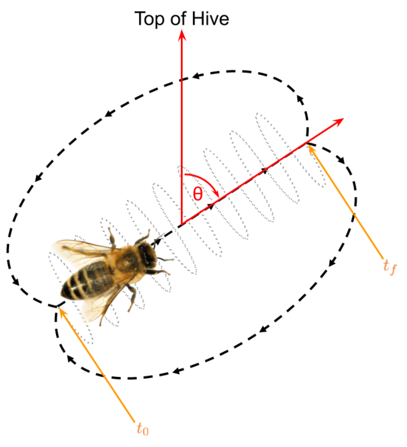
EE263 Exam Solutions, November 2023

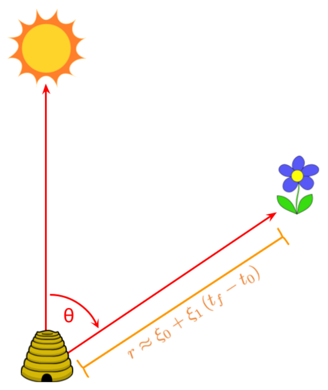
1. **Real-time interpretation of the honeybee waggle dance.** Upon discovering a food source that is farther than 100 meters from its resident hive, a forager honeybee will begin a cyclic "waggle" dance that follows the following form: (see figure 1a)

    1. The forager bee walks in one direction along a linear path, "waggling" its abdomen from side to side as it progresses along this path. (This portion of the dance is referred to in the literature as the *run* or *waggle run*.)
    2. The forager then turns either to the left or to the right, circling around to return to the start of the waggle run. (This portion of the dance is referred to as the *return*.)
    3. The forager repeats the waggle run with the next return being to the opposite side.

    This dance is repeated anywhere from 1 to ≥ 100 times, alternating between a right and left return after each run. Surrounding forager bees hug close to the dancing forager bee in order to feel out the pattern of the bee's dance.



(a) The run angle θ, the run starting time t_0, and the run finishing time t_f of a honeybee waggle dance.



(b) The angle to the food source θ with respect to the sun and the distance to the food source, r.

Figure 1: The angle to the food source with respect to the sun, with the hive at the vertex, is equivalent to the angle, θ, of the waggle run; the distance to the food source, r, is a linear function of the run duration, $\tau = t_f - t_0$.

This dance encodes geographical information about the location and quality of an identified food source (see Figure 1b). When the waggle dance is performed on a vertical surface (e.g., inside a vertically situated hive), the angle θ from the top (upwards) of the hive to the linear path of the waggle run is equal to the angle from the sun to the located food source with the hive being the vertex of this angle; the duration $\tau = t_f - t_0$ of the waggle has a linear relationship with the distance r from the hive to the located food source.

Thus, when a forager bee locates a food source and returns to the hive, its waggle dance signals to the honeybees that follow its dance two pieces of information:

- $\theta(t)$ is the angle to the located food source with respect to the location of the sun at time t; and
- r is the distance to the food source in kilometers.

Furthermore, r can be approximately expressed as

$$r \approx \xi_0 + \xi_1 \tau, \tag{1}$$

where ξ_0 and ξ_1 are fit coefficients, and $\tau = t_f - t_0$ is the duration of the run in the forager bee's waggle dance.

a) We will start by first fitting the coefficients ξ_0 and ξ_1 in the linear approximation of r. In the data file `waggle_dance_data.json`, you will use the vectors `tau` and `r` to fit the coefficients. The elements in `tau` are measurements that represent durations (in seconds) of waggle runs from m different dances, and the elements in `r` are the corresponding distances (in kilometers) to the different food sources. Choose ξ_0 and ξ_1 to minimize

$$\sum_{i=1}^{m} (\xi_0 + \xi_1 \tau_i - r_i)^2$$

b) Next, we will use least squares to find the equation for the line $y = \beta_0 + \beta_1 x$ for a particular waggle dance. In `waggle_dance_data.json`, you will see additional vectors:

- `x`, which represents the x-coordinates $(x(1), \ldots, x(n))$ at each time step $t = 1, \ldots, n$ of a honeybee that is performing a waggle dance;
- `y`, which represents the y-coordinates $(y(1), \ldots, x(n))$ at each time step $t = 1, \ldots, n$ of the honeybee's waggle dance;
- `w`, which is a binary variable, denoting whether at time step t the honeybee is on the *waggle run* ($w(t) = 1$) or *return* ($w(t) = 0$). In this particular dance, you will see a *waggle run*, followed by a *return* to the left, followed by a second *waggle run*, and finally, a *return* to the right. Thus, there are only 2 waggle runs in this particular dance.

To find the line along which the honeybee is waggling, you will need to use least squares *only on those points where the bee is on a waggle run.* Specifically, choose β_0 and β_1 to minimize

$$\sum_{t|w(t)=1} (\beta_0 + \beta_1 x(t) - y(t))^2$$

Report your fitted equation $\hat{y} = \beta_0 + \beta_1 x$. Plot the fitted line, and all of the points $(x(t), y(t)), t = 1, \ldots, n$, with points on the *waggle run* in one color, and points on the *return* in another color.

c) Using the data of our particular waggle dance, compute the average duration τ of this particular dance's waggle runs. To do this, you will: measure the length (in frames) of both waggle runs present in the dance; take the average; and then convert the length from frames to seconds. The framerate of the time steps is 30 frames per second; that is, 1 time step forward is equivalent to 1/30 of a second.

d) Now, using your results from parts a), b) and c) (and a little bit of trigonometry), provide an estimate of the angle $\hat{\theta}$ between the sun and the food source described by this honeybee's waggle dance (this will be the same angle between the vector along which the bee is waggling and the top of the hive, or the positive y-axis), as well as the approximate distance (in kilometers) to the food source.

Hint: Don't forget to take into account which way the honeybee is moving along the line you computed in part c): the angle you are aiming to compute is between the vector that represents the forward direction of the bee's movement along the line and the positive y-axis. If you don't take that into account, your estimate will be off by 180 degrees or π radians.

e) Using the same data in part b), use *recursive* least squares to create a *real-time* estimate of the line along which the honeybee is waggling at each time step; that is, for each time step t, you should have an equation $y = \beta_{0,t} + \beta_{1,t} x$ that is fit to all of the points seen so far. *As before, you should only be fitting to points that are on the waggle run. This means that your estimate of the line will not be updated when the bee is on the return.*

At each time step, use trigonometry to provide a *real-time* estimate of the angle between the sun and the located food source. Plot your real-time estimated angle $\hat{\theta}_t$ against the time step t. In your plot, include a horizontal line that represents the estimated angle $\hat{\theta}$ you calculated in part c). Do the recursively estimated angles $\hat{\theta}_t$ converge to your estimate $\hat{\theta}$ in part d)?

Solution.

a) Let $r \in \mathbb{R}^m$ be the vector of distances given to us in `r`; let $T \in \mathbb{R}^{m \times 2}$ be the data matrix given by

$$T = \begin{bmatrix} 1 & \tau_1 \\ 1 & \tau_2 \\ \vdots & \vdots \\ 1 & \tau_m \end{bmatrix}; \tag{2}$$

and let $\xi = [\xi_0, \xi_1]^T \in \mathbb{R}^2$ be our parameter vector.

To fit the linear approximation, we are solving the least squares problem given by

$$\underset{\xi}{\text{minimize}} \; \|r - T\xi\|_2^2, \tag{3}$$

the solution to which is $\hat{\xi} = (T^T T)^{-1} T^T r$, provided T is full rank.

Using Julia, we find our fitted coefficients are

$$\hat{\xi}_0 = -0.5677, \; \hat{\xi}_1 = 1.2529. \tag{4}$$

The Julia code to accomplish this is shown below:

```
include("../code/readclassjson.jl")

data = readclassjson("../data/waggle_dance_data.json")
m = data["m"]
r = data["r"]
tau = data["tau"]

T = reduce(hcat, [ones(m), tau])  # data matrix
xi = T \ r                        # Least Squares Solution

display(xi)
```

b) This problem is similar to part a), except we first need to filter out all of the points $(x(t), y(t))$ which are *not* on the waggle run.

Let $\tilde{x}_1, \ldots, \tilde{x}_k$ be all of the points $x(t)$ such that $w(t) = 1$. Similarly, let $\tilde{y}_1, \ldots, \tilde{y}_k$ be all of the points $y(t)$ such that $w(t) = 1$. Note that order is preserved in these new notations; that is, for $\tilde{x}_i, \tilde{x}_j$, if $i < j$, then $\tilde{x}_i$ occurred before $\tilde{x}_j$ in the original time series.

Then, let $\tilde{y} \in \mathbb{R}^k$ be the vector composed of the points $\tilde{y}_i, i = 1, \ldots, k$. Let $\tilde{X} \in \mathbb{R}^{k \times 2}$ be our data matrix, where the first column of $\tilde{X}$ is the all-ones vector and the second column of $\tilde{X}$ are our $\tilde{x}_i, i = 1, \ldots, k$ values. Finally, let $\beta = [\beta_0, \beta_1]^T \in \mathbb{R}^2$ be our parameter vector.

Then, we aim to solve the least squares problem

$$\underset{\beta}{\text{minimize}}\ \|\tilde{y} - \tilde{X}\theta\|_2^2, \tag{5}$$

the solution to which is given by $\hat{\beta} = (\tilde{X}^T\tilde{X})^{-1}\tilde{X}^T\tilde{y}$.

Using Julia, we find our fitted coefficients are

$$\beta_0 = 104.4739,\ \beta_1 = -0.0197. \tag{6}$$

We plot the fitted line (along with an arrow demonstrating the honeybee's direction of motion) and all of the points in the waggle dance below:

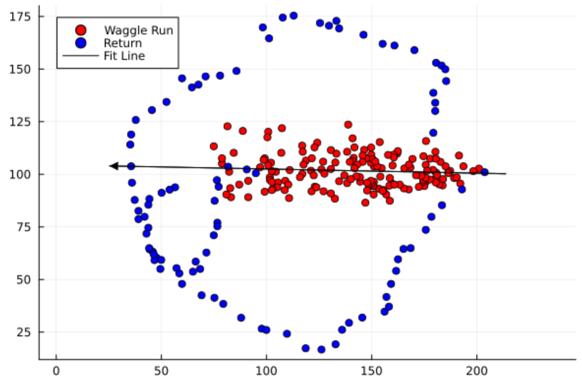


Figure 2: The fitted line $\hat{y} = \beta_0 + \beta_1 x$ along with the points of the waggle dance. Red points signify the waggle run; blue points signify the return. The direction of the dance was determined by looking at the progression of the points over time.

The Julia code is shown below:

```
using Plots

include("../code/readclassjson.jl")

data = readclassjson("../data/waggle_dance_data.json")
n = data["n"]
t = data["t"]
x = data["x"]
y = data["y"]
w = data["w"]

k = sum(w)                # number of points on waggle run
x_run = x[w .== 1]        # x-coords on waggle run
```

```
y_run = y[w .== 1]     # y-coords on waggle run
x_return = x[w .== 0]  # x-coords on return
y_return = y[w .== 0]  # y-coords on return

X = reduce(hcat, [ones(k), x_run])  # data matrix
beta = X \ y_run                    # Least Squares solution
display(beta)

# First Waggle Run is from t = 1 to t = 65 (see Part C)
direction = (x[65] - x[1]) > 0 ? :head : :tail

# Plot the fitted line and points of the dance
i = [minimum(x) - 10, maximum(x) + 10]
line = (beta[2] .* i) .+ beta[1]
scatter(x_run, y_run, mc=:red, aspect_ratio=:equal, label="Waggle Run")
scatter!(x_return, y_return, mc=:blue, label="Return")
plot!(i, line, arrow=(:closed, direction), color=:black, label="Fit Line")
savefig("../graphics/fitted_waggle_run.png")
```

c) The two waggle runs start at time steps $t = 1$ and $t = 110$, respectively, and end at time steps $t = 65$ and $t = 203$. Therefore, the duration of the two dances, in frames, are 64 frames and 93 frames. Therefore, on average, the honeybee spends 78.5 frames on the waggle run. Converting this to seconds (using the framerate of 30 FPS), this means the honeybee spends, on average, 2.6167 seconds on the waggle run.

The Julia code is shown below:

```
include("../code/readclassjson.jl")

data = readclassjson("../data/waggle_dance_data.json")
w = data["w"]
FPS = 30  # Framerate = 30 FPSs

start_frames = [1, findall(>(0), diff(w))[1] + 1]  # Frames for start of dance
end_frames = findall(<(0), diff(w))                # Frames for end of dance

avg_duration_frames = sum(end_frames - start_frames) / 2  # Avg. of 2 dances
avg_duration_secs = avg_duration_frames / FPS             # Convert to seconds
display(avg_duration_secs)
```

Note: Another possible answer is that the duration of the dances are 65 frames and 94 frames, respectively, in which case the answer will be 2.65 seconds on the waggle run. (Because this isn't a signals processing course, and we didn't describe how the data was recorded, either of these answers is acceptable.)

d) The line that we fit to the waggle run in part b) has a slope of $\beta_1 = -0.0197$. Therefore, the angle between the positive x-axis and this line will be $\tan^{-1}(\beta_1) = -0.0197$ rads, or -1.1286 degrees.

However, the honeybee was walking from right to left: this means that we need to add π rads or 180 degrees, giving us 3.1219 rads, or 178.8714 degrees.

Finally, this angle that we currently have is counterclockwise from the positive x-axis. We need it to be measured from the Sun, which is at the positive y-axis. Therefore, we subtract $\frac{\pi}{2}$ rads or 90 degrees, to get our final estimate of the angle $\hat{\theta}$ from the Sun to the located Food Source:

$$\hat{\theta} = \tan^{-1}(\beta_1) + \frac{\pi}{2} = 1.5511 \text{ rads}, \tag{7}$$

or 88.8702 degrees counterclockwise from the Sun.

Using our results from part a) and part c), we then compute the approximate distance to the food source from the hive to be

$$r \approx \hat{\xi}_0 + \hat{\xi}_1 \tau = -0.5677 + 1.2529 \cdot 2.6167 = 2.7106 \text{ kilometers}. \tag{8}$$

Note: In line with the alternate possible answer from part c), another acceptable answer for the approximate distance is $r = 2.7524$ kilometers.

The Julia code is shown below:

```
xi = [-0.5676571122194828, 1.2528547203255607]       # Least Squares Solution fro
beta = [ 104.47393303570679, -0.019721609390038978]  # Least Squares Solution fro
tau = 2.6166666666666667                             # Avg. Duration from Part C

# Computing Angle
theta_rads = atan(beta[2]) + (pi/2)
theta_degs = theta_rads * (180/pi)
print("Angle from the Sun: \n")
print(theta_rads, " radians counterclockwise, or\n")
print(theta_degs, " degrees counterclockwise\n\n")

# Computing Distance
r = xi[1] + xi[2] * tau
print("Distance from the Hive:\n", r, " kilometers")
```

e) From the recursive estimation slides (lecture 14) and our earlier results, we can formulate the following algorithm to provide a real-time estimate $\hat{\theta}_t$ of the angle between the sun and the located food source:

```
Given: $(x_i, y_i, w_i), i = 1, \ldots, n$

$P \leftarrow 0 \in \mathbb{R}^{n \times n}$
$q \leftarrow 0 \in \mathbb{R}^n$
for $i = 1, \ldots, n$ :
    if $w_i = 1$: (i.e., $(x_i, y_i)$ is on the waggle run)
        $\tilde{x}_i \leftarrow \begin{bmatrix} 1 \\ x_i \end{bmatrix}$
        $P \leftarrow P + \tilde{x}_i \tilde{x}_i^T$
        $q \leftarrow q + y_i \tilde{x}_i$
        if $P$ is invertible:
            $\beta_i \leftarrow P^{-1} q$
            $\hat{\theta}_i \leftarrow \tan^{-1}(\beta_{i,1}) + \frac{\pi}{2}$
```

Below is our plot, demonstrating that the recursive estimates $\hat{\theta}_t$ *do converge* to the full least squares estimate $\hat{\theta}$ from part d).

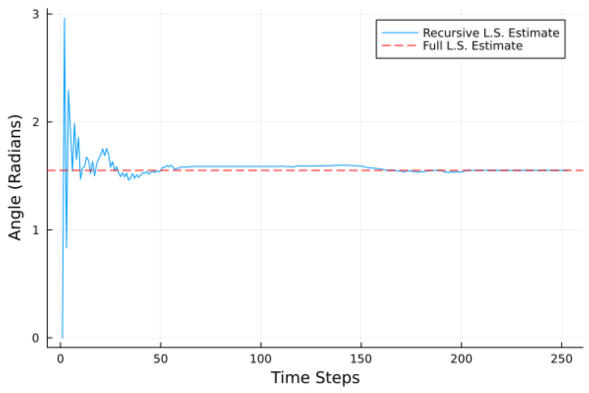


Figure 3: The recursive least squares estimate $\hat{\theta}_t$ plotted over time steps t.

The Julia code to accomplish this is shown below:

```
using LinearAlgebra, Plots

include("../code/readclassjson.jl")
```

```
data = readclassjson("../data/waggle_dance_data.json")
n = data["n"]
t = data["t"]
x = data["x"]
y = data["y"]
w = data["w"]
full_ls_theta = 1.5510772736616003

P = zeros(2, 2)
q = zeros(2)
theta = 0.0
angles = Float64[]
for i in t
    if w[i] == 1
        x_tilde = [1, x[i]]
        global P = P + (x_tilde * x_tilde')
        global q = q + (y[i] * x_tilde)
        if rank(P) == 2
            beta = P \ q
            global theta = atan(beta[2]) + (pi/2)
        end
    end
    push!(angles, theta)
end

plot(t, angles, label="Recursive L.S. Estimate")
hline!([full_ls_theta], ls=:dash, lc=:red, label="Full L.S. Estimate")
xlabel!("Time Steps")
ylabel!("Angle (Radians)")
savefig("../graphics/recursive_convergence.png")
```

Problem & Data Courtesy of: G. Holt, P. Murray, D. Grimsman and S. Warnick, "Developing Ecological Sensors for Real-Time Interpretation of Honeybee Communication," 2022 IEEE Conference on Control Technology and Applications (CCTA), Trieste, Italy, 2022, pp. 69-75, doi: 10.1109/CCTA49430.2022.9966064.

2. **There and back again.** A unit mass moves in 1-dimension. It starts at position 0, with velocity 0, at time $t = 0$. We are going to move it from its starting position to position 1, and then move it back to the origin. Ideally we would like it to reach position 1 at time T_1 and reach the origin again at time T_2.

   The force acting on it during the interval $j-1 \leq t < j$ is given by x_j for $j = 1, \ldots, T_2$. Let y_1 be the position of the mass at time T_1 and y_2 be the position of the mass at time T_2. We will use parameters $T_1 = 10$ and $T_2 = 20$.

a) Let

$$y = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}$$

Find the matrix A such that $y = Ax$.

b) Find the minimum norm sequence of forces x which moves the mass so that at time T_1 its position is 1, and at time T_2 its position is 0 again. Plot x (that is plot x_j versus time j).

c) Using the optimal sequence of forces you found in the previous part, compute the position and velocity of the mass as a function of time. Plot these.

d) Now we would like to trade-off the two objectives

$$J_1 = \sum_{j=1}^{T_2} \|x_j\|^2 \qquad J_2 = (y_1 - 1)^2 + y_2^2$$

Compute the optimal trade-off curve, and plot J_1 against J_2.

e) Find the sequence of forces x which has the smallest norm and achieves a position error J_2 of less than or equal to 0.2. Plot x, and the corresponding position and velocity of the mass as a function of time.

Solution.

a) We will use the following equations of motion:

$$v_n = v_{n-1} + a_n t \qquad p_n = p_{n-1} + v_{n-1} t + \tfrac{1}{2} a_n t^2$$

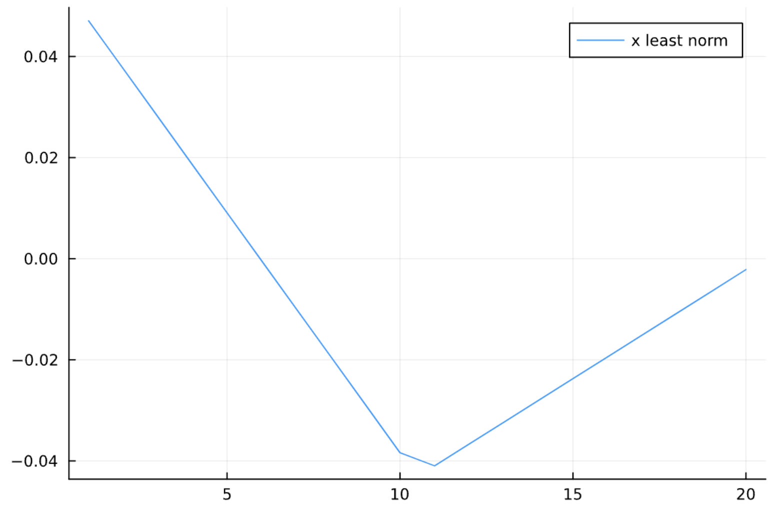
Where v is velocity, p is position, a is acceleration and t is time. Combining equations 1 and 2, using $t = 1$ and $a_n = x_n$ we obtain the following equation for p_n and v_n:

$$v_n = \sum_{i=1}^{n} x_i \qquad p_n = \sum_{i=1}^{n} ((n-1) + \tfrac{1}{2}) x_i$$

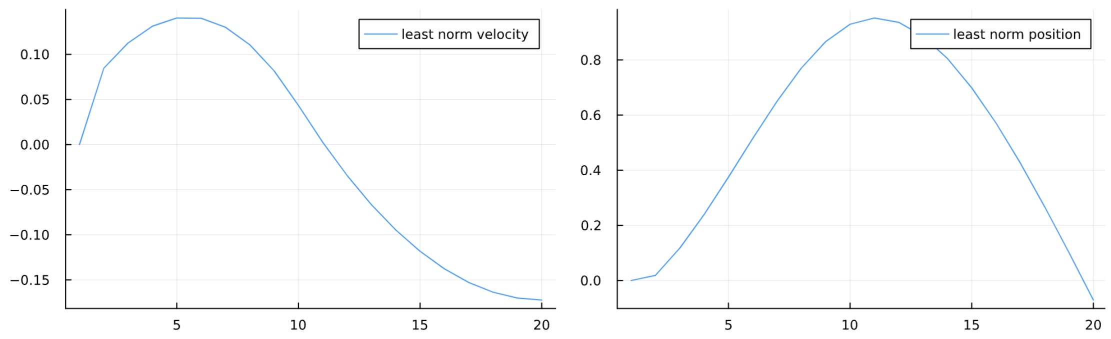
Since $y_1 = p_{10}$ and $y_2 = p_{20}$,

$$A = \begin{bmatrix} 10 - \frac{1}{2} & 10 - \frac{3}{2} & \cdots & \frac{1}{2} & 0 & \cdots & 0 \\ 20 - \frac{1}{2} & 20 - \frac{3}{2} & \cdots & \cdots & \cdots & \frac{3}{2} & \frac{1}{2} \end{bmatrix}$$

b) The min norm solution is given by $x_{\text{leastnorm}} = A^\mathsf{T}(AA^\mathsf{T})^{-1}y$. It is plotted below.



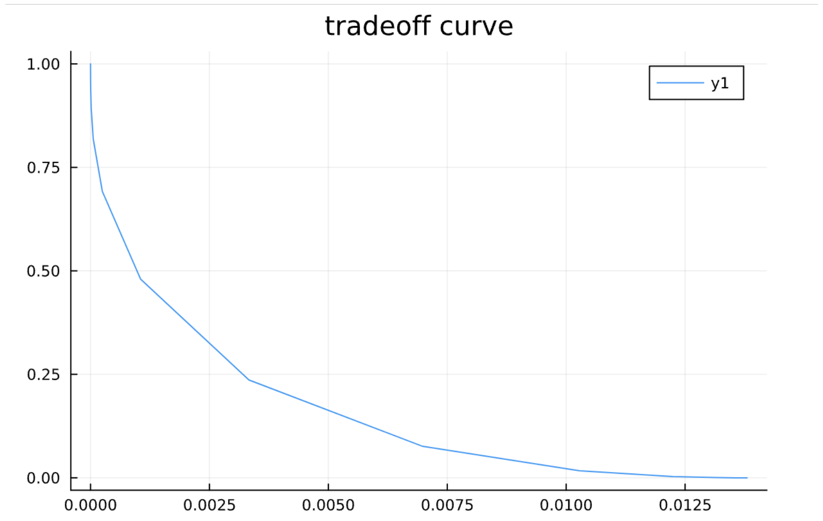
c) Using the aforementioned formulas for p_n and v_n we will complete and plot the values using the least norm solution we found in part b).



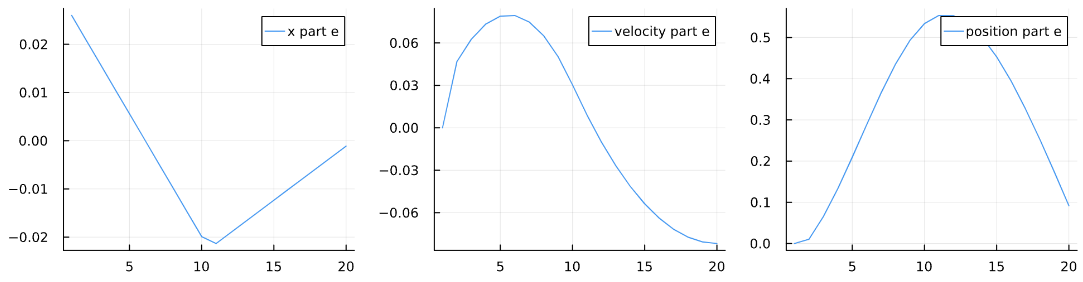
d) We will re-write the two objective functions as:

$$J_1 = \sum_{j=1}^{T_2} \|x_j\|^2 = \|x\|^2 \qquad J_2 = (y_1 - 1)^2 + y_2^2 = \left\| \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} - \begin{bmatrix} 1 \\ 0 \end{bmatrix} \right\|^2 = \left\| Ax - \begin{bmatrix} 1 \\ 0 \end{bmatrix} \right\|^2$$

This is a multi-objective least squares problem. We must minimize $J_1 + \mu J_2$. The tradeoff curve is displayed below.



e) Since there is a tradeoff between J_1 and J_2, we will set J_2 to be as close to 0.2 as possible since this will minimize J_1. Choosing μ to be around 58.2 gives us $J_1 = 0.00391$ and $J_2 = 0.1996$. We use this value of μ to minimize $J_1 + \mu J_2$ and obtain x. The plots of x, position and velocity are below.

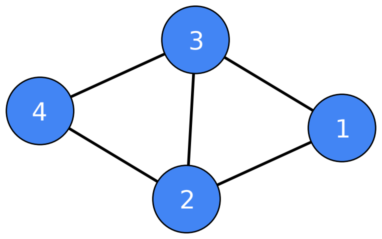


The complete Julia code for all the parts is shown below:

```
#part b

using LinearAlgebra, Plots
y = [1, 0]
A = [10-(1/2) 10-(3/2) 10-(5/2) 10-(7/2) 10-(9/2) 10-(11/2) 10-(13/2) 10-(15/2)
    10-(17/2) 10-(19/2) 0 0 0 0 0 0 0 0 0 0; 20-(1/2) 20-(3/2) 20-(5/2) 20-(7/2)
    20-(9/2) 20-(11/2) 20-(13/2) 20-(15/2) 20-(17/2) 20-(19/2) 20-(21/2) 20-(23/2
    20-(25/2) 20-(27/2) 20-(29/2) 20-(31/2) 20-(33/2) 20-(35/2) 20-(37/2) 20-(39/
x_ln = A'*pinv(A*A')*y
plot(x_ln, label="x least norm")

#part c

v = zeros(20)
```

```
p = zeros(20)
for i in range(2,length(x_ln))
    v[i] = sum(x_ln[1:i])
end
for i in range(2, length(x_ln))
    p[i] = p[i-1] + v[i-1] + x_ln[i]/2
end
p1 = plot(v, label="least norm velocity")
p2 = plot(p, label ="least norm position")
plot(p1, p2, layout=(1, 2), size=(1000, 300))

#part d

J1 = zeros(50)
J2 = zeros(50)
mu = 10 .^(range(-10,stop=10,length=50));

for i in 1:length(mu)
    x = pinv(A'*A+mu[i]*I(20))*A'*y
    J1[i] = (norm(x))^2
    J2[i] = (norm(A*x - y))^2
end
plot(J2,J1, title = "tradeoff curve")

#part e

x = pinv(A'*A+58.2*I(20))*A'*y
J1_e = (norm(x))^2
J2_e = (norm(A*x - y))^2
v_e = zeros(20)
p_e = zeros(20)
for i in range(2,length(x))
    v_e[i] = sum(x[1:i])
end
for i in range(2, length(x))
    p_e[i] = p_e[i-1] + v_e[i-1] + x[i]/2
end
p1 = plot(x, label="x part e")
p2 = plot(v_e, label="velocity part e")
p3 = plot(p_e, label ="position part e")
plot(p1, p2, p3, layout=(1, 3), size=(980, 250))
```

3. Graph layout by quadratic minimization. The objective of this question is to develop a semi-automatic method of drawing graphs. By a graph, we mean a collection of vertices linked by edges. For example



Consider a graph with n vertices and m edges. Each edge $k \in \{1, \ldots, m\}$ has a source vertex $\operatorname{src}(k)$ and a destination vertex $\operatorname{dst}(k)$. In this question, the orientation of the edge is irrelevant; we can swap the source and destination without changing the graph.

A graph has an associated *incidence matrix* $Q \in \mathbb{R}^{n \times m}$, where

$$Q_{ik} = \begin{cases} 1 & \text{if } i = \operatorname{src}(k) \\ -1 & \text{if } i = \operatorname{dst}(k) \\ 0 & \text{otherwise} \end{cases}$$

For the example graph above, we have

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

For many applications we are only concerned with which vertices connect to which. However, in order to draw a graph, we need to specify in addition the coordinates of each vertex. In this question, we will solve an optimization problem to do this.

a) You are given the incidence matrix Q of a graph. You are also given a list of *fixed vertices* $\mathcal{V}_{\text{fixed}} \subset \{1, 2, \ldots, n\}$, and their coordinates $p_i \in \mathbb{R}^2$ for $i \in \mathcal{V}_{\text{fixed}}$. Your objective is to choose the position $x_i \in \mathbb{R}^2$ of the vertices i that are not fixed. To do this, you will solve

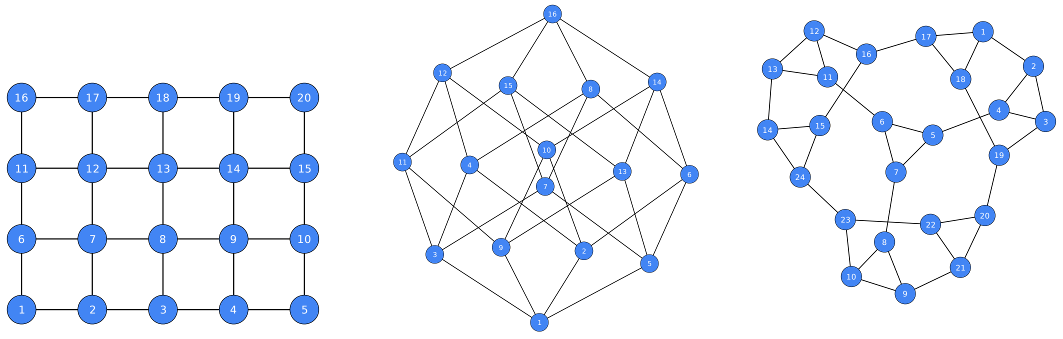
$$\begin{array}{ll} \text{minimize} & \displaystyle\sum_{k=1}^{m} \|x_{\operatorname{src}(k)} - x_{\operatorname{dst}(k)}\|^2 \\ \text{subject to} & x_i = p_i \quad \text{for } i \in \mathcal{V}_{\text{fixed}} \end{array}$$

The idea is that we would like to position the vertices so that vertices connected by an edge are close together, while maintaining the position of the fixed vertices.

Let $X \in \mathbb{R}^{n\times 2}$ be a matrix whose ith row is the transpose of the position x_i of node i. Express this optimization problem in terms of X. Be sure to define carefully any additional matrices that you need.

b) Give an algorithm for solving the optimization problem in part (a).

c) In the data file `graphs.json` you will find the incidence matrix Q for three different graphs. These graphs are drawn below, by hand. These are *not* the expected solutions.



For each graph you will also find f, a list of the fixed vertices, and P, the coordinates of the fixed vertices. Apply your algorithm to each of the graphs, and plot the resulting graphs. The file `plotgraph.jl` contains a helper file you may want to use to plot the graphs. It can be called using `plotgraph(Q, X)` where Q is the incidence matrix and X the matrix of vertex positions.

Solution. Here is the solution.

a) Let the number of fixed vertices be r, and let $f \in \mathbb{R}^r$ be a list of the indices of the fixed vertices. Let $P \in \mathbb{R}^{r\times 2}$ be a matrix for which row i is the position of the i'th fixed vertex. Define the matrix $C \in \mathbb{R}^{t\times n}$ by

$$C_{ij} = \begin{cases} 1 & \text{if } j = f_i \\ 0 & \text{otherwise} \end{cases}$$

Then we would like to solve

$$\begin{array}{ll} \text{minimize} & \|Q^\mathsf{T} X\|_F^2 \\ \text{subject to} & CX = P \end{array}$$

b) This problem can be written in terms of vector variables by considering X and P in terms of their columns. Let

$$X = \begin{bmatrix} \tilde{x}_1 & \tilde{x}_2 \end{bmatrix} \qquad P = \begin{bmatrix} \tilde{p}_1 & \tilde{p}_2 \end{bmatrix}$$

Then we have two separate problems, for $i = 1$ and $i = 2$, given by

$$\begin{array}{ll} \text{minimize} & \|Q^\mathsf{T}\tilde{x}_i\|_F^2 \\ \text{subject to} & C\tilde{x}_i = P \end{array}$$

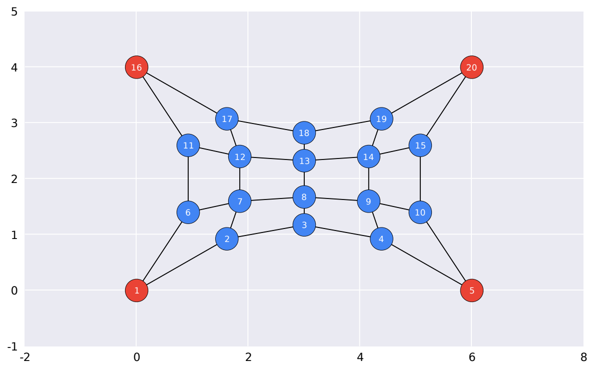
The solution to this optimization is given by

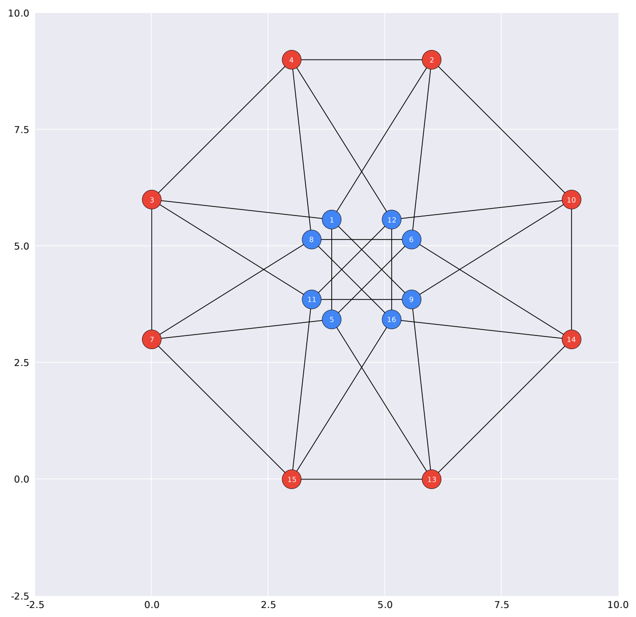
$$\tilde{x}_i = \begin{bmatrix} I & 0 \end{bmatrix} \begin{bmatrix} QQ^\mathsf{T} & C^\mathsf{T} \\ C & 0 \end{bmatrix}^{-1} \begin{bmatrix} 0 \\ \tilde{p}_i \end{bmatrix}$$

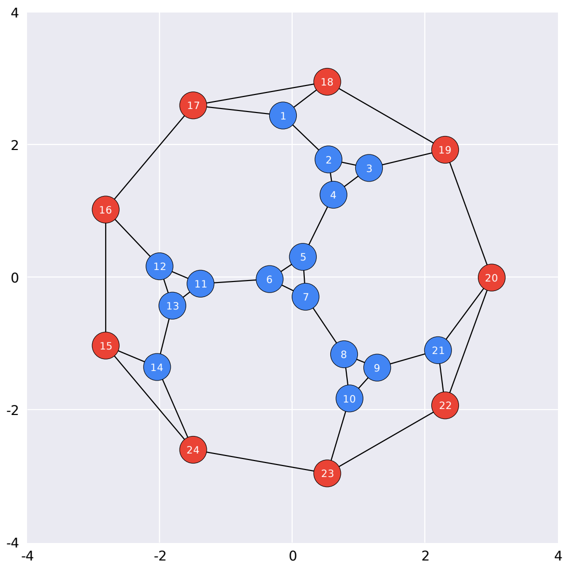
Since this gives each of the columns of X, we can write the solution for the entirety of X as

$$X = \begin{bmatrix} I & 0 \end{bmatrix} \begin{bmatrix} QQ^\mathsf{T} & C^\mathsf{T} \\ C & 0 \end{bmatrix}^{-1} \begin{bmatrix} 0 \\ P \end{bmatrix}$$

c) The resulting graphs are below.







4. Low rank factorization of matrices. We are given an $m \times n$ matrix A, and square invertible matrices U and V such that

$$A = U \begin{bmatrix} I_r & 0 \\ 0 & 0 \end{bmatrix} V \tag{9}$$

where I_r is the $r \times r$ identity matrix. Note that the zeros in the middle matrix must have the correct dimensions.

a) What is rank(A). Justify your answer.

b) Suppose that $y \in \text{range}(A)$, so with this particular y there is a solution to the equation $Ax = y$. Show that one such solution is

$$x = V^{-1} \begin{bmatrix} I_r & 0 \\ 0 & 0 \end{bmatrix} U^{-1} y$$

Again, you should be careful to ensure that the zeros have the correct dimensions.

c) Suppose instead that $y \notin \text{range}(A)$. Show that

$$U \begin{bmatrix} I_r & 0 \\ 0 & 0 \end{bmatrix} U^{-1} y \neq y$$

Note that the zeros in the matrix expression above need not be the same dimensions as in the previous parts.

d) Give, in terms of the columns of U, a basis for range(A). Justify your answer.

e) Suppose you are given the matrix A. Give a procedure, **using the QR factorization**, for computing matrices U and V satisfying equation (9).

f) Use your procedure to compute U and V for the matrix

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

In your solution, include a printout of Julia showing that equation (9) holds.

Solution.

a) Multiplication by invertible matrices does not change the rank, so

$$\text{rank}(A) = \text{rank}\, U^{-1} A V^{-1} = r$$

b) For convenience, let

$$H = \begin{bmatrix} I_r & 0 \\ 0 & 0 \end{bmatrix}$$

where $H \in \mathbb{R}^{m \times n}$, so that $A = UHV$. We need to show $Ax = y$, where $x = V^{-1}H^{\mathsf{T}}U^{-1}y$. (Notice that in this expression we must use H^{T}, since H is not square.) First, we have

$$Ax = UHH^{\mathsf{T}}U^{-1}y$$

Now we know also that $y \in \text{range}(A)$, so $y = Az$ for some vector z. That means

$$\begin{aligned} Ax &= UHH^{\mathsf{T}}U^{-1}y \\ &= UHH^{\mathsf{T}}U^{-1}UHVz \\ &= UHH^{\mathsf{T}}HVz \end{aligned}$$

Now we can use the fact that $HH^{\mathsf{T}}H = H$, and so

$$\begin{aligned} Ax &= UHVz \\ &= Az \\ &= y \end{aligned}$$

as desired.

c) For convenience, let $G = V^{-1}H^{\mathsf{T}}U^{-1}$. Then

$$\begin{aligned} AGA &= UHVV^{-1}V^{-1}H^{\mathsf{T}}U^{-1}UHV \\ &= UHV \\ &= A \end{aligned}$$

Now we have to show that if $y \in \text{range}(A)$ then

$$y = U \begin{bmatrix} I_r & 0 \\ 0 & 0 \end{bmatrix} U^{-1}y$$

If $y \in \text{range}(A)$ then there exists z such that $y = Az$, and so we have

$$\begin{aligned} y &= Az \\ &= AGAz \\ &= AGy \\ &= UHH^{\mathsf{T}}U^{-1}y \end{aligned}$$

as desired.

d) We have

$$\begin{aligned} \text{range}(A) &= \text{range}(UHV) \\ &= \text{range}(UH) \end{aligned}$$

since V is invertible. Now write

$$U = \begin{bmatrix} U_1 & U_2 \end{bmatrix}$$

so that U_1 consists of the first r columns of U. Then $UH = U_1$, and so range(A) is the span of the first r columns of U.

e) We use the full pivoted QR factorization of A to give

$$\begin{aligned} A &= Q \begin{bmatrix} R_{11} & R_{12} \\ 0 & 0 \end{bmatrix} P^\mathsf{T} \\ &= Q \begin{bmatrix} I_r & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} R_{11} & R_{12} \\ 0 & I \end{bmatrix} P^\mathsf{T} \end{aligned}$$

where we can change the lower right block of R to be the identity since it is multiplied by a zero block. Let $U = Q$ and

$$V = \begin{bmatrix} R_{11} & R_{12} \\ 0 & I \end{bmatrix} P^\mathsf{T}$$

Notice that V is invertible, since

$$V^{-1} = P \begin{bmatrix} R_{11}^{-1} & -R_{11}^{-1} R_{12} \\ 0 & I \end{bmatrix}$$

f) Some code that computes this is below.

```
using LinearAlgebra
include("qr.jl")
A = [  1  1  0  0  0  0  0 ;
      -1  0  1  1  0  0  0 ;
       0  0 -1  0  1  0  0 ;
       0 -1  0  0  0  1  0 ;
       0  0  0 -1  0 -1  1 ;
       0  0  0  0 -1  0 -1 ]
m,n = size(A)
r = 5
P,Q,R = pivotedqr(A)
X = [R[1:r,:] ; zeros(n-r,r) I(n-r)]
U = Q
V = X*P'
H = [I(r) zeros(r,n-r); zeros(m-r,n)]
# demonstration that this works
A - U*H*V
```

5. Left inverses.

a) Suppose $A \in \mathbb{R}^{m\times n}$ is left invertible, and $m > n$. Show that there are infinitely many left-inverses of A.

b) We would like to use a left-inverse C of A for sensing. Then we will have measurements $y = Ax$, and will use an estimate $\hat{x} = Cy$. If we write C in terms of its columns

$$C = \begin{bmatrix} v_1 & v_2 & \dots & v_m \end{bmatrix}$$

then we can write the estimate as

$$\hat{x} = v_1 y_1 + v_2 y_2 + \cdots + v_m y_m$$

In this case, the requirements that we have for C are that we must have $v_1 = v_2$, so that the measurements y_1 and y_2 are processed identically and their sensors may be interchanged. Given an algorithm for finding a left-inverse C with this property. Be sure to state any conditions on A necessary for there to be a solution.

c) Implement your algorithm of the previous part for the matrix

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}$$

How many solutions C are there?

d) The previous designer of the system used estimation matrix given by

$$\tilde{C} = \begin{bmatrix} -1 & -1 & 1 \\ 1 & 1 & -1 \end{bmatrix}$$

This matrix is not a left-inverse of A, and so it does not provide an unbiased estimate of x. However, it has other desirable properties, and so you would like to find an unbiased estimator C such that

$$\|C - \tilde{C}\|_F^2$$

is as small as possible. Here we use the Frobenius norm, which for a $p \times q$ matrix Z is defined by

$$\|Z\|_F^2 = \sum_{i=1}^{p}\sum_{j=1}^{q} Z_{ij}^2$$

Using the same A from part c), find such a matrix C.

e) We now consider a more general approach to finding left-inverses with specific properties. Suppose L is a left-inverse of A. Show that, for any matrix X, the matrix
$$C = L + X(I - AL)$$
is also a left-inverse of A.

f) In fact, the converse of the previous statement is true. Suppose L is a left inverse of A. Then if C is also a left-inverse of A, then there exists a matrix X such that
$$C = L + X(I - AL)$$
Show this.

Solution. Here is the solution.

a) We have mn unknowns, (the entries of C) and n^2 equations ($CA = I$). These equations are linear in the entries of C, and in fact we could rearrange them to be in the form $Ax = b$ where x is a mn-dimensional vector containing the entries of C. Since $m > n$ we have more unknowns than equations. We also know that there is at least one solution, since A is left-invertible. We have an undertermined linear system, and so there are infinitely many solutions.

b) We have to find C satisfying two constraints. The first is that it should be a left inverse, that is $CA = I$. The second is that $v_1 = v_2$, which we can write as $Cf = 0$ where $f = e_1 - e_2$. Then we let
$$B = \begin{bmatrix} A & f \end{bmatrix} \qquad G = \begin{bmatrix} I & 0 \end{bmatrix}$$
and we would like to solve the equation $CB = G$. If we write C and G in terms of their columns
$$C = \begin{bmatrix} c_1^\mathsf{T} \\ c_2^\mathsf{T} \\ \vdots \\ c_n^\mathsf{T} \end{bmatrix} \qquad G = \begin{bmatrix} g_1^\mathsf{T} \\ g_2^\mathsf{T} \\ \vdots \\ g_n^\mathsf{T} \end{bmatrix}$$
then the equation $CB = G$ is equivalent to
$$B^\mathsf{T} c_i = g_i \qquad \text{for } i = 1, \ldots, n$$
There exists a solution to these equations iff $g_i \in \operatorname{range} B^\mathsf{T}$, that is a solution to the problem exists iff
$$\begin{bmatrix} e_i \\ 0 \end{bmatrix} \in \operatorname{range}(B^\mathsf{T}) \qquad \text{for } i = 1, \ldots, n$$
In this case we can solve for c_i using any method of solving linear equations; e.g. QR factorization.

c) We have, using the notation of the previous part:

$$B = \begin{bmatrix} 1 & 2 & 1 \\ 3 & 4 & -1 \\ 5 & 6 & 0 \end{bmatrix}$$

This matrix B is invertible, and so $C = GB^{-1}$. This gives

$$C = \begin{bmatrix} -1 & -1 & 1 \\ 5/6 & 5/6 & -2/3 \end{bmatrix}$$

Because B is invertible, this solution is unique.

d) Note that we can rewrite our objective in its transpose form as follows:

$$\begin{aligned} \underset{C^T}{\text{minimize}} \ & \|C^T - \tilde{C}^T\|_F^2 \\ \text{subject to: } & A^T C^T = I \end{aligned}$$

Then, because $\|M\|_F^2 = \sum_{i=1}^n \|m_i\|_2^2$, where m_i are the columns of M, we can rewrite this problem again as follows:

$$\begin{aligned} \underset{c_1, c_2}{\text{minimize}} \ & \|c_1 - \tilde{c}_1\|_2^2 + \|c_2 - \tilde{c}_2\|_2^2 \\ \text{subject to: } & A^T c_1 = e_1 \\ & A^T c_2 = e_2, \end{aligned}$$

where c_1, c_2 are the columns of C^T (i.e., rows of C); $\tilde{c}_1, \tilde{c}_2$ are the columns of $\tilde{C}^T$ (i.e., rows of $\tilde{C}$); and $e_1, e_2 \in \mathbb{R}^2$ are the standard basis unit vectors. Finally, because c_1 and c_2 are independent in the objective functions and constraints, we can solve the above problem by actually solving two separate problems:

$$\begin{aligned} \underset{c_1}{\text{minimize}} \ & \|c_1 - \tilde{c}_1^T\|_F^2 \\ \text{subject to: } & A^T c_1 = e_1 \end{aligned}$$

and

$$\begin{aligned} \underset{c_2}{\text{minimize}} \ & \|c_2 - \tilde{c}_2^T\|_F^2 \\ \text{subject to: } & A^T c_2 = e_2 \end{aligned}$$

Both of these are general norm minimization problems with equality constraints, and can be solved with the methods outlined on slides 12-13 of lecture 13 (Least-norm solutions of underdetermined equations). The solution to the first problem is given by

$$\begin{bmatrix} c_1 \\ \lambda_1 \end{bmatrix} = \begin{bmatrix} I & A \\ A^T & 0 \end{bmatrix}^{-1} \begin{bmatrix} \tilde{c}_1 \\ e_1 \end{bmatrix}, \tag{10}$$

where you discard λ_1 after solving. Similarly, the solution to the second problem is given by

$$\begin{bmatrix} c_2 \\ \lambda_2 \end{bmatrix} = \begin{bmatrix} I & A \\ A^T & 0 \end{bmatrix}^{-1} \begin{bmatrix} \tilde{c}_2 \\ e_2 \end{bmatrix}, \tag{11}$$

where you discard λ_2 after solving. Using Julia to solve these two minimization problems, we find that our solution $C^\star$ is given by

$$C^\star = \begin{bmatrix} -1 & -1 & 1 \\ \frac{3}{4} & 1 & -\frac{3}{4} \end{bmatrix}. \tag{12}$$

It is easy to verify that this is indeed a left inverse of A. The Julia code to accomplish this is shown below:

```
using LinearAlgebra;

A = [1.0 2.0; 3.0 4.0; 5.0 6.0]
C_tilde = [-1.0 -1.0 1.0; 1.0 1.0 -1.0]
m, n = size(A)

C = similar(C_tilde)
A_tilde = reduce(vcat, [reduce(hcat, [I(m), A]),
                        reduce(hcat, [A', zeros((n, n))])])

for i in 1:2
    e_i = I(n)[i, :]
    b_tilde = reduce(vcat, [C_tilde[i, :], e_i])
    x_tilde = A_tilde \ b_tilde
    C[i, :] = x_tilde[begin:m]
end

display(C)
display(C*A)
```

e) Suppose $C = L + X(I - AL)$, then

$$\begin{aligned} CA &= (L + X(I - AL))\, A \\ &= LA + X(I - AL)A \\ &= LA + X(A - ALA) \\ &= I \end{aligned}$$

Therefore C is a left inverse of A.

f) Suppose $CA = I$. Then observe the following:

$$\begin{aligned} C &= L + C - L \\ &= L + C - (CA)L \\ &= L + C(I - AL) \\ &= L + X(I - AL) \end{aligned}$$

where $X = C$. Therefore, $C = L + X(I - AL)$ for some X.