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 \( NE^2 \), 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

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

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

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 \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.(- (t .* 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])

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:

$$B = \begin{bmatrix} b_1^T \\
\vdots \\
 b_m^T \end{bmatrix}.$$ 

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 $B$:

$$J = \sum_{i=1}^n \sum_{j=1}^n (BA_{\text{old}} - I)_{ij}^2 = \sum_{i=1}^n \| (BA_{\text{old}} - I)_{ij} \|^2 = \sum_{i=1}^n \| A_{\text{old}}^T b_i - e_i \|^2.$$ 

Thus, we want to solve the following optimization problem.

$$\begin{array}{l}
\text{minimize} \quad \sum_{i=1}^n \| A_{\text{old}}^T b_i - e_i \|^2 \\
\text{subject to} \quad A_{\text{new}}^T b_i = e_i, \quad i = 1, \ldots, n
\end{array}$$

This problem is separable in the rows of $B$, allowing us to decompose it into $n$ vector optimization problems:

$$\begin{array}{l}
\text{minimize} \quad \| A_{\text{old}}^T b_i - e_i \|^2 \\
\text{subject to} \quad A_{\text{new}}^T b_i = e_i
\end{array}$$

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 $A_{\text{new}}^T$ be fat and full rank, and

$$\begin{bmatrix} A_{\text{old}}^T \\
A_{\text{new}}^T \end{bmatrix}$$

be skinny and full rank. Equivalently, we require that $A_{\text{new}}$ be skinny and full rank, and

$$\begin{bmatrix} A_{\text{old}} & A_{\text{new}} \end{bmatrix}$$

be fat and full rank.
b) The optimal value of $J$ is 3.2361; in comparison, the value of $J$ achieved by $A^\dagger_{\text{new}}$ 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 $A^\dagger_{\text{new}}$ is 0.1000. Thus, we see that $B$ has a much lower bit error rate than $A^\dagger_{\text{new}}$.

10.1550. Some basic properties of eigenvalues. Show the following:

a) The eigenvalues of $A$ and $A^T$ are the same.

b) $A$ is invertible if and only if $A$ does not have a zero eigenvalue.
c) If the eigenvalues of $A$ are $\lambda_1, \ldots, \lambda_n$ and $A$ is invertible, then the eigenvalues of $A^{-1}$ are $1/\lambda_1, \ldots, 1/\lambda_n$.

d) The eigenvalues of $A$ and $T^{-1}AT$ are the same.

Hint: you’ll need to use the facts that $\det A = \det(A^T)$, $\det(AB) = \det A\det B$, and, if $A$ is invertible, $\det A^{-1} = 1/\det A$.

Solution.

a) The eigenvalues of a matrix $A$ are given by the roots of the polynomial $\det(sI - A)$.

From determinant properties we know that $\det(sI - A) = \det(sI - A)^T = \det(sI - A^T)$. We conclude that the eigenvalues of $A$ and $A^T$ are the same.

b) First we recall that $A$ is invertible if and only if $\det(A) \neq 0$. But $\det(A) \neq 0 \iff \det(-A) \neq 0$.

i. If $0$ is an eigenvalue of $A$, then $\det(sI - A) = 0$ when $s = 0$. It follows that $\det(-A) = 0$ and thus $\det(A) = 0$, and $A$ is not invertible. From this fact we conclude that if $A$ is invertible, then $0$ is not an eigenvalue of $A$.

ii. If $A$ is not invertible, then $\det(A) = \det(-A) = 0$. This means that, for $s = 0$, $\det(sI - A) = 0$, and we conclude that in this case $0$ must be an eigenvalue of $A$. From this fact it follows that if $0$ is not an eigenvalue of $A$, then $A$ is invertible.

c) From the results of the last item we see that $0$ is not an eigenvalue of $A$. Now consider the eigenvalue/eigenvector pair $(\lambda_i, x_i)$ of $A$. This pair satisfies $Ax_i = \lambda_ix_i$. Now, since $A$ is invertible, $\lambda_i$ is invertible. Multiplying both sides by $A^{-1}$ and $\lambda_i^{-1}$ we have $\lambda_i^{-1}x_i = A^{-1}x_i$, and from this we conclude that the eigenvalues of the inverse are the inverse of the eigenvalues.

d) First we note that $\det(sI - A) = \det(I(sI - A)) = \det(T^{-1}T(sI - A))$. Now, from determinant properties, we have $\det(T^{-1}T(sI - A)) = \det(T^{-1}(sI - A)T)$. But this is also equal to $\det(sI - T^{-1}AT)$, and the conclusion is that the eigenvalues of $A$ and $T^{-1}AT$ are the same.

11.1860. Eigenvalues of matrix products. Suppose that $A \in \mathbb{R}^{m \times n}$ and $B \in \mathbb{R}^{n \times m}$. Show that if $\lambda \in \mathbb{C}$ is a nonzero eigenvalue of $AB$, then $\lambda$ is also an eigenvalue of $BA$. Conclude that the nonzero eigenvalues of $AB$ and $BA$ are the same. Hint: Suppose that $ABv = \lambda v$, where $v \neq 0$, $\lambda \neq 0$. Construct a $w \neq 0$ for which $BAw = \lambda w$.

Solution. Suppose $ABv = \lambda v$, where $v \neq 0$ and $\lambda \neq 0$. We can write

$$BA(Bv) = B(ABv) = B(\lambda v) = \lambda Bv.$$ 

Since $\lambda \neq 0$, this implies that $Bv \neq 0$ (otherwise $ABv = 0$). Thus $\lambda$ is also an eigenvalue of $BA$, and $Bv$ is a corresponding eigenvector.

Note that the same argument does not hold if $\lambda = 0$. In that case, it is possible that $Bv = 0$, so $Bv$ cannot be an eigenvector (which is nonzero by definition).