Binary Signal Recovery - Barcode Reading, Deconvolution, & Regularization

Carter Kruse*

Abstract. Binary signal recovery is a fundamental process in digital communication and signal processing, aimed at reconstructing accurate digital information from a distorted or noisy transmission. This paper aims to investigate binary signal recovery using various regularization techniques in solving the problem of reading barcodes.

Key Words. Binary Signal Recovery, Computational Inverse Problem, Deconvolution, Regularization

AMS Subject Classifications. XXXXX

1. Introduction.

Binary signal recovery aims to determine original digital information, given a transmission influenced by noise or distortion. In today's digital world, various communication systems use binary signals, consisting of 0s and 1s, to transmit data. During transmission, signals may encounter interference, noise, or distortion, which may lead to errors or corruption of the original signal.

Binary signal recovery involves employing various algorithms and techniques to restore the original signal, thereby mitigating the effects of noise and distortion. By effectively recovering the binary signal, reliable communication and accurate data interpretation may be achieved. This paper aims to investigate binary signal recovery using various regularization techniques in solving the problem of reading barcodes.

The paper is organized as follows. The context is in section 2, the motivation is in section 3, the methodology is in section 4, the results are in section 5, and the conclusions follow in section 6.

2. Context.

The following provides the mathematical context for computational inverse problems, specifically the various regularization techniques used in the process of binary signal recovery.

Ill-Posed Problems.

Computational inverse problems deal with those where the output is known (with errors), though either the input or system is unknown. Typically, these problems belong to a class of *ill-posed* problems. A linear problem is well-posed if it satisfies three conditions.

- Existence: The problem has a solution.
- Uniqueness: The problem has at most one solution.
- Stability: The solution depends *continuously* on the data.

A problem is *ill-posed* if it fails any of these. Typically, we consider lack of "stability", which involves the consequence that arbitrarily small perturbations of the data can produce arbitrarily large perturbations of the solution. The way we approach these problems is to reformulate (stabilize/regularize the problem).

*Dartmouth College, Hanover, NH (carterjkruse@gmail.com).

The reason for lack of "stability" in a linear system $A\boldsymbol{x} = \boldsymbol{b}$ is that A is *ill-conditioned* (nearly singular), meaning the problem is effectively *under-determined*. This allows for small changes to the residual despite changes of a (nearly) null vector the vector solution.

Regularization. As such, we modify the problem so that the new solution is more stable. For example, we can enforce an upper bound δ on the norm of the solution.

$$\min_{\boldsymbol{x}} ||A\boldsymbol{x} - \boldsymbol{b}||_2 \text{ subject to } ||\boldsymbol{x}||_k < \delta$$

The solution x_δ depends in a unique, but *non-linear* way on δ.

Fredholm Integral Equation. The continuous inverse problem that is analogous to $A\boldsymbol{x} = b$ is the Fredholm integral equation of the first kind, which takes the form

$$\int_0^1 K(s,t) f(t)\, dt = g(s) \qquad 0 \leq s \leq 1$$

Here, the *kernel* K and the right-hand side g are known functions, while f is unknown. The equation establishes a relationship between f and g and K describes the precise relationship between the two quantities. If f and K are known, then we can compute g by evaluating the integral. This is called the *forward computation*. The *inverse problem* consists of computing f given g and K.

Convolution. When $K(s,t) = h(s-t)$, this is a convolution problem. When we are trying to recover f from known g, this *inverse problem* is called *deconvolution*. In essence, g can be viewed as a sort of "smoothed" form of f. This is a universal phenomenon for integral equations. In the mapping from f to g, "higher frequency" components of f are dampened compared to components of lower frequencies. So g will appear smoother than f.

The *Riemann-Lebesgue Lemma* is a precise mathematical statement of this. See *Discrete Inverse Problems* [1] for further explanation. Given this information, the *inverse problem* is a process that *amplifies* high frequencies, and the higher the frequency, the greater the amplification. This results in issues with "stability" as a small perturbation in a high frequency term may be amplified in the inverse problem.

In other words, if g has a small perturbation, this could correspond to an arbitrary large perturbation in f_p, where p is large, i.e.

$$g_\varepsilon(s) = g(s) + \varepsilon \int_0^1 K(s,t) f_p(t)\, dt$$

So if g is affected by some sort of error, this can have a drastic impact on the inverse problem. Discretization itself introduces errors when the underlying phenomenon is continuous.

Singular Value Decomposition. Given $A\boldsymbol{x} = \boldsymbol{b}$, where $A \in \mathbb{R}^{m\times n}$, we want to develop tools to analyze the issue of stability and perhaps incorporate regularization. Assuming $m \geq n$, for $A \in \mathbb{R}^{m\times n}$, the SVD of A is

$$A = U\Sigma V^T = \sum_{i=1}^{n} \boldsymbol{u}_i \sigma_i \boldsymbol{v}_i^T$$

Σ is an $n \times n$ diagonal matrix with the *singular values* $\sigma_1, \ldots, \sigma_n$.

$$\Sigma = \operatorname{diag}(\sigma_1, \ldots, \sigma_n), \sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n \geq 0$$

Matrices $U \in \mathbb{R}^{m \times n}$ and $V \in \mathbb{R}^{n \times n}$ consist of the left and right *singular vectors.*

$$U = [\boldsymbol{u}_1, \boldsymbol{u}_2, \ldots, \boldsymbol{u}_n] \quad V = [\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_n]$$

The matrices have orthonormal columns, i.e. $\boldsymbol{u}_i^T \boldsymbol{u}_j = \boldsymbol{v}_i^T \boldsymbol{v}_i = \delta_{i,j}$, or simply $U^T U = V^T V = \mathbb{I}_n$, considering

$$\delta_{i,j} \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases}$$

To find the SVD of A, we must use linear algebra. The eigenvectors of $A^T A$ make up the columns of V. The eigenvectors of AA^T make up the columns of U. The square root of eigenvalues of either $A^T A$ or AA^T make up Σ.

Condition Number. Given $A\boldsymbol{x} = \boldsymbol{b}$, how can we determine how inaccurate $\boldsymbol{x}$ may be after approximation? The *condition number* allows us to do this and is given by the following:

$$\operatorname{cond}(A) = ||A||_2 \left|\left|A^{-1}\right|\right|_2 = \sigma_1 / \sigma_n$$

To determine this, let ε be the error, i.e. $A\boldsymbol{x}_\varepsilon = \boldsymbol{b} + \boldsymbol{\varepsilon}$, where $\boldsymbol{x}_\varepsilon$ is the perturbed result/solution. Then, if A^{-1} exists, $\boldsymbol{x}_\varepsilon = A^{-1}\boldsymbol{b} + A^{-1}\boldsymbol{\varepsilon}$. The following indicates how the condition number is determined:

$$\begin{aligned}
\operatorname{cond}(A) &= \max_{\boldsymbol{\varepsilon}, \boldsymbol{b} \geq 0} \frac{\frac{\left|\left|A^{-1}\epsilon\right|\right|_2}{||A^{-1}\boldsymbol{b}||_2}}{\frac{||\boldsymbol{\varepsilon}||_2}{||\boldsymbol{b}||_2}} \\
\operatorname{cond}(A) &= \max_{\boldsymbol{\varepsilon}, \boldsymbol{b} \geq 0} \frac{\left|\left|A^{-1}\boldsymbol{\varepsilon}\right|\right|_2}{||\boldsymbol{\varepsilon}||_2} \frac{||\boldsymbol{b}||_2}{||A^{-1}\boldsymbol{b}||_2} \\
&= \max_{\boldsymbol{\varepsilon} \neq 0} \left\{ \frac{\left|\left|A^{-1}\boldsymbol{\varepsilon}\right|\right|_2}{||\boldsymbol{\varepsilon}||_2} \right\} \max_{\boldsymbol{\varepsilon} \neq 0} \left\{ \frac{||\boldsymbol{b}||_2}{||A^{-1}\boldsymbol{b}||_2} \right\} \\
&= \left|\left|A^{-1}\right|\right|_2 \max_{\boldsymbol{x} \neq 0} \left\{ \frac{||A\boldsymbol{x}||_2}{||\boldsymbol{x}||_2} \right\} \\
&= \left|\left|A^{-1}\right|\right|_2 ||A||_2
\end{aligned}$$

Role Of SVD. To compute $\boldsymbol{x} = A^{-1}\boldsymbol{x}$ using the singular value decomposition, consider the following, from *Discrete Inverse Problems* [1].

$$A\boldsymbol{x} = \sum_{i=1}^{n} \left(\boldsymbol{v}_i^T \boldsymbol{x}\right) A\boldsymbol{v}_i = \sum_{i=1}^{n} \left(\boldsymbol{v}_i^T \boldsymbol{x}\right) \boldsymbol{u}_i \sigma_i$$

$$\boldsymbol{b} = \sum_{i=1}^{n} \left(\boldsymbol{u}_i^T \boldsymbol{b}\right) \boldsymbol{u}_i$$

Thus,

$$A\boldsymbol{x} = b \rightarrow \sum_{i=1}^{n} \sigma_i \left(\boldsymbol{v}_i^T \boldsymbol{x}\right) \boldsymbol{u}_i = \sum_{i=1}^{n} \left(\boldsymbol{u}_i^T \boldsymbol{b}\right) \boldsymbol{u}_i$$

For every i (given U is orthonormal), $\sigma_i \geq 0$ (since A is invertible), so

$$\boldsymbol{x} = \frac{\boldsymbol{u}_i^T \boldsymbol{b}}{\sigma_i} \boldsymbol{v}_i$$

This is the standard form of singular value decomposition that is used for regularization of ill-posed problems.

Discrete Picard Condition. Let τ denote the level at which the computed singular values σ_i level off due to rounding errors. The discrete *Picard* condition is satisfied if, for all σ_i larger than τ, the corresponding coefficients $\left|u_i^T b\right|$ on average, decay faster than σ_i.

This condition is used to determine if the truncated SVD model is appropriate for regularization of ill-posed problems. That is, when the discrete *Picard* condition is satisfied, a meaningful solution may be obtained from the truncated SVD.

Regularization. Please read Chapter 4 of *Discrete Inverse Problems* [1] for an in-depth understanding of the various regularization techniques used throughout this paper. A high-level overview is provided as follows.

Truncated SVD. Truncated SVD is a regularization technique used to approximate a matrix A by retaining only a subset of its most important singular values and corresponding singular vectors. The singular value decomposition is given as follows.

$$\boldsymbol{x} = \frac{\boldsymbol{u}_i^T \boldsymbol{b}}{\sigma_i} \boldsymbol{v}_i$$

The truncated SVD only consider the first k values of i. To determine the value of k, we use the *Picard* log plot, demonstrated later in this paper. When σ_i is small, the contribution to $\boldsymbol{x}$ can be very large.

This regularization method is particularly useful when dealing with high-dimensional data or when trying to reduce the dimensionality while preserving significant information from the original signal. Further, the method is used to approximate a matrix with lower-rank, which is useful in removing noise from data (as in this case).

Tikhonov/Ridge (L2) Regularization. Tikhonov/Ridge (L2) regularization is a technique used to stabilize and regularize ill-posed problems, by adding a regularization term to control the solution's sensitivity to noise/numerical instability.

Given the linear inverse problem $A\boldsymbol{x} = \boldsymbol{b}$, were A is a given matrix and b is observed, issues are encountered when the matrix A is ill-conditioned, as the solution is sensitive to small perturbations in the data. To address this, we introduce a regularization term that encourages a solution with the desirable property of *smoothness*. This solution is found by minimizing the following objective function:

$$\boldsymbol{x}_\lambda = \arg\min_{\boldsymbol{x}} \|A\boldsymbol{x} - \boldsymbol{b}\|^2 + \lambda^2 \|\boldsymbol{x}\|_2^2$$

The value λ is the regularization parameter that controls the trade-off between fitting the data and the regularization term. Tikhonov/Ridge (L2) regularization penalizes solutions with large magnitudes by adding a term that considers the square of the solution's norm, thus encouraging solutions that are close fitting to the data while adhering to the regularization properties (smallness).

The value of λ determines the strength of the regularization, and is typically determined using the L-curve for Tikhonov/Ridge (L2) regularization. The choice of this λ highlights the trade-off between fitting the data and regularization.

Lasso (L1) Regularization. Lasso (L1) regularization is a further technique used to stabilize and regularize ill-posed problems, by adding a regularization term to control the solution's sensitivity to noise/numerical instability.

Given the linear inverse problem $A\boldsymbol{x} = \boldsymbol{b}$, were A is a given matrix and b is observed, issues are encountered when the matrix A is ill-conditioned, as the solution is sensitive to small perturbations in the data. To address this, we introduce a regularization term that encourages a solution with the desirable property of *sparsity*. This solution is found by minimizing the following objective function:

$$\boldsymbol{x}_\lambda = \arg\min_{\boldsymbol{x}} \|A\boldsymbol{x} - \boldsymbol{b}\|^2 + \lambda\|\boldsymbol{x}\|_1$$

The value λ is the regularization parameter that controls the trade-off between fitting the data and the regularization term. Lasso (L1) regularization penalizes solutions that are not sparse by adding a term that considers this, thus encouraging solutions that are close fitting to the data while adhering to the regularization properties (sparsity).

The value of λ determines the strength of the regularization, and the choice of this λ highlights the trade-off between fitting the data and regularization.

Typically, a sparsifying transform matrix L is used, which acts as a derivative operator on $\boldsymbol{x}$. To compute the $L1$ solution, we may use the matrix L as follows. Let $\boldsymbol{s}$ in $\mathbb{R}^n$ such that $\boldsymbol{x} = L^{-1}\boldsymbol{s}$. We find the solution $\boldsymbol{s}_{L1}$ to the following $L1$ regularization problem:

$$\boldsymbol{s}_{L1} = \min_{\boldsymbol{s}} \left\|AL^{-1}\boldsymbol{s} - \boldsymbol{b}\right\|_2^2 + \lambda\|\boldsymbol{s}\|_1$$

Then, we compute $\boldsymbol{x}_{L1} = L^{-1}\boldsymbol{s}_{L1}$. The value $\boldsymbol{s}_{L1}$ is sparse, and $\boldsymbol{x}_{L1}$ is recovered from this.

3. Motivation.

The following information is from *Discrete Inverse Problems* [1], which provides motivation as to why binary signal processing and recovery is significant.

Barcodes. In essence, a barcode is a series of alternating black and white strips, where the relative widths encode information. The most common use is on items sold in stores, in which the high-contrast images encode a string of letters and/or numbers that identifies the item. To read printed barcodes, a barcode reader (or scanner) is used, consisting of "a light source, a lens, and a photo conductor translating optical impulses into electrical ones."

To model the barcode reading process mathematically, a piecewise constant function $f(t)$ is used, representing the intensity of the barcode along a line perpendicular to the black and white bars.

Imperfections in the scanner (a mass-produced device) mean that we are unable to record the exact signal $f(t)$ but rather a blurred version $g(s)$. In our model, we assume that $f(t)$ and $g(s)$ are related by

$$\int_0^1 \exp\left\{\left(-\frac{(s-t)^2}{\zeta^2}\right)\right\} f(t)\, dt = g(s) \qquad 0 \leq s \leq 1$$

where the s axis represents the "one-dimensional photo conductor inside the scanner."

While this is not the correct model of blurring inside a barcode reader, it serves as a good approximation. This expression is a special instance of the Fredholm integral equation, with the kernel given by

$$K(s,t) = \exp\left\{\left(-\frac{(s-t)^2}{\zeta^2}\right)\right\} \qquad s,t \in [0,1]$$

The parameter ζ controls the amount of blurring (the larger the ζ the narrower the Gaussian peak and thus less blurring). The arbitrary scaling is omitted in the definition of $K(s,t)$.

Convolution. Kernels that depend only on the difference $s-t$ between the two independent variables, are called convolution kernels. The measured signal $g(s)$ is computed as the convolution

$$\int_0^1 h(s-t)\, f(t)\, dt = g(s) \qquad 0 \leq s \leq 1$$

between the exact signal $f(t)$ and the point spread function $h(t)$, in this case given by

$$h(t) = \exp\left\{\left(-\frac{t^2}{\zeta^2}\right)\right\} \qquad t \in \mathbb{R}$$

Point Spread & Deconvolution. Typically, point spread functions peak at $t = 0$ and decay away from the peak such that $h(t)$ is practically located near the peak. The point spread function is never exactly equal to zero, though for $\zeta = 0.01$ it satisfies $h(t) < 10^{-4}$ for $|t| > 0.03$, so we may consider it to be zero here.

The process of computing the function $f(t)$ given the measured signal $g(s)$ and the point spread function $h(t)$ is called deconvolution.

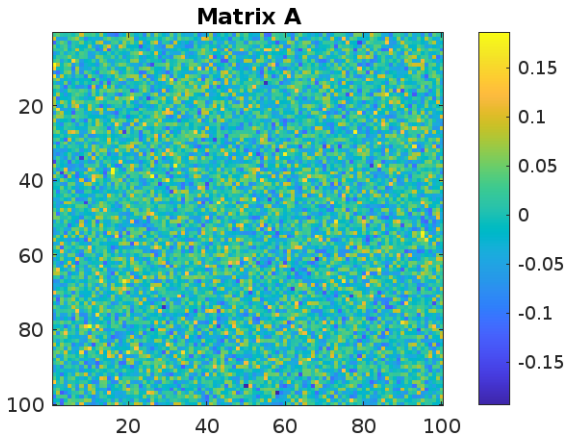
4. Methodology. Let's say we are given a "random" matrix $A \in \mathbb{R}^{n\times n}$, such that A is a matrix of normally distributed random numbers with mean $\mu = 0$ and standard deviation $\sigma = 1$. We may accomplish this using the following MATLAB code:

```
N = 100;
rng('Default');
A = randn(N, N);
A = A ./ norm(A, 2);
```

To investigate the various regularization methods, including *Truncated SVD*, *Tikhonov / Ridge (L2)*, and *Lasso (L1) Regularization*, we start by computing the singular value decomposition of A, alongside the condition number, using built-in MATLAB functions. From this we aim to visualize the matrix A to gain and understanding of its characteristics.

```
[U, S, V] = svd(A);
disp(cond(A));

figure;
imagesc(A);
colorbar;
```

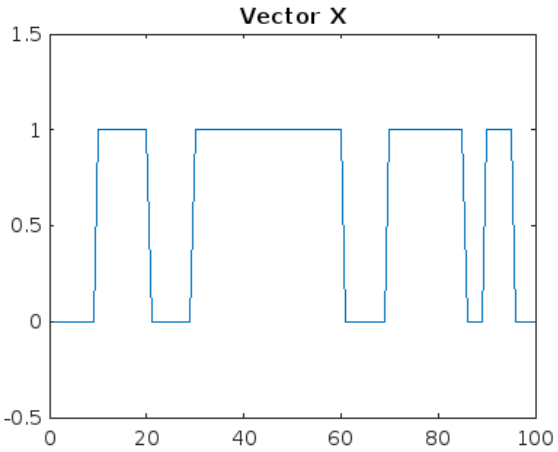
This results in an image similar to the following:



Now, we may generate a binary signal (represented by a vector $\boldsymbol{x}$) consisting of only 0s and 1s. The vector $\boldsymbol{x}$ is piecewise constant.

```
x = zeros(N, 1);
x(10:20) = 1;
x(30:60) = 1;
x(70:85) = 1;
x(90:95) = 1;

figure;
plot(x);

title('Data');

ylim([-0.5 1.5]);
```

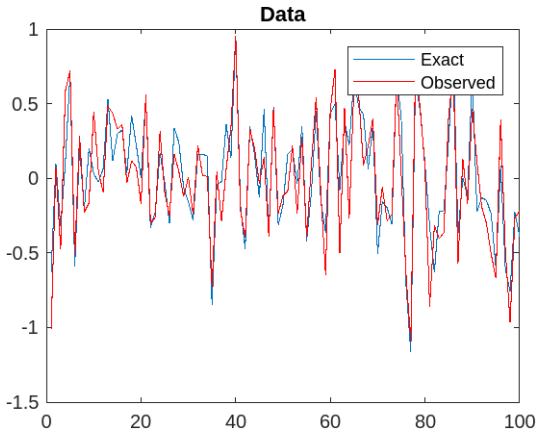
In visualizing the binary signal $\boldsymbol{x}$, the following is the result:



Now, we may use the forward computation to create the data vector $\boldsymbol{b}$, according to $\boldsymbol{b} = A\boldsymbol{x} + \boldsymbol{\varepsilon}$, where $\boldsymbol{\varepsilon}$ is Gaussian white noise with standard deviation $\eta = 0.2$. In this paper, we will only consider the use of additive white noise.

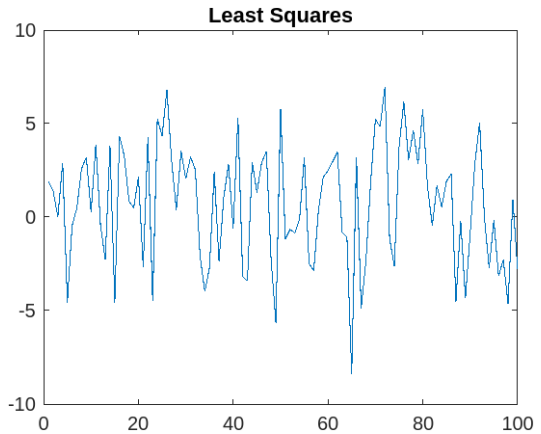
```
eta = 0.2; epsilon = eta * randn(N, 1);

b_exact = A * x;
b = b_exact + epsilon;

figure;
plot(b_exact);

title('Result');

hold on;
    plot(b, 'r');
hold off;

legend('Exact', 'Observed')
```

The following output image highlights that the noise component does not significantly contribute to the variation in the data vector $\boldsymbol{b}$, as the "random matrix" A drastically changes the signal. Regardless, a small amount of noise results in an ill-posed problem.



The typical approach to solving a problem of this type would be to compute the least squares solution, which typically leads to the appropriate results when there is no noise. To compute the least squares solution, we can use MATLAB built-in functions, displaying the results.

```
x_ls = A \ b;
% x_ls = inv(A' * A) * A' * b;

figure;
plot(x_ls);

title('Least Squares');
```

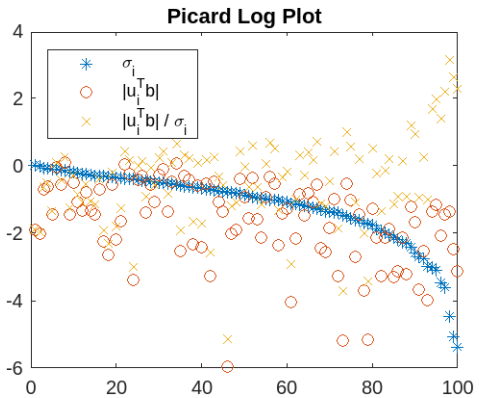


Evidently, this is *not* close to the true solution $\boldsymbol{x}$, which is directly a result of the noise introduced in the forward computation. It is reasonable to expect this kind of result, as the matrix A is ill-conditioned, as indicated by the condition number. (This value varies, depending on the random matrix A, though it is consistently greater than 200.) The matrix

A is nearly singular, thus, the effect of noise/error is amplified when solving the inverse problem. As such, we aim to use a regularization technique that allows us to approximate the binary signal.

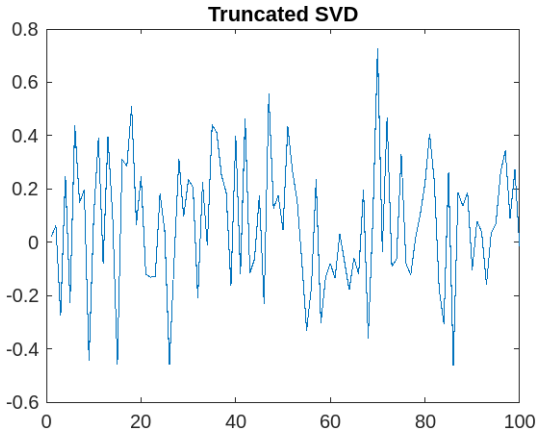
Truncated SVD. The *Picard* plot demonstrates the appropriate number of singular values to choose when using the truncated SVD regularization method.

```
figure;
plot(log(diag(S)), '*');

title('Picard Log Plot');

hold on;
    plot(log(abs(U' * b)), 'o');
    plot(log(abs(U' * b) ./ diag(S)), 'x');
hold off;

legend('\sigma_i', '|u_i^Tb|','|u_i^Tb| / \sigma_i', 'location', '
   northwest');
```



From the *Picard* plot, it appears as though 20 singular values is the absolute maximum we should use in the truncated SVD regularization method. Regardless, the plot indicates that perhaps the SVD method will not be effective in recovering the binary signal. Applying the algorithm, we arrive at the following solution.

```
max_sigma = 20;
x_tr = zeros(size(x));

for i = 1:max_sigma
    x_tr = x_tr + (U(:,i).' * b) / S(i, i) * V(:,i);
end

figure;
plot(x_tr);

title('Truncated SVD');
```



Evidently, the truncated SVD regularization method does not provide a sufficient solution. This is the case, as the *Picard* plot shows a certain level of irregularity in the $\left|u_i^T b\right|$ terms compared to the singular values σ_i, which indicates that truncated SVD is unlikely to provide meaningful information.

Further, the plot does not demonstrate a "binary" characteristic, in that it appears to jump around, rather than remain at 0 or 1 for a period of time. This prior information (along with the *Picard* plot), tells us to progress to further regularization methods.

Tikhonov/Ridge (L2) Regularization. To use Tikhonov/Ridge (L2) regularization, we compute the L-curve, using the maximum curvature plot to find a near-optimal value for λ, the regularization parameter.

```
num_lambda = 1000;

pert_error = zeros(num_lambda, 1);
bias_error = zeros(num_lambda, 1);
lambda = logspace(-5, 2, num_lambda);

c_hat = zeros(num_lambda, 1);

for i = 1:num_lambda
    reconstruct = (A' * A + lambda(i)^2 * eye(N)) \ A' * b;
    pert_error(i) = norm(A * reconstruct - b, 2)^2;
    bias_error(i) = norm(reconstruct, 2)^2;

    z_lambda = (A' * A + lambda(i)^2 * eye(N)) \ A' * (A *
       reconstruct - b);
    bias_der = 4 / lambda(i) * reconstruct' * z_lambda;

    numerator = lambda(i)^2 * bias_der * pert_error(i) + 2 *
       lambda(i) * ...
        bias_error(i) * pert_error(i) + lambda(i)^4 * bias_error(i
           ) * bias_der;
    denominator = (lambda(i)^2 * bias_error(i)^2 + pert_error(i)
       ^2)^(3/2);
```

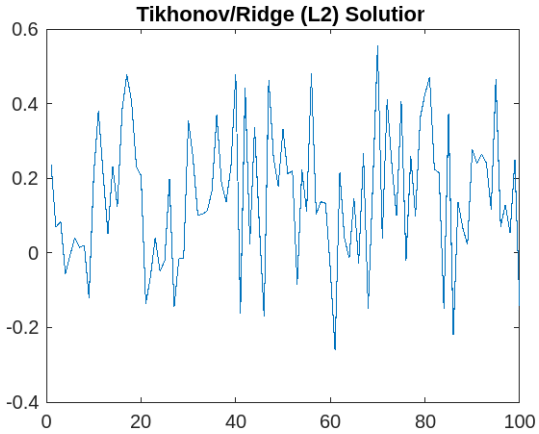
```
    c_hat(i) = 2 * bias_error(i) * pert_error(i) / bias_der *
       numerator / denominator;
end

figure;

subplot(1, 2, 1);
plot(1 / 2 * log10(pert_error), 1 / 2 * log10(bias_error));

title('L Curve');

xlabel('LOG Residual Norm');
ylabel('LOG Solution Norm');

subplot(1, 2, 2);
plot(log10(lambda), c_hat);

xlabel('LOG Lambda');
ylabel('Curvature');
```

Rather than view the plot and estimate the λ for which the curvature is at a maximum, we may use the following built-in MATLAB commands.

```
[argvalue, argmax] = max(c_hat);
lambda = lambda(argmax);

disp(lambda);
```

Now, with this information, we may compute the Tikhonov/Ridge (L2) solution.

```
x_L2 = (A' * A + lambda^2 * eye(N)) \ A' * b;

figure;
plot(x_L2);

title('Tikhonov (L2) Solution');
```

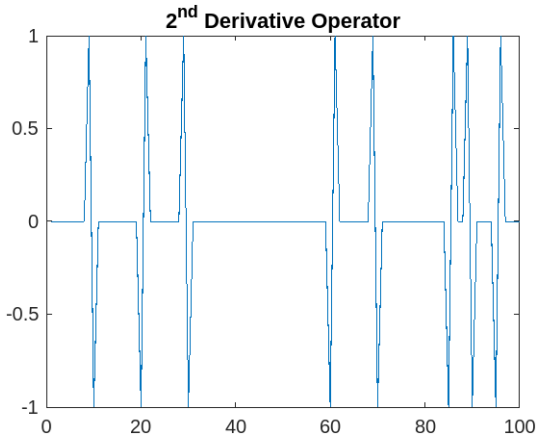


Evidently, the Tikhonov/Ridge (L2) regularization method does not provide a sufficient solution. As previously, the plot does not demonstrate a "binary" characteristic, in that it appears to jump around, rather than remain at 0 or 1 for a period of time. This prior information tells us to progress to further regularization methods.

Lasso (L1) Regularization. Now let us consider the Lasso (L1) regularization, which promotes sparsity over smallness. Perhaps this will give us a better indication of what the signal is supposed to look like.

Initially, we create the sparsifying transform matrix L using the following code. This matrix acts as a second derivative operator. To check it, we compute $L\boldsymbol{x}$ and plot the result.

```
L = -2 * eye(N) + diag(ones(N - 1, 1), 1) + diag(ones(N - 1, 1),
    -1);

value = L * x;

figure;
plot(value);

title("2^{nd} Derivative Operator");
```



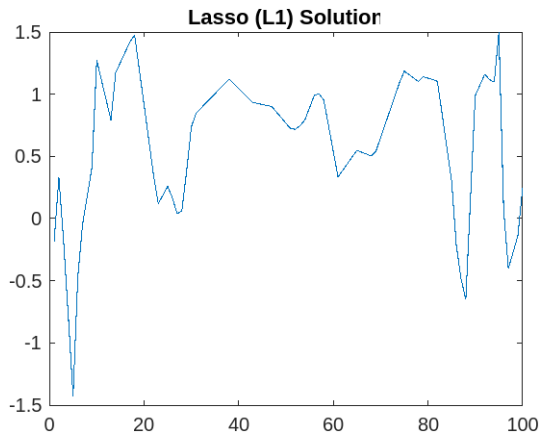
We find that the result (displayed in the plot) represents a vector that is sparse in nature. This is the case as the the matrix L acts as a 2^{nd} derivative operator and $\boldsymbol{x}$ is piecewise-constant. Thus, the 'spikes' displayed in the plot represent points where $\boldsymbol{x}$ is discontinuous (i.e. the 2^{nd} derivative is not equal to zero).

To compute the $L1$ solution, we may use the matrix L, along with the built-in *lasso* function used by MATLAB. Let $\boldsymbol{s}$ in $\mathbb{R}^n$ such that $\boldsymbol{x} = L^{-1}\boldsymbol{s}$. Using $\lambda = 0.0001$, find the solution $\boldsymbol{s}_{L1}$ to the following $L1$ regularization problem:

$$\boldsymbol{s}_{L1} = \min_{\boldsymbol{s}} \left\| AL^{-1}\boldsymbol{s} - b \right\|_2^2 + \lambda \|\boldsymbol{s}\|_1$$

Then, we compute $\boldsymbol{x}_{L1} = L^{-1}\boldsymbol{s}_{L1}$, using the 'lasso' function in MATLAB. The value $\boldsymbol{s}_{L1}$ is sparse, and $\boldsymbol{x}_{L1}$ is recovered from this.

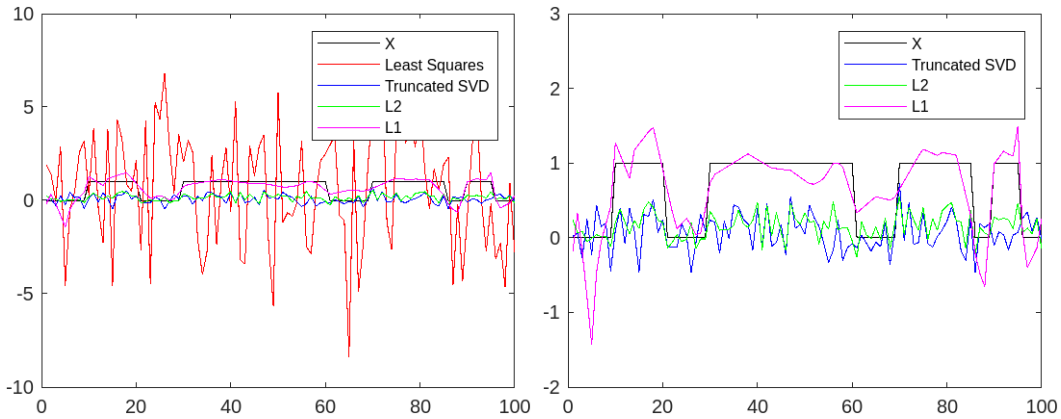
```
x_L1 = L \ lasso(A / L, b, 'Lambda', 0.0001);

figure;
plot(x_L1);

title('Lasso (L1) Solution');
```



The comparison between the various regularization techniques is provided below. The second plot eliminates the least squares solution, to allow for enhanced viewing. The code is provided as follows:

```
figure;

plot(x, 'k');

hold on;
    plot(x_ls, 'r');
    plot(x_tr, 'b');
    plot(x_L2, 'g');
    plot(x_L1, 'm');
hold off;
```

```
lgd = legend('X', 'Least Squares', 'Truncated SVD', 'L2', 'L1');
fontsize(lgd, 8, 'Points');

figure;

plot(x, 'k');

hold on;
    plot(x_tr, 'b');
    plot(x_L2, 'g');
    plot(x_L1, 'm');
hold off;

ylim([-2 3]);

lgd = legend('X', 'Truncated SVD', 'L2', 'L1');
fontsize(lgd, 8, 'Points');
```



Thus, we may determine that in the case where the matrix A is composed of random values, it is rather difficult to recover the original binary signal precisely. Given a relatively low level of noise, the truncated SVD and Tikhonov/Ridge (L2) regularization techniques are not effective. Lasso (L1) regularization seems to be a slight improvement, though perhaps choosing different values of the regularization parameter would lead to better results.

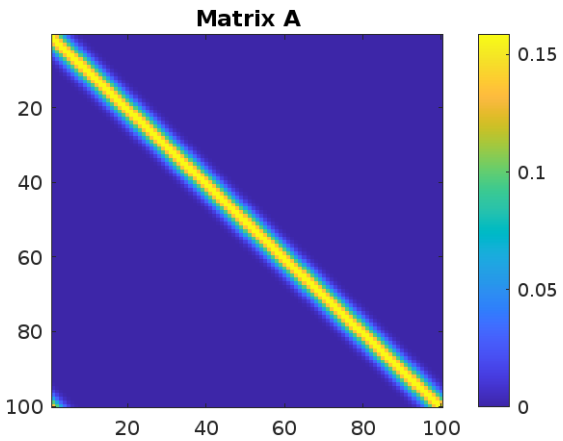
5. Results.

Gaussian Blur. Now, we transition to a more realistic application of this method, in which we do not consider a forward computation involving a "random" matrix A. Instead, we consider a matrix A that represents a 1D convolution (Gaussian blur) that serves to emulate real world scenarios.

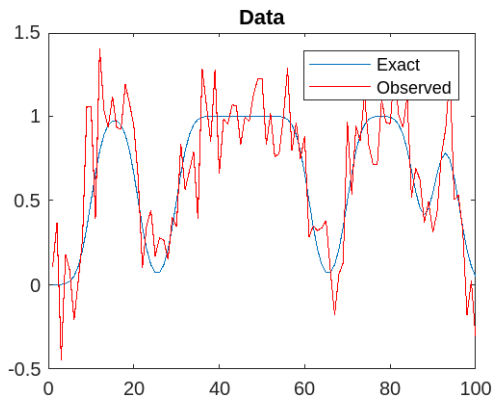
Let us consider a matrix $A \in \mathbb{R}^{n \times n}$ such that A is a matrix that represents a Gaussian blur. The value α is a factor inversely proportional to standard deviation of the Gaussian (normal distribution) used for blur. We normalize A at the end to ensure that the data is not scaled by an unknown factor.

```
N = 100;
alpha = 20;

blur_vector = fftshift(gausswin(N, alpha));
A = zeros(N);

for i = 1:N
    A(i,:) = circshift(blur_vector, i - 1, 1); % Build columns of
        'A'.
    if i < N / 2
        A(i, N / 2 + i : N) = 0;
    elseif i > N / 2
        A(i, 1 : N / 2 - i) = 0;
    end
end

A = A ./ norm(A, 2);
```

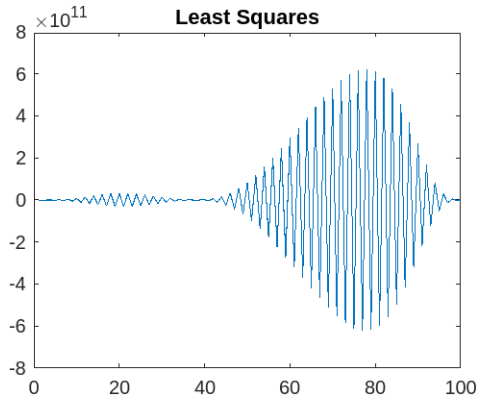
From this point, we apply the same methodology as before, with exception of a few changes. As indicated, we switch from using a "random" matrix to a Gaussian blur, as follows:



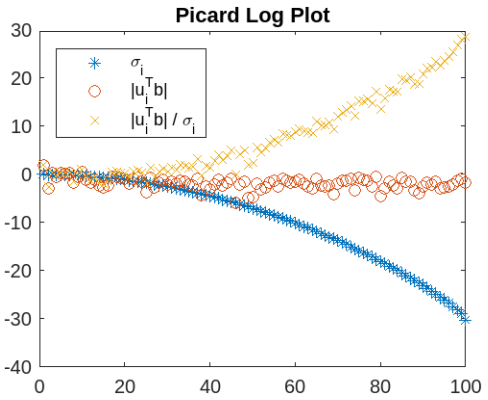
This means that the data vector $\boldsymbol{b}$ resembles the true vector $\boldsymbol{x}$ more closely.



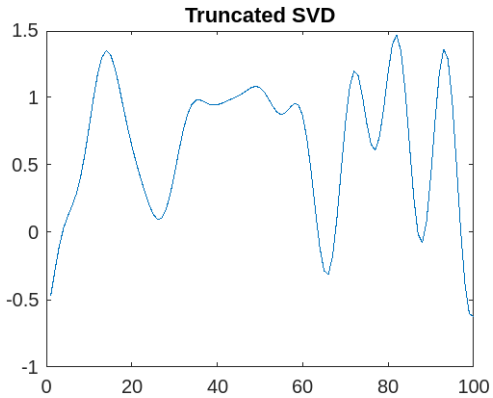
As a result of the nearly singular matrix A, the impact of noise on the least squares solution is more pronounced.



Further, when determining the maximum number of singular values to include in the analysis for the truncated SVD regularization method, the *Picard* plot satisfies the discrete *Picard* condition. In MATLAB, this is the only section of the script that requires hard-coding of values, as it is not possible to determine the best number of singular values to include based on a computer algorithm.

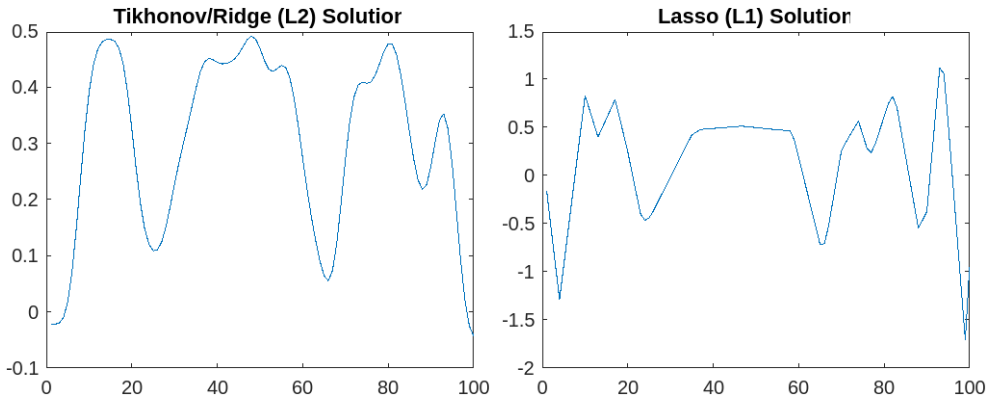


When using the truncated SVD regularization technique, the price we pay for a reduction in the variance of the signal is bias. The question that naturally follows is how accurate are we able to get to the original signal without knowing it. Given the prior information that the signal is binary, we are able to filter our results further than the truncated SVD, which will be discussed later.

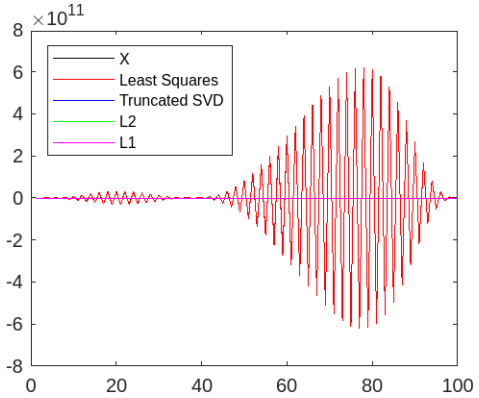


The Tikhonov/Ridge (L2) and Lasso (L1) regularization technique plots are shown as follows. Typically, the results more accurately match the binary signal, due to the construction of the matrix A.

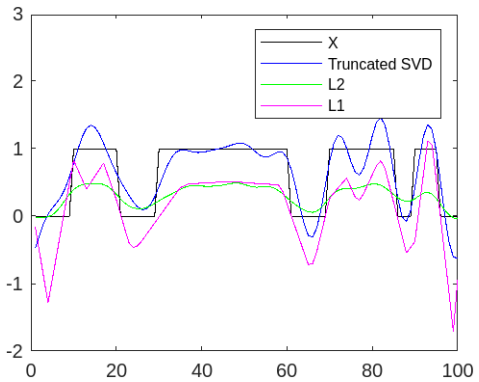
The regularization parameter λ for the Tikhonov/Ridge (L2) regularization is computed using the L-curve as before. The maximum curvature plot is used to find a near-optimal value for λ. In this case, it seems to result in over-regularization, though we choose not to deviate from the standard practice.



In the final plots displaying the results, we do not include the least squares solution, as it does not match the scale of the rest of the plots. For reference, this is what it would look like.



This is directly the result of the condition of the convolution matrix. The noise/error in the observed signal leads to a wildly inaccurate least squares solution, motivating the use of the truncated SVD as a viable option for recovering the signal. The near-singular nature of the matrix A drastically influences the way that the noise/error impacts the results. The following shows the results presented appropriately:



Point Spread Function. Now, we consider the point spread function, which is similar to a Gaussian blur, yet allows for manual implementation that gives us greater control over the variance than simply using the α value.

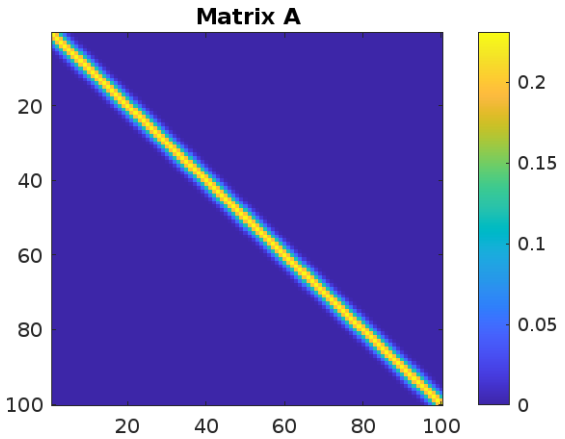
As before, we do not consider a forward computation involving a "random" matrix A, instead considering a matrix A that represents a 1D convolution (by a point spread function) that serves to emulate real world scenarios.

Let us consider a matrix $A \in \mathbb{R}^{n \times n}$ such that A is a matrix that represents a point spread function. We normalize A at the end to ensure that the data is not scaled by an unknown factor.

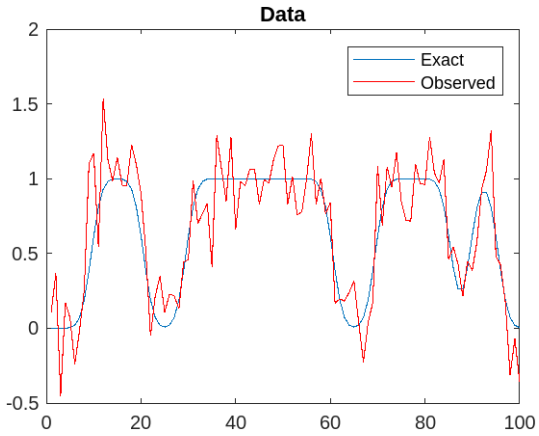
```
N = 100;
A = zeros(N, N);
sd = 3;

for i = 1:N
    for j = 1:N
        for k = 0:N
            if abs(i - j) == k
                A(i, j) = exp(-(k^2) / (2 * sd));
            end
        end
    end
end

A = A ./ norm(A, 2);
```

As perhaps implied, the point spread function follows a Gaussian distribution, where the only variable we aim to modify is the standard deviation. If we increase the standard deviation, we take into account "pixels" (values on a 1D line) that are farther away from the source. This results in a different convolution of the binary signal.

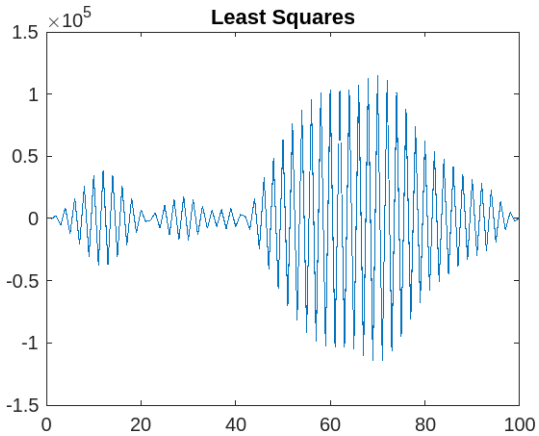
While the results are nearly identical to the Gaussian blur, the set-up of the matrix A grants us greater control of the characteristics of the convolution. We apply the same methodology as before, with exception of a few changes. As indicated, we switch from using a "random" matrix to a convolution matrix, as follows:



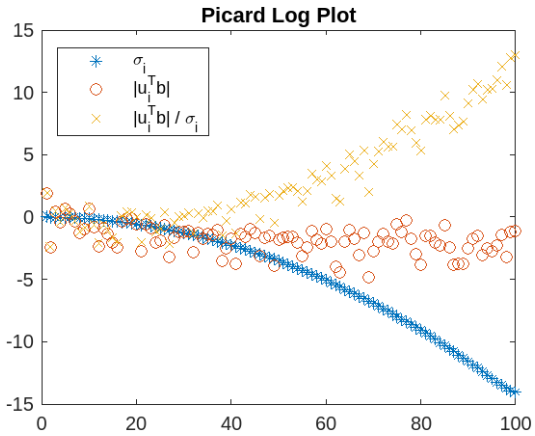
This means that the data vector $\boldsymbol{b}$ resembles the true vector $\boldsymbol{x}$ more closely.



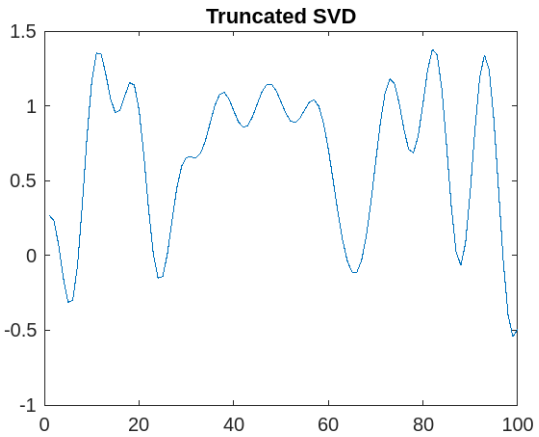
As a result of the nearly singular matrix A, the impact of noise on the least squares solution is more pronounced.



Further, when determining the maximum number of singular values to include in the analysis for the truncated SVD regularization method, the *Picard* plot satisfies the distrete *Picard* condition. In MATLAB, this is the only section of the script that requires hard-coding of values, as it is not possible to determine the best number of singular values to include based on a computer algorithm.

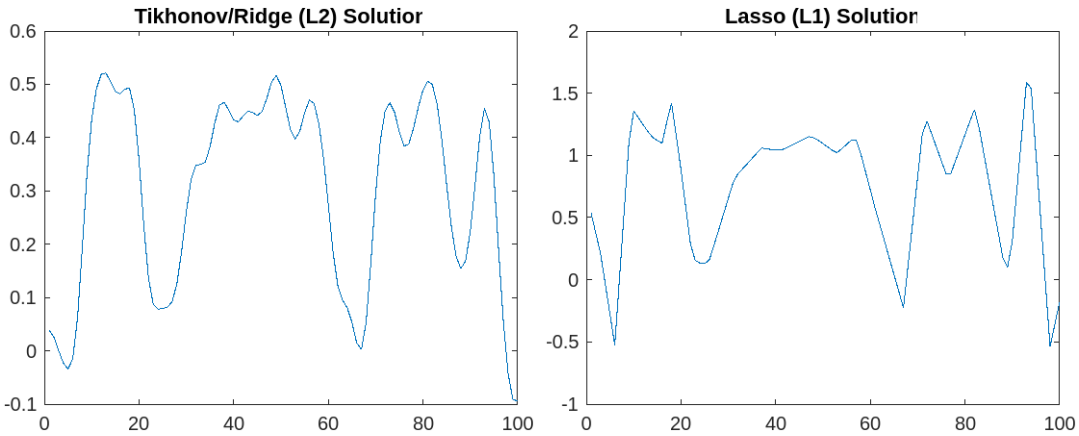


When using the truncated SVD regularization technique, the price we pay for a reduction in the variance of the signal is bias. The question that naturally follows is how accurate are we able to get to the original signal without knowing it. Given the prior information that the signal is binary, we are able to filter our results further than the truncated SVD, which will be discussed later.



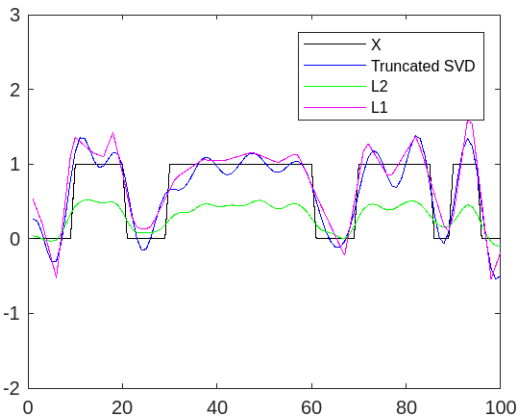
The Tikhonov/Ridge (L2) and Lasso (L1) regularization technique plots are shown as follows. Typically, the results more accurately match the binary signal, due to the construction of the matrix A.

The regularization parameter λ for the Tikhonov/Ridge (L2) regularization is computed using the L-curve as before. The maximum curvature plot is used to find a near-optimal value for λ. In this case, it seems to result in over-regularization, though we choose not to deviate from the standard practice.



In the final plots displaying the results, we do not include the least squares solution, as it does not match the scale of the rest of the plots.

This is directly the result of the condition of the convolution matrix. The noise/error in the observed signal leads to a wildly inaccurate least squares solution, motivating the use of the truncated SVD as a viable option for recovering the signal. The near-singular nature of the matrix A drastically influences the way that the noise/error impacts the results. The following shows the results presented appropriately:



Further Data Processing. To further process the data, and attempt to recover the binary signal, we may map the curves resulting from the regularization to 0 or 1 depending on a critical value (chosen to be 0.5, the midpoint).

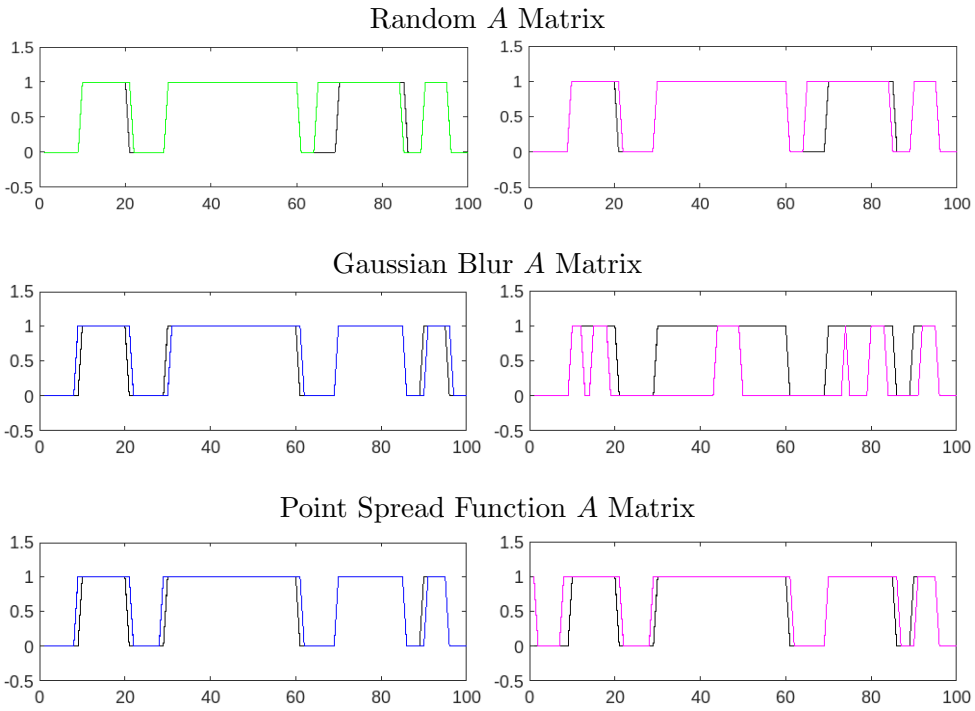
This means that for values greater than the critical value, we presume that the binary signal is a 1 at that location, whereas for values less than the critical value, we presume that the binary signal is a 0 at that location. The code to create these mappings is given as follows:

```
x_tr_final = zeros(N, 1);
x_tr_final(1:N) = x_tr(1:N);
x_tr_final(x_tr_final < 0.5) = 0;
x_tr_final(x_tr_final > 0.5) = 1;

x_L2_final = zeros(N, 1);
x_L2_final(1:N) = x_L1(1:N);
x_L2_final(x_L2_final < 0.5) = 0;
x_L2_final(x_L2_final > 0.5) = 1;

x_L1_final = zeros(N, 1);
x_L1_final(1:N) = x_L1(1:N);
x_L1_final(x_L1_final < 0.5) = 0;
x_L1_final(x_L1_final > 0.5) = 1;
```

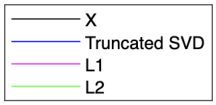
Further, the code to display the results is given as follows:

```
figure;
tiledlayout(2, 1);

nexttile;
plot(x, 'k');
hold on;
    plot(x_NAME_final, 'COLOR');
hold off;

legend('X', 'NAME');
ylim([-0.5 1.5]);

nexttile;
plot(x, 'k');
hold on;
    plot(x_NAME_final, 'COLOR');
hold off;

legend('X', 'NAME');
ylim([-0.5 1.5]);
```

The results/output are shown for each iteration, in succession.



The legend for the above plots is given as follows:



As demonstrated, when using this "mapping" to further process the data in an attempt to recover the binary signal, the $L1$ and $L2$ curves tend to produce similar results. In the case where a "random" A matrix is used, the truncated SVD is not useful, so we simply include the Tikhonov/Ridge (L2) and Lasso (L1) regularization methods.

In the case where a Gaussian blur matrix or point spread function is responsible for the matrix A, resulting in the convolution of the binary signal, the best solution seems dependent on the case. Lasso (L1) regularization tends to produce more errors than the truncated SVD method does, though it is clear that neither is able to reconstruct the signal with perfect accuracy.

Thus, this leads us to consider the significance of performing multiple iterations of various regularization methods to try to recreate the binary signal. Given the computational time and resources, an "averaging" effect of the different reconstructions may prove to be most effective in accurately recovering the binary signal.

6. Conclusions.

Final Statement. The aim of binary signal recovery is to reconstruct accurate digital information from a distorted or noisy transmission. By employing various algorithms and techniques, including truncated SVD, Tikhonov/Ridge (L2) regularization, and Lasso (L1) regularization, we are able to restore a 1D binary signal, thereby mitigating the effects of noise and distortion.

This paper explores and discusses the failures and successes of these regularization techniques, further proposing additional data processing on the basis of prior information, namely that the signal is binary. By effectively recovering the binary signal, reliable communication and accurate data interpretation may be achieved, particularly in the context of solving the problem of reading barcodes.

Extensions. While the regularization methods/techniques presented in this paper aim to recover a convoluted 1D binary signal with noise/error, this further applies to 2D signals. The 2D equivalent to a barcode is a QR code, so further exploration may involve an analysis of regularization techniques in these images.

A potential challenge with QR codes is that you must take into account the angle at which the QR code is being observed, along with other characteristics. Typically, a barcode scanning is done perpendicular to the image, though this is certainly not the case for QR codes. *Discrete Inverse Problems* [1] provides motivation as to why 2D signal processing and recovery (image deblurring) is significant.

In order to create a sharper reconstruction of a digital image from a noisy one, we may use mathematical techniques based on the model of the blurring process. This is similar to the barcode reading problem, in that a clean signal (or sharp image) is "deteriorated by blurring in the optical system."

Rather than a one-dimensional signal, a two-dimensional signal (and point spread function) is used for the sharp and blurred images.

Let us consider the continuous setting, in which image deblurring is a first-kind Fredholm integral equation of the generic form

$$\int_0^1 \int_0^1 K(\boldsymbol{s}, \boldsymbol{t})\, f(\boldsymbol{t})\, dt_1\, dt_2 \qquad \boldsymbol{s} \in [0,1] \times [0,1]$$

The functions $f(\boldsymbol{t})$ and $g(\boldsymbol{s})$ represent the sharp and blurred images, and are functions of two spatial variables $\boldsymbol{s} = (s_1, s_2)$ and $\boldsymbol{t} = (t_1, t_2)$. As with the 1D setting, discretization of the problem (by means of the midpoint quadrature rule) is necessary to transform the continuous version to a discrete image deblurring problem. This allows for regularization techniques that are applicable in a variety of applications involving digital images.

MATLAB Code. The following MATLAB code files, which are relevant for this project, are provided in the project structure.

Model_1.m, *Model_2.m*, *Model_3.m*, *Plots.m*

Acknowledgments. I would like to acknowledge and thank Professor Dylan Green* for assistance in creating this manuscript.

REFERENCES

[1] P. C. Hansen, *Discrete Inverse Problems*, SIAM, 1st ed., 2010.

* *Dartmouth College*