Questions may continue on the back. Please write clearly. What I cannot read, I will not grade. Typed homework is preferable. A good compromise is to type the words and write the math by hand.

1. This problem is partly a way to explore state trajectories, and partly a programming exercise. Do your work next to a computer running Matlab.

As usual, programming tips are typeset in a different font and with a wider margin, as for this paragraph, so expert programmers can skip this text. However, make sure you read all other text carefully. Short, regular homework sentences are often sandwiched between long blocks of programming tips.

Write a Matlab function with the following header

```matlab
function trajectory(F, x0, N, fig)
```

that plots in figure `fig` the first `N` steps, `n = 0, . . . , N − 1`, of the free evolution of the dynamic system

\[
\begin{align*}
x(0) &= x_0 \\
x(n + 1) &= Fx(n)
\end{align*}
\]

starting from state `x0`. Your code should check that the input argument `F` is a 2 × 2 matrix, and abort execution with an error message otherwise. Also check that `x0` is a column vector with two entries.

To check that a matrix `A` is, say, 2 × 3 you put an `if` statement in your code. Every `if` must have a matching `end` keyword after the code that is to be executed if the clause in the `if` is true:

```matlab
if any(size(A) ˜= [2 3])
    error('A must be 2 by 3')
end
```

The clause is anything after the `if` keyword and before a new line, a comma, or a semicolon. So in the code above the clause is

```matlab
any(size(A) ˜= [2 3])
```

Let us analyze this clause. The Matlab function `size` takes a matrix, in this case `A`, and returns its size, that is, the number of rows and columns of `A`, in a 1 × 2 matrix. As an example, say that the matrix `A` you pass when you call your function is 3 × 3. Then `size` returns the row vector `[3 3]` (try this in Matlab, with some random matrices you create). The string `˜=` means "not equal to". The result of comparing two matrices of equal size to each other with the `˜=` operator is another matrix of the same size. Each entry of this result is 1 if the result of applying the operator to the corresponding entries in the two input vectors results in a true statement, and 0 otherwise. In the example, the comparison

```matlab
[3 3] ˜= [2 3]
```

results into the vector `[1 0]`, because 3 (the first entry of `[3 3]`) is not equal to 2 (the first entry of `[2 3]`), hence the 1 in the result (the "not equal" is true), but 3 (the second entry of `[3 3]`) is equal to 3 (the second entry of `[2 3]`), hence the 0 in the result. Again, try to type

```matlab
[3 3] ˜= [2 3]
```

directly to the Matlab prompt, followed by the return key, and check that you get `[1 0]`, and that you understand why. Make this into a habit: try everything in Matlab. This is perhaps the greatest advantage of using an interpreted language like Matlab, rather than a compiled one, like, say, `C`: in an interpreted language, you can type something to the interpreter and get the result back immediately.

The Matlab function `any` then takes the vector `[1 0]` resulting from the comparison and returns 1 if any entry of the vector is nonzero, that is, if any of the comparisons is true. Otherwise, `any` returns 0 (so it returns 0 if and only if all entries of its argument are equal to 0). In summary the `if` clause
any(size(A) ~= [2 3])

returns 1 if and only if at least one of the dimensions of the matrix A is wrong. In that case, the code of the if statement, that is, anything between the clause and the matching end is executed.

The Matlab function error then prints whatever message you specified in its argument (you need the parentheses and the single quotes as above) and aborts execution of the function.

If the argument fig is omitted when the function trajectory is called, then your code should use the default value of 1 for fig.

Matlab has a built-in variable nargin that contains the number of input arguments to the current function, that is, the number of arguments with which the function is being called. For instance, if you call

```
trajectory([1 0; 0 1], [2; 1])
```

then Matlab does not complain about the fact that your definition of trajectory expects arguments rather than two. Inside the function, the variable nargin is automatically set to be equal to 2. So you need to check the value of this variable in an if statement in your function definition, and set the value of fig if needed.

Your plot should have a dot at every state point \(x(n)\) on the trajectory, and a line segment connecting consecutive points. This will give you plots in the style of Figure 2 in Chapter 3 of the supplementary class notes. However, do not put numbers next to the dots, and there is no need to display the state matrix.

For efficiency, pre-allocate a \(2 \times N\) array of zeros with the Matlab function zeros, to contain the entire trajectory. Let's say that the array is called \(x\). The first column of \(x\) will eventually contain the two values in \(x(0)\), the second will contain those in \(x(1)\), and so forth. This array is filled inside a for loop

```
for n = 1:N
    % Your code here
end
```

that fills column \(x(:, n)\) of \(x\) appropriately. Make sure that \(x(:, 1)\) is also given a proper value somewhere. After the loop, plot the second row of \(x\), that is, \(x(2, :)\) versus the first row \(x(1, :)\) with the Matlab plot function. Do help plot to find out the details. Plot this function twice: one with dots (third argument to plot set to ‘.’), and one with line segments (third argument set to ‘-’). To prevent Matlab from erasing the first plot as it draws the second, insert the instruction

```
hold on
```

between the two plot instructions. You should not assume that the figure in which you plot is clean. For instance, it may have the hold switch turned on from a previous run of your code. To reset the figure before you plot on it, use the Matlab clf command.

Draw two solid line segments to show the two coordinate axes.

To plot these two line segments, you need to know the minimum and maximum \(x_1\) and \(x_2\) coordinates, so you know how long to make the segments. After plotting the state trajectory, you can store into a variable, say, xlim, the smallest and greatest horizontal coordinates of the plot as follows:

```
xlim = get(gca, ’XLim’);
```

Of course, there is also a ’YLim’. The built-in variable gca points to the current graphic axes. If you just type get(gca), you can see the list of property-value pairs associated to the current axes (try this). Any value in this list can be read individually by a get instruction, and written to by a set instruction. Axes are contained in a figure (the window in which the plot appears), and the current figure’s property-value pairs can be accessed by get(gcf) and set(gcf, ...) as well (try!).
Hand in your code and four plots, one for each of the following sets of arguments:

\[
F = \begin{bmatrix}
0.1 & 0.4 \\
-1.2 & 1.5
\end{bmatrix}, \quad x_0 = \begin{bmatrix}
1 \\
0.5
\end{bmatrix}, \quad N = 100
\]

\[
F = \begin{bmatrix}
0.9 & 0 \\
0 & -0.95
\end{bmatrix}, \quad x_0 = \begin{bmatrix}
1 \\
1
\end{bmatrix}, \quad N = 100
\]

\[
F = \begin{bmatrix}
0.99 & -0.1 \\
0.1 & 0.99
\end{bmatrix}, \quad x_0 = \begin{bmatrix}
-0.5 \\
-0.5
\end{bmatrix}, \quad N = 300
\]

\[
F = \begin{bmatrix}
1 & 0 \\
0 & -1
\end{bmatrix}, \quad x_0 = \begin{bmatrix}
1 \\
1
\end{bmatrix}, \quad N = 100
\]

Use the Matlab title command to specify which plot is which.

To build a title string automatically, use the sprintf command. For instance,

```matlab
day = 5;
temp = 70.8;
s = sprintf('The max temperature on Feb %d was %g degrees', day, temp);
```

will make a string \( s \) with the following content:

```
The max temperature on Feb 5 was 70.8 degrees
```

Note that the symbol \%d has been replaced by the value of \( \text{day} \) displayed as an integer. The symbol \%g has been replaced by the value of \( \text{day} \) displayed as a floating-point number in the shortest possible form. Type \text{help sprintf} for more details.

Type \text{help print} for options for printing your plots. In Windows, you can also print into the clipboard with the -dmeta option, and then paste into a document.

2. Chapter 4 of the supplementary notes shows (under Russian Roulette) how to generate \( n \) values out of a binomial distribution with parameters \( m \) and \( p \). The Matlab function binomial that does so is provided with this homework assignment on the class web site.

This problem revisits the Tampa/Orlando example in Section 1.4 of the textbook (EXAMPLE 1: A Car Rental Company), and gives a new, probabilistic interpretation to the system. One can think of a state vector

\[
x(n) = \begin{bmatrix}
x_1(n) \\
x_2(n)
\end{bmatrix}
\]

whose entries \( x_1(n) \) and \( x_2(n) \) denote the actual (as opposed to average) numbers of vehicles in the Orlando and Tampa offices, respectively, at the end of day \( n \). Then, if we assume that the distribution of the number of vehicles that stay in Orlando is modeled by a binomial distribution, we can say that the number of vehicles from Orlando that stay in Orlando the next day is

\[
\text{OrlandoToOrlando} = \text{binomial}(x(1, n), p(1), 1);
\]

where \( p(1) \) is the probability that a vehicle rented in Orlando on day \( n \) stays in Orlando the next day. Make sure you understand this instruction. If necessary, re-read the theory in the supplementary notes, and examine the code for \text{binomial} to understand the meaning of each of the input arguments to the function, and of its result.

Following the example in the book, but now with a new, probabilistic interpretation, we set \( p(1) = 0.6 \). Analogously, we define the probability that a vehicle from Tampa stays in Tampa on any one day as \( p(2) = 0.7 \), and we can write

\[
\text{TampaToTampa} = \text{binomial}(x(2, n), p(2), 1);
\]
where $\text{TampaToTampa}$ denotes the number of vehicles rented in Tampa on day $n$ that stay in Tampa the next day.

(a) What happens in the Tampa/Orlando scenario in the book to cars rented in Orlando on day $n$ that do not stay in Orlando between day $n$ and day $n + 1$?

(b) Given your previous answer, what can you say about the number $\text{OrlandoToTampa}$ of cars that are rented in Orlando on day $n$ and end up in Tampa on day $n + 1$? State your answer in terms of $x_1(n)$ (which in Matlab reads $x(1, n)$) and $\text{OrlandoToOrlando}$.

(c) What can you say about the number $\text{TampaToOrlando}$ of cars that are rented in Tampa on day $n$ and end up in Orlando on day $n + 1$? State your answer in terms of $x_2(n)$ and $\text{TampaToTampa}$.

(d) Write a fragment of Matlab code that computes the two numbers $x(:, n+1)$ of cars that are in Orlando and Tampa on day $n + 1$ as a function of the two numbers $x(:, n)$ of cars that are in Orlando and Tampa on day $n$. If run multiple times, even with exactly the same input arguments, your code will give different results, to reflect the reality that what happens on any given day depends on factors that are difficult to model individually.

(e) Can the values of $x(:, n+1)$ resulting from your code ever become negative? Why or why not?

(f) Complete your code. Specifically, use the code fragment you wrote in your previous answer to write a function with header

\begin{verbatim}
function x = rentals(x0, p, N)
\end{verbatim}

that takes a two-dimensional vector $x0$ with the number of cars in Orlando and Tampa on day 0; a two-dimensional vector $p$ that has the probabilities $p(1)$ and $p(2)$ defined earlier; and a number $N$ of iterations to perform. Your function should return a $2 \times N$ array $x$ such that $x(:, n)$ contains the number of cars in Orlando and Tampa on day $n - 1$. Make sure that your function checks the sizes of the input arguments, just as you did in your trajectory function, and that the values in $x0$ and $p$ are within their appropriate ranges. Hand in your code.

(g) Write a script that runs $\text{rentals}$ 20 times with the following arguments:

\begin{verbatim}
x0 = [300; 500];
p = [0.95; 0.88];
N = 30;
\end{verbatim}

and overlays the 20 plots of the resulting $x$ matrices in two figures, one for the number of cars in Orlando, the other for the number of cars in Tampa. Make sure that the horizontal axis starts at time 0, not 1, and that your axes are labeled. Put titles on your plots. Hand in your script and your plots.

(h) Why do more cars end up in Orlando in the long run?