Homework 5 solutions

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.*, $p = [a\ \mu\ \sigma]^\mathsf{T}$. We are given a set of data,

$$t_1, \ldots, t_N, \qquad y_1, \ldots, y_N,$$

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, 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 $\Delta p^{(k)} = [\Delta a^{(k)}\ \Delta\mu^{(k)}\ \Delta\sigma^{(k)}]^\mathsf{T}$ to denote the update to the parameters, *i.e.*,

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

(Here k denotes the kth 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

$$f(t) + (\nabla_p f(t))^{\mathsf{T}} \Delta p,$$

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

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

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

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

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

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

To summarize, the algorithm repeats the following steps:

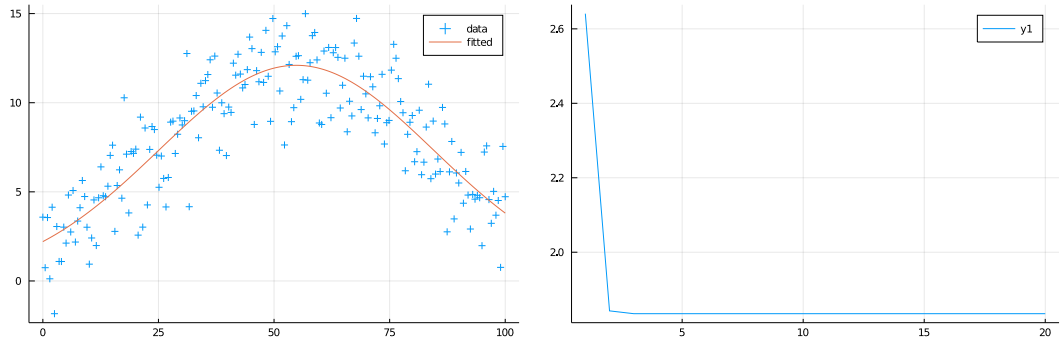
- Evaluate the vector b (which is the vector of fitting residuals.) Evaluate the partial derivatives to form the matrix A.
- Solve the least-squares problem to get Δp.
- Update the parameter vector: $p := p + \Delta p$.

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

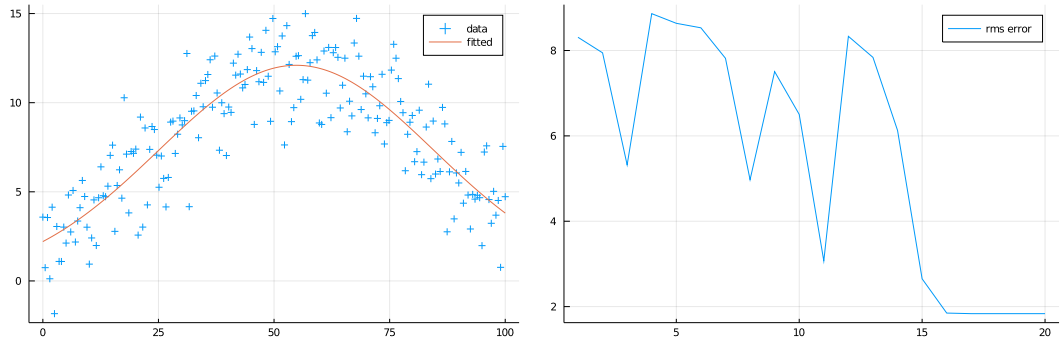
b) We used the starting parameter values $p = [11, 50, 35]^{\mathsf{T}}$, estimated visually. The amplitude $a = 11$ was estimated as a guess for the (noise-free) peak of the graph, $\mu = 50$ was estimated as its center, and $\sigma = 35$ was estimated from its spread.

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

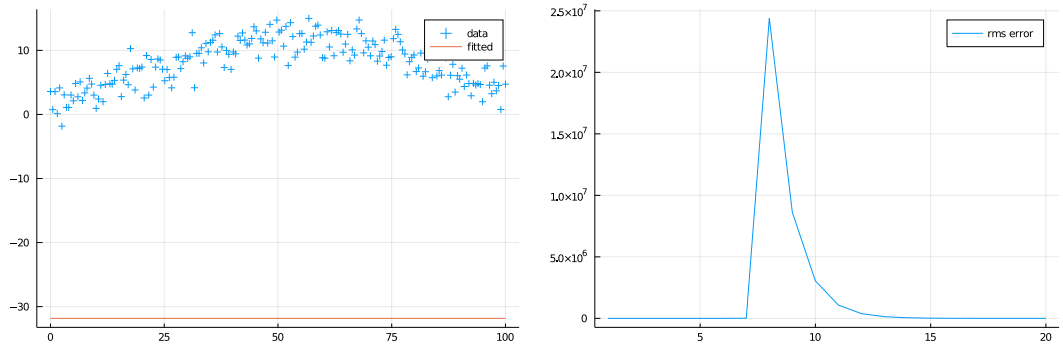
happens very quickly, in just a handful of iterations.



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")
```

8.1320. Portfolio selection with sector neutrality constraints. We consider the problem of selecting a portfolio composed of n assets. We let $x_i \in \mathbb{R}$ denote the investment (say, in dollars) in asset i, with $x_i < 0$ meaning that we hold a short position in asset i. We normalize our total portfolio as $\mathbf{1}^\mathsf{T} x = 1$, where $\mathbf{1}$ is the vector with all entries 1. (With normalization, the x_i are sometimes called *portfolio weights*.)

The portfolio (mean) return is given by $r = \mu^\mathsf{T} x$, where $\mu \in \mathbb{R}^n$ is a vector of asset (mean) returns. We want to choose x so that r is large, while avoiding risk exposure, which we explain

next.

First we explain the idea of *sector exposure*. We have a list of k economic sectors (such as manufacturing, energy, transportation, defense, ...). A matrix $F \in \mathbb{R}^{k\times n}$, called the *factor loading matrix*, relates the portfolio x to the *factor exposures*, given as $R^{\text{fact}} = Fx \in \mathbb{R}^k$. The number R_i^{fact} is the portfolio risk exposure to the ith economic sector. If R_i^{fact} is large (in magnitude) our portfolio is exposed to risk from changes in that sector; if it is small, we are less exposed to risk from that sector. If $R_i^{\text{fact}} = 0$, we say that the portfolio is *neutral* with respect to sector i.

Another type of risk exposure is due to fluctations in the returns of the individual assets. The *idiosyncratic risk* is given by

$$R^{\text{id}} = \sum_{i=1}^{n} \sigma_i^2 x_i^2,$$

where $\sigma_i > 0$ are the standard deviations of the asset returns. (You can take the formula above as a definition; you do not need to understand the statistical interpretation.)

We will choose the portfolio weights x so as to maximize $r - \lambda R^{\text{id}}$, which is called the *risk-adjusted return*, subject to neutrality with respect to all sectors, *i.e.*, $R^{\text{fact}} = 0$. Of course we also have the normalization constraint $\mathbf{1}^\mathsf{T} x = 1$. The parameter λ, which is positive, is called the *risk aversion parameter*. The (known) data in this problem are $\mu \in \mathbb{R}^n$, $F \in \mathbb{R}^{k\times n}$, $\sigma = (\sigma_1, \ldots, \sigma_n) \in \mathbb{R}^n$, and $\lambda \in \mathbb{R}$.

a) Explain how to find x, using methods from the course. You are welcome (even encouraged) to express your solution in terms of block matrices, formed from the given data.

b) Using the data given in `sector_neutral_portfolio_data.json`, find the optimal portfolio. Report the associated values of r (the return), and R^{id} (the idiosyncratic risk). Verify that $\mathbf{1}^\mathsf{T} x = 1$ (or very close) and $R^{\text{fact}} = 0$ (or very small).

Solution.

a) We define $\Sigma \in \mathbb{R}^{n\times n}$ to be a diagonal matrix with $\Sigma_{ii} = \sigma_i^2$, so $R^{\text{id}} = x^\mathsf{T}\Sigma x$. The problem we are trying to solve is

$$\begin{array}{ll} \text{maximize} & \mu^\mathsf{T} x - \lambda x^\mathsf{T}\Sigma x \\ \text{subject to} & \mathbf{1}^\mathsf{T} x = 1, \quad Fx = 0 \end{array} \tag{1}$$

with variable $x \in \mathbb{R}^n$. Maximizing an objective is equivalent to minimizing the negative of the objective, so we can rewrite this as

$$\begin{array}{ll} \text{minimize} & -\mu^\mathsf{T} x + \lambda x^\mathsf{T}\Sigma x \\ \text{subject to} & \mathbf{1}^\mathsf{T} x = 1, \quad Fx = 0 \end{array}$$

We introduce Lagrange multipliers $\kappa \in \mathbb{R}$ and $\nu \in \mathbb{R}^k$ for the two constraints, and write the Lagrangian of this problem as

$$L(x, \nu, \kappa) = -\mu^\mathsf{T} x + \lambda x^\mathsf{T}\Sigma x + \nu^\mathsf{T}(Fx) + \kappa(\mathbf{1}^\mathsf{T} x - 1).$$

The optimality conditions are then given by

$$\nabla_x L = -\mu + 2\lambda\Sigma x + F^\mathsf{T}\nu + \kappa\mathbf{1} = 0, \qquad \nabla_\nu L = Fx = 0, \qquad \nabla_\kappa L = \mathbf{1}^\mathsf{T}x - 1 = 0,$$

which we can write in block matrix form as

$$\begin{bmatrix} 2\lambda\Sigma & F^\mathsf{T} & \mathbf{1} \\ F & 0 & 0 \\ \mathbf{1}^\mathsf{T} & 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ \nu \\ \kappa \end{bmatrix} = \begin{bmatrix} \mu \\ 0 \\ 1 \end{bmatrix}.$$

To find the optimal x we solve this set of $n+k+1$ linear equations in $n+k+1$ variables.

Alternate method: Another method is to note that

$$\left\| \sqrt{\lambda}\Sigma^{1/2}x - \frac{1}{2\sqrt{\lambda}}\Sigma^{-1/2}\mu \right\|^2 = \lambda x^\mathsf{T}\Sigma x - \mu^\mathsf{T}x + \frac{1}{4\lambda}\mu^\mathsf{T}\Sigma^{-1}\mu$$

and the last term is a constant, so minimzing the left-hand side is equivalent to minimizing $-\mu^\mathsf{T}x + \lambda x^\mathsf{T}\Sigma x$. Then let $A = \sqrt{\lambda}\Sigma^{1/2}$, $b = \frac{1}{2\sqrt{\lambda}}\Sigma^{-1/2}\mu$, $C = \begin{bmatrix} \mathbf{1}^\mathsf{T} \\ F \end{bmatrix}$, $d = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$, and we can minimize $\|Ax - b\|$ subject to $Cx = d$ using the general norm minimization procedure.

b) Using the data provided we arrive at optimal values $r = 26.71$, and $R^{\text{id}} = 133.6$, with the optimal objective value of the original problem being 13.34. The following Julia code implements the method of part (a).

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("sector_neutral_portfolio_data.json")

n = data["n"]
k = data["k"]
lambda = data["lambda"]
sigma = data["sigmas"]
F = data["F"]
mu = data["mu"]

# block matrix method
sigma = diagm(sigma.^2)
M = [  2lambda*sigma        F'     ones(n,1);
          F     zeros(k,k) zeros(k,1);
     ones(1,n) zeros(1,k)     0    ]
v = [mu; zeros(k,1); 1]
```

```
@assert rank(M) == size(M, 1) == size(M, 2) == n + k + 1
y = M \ v
x = y[1:n]
@show r = mu'*x
@show Rid = lambda*x'*sigma*x
@show obj = r - Rid


# completing the square method
A = sqrt(lambda) * diagm(sigma)
b = diagm(1 ./ sigma) * mu / (2*sqrt(lambda))
C = [ones(1, n); F]
d = [1; zeros(k, 1)]

G = [A'*A C'; C zeros(k+1,k+1)]
h = [A'*b; d]
@assert rank(G) == size(G, 1) == size(G, 2) == n + k + 1
w = G \ h
x = w[1:n]
@show r = mu'*x
@show Rid = x'*sigma*x
@show obj = r - lambda*Rid
```

11.1830. Estimating a matrix with known eigenvectors. This problem is about estimating a matrix $A \in \mathbf{R}^{n\times n}$. 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 $v_1, \ldots, v_n$ of A. (Its eigenvalues $\lambda_1, \ldots, \lambda_n$, 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 $v_1, \ldots, v_n$. (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 `A` is a matrix, then `A[:]` is a (column) vector consisting of all the entries of `A`, written out column by column. Therefore `norm(A[:])` gives the squareroot of the sum of the squares of entries of the matrix `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 `a` is an mn vector, then `reshape(a,m,n)` is an $m \times n$ matrix, with elements taken from `a` (column by column).

Solution.

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

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

where $\Lambda = \mathrm{diag}(\lambda_1, \ldots, \lambda_n)$ and $V = [v_1 \ \cdots \ v_n]$. Here we know V and V^{-1}; the eigenvalues $\lambda_1, \ldots, \lambda_n$ are the unknowns to be determined. We will use W to denote $W = V^{-1}$, and we'll express its rows as $w_1^\mathsf{T}, \ldots, w_n^\mathsf{T}$, *i.e.*,

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

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

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

where $R^{(i)} = v_i w_i^\mathsf{T}$. (This is the spectral decomposition.) Note that $\hat{A}$ is a linear function of the variables here, *i.e.*, $\lambda_1, \ldots, \lambda_n$. We will show that our problem is now a completely standard least-squares problem, with unknowns $\lambda_1, \ldots, \lambda_n$. Let's define $\mathrm{vec}(S)$, 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

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

where

$$F = \left[\begin{array}{ccc} \mathrm{vec}(R^{(1)}) & \cdots & \mathrm{vec}(R^{(n)}) \end{array} \right], \qquad g = \mathrm{vec}(A^{\mathrm{meas}}).$$

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

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

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

$$(F^\mathsf{T}F)_{ij} = \mathrm{vec}(R^{(i)})^\mathsf{T}\,\mathrm{vec}(R^{(j)}) = v_i^\mathsf{T}v_j w_i^\mathsf{T}w_j.$$

We also have

$$(F^\mathsf{T}g)_i = v_i^\mathsf{T}A^{\mathrm{meas}}w_i.$$

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 $F^\mathsf{T}F$ as $F^\mathsf{T}F = (V^\mathsf{T}V) \circ (WW^\mathsf{T})$. We can express $F^\mathsf{T}g$ as $F^\mathsf{T}g = \mathrm{diag}(V^\mathsf{T}A^{\mathrm{meas}}W^\mathsf{T})$, where diag() extracts the diagonal part of a matrix. This gives us the formula

$$\lambda = \Big((V^\mathsf{T}V) \circ (WW^\mathsf{T})\Big)^{-1}\,\mathrm{diag}(V^\mathsf{T}A^{\mathrm{meas}}W^\mathsf{T}).$$

(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 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 LaTeX 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 $J_{\min} = 0.0175$.

```
>> 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
```

15.2270. Eigenvalues and singular values of a symmetric matrix. Let $\lambda_1, \ldots, \lambda_n$ be the eigenvalues, and let $\sigma_1, \ldots, \sigma_n$ be the singular values of a matrix $A \in \mathbb{R}^{n \times n}$, which satisfies $A = A^{\mathsf{T}}$. (The singular values are based on the full SVD: If $\mathrm{rank}(A) < n$, then some of the singular values are zero.) You can assume the eigenvalues (and of course singular values) are sorted, *i.e.*, $\lambda_1 \geq \cdots \geq \lambda_n$ and $\sigma_1 \geq \cdots \geq \sigma_n$. How are the eigenvalues and singular values related?

Solution. Since A is symmetric it can be diagonalized using an orthogonal matrix Q as

$$A = Q \begin{bmatrix} \lambda_1 & & \\ & \ddots & \\ & & \lambda_n \end{bmatrix} Q^{\mathsf{T}}$$

where $\lambda_1, \ldots, \lambda_n$ are the eigenvalues of A. Suppose that the columns of Q are ordered such that $|\lambda_1| \geq |\lambda_2| \geq \ldots \geq |\lambda_n|$. Thus

$$A = Q \begin{bmatrix} \mathrm{sgn}\lambda_1 & & \\ & \ddots & \\ & & \mathrm{sgn}\lambda_n \end{bmatrix} \begin{bmatrix} |\lambda_1| & & \\ & \ddots & \\ & & |\lambda_n| \end{bmatrix} Q^{\mathsf{T}}$$

Now we define

$$U = Q \begin{bmatrix} \mathrm{sgn}\lambda_1 & & \\ & \ddots & \\ & & \mathrm{sgn}\lambda_n \end{bmatrix}, \quad \Sigma = \begin{bmatrix} |\lambda_1| & & \\ & \ddots & \\ & & |\lambda_n| \end{bmatrix}, \quad V = Q.$$

Clearly, U is a an orthogonal matrix because $UU^{\mathsf{T}} = QQ^{\mathsf{T}} = I$. Now $A = U\Sigma V^{\mathsf{T}}$is a SVD of A, and we conclude that $\sigma_i = |\lambda_i|$.

15.2290. Matrix gain compared with entry size. A matrix can have all entries large and yet have small gain in some directions, that is, it can have a small $\sigma_{\min}$. For example,

$$A = \begin{bmatrix} 10^6 & 10^6 \\ 10^6 & 10^6 \end{bmatrix}$$

has "large" entries while $\|A[1 \ -1]^{\mathsf{T}}\| = 0$. Can a matrix have all entries small and yet have a large gain in some direction, that is, a large $\sigma_{\max}$? Suppose, for example, that $|A_{ij}| \leq \epsilon$ for $1 \leq i, j \leq n$. What can you say about $\sigma_{\max}(A)$?

Solution. The answer is no. If a matrix has all its entries small, then it cannot have a large gain in some direction. The precise statement is: if $|A_{ij}| \leq \epsilon$ for all i, j then we have $\sigma_{\max}(A) \leq \epsilon n$. We can give a direct proof of this fact. Let $x \in \mathbb{R}^n$, and define $y = Ax$. y_i, the ith component of y, is the product of the ith row of A and x. By the Cauchy-Schwarz inequality, $|y_i| \leq \|a_i\|\|x\|$, where a_i is the ith row of A. Since every entry in a_i is less than ϵ, we have $\|a_i\| \leq \sqrt{n}\epsilon$. Therefore, every entry of y has absolute value less than or equal to $\sqrt{n}\epsilon\|x\|$. It follows that $\|y\| \leq \sqrt{n}\sqrt{n}\epsilon\|x\| = n\epsilon\|x\|$. All of this shows that for any x, $\|Ax\| \leq n\epsilon\|x\|$, and it follows immediately that $\sigma_{\max}(A) \leq \epsilon n$. In fact, it turns out that this bound cannot be improved. Let $\mathbf{1}$ denote the vector in $\mathbb{R}^n$ with all entries one. Let $A = \epsilon\mathbf{1}\mathbf{1}^{\mathsf{T}}$, which is a matrix with all its entries equal to ϵ. $A^{\mathsf{T}}A = \epsilon^2(\mathbf{1}^{\mathsf{T}}\mathbf{1})\mathbf{1}\mathbf{1}^{\mathsf{T}}$, and so has eigenvalues $\epsilon^2 n^2$ and $n-1$ zeros. Therefore, $\sigma_{\max}(A) = n\epsilon$.

15.2350. Two representations of an ellipsoid. In the lectures, we saw two different ways of representing an ellipsoid, centered at 0, with non-zero volume. The first uses a quadratic form:

$$\mathcal{E}_1 = \left\{ x \,\middle|\, x^{\mathsf{T}}Sx \leq 1 \right\},$$

with $S^{\mathsf{T}} = S > 0$. The second is as the image of a unit ball under a linear mapping:

$$\mathcal{E}_2 = \{ y \,|\, y = Ax, \|x\| \leq 1 \},$$

with $\det(A) \neq 0$.

a) Given S, explain how to find an A so that $\mathcal{E}_1 = \mathcal{E}_2$.

b) Given A, explain how to find an S so that $\mathcal{E}_1 = \mathcal{E}_2$.

c) What about uniqueness? Given S, explain how to find *all* A that yield $\mathcal{E}_1 = \mathcal{E}_2$. Given A, explain how to find *all* S that yield $\mathcal{E}_1 = \mathcal{E}_2$.

Solution. First we will show that

$$\mathcal{E}_2 = \left\{ y \mid y^\mathsf{T}(AA^\mathsf{T})^{-1}y \leq 1 \right\} \tag{2}$$

$$\begin{aligned}
\mathcal{E}_2 &= \{y \mid y = Ax, \|x\| \leq 1\} \\
&= \left\{y \mid A^{-1}y = x, \|x\| \leq 1\right\} \text{ since } A \text{ is invertible square matrix} \\
&= \left\{y \mid \|A^{-1}y\| \leq 1\right\} \\
&= \left\{y \mid y^\mathsf{T}A^{-T}A^{-1}y \leq 1\right\} = \left\{y \mid y^\mathsf{T}(AA^\mathsf{T})^{-1}y \leq 1\right\}
\end{aligned}$$

Since S is symmetric positive definite, the eigenvalues of S are all positive and we can choose n orthonormal eigenvectors. So $S = Q\Lambda Q^\mathsf{T}$ where $\Lambda = \text{diag}(\lambda_1, \ldots, \lambda_n) > 0$ and Q is orthogonal. Let $\Lambda^{\frac{1}{2}} = \text{diag}(\sqrt{\lambda_1}, \ldots, \sqrt{\lambda_n})$. If we let $A = Q(\Lambda^{\frac{1}{2}})^{-1} = Q\Lambda^{-\frac{1}{2}}$,

$$\begin{aligned}
(AA^\mathsf{T})^{-1} &= (Q\Lambda^{-\frac{1}{2}}\Lambda^{-\frac{1}{2}}Q^\mathsf{T})^{-1} \\
&= (Q\Lambda^{-1}Q^\mathsf{T})^{-1} = Q\Lambda Q^\mathsf{T} \\
&= S
\end{aligned}$$

Therefore, by (2) $A = Q\Lambda^{-\frac{1}{2}}$ yields $\mathcal{E}_1 = \mathcal{E}_2$. By (2), $S = (AA^\mathsf{T})^{-1}$ yields $\mathcal{E}_1 = \mathcal{E}_2$. *Uniqueness:* We show that

$$\mathcal{E}_S = \mathcal{E}_T \Leftrightarrow S = T \tag{3}$$

where $\mathcal{E}_S = \{x \mid x^\mathsf{T}Sx \leq 1\}$, $\mathcal{E}_T = \{x \mid x^\mathsf{T}Tx \leq 1\}$, $S^\mathsf{T} = S > 0$ and $T^\mathsf{T} = T > 0$. It is clear that if $S = T$, then $\mathcal{E}_S = \mathcal{E}_T$. Now we show that $\mathcal{E}_S = \mathcal{E}_T \Rightarrow x^\mathsf{T}Sx = x^\mathsf{T}Tx,\ \forall x \in \mathbb{R}^n$. Without loss of generality let's assume $\exists x_0 \in \mathbb{R}^n$ such that $x_0^\mathsf{T}Sx_0 > x_0^\mathsf{T}Tx_0 = \alpha \neq 0$. If we let $x_1 = x_0/\sqrt{\alpha}$, then $x_1^\mathsf{T}Tx_1 = 1$, but $x_1^\mathsf{T}Sx_1 > 1$, thus $x_1 \in \mathcal{E}_T$ but $x_1 \notin \mathcal{E}_S$, and therefore $\mathcal{E}_S \neq \mathcal{E}_T$. Finally, $\mathcal{E}_S = \mathcal{E}_T \Rightarrow x^\mathsf{T}Sx = x^\mathsf{T}Tx,\ \forall x \in \mathbb{R}^n \Rightarrow S = T$ by the uniqueness of the symmetric part in a quadratic form. Hence S is unique. *Given S, find all A that yield $\mathcal{E}_1 = \mathcal{E}_2$.*

The answer is

$$A = Q\Lambda^{-\frac{1}{2}}V^\mathsf{T}$$

where $V \in \mathbb{R}^{n\times n}$ is any orthogonal matrix and

$$S^\mathsf{T} = S = Q\Lambda Q^\mathsf{T} > 0$$

where Q is orthogonal and $\Lambda = \text{diag}(\lambda_1, \ldots, \lambda_n) > 0$. Let the singular value decomposition of A be

$$A = U\Sigma V^\mathsf{T}$$

where $U, V \in \mathbb{R}^{n\times n}$ are orthogonal and $\Sigma = \mathrm{diag}(\sigma_1, \ldots, \sigma_n) > 0$ (since $\det(A) \neq 0$.) By (2),

$$\begin{aligned}\mathcal{E}_2 &= \left\{ y \mid y^\mathsf{T}(AA^\mathsf{T})^{-1}y \leq 1 \right\} \\ &= \left\{ y \mid y^\mathsf{T}(U\Sigma^2U^\mathsf{T})^{-1}y \leq 1 \right\} \\ &= \left\{ y \mid y^\mathsf{T}U\Sigma^{-2}U^\mathsf{T}y \leq 1 \right\}\end{aligned}$$

Thus, if $\mathcal{E}_1 = \mathcal{E}_2$, then $S = U\Sigma^{-2}U^\mathsf{T}$ by (3). Therefore $U = Q$ and $\Sigma = \Lambda^{-\frac{1}{2}}$, and V can be any orthogonal matrix. You can also see why A's are different only by right-side multiplication by an orthogonal matrix by the following argument. By (3) and (2), $AA^\mathsf{T} = S^{-1}$. Let

$$A = \begin{bmatrix} \tilde{a}_1 & \tilde{a}_2 & \ldots & \tilde{a}_n \end{bmatrix}^\mathsf{T}$$

Then we have,

$$\begin{aligned}\|\tilde{a}_i\|^2 &= (S^{-1})_{ii}, \\ \tilde{a}_i^\mathsf{T}\tilde{a}_j &= (S^{-1})_{ij},\end{aligned}$$

and

$$\cos\theta_{ij} = \frac{(S^{-1})_{ij}}{\sqrt{(S^{-1})_{ii}(S^{-1})_{jj}}}.$$

This means that the row vectors of any A satisfying $AA^\mathsf{T} = S^{-1}$ have the same length and the same angle between any two of them. So the rows of A can vary only by the application of an identical rotation or reflection to all of them. These are the transformations preserving length and angle, and correspond to orthogonal matrices. Since we are considering row vectors, the orthogonal matrix should be multiplied on the right.