

MidtermExam

April 30, 2022

```
[1]: import numpy as np
import matplotlib.pyplot as plt
import numpy.linalg as la

import pandas as pd
import seaborn as sns
sns.set_theme(style="whitegrid")

fs=24
lw=4

rms = lambda x: np.sqrt(np.mean(np.square(x)))
```

1 Midterm Exam

1.1 Python Problem 3 [Problem 3 part e]: Least Squares Placement

1.1.1 Problem Setup

Let us recall the problem setup. The vectors p_1, \dots, p_N , each in \mathbb{R}^2 represent the locations of N factories. There are two types of factories: “square” factories and “circle” factories. The first K factories are square factories, whose locations are fixed and given. Our goal in the placement problem to choose the locations of the last $N - K$ factories i.e the circle factories.

Our choice of the locations is guided by an undirected graph; an edge between two factories means we would like them to be close to each other. In least squares placement, we choose the locations p_{K+1}, \dots, p_N so as to minimize the sum of the squares of the distances between factories connected by an edge, where the L edges of the graph are given by the set E . For a specific location of square factories $p_1 \dots p_K$, we can frame our task as solving the following optimization:

$$g(p_1, \dots, p_K) = \min_{p_{K+1}, \dots, p_N} \sum_{(i,j) \in E} \|p_i - p_j\|^2$$

In the code below, we have set up a specific instance of this problem.

```
[2]: N, K, L = 10, 5, 13

edges = [(1,6), (2,6), (5,6), (1,7), (4,7),
         (2,8), (3,8), (3,9), (5,9), (5,10), (7,9), (6, 10), (7,10)]
```

```

p1 = np.array([0.55,0.15])
p2 = np.array([0,0])
p3 = np.array([0,1])
p4 = np.array([1,1])
p5 = np.array([1,0])

fixed_locs = [p1, p2, p3, p4, p5]

```

1.1.2 Solving the problem

In parts (b) and (c), you reduced the problem to a least squares problem. Here, we set up and solve the least squares problem to obtain the optimal locations for the circle factories.

[3]: *#TODO: Create Incidence Matrix*

```

B = [[0,0,0,0,0,1,1,0,0,0]
      [0,0,0,0,0,1,0,1,0,0]
      [0,0,0,0,0,0,0,1,1,0]
      [0,0,0,0,0,0,1,0,0,0]
      [0,0,0,0,0,1,0,0,1,0]
      [1,1,0,0,1,0,0,0,0,1]
      [1,0,0,1,0,0,0,0,1,1]
      [0,1,1,0,0,0,0,0,0,0]
      [0,0,1,0,0,0,1,0,0,0]
      [0,0,0,0,1,1,1,0,0,0]]

```

[12]: *#TODO: Set up least squares problem*

```

B_T = np.transpose(B)
A = [0.55,0,0,1,1,0.15,0,1,1,0]

```

why did you make this test so hard :(

[15]: *#TODO: Solve least squares problem.*

#Hint: np.linalg.lstsq() might be helpful here.

```

X = np.linalg.lstsq(A,B)
u_m = X[0:5]
u_v = X[6:10]

```

u_m, v_m = # contains the solution to the location placement problem

1.1.3 Plotting the solution

Now, we want to plot the locations we solved for above. Show the graph edges as lines connecting the locations.

Below the variables `u_m` and `v_m` should contain the solution to the locations from above.

```
[ ]: chosen_locs = []
for i in range(len(u_m)):
    new_loc = [u_m[i], v_m[i]]
    chosen_locs.append(new_loc)

fixed_locs, chosen_locs = np.array(fixed_locs), np.array(chosen_locs)
all_locs = np.concatenate((fixed_locs, chosen_locs))

plt.figure(figsize=(10,6))
plt.tick_params(labelsize=fs-2)

for i in range(len(all_locs)):
    plt.annotate(str(i+1), (all_locs[i, 0] + 0.01, all_locs[i, 1]+0.02),
    ↪fontsize=fs-4,zorder=2)

for (source, dest) in edges:
    source_x, source_y = all_locs[source - 1][0], all_locs[source - 1][1]
    dest_x, dest_y = all_locs[dest - 1][0], all_locs[dest - 1][1]
    plt.plot([source_x, dest_x], [source_y, dest_y], color = 'xkcd:grey',
    ↪linewidth=2,zorder=1)

plt.scatter(fixed_locs[:, 0], fixed_locs[:, 1], marker = 's', color = 'red', s=
    ↪100,zorder=2)
plt.scatter(chosen_locs[:, 0], chosen_locs[:, 1], s = 120,zorder=2)

plt.xlim([-0.1,1.1])
plt.ylim([-0.1,1.1])
plt.savefig("problem3e_plot.png")
```

1.2 Python Problem 4 [Problem 4 part c]: Recursive Least squares

You are asked to implement recursive least squares using the formula given in the following equations:

$$\hat{\theta}_{m+1} = \hat{\theta}_m + R_{m+1}^{-1} x^{(m+1)} (y^{(m+1)} - (x^{(m+1)})^\top \hat{\theta}_m) \quad (1)$$

$$R_{m+1} = R_m + x^{(m+1)} (x^{(m+1)})^\top \quad (2)$$

where $R_{m+1} := \sum_{k=1}^{m+1} x^{(k)} (x^{(k)})^\top$.

You will test it on a randomly generated data set, and on the housing data set. The housing data set is provided with the exam materials but you can find it on the course website [here](#).

1.2.1 Part 1 [Random Least Squares Instance]

Part 1a Implement the recursive least squares method as a function. It should take in the following as inputs: 1. size of the data set over which you will run the algorithm n , 2. an initial

value for R_0 and an initial value for $\hat{\theta}_0$, 3. The data X you will use to recursively estimate the value of θ 4. N which is the number of features you use to compute R_0

It should return a list of R_m and $\hat{\theta}_m$ values.

Towards this end, you will need generate a random least squares instance (the code is given below). Then to ensure R_0 is invertible, we will take the first $N = 500$ rows of X and compute

$$R_0 = \sum_{i=1}^N x^{(i)}(x^{(i)})^\top \quad \text{and} \quad \hat{\theta}_0 = (R_0)^{-1} \sum_{i=1}^N x^{(i)}y^{(i)}.$$

This means that since $X \in \mathbb{R}^{m \times 2}$ (we have two features and $m = 5000$ data points), we have $n = m - N$. That is, we will run the recursive least squares method for the remaining feature vectors.

You are encouraged to play around with the size of the data m , the number N of initial feature vectors you take to compute R_0 , and the random seed. However, you do not have to submit anything on that. Just submit the notebook (and printed pdf) for the values described above.

```
[ ]: np.random.seed(20)
N=500
m=5000
n=m-N

### Generate a least squares instance
x = np.linspace(0, 1, m)
theta_true=[1,5, 3] # true theta, i.e. your recursive least squares estimate
    ↳should converge to this
y = theta_true[0]+x*theta_true[1]+theta_true[2]*x**2+np.random.normal(loc=0.
    ↳0,scale=1.5, size=len(x))

# turn y into a column vector
y = y[:, np.newaxis]

# assemble the data matrix X
# this is just the univariate fit from Mod2
X = np.vstack([np.vstack([np.ones(len(x)),x]), x**2]).T

print("dimension of X : ", np.shape(X))

## write your code to compute R_0
R0= # enter code here
print("inv(R0) : \n", la.inv(R0))

## here we can print out the condition number (this will be discussed in detail
    ↳in Mod3)
## the condition number is a measure of how "invertible" a matrix is. You want
    ↳it to be small
```

```

## if you play around with the size of N relative to m, you can see how adding
→more vectors effects the condition number
print(la.cond(R0))

## write your code to compute theta_0
theta0= #enter code here
print("initial theta : ", theta0)

def RLSQ(theta0, R0, n, N=N):
    ## Fill in code here to implement recursive least squares
    return thetas, Rs

# Run RLSQ
thetas, Rs=RLSQ(theta0,R0,n)

```

Part 1b Compute the error and plot it Run least squares on the random data instance and take the output of $\hat{\theta}_m$ and compute the error for each iterate:

$$\|\theta^{\text{lsq}} - \hat{\theta}_m\|_2 \quad \text{for each } m \in \{0, \dots, n\}$$

where θ^{lsq} is the least squares approximate solution. And, then plot it. Print out the least squares solution (e.g., obtained with numpy or the least squares formula) you computed and print out the recursive least squares solution you computed.

```

[2]: # Compute the actual least squares theta
# you will need this to compute the error
theta_lsq= np.linalg(y,x) # compute least squares solution using analytic
→formula or numpy.linalg

error= np.linalg.norm(theta_true - thetas) #compute the error using the output
→of RLSQ (thetas)

# Plot it
plt.figure(figsize=(8,5))
plt.tick_params(labelsize=fs-2)
plt.plot(error, linewidth=lw, color='xkcd:tomato red')
plt.xlabel('iterations', fontsize=fs)

# print out values
print("True theta value           : ", theta_true)
print("Numpy Least Squares Solution : ", theta_lsq.T[0])
print("Recursive Least squares solution : ", thetas[-1])

```

Part 1c [Plot the data and the estimated line] Now plot the line you estimated with the RLSQ function and the data.

```
[3]: line=# fill in code to compute  $f(x)$  using the final value of  $\hat{\theta}_m$ 
      ↪ computed from RLSQ
```

```
plt.figure(figsize=(8,6))
plt.tick_params(labelsize=fs-2)
plt.xlabel('x',fontsize=fs-2)
plt.ylabel('y',fontsize=fs-2)
plt.plot(x,y, '.', color='xkcd:red orange')
plt.plot(x,line, linewidth=lw, color='xkcd:bright blue')
```

1.2.2 Part 2 [Housing Data]

Now you will run your recursive least square method on the housing data. First load it, and then using the price as y , and area, number of beds, and a constant as features, create a least squares problem. Run your method on it, and plot the error as in part 1b.

```
[112]: np.random.seed(20)
file_loc= "../lecture-ntbks/data/housing.csv" # enter the location of housing.
      ↪ csv as a string. For example "../housing.csv" if its in the current directory.
df=pd.read_csv(file_loc)
```

```
price = df["price"];
area = np.asarray(df["area"]);
beds = np.asarray(df["beds"]);

# show head of df
df.head()
```

```
[112]:
```

	area	baths	beds	condo	location	price
0	0.941	2	2	1	2	94.905
1	1.146	2	3	0	2	98.937
2	0.909	2	3	0	2	100.309
3	1.289	2	3	0	3	106.250
4	1.020	1	3	0	3	107.502

```
[4]: X = # create the data matrix using a vector of 1's for the constant, areas and
      ↪ beds
y = # set y as the price vector
```

```
# set m to the number of rows of X
m=np.shape(X)[0]
print("m : ", m)
# set N
N=5
# define number of iterations to run RLSQ
n=m-N
```

```

print("n : ", n)

# Compute R0 using the first N features
R0=# Fill this in
print(la.cond(R0))

# compute theta0 using R0
theta0= # fill this in

# Run RLSQ
thetas, Rs=RLSQ(theta0,R0,n, N=N)

# Compute the actual least squares theta
theta_lsq= # fill this in
print(np.shape(theta_lsq))

# compute the error and plot it
error= # Fill this in
print(len(error))
plt.figure(figsize=(8,5))
plt.tick_params(labelsize=fs-2)
plt.plot(error, linewidth=lw, color='xkcd:tomato red')
plt.xlabel('iterations', fontsize=fs)

print("Numpy Least Squares Solution      : ", theta_lsq)
print("Recursive Least squares solution : ", thetas[-1])

```

[]: