Final exam solutions

This is a 15 hour take-home final with 4 problems. Please submit your solutions to GradeScope at most 15 hours after you receive the exam.

- You may use any books, notes, or computer programs (e.g., matlab), but you may not discuss the exam with others until Dec. 12, after everyone has taken the exam and the solutions are posted. The only exception is that you can ask the course staff for clarification, by emailing to the staff email address ee263-fall1819-staff@lists.stanford.edu (please include the question number in your email subject, e.g. “Q1 EE263 final ...”). We have tried pretty hard to make the exam unambiguous and clear, so we’re unlikely to say much. Please do not post any exam related questions on Piazza.

- Since you have 15 hours, we expect your solutions to be legible, neat, and clear. Do not submit 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 a few times during the exam, just in case we need to send out a clarification or other announcements. It’s unlikely we will need to do this, but you never know.

- If a problem asks for some specific answers, make sure they are obvious in your solutions. You might put a box around the answers, so they stand out from the surrounding discussion, justification, plots, etc.

- When a problem involves some computation (say, using matlab), we do not want just the final answers. We want a clear discussion and justification of exactly what you did, the well-commented matlab source code that produces the result, and the final numerical result or plots. Be sure to show us your verification that your computed solution satisfies whatever properties it is supposed to, at least up to numerical precision. For example, if you compute a vector $x$ that is supposed to satisfy $Ax = b$ (say), show us the matlab code that checks this, and the result. (This might be done by the matlab code $\text{norm}(A*x-b)$; be sure to show us the result, which should be very small.) We will not check your numerical solutions for you, in cases where there is more than one solution.

- In the portion of your solutions where you explain the mathematical approach, you cannot refer to matlab operators, such as the backslash operator. (You can, of course, refer to inverses of matrices, or any other standard mathematical construct.)
• Some of the problems are described in a practical setting, such as computational Biology or image processing. *You do not need to understand anything about the application area to solve these problems.* We have taken special care to make sure all the information and math needed to solve the problem is given in the problem description.

• Some of the problems require you to download and run a matlab file to generate the data needed. These files can be found at the URL

   http://ee263.stanford.edu/finalfun/final18data.zip

• Please respect the honor code. Although we encourage you to work on homework assignments in small groups, *you cannot discuss the final with anyone*, with the exception of EE263 course staff, until Dec. 12, when everyone has taken it and the solutions are posted online.
1. **Inverse dynamics to compute joint torques in the human body [20 points]**

Inverse dynamics is the procedure of using motion data to estimate joint torques in the human body. Joint torques are rotational moments about joints (i.e. knee, ankles, hip, etc.) that cause the human body to move around. This technique is a primary tool for analysis of human gait and other movements.

A variety of data are needed to make inverse dynamics calculations. These include the inertial properties of each limb, a record of how the limb moved, and often a measure of the ground reaction forces (i.e. forces and torques exerted by the ground on body parts touching the ground) that occurred during the motion.

Unfortunately for biomechanists, collecting position data is inherently noisy (tracking the *exact* position of a limb’s center of mass is quite difficult). A potential solution to this is to combine all movement data with ground reaction force data, which creates an overdetermined system.

In this problem, we will resolve the inverse dynamics problem by solving an overdetermined system of equations at each point in time for a 2D model of the human body. We treat the inverse dynamics problem as an overdetermined system of equations and use optimization to find a set of joint torques that best agrees with available measurements at each point in time. Said another way, we choose the joint torques that minimize the adjustments necessary to make the measurements agree with one another. Our goal then, is to find a torque vector, $\tau$, that solves the following optimization problem:

$$
\text{minimize } J = \| A\tau - b \|^2,
$$

for some appropriate $A$ matrix and $b$ vector.

Figure 1 describes the simplified mechanical system for this problem. *Note: The remainder of this paragraph is not necessary to solve the problem, but may help clarify some details*. Limbs are rigid segments connected by free-rotating joints. The body comprises of $n$ segments, with segment 1 motionless and making contact with the ground (i.e. force plate). Each segment $i$ (with $i = 1, 2, \ldots, n$), has a center-of-mass at a specified location in the 2D Cartesian coordinate system, as well as a scalar orientation in spacial coordinates, $\phi_i$. Note that $\phi_i$ is measured relative to the ground plane, not relative to the previous segment. In two dimensions, knowledge of $\phi_i$ for $i = 1, 2, \ldots, n$ is sufficient to fully determine the configuration of the system, including the location of all body segment’s center-of-mass. Hence, $\phi_i$ can be thought of as the general coordinate for each segment. Segment $i$ is connected to segment $i - 1$ through joint $i - 1$ and segment $i + 1$ through joint $i$. Each joint is acted upon by a force, $f_i \in \mathbb{R}^2$ and joint torque, $\tau_i \in \mathbb{R}$. The ground reaction forces and torques are $f_0$ and $\tau_0$, respectively, and act on the ground point of contact of segment $i = 1$. In this way, $f_0$ and $\tau_0$ act on a *fake* joint 0, and $\phi_1$ is constant and unnecessary to the problem.

The aforementioned quantities will be stacked together and referred to in vector form:
Figure 1: Configuration of the two-dimensional body segment model, with bottom segments separated for clarity. The force plate on the ground (solid horizontal line at bottom of the image) measures \( \tau_0 \) and \( f_0 \). Motion capture analysis (i.e. body markers and many cameras) is used to measure \( \phi_i \).

\[
\phi = \begin{bmatrix} \phi_2 \\ \vdots \\ \phi_n \end{bmatrix}, \quad \tau = \begin{bmatrix} \tau_1 \\ \vdots \\ \tau_{n-1} \end{bmatrix}
\]

From this nomenclature, and our understanding of physics, there are two governing equations that create our overdetermined system. First, we have the equations of motion relating joint torques and angular accelerations, expressed in the form:

\[
M \ddot{\phi} = Q\tau + g + v,
\]

where \( \ddot{\phi} \) is the second derivative of \( \phi \) with respect to time, which is collected during the experiment.

The second set of equations relate joint torques to the ground reaction forces. These equations are used to find the constraint forces necessary to keep the lowest body segment motionless with respect to the ground (i.e. ground reaction forces):
\[ C\ddot{\phi} = \begin{bmatrix} \tau_0 \\ f_0 \end{bmatrix} + c. \]

You can assume you know all the parameters in the following list:

- \( \ddot{\phi} \), which is the second derivative of \( \phi \) with respect to time, which is collected during the experiment
- \( M \) is the mass matrix, which is invertible
- \( Q \) is a square matrix converting joint torques to segment torques, but is not necessarily invertible
- \( g \) is a vector containing gravitation terms
- \( v \) is a vector containing velocity-dependent terms for the joints and segments
- \( C \) is an intermediate term relating angular accelerations and ground reaction forces
- \( c \) is a vector containing velocity-dependent terms for the ground reactions
- \( \tau_0 \) and \( f_0 \) are the estimated ground reaction torque and ground reaction force, respectively. In the 2D model, we have \( \tau_0 \in \mathbb{R} \) and \( f_0 \in \mathbb{R}^2 \).

Now we have enough information to analyze our system. Let’s begin!

(a) Let’s look at a snapshot of our system at a particular point in time. Given the equations above, find matrix \( A \) and vector \( b \) such that \( A\tau = b \) is an overdetermined system of equations, and explain how to find the \( \tau \) that minimizes \( \| A\tau - b \| \). Your solution should be analytic (i.e. in terms of the named matrices and vectors given above). In order to get the most accurate estimate of \( \tau \), we expect you to use all measurements in the experiment, which includes \( \tau_0, f_0, \) and \( \ddot{\phi} \). If you need to make assumptions about properties of matrices, please state them. Please explicitly declare the dimensions of your final \( A \) matrix and \( b \) vector.

(b) As discussed earlier, each torque and force measurement is inherently noisy. However, researchers often have prior knowledge about the relative noise in each measurement (for example, measurements made using a force plate may be less noisy than those using motion capture). Suppose each measurement error was independent and we have existing knowledge of the nonzero variance, \( \sigma_i^2 \), of each torque and force measurement. We can potentially improve our optimized solution by weighting each measurement by the inverse variance\(^1\). To be precise, we weight each measurement \( \tilde{a}_i^\top \tau - b_i \) by the inverse of the variance \( \sigma_i^2 \) associated with it.

---

\(^1\)Weighting an overdetermined system by the inverse variance of each measurement has the effect of putting more emphasis on the measurements with lower variance. In other words, we give higher weight to measurements we are more confident are accurate.
Given this new optimization task, derive an analytic solution for the optimal $\tau$. Please explicitly declare the dimensions of any new matrices/vectors you introduce in your solution. If you need to make assumptions about properties of matrices, please state them.

(c) Our solution to finding joint torques in the human body can be enhanced by eliminating particular measurement biases. Measurement biases are factors that yield error in our measurements and result in additive noise\(^2\). Recall that we usually treat noise as:

\[ y = Ax + v, \]

and we assume $v$ is small, so we can ignore it. In this part, we will NOT ignore $v$, as it has structure that can be taken into account, as explained below.

If we are clever, we can effectively eliminate the particular measurement biases that are constant over time. We accomplish this by including these biases as estimation variables, and performing the estimation across all points in time simultaneously.

Suppose we have $n_b$ sources of bias in the laboratory, whose values are unknown but do not change over time. We can assemble these biases in a vector $\beta \in \mathbb{R}^{n_b}$. Further, suppose we have a known matrix $B \in \mathbb{R}^{m \times n_b}$ (where $m$ is the number of rows of $A$), with entries:

\[ B_{ij} = \begin{cases} 1 & \text{if bias } j \text{ is added to measurement } i \\ 0 & \text{else} \end{cases} \]

Like before, our goal is to find the best estimate for $\tau$. However, we now know that all of the measurements have added constant bias dictated by $\beta$ and $B$. Our strategy is to take $N$ measurements at $N$ different times. Let $\phi(k), f_0(k), \tau_0(k)$ be the measurements taken at time $k$. Using all of these measurements, find an analytic solution for the optimal joint torques at all steps, $\tau(k)$, for $k = 1, 2, \ldots, N$, as well as the bias vector $\beta$. Please explicitly declare the dimensions of any new matrices/vectors you introduce in your solution.

**Note:** In this part, assume there is no weighting on measurements (i.e. ignore part b), or simply assume $\sigma_i^2 = 1$ for all $i$.

**Note:** We’ve purposely left the $B$ matrix definition very general, so that no matter how you construct your $A$ matrix, you can assume the $i$’th row of $B$ corresponds to the $i$’th row of $A$.

**Hint:** Though we didn’t mention this up until now, the matrices $M$, $C$, $c$, and $v$ are all $\phi$-dependent (i.e. $M = M(\phi)$, etc.). Remember, in this part of the problem, $\phi$ changes at each time, $k$.

---

\(^2\)Measurement biases that are constant in time affect measurements the same way throughout the experiment (i.e. over time). These can include hardware malfunctions in the force plate, inaccurate measurements of segment length or segment mass, etc.
Solution.

(a) Starting with

\[ M(\phi)\ddot{\phi} = Q\tau + g(\phi) + v(\phi, \dot{\phi}) \]  \hspace{1cm} (1)

\[ C(\phi)\ddot{\phi} = \begin{bmatrix} \tau_0 \\ f_0 \end{bmatrix} + c(\phi, \dot{\phi}) \]  \hspace{1cm} (2)

We want to move \( \tau \) into the second equation, so we can use \( \tau_0, f_0 \) in our calculations. To do so, we rearrange terms in the first equation to solve for \( \ddot{\phi} \).

\[ \ddot{\phi} = M(\phi)^{-1}(Q\tau + g(\phi) + v(\phi, \dot{\phi})) \]

Plugging this into the second equation, we have

\[ C(\phi)M(\phi)^{-1}(Q\tau + g(\phi) + v(\phi, \dot{\phi})) = \begin{bmatrix} \tau_0 \\ f_0 \end{bmatrix} + c(\phi, \dot{\phi}) \]

\[ C(\phi)M(\phi)^{-1}Q\tau = \begin{bmatrix} \tau_0 \\ f_0 \end{bmatrix} + c(\phi, \dot{\phi}) - C(\phi)M(\phi)^{-1}(g(\phi) + v(\phi, \dot{\phi})) \]  \hspace{1cm} (3)

Notice, 1 and 3 are linear in \( \tau \). The full over-determined system is formed by combining 1 and 3, and plugging in measurement for ground reaction forces and torques and angular accelerations into the right-hand side.

\[ \begin{bmatrix} C(\phi)M(\phi)^{-1}Q \\ M(\phi)^{-1}Q \end{bmatrix} \tau = \begin{bmatrix} \tau_0 \\ f_0 \end{bmatrix} + \begin{bmatrix} c(\phi, \dot{\phi}) \\ 0 \end{bmatrix} - \begin{bmatrix} C(\phi) \\ I_n \end{bmatrix} M(\phi)^{-1}(g(\phi) + v(\phi, \dot{\phi})) \]  \hspace{1cm} (4)

Using the following definition, our equation simplifies to the desired form.

\[ A = \begin{bmatrix} C(\phi)M(\phi)^{-1}Q \\ M(\phi)^{-1}Q \end{bmatrix} \]

\[ b = \begin{bmatrix} \tau_0 \\ f_0 \end{bmatrix} + \begin{bmatrix} c(\phi, \dot{\phi}) \\ 0 \end{bmatrix} - \begin{bmatrix} C(\phi) \\ I_n \end{bmatrix} M(\phi)^{-1}(g(\phi) + v(\phi, \dot{\phi})) \]

We have:

\[ A\tau = b \]
As we have \((n - 1)\) joints in the body, A must have \((n - 1) + p\) rows, where \(p\) is the number of ground reaction force measurements (in the 2D case, this equals 3). Further, A will have \(n - 1\) columns to align with our unknown joint torque vector, \(\tau\). We also have \(b \in \mathbb{R}^{n-1+p}\).

The least square solution for \(\tau\) is the familiar:

\[
\tau_{LS} = (A^\top A)^{-1} A^\top b
\]

We also accept the SVD solution for \(\tau\):

\[
\tau_{LS} = A^\dagger b
\]

(b) Let’s define a weight matrix \(W \in \mathbb{R}^{nm \times nm}\) as \(\text{diag}(\frac{1}{\sigma_1^2}, \frac{1}{\sigma_2^2}, \ldots, \frac{1}{\sigma_{nm}^2})\).

We can express the objective function as

\[
\sum_{i=1}^{nm} \sigma_i^{-2}(\tilde{a}^\top_i \tau - b_i)^2 = \sum_{i=1}^{nm} \left(\sqrt{\sigma_i^{-2}\tilde{a}^\top_i \tau} - \sqrt{\sigma_i^{-2}b_i}\right)^2
\]

\[
= \left\| \begin{bmatrix} \sigma_1^{-1} \tilde{a}_1^\top \tau \\ \vdots \\ \sigma_m^{-1} \tilde{a}_m^\top \tau \end{bmatrix} - \begin{bmatrix} \sigma_1^{-1} b_1 \\ \vdots \\ \sigma_m^{-1} b_m \end{bmatrix} \right\|^2
\]

\[
= \left\| W^{\frac{1}{2}} A \tau - W^{\frac{1}{2}} b \right\|^2
\]

Thus, minimizing this objective function is a least-squares problem; the solution is

\[
\tau = ((W^{\frac{1}{2}} A)^T (W^{\frac{1}{2}} A))^{-1} (W^{\frac{1}{2}} A)^T (W^{\frac{1}{2}} b) = (A^T W A)^{-1} A^T W b.
\]

Another way to think about this is as taking the gradient:

\[
J = \sum_{i=1}^{m} \frac{1}{\sigma_i^2} (\tilde{a}^\top_i \tau - b_i)^2
\]

\[
= (A\tau - b)^\top W (A\tau - b)
\]

\[
\nabla_\tau((A\tau - b)^\top W (A\tau - b)) = 0
\]

\[
\nabla_\tau(\tau^\top A^T W A \tau + b^\top W b - 2b^\top W A\tau) = 0
\]

\[
2(A^T W A \tau + 0 - A^T W b) = 0
\]

Rearranging and left-multiplying by \((A^T W A)^{-1}\):

\[
A^T W A \tau = A^T W b
\]
\[ \tau = (A^\top WA)^{-1}A^\top Wb \]

In our solution, we assume that the matrix \( A^\top WA \) is invertible (which is not guaranteed and is dependent on the scaling factors in \( W \)).

*Note:* Due to ambiguity in the question regarding weighting by \( \frac{1}{\sigma_\tau} \) or \( \frac{1}{\sigma_\tau^2} \), we accept both as elements of \( W \).

(c) For each time step \( k = 1, 2, \ldots, N \) we have the equation:

\[ A[k]\tau[k] = b[k] + B\beta \]

We can solve for each \( \tau[k] \) simultaneously by combining all time steps into a block linear system:

\[
\begin{pmatrix}
A[1] & -B & \cdots & -B \\
\vdots & \ddots & \ddots & \vdots \\
A[N] & -B & \cdots & -B \\
\end{pmatrix}
\begin{bmatrix}
\tau[1] \\
\tau[2] \\
\vdots \\
\tau[N] \\
\end{bmatrix}
= 
\begin{bmatrix}
b[1] \\
b[2] \\
\vdots \\
b[N] \\
\end{bmatrix}
\]

And using the definitions:

\[
\tilde{A} = 
\begin{pmatrix}
A[1] & \cdots & -B \\
\vdots & \ddots & \vdots \\
A[N] & \cdots & -B \\
\end{pmatrix}
\]

\[
\tilde{\tau} = 
\begin{bmatrix}
\tau[1] \\
\tau[2] \\
\vdots \\
\tau[N] \\
\end{bmatrix}
\]

\[
\tilde{b} = 
\begin{bmatrix}
b[1] \\
b[2] \\
\vdots \\
b[N] \\
\end{bmatrix}
\]

We can now solve this equation using SVD:

\[ \tilde{\tau} = \tilde{A}^{-1}\tilde{b} \]

Note that depending on the size of \( n_b \), \( \tilde{A} \) may be fat or skinny. The size of \( \tilde{A} \) is \( N(n-1+p) \times N(n-1) + n_b \), which means there are \( N(n-1+p) - N(n-1) - n_b \)
more rows than columns, which comes out to \( Np - nb \). In our particular problem, there are \( 3N - nb \) more rows than columns.

If \( nb \leq 3N \), the system would be skinny, at which point we would use the standard least-squares solution:

\[
\tilde{\tau}_{LS} = (\tilde{A}^\top \tilde{A})^{-1} \tilde{A}^\top \tilde{b}
\]

Otherwise, \( \tilde{A} \) would be fat, and we would use the standard least-norm solution, since that’s what has been taught in the course (and is a reasonable/justifiable choice for the solution given the context of the problem).

\[
\tilde{\tau}_{LN} = \tilde{A}^\top (\tilde{A} \tilde{A}^\top)^{-1} \tilde{b}
\]

**Remark:** This question was inspired by Kuo’s "A Least-Squares Estimation Approach to Improving the Precision of Inverse Dynamics".

---

Find the llama [25 points]

In some video applications, we would likely have a majority of the pixels stay relatively the same (the background), while the pixels corresponding to a moving object change dramatically over time. For applications such as video surveillance and object recognition, we would be interested in separating the pixels corresponding to the moving object of interest from the video background. In other words, given a video, we would like to produce effectively two videos. One video containing primarily just the moving object, while the other one would be relatively unchanged and contains primarily just the background.

We have a video showing a llama crossing a street. As a group of ecologists spying on this llama, we would like to track the movement of this llama in the video. In this question, you are going to separate out the pixels that contain primarily just the llama.

A video is composed of a number of video frames that play at a certain frame rate. To simplify the problem, we use a gray-scale video. Each frame can be shown as an image that contains pixel values as a matrix. For a $240 \times 426$ resolution gray-scale video, each frame would be a $m \times n = 240 \times 426$ matrix containing integer values.

In the data file, you will find a movie clip llama.mp4 in $240 \times 426$ resolution consisting of $f = 48$ frames. The movie clip itself is just for your reference, and you do not need to process it at all. We have provided the data extracted from this movie clip named llama_data (you can use the matlab function load to read a .mat file). You can work with just the llama_data file provided, in which you will find a matrix $X_f \in \mathbb{R}^{mn \times f}$ of the form:

$$X_f = [x_1 \ x_2 \ \ldots \ x_f],$$

where $x_j \in \mathbb{R}^{mn}$ represents the $j^{th}$ frame reshaped to a vector, where $j = 1, \ldots, f$.

We can approximate the video as an autonomous linear dynamic system of the form

$$x_{j+1} = Ax_j, \quad (5)$$

where $A \in \mathbb{R}^{mn \times mn}$ maps the $j^{th}$ frame to the $(j+1)^{th}$ frame.

(a) Consider the matrices:

$$X_{1,f-1} = [x_1 \ x_2 \ \ldots \ x_{f-1}], \quad X_{2,f} = [x_2 \ x_3 \ \ldots \ x_f].$$

Find a relationship between $X_{1,f-1}$ and $X_{2,f}$ using the matrix $A$.

(b) Now we define a new relationship between $X_{1,f-1}$ and $X_{2,f}$. Let the last frame, $x_f$, be related to all the previous frames in the following way:

$$x_f = \sum_{j=1}^{f-1} b_j x_j + r, \quad (6)$$

11
where \( b_j \in \mathbb{R} \) is a coefficient corresponding to the \( j^{th} \) frame, and \( r \in \mathbb{R}^{mn \times 1} \) is the error in the approximation, which we will ignore, i.e \( r \approx 0 \). Using this new relationship, find a matrix \( M \) such that:

\[
X_{2,f} = X_{1,f-1}M. \tag{7}
\]

Write the matrix \( M \) in terms of the coefficients \( b_j \).

Once the video is modeled as an autonomous linear dynamical system, the background can be considered to be the portion of the video that is associated with modes of the system that cause only small changes in pixel values \( x_j \) across different frames, while moving object portion of the video is associated with modes with significant change over frames. Now our goal is to perform a modal decomposition on \( A \) in order to separate the modes associated with background from those associated with the moving object(s).

**Note:** For the following parts, you may make approximations by using generalized pseudo-inverse in the case of the matrices not being invertible.

(c) Matrix \( A \) is a very high-dimensional matrix. In order to decrease the computational complexity, we want to perform modal decomposition on the matrix \( M \), which is of a much lower dimension. First, describe how to solve for \( M \) using equation (7). Then, using the relationships you found in part (a) and equation (7), describe how to formulate a new equation:

\[
A\Theta = \Theta\Lambda, \tag{8}
\]

where \( \Lambda \) is a diagonal matrix. Explicitly describe how to find matrices \( \Theta \) and \( \Lambda \) from the problem data.

**Hint:** You can assume \( M \) is diagonalizable. Do not try to perform diagonalization directly on \( A \), which is a matrix of dimension \( mn \times mn \).

(d) From the result of part (c) it can be shown that \( A \approx \Theta\Lambda\Theta^\dagger \) is a very good low-rank approximation to \( A \) (You do not need to show this fact). Let \( \Lambda \) be expressed as:

\[
\Lambda = \begin{bmatrix}
\lambda_1 & 0 & 0 & \cdots & 0 & 0 \\
0 & \lambda_2 & 0 & \cdots & 0 & 0 \\
\vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\
0 & 0 & 0 & \cdots & \lambda_{f-2} & 0 \\
0 & 0 & 0 & \cdots & 0 & \lambda_{f-1}
\end{bmatrix}.
\]

Using the first frame \( x_1 \) as the initial state, find \( x_f \) (the final frame) in terms of \( f, x_1, \Theta, \) and \( \Lambda \). This gives us the modal decomposed form of \( f^{th} \) frame.

**Note:** To simplify the math you can make the assumption that \( \Theta^\dagger\Theta \approx I \). Although this is not strictly correct, it is a reasonable assumption for this problem.
In order to separate out the background, we want to find the modes that are associated with smaller and slower change. *Fourier mode* $\omega_j$ indicates how rapidly the associated mode changes with time. To convert eigenvalues to Fourier modes, use the following relationship:

$$\omega_j = \ln(\lambda_j).$$ \hspace{1cm} (9)

If your eigenvalues are complex, then you would need to take the logarithm of a complex number, which is handled automatically by all standard computational software. As a result, some Fourier modes can be complex and we would want the magnitude of the background Fourier mode(s) $w_j$ to be close to 0. In other words, if a Fourier mode $w_j$ is located closer to the origin in complex plane, then the portion of the first video frame associated with it would have small change over time, and can be considered as background.

(e) To get started, modify your expression of $x_f$ from part (d) in terms of the Fourier modes $\omega_j$ instead of $\lambda_j$ in the matrix $\Lambda$.

(f) In this problem, you can take any mode(s) with $||\omega_j|| < 0.01$ to be considered as background. Find the Fourier Modes in Matlab, and sort them by magnitude. Plot the Fourier modes in complex plane. *(Note: Plot the Fourier Modes as discrete data points *i.e. use dots, circles etc. to represent each mode. Do not connect all the modes with lines.)* Clearly state how many and which Fourier mode(s) you picked for the background.

Now that you have completed the separation of the modes associated with the background, going forward you should be only working with the background specific modes. Although earlier we assumed that $\Theta^\dagger\Theta \approx I$, you might notice that this is not actually true for our example video. However, when working with just the background modes, this assumption is a valid one, and hence can be used to successfully separate background of the last frame using the first frame.

(g) Show an image of the background you extracted out for the last frame in the data. The result will not be perfect, since we have made some approximations along the way.

**Hint:** Consider using the indices from sorting of Fourier modes you did in part (f) to simplify your code. In case you run into *out of memory* error, think about the dimension of matrices and how to split the operation up to reduce memory usage. To show the image, reshape the background $x_{f,bg}$ from a vector to a matrix $\text{img}$ with correct dimensions, and then display it as an image as follows:

In Matlab:

```
colormap gray
imagesc(img)
axis equal
axis off
```
In Python:

```python
import matplotlib.pyplot as plt
plt.imshow(img, cmap='gray')
plt.show()
```

In Julia:

```julia
using PyPlot
imshow(img, cmap="gray")
```

**Note:** If you have complex values as your pixel values, only keep the real part.

(h) Provide a picture of the llama from the last frame. You can find it by subtracting background $x_{f,bg}$ from the original final video frame $x_f$.

$$x_{f,llama} = x_f - x_{f,bg}$$

Now you have successfully found the llama. The official mascot for EE263!

**Solution.**

(a) Based on the linear dynamical system form:

$$X_{1,f-1} = \begin{bmatrix} x_1 & x_2 & \ldots & x_{f-1} \end{bmatrix}$$

$$x_{j+1} = Ax_j$$

$$X_{2,f} = \begin{bmatrix} x_2 & x_3 & \ldots & x_f \end{bmatrix}$$

$$= \begin{bmatrix} Ax_1 & Ax_2 & \ldots & Ax_{f-1} \end{bmatrix}$$

$$= A \begin{bmatrix} x_1 & x_2 & \ldots & x_{f-1} \end{bmatrix}$$

Therefore, we can get:

$$X_{2,f} = AX_{1,f-1}$$

(b)

$$M = \begin{bmatrix} 0 & 0 & 0 & \ldots & 0 & 0 & b_1 \\ 1 & 0 & 0 & \ldots & 0 & 0 & b_2 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \ldots & 1 & 0 & b_{f-2} \\ 0 & 0 & 0 & \ldots & 0 & 1 & b_{f-1} \end{bmatrix}$$
(c) From (a) and (b), we have:

\[ AX_{1,f-1} = X_{2,f} \]
\[ = X_{1,f-1}M \]

M can be defined as:

\[ M = X_{1,f-1}^\dagger X_{2,f} \]

Performing diagonalization on M:

\[ AX_{1,f-1} = X_{1,f-1}V\Lambda V^{-1} \]
\[ A(X_{1,f-1}V) = (X_{1,f-1}V)\Lambda \]
\[ A\Theta = \Theta\Lambda, \text{ where } \Theta = X_{1,f-1}V \]

(d) We are given,

\[ A \approx \Theta\Lambda\Theta^\dagger \]

Then we can use the diagonalization of A to define \( A^t \)

\[ A^t = \Theta\Lambda\Theta^\dagger\Theta\Lambda\Theta^\dagger \ldots \Theta\Lambda\Theta^\dagger = \Theta\Lambda^t\Theta^\dagger; \]

Therefore the last frame can be defined using the solution of a discrete-time autonomous LDS as follows:

\[ x_f = A^{f-1}x_1 \]
\[ = \Theta\Lambda^{f-1}\Theta^\dagger x_1 \]

Note: This relationship can be used to define any frame as long as the video scene has relatively the same background as the first frame.

(e) Based on the relationship between Fourier mode and eigenvalue, you can write the following:

\[ w_j = \ln(\lambda_j) \]
\[ \lambda_j = e^{w_j} \]

Then all you need to do is change the matrix \( \Lambda \) by replacing each \( \lambda_j \) with \( e^{w_j} \)

\[ \Lambda = \begin{bmatrix} e^{w_1} & 0 & 0 & \ldots & 0 & 0 \\ 0 & e^{w_2} & 0 & \ldots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \ldots & e^{w_{f-2}} & 0 \\ 0 & 0 & 0 & \ldots & 0 & e^{w_{f-1}} \end{bmatrix} \]

\[ x_f = A^{f-1}x_1 \]
\[ = \Theta\Lambda^{f-1}\Theta^\dagger x_1 \]

15
(f) Code for parts (f), (g), and (h) shown here.

```matlab
%%
close all;
clear all;
load llama.mat
m = 240;
n = 426;
f = 48;
Llama = zeros(m*n, f-1);
BG = zeros(m*n, f-1);
X1_f1 = Xf(:, 1:f-1);
X2_f = Xf(:, 2:f);
M = X1_f1\X2_f;
[W, Lambda] = eig(M);
Lambda = diag(Lambda);
w = log(Lambda);
[B, I] = sort(abs(w));
l = 1;
I = I(1:l);
figure
plot(w, '.', 'MarkerSize', 20)
hold on
plot(real(w(I(1:l))), imag(w(I(1:l))), '.', 'MarkerSize', 20, 'Color', 'r')
xlabel('Re')
ylabel('Im')
disp('Chosen fourier modes as background:')
w(I)
Lambda = diag(exp(w(I)));
Theta = X1_f1*W; Theta_bg = Theta(:, I);

% Show results for the last frame
for t = 1:f
    bg = Theta_bg*Lambda^(t-1)*(pinv(Theta_bg)*Xf(:,1));
    llama = Xf(:,t) - bg;
    bg = reshape(bg, m, n);
    llama = reshape(llama, m, n);
    figure
colormap gray; imagesc(bg); axis off; axis equal % display last frame
    % background
figure
colormap gray; imagesc(llama); axis off; axis equal; % display last frame
    % extracted llama

%% For all frames - Not expected from students
for t = 1:f
    bg = Theta_bg*Lambda^(t-1)*(pinv(Theta_bg)*Xf(:,1));
    total = Xf(:,t);
    llama = total - bg;
    BG(:, t) = bg;
    Llama(:, t) = llama;
```
The chosen Fourier Mode for background is $-2.35 \times 10^{-4}$. The Fourier Modes in complex space are plotted below. The red point represents the Fourier mode that got chosen for background.

(g) Original last frame:
The extracted background:

(h) The extracted llama:
Remark: This question was inspired by Grosek and Kutz’s "Dynamic Mode Decomposition for Real Time Background/Foreground Separation in Video".\(^4\)

3. Hugo’s Hoverboards [35 points]

**Note:** You should not analytically solve any integrals in this problem. If integrals appear in your answer, you should (a) if the section is analytical, leave them as integrals or (b) if the section is numeric, numerically approximate them using a pre-built function, such as `trapz()` in Matlab or the equivalent in your programming language of choice.

It’s finally winter break, and you have gone to Paris for a few days on vacation. Since you don’t have much time to see the city, you decide to rent a hoverboard from the general appliance store Voltage Victor’s. Unlike previous models, these hoverboards are omni-direction, so you don’t have to worry about turning in crowded streets – just push left and off you go! The hoverboard has only two steering options: (1) right / left and (2) forward / back, each of which is continuous with no upper / lower limit (that is, you can theoretically approach the speed of light without risk of physical failure). For the sake of this problem, assume that orthogonal steering can be done simultaneously and additively (that is, if you steer forwards and right, you will go diagonally).

(a) You wish to model your hoverboard using a linear dynamic system with inputs and outputs of the form:

\[
\dot{x}(t) = Ax(t) + Bu(t) \quad y(t) = Cx(t) + Du(t).
\]

Define your state vector as your current 2D position and 2D velocity: \(x(t) = [p_x \ p_y \ v_x \ v_y]^T\). Define your output \(y(t)\) as 2D position at time \(t\). \(u(t) \in \mathbb{R}^2\) is the steering input.

i. In order to ensure accurate modeling, you take your hoverboard on a few test-drives and carefully record all data for your input, state, and output as defined above. You conduct two different types of runs to gather your data: zero-input and constant-input. First, for the zero-input tests, you conduct \(N\) runs, each of duration \(d\), in which you log both the initial and final state \((x^{(i)}(0), x^{(i)}(d), \text{respectively}, i = 1, \ldots, N)\) and apply no input throughout the run. You may assume \(N\) is sufficiently large to yield reasonable estimates. You then run \(N\) more tests, also of duration \(d\), still logging both the initial and final state, but now applying a constant input \(u\) throughout the entire duration of the test. Note that while \(u\) is constant within each test, it can vary between tests. Describe how you will use this data to estimate \(A\) and \(B\). Also determine \(C\) and \(D\) from the context of the problem and the definition of the state variable \(x\).

**Hint:** In class, we learned about the matrix exponential. There is also a matrix logarithm which inverts the matrix exponential. That is, for nicely
behaved cases, $A = \log(e^A)$. You may use the matrix logarithm in your solution, and you may assume everything is nicely behaved.

ii. Using the data in `hoverboard_data_a`, form your best estimate for $A$ and $B$. See comments in the data file for more details (JSON does not support comments, but you can open the `.m` file in any text editor and read the comments). Include your results for these matrices explicitly, either handwritten or printed neatly in your code.

**Note:** Matlab, the Python Scipy package, Julia, and Octave all have a function `logm()` which may be useful here in applying the matrix logarithm, as described in the prior part.

After conducting all the experiments above, your hoverboard suddenly gives out! Disillusioned with Voltage Victor’s, you decide to go to the specialty shop, Hugo’s Hoverboards. Since everyone at Hugo’s Hoverboards is a hoverboard specialist, they are able to provide you the specs for your hoverboard directly in `hoverboard_data_b`. Use this $A, B$ for the remainder of the problem; $C, D$ should still be those that you determined in 3a as these are not specific to the hoverboard.

(b) Plot the impulse response of the system for this new system. Note that the impulse response $h(t)$ is defined as a matrix; please plot all components separately but on a single plot and clearly label each component. Plot the time range $t \in [0, 10]$. If any impulsive terms exist in your impulse response, you may ignore them for the sake of plotting.

(c) In order to focus on taking in the city, you decide to plan out your route the night before and write a simulation to pre-compute your desired steering commands. (You leave manual over-ride so that you can manage crowded streets and other unpredictable situations.)

i. Show how to discretize your system using the discretization step $h$. You will work with a discrete LDS for parts 3c-3e. You should show the discretized form of the problem and explicitly define all coefficients and matrices.

ii. Using the discretized system, show how to plan the minimum energy route for a specific itinerary, ignoring street confines; that is, minimize $\mathcal{E} = \sum_{k=0}^{N-1} \|u_d(k)\|^2$; where $N$ is the total number of inputs. Begin at $x(0) = 0$; next reach site $s_1$ at time $\tau_1$; next reach site $s_2$; next reach site $s_3$ at time $\tau_3$. You must visit the sites in order, but you have no specific arrival time for site $s_2$. You may assume that $\tau_1, \tau_2, \tau_3$ are all evenly divisible by $h$. For modeling simplicity, you may assume that you do not remain at any stop after arrival.

iii. Using the discretization step $h = 1$ and the data in `hoverboard_data_c`, implement your approach from 3(c)ii. You must visit the sites in the order given. Sites with required arrival times are specified in `ar_t` – see comments in the data file for more details. Note that the arrival time given in the data
file is in terms of the original continuous time variable \( t \). Plot your trajectory.

(Note: this should be a single plot displaying both the \( x \) and \( y \) coordinates. There should not be a time axis.)

(d) Does changing the discretization step \( h \) in 3c allow you to meet your itinerary requirements at a lower energy? Please provide a brief (2 sentences max) explanation why or why not that does not invoke numerical results.

(e) You show your route plan from 3(c)iii to your friend, who comments that you must be planning to jump off your hoverboard, since you will have nonzero velocity when you reach the sites. She argues that, unfortunately, you will not even be able to stop your hoverboard at all of your sites given your current required arrival times (to stop you need \( v_x = v_y = 0 \)). Is your friend correct? If yes, propose a simple modification to your route-planning algorithm and briefly explain how this will allow you to stop. If no, briefly explain why. You do not have to implement your modification.

(f) Unfortunately the dynamics in part 3a aren’t quite right. Over the course of the day, your hoverboard loses battery capacity. As a result, applying the same input produces a weaker output. We model this decay as follows:

\[
\begin{align*}
\dot{x}(t) &= Ax(t) + f(t)Bu(t) \\
y(t) &= Cx(t) + Du(t),
\end{align*}
\]

for scalar-valued function \( f(t) = \max(2 - \gamma^{-t+\theta}, 0) \) where \( \gamma \in (0, 1) \) and \( \theta > 1 \). Given these new dynamics, will the reachable set \( \mathcal{R}_t \) change? In other words, if some state \( x_{des} \) was originally reachable at time \( t \) with some input \( u(t) \), is it still reachable for some input \( u'(t) \) under these new dynamics? Justify your answer.

Solution.

(a) **Topic:** Setting up I/O systems and basic least-squares

i. No data is needed to determine \( C, D \) in the model:

\[
C = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}, \quad D = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}
\]

For the state equation, we recall that

\[
x(t) = e^{tA}x(0) + \int_0^t e^{(t-\tau)A}Bu(\tau)d\tau.
\]  

(10)

To solve for \( A \), we run tests for a fixed duration \( d \) with no input (\( u = 0 \)). This gives us \( x(d) = e^{dA}x(0) \). If we run a series of tests, all with duration \( d \) and input \( u = 0 \) but with various initial conditions \( x_0 = x(0) \), we can stack our results to arrive at the matrix equation

\[
\begin{bmatrix} x^{(1)}(d) \\ x^{(2)}(d) \\ x^{(3)}(d) \\ \vdots \end{bmatrix} = e^{dA} \begin{bmatrix} x^{(1)}(0) \\ x^{(2)}(0) \\ x^{(3)}(0) \\ \vdots \end{bmatrix}
\]

\[
X(d) = e^{dA}X(0),
\]

\[
22
\]
where \( x^{(i)}(t) \) is the state variable for the \( i \)th test. Rearranging as in HW3Q6, we can now solve for each row of \( e^{dA} \) independently, since

\[
\tilde{x}_i = X(0)^T e_i^{dA},
\]

where \( \tilde{x}_i \) is the transpose of the \( i \)th row of \( X(d) \) and \( e_i^{dA} \) is the transpose of the \( i \)th row of \( e^{dA} \). Finally, we take the matrix logarithm to find \( A \).

To solve for \( B \) once \( A \) is known, we run a new series of tests (again for some fixed duration \( d \), although this duration can differ for that used to solve for \( A \)). However, instead of setting \( u(t) = 0 \ \forall t \), we now set \( u(t) = u \ \forall t \), where \( u \) can differ between tests but is constant within each test. This allows us the to simplify 10 as

\[
x(t) = e^{tA}x(0) + \left( \int_0^t e^{(t-\tau)A}d\tau \right) Bu \\
\left( \int_0^t e^{(t-\tau)A}d\tau \right)^{-1} (x(t) - e^{tA}x(0)) = Bu.
\]

We can now solve for \( B \) in the same manner that we solved for \( e^{dA} \).

---

### ii.

```matlab
clear all; close all;
Q3_data_a;

%% True A,B used to generate data

% A = [0 0 1 0;
%      0 0 0 1;
%      0 0 -0.31 -0.12;
%      0 0 -0.07 -0.24];
%
% B = [0 0;
%      0 0;
%      3.4 0.22;
%      0.18 3.7];

%% Define states

X0 = [p0; v0];
Xd_h = [pd_h; vd_h];
Xd_nh = [pd_nh; vd_nh];

%% Find A

ehat = zeros(4,4);
ehat(1,:) = ((X0.') \ (Xd_h(1,:).')).';
ehat(2,:) = ((X0.') \ (Xd_h(2,:).')).';
ehat(3,:) = ((X0.') \ (Xd_h(3,:).')).';
```
\( \text{ehat}(4,:) = ((X0.') \ (Xd_h(4,:).')).'; \)

\[
A = \logm(\text{ehat})/\text{durr};
\]

\[
\begin{array}{cccc}
0.0003 & -0.0004 & 0.9929 & 0.0034 \\
0.0001 & -0.0000 & -0.0032 & 0.9987 \\
-0.0001 & 0.0001 & -0.3072 & -0.1209 \\
-0.0001 & 0.0000 & -0.0686 & -0.2397 \\
\end{array}
\]

%%% Find B

num_steps = 10000;

t = linspace(0,\text{durr},\text{num_steps});
to_int = zeros(4,4,\text{num_steps});
for i = 1:\text{num_steps}
    to_int(:,:,i) = expm(t(i)*A);
end
int = trapz(t,to_int,3);

b = inv(int) * (Xd_nh - \expm(durr*A) * X0);

B = zeros(4,2);
B(1,:) = ((U.') \ (b(1,:).')).';
B(2,:) = ((U.') \ (b(2,:).')).';
B(3,:) = ((U.') \ (b(3,:).')).';
B(4,:) = ((U.') \ (b(4,:).')).';

\[
\begin{array}{cccc}
0.0597 & -0.0357 & \\
0.0208 & 0.0126 & \\
3.3774 & 0.2299 & \\
0.1712 & 3.6975 & \\
\end{array}
\]

%%% Find C

% This follows directly from the definition of the state

C = [1 0 0 0; 
     0 1 0 0];

(b) **Topic:** Impulse/step responses.
The impulse response are given by

\[
h(t) = Ce^{tA}B + D\delta(t).
\]

```matlab
clear all; close all;
Q3_data_b;
t = linspace(0,10,100);
h1 = zeros(2,length(t));
h2 = zeros(2,length(t));
for i = 1:length(t)
```
\begin{verbatim}
figure(1)
plot(t, h1, t, h2);
xlabel('t', 'interpreter', 'latex')
ttl_txt = 'Impulse response';
title(ttl_txt, 'interpreter', 'latex')
lgd = legend('$u_1, y_1$', '$u_1, y_2$', '$u_2, y_1$', '$u_2, y_2$', 'Location', 'best');
set(lgd, 'Interpreter', 'latex');
set(gca, 'fontsize', 15)
\end{verbatim}

(c) **Topic:** Discritization and mimium energy solutions

i. \[ A_d = e^{hA} \quad B_d = \left( \int_0^h e^{\tau A} d\tau \right) B \quad C_d = C \]

ii. Let \( \kappa_1 = \tau_1/h \) and \( \kappa_3 = \tau_3/h \). Note that, since \( x(0) = 0 \), \( y_d(k) \) is given by the following:

\[
y_d(k) = \begin{bmatrix} CB & CAB & CA^2B & \cdots & CA^{k-1}B \end{bmatrix} \begin{bmatrix} u(k-1) \\ \vdots \\ u(0) \end{bmatrix}
\]
Stacking all elements of our itinerary, we get

\[
\begin{bmatrix}
s_1 \\
s_2 \\
s_3
\end{bmatrix} =
\begin{bmatrix}
0 & \cdots & 0 & CB & \cdots & CA^{\kappa_1-1}B \\
0 & \cdots & CB & \cdots & \cdots & \cdots & CA^{\kappa_2-1}B \\
CB & \cdots & \cdots & \cdots & \cdots & \cdots & CA^{\kappa_3-1}B
\end{bmatrix}
\begin{bmatrix}
u(\kappa_3-1) \\
\vdots \\
u(0)
\end{bmatrix}
\]

Thus our energy minimizing input is simply the least norm solutions. However, notice that in the above, we do not actually know \(\kappa_2\); however, we do know that \(\kappa_1 < \kappa_2 < \kappa_3\). Thus we can iterate over all possible \(\kappa_2\)'s and choose that which allows us the minimum energy input.

```matlab
% ii.
clear all; close all;
Q3_data_b;
Q3_data_c;
x0 = zeros(4,1);

%% Discritize matrices
h = 1;
Ad = expm(h*A);
Cd = C;

tau = linspace(0,h,1000);
e_tauA = zeros(4,4,length(tau));
for i = 1:length(tau)
    e_tauA(:,:,i) = expm(tau(i)*A);
end
Bd = trapz(tau,e_tauA,3)*B;

ar_k = round(ar_t(3,:)/h);
k = zeros(1,size(sites,2)-1);
k([2,3,5,6,7]) = ar_k;

%% Solve system for all possible arrival times
poss_k1 = 0+1:k(2)-1;
poss_k4 = k(3)+1:k(5)-1;
U_search = zeros(2*k(end),length(poss_k1)*length(poss_k4));
E_search = zeros(1,length(poss_k1)*length(poss_k4));
ind = 1;
for k1 = poss_k1
    for k4 = poss_k4
        k(1) = k1; k(4) = k4;
        G = zeros(length(k)*2, 2*k(end));
        H = zeros(length(k)*4, 2*k(end));
        F = zeros(length(k)*2, 4);
        y = zeros(length(k)*2, 1);
        for i = 1:length(k)
            % Find G
            for j = 1:k(i)
                G(2*i-1:2*i,2*j-1:2*j) = C*Ad^(k(i)-j)*Bd;
                H(4*i-3:4*i,2*j-1:2*j) = Ad^(k(i)-j)*Bd;
            end
        end
    end
end
```

26
% Find F
F(2*i-1:2*i, :) = C*Ad^(k(i));
y(2*i-1:2*i) = sites(:,i);
end
U_search(:,ind) = G' * inv(G*G') * (y-F*x0);
E_search(ind) = norm(U_search(:,ind));
ind = ind+1;
end

[E,ind] = min(E_search);
U = U_search(:,ind);
U = reshape(U,2,[]);

%% Simulate trajectory
T=k(end); % final time
X=zeros(4, T+1);
X(:,1)=x0;
for t=1:T
    X(:, t+1)=Ad*X(:,t)+Bd*U(:,t);
end

%% Plot
plot(sites(1,:), sites(2,:), 'ro')
hold on
plot(X(1,:), X(2,:), 'b')

(d) **Topic**: The effects of discretization.
Intuitively, you’d want to be able to define a continuous (i.e., non-discretized) input for minimal energy, as this gives us the most degrees of freedom to minimize energy.

(e) **Topic:** Reachability

We note that in 3c, we only cared about our position coordinates matching those desired. Here, we also must have our velocities in the $x_1$ and $x_2$ directions equal to zero to stop. These values are part of our state, so this becomes a reachability problem for our discretized system. Recall that this system is given by

$$x(t + h) = A_d x(t) + B_d u(t)$$

where

$$A_d = e^{hA}$$
$$B_d = \int_0^h e^{\tau A} d\tau B$$

but you already solved for these in part 3c. The reachable set of states for this discretized system at time $t + hk$ starting from $x(t)$ at time $t$ is given by $\text{range}(C_k)$ where

$$C_k = [\begin{bmatrix} B_d & A_d B_d & \cdots & A_d^{k-1} B_d \end{bmatrix}]$$

We note that $[B_d \ A_d B_d \ \cdots \ A_d^{k-1} B_d]$ has rank $4 = n$ (size of state $x$), so all points are reachable after 2 steps. Thus, the only point in the data of potential concern is

$$x(17) = \begin{bmatrix} 4 \\ 16 \\ 0 \\ 0 \end{bmatrix} \text{ from } x(16) = \begin{bmatrix} -3 \\ -9 \\ 0 \\ 0 \end{bmatrix} \text{ in one discretization step since } h = 1$$

We try to solve

$$x(17) = A x(16) + B_d u$$

and see that it has no exact solutions. Your friend is correct. However, we see that this problem is due to the discretization interval chosen only giving you one chance to adjust the input between $x(16)$ and $x(17)$. A simple modification to avoid is problem is decreasing the discretization interval which will allow us to change the input more than once between these two stops.

(f) **Topic:** Changing Dynamics.

This change actually does not affect the reachable set at time $t$, $R_t$, of the system as long as $f(t) > 0$ for some $\epsilon > t > 0$. (Note that $f$ is monotone decreasing, so once it becomes non-positive it will stay non-positive.) We can put an infinite amount of energy into the system at this instant, which will allow us to get to a
state $x_1$ at time $\epsilon$. This follows from the fact that for a continuous-time LDS, for $t > 0$ we know $\mathcal{R}_t = \mathcal{R} = \text{range}(\mathcal{C})$ where

$$\mathcal{C} = \begin{bmatrix} B & AB & \ldots & A^{n-1}B \end{bmatrix}$$

If we want to get to $x_{\text{des}}$ at time $\tau$, we choose

$$x_1 = \left(e^{(\tau-\epsilon)A}\right)^{-1}x_{\text{des}},$$

as our system will evolve autonomously by $e^{(\tau-\epsilon)A}$ between $\epsilon$ and $\tau$ if no input is given. Now we only need to verify that $f(t)$ is not zero for all $t > 0$. To find when $f(t)$ equals zero, we solve

$$2 - \gamma^{-t+\theta} = 0$$

$$\log 2 = (-t + \theta) \log \gamma$$

$$t = \theta - \frac{\log 2}{\log \gamma}$$

Examining $t$, we note that $\log 2 > 0$ and $\log \gamma < 0$. Thus, $-\frac{\log 2}{\log \gamma}$ is positive. We are given that $\theta$ is positive as well, so $f(t) > 0$ until some time $t > 0$. Thus, the reachable set $\mathcal{R}_t$ does not change.
4. Find the right protein [20 points]

A rigid transformation is a mapping from $\mathbb{R}^n$ to $\mathbb{R}^n$ that is the composition of a translation and a rotation. Mathematically, we can express a rigid transformation $\phi$ as $\phi(x) = Rx + t$, where $R$ is an $n \times n$ orthogonal matrix and $t \in \mathbb{R}^n$ is a vector.

We are given a set of pairs of points $(x_i, y_i)$ in $\mathbb{R}^n$, $i = 1, \ldots, m$, and wish to find a rigid transformation that best maps the $x_i$’s to the $y_i$’s. We can write the problem as:

\[
\begin{align*}
\text{minimize} & \quad J = \sum_{i=1}^{m} \| Rx_i + t - y_i \|_2^2 \\
\text{subject to} & \quad R^\top R = I_n,
\end{align*}
\]

where $I_n$ is the $n \times n$ identity matrix, and $R$ and $t$ are the optimization variables.

This problem arises in image processing, to provide ways to deform an image (represented as a set of two-dimensional points) based on the manual selection of a few points and their transformed counterparts:

(a) Assume that $R$ is fixed and known in (11). Express the optimal $t$ as a function of $R$.

(b) Show that the corresponding optimal value of the objective function $\tilde{J}$ (now a function of $R$ only), can be written as the original objective function $J$, with $t = 0$ and $x_i, y_i$ replaced with their centered counterparts $\bar{x}_i, \bar{y}_i$, respectively:

\[
\tilde{J} = \sum_{i=1}^{m} \| R\bar{x}_i - \bar{y}_i \|_2^2,
\]

where:

\[
\begin{align*}
\bar{x}_i &= x_i - \hat{x}, \quad \hat{x} = \frac{1}{m} \sum_{i=1}^{m} x_i \\
\bar{y}_i &= y_i - \hat{y}, \quad \hat{y} = \frac{1}{m} \sum_{i=1}^{m} y_i.
\end{align*}
\]

(c) Show that the problem of minimizing $\tilde{J}$ can be written as:

\[
\begin{align*}
\text{minimize} & \quad \| RX - Y \|_F^2 \\
\text{subject to} & \quad R^\top R = I_n,
\end{align*}
\]

for appropriate known matrices $X, Y$, which you need to determine explicitly.

**Hint**: Refer to homework 7 for the definition of the Frobenius norm of a matrix.
(d) Show that the problem can be further written as:

$$\begin{align*}
\text{maximize} & \quad \text{trace } RZ \\
\text{subject to} & \quad R^\top R = I_n,
\end{align*}$$

for an appropriate known $n \times n$ matrix $Z$, which you need to determine explicitly.

**Hint:** Remember that trace $AB = \text{trace } BA$ for any $A, B \in \mathbb{R}^{n \times n}$.

(e) Show that $R = VU^\top$ is optimal for (13), where $Z = U\Sigma V^\top$ is the full SVD of $Z$.

**Hint:** Using a change of variables, show that any problem can be reduced to an equivalent problem:

$$\begin{align*}
\text{maximize} & \quad \text{trace } MS \\
\text{subject to} & \quad M^\top M = I_n,
\end{align*}$$

where $S$ is a known diagonal matrix with non-negative entries. Then use without proof the fact that $M = I_n$ is optimal for this problem.

(f) Show the fact you used in the previous part: assume $Z$ is diagonal with non-negative entries, and show that $R = I_n$ is optimal for (13).

**Hint:** Show that $R^\top R = I_n$ implies $|R_{ii}| \leq 1$, $i = 1, \ldots, n$, and using that fact, prove that the optimal value is less than or equal to trace $Z$.

(g) Structural biologists have successfully utilized 3D protein structure alignment to study the functional and evolutionary relationships between proteins. Figure 2 shows such a structural alignment between the muscarinic acetylcholine receptors $M_2$ and $M_3$, which are important drug targets for airway and central nervous system diseases.

![Figure 2: Protein alignment](image)

A common metric for the similarity between two protein structures is the root-mean-square deviation, or RMSD of atomic positions in the two proteins. If we assume that both proteins have $m$ atoms, and the position of $i^{th}$ atom in protein 1 is denoted $x_i \in \mathbb{R}^3$ and the position of $i^{th}$ atom in protein 2 is denoted $y_i \in \mathbb{R}^3$, ...
the RMSD can be expressed as:

\[
\text{RMSD} = \left( \frac{1}{m} \sum_{i=1}^{m} \|x_i - y_i\|_2^2 \right)^{\frac{1}{2}}.
\] (14)

You are working in lab one day when you are given samples of 2 new muscarinic receptor proteins Z and W. After some testing you find out that both of these proteins have chemical compositions very similar to that of M_2. Later that day, one of your lab mates tells you that one of the proteins Z and W is in fact the same as M_2 but she doesn’t know which one and asks you to help her identify it. Since chemical testing was inconclusive, you decide to have another lab measure the atomic positions of Z and W for you and then find their RMSD relative to M_2. The sample with an RMSD of less than 1 would be the same as M_2 with very high probability. The test results are emailed to you the next day in a data file receptor_coordinates_data, which includes matrices for the coordinates of M_2, Z, and W. The \(i\)th row of each matrix represents a vector in \(\mathbb{R}^3\) corresponding to the location of the atom \(i\) in the structure. Before you compute the RMSD of Z and W relative to M_2, you remember that the proteins might have rotated to any random orientation and also shifted to any random position relative to the origin during the measurements. Therefore, you will have to align the atomic positions of the proteins the best you can before calculating the minimum RMSDs.

Based on the results of parts (a)-(f) briefly explain how to calculate the correct RMSD of Z and W relative to M_2. Then carry out your method on the problem data and report the two RMSD values. Identify which protein is most likely the same as M_2 and provide a 3D plot of the aligned atomic positions of M_2 and this protein.

**Hint:** You can use the `plot3` function in Matlab or its equivalent in other programming languages to generate a 3D plot of the protein structure.

**Solution.**

(a) For fixed \(R\), denoting by \(z_i = y_i - Rx_i\), \(i = 1, \ldots, m\), we obtain the problem in variable \(t\):

\[
\text{minimize} \quad \sum_{i=1}^{m} \|t - z_i\|_2^2.
\] (15)

The solution to this least-squares problem can be obtained either by setting the derivative of the objective function with respect to \(t\) to zero, or treating it as a multi-objective least squares problem. The optimal \(t\) would be given by:

\[
t^* = \hat{z} = \frac{1}{m} \sum_{i=1}^{m} z_i = \hat{y} - R\hat{x}.
\] (16)
(b) The corresponding value of the objective function is:

\[ \tilde{J} = \sum_{i=1}^{m} \| z_i - \hat{z} \|^2_2 = \sum_{i=1}^{m} \| y_i - \hat{y} - R(x_i - \hat{x}) \|^2_2 = \sum_{i=1}^{m} \| R\bar{x}_i - \bar{y}_i \|^2_2. \]  

(17)

(c) We consider the problem:

\[
\begin{align*}
\text{minimize} \quad & \ J = \sum_{i=1}^{m} \| R\bar{x}_i - \bar{y}_i \|^2_2 \\
\text{subject to} \quad & \ R^\top R = I_n.
\end{align*}
\]

We can express the objective function as stated with:

\[ X = [\bar{x}_1, \ldots, \bar{x}_m] \in \mathbb{R}^{n \times m}, \quad Y = [\bar{y}_1, \ldots, \bar{y}_m] \in \mathbb{R}^{n \times m}. \]  

(19)

Note that this makes the columns of the matrix \( RX - Y \) equal to \( R\bar{x}_i - \bar{y}_i \), \( i = 1, \ldots, m \).

(d) We have, for any \( R \) with \( R^\top R = I_n \):

\[
\| RX - Y \|^2_F = \text{trace} \ (RX - Y)^\top (RX - Y) = \text{trace} \ X^\top RX - 2\text{trace} \ Y^\top RX + \text{trace} \ Y^\top Y = c - 2\text{trace} \ Y^\top RX,
\]

where \( c = \text{trace} \ X^\top X + \text{trace} \ Y^\top Y \) is a constant. Note that \( \text{trace} \ Y^\top RX = \text{trace} \ RXY^\top \). Hence, the problem can be expressed as claimed, with \( Z = XY^\top \in \mathbb{R}^{n \times n} \).

(e) Let \( Z = U\Sigma V^\top \) be the SVD of \( Z \). Then we get:

\[
\text{trace} \ RZ = \text{trace} \ R(U\Sigma V^\top) = \text{trace} \ (V^\top RU)\Sigma = \text{trace} \ M\Sigma,
\]

where \( M \) is the new variable \( M = V^\top RU \). We note that \( R^\top R = I_n \) translates to \( M^\top M = I_n \). We have reduced the problem to the case where \( Z \) is replaced with the diagonal matrix \( \Sigma \):

\[
\begin{align*}
\text{maximize} \quad & \ \text{trace} \ M\Sigma \\
\text{subject to} \quad & \ M^\top M = I_n.
\end{align*}
\]

(22)

Using the fact that \( M = I_n \) is optimal for (22), we obtain \( V^\top RU = I_n \) is optimal for (13), which leads to the desired result.

(f) The fact that \( M = I_n \) is optimal for the above problem stems from the fact that since \( M \) is orthogonal, \( |M_{ii}| \leq 1, \ i = 1, \ldots, n \). In fact, defining \( e_i \) to be the \( i \)th unit vector in \( \mathbb{R}^n \) we get:

\[ 1 = (M^\top M)_{ii} = e_i^\top M^\top Me_i = \| Me_i \|^2_2 = \sum_{j=1}^{n} M^2_{ji} \geq M^2_{ii}. \]  

(23)
Thus, with \( r \leq m \) being the ranks of \( Z \), and \( \sigma_1, \ldots, \sigma_r \) its singular values we get:

\[
\text{trace } M \Sigma = \sum_{i=1}^{r} M_{ii} \sigma_i \leq \sum_{i=1}^{r} \sigma_i. \tag{24}
\]

The result follows from the fact that the upper bound is attained when \( M = I_n \).

Using the results of parts (a)-(f) we can find the best rigid transformations that map the atomic positions of \( Z \) and \( W \) to that of \( M_2 \). We then apply the transformations to the atomic positions to align the proteins and use the aligned positions to calculate the RMSDs. The following Matlab code finds the best RMSD of \( Z \) and \( W \) relative to \( M_2 \):

\[
\text{RMSD}_Z = 0.8537, \quad \text{RMSD}_W = 5.7862, \tag{25}
\]

Therefore, protein \( Z \) is the same as \( M_2 \).

```matlab
clear all
close all
clc

receptor_coordinates_data % Load the data file

x = Z';
y = M2';

x_hat = mean(x,2);
y_hat = mean(y,2);

x_bar = x - x_hat;
y_bar = y - y_hat;

Z = x_bar*y_bar';
[U S V] = svd(Z);
R = V*U';
t = y_hat - R*x_hat;

RMSD_Z = sqrt((1/length(M2))*sum(sum((R*x+t-y).^2)))

figure
plot3(M2(:,1),M2(:,2),M2(:,3),'linewidth',2)
axis image
hold on
Z_aligned = (R*x+t)';
plot3(Z_aligned(:,1),Z_aligned(:,2),Z_aligned(:,3),'linewidth',2)
grid on
xlabel('x','fontsize',14)
ylabel('y','fontsize',14)
zlabel('z','fontsize',14)
legend('M2','Z')
set(gca,'fontsize',14)
```
\[ x = W'; \]
\[ y = M2'; \]

\[ x_{\text{hat}} = \text{mean}(x,2); \]
\[ y_{\text{hat}} = \text{mean}(y,2); \]

\[ x_{\text{bar}} = x - x_{\text{hat}}; \]
\[ y_{\text{bar}} = y - y_{\text{hat}}; \]

\[ Z = x_{\text{bar}}*y_{\text{bar}}'; \]
\[ [U \ S \ V] = \text{svd}(Z); \]
\[ R = V*U'; \]
\[ t = y_{\text{hat}} - R*x_{\text{hat}}; \]

\[ \text{RMSD}_W = \sqrt{\frac{1}{\text{length}(M2)}*\text{sum}(\text{sum}((R*x+t-y).^2))} \]