theorem 1. *If $\mathcal{F}$ is convex, coercive, and lower semi-continuous, then there always exists a function u that minimizes $\mathcal{F}$. If $\mathcal{F}$ is strictly convex, the function u is unique. If $\mathcal{F}$ is Gâteaux differentiable, then u satisfies*

$$\frac{\delta \mathcal{F}}{\delta u} = 0 \tag{11}$$

We state without proof that the functional in Equation (8) satisfies the hypothesis of Theorem 1. Let us now constructively show that $\mathcal{F}$ is Gâteaux differentiable.

$$\begin{aligned}
\mathcal{F}(u) &= \int_\Omega |\nabla u|^2 + \frac{\lambda}{2}(u - u_0)^2 \, dx \\
\mathcal{F}(u + \epsilon\eta) &= \int_\Omega |\nabla u + \epsilon\nabla\eta|^2 + \frac{\lambda}{2}(u + \epsilon\eta - u_0)^2 \, dx \\
\frac{d}{d\epsilon}\mathcal{F}(u + \epsilon\eta) &= \int_\Omega \frac{d}{d\epsilon}(|\nabla u|^2 + 2\epsilon\nabla u\nabla\eta + \epsilon^2|\nabla\eta|^2) + \frac{\lambda}{2}\frac{d}{d\epsilon}(u + \epsilon\eta - u_0)^2 \, dx \\
\frac{d}{d\epsilon}\mathcal{F}(u + \epsilon\eta) &= \int_\Omega 2\nabla u\nabla\eta + 2\epsilon|\nabla\eta|^2 + \lambda(u + \epsilon\eta - u_0)\eta \, dx \\
\frac{d}{d\epsilon}\mathcal{F}(u + \epsilon\eta) \, \Big|_{\epsilon=0} &= \int_\Omega 2\nabla u\nabla\eta + \lambda(u - u_0)\eta \, dx \quad \Big| \quad \text{Apply Divergence Theorem} \\
\frac{d}{d\epsilon}\mathcal{F}(u + \epsilon\eta) \, \Big|_{\epsilon=0} &= \int_\Omega -2(\Delta u)\eta + \lambda(u - u_0)\eta \, dx + \int_{\partial\Omega} 2\eta\nabla u \cdot \vec{n} \, dx \quad \Big| \quad \text{Boundary Conditions (7)} \\
\frac{d}{d\epsilon}\mathcal{F}(u + \epsilon\eta) \, \Big|_{\epsilon=0} &= \int_\Omega \big[-2\Delta u + \lambda(u - u_0)\big]\eta \, dx \\
\frac{\delta \mathcal{F}}{\delta u} &= -2\Delta u + \lambda(u - u_0)
\end{aligned}$$

Now that we have the Gâteaux derivative, we can apply Theorem 1 and solve for u, which gives us the following Boundary Value Problem.

$$\begin{cases} -2\Delta u + \lambda(u - u_0) = 0 \\ \frac{\partial u}{\partial \vec{n}} = 0 \end{cases} \tag{12}$$

Let's recap. We started with the notion that our given noisy signal is the result of additive noise corruption: Equation (3). We made several simplifying assumptions on n and u: Equations (3 - 7). We used those assumptions to frame the problem as a minimization model: Equation (8). Lastly, using ideas from functional analysis, we turned our minimization problem into the solution of a differential equation: Equation (12).

2 Methodology

We solve Equation (12) via two methods: (1) gradient descent and (2) direct solution. (The term "direct" here is a bit misleading. We actually use iterative schemes, but we will apply those schemes directly to the elliptic PDE).

2.1 Gradient Descent

The method of gradient descent designs a sequence of energy-decreasing functions

$$\{u^k\}_{k=1}^n \tag{13}$$

that converges to the solution. To show that $\mathcal{F}$ is energy-decreasing over this sequence, consider the Taylor expansion

$$\mathcal{F}(u^{k+1}) = \mathcal{F}(u^k + \Delta t\eta) \quad \Big| \quad \text{Where } \eta \text{ is some direction.} \tag{14}$$

$$\mathcal{F}(u^{k+1}) = \mathcal{F}(u^k) + \Delta t\eta\frac{\delta\mathcal{F}}{\delta u} + ... \quad \Big| \quad \text{Let } \eta = -\frac{\delta\mathcal{F}}{\delta u} \tag{15}$$

$$\mathcal{F}(u^{k+1}) = \mathcal{F}(u^k) - \Delta t\left|\frac{\delta\mathcal{F}}{\delta u}\right|^2 + ... \tag{16}$$

Since the derivatives of $\mathcal{F}$ are bounded (u is smooth), for small enough Δt we can just ignore the higher order terms. Thus, we have shown that

$$|\mathcal{F}(u^{k+1})| \leq |\mathcal{F}(u^k)| \tag{17}$$

so $\mathcal{F}$ is energy decreasing over the sequence that progresses in the direction specified in Equation (15). We can now consider our elliptic differential equation as a time-dependent parabolic differential equation. For convenience, we rewrite the given noisy signal u_0 as u^0.

$$\begin{cases} \frac{\partial u}{\partial t} = 2\Delta u - \lambda(u - u^0) \\ \frac{\partial u}{\partial \vec{n}} = 0 \\ u(x, t = 0) = u^0 \end{cases} \tag{18}$$

Note: We don't have to start with initial condition $u(x, t = 0) = u^0$. However, since the means are equal (Equation (5)), this is likely a good starting point.

2.1.1 Numerical Implementation

We discretize the PDE in both space and time using second order centered difference and forward Euler, respectively. For convenience, we scale our spatial dimension so that $\Delta x = 1$. For a one-dimensional case, this gives us

$$u_j^{n+1} = u_j^n + 2\Delta t(u_{j-1}^n - 2u_j^n + u_{j+1}^n) - \Delta t\lambda(u_j^n - u_j^0). \tag{19}$$

We handle the boundary conditions with a second order first derivative

$$u_1^n - u_{-1}^n = 0, \qquad u_{N+1}^n - u_{N-1}^n = 0.$$

The stability condition on this scheme is

$$2\Delta t \leq \frac{1}{2}$$
$$\Delta t \leq \frac{1}{4}$$

The two dimensional case is very similar. Just as above, we scale the problem so that $\Delta x = \Delta y = 1$.

$$u_{j,k}^{n+1} = u_{j,k}^n + 2\Delta t(u_{j-1,k}^n + u_{j+1,k}^n + u_{j,k-1}^n + u_{j,k-1}^n - 4u_{j,k}^n) - \Delta t\lambda(u_{j,k}^n - u_{j,k}^0). \tag{20}$$

with boundary conditions

$$\begin{aligned} u_{1,k}^n - u_{-1,k}^n = 0, \qquad & u_{N+1,k}^n - u_{N-1,k}^n = 0 \qquad \text{For all } k \\ u_{j,1}^n - u_{j,-1}^n = 0, \qquad & u_{j,N+1}^n - u_{j,N-1}^n = 0 \qquad \text{For all } j \end{aligned}$$

and stability condition

$$\Delta t < \frac{1}{8}$$

2.2 Direct Solution

Instead of creating artificial time, we can directly solve the elliptic PDE.

$$\begin{cases} -2\Delta u + \lambda(u - u_0) = 0 \\ \frac{\partial u}{\partial \vec{n}} = 0 \end{cases} \tag{21}$$

Just as before, we implement a second order centered difference discretization for our spatial variable with $\Delta x = \Delta y = 1$. In two-dimensions, this gives us

$$-2(u_{j-1,k}^n + u_{j+1,k}^n + u_{j,k-1}^n + u_{j,k-1}^n - 4u_{j,k}^n) + \lambda u_{j,k} = \lambda u_{j,k}^0 \tag{22}$$

with boundary conditions

$$\begin{aligned} u_{1,k} - u_{-1,k} = 0, \qquad & u_{N+1,k} - u_{N-1,k} = 0 \qquad \text{For all } k \\ u_{j,1} - u_{j,-1} = 0, \qquad & u_{j,N+1} - u_{j,N-1} = 0 \qquad \text{For all } j \end{aligned}$$

Implementing this scheme essentially gives us $Au = b$. Normally, the discrete Laplacian with Neumann conditions is singular. However, with nonzero λ, the matrix becomes nonsingular. Direct solution techniques can be costly for very large matrices, even if they are sparse. To deal with this, we will implement Gauss-Seidel and Conjugate Gradient methods.

3 Results

3.1 Gradient Descent

Our one-dimensional numerical experiments were done using $\Delta x = 1$, $\Delta t = 1/5$, $tol = 1 \times 10^{-6}$, and convergence is measured in the L^2-norm. Our two-dimensional numerical experiments were done using $\Delta x = 1$, $\Delta y = 1$, $\Delta t = 1/9$, $tol = 1 \times 10^{-3}$, and convergence is measured in the L^2-norm. For $0 \leq \lambda < 2$, we see convergence. Figures (3 - 7) show examples of both the one-dimensional signal and two-dimensional image for various values of λ.

3.2 Finite Difference

Our one-dimensional numerical experiments were done with $\Delta x = 1$, $tol = 1 \times 10^{-6}$, and convergence measured in the L^2-norm. Our two-dimensional numerical experiments were done with $\Delta x = 1$, $\Delta y = 1$, $tol = 1 \times 10^{-3}$, and convergence measure in the L^2-norm. Since we are not time-stepping, we don't have any stability requirements. The only restriction is that the elliptic operator be nonsingular, which requires $\lambda \neq 0$. Conjugate gradient returns the same results as Gauss-Seidel and is asymptotically faster. Thus, we only show results from Conjugate Gradient for both our one-dimensional signal and two-dimensional image for various values of λ. These are presented in Figures (9 - 16).

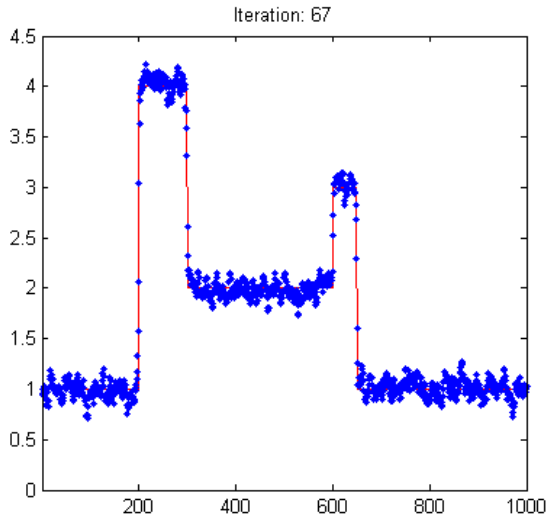

Figure 3: Gradient Descent method with $\lambda = 1$. System quickly converges to steady-state solution. However, the signal is still very noisy.

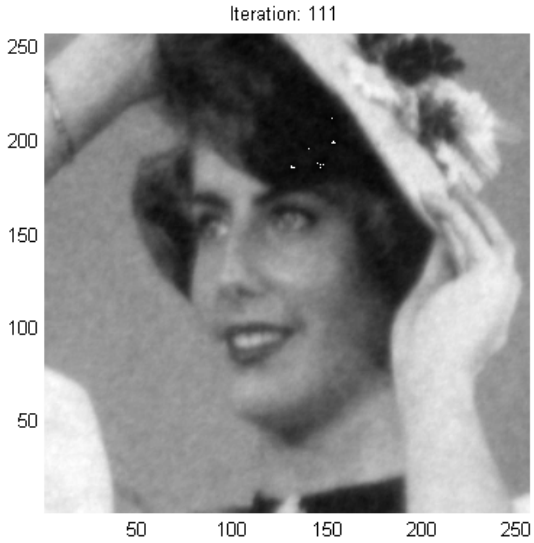

Figure 4: Gradient Descent method with $\lambda = 1$. System quickly converges to steady-state solution. However, the image is still very noisy.

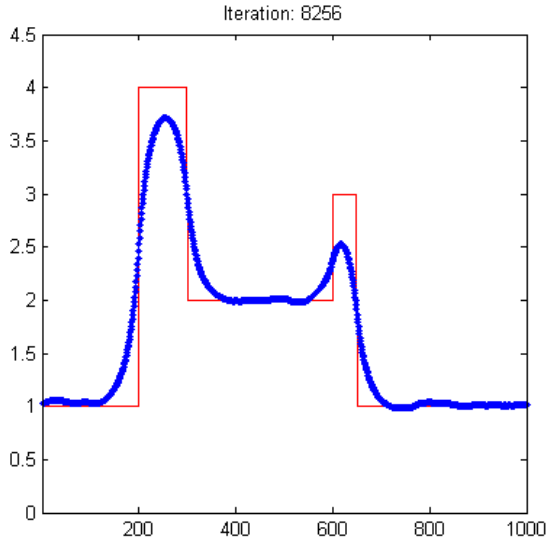

Figure 5: Gradient Descent method with $\lambda = 0.004$. Very slow convergence. The signal is no longer noisy, but it has smoothed out the edges.

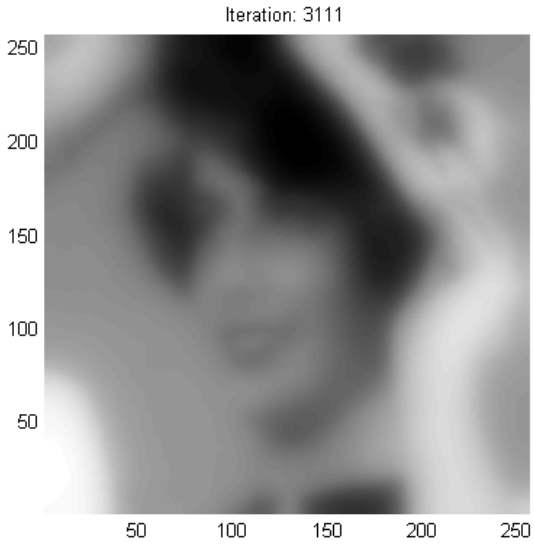

Figure 6: Gradient Descent method with $\lambda = 0.02$. Very slow convergence. The image is no longer noisy, but it has become very blurred.

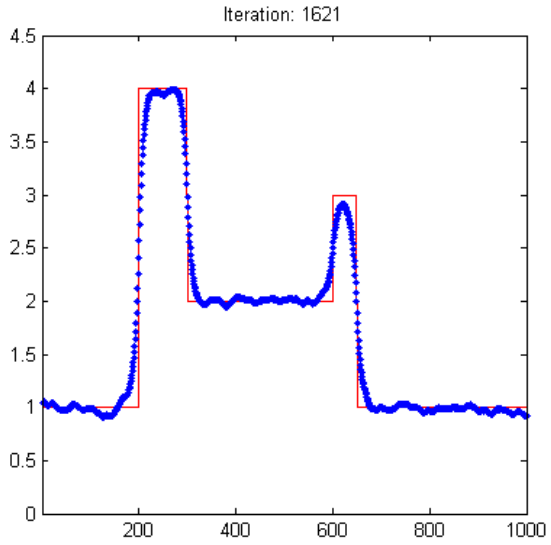

Figure 7: Gradient Descent method with $\lambda = 0.025$. The system converged in a reasonable number of iterations. We have attempted to balance between noise and smoothness.

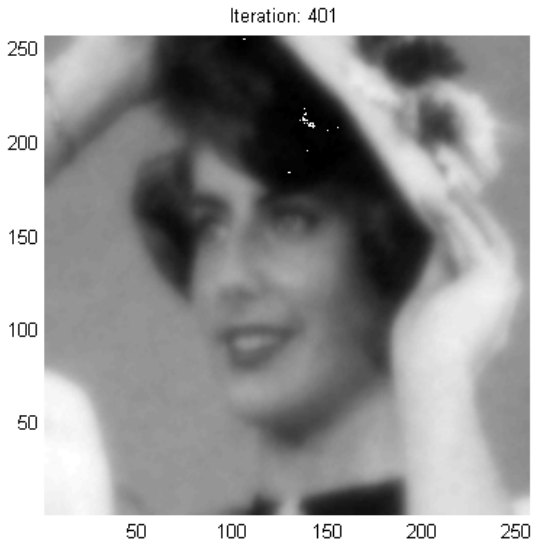

Figure 8: Gradient Descent method with $\lambda = 0.2$. The system converged in a reasonable number of iterations. We have attempted to balance between noise and blur.

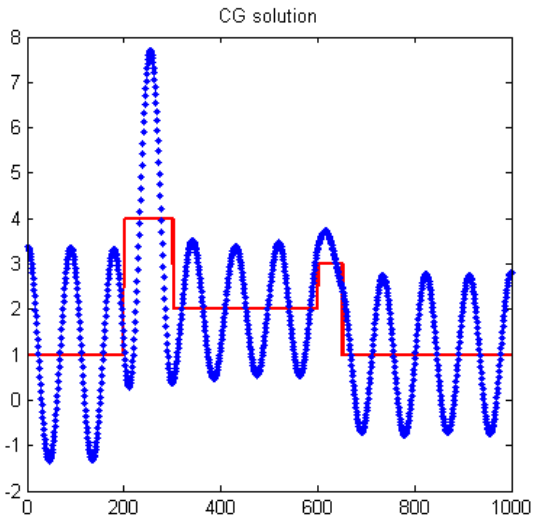

Figure 9: Conjugate Gradient method with $\lambda = -0.01$. While the system is nonsingular for this choice of λ, the signal is completely destroyed.

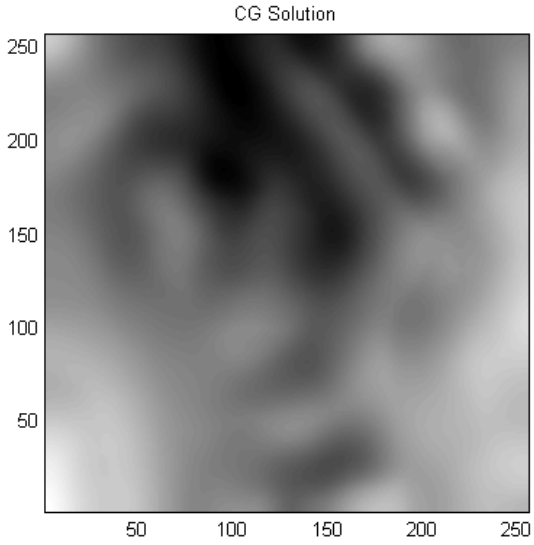

Figure 10: Conjugate Gradient method with $\lambda = -0.01$. While the system is nonsingular fo this choice of λ, the image is completely destroyed.

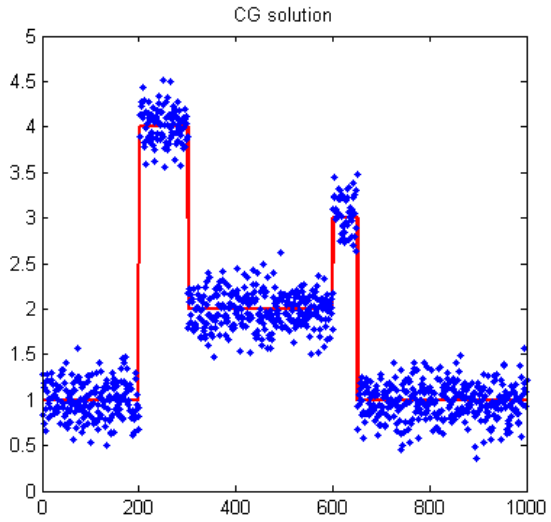

Figure 11: Conjugate Gradient method with $\lambda = 1000$. The convergence is almost instantaneous, but the result is useless. The signal has not been denoised at all.

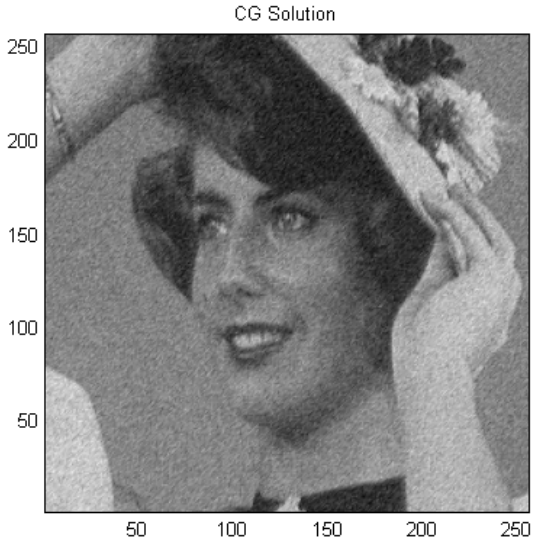

Figure 12: Conjugate Gradient method with $\lambda = 1000$. The convergence is almost instantaneous, but the result is useless. The image has not been denoised at all.

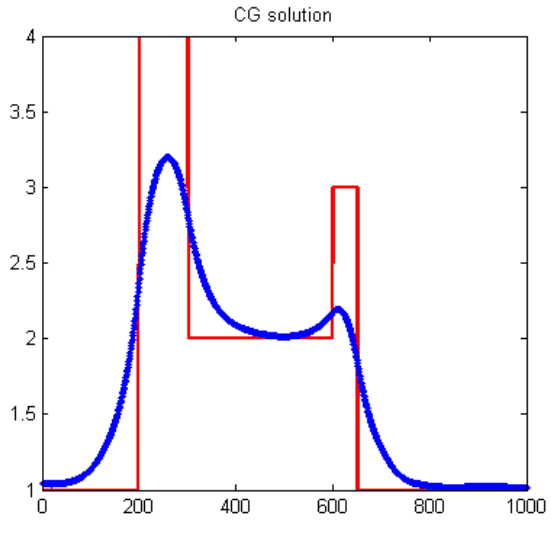

Figure 13: Conjugate Gradient method with $\lambda = 0.001$. Convergence in just under 400 iterations. The signal is denoised but has introduced undesired smoothing.

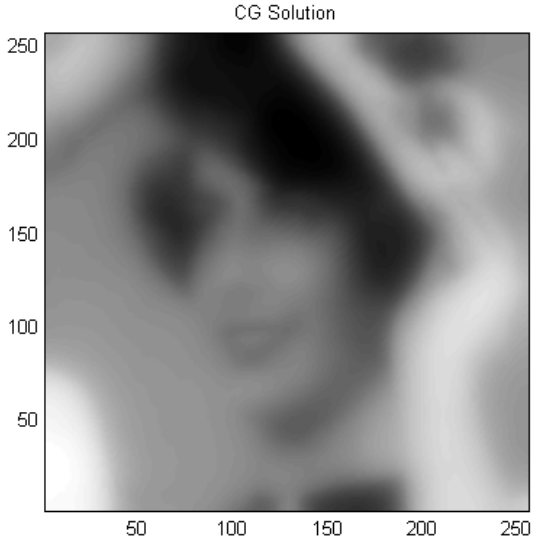

Figure 14: Conjugate Gradient method with $\lambda = 0.02$. Convergence in about 50 iterations. The image is denoised but is too blurry.

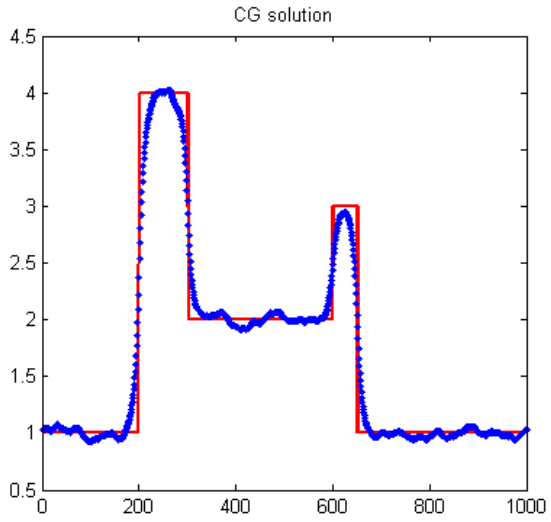

Figure 15: Conjugate Gradient method with $\lambda = 0.025$. The system converged very quickly. We have attempted to balance between noise and blur.

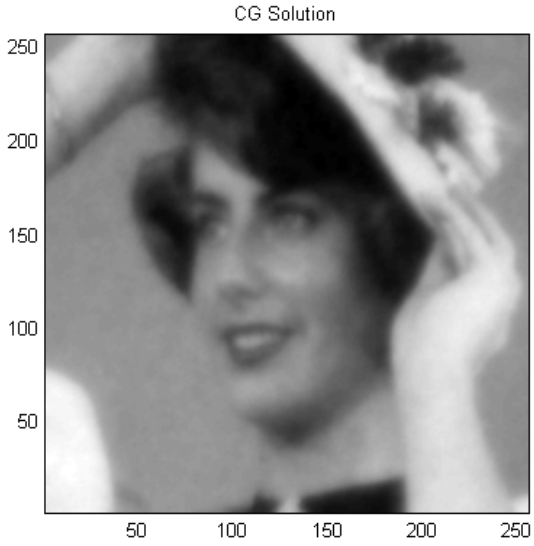

Figure 16: Conjugate Gradient method with $\lambda = 0.25$. The system converged very quickly. We have attempted to balance between noise and blur.

4 Observation and Conclusions

Among our numerical implementations, conjugate gradient was clearly the optimal choice. It converged quickly even when given non-symmetric matrices. The stability restriction on Gradient Descent requires too many iterations; an ADI scheme would greatly benefit this method. Gauss-Seidel was definitely the worst option. The convergence rate is not as fast as conjugate gradient, and forming the $(L+D)^{-1}$ matrix (explicitly or implicitly) resulted in huge overhead. Finally, we note that even though the systems are very large, direct solve is still the fastest choice. While LU factorization is out of the question, Matlab's tridiagonal and block-tridiagonal solvers performed even faster than Conjugate Gradient on symmetric matrices.

From the above figures, we immediately see that while many values of λ are mathematically valid, most of those values don't fit our purposes. We saw solutions that were too blurry, too noisy, and solutions with completely destroyed signals. By testing different values of λ, we arrived at what we believe to be an optimal balance between noise and blur. See Figures (7 - 8) for Gradient Descent and (15 - 16) for Conjugate Gradient.

The Tikhonov model (8) assumes a smooth solution, which forces us to balance between noise and blur. We could construct a criterion that resulted in the optimal balance, but that is like choosing the best of bad options. Instead, we will suggest (without giving detail) a model that allows us to achieve smoothness where we want while still maintaining edges.

If we choose to minimize the gradients in the L^1-norm instead of the the L^2, then the variational of our functional becomes

$$\frac{\delta \mathcal{F}}{\delta u} = -div\left(\frac{\nabla u}{|\nabla u|}\right) + \frac{\lambda}{2}(u - u_0) \tag{23}$$

If ∇u is small (tangent to an edge), then the divergence will be large, and smoothing will take place. If ∇u is large (across and edge), the divergence will be small, and very little smoothing will take place. We will see more of this in future reports.

Reference

Note: No outside sources were used. Everything was done directly from the lecture notes and previous knowledge of numerical schemes.

Appendix

One-Dimensional Gradient Descent

```
% Edward Rusu
% Math 6590
% Variational Image Processing
% Project 1
% 1D Signal - Gradient Descent

% This program will solve the Tikhanov optimization problem using gradient
% descent. We will just do the one-dimensional noisy signal here.

clearvars
clc


%% Load Data
[uxact u0] = LoadData; % Load the 1-dimensional signal
N = length(u0); % Number of nodes

maxIter = 100000; % Maximum number of iterations
tol = 1e-6; % Convergence tolerance
exitflag = 0; % Convergence condition
L = 2; % Norm to check convergence

lambda = 1; % Value for lambda
r = 1/5; % CFL conditions, 1/4 is max for stability

% Rather than using ghost points for the Neumann condition, we simply treat
% them as algebraic constraints, do some solving, and find the correct
% expression for our interior points


%% Create 1D Neumann-Laplacian Operator
e = ones(N,1);
Lapl = spdiags([e -2*e e],[-1 0 1],N,N); % Interior Operator
Lapl(1,2) = 2; % Left boundary
Lapl(N,N-1) = 2; % Right boundary


%% Gradient Descent
uN = u0; % Initial condition

figure(1), clf
for j = 1:maxIter
    % Iterate
    u0 = uN;
    uN = u0 + r*2*Lapl*u0 - r*lambda*(u0-u0);

    % Animation
    if (mod(j,1) == 0) % Only occasionally plot
        plot(1:N,uxact,'r-',1:N,uN,'b.'), axis([1 1000 0 4.5])
        title(['Iteration: ' num2str(j)])
        drawnow
        pause(0.01)
    end

    % Check convergence
```

```
    if (norm(uN - u0,L) < tol)
        exitflag = 1; % Converged
        break;
    end
end

if (exitflag == 1)
    disp(['Converged in ' num2str(j) ' iterations.']);
else
    disp('Did not converge. Consider increasing max iteration or loosening tolerance.');
end
```

Two-Dimensional Gradient Descent

```
% Edward Rusu
% Math 6590
% Variational Image Processing
% Project 1
% 2D Signal - Gradient Descent

% This program will solve the Tikhanov optimization problem using gradient
% descent. We will just do the two-dimensional noisy image here.

clearvars
clc


%% Load Data
[~,~,uxact u0_] = LoadData; % Load the 2-dimensional signal
N = length(u0_); % Number of nodes in a single dimension
u0 = u0_(:); % Reshape by stacking columns

maxIter = 5000; % Maximum number of iterations
tol = 1e-3; % Convergence tolerance
exitflag = 0; % Convergence condition
L = 2; % Norm to check convergence

lambda = 0.2; % Value for lambda
r = 1/9; % CFL conditions, 1/8 is max for stability

% Rather than using ghost points for the Neumann condition, we simply treat
% them as algebraic constraints, do some solving, and find the correct
% expression for our interior points


%% Create 2D Neumann-Laplacian Operator

% First create 1D operator
e = ones(N,1);
Lapl1 = spdiags([e -2*e e],[-1 0 1],N,N); % Interior Operator
Lapl1(1,2) = 2;
Lapl1(N,N-1) = 2; % Right boundary

% Then kron into 2D operator
Lapl2 = kron(Lapl1,speye(N)) + kron(speye(N),Lapl1);


%% Gradient Descent
uN = u0; % Initial Condition
```

```
figure(3), clf
for j = 1:maxIter
    % Iterate
    u0 = uN;
    uN = u0 + r*2*Lapl2*u0 - r*lambda*(u0 - u0);

%     Animation
    if (mod(j,10) == 1) % Only occasionally plot
        pcolor(flipud(reshape(uN,N,N))), axis tight, colormap gray, shading interp
        title(['Iteration: ' num2str(j)])
        drawnow
        pause(0.01)
    end

    % Check convergence
    if (norm(uN - u0,L) < tol)
        exitflag = 1; % Converged
        break;
    end
end

if (exitflag == 1)
    disp(['Converged in ' num2str(j) ' iterations.']);
else
    disp('Did not converge. Consider increasing max iteration or loosening tolerance.');
end
```

One-Dimensional Iterative

```
% Edward Rusu
% Math 6590
% Variational Image Processing
% Project 1
% 1D Signal - GS and CG

% This program will solve the Tikhanov optimization problem using
% Gauss-Seidel and Conjugate Gradient Methods. We will just do the
% one-dimensional noisy signal here.

clearvars
clc


%% Load Data
[uxact u0] = LoadData; % Load the 1-dimensional signal
N = length(u0); % Number of nodes

maxIter = 10000; % Maximum number of iterations
tol = 1e-6; % Convergence tolerance
exitflag = 0; % Convergence condition
L = 2; % Norm to check convergence

lambda = 0.025; % Value for lambda

% Rather than using ghost points for the Neumann condition, we simply treat
% them as algebraic constraints, do some solving, and find the correct
% expression for our interior points
```

```
%% Create 1D Elliptic Operator and RHS
e = ones(N,1);
Elli = lambda*speye(N) + spdiags([-2*e 4*e -2*e],[-1 0 1],N,N); % Set up Elliptic Operator
Elli(1,2) = -4; Elli(N,N-1) = -4;

rhs = lambda*u0;


%% Exact

% Before doing anything, we will perform a direct solve for comparison
udir = Elli\rhs; % Direct solve with tri-diagonal solver

figure(1), clf
plot(1:N, uxact,'r-','linewidth',2), hold on
plot(1:N, udir,'b.');
title('Exact Solution');


%% Gauss-Seidel
% Gauss-Seidel is x_{k+1} = (I-inv(P)*A)x_{k} + inv(P)*b

Elli_diag = spdiags(Elli); % Grab lower and main diagonal from Elli
PC = spdiags([Elli_diag(:,1) Elli_diag(:,2)], [-1 0],N,N); % Create GS preconditioner
PCinv = PC\speye(N); % Invert the preconditioner
rhsPC = PCinv*rhs; % Modify the rhs vector
GS = speye(N) - PCinv*Elli; % Gauss-Seidel Matrix = I-inv(P)*A

% Iteration here
uN = u0; % Initial condition

figure(1), clf
for n = 1:maxIter
    % Iterate
    uO = uN;
    uN = GS*uO + rhsPC;

    % Animate
    if (mod(n,1) == 0) % Only occasionally plot
        plot(1:N,uxact,'r-',1:N,uN,'b.'), axis([1 1000 0 4.5])
        title(['GS iteration: ' num2str(n)]);
        drawnow
        pause(0.01)
    end

    % Check Convergence
    if (norm(uN-uO,L) < tol)
        exitflag = 1;
        break;
    end
end

if (exitflag == 1)
    disp(['GS converged in ' num2str(n) ' iterations.']);
else
    disp('GS did not converge. Consider increasing max iteration or loosening tolerance.');
end
```

```
%% Conjugate Gradient

[uCG,exitflag,res,iterOut] = cgs(Elli,rhs,tol,maxIter,[],[],u0);

if (exitflag == 0)
    disp(['CG converged in ' num2str(iterOut) ' iterations.']);
else
    disp('CG did not converge. Consider increasing max iteration or loosening tolerance.');
end

figure(4), clf
plot(1:N, uxact,'r-','linewidth',2), hold on
plot(1:N, uCG,'b.');
title('CG solution');
```

Two-Dimensional Iterative

```
% Edward Rusu
% Math 6590
% Variational Image Processing
% Project 1
% 2D Signal - GS and CG

% This program will solve the Tikhanov optimization problem using
% Gauss-Seidel and Conjugate Gradient Methods. We will just do the
% two-dimensional noisy signal here.

clearvars
clc


%% Load Data
[~,~,uxact u0] = LoadData; % Load the 2-dimensional signal
N = length(u0); % Number of nodes in one dimension
u0 = u0(:); % Stack the columns

maxIter = 1000; % Maximum number of iterations
tol = 1e-3; % Convergence tolerance
exitflag = 0; % Convergence condition
L = 2; % Norm to check convergence

lambda = 0.25; % Value for lambda

% Rather than using ghost points for the Neumann condition, we simply treat
% them as algebraic constraints, do some solving, and find the correct
% expression for our interior points


%% Create 2D Elliptic Operator and RHS
% First create 1D
e = ones(N,1);
Elli = spdiags([e -2*e e],[-1 0 1],N,N); % 1D
Elli(1,2) = 2; Elli(N,N-1) = 2;

% Then kron for 2d
Elli2_ = kron(speye(N),Elli) + kron(Elli,speye(N));
Elli2 = -2*Elli2_ + lambda*speye(N^2);
```

```
rhs = lambda*u0;


%% Exact
% Before doing anything, we will perform a direct solve for comparison
udir = Elli2\rhs; % Direct solve with elimination

figure(1), clf
subplot(1,2,1)
pcolor(flipud(uxact)), colormap gray, axis tight, shading interp
subplot(1,2,2)
pcolor(flipud(reshape(udir,N,N))), colormap gray, axis tight, shading interp


%% Gauss-Seidel

% Gauss-Seidel involves the inversion of a lower-triangular matrix. For
% this case, the matrix is so large (almost 60,000) that this inversion is
% impractical. Instead, we will implement Gauss-Seidel with a for loop.

figure(2), clf
uN = rhs;
for n = 1:maxIter
    % Iterate
    uO = uN;
    for j = 1:N^2
        E_dot = Elli2*uN;
        uN(j) = 1./Elli2(j,j)*(rhs(j) - E_dot(j) + Elli2(j,j)*uN(j));
    end

    % Animate
    if (mod(n,1) == 0) % Only occasionally plot
        pcolor(flipud(reshape(uN,N,N))), axis tight, colormap gray, shading interp
        title(['GS iteration: ' num2str(n)]);
        drawnow
        pause(0.01)
    end

    % Check convergence
    if (norm(uN-uO,L) < tol)
        exitflag = 1;
        break;
    end
end

if (exitflag == 1)
    disp(['GS converged in ' num2str(n) ' iterations.']);
else
    disp('GS did not converge. Consider increasing max iteration or loosening tolerance.');
end


%% Conjugate Gradient

[uCG,exitflag,res,iterOut] = cgs(Elli2,rhs,tol,maxIter,[],[],u0);

if (exitflag == 0)
    disp(['CG converged in ' num2str(iterOut) ' iterations.']);
else
```

```
    disp('CG did not converge. Consider increasing max iteration or loosening tolerance.');
end

figure(3), clf
temp = reshape(uCG,N,N);
pcolor(flipud(reshape(uCG,N,N))), axis tight, colormap gray, shading interp
title('CG Solution');
```