Linear Algebra HW 10

gjd9961

November 2021

1 Problem 10.1

Let $A \in \mathbb{R}^{n \times m}$ and $y \in \mathbb{R}^n$, we consider the least squares problem as minimize $||Ax - y||^2$ with respect to $x \in \mathbb{R}^m$.

a) Show that $x^{LS} \perp Ker(A)$

To show this property, we can leverage a couple pieces of information that we have from lecture, as well as proofs from past homeworks. We know from our work with SVD that any matrix $A \in \mathbb{R}^{n \times m}$ can be represented as $A = U\Sigma V^T$, and that the first r columns of U will form the basis of the Image of A, where the $Rank(A) = r$, and that the first r columns of V will define the basis of the row space, and the columns $V_{r+1}, \dots, V_{m-r}$ will define the basis of the Kernel of A. We also know the following that $A^\dagger = V\Sigma' U^T$.

By definition, x^{LS} is a vector in the input space, $\mathbb{R}^m$, and thus will be some linear combination of $v_1, \dots, v_r$, that is the first r columns of V. Becasue all of the columns of V are orthonormal, it follows that the columns $v_{r+1}, \dots, v_{m-r}$ that form the basis of $Ker(A)$ are orthogonal to the first r columns of V, so $v_1, \dots, v_r \perp v_{r+1}, \dots, v_{m-r}$ and thus $x^{LS} \perp Ker(A)$

b) Deduce that x^{LS} is the solution that has the smallest Euclidian norm.

We know from lecture that we can take a solution for x^{LS} and add some vector from the Kernel of A, that solution will still be a minimizer. We can express it with the following set notation:

$$\{A^\dagger y + v | v \in Ker(A)\}$$

But since we are adding non-zero vectors and thus values to the minimizer solution, it's euclidean norm will grow. Lets define an alternative solution to our minimizer x^{LS}, and lets call it x^{Alt} where x^{ALT} is $A^\dagger y + v$ where $v \in Ker(A)$.

$$\begin{aligned}
&||x^{LS}||^2 < ||x^{Alt}||^2 \\
&||A^\dagger y||^2 < ||A^\dagger y + v||^2 \\
&||A^\dagger y||^2 < ||A^\dagger y||^2 + 2\langle v, A^\dagger y\rangle + ||v||^2 \\
&||A^\dagger y||^2 < ||A^\dagger y||^2 + ||v||^2 \text{ since } A^\dagger y \perp v \text{ by part (a)}
\end{aligned} \tag{1}$$

Therefore, the minimizer with the smallest norm is x^{LS}.

2 Problem 10.2

Let $A \in \mathbb{R}n \times d$ and $y \in \mathbb{R}^n$. The ridge regression adds a l_2 penalty to the least squares term. Minimize $||Ax - y||^2 + \lambda||x||^2$ with respect to $x \in \mathbb{R}^d$ for some penalization parameter $\lambda > 0$

a) Without solving 2, show that 2 admits a unique solution. You can use HW 9 results, but justify everything you use.

We know from HW 9 that if you add a convex function to a convex function the result function is a convex function. So therefore, the function will have a minimizer. But we're not sure if this minimizer is unique.

Fortunately, we also know from HW 9 that if a matrix is Positive Definite, then it is strictly convex, and admits a unique minimizer. Furthermore, we know that we can make a matrix positive definite by making its eigenvalues positive definite. We also know from past home-works that we can add some $\lambda \times Id_n$ to a matrix A where $\lambda > 0$ and the result is the eigenvalues of A get increased by λ. When we play around with the equation in part b, we will show how the ridge regression minimizer function is positive definite (and invertible) and thus will admit a unique solution.

b) Show that the solution is given by:

$$x^{Ridge} = (A^T A + \lambda Id_n)^{-1} A^T y$$

Justify your answer precisely including why $(A^T A + \lambda Id_n)^{-1}$ exists.

So here what we want to do is take the gradient of the original minizer function, and set it to 0. Then well get our expression, and we can elaborate why $(A^T A + \lambda Id_n)^{-1}$ exists.

$$\begin{aligned} f(x) &= ||Ax - y||^2 + \lambda||x||^2 \\ \nabla f(x) &= A^T(Ax - y) + \lambda x \text{ using the properties from lec/hw 9} \\ \nabla f(x) = 0 &= A^T Ax - A^T y + \lambda x \text{ set the gradient to 0 and solve for x} \\ A^T Ax + \lambda x &= A^T y \\ (A^T A + \lambda Id_n)x &= A^T y \quad \square \end{aligned} \tag{2}$$

Like we said in part a) we know that for some choice of $\lambda > 0$ we can make $A^T A$ positive definite and therefore, strictly convex and invertible (which is why $(A^T A + \lambda Id_n)^{-1}$ exists). Therefore, for some λ we have:

$$x^{ridge} = (A^T A + \lambda Id_n)^{-1} A^T y$$

3 Problem 10.3

Recall that $||M||_{sp}$ denotes the spectral norm of a matrix M.
a) Let $A \in \mathbb{R}^{n \times m}$. Show that for all $x \in \mathbb{R}^m$:

$$||Ax|| \leq ||A||_{sp}||x||$$

To solve this we're going to have to reexpress some of the terms in our inequality. Firstly, $||Ax|| = \sqrt{x^T A^T Ax}$ We can evaluate this expression further as we know a few properties of the $A^T A$ term in the middle. Firstly, that $A = U \Sigma V^T$:

$$A^T A = V \Sigma^T U^T U \Sigma V^T = V \Sigma^2 V^T$$

Where Σ is the squared eigenvalues of $A^T A$, and U, V^T are orthogonal matrices. And, since we know that $x \in \mathbb{R}^m$ and $V \in \mathbb{R}^{m \times m}$ and V is orthonormal, we can express x in terms of V.

$$x = \sum_{i=1}^{min(n,m)} \alpha_i v_i$$

where $\alpha \in \mathbb{R}$ and $v_1, \ldots, v_m$ are the orthonormal columns of V. So what we now have is:

$$\begin{aligned} ||Ax||^2 = x^T A^T Ax = & \sum_{i=1}^{min(n,m)} \alpha_i v_i^T \times A^T A \sum_{i=1}^{min(n,m)} \alpha_i v_i \\ ||Ax||^2 = & \sum_{i=1}^{min(n,m)} \alpha_i v_i^T \times (\sum_{i=1}^{min(n,m)} \alpha_i (A^T A v_i)) \\ ||Ax||^2 = & \sum_{i=1}^{min(n,m)} \alpha_i v_i^T \times (\sum_{i=1}^{min(n,m)} \alpha_i (\lambda_i v_i)) \\ ||Ax||^2 = & c_i^2 \lambda_i u_i^T u_i = c_i^2 \lambda_i \end{aligned} \tag{3}$$

Therefore, the upper bound of the transformation $||Ax||^2$ and thus $||Ax||$ is constrained by the largest Eigenvalue of $||Ax||$ hence the connection to the spectral norm of A. Therefore, we have:

$$||Ax|| \leq \sqrt{\lambda_{max} \sum_{i=1}^{min(n,m)} c_i^2} = ||A||_{sp}||x|| \quad \square$$

b) Show that for all $A \in \mathbb{R}^{n \times m}$ and $B \in \mathbb{R}^{m \times k}$:

$$||AB||_{sp} \leq ||A||_{sp}||B||_{sp}$$

we can use a little trick with a vector with norm 1 to show this property, and with using what we saw in part a. Let $y \in \mathbb{R}^k$ and $||y|| = 1$, therefore:

$$||ABy|| \leq ||A||_{sp}||By|| \leq ||A||_{sp}||B||_{sp}||y|| = ||A||_{sp}||B||_{sp}$$

We know that to maximize $||ABy||$ we'd have to set $||y|| = 1$ which is the definition of the spectral norm of $||AB||$ so therefore:

$$||AB||_{sp} \leq ||A||_{sp}||B||_{sp} \quad \square$$

4 Problem 10.4

In [22]:
```
import matplotlib.pyplot as plt
import numpy as np
from numpy import linalg
```

Polynomial regression as linear least squares

From the knowledge of a sample of pair of scalar values $\{a_i, y_i\}_{i=1}^n$, we would like to predict the relation between x and y. One simple way to go beyond linear regression is to consider polynomial regression: for example we could try to model y as a polynomial of degree 3 of a. We would look for $(x_0, x_1, x_2, x_3) \in \mathbb{R}^4$ such that the values a_i and y_i are linked as $y_i \simeq x_0 + x_1 a_i + x_2 a_i^2$.

This problem can be mapped to linear regression by considering that we have for each a a feature vectors of dimension $d + 1$ when considering the fit of a polynomial of degree d. This feature vector is $(1, a, a^2, \cdots, a^d)$. Such that the full data matrix is

$$A = \begin{bmatrix} 1 & a_1 & \cdots & a_1^d \\ 1 & a_2 & \cdots & a_2^d \\ \vdots & \vdots & \vdots & \vdots \\ 1 & a_n & \cdots & a_n \end{bmatrix} \in \mathbb{R}^{n \times (d+1)}.$$

As a exercise below we will consider data that was created from a polynomial of dimension 3, to which noise is added. Assuming that we do not know the degree of the generated polynomial, we will try to fit with $d = 5$ and $d = 2$ and investigate ridge regression.

In [23]:
```
## Helper functions to setup the problem
def get_data_mat(a, deg):
    """
    Inputs:
    a: (np.array of size N)
    deg: (int) max degree used to generate the data matrix

    Returns:
    A: (np.array of size N x (deg_true + 1)) data matrix
    """
    A = np.array([a ** i for i in range(deg + 1)]).T
    return A

def draw_sample(deg_true, x, N, eps=0):
    """
    Inputs:
    deg_true: (int) degree of the polynomial g
    a: (np.array of size deg_true) parameter of g
    N: (int) size of sample to draw
    eps: noise level

    Returns:
    x: (np.array of size N)
```

```
    y: (np.array of size N)
    """
    a = np.sort(np.random.rand(N))
    A = get_data_mat(a, deg_true)
    y = A @ x + eps * np.random.randn(N)
    return a, y
```

(a) Complete the three functions below to obtain

- the least square estimator x^{LS}
- the ridge estimator x^{Ridge}
- the mean square error $||Ax - y||^2/n$

In [24]:

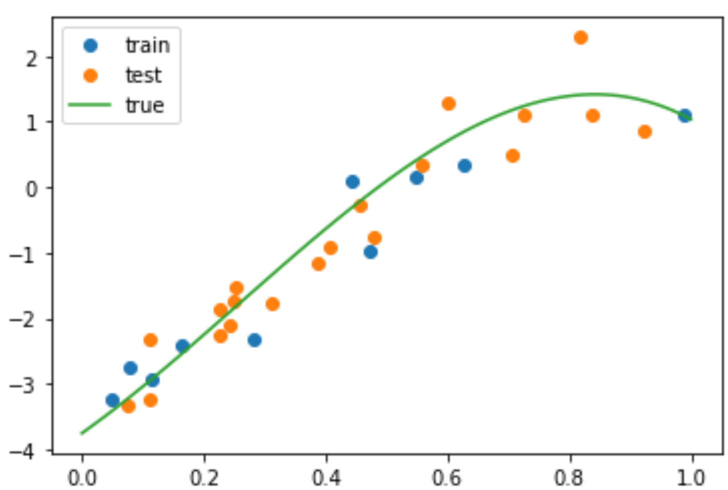
```
def least_square_estimator(Matrix, vector):

    #Get the SVD decomp using numpy
    U, sigma, V= linalg.svd(Matrix, full_matrices=False)

    #Scale the singular values
    sigma = 1/sigma

    #Use numpy to make it a diagonal matrix
    sigma_diagonal = np.diag(sigma)

    #Calculate A dagger, the inverse of the SVD
    A_dagger = V.T@sigma_diagonal@U.T

    #Compute the least squared solution
    x_least_squared = A_dagger@vector
    return(x_least_squared)

def ridge_estimator(Matrix, Vector, penalty):

    #Create A^TA, the square symmetric matrix version of the input mat
    squared_version_of_matrix = Matrix.T@Matrix

    #Initialie an identity matrix and scale the diagonal by the labmda
    scaled_identity = np.identity(squared_version_of_matrix.shape[1])*

    #Shit the eigenvalues of the square matrix A^TA by A^TA + lambda*I
    squared_version_of_matrix= squared_version_of_matrix + scaled_iden

    #Use numpy to calculate the inverse
    inverse = np.linalg.inv(squared_version_of_matrix)

    #Calculate the ridge regression solution
    x_ridge_regression_solution = inverse@Matrix.T@Vector

    return(x_ridge_regression_solution)


def mean_squared_error(x, A, y):
```

```
    #Initialize variables
    total_error_var = 0
    estimation = A@x

    for i in range(len(y)):
        #Loop through for each prediction and find the total squared e
        squared_error = (estimation[i]-y[i])**2
        total_error_var += squared_error

    return(total_error_var/len(y))
```

In [25]:

```
# This cells generates the data - for your submission do not change it
# But for your own curiosity, do not hesitate to investigate what is

np.random.seed(45) # fixing seed so everyone should see the same data
N = 10
deg_true = 3 # degree of true polynomial
eps = 0.5 # noise amplitude
x_true = np.array([-3.75307359,  6.58178662,  6.23070014, -8.02457871]

# radom input data
a_tr, y_tr = draw_sample(deg_true, x_true, N, eps=eps) # training data
a_te, y_te = draw_sample(deg_true, x_true, 2 * N, eps=eps) # testing d

a_plot = np.linspace(0, 1, 100)
A_plot = get_data_mat(a_plot, deg_true)
plt.plot(a_tr, y_tr,'o', label='train')
plt.plot(a_te, y_te,'o', label='test')
plt.plot(a_plot, A_plot @ x_true,'-', label='true')
plt.legend()
```

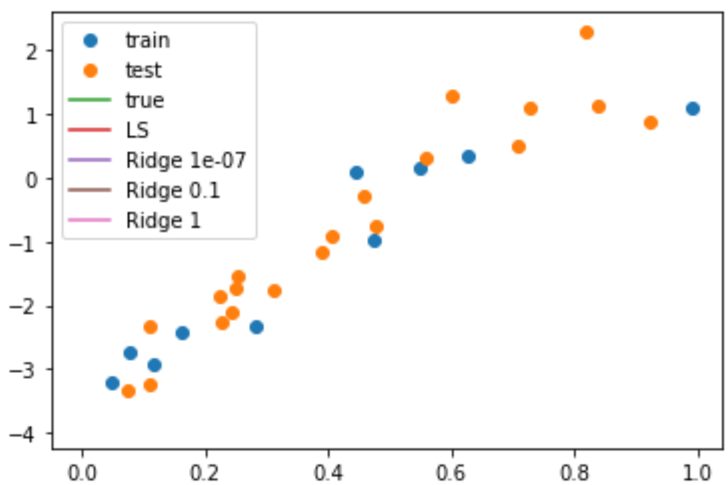
Out[25]: `<matplotlib.legend.Legend at 0x7fe1547fc460>`



(b) Complete the code below to visualize the prediction of x^{LS} and x^{Ridge} for λ in $[1e-7, 0.1, 1]$, using in all cases a prediction model of degree 5. The output of the cell should be a plot as above, where you added three lines of predictions for all values of $a \in [0,1]$: line LS, line ridge $\lambda = 1e-7$, line ridge $\lambda = 0.1$, line ridge $\lambda = 1$.

In [26]:
```
a_plot = np.linspace(0,0,100)
A_plot = get_data_mat(a_plot, deg_true)
plt.plot(a_tr, y_tr,'o', label='train')
plt.plot(a_te, y_te,'o', label='test')
plt.plot(a_plot, A_plot @ x_true,'-', label='true')

deg_pred = 5

A_tr = get_data_mat(a_tr, deg_pred)
A_te = get_data_mat(a_te, deg_pred)

x_ls = least_square_estimator(A_tr, y_tr)

A_plot = get_data_mat(a_plot, deg_pred)
plt.plot(a_plot, A_plot @ x_ls, label='LS')

for lbd in [1e-7, 0.1, 1]:

    x_ridge_reggresion = ridge_estimator(A_tr,y_tr,lbd)
    plt.plot(a_plot, A_plot@ x_ridge, label='Ridge '+str(lbd))


plt.legend()
```

Out[26]: <matplotlib.legend.Legend at 0x7fe1557bca30>

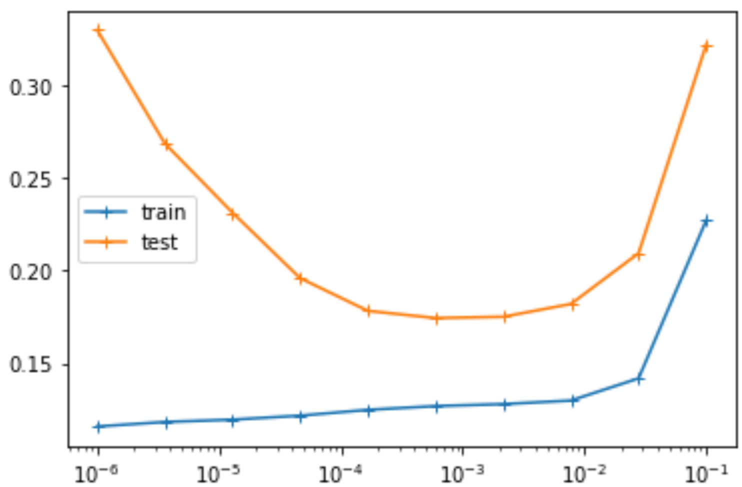


Type *Markdown* and LaTeX: α^2

(c) Use the `mean_squared_error` to make a plot of the training error and the test error as a function of λ as we have seen in the lecture (range given below). Which value of λ would you choose? Does that align with your intuition from the plots above?

In [27]:
```
tr_mse = []
te_mse = []
lbds = np.logspace(-6, -1, 10)
minimal_error = 100



for lbd in lbds:

    #Calculate ridge regression error using the functions we defined
    x_ridge_regression_error = ridge_estimator(A_tr, y_tr, lbd)

    #Get training error
    training_error = mean_squared_error(x_ridge_regression_error, A_tr

    #Get testing error
    testing_error = mean_squared_error(x_ridge_regression_error, A_te,

    #Append the train/test errors
    tr_mse.append(training_error)
    te_mse.append(testing_error)

    #Check if we found a minimal lambda, using the error terms
    if testing_error < minimal_error:
        min_lambda = lbd
        minimal_error = testing_error

plt.plot(lbds, tr_mse, '-+', label='train')
plt.plot(lbds, te_mse,'-+', label='test')
plt.xscale('log')
plt.legend()
```

Out[27]:
```
<matplotlib.legend.Legend at 0x7fe155aa5dc0>
```

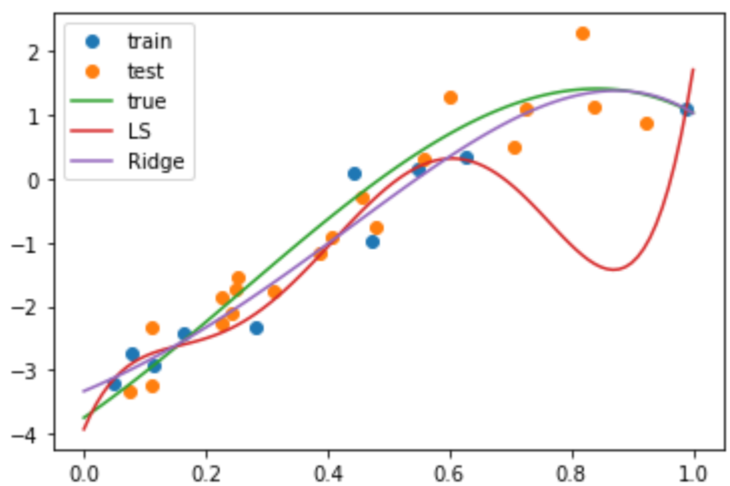


Type *Markdown* and LaTeX: α^2

(d) For the optimal value of λ compare x^{LS}, x^{Ridge} and x^{true}.

In [34]:
```
#Set matplotlib code for visuals
a_plot = np.linspace(0,1,100)
A_plot = get_data_mat(a_plot, deg_true)
plt.plot(a_tr, y_tr,'o', label='train')
plt.plot(a_te, y_te,'o', label='test')
plt.plot(a_plot, A_plot @ x_true,'-', label='true')
deg_pred = 5


#Retrive the training and test data
training_data = get_data_mat(a_tr, deg_pred)
testing_data = get_data_mat(a_te, deg_pred)

x_least_squared_error = least_square_estimator(training_data, y_tr)

A_plot = get_data_mat(a_plot, deg_pred)
plt.plot(a_plot, A_plot @ x_least_squared_error, label='LS')


lbd = np.logspace(-6, -1, 10)[5]
x_ridge_regression = ridge_estimator(training_data,y_tr,lbd)
plt.plot(a_plot, A_plot@ x_ridge_regression, label='Ridge')


plt.legend()
```

Out[34]: <matplotlib.legend.Legend at 0x7fe156767df0>



The ridge regression curve is a better approximation of the true function, given it tracks it more closely on the graph, enough so that we can clearly see it

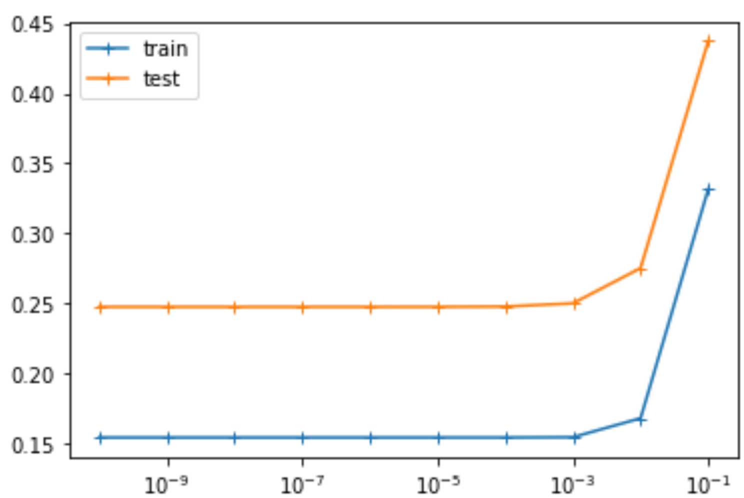
(e) Repeat the same operation with a fitting model of degree 2 (`deg_pred=2`). What are your findings related to the optimal degree of regularizations in this case?

In [37]:
```
#Initialize variables and such from previous functions
deg_pred = 2
```

```
training_data = get_data_mat(a_tr, deg_pred)
testing_data = get_data_mat(a_te, deg_pred)
tr_mse = []
te_mse = []
lbds = np.logspace(-10, -1, 10)
min_error = 100

#Loop through the lambdas
for lbd in lbds:

    #Calculate the ridge regression errors
    x_ridge_raining_data = ridge_estimator(training_data, y_tr, lbd)
    error_train = mean_squared_error(x_ridge_raining_data, training_da
    error_test = mean_squared_error(x_ridge_raining_data, testing_data

    #Append train/test errors
    tr_mse.append(error_train)
    te_mse.append(error_test)

    #Again, see if we found the minimal error
    if error_test < min_error:
        min_lambda = lbd
        min_error = error_test

print(min_lambda)

plt.plot(lbds, tr_mse, '-+', label='train')
plt.plot(lbds, te_mse,'-+', label='test')
plt.xscale('log')
plt.legend()
```

```
1e-10
```

Out[37]: <matplotlib.legend.Legend at 0x7fe156428be0>



We can clearly see that from our graph, as we increase lambda we get worse an worse results, without seeing any benefeits. Since with ridge regression our lambda parameter must be greater than 0, it is not helping at all. We would be better off doing LS regression, and not doing regularization at all. Optimal lambda would be 0.

In [38]:
```
a_plot = np.linspace(0,1,100)
A_plot = get_data_mat(a_plot, deg_true)
plt.plot(a_tr, y_tr,'o', label='train')
plt.plot(a_te, y_te,'o', label='test')
plt.plot(a_plot, A_plot @ x_true,'-', label='true')
deg_pred = 2

training_data = get_data_mat(a_tr, deg_pred)
testing_data = get_data_mat(a_te, deg_pred)

x_least_squared_regression = least_square_estimator(training_data, y_t

A_plot = get_data_mat(a_plot, deg_pred)
plt.plot(a_plot, A_plot @ x_least_squared_regression, label='LS')


lbd = np.logspace(-6, -1, 10)[5]
x_ridge_regression = ridge_estimator(A_tr,y_tr,lbd)
plt.plot(a_plot, A_plot@ x_ridge_regression, label='Ridge')


plt.legend()
```

Out[38]: <matplotlib.legend.Legend at 0x7fe156a8feb0>