4.830. **True/false questions about linear algebra.** Determine whether each of the following statements is true or false. In each case, give either a proof or a counterexample.

a) If $Q$ has orthonormal columns, then $\|Q^Tw\| \leq \|w\|$ for all vectors $w$.

b) Suppose $A \in \mathbb{R}^{m \times p}$ and $B \in \mathbb{R}^{m \times q}$. If $\text{null}(A) = \{0\}$ and $\text{range}(A) \subset \text{range}(B)$, then $p \leq q$.

c) If $V = [V_1 \ V_2]$ is invertible and $\text{range}(V_1) = \text{null}(A)$, then $\text{null}(AV_2) = \{0\}$.

d) If $\text{rank}([A \ B]) = \text{rank}(A) = \text{rank}(B)$, then $\text{range}(A) = \text{range}(B)$.

e) Suppose $A \in \mathbb{R}^{m \times n}$. Then, $x \in \text{null}(A^T)$ if and only if $x \notin \text{range}(A)$.

f) Suppose $A$ is invertible. Then, $AB$ is not full rank if and only if $B$ is not full rank.

g) If $A$ is not full rank, then there is a nonzero vector $x$ such that $Ax = 0$.

**Solution.**

a) The statement is true. Suppose $Q \in \mathbb{R}^{m \times n}$. Because the columns of $Q$ are orthonormal, and hence linearly independent, we know that $m \geq n$. If $m = n$, then $Q$ is an orthogonal matrix, so $QQ^T$, and we have that

$$\|Q^Tw\|^2 = w^T(QQ^T)w = w^Tw = \|w\|^2.$$  

Now consider the case when $m > n$. Let $q_1, \ldots, q_n$ be the columns of $Q$:

$$Q = [q_1 \cdots q_n].$$

Then, we can extend $(q_1, \ldots, q_n)$ to an orthonormal basis $(q_1, \ldots, q_m)$ for $\mathbb{R}^m$. Define the matrix $\hat{Q} \in \mathbb{R}^{m \times (m-n)}$ such that

$$\hat{Q} = [q_{m+1} \cdots q_n].$$

Then, we have that $[Q \ \hat{Q}]$ is an orthogonal matrix, so that

$$\|w\|^2 = \left\|\begin{bmatrix} Q & \hat{Q} \end{bmatrix}w\right\|^2 = \left\|\begin{bmatrix} Q^T \hat{Q}^T \end{bmatrix}w\right\|^2 = \|Q^Tw\|^2 + \|\hat{Q}^Tw\|^2 \geq \|Q^Tw\|^2.$$  

Combining these results, we see that if the columns of $Q$ are orthonormal, then $\|Q^Tw\| \leq \|w\|$ for all vectors $w$. (This result is known as Bessel’s inequality.)

b) The statement is true. Because $\text{range}(A) \subset \text{range}(B)$, we have that

$$\text{rank}(A) = \dim(\text{range}(A)) \leq \dim(\text{range}(B)) = \text{rank}(B).$$
Since the rank of a matrix is bounded by its number of columns, we have that
\[ \text{rank}(B) \leq q. \]

Conservation of dimension tells us that
\[ \text{rank}(A) = \dim(\mathbb{R}^p) - \dim(\text{null}(A)) = \dim(\mathbb{R}^p) - \dim(\{0\}) = p - 0 = p. \]

Combining these results, we have that
\[ p = \text{rank}(A) \leq \text{rank}(B) \leq q. \]

c) The statement is true. Suppose \( y \in \text{null}(AV_2) \). Then, \( V_2y \in \text{null}(A) = \text{range}(V_1) \), so there exists a vector \( x \) such that \( V_1x = V_2y \). Therefore, we have that
\[ V_1x - V_2y = [V_1 V_2] \begin{bmatrix} x \\ -y \end{bmatrix} = 0. \]

Because \([V_1 V_2]\) is invertible, this implies that
\[ \begin{bmatrix} x \\ -y \end{bmatrix} = 0, \]
and hence that \( y = 0 \). Thus, we see that \( \text{null}(AV_2) = \{0\} \).

d) The statement is true. It is sufficient to show that if \( \text{rank}(\begin{bmatrix} A & B \end{bmatrix}) = \text{rank}(A) \), then \( \text{range}(A) = \text{range}(\begin{bmatrix} A & B \end{bmatrix}) \). This implies that if \( \text{rank}(\begin{bmatrix} A & B \end{bmatrix}) = \text{rank}(A) = \text{rank}(B) \), then
\[ \text{range}(A) = \text{range}(\begin{bmatrix} A & B \end{bmatrix}) = \text{range}(B). \]

Consider any \( y \in \text{range}(A) \). There exists a vector \( x \) such that \( Ax = y \). Then, we have that
\[ \begin{bmatrix} A & B \end{bmatrix} \begin{bmatrix} x \\ 0 \end{bmatrix} = Ax = y, \]
so that \( y \in \text{range}(\begin{bmatrix} A & B \end{bmatrix}) \). Thus, we have that \( \text{range}(A) \subseteq \text{range}(\begin{bmatrix} A & B \end{bmatrix}) \). (Note that this result holds for any matrices \( A \) and \( B \).) Now suppose that \( \text{rank}(A) = \text{rank}(\begin{bmatrix} A & B \end{bmatrix}) \).

Let \((q_1, \ldots, q_m)\) be a basis for \( \text{range}(A) \). Since \( \text{range}(A) \) is a subspace of \( \text{range}(\begin{bmatrix} A & B \end{bmatrix}) \), we can extend this basis to a basis \((q_1, \ldots, q_n)\) for \( \text{range}(\begin{bmatrix} A & B \end{bmatrix}) \). However, it must be the case that
\[ n = \text{rank}(\begin{bmatrix} A & B \end{bmatrix}) = \text{rank}(A) = m. \]

Thus, \((q_1, \ldots, q_m)\) is a basis for both \( \text{range}(A) \) and \( \text{range}(\begin{bmatrix} A & B \end{bmatrix}) \). Therefore, for any \( y \in \text{range}(\begin{bmatrix} A & B \end{bmatrix}) \), there exist scalars \( c_1, \ldots, c_m \) such that
\[ y = c_1q_1 + \cdots + c_mq_m. \]

Because \((q_1, \ldots, q_m)\) is also a basis for \( \text{range}(A) \), this implies that \( y \in \text{range}(A) \). This shows that if \( \text{rank}(A) = \text{rank}(\begin{bmatrix} A & B \end{bmatrix}) \), then \( \text{range}(\begin{bmatrix} A & B \end{bmatrix}) \subseteq \text{range}(A) \), and thereby completes the proof.
e) The statement is false. In fact, neither direction of the equivalence is true. Consider the matrix

\[ A = \begin{bmatrix} 1 \\ 0 \end{bmatrix}. \]

For \( x = 0 \), we have that \( x \in \text{null}(A^T) \), and \( x \in \text{range}(A) \); for \( x = (1, 1) \), we have that \( x \notin \text{range}(A) \), and \( x \notin \text{null}(A^T) \).

f) The statement is true. We claim that if \( A \) is invertible, then \( \text{rank}(AB) = \text{rank}(B) \). This implies that \( AB \) is full rank if and only if \( B \) is full rank. Let \( (q_1, \ldots, q_k) \) be a basis for \( \text{range}(B) \). We claim that \( (Aq_1, \ldots, Aq_k) \) is a basis for \( AB \). Consider any \( y \in \text{range}(AB) \). There exists a vector \( x \) such that \( (AB)x = y \). Note that \( Bx \) is a vector in \( \text{range}(B) \), so there exist scalars \( c_1, \ldots, c_k \) such that

\[ Bx = c_1 q_1 + \cdots + c_k q_k. \]

Then, we have that

\[ y = (AB)x = A(Bx) = A(c_1 q_1 + \cdots + c_k q_k) = c_1 (Aq_1) + \cdots + c_k (Aq_k). \]

This shows that \( (Aq_1, \ldots, Aq_k) \) spans \( \text{range}(AB) \). (This is true for any matrices \( A \) and \( B \).) Suppose there exist scalars \( d_1, \ldots, d_k \) such that

\[ d_1 (Aq_1) + \cdots + d_k (Aq_k) = A(d_1 q_1 + \cdots + d_k q_k) = 0. \]

Since \( A \) is invertible, this implies that

\[ d_1 q_1 + \cdots + d_k q_k = A^{-1} 0 = 0. \]

Then, because \( (q_1, \ldots, q_k) \) is a basis, and hence linearly independent, we have that

\[ d_1 = \cdots = d_k = 0. \]

This shows that \( (Aq_1, \ldots, Aq_k) \) is linearly independent, and completes the proof that \( (Aq_1, \ldots, Aq_k) \) is a basis for \( \text{range}(AB) \). Since the dimension of a subspace is the number of vectors in a basis for the subspace, we have that

\[ \text{rank}(B) = \dim(\text{range}(B)) = k = \dim(\text{range}(AB)) = \text{rank}(AB). \]

g) The statement is true. If \( A \) is strictly fat, then there exists a nonzero vector \( x \) such that \( Ax = 0 \) irrespective of whether of \( A \) is full rank. Suppose \( A \) is skinny (or square). If \( A \) is not full rank, then the columns of \( A \) must be linearly dependent: that is, there exist scalars \( x_1, \ldots, x_n \), at least one of which is nonzero, such that

\[ x_1 A_{x_1} + \cdots + x_n A_{x_n} = 0. \]

In matrix form, this equation says that

\[ \begin{bmatrix} A_{x_1} & \cdots & A_{x_n} \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} = Ax = 0, \]

where \( x \neq 0 \). □
6.1240. Iteratively reweighted least squares for 1-norm approximation. In an ordinary least squares problem, we are given $A \in \mathbb{R}^{m \times n}$ (skinny and full rank) and $y \in \mathbb{R}^m$, and we choose $x \in \mathbb{R}^n$ in order to minimize

$$\|Ax - y\|_2^2 = \sum_{i=1}^{m} (\tilde{a}_i^T x - y_i)^2.$$ 

Note that the penalty that we assign to a measurement error does not depend on the sensor from which the measurement was taken. However, this is not always the right thing to do: if we believe that one sensor is more accurate than another, we might want to assign a larger penalty to an error in the measurement from the more accurate sensor. We can account for differences in the accuracies of our sensors by assigning sensor $i$ a weight $w_i > 0$, and then minimizing

$$\sum_{i=1}^{m} w_i (\tilde{a}_i^T x - y_i)^2.$$ 

By giving larger weights to more accurate sensors, we can account for differences in the precision of our sensors.

a) Weighted least squares. Explain how to choose $x$ in order to minimize

$$\sum_{i=1}^{m} w_i (\tilde{a}_i^T x - y_i)^2,$$

where the weights $w_1, \ldots, w_m > 0$ are given.

b) Iteratively reweighted least squares for $\ell_1$-norm approximation. Consider a cost function of the form

$$\sum_{i=1}^{m} w_i (x)(\tilde{a}_i^T x - y_i)^2. \quad (1)$$

One heuristic for minimizing a cost function of the form given in (1) is iteratively reweighted least squares, which works as follows. First, we choose an initial point $x^{(0)} \in \mathbb{R}^n$. Then, we generate a sequence of points $x^{(1)}, x^{(2)}, \ldots \in \mathbb{R}^n$ by choosing $x^{(k+1)}$ in order to minimize

$$\sum_{i=1}^{m} w_i (x^{(k)})(\tilde{a}_i^T x^{(k+1)} - y_i)^2.$$ 

Each step of this algorithm involves updating our weights, and solving a weighted least squares problem. Suppose we want to use this method to solve minimize the $\ell_1$-norm approximation error, which is defined to be

$$\|Ax - y\|_1 = \sum_{i=1}^{m} |\tilde{a}_i^T x - y_i|,$$

where the matrix $A \in \mathbb{R}^{m \times n}$ and the vector $y \in \mathbb{R}^m$ are given. How should we choose the weights $w_i(x)$ to make the cost function in (1) equal to the $\ell_1$-norm approximation error?
c) Numerical example. The file \texttt{l1_irwls_data.json} contains data \((t_1, y_1), \ldots, (t_m, y_m)\).

We want to fit an affine model to this data:
\[
y_i = x_1 + x_2 t_i, \quad i = 1, \ldots, m.
\]

Choose \(x^{(0)}\) to be the vector of least-squares parameter estimates: that is, choose \(x^{(0)}\) in order to minimize
\[
\sum_{i=1}^{m} ((x_1^{(0)} + x_2^{(0)} t_i) - y_i)^2.
\]

Generate \(x^{(1)}, x^{(2)}, \ldots\) using iteratively reweighted least squares for \(\ell_1\)-norm approximation. You can stop generating iterates when \(\|x^{(k+1)} - x^{(k)}\| < 10^{-6}\). Report your values of \(x^{(0)}\) and the final \(x^{(k)}\) in your sequence of points. Draw a scatterplot of the data points \((t_i, y_i)\). Add the fitted lines corresponding to \(x^{(0)}\) and the final \(x^{(k)}\) to your scatterplot. What do you observe?

Remark. Suppose we fit the least-squares line to some data. Then, a point that is very far from the least-squares line may be an outlier: that is, a point that does not seem to follow the same model as the rest of the data. Because such points may not follow the same model as the rest of data, it may make sense to give such points less weight. This idea is the intuition behind iteratively reweighted least squares for \(\ell_1\)-norm approximation.

Solution.

a) We can express the objective function as
\[
\sum_{i=1}^{m} w_i (\tilde{a}_i^T x - y_i)^2 = \sum_{i=1}^{m} (\sqrt{w_i} \tilde{a}_i^T x - y_i)^2
\]
\[
= \left\| \begin{bmatrix} \sqrt{w_1} \tilde{a}_1^T \\ \vdots \\ \sqrt{w_m} \tilde{a}_m^T \end{bmatrix} x - \begin{bmatrix} \sqrt{w_1} y_1 \\ \vdots \\ \sqrt{w_m} y_m \end{bmatrix} \right\|^2
\]
\[
= \| W^{\frac{1}{2}} A x - W^{\frac{1}{2}} y \|^2,
\]
where \(W^{\frac{1}{2}} = \text{diag}(\sqrt{w_1}, \ldots, \sqrt{w_m})\). Note that \((W^{\frac{1}{2}})^2 = W = \text{diag}(w_1, \ldots, w_m)\). Thus, minimizing this objective function is a least-squares problem; the solution is
\[
x = ((W^{\frac{1}{2}} A)^T (W^{\frac{1}{2}} A))^{-1} (W^{\frac{1}{2}} A)^T (W^{\frac{1}{2}} y) = (A^T W A)^{-1} A^T W y.
\]

b) If we choose \(w_i(x) = 1/|\tilde{a}_i^T x - y_i|\), then we have that
\[
\sum_{i=1}^{m} w_i(x) (\tilde{a}_i^T x - y_i)^2 = \sum_{i=1}^{m} \frac{1}{|\tilde{a}_i^T x - y_i|} (\tilde{a}_i^T x - y_i)^2 = \sum_{i=1}^{m} |\tilde{a}_i^T x - y_i| = \| Ax - y \|_1.
\]
Note that \(w_i(x)\) is undefined if \(w_i(x) = |\tilde{a}_i^T x - y_i| = 0\). In this case, we can just take \(w_i(x)\) to be some large value \(w_{\text{max}}\). However, note that in practice it is extremely unlikely that one of the residuals will be exactly equal to zero.
c) For fitting a line in two-dimensions, we have that $\tilde{a}_i = (1, t_i)$. The following code applies iteratively reweighted least squares for $\ell_1$-norm approximation to the data defined in 11_irwls_data.json.

```python
A = [ones(m) zeros(m)]
A[:,2] = t

x = A \ y
x_prev = x
x0 = copy(x)

while true
    w = 1.0 ./ abs.(A*x - y)
    x_prev = copy(x)
    W = diagm(vec(w))
    x = (A'*W*A) \ A'*W*y
    println(norm(x-x_prev))
    if norm(x-x_prev) < 10e-6
        print("Finished!")
        break
    end
end

scatter(t, y, label="(t,y)")
plot!(t, A*x0, label="(t, Ax0)")
plot!(t, A*x, label="(t, AxF)")
```

The initial and final parameter estimates are

$$x^{(0)} = \begin{bmatrix} 3.5084 \\ 5.5540 \end{bmatrix} \quad \text{and} \quad x^{(20)} = \begin{bmatrix} 2.0839 \\ 7.8694 \end{bmatrix}. $$

The $\ell_2$- and $\ell_1$-norm approximations of the data are given in the plot below. We see that the $\ell_2$-norm approximation is much more sensitive to the outliers than the $\ell_1$-norm approximation.
7.1040. Fitting a Gaussian function to data. A Gaussian function has the form

\[ f(t) = ae^{-(t-\mu)^2/\sigma^2}. \]

Here \( t \in \mathbb{R} \) is the independent variable, and \( a \in \mathbb{R}, \mu \in \mathbb{R}, \) and \( \sigma \in \mathbb{R} \) are parameters that affect its shape. The parameter \( a \) is called the amplitude of the Gaussian, \( \mu \) is called its center, and \( \sigma \) is called the spread or width. We can always take \( \sigma > 0 \). For convenience we define \( p \in \mathbb{R}^3 \) as the vector of the parameters, i.e., \( p = [a \ \mu \ \sigma]^T \). We are given a set of data,

\[ t_1, \ldots, t_N, \quad y_1, \ldots, y_N, \]

and our goal is to fit a Gaussian function to the data. We will measure the quality of the fit by the root-mean-square (RMS) fitting error, given by

\[ E = \left( \frac{1}{N} \sum_{i=1}^{N} (f(t_i) - y_i)^2 \right)^{1/2}. \]

Note that \( E \) is a function of the parameters \( a, \mu, \sigma, \) i.e., \( p \). Your job is to choose these parameters to minimize \( E \). You’ll use the Gauss-Newton method.

a) Work out the details of the Gauss-Newton method for this fitting problem. Explicitly describe the Gauss-Newton steps, including the matrices and vectors that come up. You can use the notation \( \Delta p^{(k)} = [\Delta a^{(k)} \ \Delta \mu^{(k)} \ \Delta \sigma^{(k)}]^T \) to denote the update to the parameters, i.e.,

\[ p^{(k+1)} = p^{(k)} + \Delta p^{(k)}. \]

(Here \( k \) denotes the \( k \)th iteration.)
b) Get the data \(t, y\) (and \(N\)) from the file `gauss_fit_data.json`, available on the class website. Implement the Gauss-Newton (as outlined in part (a) above). You’ll need an initial guess for the parameters. You can visually estimate them (giving a short justification), or estimate them by any other method (but you must explain your method). Plot the RMS error \(E\) as a function of the iteration number. (You should plot enough iterations to convince yourself that the algorithm has nearly converged.) Plot the final Gaussian function obtained along with the data on the same plot. Repeat for another reasonable, but different initial guess for the parameters. Repeat for another set of parameters that is not reasonable, i.e., not a good guess for the parameters. (It’s possible, of course, that the Gauss-Newton algorithm doesn’t converge, or fails at some step; if this occurs, say so.) Briefly comment on the results you obtain in the three cases.

Solution.

a) Minimizing \(E\) is the same as minimizing \(N E^2\), which is a nonlinear least-squares problem. The first thing to do is to find the first-order approximation of the Gaussian function, with respect to the parameters \(a, \mu,\) and \(\sigma\). This approximation is

\[
f(t) + \left( \frac{\partial f(t)}{\partial a} \Delta a + \frac{\partial f(t)}{\partial \mu} \Delta \mu + \frac{\partial f(t)}{\partial \sigma} \Delta \sigma \right),
\]

where all the partial derivatives are evaluated at the current parameter values. In matrix form, this first-order approximation is

\[
f(t) + (\nabla_p f(t))^T \Delta p,
\]

where \(\nabla_p\) denotes the gradient with respect to \(p\). These partial derivatives are:

\[
\begin{align*}
\frac{\partial}{\partial a} f(t) &= e^{-(t-\mu)^2/\sigma^2} \\
\frac{\partial}{\partial \mu} f(t) &= \frac{2a(t-\mu)}{\sigma^2} e^{-(t-\mu)^2/\sigma^2} \\
\frac{\partial}{\partial \sigma} f(t) &= \frac{2a(t-\mu)^2}{\sigma^3} e^{-(t-\mu)^2/\sigma^2}
\end{align*}
\]

The Gauss-Newton method proceeds as follows. We find \(\Delta p\) that minimizes

\[
\sum_{i=1}^{N} \left( f(t_i) + \nabla_p f(t_i)^T \Delta p - y_i \right)^2,
\]

and then set the new value of \(p\) to be \(p := p + \Delta p\). Finding \(\Delta p\) is a (linear) least-squares problem. We can put this least-squares problem in a more conventional form by defining

\[
A = \begin{bmatrix}
\nabla_p f(t_1)^T \\
\vdots \\
\nabla_p f(t_N)^T
\end{bmatrix}, \quad b = \begin{bmatrix}
y_1 - f(t_1) \\
\vdots \\
y_N - f(t_N)
\end{bmatrix}.
\]

Then, \(\Delta p\) is found by minimizing \(\|A \Delta p - b\|\). Thus, we have

\[
\Delta p = (A^T A)^{-1} A^T b.
\]
To summarize, the algorithm repeats the following steps:

- Evaluate the vector \( b \) (which is the vector of fitting residuals.) Evaluate the partial derivatives to form the matrix \( A \).
- Solve the least-squares problem to get \( \Delta p \).
- Update the parameter vector: \( p := p + \Delta p \).

This can be repeated until the update \( \Delta p \) is small, or the improvement in \( E \) is small.

b) We used the starting parameter values \( p = [11, 50, 35]^T \), estimated visually. The amplitude \( a = 11 \) was estimated as a guess for the (noise-free) peak of the graph, \( \mu = 50 \) was estimated as its center, and \( \sigma = 35 \) was estimated from its spread.

The results are shown below. The final fit clearly is good (at least, visually), at \( a \approx 12.10, \mu \approx 54.81, \sigma \approx 42.02 \). The final RMS fit level is around \( E \approx 1.83 \), and convergence happens very quickly, in just a handful of iterations.

Now we try with another starting point, \( p = (10, 20, 10) \). The final fit is the same (well, \( \sigma \) landed on \(-42.02 \), but that doesn’t matter). It does tend to bounce around a bit more before converging, which is indicative of its nonlinearity. That it landed in the same place bolsters our confidence that the fit found in our first run (the same as this one) is probably the best fit possible.

For other poor initial guesses, however, the algorithm fails to converge. For example, with initial parameter estimate \( p = (5, 20, 10) \), there’s a miserable spike before coming
back to $E \approx 105$, a clearly not optimal fit.

The Julia code for the Gauss-Newton method is given below.

Note: Julia supports Unicode characters, so if you type something like \sigma then Tab, Jupyter and Julia’s REPL will convert it to the Greek letter, in place of the Latin-spelled sigma. But B\LaTeX doesn’t support Unicode characters, so we Latinized the Greek letters to print the code here.

```julia
using LinearAlgebra
using Plots
include("readclassjson.jl")
data = readclassjson("gauss_fit_data.json")

N = data["N"]
t = data["t"]
y = data["y"]

function fit_gaussian(p_init)
    p = p_init
    E = Float64[]
    for i = 1:20
        a, mu, sigma = p
        w = exp.(-w .- mu).^2 / sigma^2)
        A =[w 2*a*(t.-mu)/sigma^2 .* w 2*a*(t.-mu).^2/sigma^3 .* w]
f = a .* w
    b = y - f
    Deltap = A \ b
    p += Deltap
    push!(E, sqrt(sum((f - y).^2) / N))
    end
    p..., E
end

fit_gaussian(a_init, mu_init, sigma_init) = fit_gaussian([a_init, mu_init, sigma_init])
```

10
a, mu, sigma, E = fit_gaussian(20, 50, 15)
f = a * exp.(-(t .- mu).^2 / sigma^2)
scatter(t, y, label="data", marker=:+)
plot!(t, f, label="fitted")
plot(E, label="rms error")

8.160. Designing an equalizer for backwards-compatible wireless transceivers. You want to design the equalizer for a new line of wireless handheld transceivers (more commonly called walkie-talkies). The transmitter for the new line of transceivers has already been designed (and cannot be changed) – if the input signal is \( x \in \mathbb{R}^n \), then the transmitted signal is \( y = A_{\text{new}} x \in \mathbb{R}^m \), where \( A_{\text{new}} \in \mathbb{R}^{m \times n} \) is known. An equalizer for \( A_{\text{new}} \) is a matrix \( B \in \mathbb{R}^{n \times m} \) such that \( By = x \) for every \( x \in \mathbb{R}^n \).

The new line of transceivers will replace an older model. Given an input signal \( x \in \mathbb{R}^n \), the old line of transceivers transmit a signal \( y_{\text{old}} = A_{\text{old}} x \in \mathbb{R}^m \), where \( A_{\text{old}} \in \mathbb{R}^{m \times n} \) is known. In addition to providing exact equalization for the new line of transceivers, you want your equalizer to be able to at least partially equalize signals transmitted using the old line of transceivers. In other words, to the extent that it is possible, you want the new line of transceivers to be backwards compatible with the old line of transceivers.

a) Explain how to find an equalizer \( B \) that minimizes

\[
J = \|BA_{\text{old}} - I\|_F^2 = \sum_{i=1}^{n} \sum_{j=1}^{n} (BA_{\text{old}} - I)_{ij}^2
\]

among all \( B \) that exactly equalize \( A_{\text{new}} \). Such a \( B \) is an exact equalizer for \( A_{\text{new}} \), and an approximate equalizer for \( A_{\text{old}} \). State any assumptions that are needed for your method to work.

b) The file `backwards_compatible_transceiver_data.json` defines the following variables.

- \( A_{\text{new}} \), the \( m \times n \) matrix that describes the transmitter used in the new line of transceivers
- \( A_{\text{old}} \), the \( m \times n \) matrix that describes the transmitter used in the old line of transceivers
- \( x \), a vector of length \( n \) that serves as an example input signal

Apply your method to this example data. Report the optimal value of \( J \). The pseudoinverse \( A_{\text{new}}^\dagger \) is another exact equalizer for \( A_{\text{new}} \). Compare the optimal value of \( J \), and the value of \( J \) achieved by \( A_{\text{new}}^\dagger \).

c) The example signal \( x \) defined in the data file is a binary signal. Form the signal \( y_{\text{old}} = A_{\text{old}} x \) transmitted by the old line of transceivers, and construct an estimate of \( x \) by equalizing \( y_{\text{old}} \) using \( B \), and then rounding the result to a binary signal. More concretely, compute the estimate \( \hat{x} \in \mathbb{R}^n \), where

\[
\hat{x}_i = \begin{cases} 
1 & (By_{\text{old}})_i > \frac{1}{2}, \\
0 & \text{otherwise}.
\end{cases}
\]
Report the bit error rate of your estimate, which is defined as
\[ \frac{1}{n} \sum_{i=1}^{n} I(x_i \neq \hat{x}_i), \]
where \( I(x_i \neq \hat{x}_i) \) is an indicator function:
\[
I(x_i \neq \hat{x}_i) = \begin{cases} 
1 & x_i \neq \hat{x}_i, \\
0 & \text{otherwise}.
\end{cases}
\]
Similarly, report the bit error rate if \( A_{\text{new}}^\dagger \) is used as the equalizer.

**Solution.**

a) Write the equalizer \( B \in \mathbb{R}^{n \times m} \) in terms of its rows:
\[
B = \begin{bmatrix} b_1^T \\ \vdots \\ b_n^T \end{bmatrix}.
\]
We require that \( B \) be an exact equalizer for \( A_{\text{new}} \): that is, \( BA_{\text{new}} = I \). We can express this condition in terms of the rows of \( B \) as
\[
A_{\text{new}}^T b_i = e_i, \quad i = 1, \ldots, n,
\]
where \( e_i \) denotes the \( i \)th standard basis vector in \( \mathbb{R}^n \). Similarly, we can write our objective in terms of the rows of \( B \):
\[
J = \sum_{i=1}^{n} \sum_{j=1}^{n} (BA_{\text{old}} - I)_{ij}^2 = \sum_{i=1}^{n} \| (BA_{\text{old}} - I)_{i*} \|^2 = \sum_{i=1}^{n} \| A_{\text{old}}^T b_i - e_i \|^2.
\]
Thus, we want to solve the following optimization problem.

\[
\begin{align*}
\text{minimize} & \quad \sum_{i=1}^{n} \| A_{\text{old}}^T b_i - e_i \|^2 \\
\text{subject to} & \quad A_{\text{new}}^T b_i = e_i, \quad i = 1, \ldots, n
\end{align*}
\]
This problem is separable in the rows of \( B \), allowing us to decompose it into \( n \) vector optimization problems:
\[
\begin{align*}
\text{minimize} & \quad \| A_{\text{old}}^T b_i - e_i \|^2 \\
\text{subject to} & \quad A_{\text{new}}^T b_i = e_i
\end{align*}
\]
for \( i = 1, \ldots, n \). Each of these problems is a linearly constrained minimum-norm problem; the solution of such a problem can be obtained by solving the following system of equations:
\[
\begin{bmatrix} A_{\text{old}} A_{\text{old}}^T & A_{\text{new}} \\ A_{\text{new}}^T & 0 \end{bmatrix} \begin{bmatrix} b_i \\ \lambda_i \end{bmatrix} = \begin{bmatrix} A_{\text{old}} e_i \\ e_i \end{bmatrix}, \quad i = 1, \ldots, n.
\]
This method works as long as each of these optimization problems is feasible – that is, as long as we can find a matrix \( B \in \mathbb{R}^{n \times m} \) such that

\[
BA_{\text{new}} = I.
\]

In other words, we require that \( A_{\text{new}} \) be skinny and full rank (or, equivalently, left invertible). In order for the KKT system to have a unique solution, we require that \( A_{\text{new}}^\top \) be fat and full rank, and

\[
\begin{bmatrix}
A_{\text{old}}^\top \\
A_{\text{new}}^\top
\end{bmatrix}
\]

be skinny and full rank. Equivalently, we require that \( A_{\text{new}} \) be skinny and full rank, and \( [A_{\text{old}} \quad A_{\text{new}}] \) be fat and full rank.

b) The optimal value of \( J \) is 3.2361; in comparison, the value of \( J \) achieved by \( A_{\text{new}}^\dagger \) is 8.0901, which is significantly higher.

c) The bit error rate using the equalizer \( B \) is 0.0333, while the bit error rate using \( A_{\text{new}}^\dagger \) is 0.1000. Thus, we see that \( B \) has a much lower bit error rate than \( A_{\text{new}}^\dagger \).
8.1460. **Filling-in missing data.** In this problem we have a signal, \( y_i \in \mathbb{R} \) for \( i = 1, \ldots, n \), which we view as \( y \in \mathbb{R}^n \). We will have \( n = 100 \). The signal \( y \) comes from measurements of a physical system, and so \( y_{i+1} \) is measured a short time interval after \( y_i \). Unfortunately, during the data acquisition process some of the data was lost and so the signal we have has gaps in it. Specifically, we have a known set \( K \subset \mathbb{Z} \) and we know \( y_i \) only for values \( i \in K \).

The data for this problem is in the file `missing.json`. The supplied vector `known` contains the list of known points \( K \), and the vector `yknown` is the list of values of \( y \) at the points in \( K \). The length of `yknown` is therefore \( |K| \).

a) For a signal \( z \in \mathbb{R}^n \), we define the discrete derivative \( z^{\text{der}} \in \mathbb{R}^{n-1} \) by

\[
z^{\text{der}}_i = z_{i+1} - z_{i} \quad \text{for } i = 1, \ldots, n-1
\]

Find the matrix \( G \) such that \( z^{\text{der}} = Gz \).

b) Our first approach will be to find the signal \( z \) which minimizes \( ||z^{\text{der}}|| \) and satisfies

\[
z_i = y_i \quad \text{if } i \in K
\]

Give a method finding the optimal \( z \).

c) Find the optimal \( z \) in the previous part and plot \( z_i \) against \( i \). Be sure to plot the points \((i, z_i)\), not just a line joining them.

d) One way to do a better job at filling in the missing data is to put additional criteria on our estimate. Here we will do this by additionally penalizing the second derivative of \( z \). Define the discrete second derivative \( z^{\text{hes}} \in \mathbb{R}^{n-2} \) by

\[
z^{\text{hes}}_i = z_{i+2} - 2z_{i+1} + z_{i} \quad \text{for } i = 1, \ldots, n-2
\]

Find the matrix \( H \) such that \( z^{\text{hes}} = Hz \).

e) Define the two objective functions

\[
J_1 = ||Gz||^2 \quad J_2 = ||Hz||^2
\]

We would like to find the signal \( z \) that minimizes

\[
J_1 + \mu J_2
\]

and satisfies

\[
z_i = y_i \quad \text{if } i \in K
\]

Give a method for finding the optimal \( z \).

f) Plot the trade-off curve of \( J_2 \) (on the vertical axis) versus \( J_1 \) (on the horizontal axis). Give the interpretation of the endpoints of this curve.

g) Find the optimal \( z \) for the three different cases \( \mu = 5, 20, 100 \).
Solution.

a) The matrix $G$ is

$$G_{ij} = \begin{cases} 
-1 & \text{if } i = j \\
1 & \text{if } j = i + 1 \\
0 & \text{otherwise}
\end{cases}$$

b) The problem is in the form

$$\begin{align*}
\text{minimize} & \quad \|Az - b\| \\
\text{subject to} & \quad Cz = d
\end{align*}$$

which (from the lecture notes) has solution

$$\begin{bmatrix}
x \\
\lambda
\end{bmatrix} = \begin{bmatrix} A^T A & C^T \\ C & 0 \end{bmatrix}^{-1} \begin{bmatrix} A^T b \\ d \end{bmatrix}$$

Here we have $A = G$, $b = 0$. The matrix $C$ consists of the rows $i$ of the identity matrix for which $i \in K$, and $d = y$.

c) The optimal $z$ is below.

![Graph of a function]

d) The matrix $H$ is

$$H_{ij} = \begin{cases} 
1 & \text{if } i = j \\
-2 & \text{if } j = i + 1 \\
1 & \text{if } j = i + 2 \\
0 & \text{otherwise}
\end{cases}$$

e) This also has the same form as part b). In this case

$$A = \begin{bmatrix} D \\ \sqrt{\pi}H \end{bmatrix}$$
f) The trade-off curve is

![Graph showing the trade-off curve with points at various values of $J_2$.]

The bottom right corner is the solution when we minimize $J_2$, with $J_1$ unconstrained. It is the solution with smallest second derivative that interpolates the data. Similarly, the upper left corner minimizes $J_1$, and is the solution with smallest first derivative that interpolates the data.

g) The optimal $z$ with $\mu = 5$ is below.

![Graph showing the optimal $z$ with $\mu = 5$.]

The optimal $z$ with $\mu = 20$ is below.

16
The optimal $z$ with $\mu = 100$ is below.