You are *not* allowed to work with your peers. Show all work and derivations to receive credit. Write clearly and legibly for your own benefit.

Name : Jasmine Soh

SID : 1777662

Attention!!: You will have to upload the final Python notebook to Canvas in the midterm exam assignment to get credit in addition to attaching a print out pdf to the exam you submit!

Problem 1 (Linear Independence, Matrices) [6pts].

Consider three vectors $u, v, w \in \mathbb{R}^3$ that are linearly independent, and let $a, b, c \in \mathbb{R}^3$ be defined as

$$a = u + v,\ b = v + w,\ c = u + w,\ d = u - v.$$

a. Suppose vectors u and v have unit length. What is the angle between the vectors a and d?

b. Again suppose vectors u and v have unit length. Give an expression for a left-inverse of the matrix $A = \begin{bmatrix} a & d \end{bmatrix}$ (your expression can depend on a and d).

c. Show that nullspace of the matrix $B = \begin{bmatrix} a & b & c \end{bmatrix}$ contains only the zero vector $\mathbf{0} \in \mathbb{R}^3$.

Solution.

a. $90°$

b. $XA = I$

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \begin{bmatrix} a & d \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \begin{bmatrix} u+v & u-v \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

$$\left.\begin{aligned} x_1(u+v) &= 1 \\ x_2(u+v) &= 0 \\ x_1(u-v) &= 0 \\ x_2(u-v) &= 1 \end{aligned}\right\} \quad x_1 = \frac{1}{u+v},\ x_2 = \frac{1}{u-v}$$

$$\boxed{X = \begin{bmatrix} 1/a \\ 1/d \end{bmatrix}}$$

C. If the nullspace of the matrix only contains the zero vector, then this means a,b,c are linear independent. We see that a = u + v, b = v + w, and c = u + w and it is given that u,v,w are linear independent so that means a, b, c are linear independent. Thus the nullspace only contains the zero vector by definition.

Problem 2 (Matrix inverse & properties) [6pts]. Consider the following $n \times n$ matrix,

$$S = \begin{bmatrix} 1 & 0 & \dots & 0 & 0 \\ 1 & 1 & \dots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 1 & 1 & \dots & 1 & 0 \\ 1 & 1 & \dots & 1 & 1 \end{bmatrix}$$

What does the matrix S do when applied to a vector? Find the inverse of S: Solve $SX = I$ for the unknown matrix X by writing the linear equations each column of X should satisfy. What is the interpretation of S^{-1}?

Considering SX=B where S is a lower triangular matrix containing only 1's, then when S is applied to some vector X, the result B is a matrix with the ith value equivalent to the sum of the first i indices of X.

finding the inverse:

$$\underset{S}{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix}} \underset{X}{\begin{bmatrix} 1 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & 0 & -1 & 1 \end{bmatrix}} = \underset{\underline{I}}{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}}$$

X looks like a matrix w/ ones along the diagonal and negative ones to the left of the diagonal.

$s_{11} x_{11} + \dots + s_{1n} x_{n1} = 1$ ← column 1 of x has to satisfy

$s_{21} x_{12} + \dots + s_{2n} x_{n2} = 1$ ← column 2 of x has to satisfy

$\vdots \qquad \vdots$

$s_{n1} x_{1n} + \dots + s_{nn} x_{nn} = 1$ ← column n of x has to satisfy

all other equations set to 0

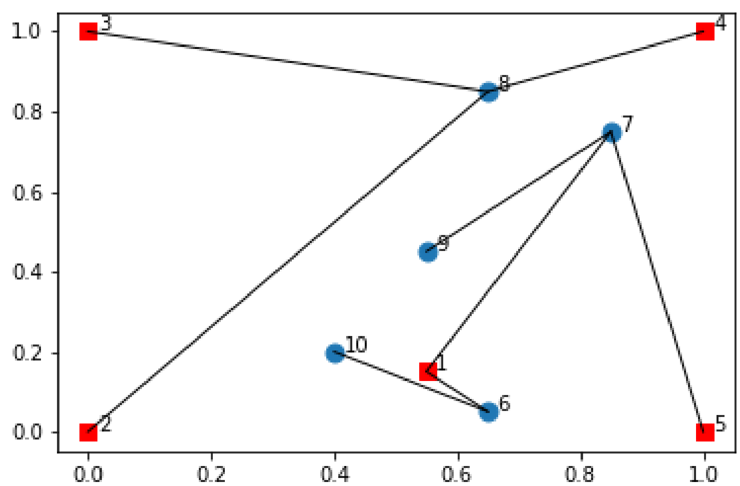
Problem 3 (Least Squares Placement) [18pts]. This problem has parts a–e. We have provided space below each problem for the solution.

The vectors $p_1, \ldots, p_N$, each in $\mathbb{R}^2$ represent the locations of N objects. There are two types of objects: factories and warehouses. The first K objects are factories, whose locations are fixed and given. Our goal in the placement problem to choose the locations of the last $N-K$ objects i.e the warehouses.

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

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

For illustration, see the figure below. The factories are denoted by red squares, and warehouses by blue circles; we want to move the blue circles so that the objective is minimized.



a. In each of the simplified cases below, state what the optimal objective value would be. Further, what assignment to $\{p_{K+1}, \ldots p_N\}$ would achieve this optimal objective? If there are multiple optimal assignments, state any one.

   i. The factories are fixed at $p_{N-K+1} = \cdots = p_{N-1} = p_N = \mathbf{0}$.

   ii. There is $K = 1$ factory. Write your answer in terms of the fixed location p_1.

   iii. All edges are from one warehouse to another. In other words, for any $(i,j) \in E$, both (p_i, p_j) are warehouses. Write your answer in terms of E.

Solution part a.

i. if all the factories are fixed at zero, then the optimal assignment will just be placed at zero also since any other placement would not minimize the distance. Optimal objective value is zero.

ii. If there is only one factory, then the optimal assignment should be placed at that factory since that is the only distance to consider. Optimal objective value is 0. The optimal location is at P1.

iii. The optimal assignment would be Pi = Pj in E because then the optimal objective value would be 0.

b. Now, we move towards solving the problem in the more general case.

Let $\mathcal{D}$ be the Dirichlet energy of the graph, which is defined as follows (more details are given in §7.3 of VMLS, and specifically page 135; you may want to take a look):

> Consider a graph $\mathcal{G} = (\{1, \dots, N\}, E)$ composed of a set of edges E and N nodes $\{1, \dots, N\}$ such that nodes $i, j \in \{1, \dots, N\}$ are connected if and only if there exists an edge $(i, j) \in E$. Let B be the incidence matrix of the graph, and let $w \in \mathbb{R}^N$ be the *potential* for the graph—i.e., w_i is the value of some quantity at node $i \in \{1, \dots, N\}$. The Dirichlet energy is $\mathcal{D}(w) = \|B^\top w\|^2$ and can be equivalently expressed as
>
> $$\mathcal{D}(v) = \sum_{(i,j)\in E} (w_j - w_j)^2$$
>
> which is the sum of the squares of the potential differences of w across all edges in the graph.

Show that the sum of the squared distances between the N factories can be expressed as $\mathcal{D}(u) + \mathcal{D}(v)$, where $u = ((p_1)_1, \dots, (p_N)_1)$ and $v = ((p_1)_2, \dots, (p_N)_2)$ are N-vectors containing the first and second coordinates of the factories, respectively.

Solution part b.

$D(u) + D(v)$ for first and second coordinates

$$= (u_{i_1} - u_{j_1})^2 + \dots + (u_{i_L} - u_{j_L})^2$$
$$+ (v_{i_1} - v_{j_1})^2 + \dots + (v_{i_L} - v_{j_L})^2$$

$$= \left\| \begin{bmatrix} u_{i_1} - u_{j_1} \\ v_{i_1} - v_{j_1} \end{bmatrix} \right\|^2 + \dots + \left\| \begin{bmatrix} u_{i_L} - u_{j_L} \\ v_{i_L} - v_{j_L} \end{bmatrix} \right\|^2$$

$$= \|p_{i_1} - p_{j_1}\|^2 + \dots + \|p_{i_L} - p_{j_L}\|^2$$

c. Express the least squares placement problem as a least squares problem, with variable $x = (u_{1:(N-K)}, v_{1:(N-K)})$. In other words, express the objective above (the sum of squares of the distances across edges) as $\|Ax - b\|^2$, for an appropriate $m \times n$ matrix A and m-vector b. You will find that $m = 2L$.

Hint: You can use the fact that $\mathcal{D}(y) = \left\|B^\top y\right\|^2$, where B is the incidence matrix of the graph.

Solution part c.

$$D(u) + D(v) = \|B^T u\|^2 + \|B^T v\|^2$$

$$\begin{bmatrix} B^T & 0 \\ 0 & B^T \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = 0 \quad \text{where } x_1 = u_{1:N-K}, \; x_2 = v_{1:N-K}$$

$$\begin{bmatrix} B^T & 0 \\ 0 & B^T \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = 0$$

d. Solve the least squares placement problem for the specific problem with $N = 10$, $K = 5, L = 13$, fixed locations

$$p_1 = (0.55, 0.15), \quad p_2 = (0, 0), \quad p_3 = (0, 1), \quad p_4 = (1, 1), \quad p_5 = (1, 0),$$

The edges are:

$$\begin{gathered}(1,6), \quad (2,6), \quad (5,6), \quad (1,7), \quad (4,7), \quad (2,8), \\ (3,8), \quad (3,9), \quad (5,9), \quad (5,10), \quad (7,9), \quad (6,10), \quad (7,10).\end{gathered}$$

Using the provided Python notebook, plot the locations, showing the graph edges as lines connecting the locations. Specifically, solve **Python-P3d** in the provided notebook and print the pdf and attach it at the end of your exam. **You will have to upload the final notebook to Canvas to get credit in addition to attaching a print out pdf to the exam you submit!**

Problem 4 (Recursive Least Squares) [12pts]. Suppose we have data $(x^{(1)}, \dots, x^{(m)})$ and $(y^{(1)}, \dots, y^{(m)})$ and we believe that $y^{(i)} \approx (x^{(i)})^\top \theta$—that is, $y^{(i)}$ is approximately a linear function of $x^{(i)}$. The $x^{(i)} \in \mathbb{R}^n$ are n-dimensional vectors and $y^{(i)} \in \mathbb{R}$ are scalars. The least squares method seeks to minimize

$$J(\theta) = \sum_{k=1}^{m} (y^{(k)} - (x^{(k)})^\top \theta)^2$$

We saw in lecture that

$$\hat{\theta}_m := \left(\sum_{k=1}^{m} x^{(k)} (x^{(k)})^\top \right)^{-1} \sum_{k=1}^{m} x^{(k)} y^{(k)}$$

minimizes $J(\theta)$. There are three parts to the problem and on the next page put your solutions to parts a and b. For part c, print your pdf and attach it at the end of your exam. You will also need to provide the ipynb of your solutions.

a. Define $X_m^\top = [x^{(1)}\ x^{(2)}\ \cdots\ x^{(m)}]$ and $Y_m = [y^{(1)}\ y^{(2)}\ \cdots\ y^{(m)}]^\top$. Show that

$$\hat{\theta}_m := (X_m^\top X_m)^{-1} X_m^\top Y_m$$

minimizes $\|Y_m - X_m \theta\|^2$ by taking the derivative of $\|Y_m - X_m \theta\|^2$ and setting it to zero.

b. Now suppose one more data pair $(x^{(m+1)}, y^{(m+1)})$ becomes available and define

$$X_{m+1}^\top = [X_m^\top\ x^{(m+1)}] \quad \text{and} \quad Y_m = [Y_m^\top\ y^{(m+1)}]^\top$$

Hence, the new least squares solution is

$$\hat{\theta}_{m+1} := (X_{m+1}^\top X_{m+1})^{-1} X_{m+1}^\top Y_{m+1} = R_{m+1}^{-1} \sum_{k=1}^{m+1} x^{(k)} y^{(k)} \tag{1}$$

where

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

Show that the least squares solution can be obtained by the following recursive procedure:

$$\hat{\theta}_{m+1} = \hat{\theta}_m + R_{m+1}^{-1} x^{(m+1)} (y^{(m+1)} - (x^{(m+1)})^\top \hat{\theta}_m) \tag{2}$$

$$R_{m+1} = R_m + x^{(m+1)} (x^{(m+1)})^\top \tag{3}$$

That is, show that these equations hold given the definition of R_m and the expression for the least squares solution.

Hint: First, verify (3) holds given the definition of R_{m+1}. Then, to verify (2), take the expression in (1) for $\hat{\theta}_{m+1}$, and expand the sum $\sum_{k=1}^{m+1} x^{(k)} y^{(k)} = x^{(m+1)} y^{(m+1)} + \sum_{k=1}^{m} x^{(k)} y^{(k)}$. Now, use (3) and the expression you have for $\hat{\theta}_m$ to show (2) holds.

c. Implement recursive least squares. In the provided Python notebook, solve **Python-P4**. **You will have to upload the final notebook to Canvas to get credit in addition to attaching a print out pdf to the exam you submit!**

Solution.

a. **Provide your solution to part a. here.**

$$\frac{d}{d\theta}\left\| Y_m - X_m\theta \right\|^2 = -2\left(Y_m - X_m\theta\right)X_m$$

set derivative equal to zero

$$-2(Y_m - X_m\hat{\theta})X_m = 0$$

$$-2X_m^T(Y_m - X_m\hat{\theta}) = 0$$

$$-2X_m^T Y_m + 2X_m^T X_m\hat{\theta} = 0$$

$$X_m^T Y_m = X_m^T X_m\hat{\theta}$$

$$\Rightarrow \hat{\theta} = (X_m^T X_m)^{-1}(X_m^T Y_m)$$

b. **Provide your solution to part b. here.**

$$\hat{\theta}_{m+1} = R_{m+1}^{-1} \sum_{k=1}^{m+1} x^k y^k$$

$$= R_{m+1}^{-1} \left(x^{m+1} y^{m+1} + \sum_{k=1}^{m} x^k y^k \right)$$

$$= \hat{\theta}_m + R_{m+1}^{-1} \left(x^{m+1} y^{m+1} - (R_{m+1} - R_m)\hat{\theta}_m \right)$$

$$= \hat{\theta}_m + R_{m+1}^{-1} \left(x^{m+1} y^{m+1} - (R_{m+1} - R_m) R_m^{-1} \sum_{k=1}^{m} x^k y^k \right)$$

$$= \hat{\theta}_m + R_{m+1}^{-1} x^{m+1} \left(y^{m+1} - (x^{m+1})^T \theta_m \right)$$

Problem 5 (Least Norm Solution) [6pts]. In class we saw how to obtain the least squares approximate solution to $Ax = b$ corresponding to an over-determined set of equations—that is, where $A \in \mathbb{R}^{m\times n}$ with $m > n$ ("tall matrix"). In this case, there is rarely an exact solution to $Ax = b$, and instead we find an approximate solution by minimizing $\|Ax - b\|^2$.

If on the other hand A is a "wide matrix", meaning $n > m$, then the set of equations is under-determined. This means there are more unknowns than equations and hence, potentially many solutions. In this case we select, amongst the solutions, the one with the smallest norm—i.e., the least norm solution. That is, we seek $\hat{x}$ such that $A\hat{x} = b$ and $\|\hat{x}\| \le \|x\|$ for all other x such that $Ax = b$.

Consider an under-determined system of equations $Ax = b$ where $b \in \mathbb{R}^n$ and $A \in \mathbb{R}^{m\times n}$ are given with $n > m$. Suppose that A is full rank and $b \in \text{range}(A)$—this ensures there is at least one solution. Show that $\hat{x} = A^\top(AA^\top)^{-1}b$ is the least norm solution to $Ax = b$. In particular, letting $\mathcal{S} = \{x \in \mathbb{R}^n \mid Ax = b\}$ be the set of solutions to $Ax = b$, you need to show that

$$\|\hat{x}\| \le \|x\| \quad \forall\, x \in \mathcal{S}.$$

Solution.

Let the least squares solution be $\hat{x} = A^T(AA^T)^{-1}b$

We can say AA^T is invertible because A is full rank.

Goal: minimize $\|x\|$ considering $Ax = b$

assume $A(x-\hat{x}) = 0$ then

$$\hat{x} = (x-\hat{x})^T A^T (AA^T)^{-1} b$$

$$(x-\hat{x})^T \hat{x} = (x-\hat{x})^T A^T (AA^T)^{-1} b$$

$$= \underbrace{A(x-\hat{x})^T}_{=0} (AA^T)^{-1} b$$

this means $(x-\hat{x})^T \hat{x} = 0$ and thus orthogonal

In class we talked about the orthogonality principle.

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 $p_1, \ldots, p_N$, 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 $p_{K+1}, \ldots, p_N$ 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 $p_1 \ldots p_K$, we can frame our task as solving the following optimization:

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

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:

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

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

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 computer the error for each iterate:

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

where $\theta^{\mathtt{lsq}}$ 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]:
```python
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]:
```python
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]:
```python
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
```

```python
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])
```

[ ]:

```python

```