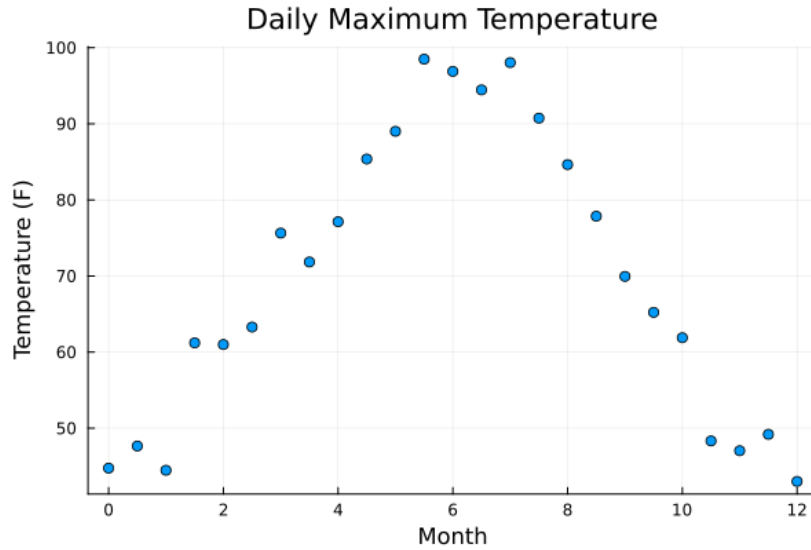


EE263 Homework 6

Fall 2025

6.3200. 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 $n + 1 = 25$ 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, \dots, 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 $j = 1, \dots, m$ 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 $\alpha_0, \dots, \alpha_m$ to minimize

$$\sum_{i=0}^n \|y_i - g(x_i)\|_2^2.$$

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

$$J^{\text{test}} = \sum_{i=0}^{n_{\text{test}}} \|y_i^{\text{test}} - g(x_i^{\text{test}})\|_2^2.$$

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

$$\sum_{i=0}^n \|y_i - g(x_i)\|_2^2$$

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

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

So we define

$$y = \begin{bmatrix} y_0 \\ \vdots \\ y_n \end{bmatrix} \quad F = \begin{bmatrix} f_0(x_0) & f_1(x_0) & \dots & f_m(x_0) \\ f_0(x_1) & f_1(x_1) & \dots & f_m(x_1) \\ \vdots & \vdots & \ddots & \vdots \\ f_0(x_n) & f_1(x_n) & \dots & f_m(x_n) \end{bmatrix} \quad \alpha = \begin{bmatrix} \alpha_0 \\ \vdots \\ \alpha_m \end{bmatrix}$$

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 $q \in \mathbb{R}^a$ 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

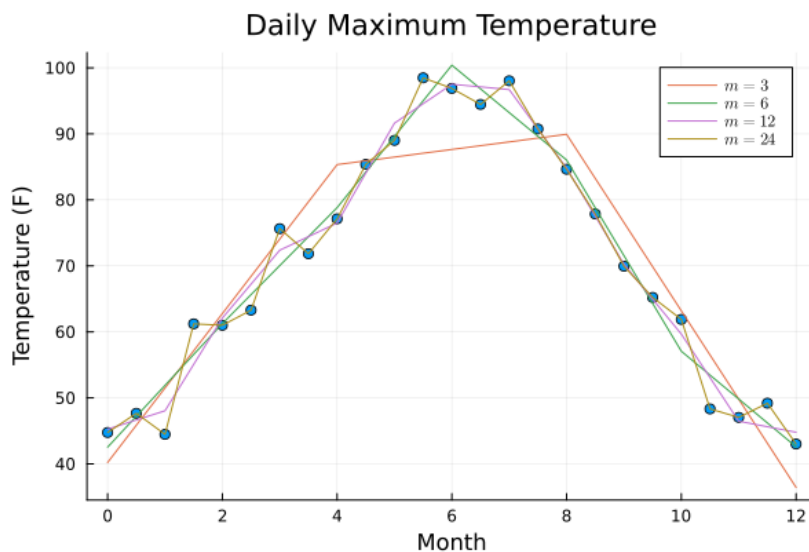
$$z = \begin{bmatrix} z_0 \\ z_1 \\ \vdots \\ z_m \end{bmatrix}$$

with $z_0 \in \mathbb{R}$ and $z_i \in \mathbb{R}^a$ for $i > 0$. Suppose $z = Fx = 0$, then since $z_0 = 0$ we have $x_0 = 0$. Then

$$z_1 = x_0 + qx_1$$

and since $z_1 = 0$ and $x_0 = 0$ we must have $x_1 = 0$. 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 = []

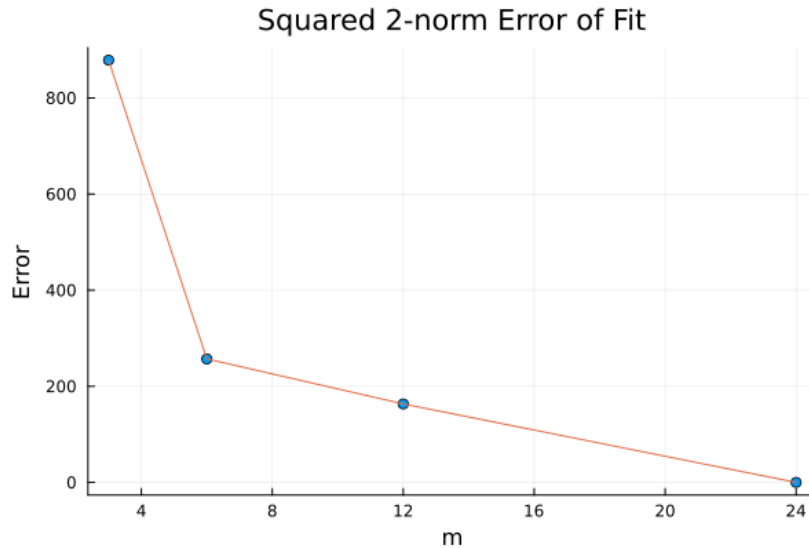
    for j=0:m
        function f(x)
            if x < 12*(j-1)/m
                return 0.0
            elseif x > 12*j/m
                return 1.0
            else
                return m*x/12-(j-1)
            end
        end

        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.

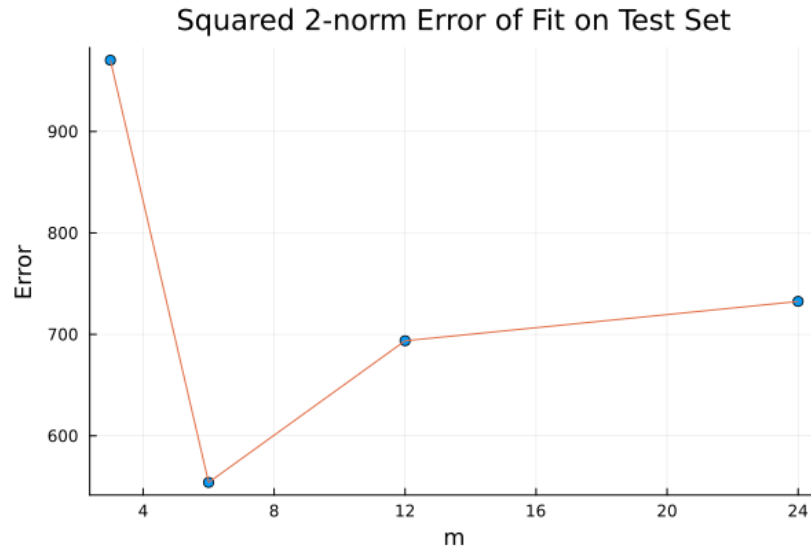


It's expected that students simply modify their code from the previous part to also record the squared 2-norm error between the fitted function and the data. The following code can generate this plot.

```
#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 begins 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)

```

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, μ is called its *center*, and σ 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,

$$t_1, \dots, t_N, \quad y_1, \dots, y_N,$$

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 , μ , σ , *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 , μ , and σ . 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

$$f(t) + (\nabla_p f(t))^\top \Delta p,$$

where ∇_p denotes the gradient with respect to p . These partial derivatives are:

$$\begin{aligned} \frac{\partial}{\partial a} f(t) &= e^{-(t-\mu)^2/\sigma^2} \\ \frac{\partial}{\partial \mu} f(t) &= \frac{2a(t-\mu)}{\sigma^2} e^{-(t-\mu)^2/\sigma^2} \\ \frac{\partial}{\partial \sigma} f(t) &= \frac{2a(t-\mu)^2}{\sigma^3} e^{-(t-\mu)^2/\sigma^2} \end{aligned}$$

The Gauss-Newton method proceeds as follows. We find Δp that minimizes

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

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

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

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

$$\Delta p = (A^\top A)^{-1} A^\top b.$$

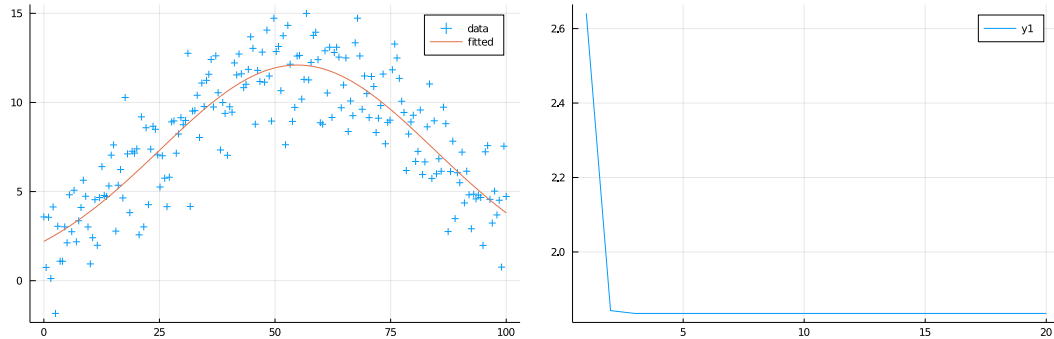
To summarize, the algorithm repeats the following steps:

- 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 Δp .
- Update the parameter vector: $p := p + \Delta p$.

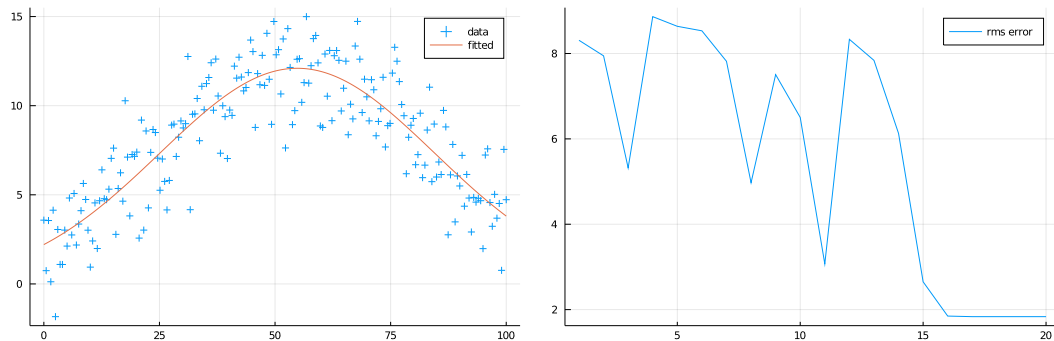
This can be repeated until the update Δp is small, or the improvement in E is small.

- 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.

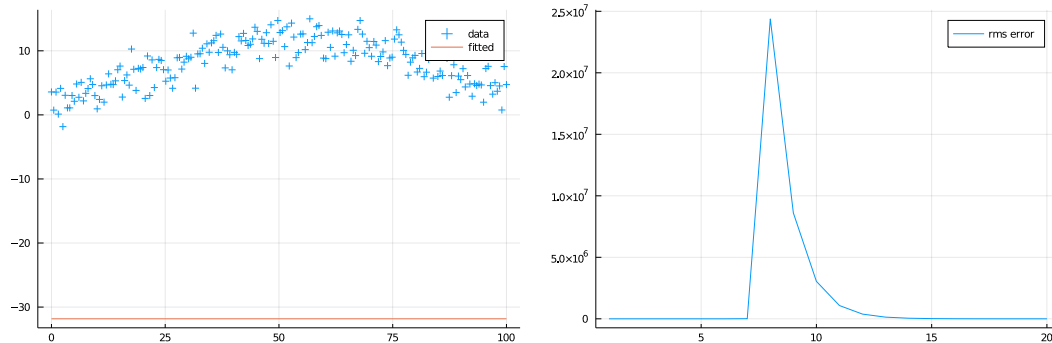


Now we try with another starting point, $p = (10, 20, 10)$. The final fit is the same (well, σ landed on -42.02 , but that doesn't matter). It does tend to bounce around a bit more before converging, which is indicative of its nonlinearity. That it landed in the same place bolsters our confidence that 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.

```
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")

```

8.1460. Filling-in missing data. In this problem we have a signal, $y_i \in \mathbb{R}$ for $i = 1, \dots, n$, 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

$$z_i^{\text{der}} = z_{i+1} - z_i \quad \text{for } i = 1, \dots, n-1$$

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

$$z_i^{\text{hes}} = z_{i+2} - 2z_{i+1} + z_i \quad \text{for } i = 1, \dots, n-2$$

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

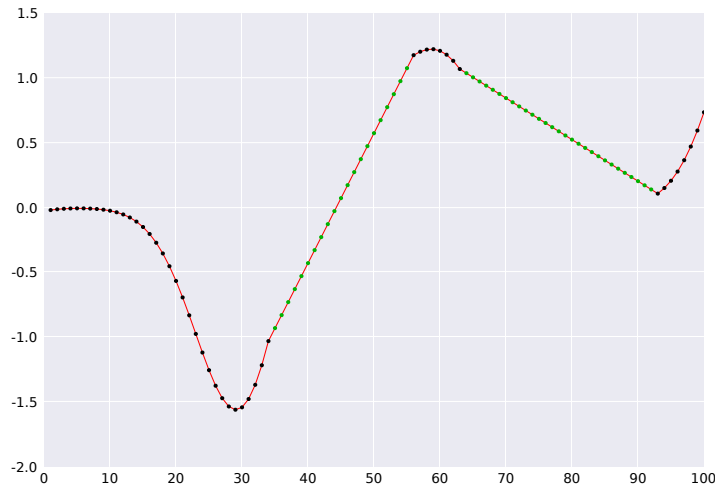
$$\begin{aligned} & \text{minimize} && \|Az - b\| \\ & \text{subject to} && Cz = d \end{aligned}$$

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.



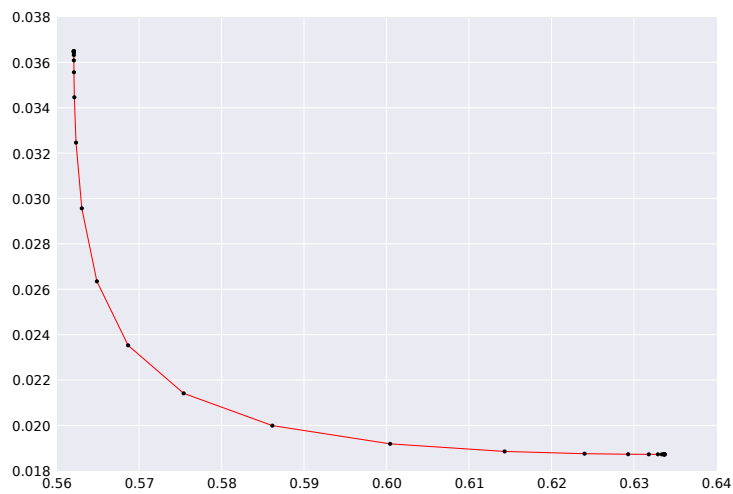
d) The matrix H is

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

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

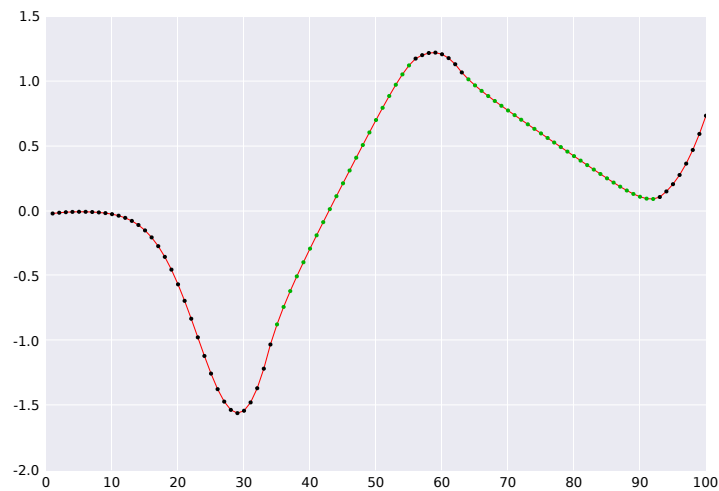
$$A = \begin{bmatrix} D \\ \sqrt{\mu}H \end{bmatrix}$$

f) The trade-off curve is

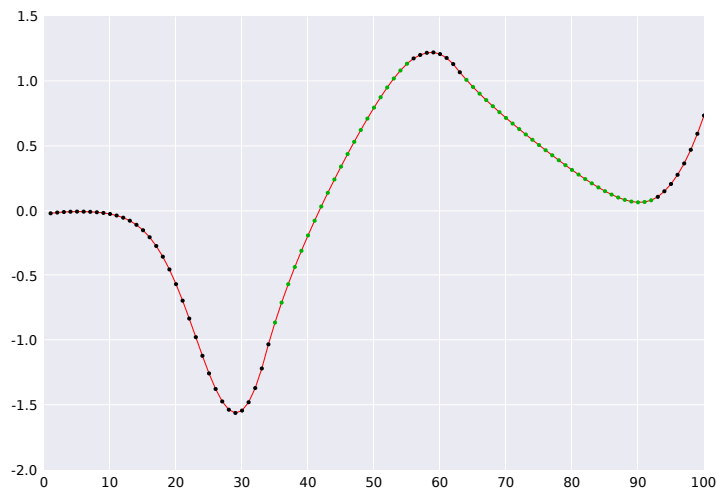


The bottom right corner is the solution when we minimize J_2 , with J_1 unconstrained. It is the solution with smallest second derivative that interpolates the data. Similarly the upper left corner minimizes J_1 , and is the solution with smallest first derivative that interpolates the data.

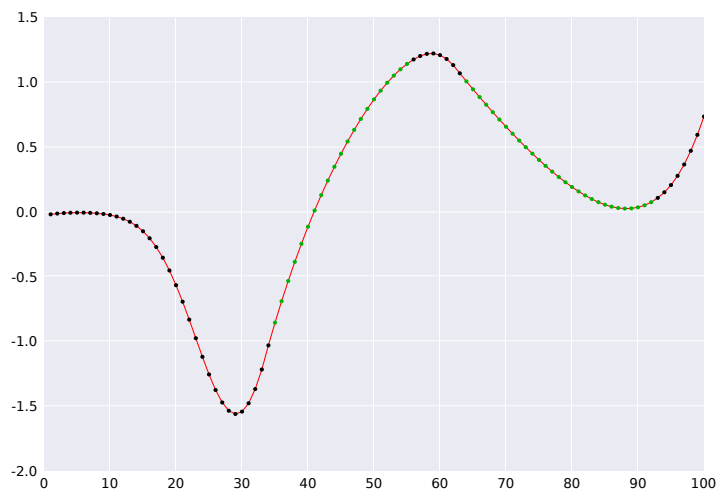
g) The optimal z with $\mu = 5$ is below.



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



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



Here is the Julia code that solves this problem.

```
using LinearAlgebra, Plots
include("readclassjson.jl")

data = readclassjson("../data/missing.json")
known = data["known"]
K = size(known)[1]
y = data["y"]
```

```

n = maximum(known)
# part (a)
G = [-I zeros(n-1)] + [zeros(n-1) I]

# construct the matrix C for the constraint
C = zeros(K,n)
for k=1:K
    C[k,known[k]] = 1
end

# parts (b), (c)
A_big = [2.0*G'*G C'
         C         zeros(K,K)]
b_big = [zeros(n); y]
z1 = (A_big \ b_big)[1:n]

plot(z1)
scatter!(known, y)

# part (d), (e)
H = [I zeros(n-2,2)] + [zeros(n-2) -2.0*I zeros(n-2)] + [zeros(n-2,2) I]

function find_z(mu)
    A_big = [2.0*G'*G + mu*2.0*H'*H C'
            C         zeros(K,K)]
    b_big = [zeros(n); y]
    z = (A_big \ b_big)[1:n]
    return z
end

# part (f) Plot the tradeoff curve
mus = [0.0 0.001 0.005 0.01 0.05 0.1 0.5 1.0 5.0 10.0 50.0 100.0]
J1 = []
J2 = []
for mu in mus
    z = find_z(mu)
    push!(J1, norm(G*z)^2)
    push!(J2, norm(H*z)^2)
end
plot(J1, J2)
xlabel!("J1"); ylabel!("J2")
# The endpoints of the curve correspond to the places
# where one or the other objective dominates.
# the left endpoint is dominated by J2
# the right endpoint is dominated by J1

```

```

# part (g) find the optimal z for 3 cases
z_5 = find_z(5.0)
z_20 = find_z(20.0)
z_100 = find_z(100.0)
plot(z_5, label="mu=5")
plot!(z_20, label="mu=20")
plot!(z_100, label="mu=100")
scatter!(known, y, label="Known")

```

11.1830. Estimating a matrix with known eigenvectors. This problem is about estimating a matrix $A \in \mathbb{R}^{n \times n}$. The matrix A is not known, but we do have a noisy measurement of it, $A^{\text{meas}} = A + E$. Here the matrix E is measurement error, which is assumed to be small. While A is not known, we do know real, independent eigenvectors v_1, \dots, v_n of A . (Its eigenvalues $\lambda_1, \dots, \lambda_n$, however, are not known.) We will combine our measurement of A with our prior knowledge to find an estimate \hat{A} of A . To do this, we choose \hat{A} as the matrix that minimizes

$$J = \frac{1}{n^2} \sum_{i,j=1}^n (A_{ij}^{\text{meas}} - \hat{A}_{ij})^2$$

among all matrices which have eigenvectors v_1, \dots, v_n . (Thus, \hat{A} is the matrix closest to our measurement, in the mean-square sense, that is consistent with the known eigenvectors.)

- a) Explain how you would find \hat{A} . If your method is iterative, say whether you can guarantee convergence. Be sure to say whether your method finds the exact minimizer of J (except, of course, for numerical error due to roundoff), or an approximate solution. You can use any of the methods (least-squares, least-norm, Gauss-Newton, low rank approximation, *etc.*) or decompositions (QR, SVD, eigenvalue decomposition, *etc.*) from the course.
- b) Carry out your method with the data

$$A^{\text{meas}} = \begin{bmatrix} 2.0 & 1.2 & -1.0 \\ 0.4 & 2.0 & -0.5 \\ -0.5 & 0.9 & 1.0 \end{bmatrix}, \quad v_1 = \begin{bmatrix} 0.7 \\ 0 \\ 0.7 \end{bmatrix}, \quad v_2 = \begin{bmatrix} 0.3 \\ 0.6 \\ 0.7 \end{bmatrix}, \quad v_3 = \begin{bmatrix} 0.6 \\ 0.6 \\ 0.3 \end{bmatrix}.$$

Be sure to check that \hat{A} does indeed have v_1, v_2, v_3 as eigenvectors, by (numerically) finding its eigenvectors and eigenvalues. Also, give the value of J for \hat{A} . **Hint.** You might find the following useful (but then again, you might not.) In Julia, if \mathbf{A} is a matrix, then $\mathbf{A}[:,]$ is a (column) vector consisting of all the entries of \mathbf{A} , written out column by column. Therefore $\text{norm}(\mathbf{A}[:,])$ gives the squareroot of the sum of the squares of entries of the matrix \mathbf{A} , *i.e.*, its Frobenius norm. The inverse operation, *i.e.*, writing a vector out as a matrix with some given dimensions, is done using the function `reshape`. For example, if \mathbf{a} is an mn vector, then `reshape(a, m, n)` is an $m \times n$ matrix, with elements taken from \mathbf{a} (column by column).

Solution.

a) The matrix \hat{A} must have the form

$$\hat{A} = V\Lambda V^{-1},$$

where $\Lambda = \text{diag}(\lambda_1, \dots, \lambda_n)$ and $V = [v_1 \ \dots \ v_n]$. Here we know V and V^{-1} ; the eigenvalues $\lambda_1, \dots, \lambda_n$ are the unknowns to be determined. We will use W to denote $W = V^{-1}$, and we'll express its rows as $w_1^\top, \dots, w_n^\top$, *i.e.*,

$$W = \begin{bmatrix} w_1^\top \\ \vdots \\ w_n^\top \end{bmatrix}.$$

Then we can express $\hat{A} = V\Lambda V^{-1}$ as

$$\hat{A} = \sum_{i=1}^n \lambda_i R^{(i)}$$

where $R^{(i)} = v_i w_i^\top$. (This is the spectral decomposition.) Note that \hat{A} is a linear function of the variables here, *i.e.*, $\lambda_1, \dots, \lambda_n$. We will show that our problem is now a completely standard least-squares problem, with unknowns $\lambda_1, \dots, \lambda_n$. Let's define $\text{vec}(S)$, where S is an $n \times n$ matrix, as a vector with n^2 entries, which are just the entries of S written out column by column (or row by row; you just need a fixed method for writing out all entries of a matrix as a vector). Then we have

$$n^2 J = \sum_{i,j=1}^n \left(A_{ij}^{\text{meas}} - \sum_{k=1}^n \lambda_k R_{ij}^{(k)} \right)^2 = \|F\lambda - g\|^2,$$

where

$$F = [\text{vec}(R^{(1)}) \ \dots \ \text{vec}(R^{(n)})], \quad g = \text{vec}(A^{\text{meas}}).$$

(The dimensions are $F \in \mathbb{R}^{n^2 \times n}$ and $g \in \mathbb{R}^{n^2}$.) The solution is given by

$$\lambda = (F^\top F)^{-1} F^\top g.$$

We can work out the $n \times n$ matrix $F^\top F$, which has an interesting form. (But you didn't really need to do it.) We have

$$(F^\top F)_{ij} = \text{vec}(R^{(i)})^\top \text{vec}(R^{(j)}) = v_i^\top v_j w_i^\top w_j.$$

We also have

$$(F^\top g)_i = v_i^\top A^{\text{meas}} w_i.$$

These expressions can be written out compactly using the so-called *Hadamard product* of matrices, defined as follows: $(A \circ B)_{ij} = A_{ij} B_{ij}$. (Thus, the Hadamard product of two matrices is just the element-by-element product.) We can express $F^\top F$ as $F^\top F = (V^\top V) \circ (W W^\top)$. We can express $F^\top g$ as $F^\top g = \text{diag}(V^\top A^{\text{meas}} W^\top)$, where $\text{diag}()$ extracts the diagonal part of a matrix. This gives us the formula

$$\lambda = \left((V^\top V) \circ (W W^\top) \right)^{-1} \text{diag}(V^\top A^{\text{meas}} W^\top).$$

(Of course we did not expect you to mention or use this notation!)

- b) The following Julia code computes \hat{A} based on the solution method described above. Note that $\text{vec}(S)$ (a vector with the entries of S written out column by column) can be obtained using Julia command `S[:]` as suggested in the problem hint.

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

```
using LinearAlgebra
Ameas = [2.0 1.2 -1.0; 0.4 2.0 -0.5; -0.5 0.9 1.0]
V = [0.7 0.3 0.6; 0.0 0.6 0.6; 0.7 0.7 0.3]
W = inv(V)
n = 3

F = hcat((V[:,i:i] * W[i:i,:])[:] for i in 1:n...)
g = Ameas[:]
@assert rank(F) == size(F, 2) < size(F, 1)
lambda = F \ g

Ahat = V * diagm(lambda) * W
Jhat = norm(Ameas[:] - Ahat[:])^2 / n^2

eigen(Ahat)

# check eigenvectors match
V ./ norm.(eachcol(V))'
```

Here we give our \hat{A} solution and check that it indeed has eigenvectors v_1, v_2 , and v_3 . The minimum value for J is $J_{\min} = 0.0175$. A common error is forgetting $1/n^2$ which would yield $n^2 J_{\min} = 0.157$.

```
>> lambda = F \ g
3-element Array{Float64,1}:
 0.7840290263886044
 1.80014104068647
 2.4158299329249253

>> Ahat = V * diagm(lambda) * W
3×3 Array{Float64,2}:
 1.74724  1.1502  -0.96321
 0.527733 2.15196 -0.527733
-0.316769 0.974285  1.1008

>> Jhat = norm(Ameas[:] - Ahat[:])^2 / n^2
0.017461333438605436
```

```

>> eigen(Ahat)
Eigen{Float64,Float64,Array{Float64,2},Array{Float64,1}}
values:
3-element Array{Float64,1}:
 0.7840290263886041
 1.80014104068647
 2.4158299329249258
vectors:
3×3 Array{Float64,2}:
 0.707107    0.309426  -0.666667
 4.16334e-16 0.618853  -0.666667
 0.707107    0.721995  -0.333333

>> V ./ norm.(eachcol(V))' # check normalized target eigenvectors
3×3 Array{Float64,2}:
 0.707107  0.309426  0.666667
 0.0       0.618853  0.666667
 0.707107  0.721995  0.333333

```

15.2200. Properties of symmetric matrices. In this problem P and Q are symmetric matrices. For each statement below, either give a proof or a specific counterexample.

- a) If $P \geq 0$ then $P + Q \geq Q$.
- b) If $P \geq Q$ then $-P \leq -Q$.
- c) If $P > 0$ then $P^{-1} > 0$.
- d) If $P \geq Q > 0$ then $P^{-1} \leq Q^{-1}$.
- e) If $P \geq Q$ then $P^2 \geq Q^2$.

Hint: you might find it useful for part (d) to prove $Z \geq I$ implies $Z^{-1} \leq I$.

Solution.

- a) By definition, $A \geq B$ if and only if $A - B \geq 0$. So, if $P \geq 0$, then $P + Q - Q \geq 0$ and therefore $P + Q \geq Q$.
- b) If $P \geq Q$ then $P - Q \geq 0$, and by definition $-(P - Q) \leq 0$ or $-P + Q \leq 0$ so finally $-Q \geq -P$.
- c) If $P > 0$ then all eigenvalues of P are strictly positive and P^{-1} exists. If $\lambda_1, \dots, \lambda_n > 0$ are the eigenvalues of P then the eigenvalues of P^{-1} are $1/\lambda_1, \dots, 1/\lambda_n$. Since $\lambda_i > 0$ then $1/\lambda_i > 0$ so the eigenvalues of P^{-1} are all positive and therefore $P^{-1} > 0$.

- d) First we prove the hint, *i.e.*, if $Z \geq I$ then $Z^{-1} \leq I$. Suppose the eigenvalues of $Z \in \mathbb{R}^{n \times n}$ are $\lambda_1, \dots, \lambda_n$. Then the eigenvalues of $Z - I$ are $\lambda_1 - 1, \dots, \lambda_n - 1$ because if v_i is the eigenvector associated with λ_i then

$$(Z - I)v_i = Zv_i - v_i = \lambda_i v_i - v_i = (\lambda_i - 1)v_i.$$

By definition, if $(\lambda_i - 1)v_i = (Z - I)v_i$ then $\lambda_i - 1$ is an eigenvalue of $Z - I$. Since $Z \geq I$ or $Z - I \geq 0$ then all eigenvalues of $Z - I$ are nonnegative or $\lambda_i \geq 1$.

The eigenvalues of Z^{-1} are $1/\lambda_1, \dots, 1/\lambda_n$ and from $\lambda_i \geq 1$ we conclude that $1/\lambda_i \leq 1$ or the eigenvalues of Z^{-1} are all less than or equal to 1.

The eigenvalues of $Z^{-1} - I$ are $1/\lambda_1 - 1, \dots, 1/\lambda_n - 1$ and therefore are all nonpositive. Hence $Z^{-1} - I \leq 0$ or $Z^{-1} \leq I$ and we are done.

Now we prove that $P \geq Q > 0$ implies that $P^{-1} \leq Q^{-1}$ or $P^{-1} - Q^{-1} \leq 0$.

To apply the hint $Z - I \geq 0$ to the expression $P - Q \geq 0$, we would like to “turn Q into I ” using Q ’s eigenvalue decomposition’ $Q = U\Lambda U^T$. Since $Q > 0$ then $\Lambda > 0$ and therefore $Q^{-1/2} = Q^{-T/2} = U\Lambda^{-1/2}U^T$ exists (if $\Lambda \geq 0$, this would not work because it would be possible to have $\lambda_i = 0$ which would make $1/\lambda_i$ undefined).

By congruence, $P - Q \geq 0$ implies that

$$\left(Q^{-1/2}\right)^T (P - Q) Q^{-1/2} \geq 0$$

or

$$\left(Q^{-1/2}\right)^T P Q^{-1/2} - \left(Q^{-1/2}\right)^T Q Q^{-1/2} \geq 0$$

and therefore

$$\left(Q^{-1/2}\right)^T P Q^{-1/2} - I \geq 0.$$

Now according to the hint (take $Z = \left(Q^{-1/2}\right)^T P Q^{-1/2}$) we have

$$\left(\left(Q^{-1/2}\right)^T P Q^{-1/2}\right)^{-1} - I \leq 0$$

which simplifies to

$$Q^{1/2} P^{-1} \left(Q^{1/2}\right)^T - I \leq 0.$$

To recover the desired $P^{-1} - Q^{-1} \leq 0$, we multiply our expression with $Q^{-1/2}$ and $\left(Q^{-1/2}\right)^T$. Again by congruence this implies

$$Q^{-1/2} \left(Q^{1/2} P^{-1} \left(Q^{1/2}\right)^T - I\right) \left(Q^{-1/2}\right)^T \leq 0$$

which simplifies to

$$P^{-1} - Q^{-1/2} \left(Q^{-1/2}\right)^T \leq 0$$

and (since Q is symmetric)

$$P^{-1} - Q^{-1} \leq 0.$$

- e) The statement is false. A simple counterexample is $P = -1$ and $Q = -2$.