Homework Set 1, CPSC 8420-843

Matthew Collins

November 2, 2024

1 Lasso

The plots in Figure 1 compare the behavior of the Lasso regression coefficients as the regularization parameter λ increases from 0.01 to 1000 using two different approaches: coordinate descent and Scikit-learn's Lasso solver. Both methods display the expected behavior of Lasso regression, where the coefficients shrink towards zero as λ increases, enforcing sparsity in the solution.

In the coordinate descent plot (left), the values of x change significantly for small values of λ. As λ increases, most coefficients are gradually driven to zero, with only a few remaining non-zero. This trend aligns with the behavior expected from Lasso regression, where larger λ values penalize larger coefficients more heavily, ultimately reducing their magnitude.

The Scikit-learn Lasso plot (right) demonstrates a similar pattern. Coefficients that start with larger magnitudes shrink towards zero as λ increases. The overall behavior is consistent with that observed in the coordinate descent method: at low λ, coefficients vary more widely, while at higher λ values, most coefficients become zero or very close to zero.

Both methods exhibit very similar trends, particularly with respect to how coefficients are sparsified. The small discrepancies are due to the underlying differences in the optimization process used by the Scikit-learn solver and the coordinate descent implementation. However, the overall consistency between the two methods suggests that the coordinate descent implementation is functioning correctly.

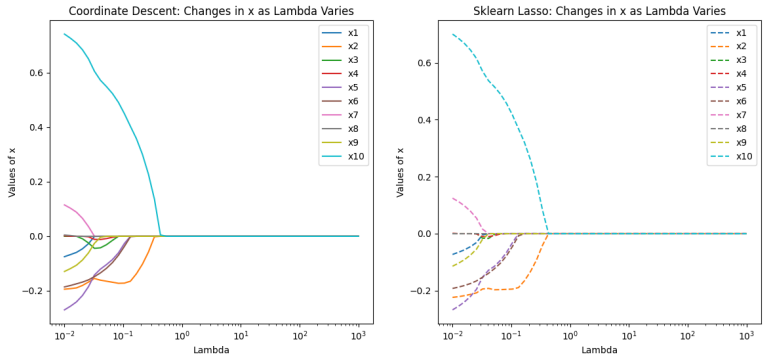

Figure 1: Comparison of changes in x as λ varies using Coordinate Descent (left) and Scikit-learn Lasso (right).

2 Least Squares Extension

Let $A \in \mathbb{R}^{n\times n}$, $C \in \mathbb{R}^{n\times n}$, $X \in \mathbb{R}^{n\times n}$, and $Y \in \mathbb{R}^{n\times n}$ be matrices. The goal is to find X such that:

$$AX + XC = Y$$

The vectorization of a matrix product can be written as:

$$\text{vec}(AX) = (I_n \otimes A)\text{vec}(X)$$
$$\text{vec}(XC) = (C^T \otimes I_n)\text{vec}(X)$$

Thus, the original equation becomes:

$$\text{vec}(AX + XC) = (I_n \otimes A)\text{vec}(X) + (C^T \otimes I_n)\text{vec}(X) = \text{vec}(Y)$$

Simplifying this gives:

$$(A \otimes I_n + I_n \otimes C^T)\text{vec}(X) = \text{vec}(Y)$$

Let $A_{\text{least\_squares}} = A \otimes I_n + I_n \otimes C^T$ and $b = \text{vec}(Y)$. We now have the linear system:

$$A_{\text{least\_squares}}\text{vec}(X) = b$$

To solve for X, we perform the singular value decomposition (SVD) of $A_{\text{least\_squares}}$:

$$A_{\text{least\_squares}} = U\Sigma V^T$$

where U and V are orthogonal matrices, and Σ is a diagonal matrix containing the singular values of $A_{\text{least\_squares}}$. The pseudoinverse of $A_{\text{least\_squares}}$ is given by:

$$A^{+}_{\text{least\_squares}} = V\Sigma^{+}U^{T}$$

where Σ^{+} contains the reciprocals of the non-zero singular values of $A_{\text{least\_squares}}$. Thus, the least squares solution is:

$$\text{vec}(X) = A^{+}_{\text{least\_squares}}b = V\Sigma^{+}U^{T}b$$

Finally, we reshape $\text{vec}(X)$ back into an $n \times n$ matrix to obtain the solution X.

```
X solved using least squares:
 [[ 0.50378786  0.91529498 -0.31212324  1.32862553]
 [ 3.28431873 -1.46885043 -2.67890855  3.10340751]
 [ 1.03389951  0.11449488 -0.34447027  1.30126712]
 [-5.08829341  3.90298648  6.477804   -4.16816548]]
X (ground truth):
 [[0.45615033 0.56843395 0.0187898  0.6176355 ]
 [0.61209572 0.616934   0.94374808 0.6818203 ]
 [0.3595079  0.43703195 0.6976312  0.06022547]
 [0.66676672 0.67063787 0.21038256 0.1289263 ]]
Residual:  1.0314109400307685
Norm difference:  11.649718799382761
```

Figure 2: $X_{\text{least\_squares}}$, $X_{\text{ground\_truth}}$, Residual, and Norm Difference

The results of the least squares solution in Figure 2 for the matrix equation $AX + XC = Y$ provide insight into the performance of the algorithm. The residual, which measures the discrepancy between the left-hand side and the right-hand side of the equation, was approximately 1.03. This indicates that the computed solution, X_{solution}, is fairly close to satisfying the equation. Given that the problem was solved using a least squares approach, a nonzero residual is expected.

On the other hand, the norm difference between the computed solution X_{solution} and the ground truth X^{*} was found to be significantly larger, at approximately 11.65. This large difference indicates that while the computed solution satisfies the equation well (as reflected by the small residual), it is quite different from the true solution. The system may be underdetermined, meaning that multiple solutions exist that satisfy the equation. In such cases, the least

squares approach may find a solution that minimizes the residual, but this solution may differ significantly from the ground truth. Addition of some kind of optimization algorithm such as gradient descent could possibly used to minimize the norm difference.

3 Ridge Regression

$$\hat{\beta}(\lambda) = (A^T A + \lambda I)^{-1} A^T y$$

The term $(A^T A + \lambda I)^{-1}$ is the inverse of the regularized covariance matrix $A^T A$, where λI is a diagonal matrix scaled by λ. As λ increases, the diagonal elements of $A^T A + \lambda I$ increase, causing the overall inverse $(A^T A + \lambda I)^{-1}$ to decrease in magnitude. This happens because larger values of λ make the matrix more "dominant" and reduce the influence of $A^T A$.

Thus, as λ increases, the product $(A^T A + \lambda I)^{-1} A^T y$ becomes smaller, resulting in a smaller norm $\|\hat{\beta}(\lambda)\|_2$.

$$\|\hat{\beta}(\lambda_2)\|_2 \leq \|\hat{\beta}(\lambda_1)\|_2 \quad \text{for} \quad \lambda_1 < \lambda_2$$

This directly shows that the norm of the ridge regression solution is a monotone decreasing function with respect to λ.

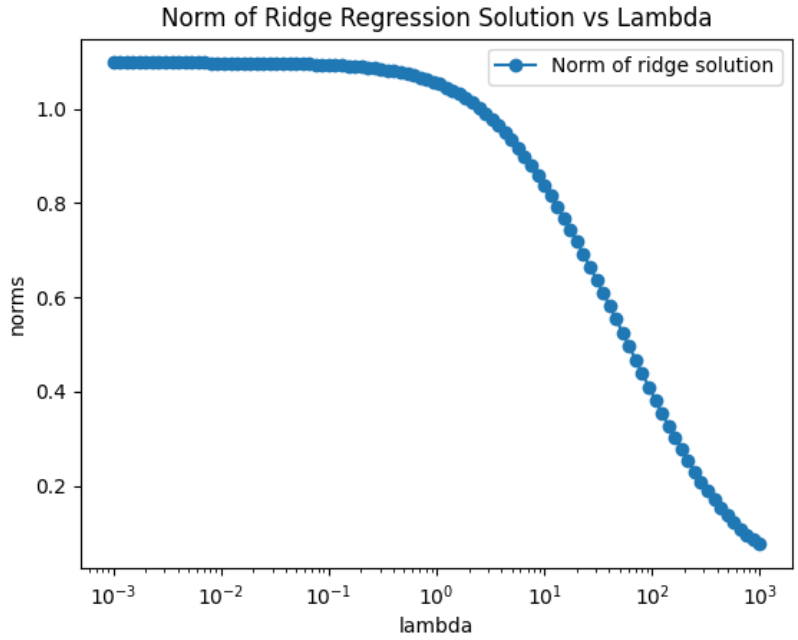

Figure 3: Norm of Ridge Regression as λ Increases

4 Linear Regression and its Extension

4.1 Part One

The Matlab code was converted to Python code, and the data was split into a training set and a testing set and shuffled A.4.

4.2 Part Two

In this assignment, the goal was to implement Ordinary Least Squares (OLS) regression on a dataset and compare the performance of this method with the built-in linear regression model from the Scikit-Learn library. I used a housing dataset, which was shuffled using a random permutation before splitting it into different training sizes of $n = \{25, 50, 75, 100, 150, 200, 300\}$. For each of these training sizes, the training set was standardized using the Z-score normalization method from the SciPy library.

I computed the Mean Squared Error (MSE) on both the training set and the remaining data used as a test set. This was done for both the OLS implementation and the Scikit-Learn linear regression model. The OLS method was implemented by calculating the pseudo-inverse of the design matrix and then computing the weights (or regression coefficients) by multiplying this pseudo-inverse with the training labels A.5

The MSE for each training size was plotted for both methods, as shown in Figure 4. The red line with diamond markers represents the MSE of the training set using the OLS method, and the black line with diamond markers shows the MSE on the test set for the OLS method. Similarly, the blue and yellow dotted lines represent the training and test MSE for the Scikit-Learn model, respectively.

We observe that the results from both the self-built Ordinary Least Squares (OLS) method and the Scikit-Learn linear regression model coincide for both the training and test data, validating the correctness of the custom OLS implementation. As more training data is included, the Mean Squared Error (MSE) for the training set increases slightly. This is because the model is exposed to more varied data, making it harder to perfectly fit all the training points, especially if the data contains noise or non-linearity. This leads to a slight increase in training error, which indicates reduced overfitting as the model generalizes better to unseen data. However, this increase in training MSE is gradual and reflects the model's improved generalization.

At the same time, as the model is trained on more data, its predictions on the test set become more accurate, resulting in a decrease in test MSE. This happens because the model learns a more accurate representation of the underlying data distribution, improving its ability to generalize to unseen data. Importantly, the test set size remains constant throughout the process, and the reduction in test MSE is due to better generalization, not a change in the size of the test set.

Eventually, the training and test error curves converge. This occurs because

the training error increases slightly with more data while the test error decreases, leading to a balance between the model's performance on the training and test sets. When the curves meet, it indicates that the model has learned enough from the data to generalize well and no longer overfits the training set or underfits the test set.

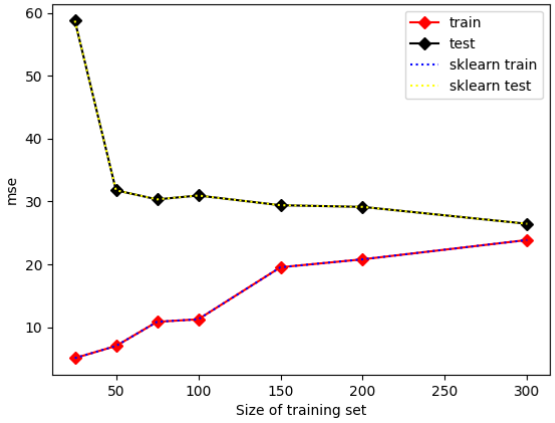

Figure 4: Mean Squared Error vs. Training Set Size for OLS and Scikit-Learn Linear Regression

4.3 Part Three

In this assignment, the task was to expand the original features of the housing dataset to higher-order terms and evaluate how increasing the degree of polynomial expansion impacts the performance of the model. Specifically, the expansion involved replacing the original features with higher powers of the features (i.e., quadratic, cubic, and so on, up to degree 6). Interaction terms between features were ignored, meaning only pure polynomial terms were included in the expansion. The expanded features were standardized in the same way as the original features, and the expanded data set was treated consistently with the previously processed data.

It is important to note that due to certain binary features in the dataset (such as a feature that takes values 0 or 1), the degree expansion of those features results in the same values as the original set. This causes the design matrix to become non-invertible, meaning that the matrix $A^T A$ cannot be inverted directly. To address this, the pseudoinverse function '`pinv()`' was used instead of the regular inverse '`inv()`' to ensure numerical stability and handle the non-invertibility of the matrix.

The custom function `ordinary_least_squares` was implemented to perform the least squares regression, using the pseudoinverse for the inversion step, while

the Scikit-Learn `LinearRegression` function was also used for comparison. The Mean Squared Error (MSE) for both the training and test sets was calculated for each degree of expansion, and the results were plotted. The plot in Figure 5 shows the relationship between the MSE and the degree of feature expansion. The red line with diamond markers represents the training MSE for the custom Ordinary Least Squares (OLS) implementation, while the black line with diamond markers shows the test MSE for the OLS method. The blue and yellow dotted lines represent the training and test MSE for the Scikit-Learn linear regression model, respectively.

As shown in the plot, the test error initially decreases at degree 2, indicating that this level of feature expansion offers the best prediction in terms of minimizing the test set MSE. However, as the degree of polynomial expansion continues to increase, the test error rises sharply, indicating the onset of overfitting. This overfitting is further evidenced by the fact that the training error remains low, even as more complex features are added. The model becomes increasingly complex with higher degrees, which allows it to fit the training data almost perfectly, but at the cost of poor generalization to unseen test data. This behavior illustrates the trade-off between bias and variance as model complexity increases. The sharp increase in test MSE at higher degrees, particularly at degrees 5 and 6, highlights this overfitting problem.

The results of the custom function were cross-validated using the Scikit-Learn package, and the trends observed in the self-built function matched those obtained using Scikit-Learn, further validating the implementation.

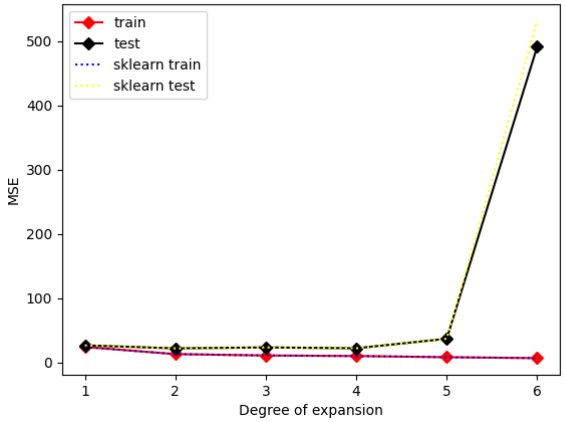

Figure 5: MSE for training and test data as a function of the degree of polynomial expansion

4.4 Part Four

In this assignment, the goal was to apply ridge regression to a degree 6 polynomial expansion of the housing dataset features. The ridge regression adds an L_2 regularization term to the least squares cost function, which helps reduce overfitting by penalizing large coefficients. The regularization strength is controlled by the parameter λ, and in this assignment, we explored the effect of different values of λ on the model's performance.

We performed the degree 6 polynomial expansion by creating higher-order terms for each feature in the dataset. This was done by raising the original features to the powers of 2 through 6, then concatenating these expanded features with the original ones. The expanded feature matrix was then standardized using Z-score normalization to ensure that all features were on the same scale.

The ridge regression was performed using both a custom function, `ridge_least_squares`, which computes the pseudoinverse of the design matrix and incorporates the regularization term, and the `Ridge` function from Scikit-Learn. For comparison, we computed the Mean Squared Error (MSE) for both the training and test sets as a function of λ, where λ was varied over a log scale between 10^{-10} and 10^{10}.

The results are presented in Figure 6. The green line with diamond markers represents the training MSE for the custom ridge regression, while the cyan line shows the test MSE for the custom implementation. Similarly, the blue and red lines represent the training and test MSE for the Scikit-Learn `Ridge` model.

As observed in the plot, the test MSE initially decreases as λ increases, indicating that a small amount of regularization improves the model's generalization by reducing overfitting. However, as λ continues to increase, the test error begins to rise again. This occurs because higher values of λ excessively penalize the model coefficients, leading to underfitting, where the model is too constrained to capture the complexity of the data. The training MSE, on the other hand, increases steadily with increasing λ, as larger regularization forces the model to fit the training data less closely.

Overall, the behavior of ridge regression demonstrates the trade-off between bias and variance: small λ values lead to overfitting (low bias but high variance), while large λ values lead to underfitting (high bias but low variance).

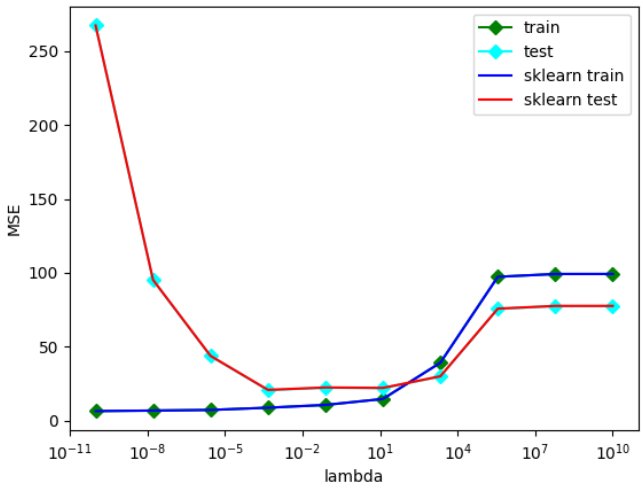

Figure 6: MSE vs. λ for the training and test sets with ridge regression.

When λ is very small in ridge regression, the solution closely resembles that of conventional ordinary least squares (OLS) regression. Therefore, with a small λ, we observe a significant gap between the training and test MSE, especially in high-degree polynomial expansions like the 6-degree expansion shown in Figure 6. This gap arises because the model overfits the training data, resulting in low training error but high test error due to poor generalization.

The ridge regression implementation in the self-built function was cross-validated using the Scikit-Learn `Ridge` function, as shown in the plot. The pseudo-inverse was used to ensure numerical stability in solving the regression, especially because high-degree polynomial expansions often lead to ill-conditioned design matrices. Ridge regression also includes a bias (or intercept) term, which was not regularized, ensuring that only the coefficients of the features were penalized. This is done by multiplying the λ regularization term with a $(p \times p)$ identity matrix, leaving the intercept term unregularized.

As λ increases, the model is regularized more strongly, which reduces the variance and prevents overfitting. This leads to a decrease in the test MSE. However, as λ becomes too large, some important coefficients are shrunk too much, causing the model to underfit, and the bias increases. This results in a rise in both the training and test MSE at higher values of λ. In summary, ridge regression balances bias and variance: small λ values can lead to overfitting, while large λ values can cause underfitting.

References

[1] Chan, Stanley H. *Introduction to Probability for Data Science*. Michigan Publishing, 2023, pp. 449-457.

[2] Xavier Bourret Sicotte. Lasso Implementation. Available at: https://xavierbourretsicotte.github.io/lasso_implementation.html.

[3] Towards Data Science. Regularized Linear Regression Models. Available at: https://towardsdatascience.com/regularized-linear-regression-models-dcf5aa662ab9.

[4] Towards Data Science. From Linear Regression to Ridge Regression. Available at: https://towardsdatascience.com/from-linear-regression-to-ridge-regression-the-lasso-and-the-elastic-net-4eaecaf5f7e6.

[5] Chan, Stanley H. *Introduction to Probability for Data Science*. Michigan Publishing, 2023, pp. 396-422.

[6] Math Stack Exchange. Least Squares Solution to Underdetermined Lyapunov Equation. Available at: https://math.stackexchange.com/questions/4808422/least-squares-solution-to-underdetermined-lyapunov-equation.

[7] AIMS Press. Solving Underdetermined Linear Systems. Available at: https://www.aimspress.com/article/doi/10.3934/mmc.2021009?viewType=HTML.

[8] Chan, Stanley H. *Introduction to Probability for Data Science*. Michigan Publishing, 2023, pp. 440-448.

[9] Medium. Ridge Regression Step-by-Step Introduction. Available at: https://medium.com/@msoczi/ridge-regression-step-by-step-introduction-with-example-0d22dddb7d54.

[10] Chan, Stanley H. *Introduction to Probability for Data Science*. Michigan Publishing, 2023, Chapter 7.

[11] Medium. Understanding Ordinary Least Squares (OLS) and Its Applications in Machine Learning. Available at: https://medium.com/@dahami/understanding-ordinary-least-squares-ols-and-its-applications-in-statistics-machine-lea

[12] Mashkar Haris. Linear Regression in Python Using Scikit-Learn. Available at: https://mashkarharis.medium.com/linear-regression-in-python-scikit-learn-526b57a11a09.

[13] ContactSunny. Linear Regression in Python Using Scikit-Learn. Available at: https://contactsunny.medium.com/linear-regression-in-python-using-scikit-learn-f0f7b125a204.

[14] Medium. Mastering Ridge Regression. Available at: https://medium.com/@bernardolago/mastering-ridge-regression-a-key-to-taming-data-complexity-98b67d343087.

[15] Medium. Ridge and Lasso Regression: Practical Implementation in Python. Available at: https://alok05.medium.com/ridge-and-lasso-regression-practical-implementation-in-python-c4a813a99bce.

A Appendix

A.1 Lasso

```
# CPSC 8420-843, Fall 2024
# Homework 1, Problem 1 (Lasso)
# Matthew Collins - Charleston Campus
# References:
#   Chan, Stanley H. Introduction to Probability for Data Science. Michigan
#   https://xavierbourretsicotte.github.io/lasso_implementation.html
#   https://towardsdatascience.com/regularized-linear-regression-models-dcf5
#   https://towardsdatascience.com/from-linear-regression-to-ridge-regressio

import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import Lasso

def lasso_coordinate_descent(A, y, lambd, num_iters=1000, tol=1e-4):
    m, n = A.shape
    x = np.zeros(n)

    for _ in range(num_iters):
        x_old = x.copy()

        for j in range(n):
            residual = y - np.dot(A, x) + A[:, j] * x[j]
            rho = np.dot(A[:, j].T, residual)

            if rho < -lambd / 2:
                x[j] = (rho + lambd / 2) / np.dot(A[:, j], A[:, j])
            elif rho > lambd / 2:
                x[j] = (rho - lambd / 2) / (A[:, j] @ A[:, j])
            else:
                x[j] = 0

        if np.linalg.norm(x - x_old, ord=2) < tol:
            break
```

```
    return x

np.random.seed(0)
A = np.random.randn(20, 10)
y = np.random.randn(20)
lambdas = np.logspace(np.log10(0.01), np.log10(1000), 50)
x_values = []
x_sklearn_values = []

print(A.shape)

for lambd in lambdas:
    x_star = lasso_coordinate_descent(A, y, 4*A.shape[1]*lambd)
    x_values.append(x_star)

    lasso = Lasso(alpha=lambd, max_iter=10000, tol=1e-4)
    lasso.fit(A,y)
    x_sklearn_values.append(lasso.coef_)

x_values = np.array(x_values)
x_sklearn_values = np.array(x_sklearn_values)

plt.figure(figsize=(14, 6))
plt.subplot(1, 2, 1)
for i in range(A.shape[1]):
    plt.plot(lambdas, x_values[:, i], label=f'x{i+1}')
plt.xscale('log')
plt.xlabel('Lambda')
plt.ylabel('Values of x')
plt.title('Coordinate Descent: Changes in x as Lambda Varies')
plt.legend()

plt.subplot(1, 2, 2)
for i in range(A.shape[1]):
    plt.plot(lambdas, x_sklearn_values[:, i], label=f'x{i+1}', linestyle='--
plt.xscale('log')
plt.xlabel('Lambda')
plt.ylabel('Values of x')
plt.title('Sklearn Lasso: Changes in x as Lambda Varies')
plt.legend()

#plt.show()
plt.savefig(".\\sln_figures\\fig1.png")
```

A.2 Least Square Extension

```
# CPSC 8420-843, Fall 2024
# Homework 1, Problem 2 (Least Squares Extension)
# Matthew Collins - Charleston Campus
# References:
#   Chan, Stanley H. Introduction to Probability for Data Science. Michigan
#   https://math.stackexchange.com/questions/4808422/least-squares-solution-
#   https://www.aimspress.com/article/doi/10.3934/mmc.2021009?viewType=HTML

import numpy as np

def solve_least_squares(A, C, Y):
    n = A.shape[0]
    I_n = np.eye(n)
    A_kron = np.kron(I_n, A)
    C_kron = np.kron(C.T, I_n)
    A_least_squares = A_kron + C_kron
    b = Y.flatten()

    U, s, Vt = np.linalg.svd(A_least_squares, full_matrices=False)
    s_inv = np.diag(1 / s)
    A_pseudo_inv = Vt.T @ s_inv @ U.T  # Pseudo-inverse using SVD

    X_vec = A_pseudo_inv @ b
    X = X_vec.reshape(n, n)

    return X


def calculate_residual(A, C, X, Y):
    residual = A @ X + X @ C - Y
    return np.linalg.norm(residual, 'fro')  # Frobenius norm of the residual

def calculate_norm_difference(X_solution, X_star):
    return np.linalg.norm(X_solution - X_star, 'fro')


n = 4
np.random.seed(0)
A = np.random.rand(n, n)*0.1
C = np.random.rand(n, n)*0.1
X_star = np.random.rand(n,n)

Y = A @ X_star + X_star @ C
X_solution = solve_least_squares(A, C, Y)
```

```
residual = calculate_residual(A, C, X_solution, Y)
norm_difference = calculate_norm_difference(X_solution, X_star)

print("X solved using least squares:\n", X_solution)
print("X (ground truth):\n", X_star)
print("Residual: ", residual)
print("Norm difference: ", norm_difference)
```

A.3 Ridge Regression Monotone

```
# CPSC 8420-843, Fall 2024
# Homework 1, Problem 3 (Monotonic Ridge Regression)
# Matthew Collins - Charleston Campus
# References:
#   Chan, Stanley H. Introduction to Probability for Data Science. Michigan
#   https://medium.com/@msoczi/ridge-regression-step-by-step-introduction-wi

import numpy as np
import matplotlib.pyplot as plt

np.random.seed(0)
m, n = 100, 50
A = np.random.randn(m, n)
y = np.random.randn(m, 1)

lambdas = np.logspace(-3, 3, 100)
norms = []

for lam in lambdas:
    beta_ridge = np.linalg.inv(A.T @ A + lam * np.eye(n)) @ A.T @ y
    norm = np.linalg.norm(beta_ridge, 2)
    norms.append(norm)

plt.plot(lambdas, norms, marker='o', label='Norm of ridge solution')
plt.xscale('log')
plt.xlabel('lambda')
plt.ylabel('norms')
plt.title('Norm of Ridge Regression Solution vs Lambda')
plt.legend()
#plt.show()
plt.savefig(".\\sln_figures\\fig3.png")
```

A.4 Loading Dataset

```
import numpy as np
from sklearn.model_selection import train_test_split

data = np.loadtxt('housing.data')

x = data[:, :13]
y = data[:, 13]
n, d = x.shape
np.random.seed(0)

perm = np.random.permutation(n)
x = x[perm]
y = y[perm]
training_samples = 300

Xtrain = x[:training_samples]
ytrain = y[:training_samples]
Xtest = x[training_samples:]
ytest = y[training_samples:]
```

A.5 MSE vs Training Size

```
# CPSC 8420-843, Fall 2024
# Homework 1, Problem 4.2
# Matthew Collins - Charleston Campus
# References:
#   Chan, Stanley H. Introduction to Probability for Data Science. Michigan
#   https://medium.com/@dahami/understanding-ordinary-least-squares-ols-and-
#   https://mashkarharis.medium.com/linear-regression-in-python-scikit-learn
#   https://contactsunny.medium.com/linear-regression-in-python-using-scikit

import numpy as np
import matplotlib.pyplot as plt
import matplotlib
import scipy.stats as stats
import sklearn.linear_model as sl
from sklearn.metrics import mean_squared_error


data = np.loadtxt('housing.data')
np.random.seed(0)
randperm = np.random.randint(0,len(data),len(data))
data = data[randperm,:]

def ordinary_least_squares(A,y):
    result = np.linalg.pinv((A.T).dot(A)).dot(A.T)
```

```
    beta = result.dot(y)
    return beta

def mse(y,beta,x):
    y_hat = x.dot(beta)
    return ((y-y_hat).T).dot(y-y_hat)[0][0]/len(y)

n = [25,50,75,100,150,200,300]
mse_test = []
mse_train = []
mse_SL_train = []
mse_SL_test = []

for Ntrain in n:
    Xtrain = stats.zscore(data[:Ntrain, :-1], axis=0)
    ytrain = data[:Ntrain, -1][..., None]

    Xtest = stats.zscore(data[Ntrain:, :-1], axis=0)
    ytest = data[Ntrain:, -1][..., None]

    Xtrain_mt = np.concatenate((np.ones((len(Xtrain), 1)), Xtrain), axis=1)
    Xtest_mt = np.concatenate((np.ones((len(Xtest), 1)), Xtest), axis=1)

    skl_model = sl.LinearRegression().fit(Xtrain, ytrain)
    beta = ordinary_least_squares(Xtrain_mt, ytrain)

    mse_train.append(mse(ytrain, beta, Xtrain_mt))
    mse_test.append(mse(ytest, beta, Xtest_mt))
    mse_SL_train.append(mean_squared_error(ytrain, skl_model.predict(Xtrain)))
    mse_SL_test.append(mean_squared_error(ytest, skl_model.predict(Xtest)))


plt.plot(n,mse_train,color='red',label='train',marker='D')
plt.plot(n,mse_test,color='black',label='test',marker='D')
plt.plot(n,mse_SL_train,color='blue',label='sklearn train', linestyle=':')
plt.plot(n,mse_SL_test,color='yellow',label='sklearn test', linestyle=':')
plt.legend()
plt.ylabel('mse')
plt.xlabel('Size of training set')
#plt.show()
plt.savefig(".\\sln_figures\\fig4_2.png")
```

A.6 MSE vs Degree

```
# Homework 1, Problem 4.3
# Matthew Collins - Charleston Campus
```

```
# References:
#   Chan, Stanley H. Introduction to Probability for Data Science. Michigan
#   https://medium.com/@dahami/understanding-ordinary-least-squares-ols-and-
#   https://mashkarharis.medium.com/linear-regression-in-python-scikit-learn
#   https://contactsunny.medium.com/linear-regression-in-python-using-scikit

import numpy as np
import matplotlib.pyplot as plt
import scipy.stats as stats
import sklearn.linear_model as sl
from sklearn.metrics import mean_squared_error

data = np.loadtxt('housing.data')
seed = 0
np.random.seed(seed)
randperm = np.random.randint(0, len(data), len(data))
data = data[randperm, :]

def ordinary_least_squares(A, y):
    result = np.linalg.pinv((A.T).dot(A)).dot(A.T)
    beta = result.dot(y)
    return beta

def mse(y, beta, X):
    y_hat = X.dot(beta)
    return ((y - y_hat).T).dot(y - y_hat)[0][0] / len(y)

Ntrain = 300
mse_test = []
mse_train = []
mse_SL_test = []
mse_SL_train = []

Xtrain = stats.zscore(data[:Ntrain, :-1], axis=0)
ytrain = data[:Ntrain, -1][..., None]
Xtrain_mt = np.concatenate((np.ones((len(Xtrain), 1)), Xtrain), axis=1)

Xtest = stats.zscore(data[Ntrain:, :-1], axis=0)
ytest = data[Ntrain:, -1][..., None]
Xtest_mt = np.concatenate((np.ones((len(Xtest), 1)), Xtest), axis=1)

linear_model = sl.LinearRegression().fit(Xtrain, ytrain)

beta = ordinary_least_squares(Xtrain_mt, ytrain)

mse_train.append(mse(ytrain, beta, Xtrain_mt))
```

```
mse_test.append(mse(ytest, beta, Xtest_mt))

mse_SL_train.append(mean_squared_error(ytrain, linear_model.predict(Xtrain)))
mse_SL_test.append(mean_squared_error(ytest, linear_model.predict(Xtest)))

degrees = [2, 3, 4, 5, 6]
features = data[:, :-1]

for deg in degrees:
    features = np.concatenate((features,data[:,:-1]**deg),axis=1)

    Xtrain = stats.zscore(features[:Ntrain,:],axis=0)
    ytrain = data[:Ntrain,-1][...,None]

    Xtest = stats.zscore(features[Ntrain:,:],axis=0)
    ytest = data[Ntrain:,-1][...,None]

    Xtrain_mt = np.concatenate((np.ones((len(Xtrain),1)),Xtrain),axis=1)
    Xtest_mt = np.concatenate((np.ones((len(Xtest),1)),Xtest),axis=1)

    skl_model = sl.LinearRegression().fit(Xtrain,ytrain)
    beta = ordinary_least_squares(Xtrain_mt, ytrain)

    mse_train.append(mse(ytrain,beta,Xtrain_mt))
    mse_test.append(mse(ytest,beta,Xtest_mt))
    mse_SL_train.append(mean_squared_error(ytrain,skl_model.predict(Xtrain)))
    mse_SL_test.append(mean_squared_error(ytest,skl_model.predict(Xtest)))


plt.plot([1]+degrees,mse_train,color='red',label='train',marker='D')
plt.plot([1]+degrees,mse_test,color='black',label='test',marker='D')
plt.plot([1]+degrees,mse_SL_train,color='blue',label='sklearn train', linest
plt.plot([1]+degrees,mse_SL_test,color='yellow',label='sklearn test', linest
plt.legend()
plt.ylabel('MSE')
plt.xlabel('Degree of expansion')
#plt.show()
plt.savefig(".\\sln_figures\\fig4_3.png")
```

A.7 Ridge Regression

```
# CPSC 8420-843, Fall 2024
# Homework 1, Problem 4.3
# Matthew Collins - Charleston Campus
# References:
#   Chan, Stanley H. Introduction to Probability for Data Science. Michigan
```

```
#   https://medium.com/@dahami/understanding-ordinary-least-squares-ols-and-
#   https://mashkarharis.medium.com/linear-regression-in-python-scikit-learn
#   https://contactsunny.medium.com/linear-regression-in-python-using-scikit
#   https://medium.com/@bernardolago/mastering-ridge-regression-a-key-to-tam
#   https://alok05.medium.com/ridge-and-lasso-regression-practical-implementa

import numpy as np
import matplotlib.pyplot as plt
import scipy.stats as stats
import sklearn.linear_model as sl
from sklearn.metrics import mean_squared_error

data = np.loadtxt('housing.data')
seed = 0
np.random.seed(seed)
randperm = np.random.randint(0, len(data), len(data))
data = data[randperm, :]

def ridge_least_squares(A, y, lam):
    iden_part = lam * np.identity(len(A.T))
    iden_part[0, 0] = 1  # Do not regularize the bias term
    result = np.linalg.pinv((A.T).dot(A) + iden_part).dot(A.T)
    beta = result.dot(y)
    return beta

def mse(y, beta, X):
    y_hat = X.dot(beta)
    return ((y - y_hat).T).dot(y - y_hat)[0][0] / len(y)

Ntrain = 300

mse_test = []
mse_train = []
mse_SL_test = []
mse_SL_train = []

lambdas = np.logspace(-10, 10, 10)

degrees = [2, 3, 4, 5, 6]
features = data[:, :-1]

for deg in degrees:
    features = np.concatenate((features, data[:, :-1] ** deg), axis=1)

Xtrain = stats.zscore(features[:Ntrain, :], axis=0)
ytrain = data[:Ntrain, -1][..., None]
```

```
Xtest = stats.zscore(features[Ntrain:, :], axis=0)
ytest = data[Ntrain:, -1][..., None]

Xtrain_mt = np.concatenate((np.ones((len(Xtrain), 1)), Xtrain), axis=1)
Xtest_mt = np.concatenate((np.ones((len(Xtest), 1)), Xtest), axis=1)

for l in lambdas:
    beta = ridge_least_squares(Xtrain_mt, ytrain, l)
    skl_model = sl.Ridge(l).fit(Xtrain, ytrain)

    mse_train.append(mse(ytrain, beta, Xtrain_mt))
    mse_test.append(mse(ytest, beta, Xtest_mt))

    mse_SL_train.append(mean_squared_error(ytrain, skl_model.predict(Xtrain)
    mse_SL_test.append(mean_squared_error(ytest, skl_model.predict(Xtest)))

plt.plot(lambdas, mse_train, color='green', label='train', marker='D')
plt.plot(lambdas, mse_test, color='cyan', label='test', marker='D')
plt.plot(lambdas, mse_SL_train, color='b', label='sklearn train')
plt.plot(lambdas, mse_SL_test, color='r', label='sklearn test')
plt.xscale('log')
plt.legend()
plt.ylabel('MSE')
plt.xlabel('lambda')

#plt.show()
plt.savefig(".\\sln_figures\\fig4_4.png")
```