LSQ Regularization

Ethan Berk, Dylan Sanders, Matthew Younce

November 2024

1 Introduction

The general idea of Least Squares is to find a best-fitting line given a set of data points which accurately represents any trends present. This line models the data, and gives us a basis for explaining the relationship between different variables or predicting reasonable values based off some input variable. This technique is incredibly useful and widely used, especially in the field of data science and statistical learning.

To understand the process and underlying algorithm of Least Squares we will start with simple linear regression, then see how we can extend to a higher dimension problems and eventually describe how to improve this function approximation. Our goal is to create a line of best fit given data $\{(x_j), f(x_j)\}$, or in other words we want to regress y on x and find a line close to these data points to form $f(x) = c_0 + c_1 x$, where c_j are unknown coefficients we must solve for. We achieve this closeness by minimizing the Least Squares Criterion. If we define $\hat{y}_i = \hat{c}_0 + \hat{c}_1 x_i$ as our prediction based on some value x_i and $E = y_i - \hat{y}_i$ as the difference between the ith predicted response (given by the model) and the observed response. If, for every response we square the residual values and sum them together we can create what is known as the *Residual Sum of Squares* or *Sum of Squared Errors*. This is a common measure of error as well as our objective function that we aim to minimize by choosing appropriate coefficient values. If we want to extend this to a multiple liner regression model, we add additional predictor variables $x_0, x_1, ..., x_p$ in a function we can call $g_j(x)$, the prediction is now modeled by $f(x) = c_0 + c_1 x + c_2 x_2 + ... + c_p x_p$. Creating a system that resembles the following, where p is the number of predictors and n is the number of data observations.

$$\begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} 1 & x_{11} & x_{12} & \cdots & x_{1p} \\ 1 & x_{21} & x_{22} & \cdots & x_{2p} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n1} & x_{n2} & \cdots & x_{np} \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_p \end{bmatrix}$$

We can solve for the unknown coefficient vector $\mathbf{c}$ in the model $M(x) = \sum_{j=1}^{n} c_j g_j(x)$ with the constraint that the sum of squared errors between the data y_i at each node x_i and the linear model $g(x)$ is minimized. We evaluate the following.

$$argmin_c \sum_{i=1}^{n} (y_i - \sum_{j=1}^{n} c_j g_j(x))^2$$

Often times the system is overdetermined, and consequently we cannot uniquely solve it. In such instances, we can form a matrix $\mathbf{M}$, which consists of the $g_j(x_i)$ entries, and we can equivalently solve for the vector c that minimizes

$$q(\vec{c}) = \|Mc - y\|_2^2$$

Through calculus techniques, we will compute the gradient of q and set it equal to zero to obtain the following.

$$\begin{aligned} \nabla q = 2M^T Mc - 2M^T y = 0 \\ M^T Mc = M^T y \end{aligned}$$

We call this matrix system the Normal Equations, and when we solve for $\mathbf{c}$ we come to the conclusion

$$c = (M^T M)^{-1} M^T y$$

This evaluation gives us our coefficient values and the line of best fit as long as the matrix $\mathbf{M}$ is full rank, meaning all the columns are linear independent, this ensures $(M^T M)^{-1}$ is invertible and therefore we can solve for $\mathbf{c}$. And, luckily for our convenience solving this will give us the coefficient values for a single and multiple predictor problem.

A natural concern that follows is to wonder how accurate the resulting model is. A common practice among data scientists is to split their entire set of data into a training and a testing set, especially in the process of creating a predictive model. They use the the training data to fit the model and verify the results with data the model has not yet seen. From the training data they obtain $f(x)$ that predicts values based on given values of x, if these values are close to their true values, then then error is considered small. A way to measure this is with the mean squared error (MSE). This is calculated by averaging the sum of squared errors. If this value is relatively small, that indicates an accurate model, that preforms well on new data. The practice of splitting the data into training and testing sets can help determine if a model over or under-fits the data. If too much the data is used to create the model, then the model might be too specific to that particular data set, and therefore be classified as over-fitting the data. On the other hand, if not enough data is used to create the model, then it is at risk for under-fitting the data, and not being able to predict responses from unseen data accurately. This underlying risk of under-fitting or over-fitting has to do with what we call the bias-variance trade off.

This is a phenomena that all data scientists have to take in consideration. A model is considered over-fit when it is too flexible, which is an indicator that there is high variance, which has to do with the model being highly specified to certain data. For example, if a model is created by a line that passes through every data point, it is incredible flexible, but it will not preform well on different data points when trying to predict new values. Alternatively, bias is an error associated with assuming real world implications for the model, like assuming a relationship between two variables is linear and creating a model based on this assumption when, in fact, the relationship between the variables is not be perfectly so. The trade off is that when one of these errors increases the other decreases. It is a goal among data scientists to find a sweet spot between the two. Often times this can be addressed through choosing a good fraction of data for the training and testing sets.

A more sufficient way to account for the trade-off is through *regularizing* the coefficients. We will explore two regularized regression techniques, Ridge Regression and Tikhonov Regression, both aim to address the risk of over-fitting as well as suppress other noises and irregularities encountered when building a regression model. They follow very similar processes as Least Squares with the exception of new objective functions for added improvement. We will walk through these mathematical derivations to understand the how they extend from the ordinary Least Squares approximation as well as describe their advantages through numerical experiments. Moreover, we will broaden these mathematical ideas to other regression types, LASSO and Elastic Net Regression, also demonstrating their affects through numerical examples.

2 Mathematical background and derivations

2.1 Ridge Derivation

Ridge regression adds the penalty term $\lambda||c||_2^2$ to the objective function. This is to penalize the model for choosing coefficients that far from zero. This is beneficial because adding the penalty term helps find the sweet spot in the bias variance trade off. We get to choose lambda greater than zero so that we can find a model that try to minimize both bias and variance. Adding the penalty reduces the variance in the model while only marginally increasing bias. With the penalty term the objective function is now defined as:

$$\begin{aligned} q(\vec{c}) &= \|Mc - y\|_2^2 + \lambda||c||_2^2 \\ &= (Mc - y)^T(Mc - y) + \lambda(c)^T(c) \\ &= c^T M^T Mc - 2c^T M^T y + y^T y + \lambda c^T c \\ &= c^T(M^T M + \lambda I)c - 2c^T M^T y = y^T y \end{aligned}$$

Like the regular least squares objective function, the ridge function is also quadratic. It is in the form:

$$q(\vec{c}) = c^T \mathbf{G}_{ridge} c + c^T \vec{b} + d$$

Where $\mathbf{G}_{ridge} = M^T M + \lambda I$, $\vec{b} = -2M^T y$, and $d = y^T y$

To find the solution for the coefficients we need to take the derivative of this function, set it equal to zero and solve for c.

$$\begin{aligned}
\nabla q &= 2(M^T M + \lambda I)c - 2M^T y \\
\nabla q &= 2(M^T M + \lambda I)c - 2M^T y = 0 \\
2M^T y &= 2(M^T M + \lambda I)c \\
M^T y &= (M^T M + I\lambda)c \\
c &= (M^T M + I\lambda)^{-1} M^T y
\end{aligned}$$

This is the solution to for the coefficients of the ridge model. Looking at the equation there is an inverted matrix which can cause problems if the matrix is ill conditioned, making the coefficients that we find likely inaccurate. Analysis of $\mathbf{G}_{ridge}$ will allow us to find when our problem is ill conditioned.

To explore the conditioning of this matrix, say we are given the eigenvalues and eigenvectors of the matrix $M^T M$ Then we can write $\mathbf{G}_{ridge}$ using the eigenvalue decomposition as:

$\mathbf{G}_{ridge} = VDV^T + \lambda I$

V is a orthonormal matrix of eigenvectors. All eigenvectors are orthogonal because $\mathbf{G}_{ridge}$ is symmetric.

D is a diagonal matrix containing all of the eigenvalues of $\mathbf{G}_{ridge}$.

All eigenvalues are greater than zero because $\mathbf{G}_{ridge}$ is positive definite.

we can use the properties of orthonormal matrices to simplify this expression.

$$\begin{aligned}
\mathbf{G}_{ridge} &= VDV^T + \lambda I \\
&= VDV^T + V\lambda I V^T \\
&= V(D + \lambda I)V^T
\end{aligned}$$

Here is the eigenvalue decomposition of $\mathbf{G}_{ridge}$ As we can see from the equation above the penalty term in our objective function increased the all of the eigenvalues of the inverted matrix by λ. This is important for calculating the condition number of the matrix because the condition number is:

$$k = \frac{\sigma_1}{\sigma_n}$$

Where σ are the singular values of the matrix. $\mathbf{G}_{ridge}$ is a symmetric positive definite matrix, meaning the singular values are the eigenvalues, the condition number of the $\mathbf{G}_{ridge}$ is:

$$k(\mathbf{G}_{ridge}) = \frac{\mu_1 + \lambda}{\mu_n + \lambda}$$

Where μ_1 is the largest eigenvalue of $M^T M$ and μ_n is the smallest eigenvalue of $M^T M$. If $M^T M$ is ill-conditioned the smallest eigenvalue μ_n is near zero. Therefore the addition of the the penalty term makes certain that the lowest singular value is not arbitrarily close to zero making $\mathbf{G}_{ridge}$ better conditioned than $\mathbf{G}_{OLS}$. The relationship that the penalty term has on the eigenvalues add another interesting benefit that ridge regression has over ordinary least squares. If $M^T M$ is singular there is no solution that will solve for the coefficients in the model using the least squares solution. If we were to use ridge regression instead of least squares in this case, the penalty term ensures that $\mathbf{G}_{ridge}$ never has zero eigenvalues and is therefore non singular. Meaning when using ridge regression you can always find a solution the ridge coefficients.

2.2 Tikhonov Derivation

To give us more control over the penalty term, we can generalize Ridge regression to:

$$\arg\min_c \|Mc - y\|_2^2 + \lambda\|Ic\|_2^2$$

Then we can incorporate a general matrix Γ instead of the identity matrix resulting in the form:

$$\arg\min_c \|Mc - y\|_2^2 + \lambda\|\Gamma c\|_2^2$$

Where Γ becomes a penalty matrix that allows us to restrict our answer c in certain directions. The min can be found in a similar process to that of ridge regression as it is a quadratic defined as

$$\begin{aligned} q(\vec{c}) &= \|Mc - y\|_2^2 + \lambda||\Gamma c\|_2^2 \\ &= (Mc - y)^T(Mc - y) + \lambda(\Gamma c)^T(\Gamma c) \\ &= c^T M^T Mc - 2c^T M^T y + y^T y + \lambda c^T \Gamma^T \Gamma c \end{aligned}$$

We can then find its gradient and the root of its gradient. This defines our minimizing solution of the regularized $Mc = y$

$$\begin{aligned} \nabla q &= 2M^T Mc - 2M^T y + 2\lambda\Gamma^T\Gamma c \\ \nabla q &= 2M^T Mc - 2M^T y + 2\lambda\Gamma^T\Gamma c = 0 \\ 2M^T y &= 2M^T Mc + 2\lambda\Gamma^T\Gamma c \\ M^T y &= (M^T M + \lambda\Gamma^T\Gamma)c \\ c &= (M^T M + \lambda\Gamma^T\Gamma)^{-1}M^T y \end{aligned}$$

If constructed correctly, the penalty matrix will not only give us more control over the resulting function, but will also improve the conditioning of the matrix. Generally, it will add to the diagonal of an almost singular matrix to increase independence and pose a more conditioned problem.

This solution can still be ill conditioned especially with large data sets of 200 + data points like we plan to perform. So we can reform the problem to allow for a better conditioned inverse:

$$\begin{aligned} \arg\min_c \|Mc - y\|_2^2 + \lambda\|\Gamma c\|_2^2 &= \arg\min_c \left\| \left[Mc - y\right] \right\|_2^2 + \left\| \left[\sqrt{\lambda}\Gamma c\right] \right\|_2^2 \\ &= \arg\min_c \left\| \begin{bmatrix} Mc - y \\ \sqrt{\lambda}\Gamma c \end{bmatrix} \right\|_2^2 && \text{By properties of block matrix} \\ &= \arg\min_c \left\| \underbrace{\begin{bmatrix} M \\ \sqrt{\lambda}\Gamma \end{bmatrix}}_{M'} c - \underbrace{\begin{bmatrix} y \\ 0 \end{bmatrix}}_{y'} \right\|_2^2 \\ &= \arg\min_c \|M'c - y'\|_2^2 \end{aligned}$$

This now looks like a un-regularized least squares so using the same derivation we find the following solution:

$$(M'^T M')c = M'^T y'$$

Now we can use QR factorization of M' to simplify. Where $M' = QR$ and Q's columns form an orthonormal basis of M'. This makes Q an orthogonal matrix where $Q^T Q = I$. And R is transform upper right triangular matrix where $M' = QR$. We write

$$
\begin{aligned}
(M'^T M')c &= M'^T y' \\
(R^T Q^T Q R)c &= R^T Q^T y' \\
R^T R c &= R^T Q^T y' \\
Rc &= Q^T y' \\
c &= R^{-1} Q^T y'
\end{aligned}
$$

This poses a much better conditioned problem as R is an upper right triangular matrix with diagonal entries that are not close to 0. This makes it a better conditioned (more singular) inverse, making it easier to compute.

2.2.1 Penalty matrix Γ background

1. **Derivative matrices:** To help with over fitting we can enforce smoothness through a discrete first derivative matrix. Given the model we are trying to fit (for example if we are fitting trig function) $f(x) = c_0 + c_1\cos(x) + c_2\sin(x) + c_3\cos(2x) + \cdots + c_{2m-1}\cos(mx) + c_{2m}\sin(mx)$. Which we represent as $f(x) = Mc$. We can then similarly define $f'(x) = -c_1\sin(x) + c_2\cos(x) - 2c_3\sin(2x) + \cdots - mc_{2m-1}\sin(mx) + mc_{2m}\cos(mx)$ which can be represented as $f'(x) = Dc$ where

$$
D = \begin{bmatrix}
0 & -\sin(x_1) & \cos(x_1) & \cdots & -mc_{2m-1}\sin(mx_1) & mc_{2m}\cos(mx_1) \\
0 & -\sin(x_2) & \cos(x_2) & \cdots & -mc_{2m-1}\sin(mx_2) & mc_{2m}\cos(mx_2) \\
\vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\
0 & -\sin(x_n) & \cos(x_n) & \cdots & -mc_{2m-1}\sin(mx_n) & mc_{2m}\cos(mx_n)
\end{bmatrix}
$$

   We now use $\Gamma = D$. This enforces smoothness because the basis functions that make a function jagged have high derivatives, when compared to the smooth parts of the function. For instance in a polynomial basis, terms of high powers stop a function from being smooth. So by adding the derivative matrix as penalty matrix, it enforces penalties on function with high derivatives. This will smooth the function out as the solution c will be more in the vector direction of the smoother basis functions with smaller first derivatives.

   A similar processes can be used on other basis functions, for instance, a polynomial model $f(x) = c_0 + c_1x + c_2x^2 + \cdots + c_mx^m$ would have the discrete first derivative matrix

$$
D = \begin{bmatrix}
0 & c_2x_1 & 2c_2x_1 & \cdots & mc_mx_1^{m-1} \\
0 & c_2x_2 & 2c_2x_2 & \cdots & mc_mx_2^{m-1} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
0 & c_2x_n & 2c_2x_n & \cdots & mc_mx_n^{m-1}
\end{bmatrix}
$$

   The smoothing allows us to account for over fitting

2. **Low Pass Filtering**: For a trigonometric basis where the model $y = f(x)$ where $x \in (-\pi, \pi)$ for $f(x) = c_0 + c_1\cos(x) + c_2\sin(x) + c_3\cos(2x) + \cdots + c_{2m-1}\cos(mx) + c_{2m}\sin(mx)$. If we are trying to fit noisy data from a combination of trig functions, (often audio or periodic signals) a higher degree model will often over fit the data. Such that higher frequency functions will take over and essentially oscillate between data points versus actually modeling the true function. We enforce a penalty on the high frequency basis functions as we assume it to be fitting the noise. However, we will not penalize the low frequency function as we assume them to be fitting the true function.

   This can be done through a diagonal matrix D as it will directly multiply the coefficients Dc, where we will have higher diagonal elements $d_{(k,k)}$ and for coefficient a we want to penalize c_k. Given that the basis functions are in order of frequency, we can simply say for all $k \leq k_0$ (this would be all functions with frequency less than or equal to $\frac{k_0}{2\pi}$) that $d_{(k,k)} = 0$ (no penalty). And for all other $d_{(k,k)}$ where $i > k_0$ we have $d_{(i,i)} = k^c$ for some constant c. Which provides that for functions with a frequency $\frac{k}{2\pi}$ higher than $\frac{k_0}{2\pi}$ there will be a penalty proportional to a power of k.

3. **Eigenvalues of Gram Matrix**: Rather than blindly penalizing basis functions that we think are noise, we would like to penalize functions that we can prove are noise. Essentially, this means we penalize parts of the function that are accounting for small variance rather than large overall trends. We can find this by using the covariance matrix M^TM and a singular value decomposition. Where $M = U\Sigma V^T$ such that Σ is a diagonal matrix with values $\Sigma_{(i,i)} = \sigma_i$ where $\sigma_1 \geq \sigma_2 \geq \sigma_3 \geq \cdots \geq \sigma_m$. Each σ_i is a singular value of M where $\sigma_i = \sqrt{\lambda_j}$ for the ith largest non zero eigenvalue λ_j of the matrix M^TM (same non zero eigenvalues as MM^T). And, U is a unitary ($U^{-1} = U^T$) matrix with columns equal to the eigenvectors of MM^T. V is also unitary and has columns equal to the eigenvectors of M^TM. This allows us to rewrite $M^TM = V\Sigma U^T U\Sigma V^T = V\Sigma^2 V^T$, $\sigma_i = \sqrt{\lambda_j}$ $\Sigma^2 = \Lambda$ and its diagonal entries are the eigenvalues of M^TM in size order. Giving us the eigenvalue decomposition $M^TM = V\Lambda V^T$. Further more by properties of the covariance matrix M^TM we know its largest eigenvalue and corresponding eigenvector represent the largest variance trend in the data. So the smaller eigenvalues represent small variance or small inconsequential trend in the data, which we can assume to be noise. So we can assign different penalty in the direction of different eigenvectors based on their eigenvalue. We write

$$\lambda \sum d_j (v_j^T c)^2 = \lambda \|DV^T c\|_2^2$$

This will penalize the solution in the direction of the eigenvectors v_j based on a value d_j which we will get from λ_j. It is simplified to use the diagonal Matrix D with entries $D_{i,i} = d_i$. Now we can use d_i to specifically penalize in the direction of solutions that are based on small variance (noise) and not penalize directions of high variance (true function).

Given that the largest eigenvalue is λ_1 (highest variance), we can quantize if some λ_i does or does not need a penalty based it size compered to λ_1. Where if $\frac{\lambda_i}{\lambda_1} \approx 1$ then λ_i also has a relatively high variance so does not need to be penalized. But if $\frac{\lambda_i}{\lambda_1} \approx 0$ then the direction v_i has small variance and likely is noise so we penalize a lot. We can do this 2=two ways (1) $d_i = \frac{1}{\lambda_i}$, (2) set $d_j = 0$ if $\lambda_i > e\lambda_1$ for some chosen constant $c \in (0, 1)$, and set $d_j = 1$ if not.

3 Numerical Experiments

3.1 Numerical Experiments: Ridge Regression

Here, we will execute numerical experiments which expose the effect of adding a penalty term to the Least Squares problem.

3.1.1 Linear Model

To begin will create data by sampling from the linear function:

$$f(x) = 3x + 2 + \epsilon$$

where ϵ is an error term from a normal distribution with mean 0 and variance 1 to add noise to the data. We sampled 200 observations from this function and split them randomly into a training data set and a testing data set each with 100 observations. The training data set was used to find the coefficients while the testing set is used to find out how accurate the model is.

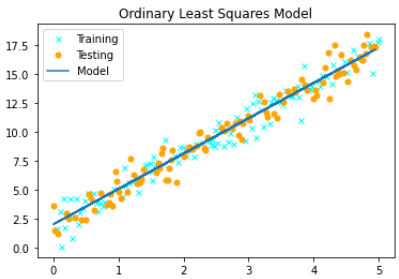


Figure 1: Ordinary least square approximation

Here we can see the linear ordinary least squares model matches the data very well. This is also seen in the sum of squared error (SSE) which this model achieved a value of 76.23. Meaning that on average each of the test points is within one unit of the models prediction.

Now we will explore how this compares to a ridge model.

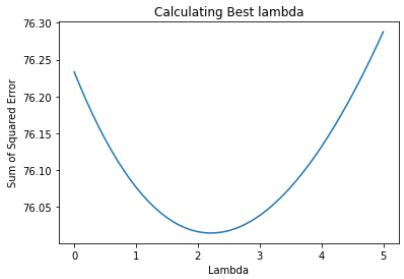


Figure 2: Cross validation of lambda

The value of lambda that minimize the SSE is for the ridge model is 2.2. This model achieved a SSE of 76.01 which is slightly better than what the ordinary least squares model was able to achieve, meaning that this ridge model was more accurate.

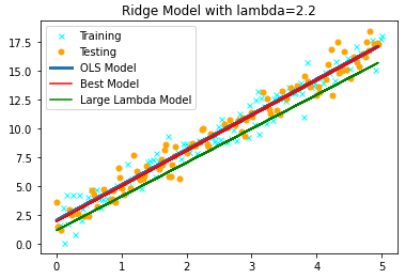


Figure 3: Comparison of model fits on data

The penalty term in ridge regression shrinks all of coefficients towards zero larger that lambda gets. This can be seen in the figure above. The highly penalized model has a flatter slope and smaller y-intercept compared to the ordinary least squares. Although for smaller values of lambda this difference is harder to see which is why sometimes ridge regression models can sometimes fit data better than ordinary least squares.

3.1.2 Polynomial Model

Now we will see how the ridge method compares to ordinary least squares when we try and model polynomial shaped data with the formula:

$$f(x) = x^2 + \epsilon$$

Instead of using the model with the same shape as our data this time we are going to use the a fifth order polynomial to fit this data.

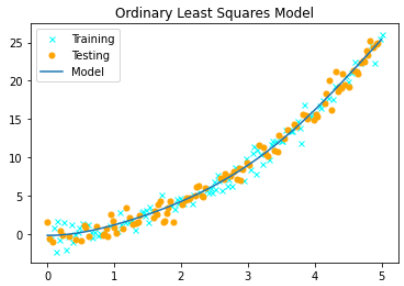


Figure 4: Ordinary least squares approximation

Even though our model does not have the same shape as the data the ordinary least squares solution still fits the data extremely well. This is seen when we run predictions on the test data set and the model achieves and SSE of 77.5.

Now to explore how ridge ridge regression performs on the more complicated data we need to find the lambda value that minimizes the error of the ridge model on the test data

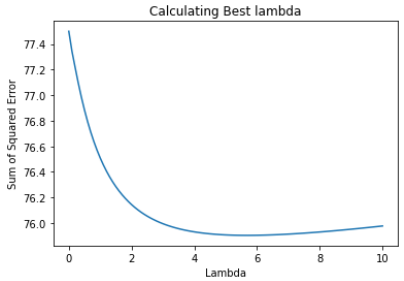


Figure 5: Cross validation of lambda

For the this complicated model more shrinkage is required so the best value of lambda is 5.7. This model achieved a SSE of 75.90, meaning that the ridge model is more accurate compared to ordinary least squares.

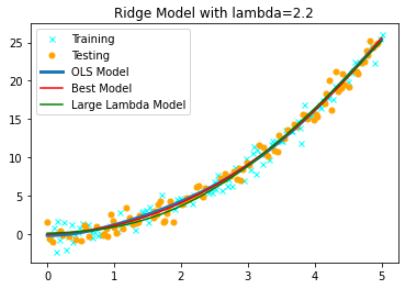


Figure 6: Comparison of model fits on data

For the this more complicated data higher ridge penalties don't affect the model as much as the linear data example. Therefore all of the models look very similar on the interval. This is what allowed for the best ridge penalty to be higher here than in the linear example.

For each of the each of the examples above it was important that we found the accuracy of the model using test data that the model has never seen before. This is because we don't want to over fit the model to the training data. This means that the models coefficients are too specific to the training data and does not represent the true relationship between the variables. This is why we use a test set or validation set to validate the true accuracy of the model.

3.2 Numerical Experiments: Tikhonov Regularization

Given our 3 penalty matrices we can now run numerical experiments to compare them.

3.2.1 Polynomial fit

Given an arbitrary 5th degree polynomial shown in Figure 7

$$0.217979x^5 - 2.34262x^4 + 8.16723 * x^3 - 9.33027 * x^2 + 1.79156$$

We can sample 200 pts in $x \in (0, 4.7)$ and add noise from a normal distribution with $\mu = 0$ and $\sigma^2 = 1$ and let half be our training set and half be our validation set to allow for cross validation similar to that of ridge regression.

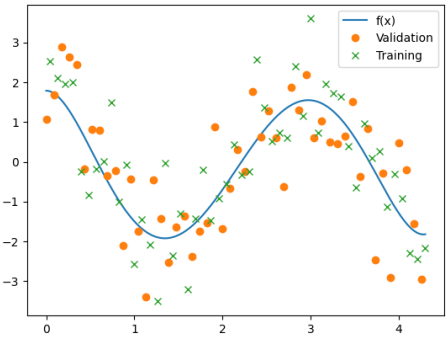
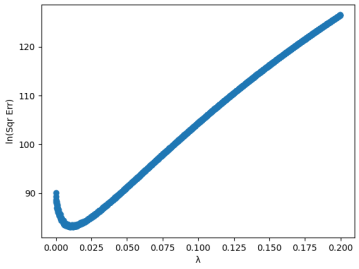
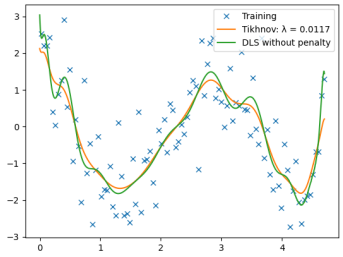


Figure 7: F(x) and data pts

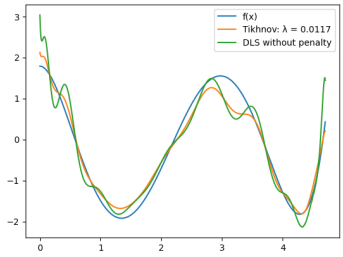
Derivative penalty: First we explore the derivative penalty matrix. We fit it over a high degree of fit $m = 21$ where we would expect the model over fit. To perform the cross validation we will perform the fit over the training data for a linear space of values of λ and graph the squared error against the validation data. We can then pick the lambda with the least squared error and graph it against the original function and a traditional non regularized discrete least square regression.



(a) Cross reference: sqr err vs λ



(b) DLS and Tikhonov against test data



(c) DLS and Tikhonov against f(X)

Method	square error
DLS	121.231
Tikhonov	60.137

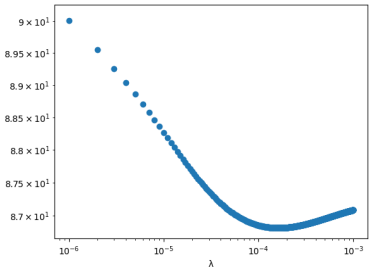
(d) Final error

Figure 8: 200 data pts and m=21 derivative penalty matrix

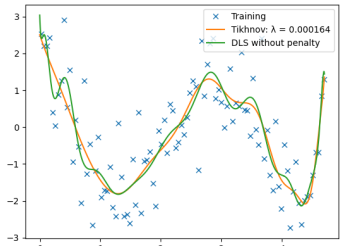
We can see there is a clear minimum at what was found to be $\lambda = 0.0117$ (8a) found over a linear space from 0 to 0.2 with 0.0001 step of λ values. In addition, we can see in (8b) that the DLS without penalty over-fits the training data often oscillating around it. The Tikhonov with the derivative penalty smooths it out creating a visibly lower degree polynomial. This results in a better fit (8c) and less than half the square error in comparison to f(x)(9d).

Gram Eigenvector penalty: We now analyze a similar approach but with the Gram Eigenvector penalty. We will use the second form where we set $d_j = 0$ if $\lambda_i > e\lambda_1$ for some chosen constant $c \in (0, 1)$, and set $d_j = 1$ if not, as it proved to have similar results to method 1. For c we choose 0.001 but in reality any choice of $c > 10^-5$ will have similar results because the noise is normally distributed the v_1 eigenvector hold show the big trend of the data

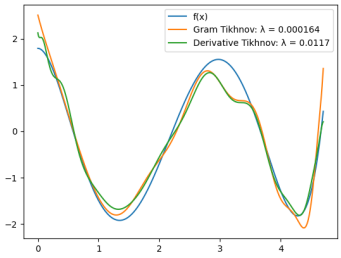
and the other eigenvectors are basically all noise. Shown as the small λ_i where $i > 1$ has a value $<\approx 10^{-5} \cdot \lambda_1$ much smaller than λ_1



(a) Cross reference Log-Log Plt: sqr err vs λ



(b) DLS and Tikhonov against test data



(c) Tikhonov Gram against Tikhonov f(X) Derivative

Method	square error
DLS	121.231
Tikhonov Gram	69.32172

(d) Final error

Figure 9: 200 data pts and m=21 Gram penalty matrix

We plot λ vs error on a log-log plot (9a) as it better depicts the nature of the min. We get a minimum with a quite small λ but its effects of smoothing the graph is quite visible (9b). When comparing gram and the derivative matrix (9c) performance, we see that derivative takes the slight lead (9d). But I think gram especially up to $x = 2$ does a much better job on the eyeball norm (TODO might find a better example to show how gram does for final draft if we have time).
The smoothing, over fit reducing effects of Tikhonov are more visible at lower amounts of data pts as then over fit becomes a much more clear issue. So with the same polynomial will show results with 100 total data pts but still $m = 21$

Method	err 150 data pts	err 100 data pts	err 50 data pts
DLS	492.79	1086.069767635184	9234023.654065767
Gram	111.08	75.949	1009.936
Derivative	59.5577	78.01011152317035	428.78

Table 1: Square err at based on data pts

It can easily be seen that as data pts go down the over fitting increase this can be seen better in the graph from this experiment in the appendix. While the Tikhonov stays relatively good. There some obvious other random factors at play given that the error for Derivative method is less for 150 data pts than 200 (1). This might have something to do with the randomness of the test pts and the 150 we choose fitting the function a bit better. For both we see the value of λ go up as they need more penalty given the higher over fit.

3.2.2 Trigonometric Fit

We preform the same process as last time this time fitting the trig function.

$$f(x) = 2(2 * (-0.53 \sin(3 * x) + 0.2 \cos(3 * x) - 0.32 \sin(6 * x) + 0.53 \cos(6 * x)))$$

Using the same model for error with again 200 pts and stranded normal error we get figure (10)

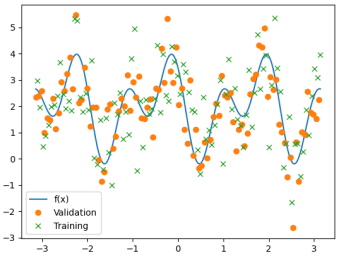
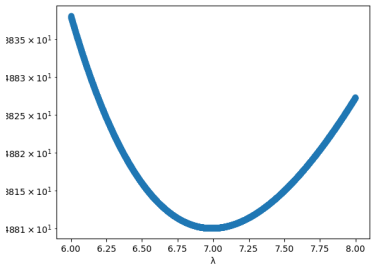
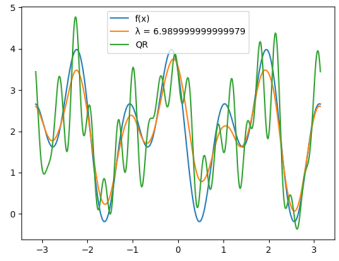


Figure 10: F(x) with data pts

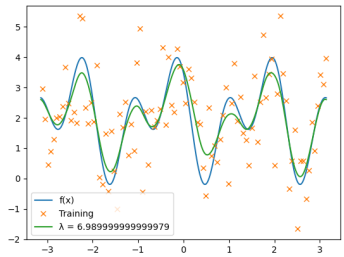
We will create a fit with $m = 20$ allowing severe over fit to show the benefit of Tikhonov.
Low Pass Penalty Matrix: for the low pass matrix we must decide on a value k_0 such that any frequency below $\frac{k_0}{2\pi}$ will not be penalized. As we know our function is made up at most by a frequency $\frac{6}{2\pi}$, a natural choice would be $k_0 = 6$. However this can be found just as before using cross validation and once again a value of around 6 will be found. This is done in the appendix (TODO). In addition we will make the penalty proportional to k^1 as in testing making it proportional to higher power often resulted in compensation from λ and not much difference in error but more though experiments could be done.



(a) Cross reference log Y Plt: sqr err vs λ



(b) DLS and Tikhonov against f(x)



(c) Tikhonov against data and f(x)

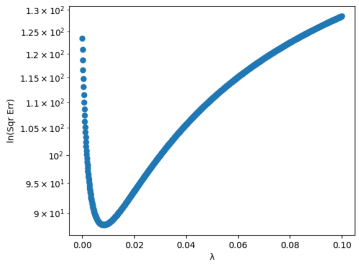
Method	square error
DLS	646.190
Tikhonov	134.242

(d) Final error

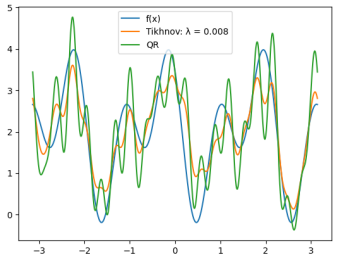
Figure 11: 200 data pts and m=20 Low Pass penalty matrix

First, there is nice minimum for λ around 7 from the figure (11a). We also see the intense over fit from plain DLS in fig (11b) contrasting the accurate fit done by Tikhonov. The fit is exceptional, so to exemplify we include figure (11c). It is obvious how random the test points look compared to how accurate of an approximation the Tikhonov with low pass creates. This is exemplified by the square error compared to the function (11d) where it is around 5 times better of an approximation.

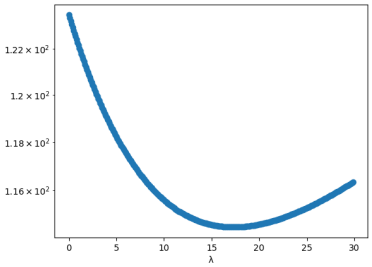
Gram Eigenvalue and Derivative penalty matrices: We repeat the same process as for the polynomial fit with the 2 other types of penalty matrices. Except, of course, with a derivative matrix based on the trig basis. They give us the following results using the same data as we did for the previous fit



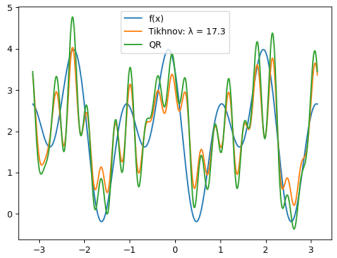
(a) Derivative Penalty, Cross reference log Y Plt: sqr err vs λ



(b) DLS and Derivative Penalty Tikhonov against f(x)



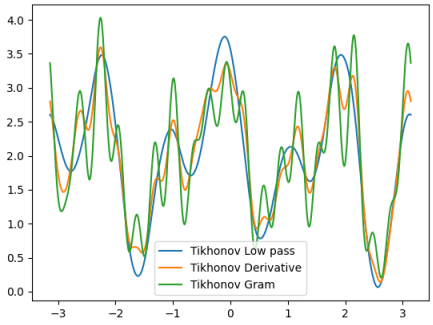
(c) Gram Penalty, Cross reference log Y Plt: sqr err vs λ



(d) DLS and Gram Penalty Tikhonov against f(x)

Figure 12: 200 data pts and m=20 Derivative and Gram penalty matrix

Both have good graphs for cross reference , (12a, 12c) where we see a clear minimum with little to no noise. The graph of the approximations definitely reduce the over-fit (12b, 12d). However in particular the gram penalty (12d) did not perform very well. This might be due to the nature of the function and the normally distributed error might require a bit more investigation. We can also compare all three penalty matrices along with their errors



(a) Gram Penalty vs Derivative Penalty vs Low pass Penalty Tikhonov

Method	square error
DLS	646.190
Low Pass	134.23
Derivative	310.258854
Gram	580.06

(b) Final error

Figure 13: All 3 penalty matrices

The low pass filter performed much better than the other two as seen from the errorn (13b). In addition, its performance is clearly seen looking at how much less over fit there is (13a). However there is a trade off, it takes

more information to construct as there are two free input parameters you have to set or cross reference for.

3.2.3 Noise level - Low Pass Penalty matrix

Finally, we will examine more of the effect of noise given the best performer from the last exercise, the low pass filter. We will look at varying noise levels and its effect on the optimal λ and the performance of Tikhonov regularization. We will follow the exact same parameters as the last experiment with the exception of a variation constant that multiplies the standard normal error. We will test on values of Var:(0.5, 1, 2, 4). To visualize the amount of error we include, a graph (14) of the different variations compared to the function.

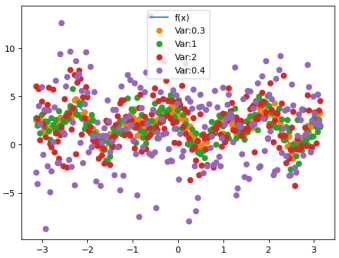


Figure 14: Different Variances

We can now perform a fit at each level all with the low pass filter penalty matrices. Note: in previous experiments we have seeded the random number generator to have consistent results. Though testing the optimal value for lambda heavily depended randomly on the seed. So for consistency sake, we will again seed the number generator.

	Var:0.3	Var:1	Var:2	Var:4
DLS	58.157	646.19	2584.76	10339.04
Tikhonov	12.081	134.24	536.96	2147.87
Tikhonov(λ value)	6.989	6.989	6.989	6.989

Table 2: Error with variance in all data pts

We see the the error is consistently around 5 times better for the Tikhonov than for DLS without regularization. But again these results take a seeded error so the variance essentially just move each point outward. This is why we see no change in the optimal λ value as there is no actual change in the data pts except for a linear scale. In addition, the low pass filter does not take in the data pts. So the penalty term is unchanged and a similar optimization would occur because the validation points are also just scaled outward. If we were instead to leave the error unseeded, we would see varying values of λ with no clear pattern. Once again it is a function of where the over fit and validation points are, and the scale of the variation accounts for itself. So instead, we can change the variation in only the training data not the validation.

	Var:0.3	Var:1	Var:2	Var:4
DLS	22.59	251.066	1004.26	4017.058
Tikhonov	8.679	76.66	299.43	1189.92
Tikhonov(λ value)	0.51	2.38	5.06	10.36

Table 3: Error with different variance in just training pts

Again, we see consistent performance in comparison to DLS. In addition, there is a slight advantage as the variance increases going from 2.6 times better to 3.4 times better. This makes sense as there is likely more over fit to fix. Now that we only change the variation in the training pts, we see the optimal value of λ increasing. This fits with intuition as the more variance there is the more penalty is needed to get the good result compared to the validation.

4 Independent extension

4.1 Introduction

An important thing to consider when constructing a regression model is what variables to include. Intuitively, you want the to include predictors that are the most telling and add significant insight into what the model aims to illustrate. Two additional regression methods, LASSO and Net Elastic Constraint, not only address the risk of over fitting, but they also act as variable selectors. Overall this improves the interoperability of the resultant model because some coefficients are exactly zero, consequently eliminating some predictors. Both have an advantage over Ridge Regression because they can shrink the coefficients all the way to zero, whereas in Ridge the coefficients cannot.

LASSO Regression aims to minimize the sum of squared errors plus a penalty term $\lambda \sum_{j=1}^{n} |c_j|$. The reason as to why LASSO can completely eliminate predictors has to do with the absolute value operation added to to the coefficients. Due to the nature of the absolute value, constraint regions can have sharp edges that the Ridge Regression constraint does not achieve. If we look at the contour graph of the error associated with the model, and it is generally more possible for the corner of the constraint region associated with LASSO to intersect with the error at an axis.

LASSO Regression also has its limitations. One being that it cannot select more p predictors than n observations, and secondly there is still room for improvement in the accuracy of LASSO. This is where net elastic constraint comes into play, it targets both of these impediments to enhance.

Given data $(x_j), f(x_j)$ elastic net uses two penalty terms (λ_1, λ_2) that estimate the coefficient vector c, it essentially combines Ridge Ridge Regression and Lasso, using both penalty terms. It minimizes the sum of squared errors plus the following combined penalty term

$$\lambda_1 \sum_{j=1}^{n} |c_j| + \lambda_2 \sum_{j=1}^{n} c_j^2$$

Solving this will result in a model that not only shrinks the coefficient estimates but also promotes interoperability by eliminating predictors if needed.

4.2 Derivation of Lasso and Elastic Net Regression

We will derive the Lasso and Elastic Net Regression methods from their respective minimizer equation. We will analyze the effect that each penalty term has on the conditioning of the matrix that needs to be inverted to solve for the coefficients for each method. Finally will explore the theoretical reason that the lasso method is able to zero out the unimportant coefficients and why other methods like ridge regression do not.

4.3 Numerical experiments

In this section we will perform numerical experiments to explore how the Lasso penalty and different Elastic Net Regression penalties affect the least squares solution as well as how it differs from the Ridge solution. We will compare the line of best fit that the Lasso and Elastic Net Regression objective functions obtain and compare them to the various models that we made in Section 2. We will also compare how the Ridge,Lasso, and Elastic Net Regression perform when we introduce an unimportant variable into the data.

References

[1] Saptashwa Bhattacharyya. *Ridge and Lasso Regression: L1 and L2 Regularization — towardsdatascience.com.* 2018.

[2] Gene H. Golub, Per Christian Hansen, and Dianne P. O'Leary. "Tikhonov Regularization and Total Least Squares". In: *SIAM Journal on Matrix Analysis and Applications* 21.1 (1999), pp. 185–194. DOI: 10.1137/S0895479897326432. eprint: https://doi.org/10.1137/S0895479897326432. URL: https://doi.org/10.1137/S0895479897326432.

[3] Gareth James. *An Introduction To Statistical Learning With Applications In R.* Springer-Verlag New York Inc., 2013. ISBN: 9781461471370. URL: https://openlibrary.org/books/OL26184759M/An_Introduction_To_Statistical_Learning_With_Applications_In_R.

[4] Hui Zou and Trevor Hastie. “Regularization and Variable Selection Via the Elastic Net”. In: *Journal of the Royal Statistical Society Series B: Statistical Methodology* 67.2 (Mar. 2005), pp. 301–320. ISSN: 1369-7412. DOI: 10.1111/j.1467-9868.2005.00503.x. eprint: https://academic.oup.com/jrsssb/article-pdf/67/2/301/49795094/jrsssb_67_2_301.pdf. URL: https://doi.org/10.1111/j.1467-9868.2005.00503.x.

Appendix A Ridge Code

```
import numpy as np
from numpy.linalg import svd

def ridge_regression(M,y,l):
    MT = np.transpose(M)
    size=M.shape[1]
    U,D,V=svd(MT @ M + l*np.identity(size))
    D_inv=np.diag(1/D)
    c = V.T @ D_inv @ U.T @ MT @ y
    return c

def GenM(x, m):
    M = np.zeros((x.size, m+1))
    for i in range(0, x.size):
        for j in range(0, m+1):
            M[i, j]  = x[i]**j
    return M


def evalmodel(c,x):
    yhat=np.zeros(len(x))
    for i in range(len(c)):
        yhat+=c[i]*x**i
    return yhat


def cv_lambda(l):
    return np.linspace(0,l,10*l+1)
```

Appendix B Tikhonov Code

```
import matplotlib.pyplot as plt
import numpy as np
import math
from numpy.linalg import inv
from numpy.linalg import svd
from numpy.linalg import qr
from numpy.linalg import norm


def DLSQR(M, y):
    Q, R = qr(M)
    QT = Q.T
    c = inv(R) @ QT @ y
    return c



def TikhonovRegression(M, y, l, G):
    MT = np.transpose(M)
    GT = np.transpose(G)
    c = inv(MT @ M + l * (GT @ G)) @ MT @ y
    return c

def TikhonovRegressionQR(M, y, l, G):
    Mp = np.block([[M],[np.sqrt(l)*G]])
    Z = np.zeros(G.shape[0])
    yp = np.block([y,Z])
    Q, R = qr(Mp)
    QT = Q.T
    c = inv(R) @ QT @ yp
```

```
    return c

def derivativePolyPenaltyMatrix(x, m):
    D = np.zeros((x.size, m+1))
    for i in range(0, x.size):
        for j in range(1, m+1):
            D[i, j]  = j * x[i]**(j-1)
    return D
def derivativeTrigPenaltyMatrix(x, m):
    D = np.zeros((x.size, 2*m+1))
    for i in range(0, x.size):
        for k in range(1, m+1):
            D[i, 2*k -1]  = -k* np.sin(k * x[i])
            D[i, 2*k]  = k* np.cos(k * x[i])
    return D


def gramPenaltyMatrixPoly(x, m, e):
    M = GenMPolyBasis(x,m)
    U, s, VT = svd(M)
    L = s**2
    D = np.identity(m+1)
    l1 = L[0]
    for j in range(0, m+1):
        # D[j,j] = 1/L[j]
        if(L[j] > e * l1):
            D[j,j] = 0
        else:
            D[j,j] = 1

    return D @ VT


def gramPenaltyMatrixTrig(x, m, e):
    M = GenMTrigBasis(x,m)
    # print(M)
    U, s, VT = svd(M)
    L = s**2
    print(s)
    D = np.identity(2*m+1)
    l1 = L[0]
    for j in range(0, 2*m+1):
        # D[j,j] = 1/L[j]
        if(L[j] > e * l1):
            D[j,j] = 0
        else:
            D[j,j] = 1

    print(D)

    return D @ VT


def lowPassPenaltyMatrix(m, k0, c, d):
    G = np.zeros((2*m+1,2*m+1))
    for k in range(1, m+1):
        if k > k0:
            G[2*k -1, 2*k -1] = c*k**d
            G[2*k, 2*k]  = c*k**d
    return G


def GenMPolyBasis(x, m):
    M = np.zeros((x.size, m+1))
    for i in range(0, x.size):
        for j in range(0, m+1):
            M[i, j]  = x[i]**j
    return M



def GenMTrigBasis(x, m):
```

```
    M = np.ones((x.size, 2*m +1))
    for i in range(0, x.size):
        for k in range(1, m+1):
            M[i, 2*k -1]  = np.cos(k * x[i])
            M[i, 2*k]  = np.sin(k * x[i])
    return M



def evalTrigFunc(c, xeval):
    m = int((c.size -1)/2)
    y = np.zeros(xeval.size)

    for i in range(0, xeval.size):
        y[i] = c[0]
        for k in range(1, m + 1):
                y[i]  += c[2*k -1]  * np.cos(k * xeval[i])
                y[i]  += c[2*k]  * np.sin(k * xeval[i])
    return y

def evalPolyFunc(c, xeval):
    return np.polyval(np.flip(c), xeval)
```

Appendix C Tikhonov Poly/Trig test code

```
from Tikhonov import *
import numpy as np

np.random.seed(13)
f = lambda x: 0.217979 * x**5-2.34262*x**4+8.16723*x**3-9.33027*x**2+1.79156

a = 0
b = 4.7
x = np.linspace(a, b, 50)
y =  f(x)+np.random.randn(50)

xVal = x[::2]
yVal = y[::2]

xTrain = x[1::2]
yTrain = y[1::2]
m = 21
l = np.arange(0, 0.8, 0.000001)
sqErr = np.zeros(l.size)

M = GenMPolyBasis(xTrain, m)
G = gramPenaltyMatrixPoly(xTrain, m, 0.001)
# G = derivativePolyPenaltyMatrix(xTrain, m)

for i in range(0, l.size):
    pTik = TikhonovRegressionQR(M, yTrain, l[i], G)
    yeval = evalPolyFunc(pTik, xVal);
    sqErr[i] = norm(yVal - yeval)**2
plt.plot(l,sqErr, "o")
plt.xlabel("l")
plt.ylabel("ln(Sqr Err)")
plt.show()


pTik2 = TikhonovRegressionQR(M, yTrain, l[np.argmin(sqErr)], G)

pTik1 = DLSQR(M, yTrain)

xeval =  np.linspace(a, b , 1000);

yevalTik1  = evalPolyFunc(pTik1, xeval)
yevalTik2  = evalPolyFunc(pTik2, xeval)


plt.plot(xeval,f(xeval), label = "f(x)")
plt.plot(xVal,yVal, 'o', label = "Validation")
```

```
plt.plot(xTrain,yTrain, 'x', label = "Training")
plt.plot(xeval,yevalTik2, label = f"Tikhnov: l = {l[np.argmin(sqErr)]}")
plt.plot(xeval,yevalTik1, label = "DLS without penalty")
plt.legend()
plt.show()

yerreval1 = evalPolyFunc(pTik1, xVal);
yerreval2 = evalPolyFunc(pTik2, xVal);

print(f"DLS error =  {norm(f(xeval) - yevalTik1)**2}")
print(f"Tikhonov err =  {norm(f(xeval) - yevalTik2)**2}")
print(f"Tikhonov l =  {l[np.argmin(sqErr)]}")
```