EE263 Exam, November 2023

- This is a 24-hour take-home midterm. Please turn it in on Gradescope. Be aware that you *must* turn it in within 24 hours of downloading it. After that, Gradescope will not let you turn it in and we cannot accept it.
- You may use any books, notes, or computer programs. You may not discuss the exam or course material with others, or work in a group.
- The exam should not be discussed at all, with anyone, until 11/7 after everyone has taken it.
- If you have a question, please submit a private question on Ed, or email the staff mailing list. We have tried very hard to make the exam unambiguous and clear, so unless there is a mistake on the exam we're unlikely to say much.
- We expect your solutions to be legible, neat, and clear. Do not hand in your rough notes, and please try to simplify your solutions as much as you can. We will deduct points from solutions that are technically correct, but much more complicated than they need to be.
- Please check your email during the exam, just in case we need to send out a clarification or other announcement.
- Start each question on a new page, and make sure to correctly assign pages to problems in gradescope.
- When a problem involves some computation (say, using Julia), we do not want just the final answers. We want a clear discussion and justification of exactly what you did as well as the final numerical result.
- **You must turn in your code**. Include the code in your pdf submission.
- In the portion of your solutions where you explain the mathematical approach, you *cannot* refer to Julia operators, such as the backslash operator. (You can, of course, refer to inverses of matrices, or any other standard mathematical constructs.)
- Some of the problems require you to download data or other files. These files can be found at the URL

  `https://ee263.stanford.edu/mid23`

- Good luck!

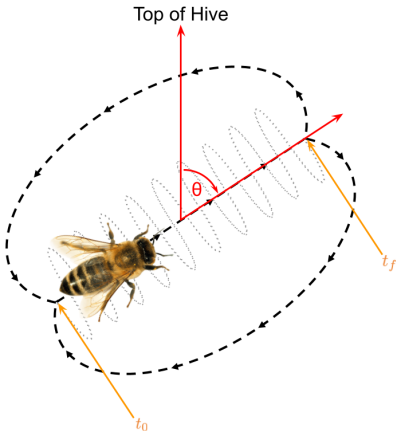
1. **Real-time interpretation of the honeybee waggle dance.** Upon discovering a food source that is farther than 100 meters from its resident hive, a forager honeybee will begin a cyclic "waggle" dance that follows the following form: (see figure 1a)

    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*.)

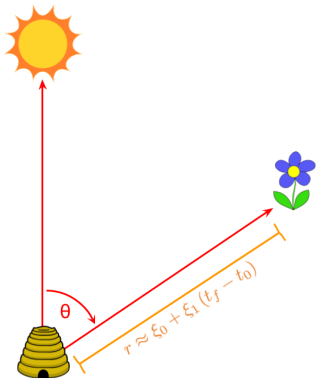
    2. The forager then turns either to the left or to the right, circling around to return to the start of the waggle run. (This portion of the dance is referred to as the *return*.)

    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)?

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.

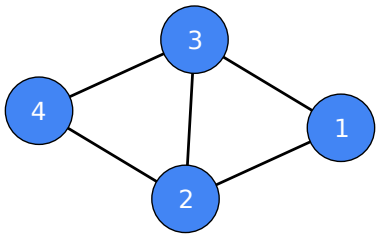
d) Now we would like to trade-off the two objectives

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

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.

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 $\text{src}(k)$ and a destination vertex $\text{dst}(k)$. 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

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

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

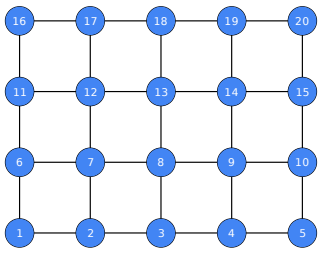
$$\begin{array}{ll} \text{minimize} & \displaystyle\sum_{k=1}^{m} \|x_{\text{src}(k)} - x_{\text{dst}(k)}\|^2 \\ \text{subject to} & x_i = p_i \quad \text{for } i \in \mathcal{V}_{\text{fixed}} \end{array}$$

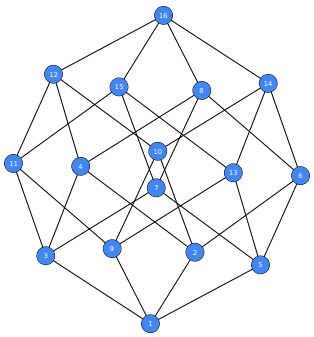
The idea is that we would like to position the vertices so that vertices connected by an edge are close together, while maintaining the position of the fixed vertices.

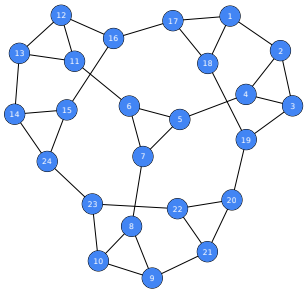
Let $X \in \mathbb{R}^{n\times 2}$ be a matrix whose ith row is the transpose of the position x_i of node i. Express this optimization problem in terms of X. Be sure to define carefully any additional matrices that you need.

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.

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

$$A = U \begin{bmatrix} I_r & 0 \\ 0 & 0 \end{bmatrix} V \tag{2}$$

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 $y \in$ 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 $y \notin$ 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 (2).

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 (2) holds.

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.