EE263 mid-term exam, November 2022

- This is a 24-hour take-home midterm. Please turn it in on Gradescope. Be aware that you *must* turn it in within 24 hours of downloading it. After that, Gradescope will not let you turn it in and we cannot accept it.
- You may use any books, notes, or computer programs. You may not discuss the exam or course material with others, or work in a group.
- The exam should not be discussed at all until 11/7 after everyone has taken it.
- If you have a question, please submit a private question on Ed, or email the staff mailing list. We have tried very hard to make the exam unambiguous and clear, so unless there is a mistake on the exam we're unlikely to say much.
- We expect your solutions to be legible, neat, and clear. Do not hand in your rough notes, and please try to simplify your solutions as much as you can. We will deduct points from solutions that are technically correct, but much more complicated than they need to be.
- Please check your email during the exam, just in case we need to send out a clarification or other announcement.
- Start each question on a new page. Correctly assign pages to problems in gradescope. We may take off points if a submission does not do so.
- We will be more thorough grading the midterm than with the homeworks. Please show the work you do, as it especially helps us give partial credit.
- When a problem involves some computation (say, using Julia), we do not want just the final answers. We want a clear discussion and justification of exactly what you did as well as the final numerical result.
- Because this is an exam, **you must turn in your code**. Include the code in your pdf submission. We reserve the right to deduct points for missing code.
- In the portion of your solutions where you explain the mathematical approach, you *cannot* refer to Julia operators, such as the backslash operator. (You can, of course, refer to inverses of matrices, or any other standard mathematical constructs.)
- Some of the problems require you to download data or other files. These files can be found at the URL

  `http://ee263.stanford.edu/mid22.html`
- Good luck!

1. **Recursive estimation.** A piecewise constant signal is filtered by convolving it with a smooth function. We start with $x \in \mathbb{R}^n$, and upsample (repeat values) to create $u \in \mathbb{R}^m$. The upsampling repeats each value k times, so that $m = kn$. The constant signal $u \in \mathbb{R}^m$ is given by

$$u_i = x_j \text{ for } k(j-1) < i \le kj$$

The signal u is convolved with a smooth function r, given by

$$r_j = \exp(-j^2/\sigma^2)$$

which is defined for $-q \le j \le q$. The convolution operation generates output $y \in \mathbb{R}^m$, given by

$$y_i = w_i + \sum_{j=\max(1,i-q)}^{\min(m,i+q)} r_{i-j} u_j \tag{1}$$

where w is random measurement noise. We have $n = 10$, $k = 5$, $\sigma = 2$, $q = 10$. We will use regularization parameter $\mu = 0.1$. The file `recursive.json` contains x, y and w, which satisfy equation (1).

a) Find matrix C such that $u = Cx$.

b) Find matrix B such that $y = Bu + w$.

c) Let $A = BC$. Find x^{reg}, the regularized least-squares estimate of x given y. That is, x^{reg} is the x that minimizes

$$\|Ax - y\|^2 + \mu\|x\|^2$$

Plot x^{reg} and x on the same plot. (*i.e.*, plot x_i versus i)

d) We would like to use a recursive method to compute the regularized least-squares estimate. Recall the usual recursive-least-squares algorithm:

```
$P(0) = 0 \in \mathbb{R}^{n \times n}$
$q(0) = 0 \in \mathbb{R}^n$
for $i = 0, 1, \dots,$
    $P(i+1) = P(i) + a_{i+1}a_{i+1}^\mathsf{T}$
    $q(i+1) = q(i) + y_{i+1}a_{i+1}$
```

where a_i^T is the i'th row of A, and y_i is the corresponding ith measurement. Then the estimate based on $y_1, \dots, y_i$ is $x_{\text{ls}}(i) = P(i)^{-1}q(i)$.

Explain how to modify this algorithm to recursively compute the regularized least-squares estimate.

e) Apply your algorithm to the given data. Plot your estimate when $i = 18$ and when $i = 30$.

f) After applying your algorithm, when $i = m$, you will have computed the same regularized least-squares estimate you did in part (c), but in a different way. At this point, you realize that the data $y_1, \ldots, y_{20}$ was incorrect, and you would like to remove it from your estimate. However, you have already thrown away $y_{21}, \ldots, y_m$. Give an algorithm to adjust your estimate to remove the effect of measurments $y_1, \ldots, y_{20}$. Plot the resulting estimate of x. Note that you only have access to $y_1, \cdots, y_{20}$, the final P and q from part (d), and $a_1, \cdots, a_{20}$.

Solution. Here is the solution.

a) We have,
$u_1 = u_2 = ... = u_k = x_1$
$u_{k+1} = u_{k+2} = ... = u_{2k} = x_2$
...
$u_{(n-1)k+1} = u_{(n-1)k+2} = ... = u_m = x_n$.
From this system of equation, we can construct C so that similar rows repeat k times,

$$C = \begin{bmatrix} 1 & 0 & 0 & ... & 0 \\ 1 & 0 & 0 & ... & 0 \\ \vdots & \vdots & \vdots & & \vdots \\ 0 & 1 & 0 & ... & 0 \\ 0 & 1 & 0 & ... & 0 \\ \vdots & \vdots & \ddots & & \vdots \\ \vdots & \vdots & & \ddots & \vdots \\ 0 & 0 & 0 & ... & 1 \end{bmatrix}$$

b) We can construct the convolution matrix using the following exuations for y_i,
$y_1 = r_0 * u_1 + r_{-1} * u_2 + ... + r_{-q} * u_{q+1}$
$y_2 = r_1 * u_1 + r_0 * u_2 + ... + r_{-q} * u_{q+2}$
...
$y_m = r_q * u_q + r_{q-1} * u_{q+1} + ... + r_0 * u_m$
From this system of equation, we can construct C so that similar rows repeat k

times,

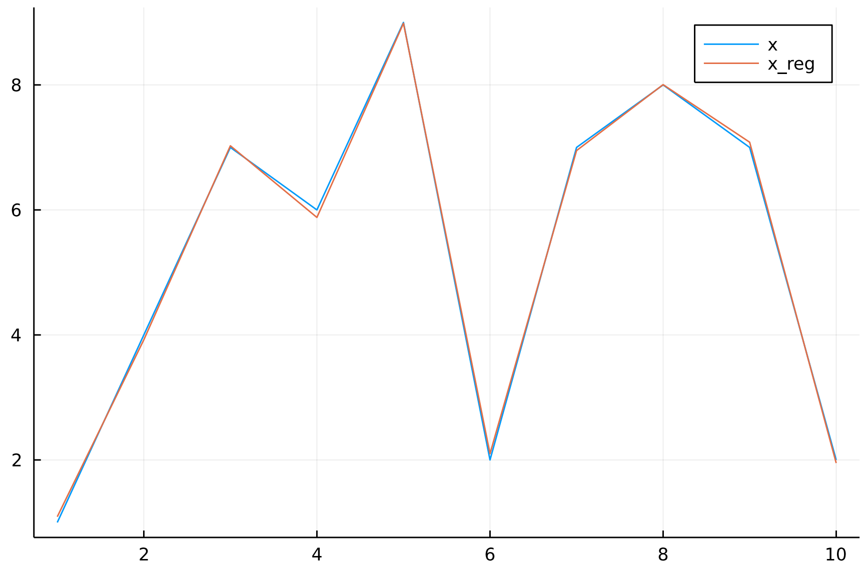
$$B = \begin{bmatrix} r_0 & r_{-1} & r_{-2} & \dots & r_{-q} & 0 & \dots & 0 \\ r_1 & r_0 & r_{-1} & \dots & r_{-q+1} & r_{-q} & \dots & 0 \\ r_2 & r_1 & r_0 & \dots & r_{-q+2} & r_{-q+1} & \dots & 0 \\ \vdots & \vdots & \vdots & \dots & & & & \vdots \\ r_q & r_{q-1} & r_{q-2} & \dots & r_0 & r_{-1} & \dots & 0 \\ \vdots & \vdots & \dots & & \ddots & & & \vdots \\ \vdots & \vdots & & & & \ddots & & \vdots \\ \vdots & \vdots & & & & & \ddots & \vdots \\ 0 & 0 & 0 & \dots & \dots & \dots & r_1 & r_0 \end{bmatrix}$$

For the above given values,

$$B = \begin{bmatrix} 1.0 & 0.778801 & 0.367879 & \dots & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.778801 & 1.0 & 0.778801 & & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.367879 & 0.778801 & 1.0 & & 0.0 & 0.0 & 0.0 & 0.0 \\ \vdots & & & \ddots & & & & \vdots \\ 0.0 & 0.0 & 0.0 & & 0.0 & 1.0 & 0.778801 & 0.367879 \\ 0.0 & 0.0 & 0.0 & & 0.0 & 0.778801 & 1.0 & 0.778801 \\ 0.0 & 0.0 & 0.0 & \dots & 0.0 & 0.367879 & 0.778801 & 1.0 \end{bmatrix}$$

c) Regularized least-square solution,

$$x = (A^T A + \mu I)^{-1} A^T y$$



The code is given below.

d) In general, we can compute $x_{ls}(m) = (\sum_{i=1}^m \tilde{a}_i\tilde{a}_i^T)^{-1}\sum_{i=1}^m y_i\tilde{a}_i$ recursively using the method given.

Here, we have

$$x = (A^TA + \mu I)^{-1}A^Ty$$

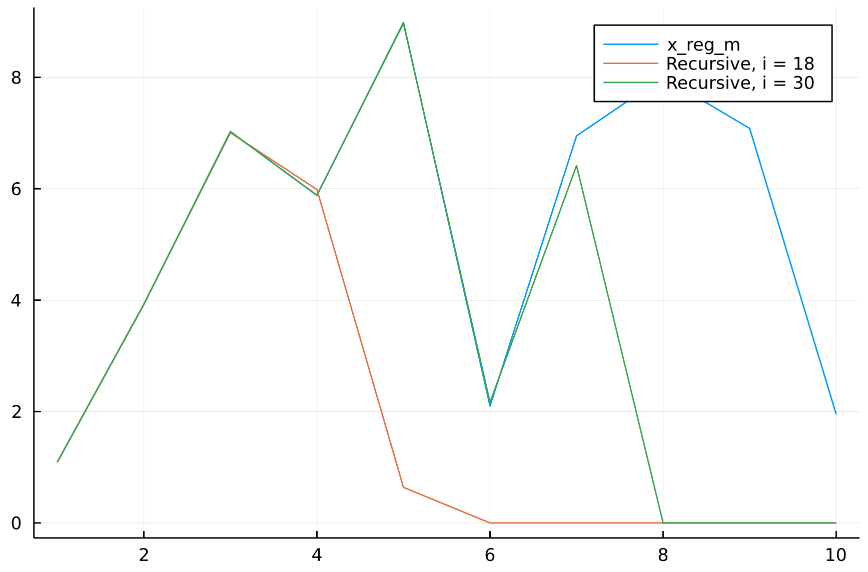
which can be written as

$$x_{ls}(m) = (\sum_{i=1}^{m} \tilde{a}_i\tilde{a}_i^T + \mu I)^{-1}\sum_{i=1}^{m} y_i\tilde{a}_i$$

So, we could modify the initialization, $P(0) = \mu I$.

NOTE: Modifying the update step to include $\mu/m * I$ is not correct as that will vary with i and does not include the corresponding regularization factor beyond the i. Modifying the update step so that any 10 steps can update μ is also wrong as it depends on i again.

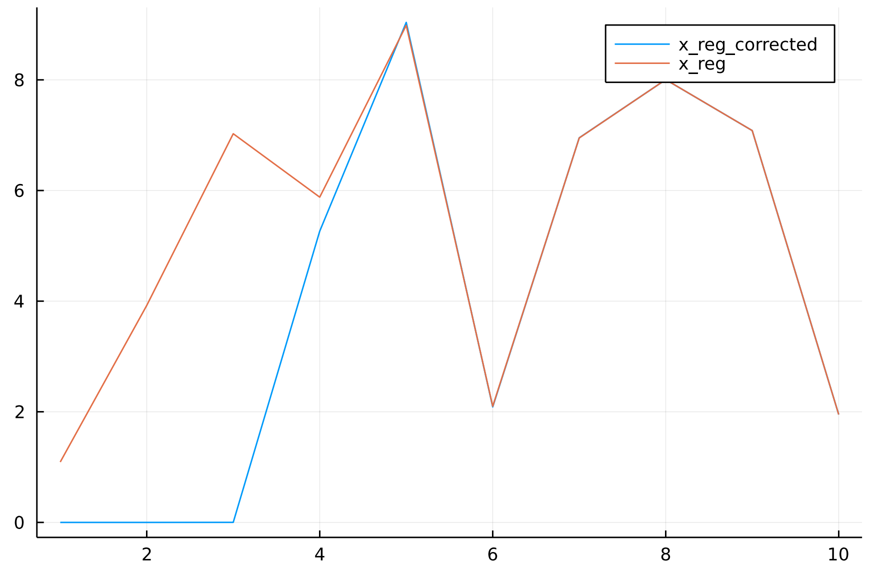
e) Recursive method,



The code is given below.

f) We can subtract the 20 update steps corresponding to y_1 to y_{20} from our final P and Q before computing x_reg

Resulting estimate of x,



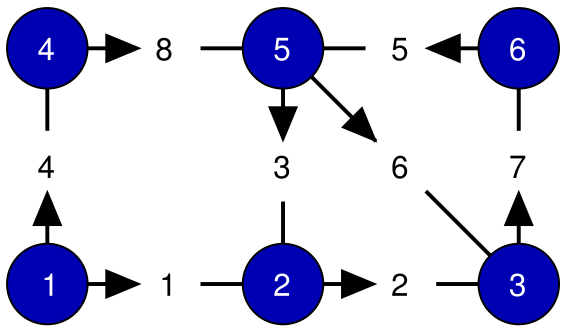
Note: Regularization terms must not be subtracted.

```
using LinearAlgebra
using Plots
using ToeplitzMatrices

include("readclassjson.jl")
data = readclassjson("recursive.json")

w = data["w"]
x = data["x"]
y = data["y"]

n = 10
k = 5
sigma = 2
q = 10
mu = 0.1
m = n*k

# a) Matrix C

C = zeros(k*n, n)

for i = 1:k*n
    C[i, floor(Int, (i-1)/k) + 1] = 1
```

```
end
display(C)

# b) Matrix B

r(j) = exp(-j^2/sigma^2)

row_1 = zeros(n*k)
r_j = [r(j) for j = 0:-1:-q]
row_1[1:length(r_j)] = r_j

col_1 = zeros(n*k)
c_j = [r(j) for j = 0:q]
col_1[1:length(c_j)] = c_j

B = Toeplitz(row_1, col_1)
display(B)

# c) x_reg

A = B*C

x_reg = inv(A'*A + mu*I(n))*A'*y

plot(x, label = "x")
plot!(x_reg, label = "x_reg")
savefig("../graphics/Reg_LS.pdf")

# e) Recursive method

# Method 1
P = mu * I(n)
q = zeros(n)
for i = 1:k*n
    P += A[i, :]*A[i, :]'
    q += y[i]*A[i, :]
    if i == 18
        global x_reg_18 = inv(P)*q
    elseif i == 20
        global P_20 = P
        global q_20 = q
    elseif i == 30
        global x_reg_30 = inv(P)*q
    end
end
x_reg_m = inv(P)*q

plot(x_reg_m, label = "x_reg_m")
plot!(x_reg_18, label = "Recursive, i = 18")
plot!(x_reg_30, label = "Recursive, i = 30")
savefig("../graphics/Recursive_Reg_LS.pdf")

# f) Remove incorrect measurements

for i = 1:20
    global P -= A[i, :]*A[i, :]'
    global q -= y[i]*A[i, :]
end
x_reg_corrected = inv(P)*(q)
plot(x_reg_corrected, label = "x_reg_corrected")
plot!(x_reg, label = "x_reg")
savefig("../graphics/Corrected_Reg_LS.pdf")
```

2. Synchronicity.

a) The following graph shows a cluster of n machines in a data center, arranged in the form of a directed graph. Each machine has a clock, and communicates with its neighbors, to determine the *clock difference* between them. Specifically, machine i has a clock which reads c_i, in seconds. For each edge $i \to j$, we measure the clock difference $c_i - c_j$. For simplicity, we assume that this clock difference can be (approximately) measured by accounting for the known communication latency between the machines.



Edges are numbered $1, \ldots m$. Let y_e be the clock difference measured along edge e. Then we have $y = B^\mathsf{T} c$ for some matrix B. Find B for the graph shown above.

b) Since we are only measuring clock *differences*, we do not expect to be able to determine c unambiguously. For the above graph, given a measurement $y \in \mathbb{R}^m$, what is the set of c consistent with this measurement?

c) A *cycle* in the graph is a sequence of distinct vertices, such at $1, 2, 5, 4, 1$ which start and end at the same vertex, and form a loop. The direction of the arrow is ignored in a cycle, all that matters is that there is an edge between successive vertices. For example, $2, 3, 5, 2$ is a cycle but $1, 2, 4, 1$ is not.

Show that the sum of clock differences around a cycle is zero, for a general graph.

d) Given a cycle in the graph, show how to construct a vector x in the nullspace of B. Hence construct a matrix K with the maximum number of columns such that

$$BK = 0 \qquad \text{and} \qquad \text{null}(K) = 0$$

and all entries of K are -1, 0 or 1.

e) We measure the following clock differences

$$y = (-0.56, -0.7, 1.13, -1.12, -1.45, 0.43, 1.02, -0.57)$$

Show that these values are *consistent* difference measurements; that is, there exists a vector of clock values c such that $y = B^{\mathsf{T}}c$. Do this without solving the least squares for c and then comparing y and $B^{\mathsf{T}}c$.

f) Assume the first clock has $c_1 = 0$. Given y in the previous part, find $c_2, \ldots, c_n$.

g) Now consider the case where we cannot measure clock differences perfectly; we measure instead

$$y = B^{\mathsf{T}}c + w$$

where w is some small error. We would like to find an estimate of the clocks c. To do this, we decide to solve

$$\begin{array}{ll} \text{minimize} & \|y - B^{\mathsf{T}}c\| \\ \text{subject to} & c_1 = 0 \end{array}$$

Give an algorithm for doing this.

h) We make a noisy measurement of the clock differences

$$y = (0.487, -0.128, 0.789, 0.245, 0.184, 0.506, -0.839, -0.647)$$

Using your algorithm from the previous part, estimate c.

Solution. Here is the solution.

a) The latency across an edge is measured as the clock difference between the originating node of the edge and the terminating node. If we let y_e be the clock difference on edge e, and c_i be the clock value on node i, then we get the following system of equations:

$$\begin{aligned} y_1 &= c_1 - c_2, \\ y_2 &= c_2 - c_3, \\ y_3 &= c_5 - c_2, \\ y_4 &= c_1 - c_4, \\ y_5 &= c_6 - c_5, \\ y_6 &= c_5 - c_3, \\ y_7 &= c_3 - c_6, \\ y_8 &= c_4 - c_5. \end{aligned}$$

We let y be the vector of clock differences and c be the vector of clock times. We can construct some B such that $y = B^\top c$ from the system of equations above. We get that

$$B^\top = \begin{bmatrix} 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 1 \\ 0 & 0 & -1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & -1 \\ 0 & 0 & 0 & 1 & -1 & 0 \end{bmatrix}.$$

Thus,

$$B = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ -1 & 1 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & -1 & 1 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & -1 & 1 & 0 & -1 \\ 0 & 0 & 0 & 0 & 1 & 0 & -1 & 0 \end{bmatrix}.$$

b) Since we are only measuring the clock differences, we can only determine the clock times up to a uniform constant error across all of the clocks. To illustrate this, notice that each y_e is the difference of c. Thus if we set c to be a vector of all ones, then y_e is 0 for all edges e. Thus, any vector with uniform elements will be in the nullspace of $B^\top$. We deduce that if c is a vector of clock times, u is a vector of uniform elements, then $B^\top c = B^\top(c + u)$.

c) Suppose we have a cycle of n vertices. Without loss of generality (since we can arbitrarily re-number the vertices and edges as long as the graph is preserved), we assume edge 1 goes from vertex 1 to 2, edge 2 goes from vertex 2 to 3, so on, and edge n goes from vertex n to 1. It follows that

$$\begin{aligned}\sum_{i=1}^{n} y_i &= \sum_{i=1}^{n-1}(c_i - c_{i+1}) + (c_n - c_1) \\ &= \sum_{i=1}^{n} c_i + \sum_{i=1}^{n} c_i \\ &= 0.\end{aligned}$$

d) Notice that B is a matrix such that b_{ij} is 1 if edge j originates in vertex i, -1 if edge j terminates in vertex i, and 0 otherwise. So when we consider Bv for any vector v, v is a vector of weights we assign to the edges, and the ith element of Bv is the difference between the the sum of the weights of edges coming from vertex i and the sum of the weights of edges going to vertex i.

It follows from part c) that if we have a cycle in the graph with some vertices V_C and some edges E_C that if we make a vector v such that $v_e = 1$ for $e \in E_C$ and $v_e = 0$ otherwise, then $c^\top Bv$ will be zero for any clock timings c. This is because $c^\top Bv$ takes the sum of the clock differences on all of the edges indicated in v. Since the edges indicated in v form a cycle, this sum is zero for any timings. This in turn implies Bv is the zero vector.

Now we know that vector v is in the nullspace of B if it is the indicator vector of a cycle as described above. Thus we can construct K such that the columns of K are the indicator vectors of cycles in the graph. This will ensure that $BK = 0$. If we have no repeated columns or zero columns, then null(K) will be zero. The graph has two cycles. One consists of routes $5, 6, 7$ while the other consists of routes $3, 2, 7, 5$. Finally, notice that if a vector v assigns an edge a negative weight, it flips the terms of the clock difference, and is equivalent to assigning an edge going the opposite direction a positive weight. Thus, we can "create a new cycle" by assigning edge 1 a weight of -1 and assigning edges $3, 4, 8$ a weight of 1. Thus we can construct

$$K = \begin{bmatrix} 0 & 0 & -1 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

We know from part a) that we can uniquely determine the clock timings up to a constant offset, so the only vectors in the null space of $B^\top$ are the vectors with uniform elements. Thus, $B^\top$ is rank 5, and it follows that B is rank 5. Since K is rank 3 as it clearly has 3 independent columns (each column has a 1 in a row where the others have zeroes), we can deduce that K is a basis of the nullspace of K. This holds as the nullspace of B must be rank 3, K has 3 independent columns, and the columns of K are in the nullspace of B. We can confirm this in Julia by checking `rank(B)`.

e) We know that the columns of K are a basis of the null space of B. The fundamental theorem of linear algebra gives us that the null space of B is the orthogonal complement of the range of $B^\top$. Thus, if the given clock difference vector is orthogonal to the columns of K, then it is in the range of $B^\top$ and is thus consistent. We can check this by taking $K^\top y$. In Julia, we compute the result to be the zero vector, so y is consistent.

f) If we assume that $c_1 = 0$, then we can remove the first column from $B^\top$ to get a matrix $B_0^\top$, remove the first element from c to get the vector c_0, and solve the least squares problem for $y = B_0^\top c_0$. This works since y is a combination of the columns of $B^\top$ weighted by c. If $c_0 = 0$, then the first column of $B^\top$ contributes nothing towards y, and it can be removed. Additionally, we know that the clock timings are uniquely determined up to a constant offset. If we set $c = 0$, we fix that constant offset, so the clock timings are fixed. We get that

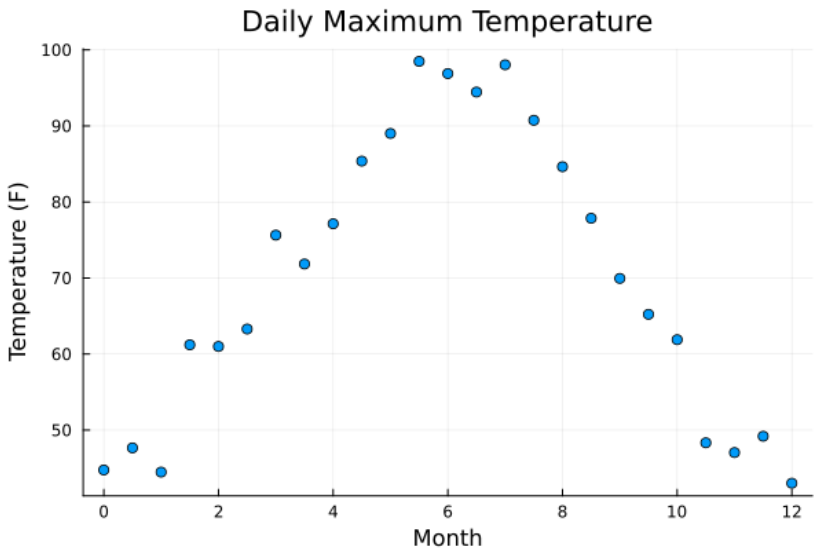
$$c_1 = 0, \quad c_2 = 0.56, \quad c_3 = 1.26, \quad c_4 = 1.12, \quad c_5 = 1.69, \quad c_6 = 0.24.$$

g) Since we are fixing $c_1 = 0$ still, we continue to use $B_0^\top$ and c_0. Since y is not exactly in the range of $B^\top$, and $B_0^\top$ is full rank, we can use the left inverse of $B_0^\top$ to find the best estimates c. In execution, this is exactly the same as what we did in the previous part.

h) We execute the method from the previous two parts for the given y and see that

$$c_1 = -0, \quad c_2 = -0.4485, \quad c_3 = -0.2666, \quad c_4 = -0.2835, \quad c_5 = 0.3250, \quad c_6 = 0.5407.$$

3. Fitting a Piecewise Linear Function to Data. Last year, we sampled the maximum daily temperature outside Packard twice a month. Looking at the plotted results, we believe there's a clear trend in the temperatures over the time of year. We have collected $n = 25$ datapoints. Each data point consists of two values: x and y. The x value ranges from 0 to 12 and describe when the data was collected in months since the start of the year. The y value is the recorded temperature in Fahrenheit. We have two data sets, a training set and a test set, in the file `tempdata.json`.



In order to better describe the relationship between time of year and daily maximum temperature, we will fit a piecewise linear function g to the training data set. We will use $m+1$ piecewise affine functions $f_0, f_1, \ldots, f_m$, and approximate the data by the function g, which is a linear combination of them.

$$g(x) = \sum_{j=0}^{m} \alpha_j f_j(x)$$

Here $f_0 = 1$ and for $j = 1, \ldots, m$ we have

$$f_j(x) = \begin{cases} 0 & \text{if } x < (j-1)\frac{12}{m} \\ \frac{mx}{12} - (j-1) & \text{if } (j-1)\frac{12}{m} \leq x \leq j\frac{12}{m} \\ 1 & \text{if } j\frac{12}{m} < x \end{cases}.$$

The functions f_i are very simple piecewise linear step functions. We recommend graphing a couple for variable m and j to get intuition for what these functions are.

a) Our objective is to select weights $\alpha_0, \ldots, \alpha_m$ to minimize

$$\sum_{i=1}^{n} \|y_i - g(x_i)\|_2^2.$$

Express this objective in the form

$$\text{minimize } \|y - F\alpha\|_2^2$$

for some known vector y, known matrix F, and unknown vector α.

b) Suppose n is a multiple of m, and there are $n+1$ data points that are evenly spaced in x from 0 to 12 inclusive, so that gap between points is $12/n$. Show that the matrix F is full rank.

c) Use Julia to solve this problem for $m = 3, 6, 12, 24$ for the training data x^{train} and y^{train}. Plot the data points (x, y) along with the fitted function g for each value of m.

d) Plot the minimal squared 2-norm error ($\|y - F\alpha\|_2^2$) for $m = 3, 6, 12, 24$. Is there a point where adding more complexity to the model (increasing m) offers clearly diminishing returns?

e) We now turn to validation of the fit. We have an additional data set, x^{test} and y^{test}, which we will use to test the accuracy of our model. The test error is

$$J^{\text{test}} = \sum_{i=1}^{n} \|y_i^{\text{test}} - g(x_i^{\text{test}})\|_2^2.$$

Here g is the function you found in part (c) above. Plot the test error J^{test} versus m for $m = 3, 6, 12, 24$. Note that this does not involve recomputing α. What does this say about your answer to part (d).

Solution.

a) We want to minimize the sum

$$\sum_{i=1}^{n} \|y_i - g(x_i)\|_2^2$$

Substituting the given definition of g into the sum yeilds the expression

$$\sum_{i=1}^{n} \|y_i - \sum_{j=0}^{m} \alpha_j f_j(x_i)\|_2^2$$

So we define

$$y = \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix} \quad F = \begin{bmatrix} f_0(x_1) & f_1(x_1) & \dots & f_m(x_1) \\ f_0(x_2) & f_1(x_2) & \dots & f_m(x_2) \\ \vdots & \vdots & \ddots & \vdots \\ f_0(x_n) & f_1(x_n) & \dots & f_m(x_n) \end{bmatrix} \quad \alpha = \begin{bmatrix} \alpha_0 \\ \vdots \\ \alpha_m \end{bmatrix}$$

Then the problem is

$$\text{minimize } \|y - F\alpha\|_2^2$$

b) Suppose $n = am$ where a is a positive integer. Then the matrix $F \in \mathbb{R}^{(n+1)\times(m+1)}$ is

$$F = \begin{bmatrix} 1 & 0 & 0 & \dots & & \\ 1 & q & 0 & \dots & & \\ 1 & 1 & q & 0 & & \\ \vdots & & & & & \\ & & & & & \\ 1 & 1 & \dots & & 1 & q \end{bmatrix}$$

where $q \in \mathbb{R}$ is given by

$$q = \begin{bmatrix} 1/a \\ 2/a \\ \vdots \\ 1 \end{bmatrix}$$

We can see that $\text{null}(F) = \{0\}$ as follows. Let $z = Fx$, partitioned compatibility with F, so that

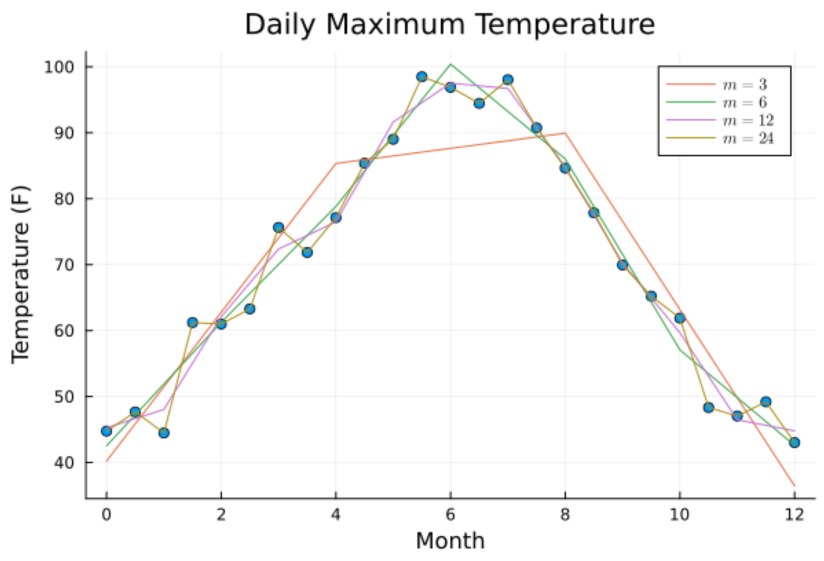
$$z = \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_m \end{bmatrix}$$

with $z_1 \in \mathbb{R}$ and $z_i \in \mathbb{R}^a$ for $i > 1$. Suppose $z = Fx = 0$, then since $z_1 = 0$ we have $x_1 = 0$. Then

$$z_2 = x_1 + qx_2$$

and since $z_2 = 0$ and $x_1 = 0$ we must have $x_2 = 0$. Continuing in this way we see that the only solution to $Fx = 0$ is $x = 0$, and so $\text{null}(F) = \{0\}$.

c) For this part, students simply need to implement the matrices and vectors they discovered in part a) and solve for α via least squares in Julia. The correct plot for this part is as follows.

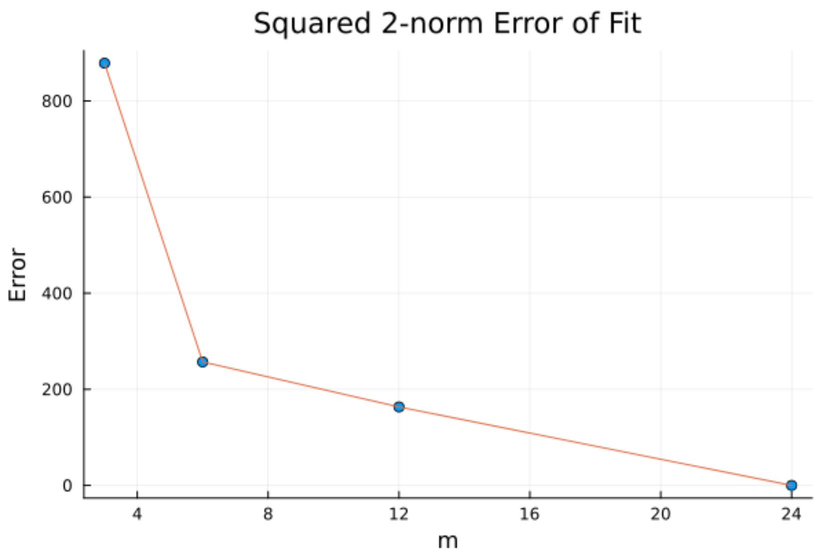


The following code can be used to generate this plot in Jupyter Notebooks with Julia.

```
#Block 1: Imports
using LinearAlgebra
using Random, Distributions
using Plots
using LaTeXStrings
include("readclassjson.jl");

#Block 2: Data Loading
data = readclassjson("../data/tempdata.json")
x_train = data["x_train"]
y_train = data["y_train"]
scatter(x_train, y_train, label=false, xticks=[0,2,4,6,8,10,12],
title="Daily Maximum Temperature", ylabel="Temperature (F)", xlabel="Month");

#Block 3: Solving for alpha
n = 25
ms = [3,6,12,24]
error = []

for m=ms
fs = []
```

```
for j=0:m
function f(x)
if x < 12*(j-1)/m
return 0.0
elseif x > 12*j/m
return 1.0
else
return m*x/12-(j-1)
end
end

push!(fs, f)
end

G = reshape([fs[j](x_train[i]) for j=1:m+1 for i=1:n],n,m+1)
alpha = G \ y_train
y_fit = G * alpha
push!(error, norm(y_train-y_fit)^2)
plot!(x_train, G * alpha, label=LaTeXString("\$m = $m\$"))
end

plot!()
```

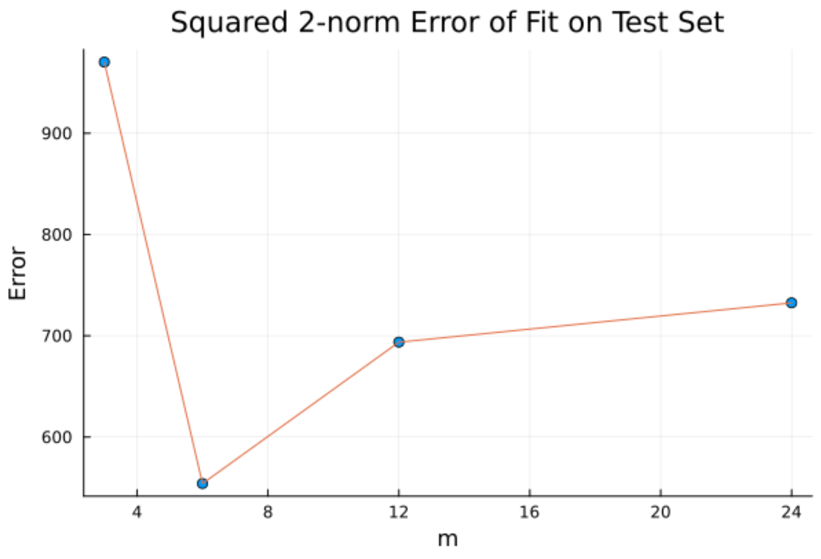
d) For this part, students should comment that diminishing returns takes effect at $m = 6$. We see the error drops tremendously as m is raised from 3 to 6. However, doubling m to 12 and again to 24 offers a much shallower reduction in the error. The correct plot for this part is as follows.



It's expected that students simply modify their code from the previous part to also record the squared 2-norm error between the fitted function and the data. The following code can generate this plot.

```
#Block 4: Plotting the error
scatter(ms, error, label=false, xticks=[0,4,8,12,16,20,24],
title=L"Squared $2$-norm Error of Fit", ylabel="Error", xlabel=L"m")
plot!(ms, error, label=false)
```

e) For this part, students should see that the piecewise linear model clearly overfits after $m = 6$. We see the error still decreases steeply as m is raised from 3 to 6. Where the error previously dropped shallowly though, the error now rises again as m is raised from 6 to 12 and again when m is raised to 24. This example demonstrates that adding more components to the model might increase the fit of the model to the data it is being fit to, but it doesn't necessarily increase the model's accuracy to the underlying phenomenon. Students should make some comment about the overfit demonstrating that the point of diminishing returns may also indicate the point where the model beings to overfit to the data. The correct plot for this part is as follows.



Now, students need to test the fitted function and α from the previous part against the 24 new test points. It is critical that students do not refit g to the new function. The following code can generate this plot.

```
#Block 5: Calculating the test error
x_test = data["x_test"]
```

```
y_test = data["y_test"]
test_error = []

for m=ms
fs = []

for j=0:m
function f(x)
if x < 12*(j-1)/m
return 0.0
elseif x > 12*j/m
return 1.0
else
return m*x/12-(j-1)
end
end

push!(fs, f)
end

G = reshape([fs[j](x_train[i]) for j=1:m+1 for i=1:n],n,m+1)
G_test = reshape([fs[j](x_test[i]) for j=1:m+1 for i=1:n-1],n-1,m+1)
alpha = G \ y_train
y_fit = G * alpha
y_fit_test = G_test * alpha
push!(test_error, norm(y_test-y_fit_test)^2)
end

scatter(ms, test_error, label=false, xticks=[0,4,8,12,16,20,24],
title=L"Squared $2$-norm Error of Fit on Test Set", ylabel="Error", xlabel=L"m")
plot!(ms, test_error, label=false)
```

4. Some true or false questions. For each of the statements below, state whether it is true or false. If true, give a brief one-sentence explanation why. If false, give a counterexample.

a) If $f : \mathbb{R}^n \to \mathbb{R}^n$ is a linear function and $A \in \mathbb{R}^{n\times n}$, then $f(Ax) = Af(x)$.

b) There exists a matrix $A \in \mathbb{R}^{m\times n}$ and a vector $x \in \mathbb{R}^n$ such that $A^\mathsf{T}Ax = 0$ but $Ax \neq 0$.

c) If $u, v, w \in \mathbb{R}^n$, and $\operatorname{rank}\begin{bmatrix} u & v & w \end{bmatrix} = 3$, then $\operatorname{rank}\begin{bmatrix} u+v & v+w & w+u \end{bmatrix} = 3$.

d) If $\operatorname{rank}(A) < \operatorname{rank}\begin{bmatrix} A & B \end{bmatrix}$, then $\operatorname{rank}(A) < \operatorname{rank}\begin{bmatrix} A \\ B \end{bmatrix}$.

e) For any $A, B \in \mathbb{R}^{m\times n}$, $\operatorname{rank}\begin{bmatrix} A & B \end{bmatrix} = \operatorname{rank}\begin{bmatrix} A & A+B \end{bmatrix}$.

f) If $A \in \mathbb{R}^{m\times n}$ and $B \in \mathbb{R}^{m\times k}$, then $\operatorname{range}(A) \perp \operatorname{range}(B)$ if and only if $A^{\mathsf{T}}B = 0$.

g) If $A \in \mathbb{R}^{n\times n}$ and $\operatorname{rank}(A) = r$, then $\dim \operatorname{null}\begin{bmatrix} A & A^2 \end{bmatrix} = 2n - r$.

h) Suppose $A = \begin{bmatrix} Q_1 & Q_2 \end{bmatrix}\begin{bmatrix} R_1 \\ 0 \end{bmatrix}$ with $\begin{bmatrix} Q_1 & Q_2 \end{bmatrix}$ orthogonal, with $Q_1 \in \mathbb{R}^{m\times r}$ and $R_1 \in \mathbb{R}^{r\times n}$. If $\operatorname{null}(R_1^{\mathsf{T}}) = \{0\}$ then $\operatorname{range}(A) = \operatorname{range}(Q_1)$.

i) Suppose $A = \begin{bmatrix} Q_1 & Q_2 \end{bmatrix}\begin{bmatrix} R_1 \\ 0 \end{bmatrix}$ with $\begin{bmatrix} Q_1 & Q_2 \end{bmatrix}$ orthogonal, with $Q_1 \in \mathbb{R}^{m\times r}$ and $R_1 \in \mathbb{R}^{r\times n}$. If $\operatorname{null}(R_1^{\mathsf{T}}) = \{0\}$ and $x \notin \operatorname{range}(A)$, then $Q_1^{\mathsf{T}}x = 0$.

j) Suppose $A = \begin{bmatrix} Q_1 & Q_2 \end{bmatrix}\begin{bmatrix} I & N \\ 0 & 0 \end{bmatrix}$ with $\begin{bmatrix} Q_1 & Q_2 \end{bmatrix}$ orthogonal, with $Q_1 \in \mathbb{R}^{m\times r}$ and $N \in \mathbb{R}^{r\times(n-r)}$. Then $\operatorname{null}(A) = \operatorname{range}\begin{bmatrix} -N \\ I \end{bmatrix}$.

Solution.

a) If $f : \mathbb{R}^n \to \mathbb{R}^n$ is a linear function and $A \in \mathbb{R}^{n\times n}$, then $f(Ax) = Af(x)$.

This is **false**. For example if $f(x) = Bx$ we may have $AB \neq BA$.

b) There exists a matrix $A \in \mathbb{R}^{m\times n}$ and a vector $x \in \mathbb{R}^n$ such that $A^{\mathsf{T}}Ax = 0$ but $Ax \neq 0$.

This is **false**. It is always true that $\operatorname{null}(A^{\mathsf{T}}A) = \operatorname{null}(A)$.

c) If $u, v, w \in \mathbb{R}^n$, and $\operatorname{rank}\begin{bmatrix} u & v & w \end{bmatrix} = 3$, then $\operatorname{rank}\begin{bmatrix} u+v & v+w & w+u \end{bmatrix} = 3$.

This is **true**, because

$$\begin{bmatrix} u+v & v+w & w+u \end{bmatrix} = \begin{bmatrix} u & v & w \end{bmatrix}\begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix}$$

and $\operatorname{rank}(AB) = \operatorname{rank}(A)$ whenever B is invertible.

d) If $\operatorname{rank}(A) < \operatorname{rank}\begin{bmatrix} A & B \end{bmatrix}$, then $\operatorname{rank}(A) < \operatorname{rank}\begin{bmatrix} A \\ B \end{bmatrix}$.

This is **false.** For example, let A and B be linearly independent vectors. Then,

$$\operatorname{rank}(A) = \operatorname{rank}\begin{bmatrix} A \\ B \end{bmatrix} = 1$$

but

$$\operatorname{rank}\begin{bmatrix} A & B \end{bmatrix} = 2$$

e) For any $A, B \in \mathbb{R}^{m\times n}$, $\operatorname{rank}\begin{bmatrix} A & B \end{bmatrix} = \operatorname{rank}\begin{bmatrix} A & A+B \end{bmatrix}$.

This is **true**, because

$$\begin{bmatrix} A & A+B \end{bmatrix} = \begin{bmatrix} A & B \end{bmatrix} \begin{bmatrix} I & I \\ 0 & I \end{bmatrix}$$

and $\operatorname{rank}(AB) = \operatorname{rank}(A)$ whenever B is invertible.

f) If $A \in \mathbb{R}^{m\times n}$ and $B \in \mathbb{R}^{m\times k}$, then $\operatorname{range}(A) \perp \operatorname{range}(B)$ if and only if $A^{\mathsf{T}}B = 0$.

This is **true.** To show *if*, suppose $y \in \operatorname{range}(A)$ and $z \in \operatorname{range}(B)$. Then $y = Az$ and $z = Bw$ for some z and w. Then $y^{\mathsf{T}}z = z^{\mathsf{T}}A^{\mathsf{T}}Bw = 0$. Conversely, if $\operatorname{range}(A) \perp \operatorname{range}(B)$ then $z^{\mathsf{T}}A^{\mathsf{T}}Bw = 0$ for all z, w and hence $A^{\mathsf{T}}B = 0$.

g) If $A \in \mathbb{R}^{n\times n}$ and $\operatorname{rank}(A) = r$, then $\dim \operatorname{null}\begin{bmatrix} A & A^2 \end{bmatrix} = 2n - r$.

This is **true.** We have $\operatorname{range}(A^2) \subset \operatorname{range}(A)$, and so $\operatorname{rank}\begin{bmatrix} A & A^2 \end{bmatrix} = \operatorname{rank}(A) = r$. Then the result follows from conservation of dimension.

h) Suppose $A = \begin{bmatrix} Q_1 & Q_2 \end{bmatrix}\begin{bmatrix} R_1 \\ 0 \end{bmatrix}$ with $\begin{bmatrix} Q_1 & Q_2 \end{bmatrix}$ orthogonal, with $Q_1 \in \mathbb{R}^{m\times r}$ and $R_1 \in \mathbb{R}^{r\times n}$. If $\operatorname{null}(R_1^{\mathsf{T}}) = \{0\}$ then $\operatorname{range}(A) = \operatorname{range}(Q_1)$.

This is **true.** Clearly $\operatorname{range}(A) \subset \operatorname{range}(Q_1)$, since $A = Q_1R$. But if $y \in \operatorname{range}(Q_1)$ then $y = Q_1w$ for some w. Since $\operatorname{null} R_1^{\mathsf{T}} = \{0\}$ we have $\operatorname{range}(R_1) = \mathbb{R}^r$ and so for any w there exists z such that $w = R_1z$. Hence $y = Q_1R_1z = Az$.

i) Suppose $A = \begin{bmatrix} Q_1 & Q_2 \end{bmatrix}\begin{bmatrix} R_1 \\ 0 \end{bmatrix}$ with $\begin{bmatrix} Q_1 & Q_2 \end{bmatrix}$ orthogonal, with $Q_1 \in \mathbb{R}^{m\times r}$ and $R_1 \in \mathbb{R}^{r\times n}$. If $\operatorname{null}(R_1^{\mathsf{T}}) = \{0\}$ and $x \notin \operatorname{range}(A)$, then $Q_1^{\mathsf{T}}x = 0$.

This is **False.** Consider

$$Q_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \quad Q_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \quad R_1 = 1$$

Then $A = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}$, so $x = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$ is not in $\operatorname{range}(A)$, but does not have $Q_1^{\mathsf{T}}x = 0$.

j) Suppose $A = \begin{bmatrix} Q_1 & Q_2 \end{bmatrix}\begin{bmatrix} I & N \\ 0 & 0 \end{bmatrix}$ with $\begin{bmatrix} Q_1 & Q_2 \end{bmatrix}$ orthogonal, with $Q_1 \in \mathbb{R}^{m\times r}$ and $N \in \mathbb{R}^{r\times(n-r)}$. Then $\operatorname{null}(A) = \operatorname{range}\begin{bmatrix} -N \\ I \end{bmatrix}$.

This is **true.**. We have $Ax = 0$ if and only if $\begin{bmatrix} I & N \end{bmatrix}x = 0$ which holds if and only if $x \in \operatorname{range}\begin{bmatrix} -N \\ I \end{bmatrix}$.

5. **Some properties of the rank.** In this problem, we will prove a useful inequality about the rank of matrices, intuitively indicating that the rank of the multiplication of two matrices cannot be *too small* if the original matrices have high ranks. Each part of the problem consists of a small step of the proof.

   a) Let I_k be the $k \times k$ identity matrix. Then, for arbitrary matrices A and B with appropriate dimensions, what is the rank of the following matrices M_1 and M_2? Justify your answer.

$$M_1 = \begin{pmatrix} I_k & 0 \\ A & I_\ell \end{pmatrix}, \quad M_2 = \begin{pmatrix} I_k & B \\ 0 & I_\ell \end{pmatrix}$$

   b) Let $U \in \mathbb{R}^{(k+\ell)\times m}$ be an arbitrary matrix of rank r. Express the rank of $M_1 U$ and $U^T M_2$ in terms of k, ℓ, m, and r, and justify your answer.

   c) Let $C \in \mathbb{R}^{p\times n}$ and $D \in \mathbb{R}^{n\times q}$ be arbitrary matrices. Find two block matrices M_1 and M_2 with similar block structure to the ones introduced in part (a) such that

$$M_1 \begin{pmatrix} I_n & D \\ C & 0 \end{pmatrix} M_2 = \begin{pmatrix} I_n & 0 \\ 0 & -CD \end{pmatrix}.$$

   d) Using the results from the previous parts, show that

$$\operatorname{rank} \begin{pmatrix} I_n & D \\ C & 0 \end{pmatrix} - \operatorname{rank}(CD)$$

   is a constant. Find this constant in terms of n, p, and q, and justify your answer.

   e) Show that

$$\operatorname{rank}(C) + \operatorname{rank}(D) \leq \operatorname{rank} \begin{pmatrix} I_n & D \\ C & 0 \end{pmatrix}.$$

   Note that you can solve this part independently, without needing any results from the previous parts. Also, even if you cannot prove the result of this part, you can use if for the next part.

   Hint: one way to approach this problem is to first show that, if C and D are full-column rank, then for any matrix J,

$$\operatorname{rank} \begin{bmatrix} J & D \\ C & 0 \end{bmatrix} = \operatorname{rank}(D) + \operatorname{rank}(C)$$

   f) Use the results from parts (d) and (e) to show that

$$\operatorname{rank}(C) + \operatorname{rank}(D) - \operatorname{rank}(CD) \leq s,$$

   where s is the constant you found in part (d).

Solution.

a) Both M_1 and M_2 are full rank since

$$M_1 x = M_1 \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = 0 \Rightarrow \begin{bmatrix} x_1 \\ Ax_1 + x_2 \end{bmatrix} = 0 \Rightarrow \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = 0 \Rightarrow x = 0 \Rightarrow \operatorname{rank}(M_1) = k+\ell.$$

A similar argument holds for M_2.

b) Since M_1 is full rank, $\operatorname{null}(U) = \operatorname{null}(M_1 U)$, so $\operatorname{rank}(M_1 U) = \operatorname{rank}(U) = r$. Similarly, we have that $\operatorname{rank}(U^T M_2) = \operatorname{rank}(M_2^T U) = r$.

c)

$$M_1 = \begin{pmatrix} I_n & 0 \\ -C & I_p \end{pmatrix}, \quad M_2 = \begin{pmatrix} I_n & -D \\ 0 & I_q \end{pmatrix}$$

d) From part b, we know that

$$\operatorname{rank}\left(M_1 \begin{pmatrix} I_n & D \\ C & 0 \end{pmatrix} M_2 \right) = \operatorname{rank}\begin{pmatrix} I_n & D \\ C & 0 \end{pmatrix}.$$

From part c, we know

$$\operatorname{rank}\left(M_1 \begin{pmatrix} I_n & D \\ C & 0 \end{pmatrix} M_2 \right) = \operatorname{rank}\begin{pmatrix} I_n & 0 \\ 0 & -CD \end{pmatrix},$$

and the RHS of this expression is $\operatorname{rank}(CD) + n$, so

$$\operatorname{rank}\begin{pmatrix} I_n & D \\ C & 0 \end{pmatrix} - \operatorname{rank}(CD) = n.$$

e) Let's remove the minimum required number of columns from C and D such that the remaining columns are independent. The resulting matrices are denoted by $\tilde{C}$ and $\tilde{D}$, having $\operatorname{rank}(C)$ and $\operatorname{rank}(D)$ columns respectively. If we do this on $\begin{pmatrix} I_n & D \\ C & 0 \end{pmatrix}$, then the resulting matrix is $\begin{pmatrix} \tilde{I}_n & \tilde{D} \\ \tilde{C} & 0 \end{pmatrix}$, such that $\tilde{I}_n$ is the identity matrix with some columns removed. Now, if

$$\begin{pmatrix} \tilde{I}_n & \tilde{D} \\ \tilde{C} & 0 \end{pmatrix} x = \begin{pmatrix} \tilde{I}_n & \tilde{D} \\ \tilde{C} & 0 \end{pmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} \tilde{x}_1 + \tilde{D} x_2 \\ \tilde{C} x_1 \end{bmatrix} = 0,$$

then $x_1 = 0$ because $\tilde{C}$ is full rank. Then $\tilde{x}_1$, which is x_1 appended with some zeros, is also zero, so $\tilde{D} x_2 = 0$, which implies that $x_2 = 0$ since $\tilde{D}$ is full rank. Thus, $x = 0$ and $\begin{pmatrix} \tilde{I}_n & \tilde{D} \\ \tilde{C} & 0 \end{pmatrix}$ is full rank, which means that its rank is $\operatorname{rank}(C) +$

rank(D). Note that this matrix is constructed by removing some columns from $\begin{pmatrix} I_n & D \\ C & 0 \end{pmatrix}$, and this process can only decrease the rank. Thus,

$$\operatorname{rank}(C) + \operatorname{rank}(D) = \operatorname{rank} \begin{pmatrix} \tilde{I}_n & \tilde{D} \\ \tilde{C} & 0 \end{pmatrix} \leq \operatorname{rank} \begin{pmatrix} I_n & D \\ C & 0 \end{pmatrix}.$$

f)

$$n = \operatorname{rank} \begin{pmatrix} I_n & D \\ C & 0 \end{pmatrix} - \operatorname{rank}(CD) \geq \operatorname{rank}(C) + \operatorname{rank}(D) - \operatorname{rank}(CD) \Rightarrow s = n$$