Lab 3 - Descrete Dynamical Systems abd Markov Models - New

1 Discrete (Time) Linear Dynamical Systems

Dynamical systems are mathematical models of phenomena that change with time. Through algebra, calculus and differential equations you are probably familiar with the many models wherein we assume that time is a continuous (or smooth) variable t. However, there are many times that we want to assume that changes to the system with time only happen at certain discrete steps; this could be for computational reasons like a numerical approximation of a continuous system, or because the phenomena we are modeling is actually discrete in nature.

In this lab, we will use linear algebra to explore models of discrete (time) linear dynamical systems (DLDS). Note that the time is often suppressed, although there are other ways that a linear system could be discrete. We will explore the possible dynamics of the discrete systems used in population modeling and develop some programmatic and graphical tools to better understand and explore dynamical systems.

This lab is about exploring linear systems for yourself and discovering what kind of behavior they can exhibit, and how they are used. Along the way, we will develop some tools to effectively visualize discrete linear systems. You should take the **Questions** and **Exercises** seriously, as they will help you build to answering the questions.

This lab should take approximately 2-4 hours.

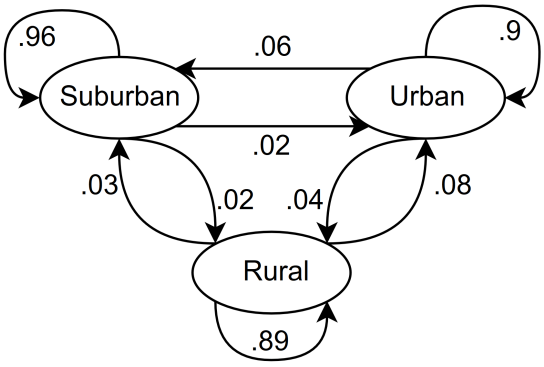
1.1 A Warm-Up with Population Dynamics

In this lab, we will get some hands on experience with discrete time dynamical systems. A **discrete time dynamical system** models a changing system when time is broken up into finite chunks $t_0, t_1, \ldots$. We start with a standard example of population dynamics.

Consider for simplicity a closed system in a major metropolitan area. Assume there are three populations: the rural population R, the suburban population S and the urban population U. Each year, a proportion of each population moves between each zone:

- 8% of the **rural** population moves to the urban center, and 3% moves to a suburban center.
- 6% of the **urban** population moves to a suburban center and 4% moves to a rural center.
- 2% of the **suburban** population moves to a rural center and 2% move to an urban center.

We can represent the following data by a **weighted directed graph**



Each arrow (**edge**) of the graph denotes the percentage (as a decimal) of the population that moves from one state to another. Note that often the self referential arrows are not drawn since the static population is implied from the portions of the population that are leaving. Given such a model, there are a few questions we can ask:

- Given some initial distribution of population R_0, U_0, S_0, what will the population of each region be after 1 years? After 10 years?
- What about in the long term? Does the entire population end up in one state? Does the initial population matter?
- How can we model these dynamics efficiently?

Lets start with the first question: Given initial regional populations of R_0, U_0 and S_0, we can compute the population after a year as

$$\begin{aligned} R_1 &= .89R_0 + .04U_0 + .02S_0 \\ U_1 &= .08R_0 + .9U_0 + .02S_0 \\ S_1 &= .03R_0 + .06U_0 + .96S_0 \end{aligned}$$

Repeating, we can compute the population in 2 years from (R_1, U_1, S_1) the same way

$$\begin{aligned} R_2 &= .89R_1 + .04U_1 + .02S_1 \\ U_2 &= .08R_1 + .9U_1 + .02S_1 \\ S_2 &= .03R_1 + .06U_1 + .96S_1 \end{aligned}$$

and in general we can compute the populations at year $i+1$ from year i as

$$\begin{aligned} R_{i+1} &= .89R_i + .04U_i + .02S_i \\ U_{i+1} &= .08R_i + .9U_i + .02S_i \\ S_{i+1} &= .03R_i + .06U_i + .96S_i \end{aligned}$$

It is clear that while the computations are doable they will rapidly get out of hand. However, matrix algebra gives us a much easier way to write this system:

$$\begin{bmatrix} R_{i+1} \\ U_{i+1} \\ S_{i+1} \end{bmatrix} = \begin{pmatrix} .89 & .04 & .02 \\ .08 & .9 & .02 \\ .03 & .06 & .96 \end{pmatrix} \begin{bmatrix} R_i \\ U_i \\ S_i \end{bmatrix}.$$

Let

$$A = \begin{pmatrix} .89 & .04 & .02 \\ .08 & .9 & .02 \\ .03 & .06 & .96 \end{pmatrix} \text{ and } P_i = \begin{bmatrix} R_i \\ U_i \\ S_i \end{bmatrix} .$$

We can then write the population after 2 years as

$$P_2 = AP_1 = A(AP_0) = A^2P_0 .$$

In general, we can write the population at time i as A^iP_0 and use theory, or computational tools like MATLAB, to solve the problem efficiently.

1.1.1 Exercise:

(Do not turn in)

For example, in the year 2000 for a small midwestern city with 1.2 million people in the region there are $U_0 = 360,000$ people in the urban area, $S_0 = 530,000$ people in the suburban area and $R_0 = 310,000$ in the rural area.

- What is the population after 1 year?
- What is the population after 10 years?
- What is the population as we let the number of years go to ∞?
- What if we start with a different population in the rural, urban and suburban areas? Can you find a rule for what the final population distribution will be?

1.2 Exploring Discrete Predator-Prey Models Graphically:

The pacific tree frog prey primarily on beetles, spiders, and flies. In a simple lakeside ecosystem, beetles with no natural predators will grow proportional to their current population. Let $B^{(n)}$ be the population of beetles at year n. If every adult produces α_B offspring yearly and each year β_B adults die naturally, the year to year beetle population is given by

$$B^{(n+1)} = (1 + \alpha_B - \beta_B)B^{(n)} .$$

Note that if $\alpha_B > \beta_B$, the beetle population increases each year, while if $\alpha_B < \beta_B$ the population decreases each year. In general, we will assume $\alpha_B > \beta_B$.

Lets assume the survival pressure on frogs is limited to their ability to find food, that is a practically infinite number of frogs are born as tadpoles but only the small number able to eat sufficiently survive to adulthood. In addition, let us assume that the beetles are evenly distributed across the pond, and that the chance of survival to adulthood is directly proportional by ρ_F to the number of beetles present (that is, if there twice as many beetles, twice as many frogs will reach adulthood). Letting β_F be the natural death rate of frogs, the population at year n is given by

$$F^{(n+1)} = \rho_B B^{(n)} + (1 - \beta_F)F^{(n)} .$$

Assuming each frog eats γ_B beetles, the year to year beetle population in the presence of a predator is

$$B^{(n+1)} = (1 + \alpha_B - \beta_B)B^{(n)} - \gamma_F F^{(n)} .$$

Combining these equations, we can write the vectors of populations as a discrete dynamical system

$$\begin{bmatrix} B^{(n+1)} \\ F^{(n+1)} \end{bmatrix} = \begin{bmatrix} 1+\alpha_B-\beta_B & -\gamma_F \\ \rho_B & (1-\beta_F) \end{bmatrix} \begin{bmatrix} B^{(n)} \\ F^{(n)} \end{bmatrix}$$

Example 1: Assume $B^{(n)}$ is in millions of beetles, while F is in hundreds of frogs. Without a predator, the birth rate of beetles is $\alpha_B = 1.3$ and the death rate is $\beta_B = .85$. Each adult frog must eat 350 beetles yearly, so in our units $\gamma_F = .035$. The natural frog death rate is $\beta_F = .25$ frogs per year and it is estimated for each increase of 13,000 beetles the number of frogs increases by 1, so $\rho_B = .013$. This yields the following equations:

$$\begin{bmatrix} B^{(n+1)} \\ F^{(n+1)} \end{bmatrix} = \begin{bmatrix} 1.45 & -.035 \\ .013 & .75 \end{bmatrix} \begin{bmatrix} B^{(n)} \\ F^{(n)} \end{bmatrix}$$

Think about the following questions, and use MATLAB to explore them a bit yourself. We will work towards answering them in the next section.

Question: Assume that initially there are 1,000,000 beetles and 3000 frogs. What is the long term behavior of the populations, as in do the frogs die out? Do the beetles die out? Does the frogs population experience unbounded growth? What about the beetles?

Question: Assume initially there are 1,000,000 beetles. Find the approximate maximum initial frog population that leads to the survival of the frogs.

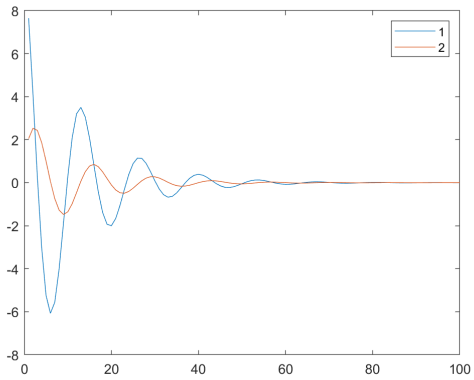
Example 2: Changing the parameters, what if we have observed the following discrete differential equation

$$\begin{bmatrix} B^{(n+1)} \\ F^{(n+1)} \end{bmatrix} = \begin{bmatrix} .9 & -1.35 \\ .13 & .75 \end{bmatrix} \begin{bmatrix} B^{(n)} \\ F^{(n)} \end{bmatrix}$$

If there are initially 10 million beetles and 100 frogs what happens to the population over time? How can we explain these dynamics in terms of the matrix?

1.2.1 Visualizing Discrete Dynamical Systems

We want to write a function that visualizes the progression of a discrete dynamical system. As a first pass, lets assume that all populations are of similar scales and try to plot them on the same graph as below



Input: To simulate a discrete dynamical system of the form $P^{(n+1)} = AP^{(n)}$, the function should take as an input the $k \times k$ matrix A, the initial population vector $P^{(0)}$ of length k, and the number of iterations to simulate N.

Output: A $k \times N$ array containing the populations at each time $n = 1, \ldots, N$. The function should also graph all of these plots on the same axis.

Complete the code below to construct the function:

```
function Y = plotmodel(A,P,N)
    k =                         % Set k to the length of P
    Y =                         % Set Y to a k by N array of zeros

    Y(:,1) = P;                 % Set the first column of Y to be the initial vector P
    for i=2:N
        Y(:,i) =                % Compute the population after i steps and put it in i+1'th column
    end

    % Now that we've computed the populations at each time step we should graph them

    for i = 1:k
        plot(Y(i,:),'DisplayName',"Population "+num2str(i))
        hold on
    end
    legend
    hold off
end
```

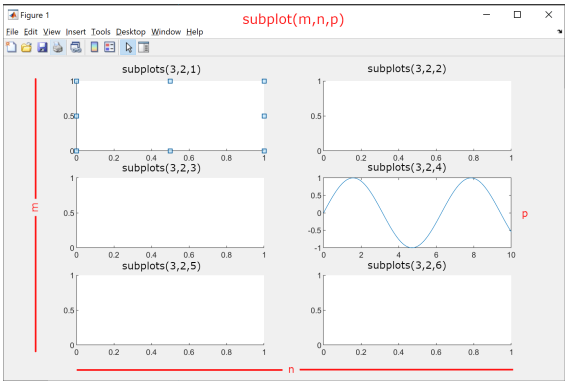
You should also add axis labels.

Exercise: Use the `plotmodel` function to plot the first 20 years of the predator prey model and answer the questions above.

Better Visualization You may have noticed when exploring this system of equations that as soon as the relative scale between populations becomes different the visualization is dominated

by the larger quantity, sometime to the extent of obscuring important information. For dynamical systems modeling variables of different scales it can be useful to graph the solutions on different axes, using MATLAB's `subplots` function.

- `subplots(m,n,p)` - Calling `subplots` tells MATLAB that any subsequent plot calls should draw to a subplot, not the whole axis. Specifically, it says that for a m by n grid of subplots, plot calls should plot to p'th plot as shown below:



For example, to plot to the **axis** in the second column and second row as above, we would use

```
t = 0:.01:10
subplot(3,2,4)
plot(t,sin(t))
```

Example: What does the code below produce?

```
t = 0:.01:10
subplot(3,1,1)
plot(t,sin(t))

subplot(3,1,2)
plot(t,t.^2)

subplot(3,1,3)
plot(t,t)
```

Exercise: Modify the `plotmodel` function from before to plot each population to a different subplot. It will probably be easiest to list the subplots vertically. Call this new function `plotmodelsub`.

1.2.2 Plotting For Long Term Behavior

Plotting to subplots is certainly much cleaner, but it also has a problem: For a large number of populations, the subplots quickly become so small they don't help us understand anything. For a large number of populations, we may only care about the long term behavior, that is which populations explode, which die off, and which if any stabilize. When looking at the long term, we don't need to include a scale to infer the behavior and can efficiently plot each graph to the same axis as below.

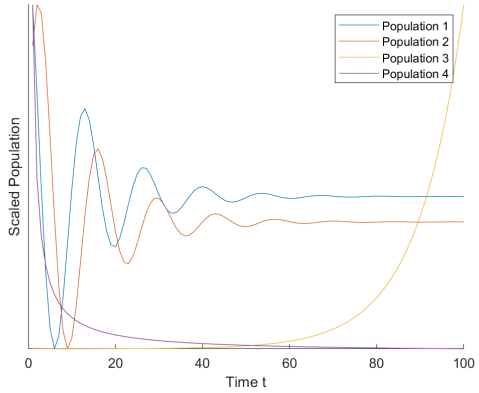
To do this we need to put all of the plots on the same scale, say between 0 and 1. The easiest way to do this is to subtract off the minimum and then scale by the range:

$$Y_{scaled} = \frac{Y - Y_{min}}{Y_{max} - Y_{min}} \subset [0,1]\,.$$

The MATLAB's `max` and `min` functions give the maximum and minimum of an array `Y` so we can plot an array of values between 0 and 1 via

```
YS = (Y - min(Y))/(max(Y) - min(Y))
```

Exercise: Modify the code to `plotmodel` to plot all of the populations on the same axis, scaling so that each population is represented as being between 0 and 1. For completeness you should use `set(gca,'YTick',[])` to remove the Y scale, since in this case the y values are unscaled as below.



1.2.3 Phase Plot Trajectories

In the special case of a two dimensional discrete dynamical system there is a final way we can view the relative population sizes and that is to plot the evolution of the populations $\big(P_1(t), P_2(t)\big)$ as points in the **phase space**, or the P_1 vs P_2 plane. This will give us another view of how population P_1 depends on both time, and population P_2.

Plotting the phase space trajectories is relatively simple once we've computed the long term behavior. In fact the only modification to the code is to change

```
function Y = plottraj(A,P,N)
    k =                         % Set k to the length of P
    Y =                         % Set Y to a k by N array of zeros

    Y(:,1) = P;                 % Set the first column of Y to be the initial vector P
    for i=2:N
        Y(:,i) =                % Compute the population after i steps and put it in i+1'th column
    end

    % Now that we've computed the populations at each time step we should graph them

    figure
```

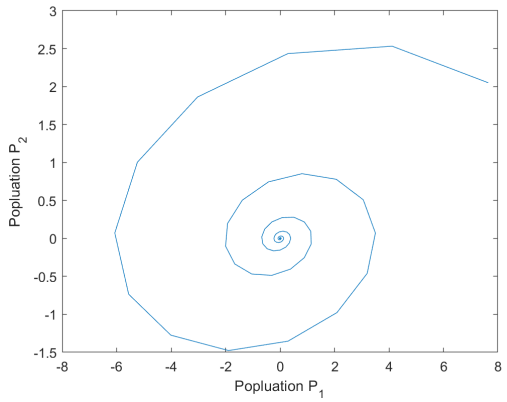
```
    plot(Y(1,:),Y(2,:))
end
```

Using the parameters from **Example 2** we

```
A = [.9,-1.35;
     .13, .75];

y = plottraj(A,[10;1],100);
```

we observe that the trajectory in the phase space is a spiral.



However, this plot doesn't tell us if the spiral is increasing or decreasing. We want to add in arrows that show directionality. MATLAB's way of doing this is with a **quiver plot**.

- `quiver(x,y,u,v)` - Plot at the points with (x, y) coordinates given by `x` and `y` an arrow pointing to $(x + u, y + v)$, that is an arrow with components u,v.

We want to attach an arrow to each point $(P_1(n+1), P_2(n+1))$ pointing towards $(P_1(n+1), P_2(n+1))$. To do this, we must construct

- Vectors `x` and `y` containing the coordinates of the first $N-1$ points since we don't strictly know where to point the N'th vector.
- Vectors `u` and `v` pointing from the n'th point to the $n+1$'st point, that is such that $(u(n), v(n)) = (x(n+1) - x(n), y(n+1) - y(n))$ for $n = 1, \ldots, N-1$.

Quiver will automatically scale the arrows to avoid overlaps, so that will tend to not point the whole way from dot to dot.

We also want to plot dots instead of lines. We could use `plot` with the `'o'` option, but it's easier and cleaner to make scatter plots using the `scatter(x,y)` function. We will also specify `'filled'` to indicate the points should be filled in.

```
figure
scatter(Y(1,:),Y(2,:),'filled')
hold on
```

```
x = Y(1,1:N-1);
y = Y(2,1:N-1);
u = Y(1,2:N) - Y(1,1:N-1);
v = Y(2,2:N) - Y(2,1:N-1);

quiver(x,y,u,v)

hold off
```

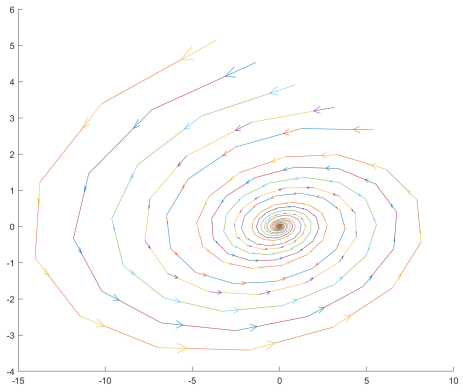
Add the code above to your `plotphase(A,P,N)` function.

Question: What is the long term behavior of the *system* $P^{(n)} = AP^{(n-1)}$? Does this behavior depend on the initial conditions $P^{(0)}$?

Question: In the model neither population can become negative. We assume that if a population falls below 0 we set it to 0 for all future steps n. Under this assumption, what happens to the *model population* in the long term

1.3 Full Phase Plot

How do the initial conditions $P^{(0)}$ change the long term behavior in a discrete linear system? We could plot a large number of trajectories using a `for` loop (and I would encourage you to try):



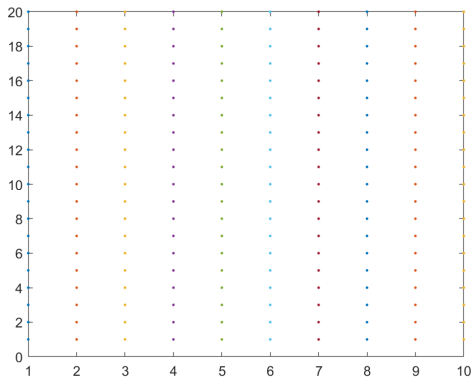
There is another visualization we can produce: Since each update step is *local* (Eg. At each step, we know about the current population, not the population 100 steps ago) if we just make a diagram of all of the local steps on a grid we can trace out *any* trajectory visually, giving us a holistic idea of the space of all trajectories.

First, we want to generate the x and y coordinates for a grid. It's a nice exercise to do this yourself, but MATLAB also include the function `meshgrid` to do it for us:

- `[X,Y] = meshgrid(x,y)` - Given a range of x values $x = [x_1, ..., x_k]$ and a range of y values $y = [y_1, ..., y_\ell]$, Meshgrid generates the x and y coordinates of a grid (x_i, y_j), $i = 1, \ldots, k$, $j = 1, \ldots, \ell$. It then returns an ordered pair, containing the x and y coordinates which we save in vectors `X` and `Y`.

For example, the following code produces a grid and prints it:

```
x = 1:10
x = 1:20
[X, Y] = meshgrid(x,x)
plot(X,Y,'.')
```



Given a grid as a list of x coordinates and a list of y coordinates at time $n = 0$, we want to produce the x and y coordinates at time $n = 1$. We could easily loop through the lists like so

```
for i = 1:length(X)
    P = [X(i);Y(i)]
    P_new = A*P
```

adding a line to perform the quiver plot at each step, but this is a bad solution for several reasons: First, it's much more efficient to produce all of the vectors in one go and then plot them. Second, it's intellectually and mathematically cleaner to state the problem as a single mathematical operation than as a repeated process whose effect can be hard to trace. Third, MATLAB is built from the ground up process matrix multiplication, it is often much faster to use it than to use a loop. All in all, these reasons combine to the all important golden rule of MATLAB:

- **The Golden Rule Of MATLAB:** Never use a loop when you could use a matrix multiplication.

This is related to my own silver rule of MATLAB

- **The Silver Rule Of MATLAB:** If you're using a loop, unless you are producing an animination or tuning hyperparameters, something has gone horribly wrong.

So how can we calculate $P^{(1)}$ from $P^{(0)}$ for all of the grid elements contained in `X` and `Y` using matrix multiplication? Take a moment to think on this before looking at the answer.

Solution: For vectors `x` of length k and `y` of length `\ell`, each of the vectors `X` and `Y` have length $k \cdot \ell$. We can form the $2 \times (k \cdot \ell)$ matrix of coordiantes

$$XY^{(0)} = \begin{bmatrix} x_1 & x_2 & \dots \\ y_1 & y_2 & \dots \end{bmatrix}$$

and calculate all of the new coordiantes with one multiplication using $XY^{(1)} = AXY^{(0)}$. Since the `X` and `Y` produce by `meshgrid` are matricies we have to flatten them using the `reshape` function:

- `B = reshape(A,m_1,m_2,m_3,...,m_l)` - Takes an array of dimension $n_1, n_2, \ldots, n_k$ and reshapes it to be of dimension $m_1, m_2, m_3, ..., m_\ell$, provided $m_1 \times m_2 \times ... \times m_\ell = n_1 \times n_2 \times ... \times n_k$.

In this case we just want to flatten a vector, so we will be reshaping it a row vector with length to the number of elements in the array using `reshape(X,1,numel(X))`.

Note: There is an even slicker way to do this, using the fact that `X(:)` returns a column vector of all the elements we could use `X(:).'` to then transpose it to a row vector. However reshape is a good function to know about in genearl.

Finally, let produce the matrix and the quiver plot:

```
function Y = plotphase(A,x_range,y_range)
    [X,Y] = meshgrid(x_range,y_range);     % Produce a grid for n=0 values
    X = reshape(X,1,numel(X))              % Flatten X to column vector
    Y = reshape(Y,1,numel(Y))
    XY = [X;Y];                            % Produce 2xnumel(X) matrix

    XY_new = A*XY;                         % Compute values at n=1

    UV = XY_new - XY;                      % Compute difference betwen x and y values
    plot(X,Y,'.')
    hold on
    quiver(X,Y,UV(1,:),UV(2,:))
    hold off
end
```

1.4 Problem:

You have produce 4 functions, `plotmodel`, `plotmodelsub`, `plottraj` and `plotphase`. Using these for functions, do your best to answer the following question:

Find values of α, β, γ and ρ such that the populations

1. Both die out.
2. Both grow exponentially.
3. The beetles grow exponentially, but the frogs die out.
4. The frogs grow exponentially but the beetles die out. Explain why this set of values is not in the *model domain* even though it is mathematically well defined.

Turn in: The graphs and parameter values for 1-4. You should also turn in an explanation for 4.