MatLab Term Test 2 Prep

Syntax

Vectors and Matrices

you can use `linspace` to create $1 \times n$ vectors

```
x = linspace(0,10,100); % 100 evenly spaced points between 0 and 10
% Find the transpose
x_transpose = x'; % n x 100 vector
```

Create some matrix:

```
A = [1 1 1 ; 2 2 2 ; 3 3 3]; % seperate rows by semi-colon
```

Solving a linear system

Suppose `A = [1 1; 2 2] and` $A\vec{x} = b$. Then you can computer x using Matlab's backslash operater:

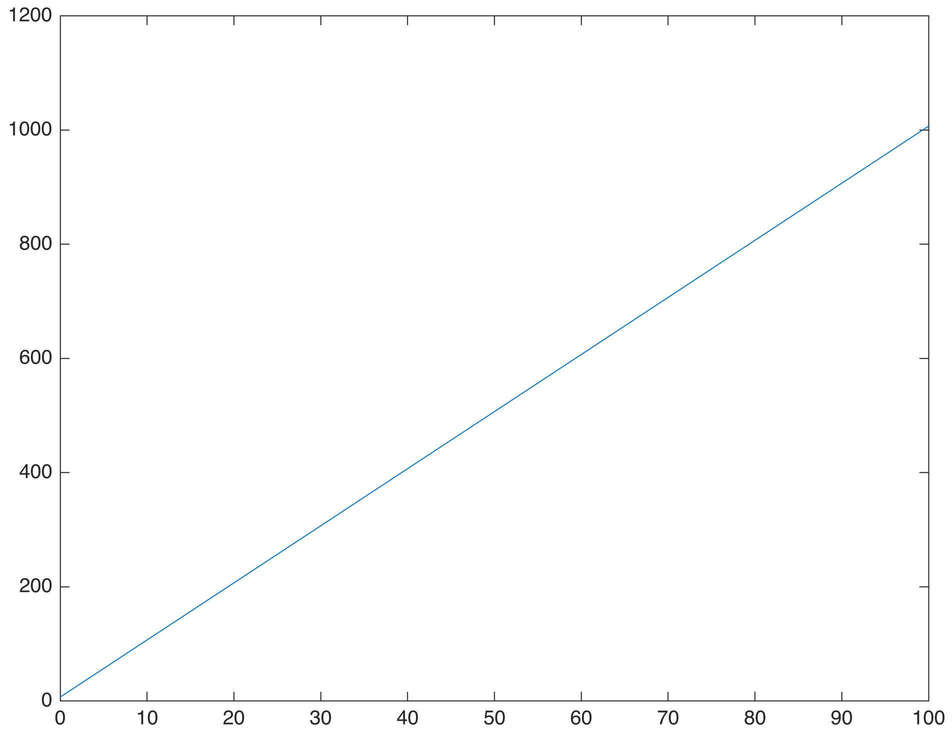
```
A = [1 1; 3 2];
b = [6 ; 4];

sol = A\b % output the solution of the system.
```

```
sol = 2×1
    -8
    14
```

Plotting functions

Say you want to plot $y = 10x + 7$. Generate your domain using `linspace`

```
x = linspace(0, 100, 100);
% generate y plot
y = 10*x + 7;
plot(x,y); % plots x against y
```

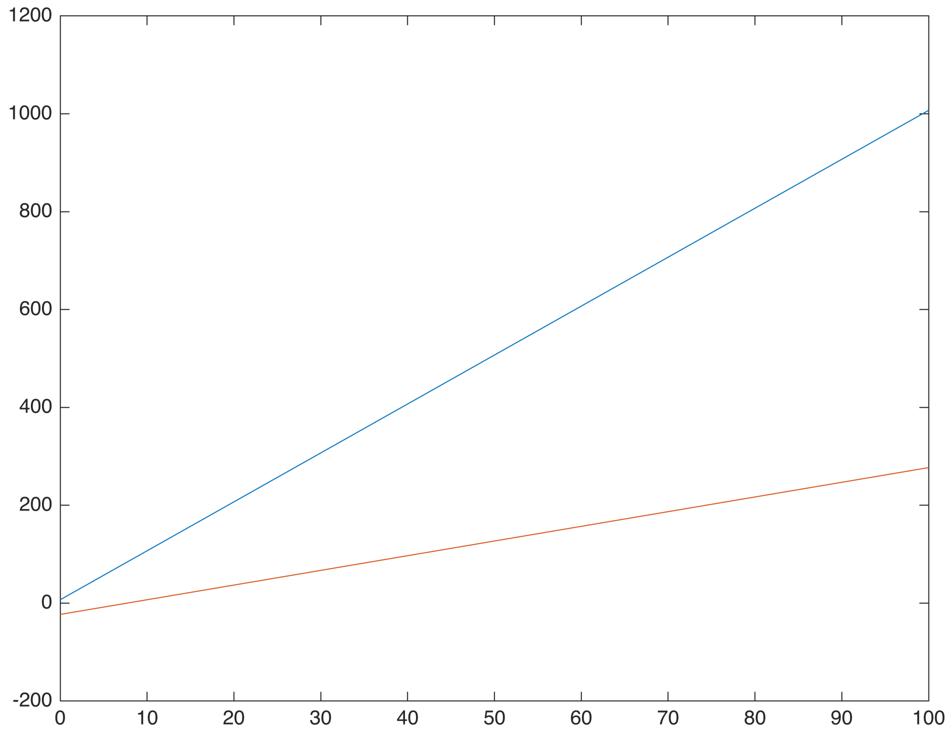


Suppose you want to generate multiple plots on the same graph. Use `hold on`

```
f1 = figure % create a figure and name it f1
```

```
f1 = 
  Figure (2) with properties:

      Number: 2
        Name: ''
       Color: [0.9400 0.9400 0.9400]
    Position: [1000 818 560 420]
       Units: 'pixels'

  Show all properties
```

```
y1 = 10*x + 7;
y2 = 3*x-23;
plot(x,y1)
hold on
plot(x,y2)
hold off % put hold off or it continues plotting on the same plot
```

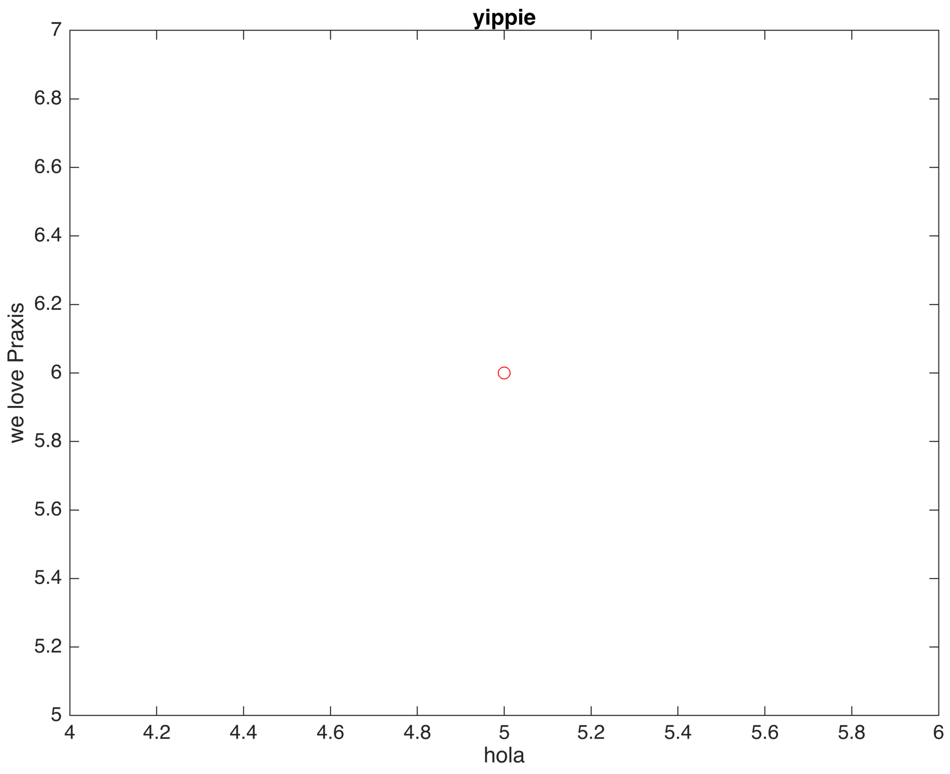


If you want to plot a point, use the `plot`

```
% plot the point 5,6
plot(5,6,'ro')
```

You can also label plots

```
xlabel("hola")
ylabel("we love Praxis")
title("yippie")
```



For loops

Indices start at **1 not 0.** Here's a `for` loop. Note both bounds are inclusive

```
for i = 0:5
    i;
end % must have `end` or you'll get error
```

To specify step size, add middle parameter

```
for i = 0:0.2:5
    i;
end
```

You can use nested `for` loops to, `for` example, populate a matrix

```
mat = []; % initalize an empty matrix
for i = 1:5 % you cannot have 0 index
    for j = 1:5
        mat(i,j) = 5;
    end
end
```

Something interesting: if you remove the semi-colon, you'll see that once a new row is started the entire row is populated with `0`.

Linear Algebra Stuff

Least Squares Regression

long winded and rambly explanation sorry, didn't have time to plan out the explanation.

If we have a set of data points that land on some function exactly, we know how to determine the regression line (the data points form a consistent matrix vector system which we can solve for).

Data from measurements does not often fit a function exactly so the solution to its matrix-vector system is inconsistent. We still want to find a regression line, which we can do by allowing error. The best regression line has the smallest error.

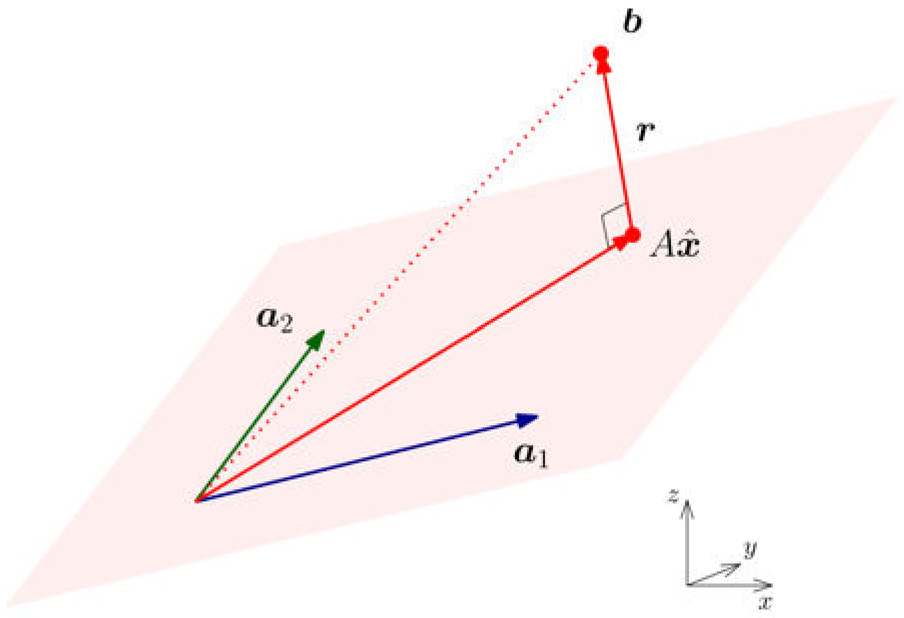
Start by setting our linear system. We have a list of (x, y) coordinate pairs and know that the trend line is $y = a + bx$. Let's store this info with matrices and vectors (I haven't thought much on how to think about matrices as linear transformations, but for now just use matrices+vectors to store information compactly).

Specifically, we want the form $A\vec{c} = \vec{y}$ to spit out $y = a + bx$, for every data point we input. Use what we know about matrix vector multiplication to get this behaviour, i.e. for a data set $(x_1, y_1), (x_2, y_2), .., (x_n, y_n)$:

Note I drop the vector notation because it looks somewhat bad in Matlab.

$\begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_n \end{bmatrix}\begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$. We define $A = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_n \end{bmatrix}$, $c = \begin{bmatrix} a \\ b \end{bmatrix}$, $y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$.

There is some $\vec{c} = \begin{bmatrix} a \\ b \end{bmatrix}$ s.t. the distance between $\vec{y}$ and $A\vec{c}$ is the smallest. Let's call the smallest distance between $\vec{y}$ and the column space of $A\vec{c}$ a vector $\vec{e}$ (technically $|\vec{e}|$). This distance is guaranteed to be minimum when $\vec{e}$ is perpendicular to the column space of A.



(this image uses different variable names, but the idea is the same. $\vec{e}$ is smallest when it is perpendicular to the red plane)

The column space of A being perpendicular to $\vec{e}$ implies that each column is perpendicular to $\vec{e}$, so each column dotted with $\vec{e}$ is 0.

We can achieve this operation by first taking the transpose of A, and mulitplying it with $\vec{e}$ (if this idea is confusing, think about how matrix vector is done mechanically: take a row from matrix and dot product it with the vector. When you take the transpose, you transposed row is your original column, so in effect we are taking the dot product of a vector with the columns of the untransposed matrix).

Put this information into equations.

(i) the error vector is the vector between $A\vec{c}$ and $\vec{y}$: $\vec{e} = \vec{y} - A\vec{c}$

(ii) the transpose of A times $\vec{e}$ is 0:

$$A^{\mathrm{T}}\vec{e} = 0 \rightarrow A^{\mathrm{T}}(\vec{y} - A\vec{c}) = 0$$

$$A^{\mathrm{T}}\vec{y} = (AA^{\mathrm{T}})\vec{c}$$

Solve for $\vec{c}$ using Gaussian elimination. OR!! because this is Matlab!! you can just use the backslash operator

```
y = [ 1; 2 ]; % solution set
A = [1 1; 1 2]; % random matrix A.
A_transpose = A.'; % transpose of A
```

```
A_star = A*A_transpose; % dummy variable

c = A_star\(A_transpose * y)
```

```
c = 2×1
     0
     1
```

(on a clarity of explanations note, I probably should've denoted the least squares solution as $\vec{c}_{\text{LS}}$ and called the solutions vector something other than $\vec{y}$.

AND THAT'S LEAST SQUARES REGRESSION YIPPIEEEEEEEEEE

Solutions to Differential Equations

Curves are locally straight lines if you zoom in enough, which we can leverage to predict the shape of a function. This implies for some function $y(t)$, $y(t+\Delta t) \approx y(t) + \frac{dy}{dt}\Delta t$, and this approximation is best for arbitrarily tiny values of t. We can extend this idea to plotting out ODEs. It's sometimes hard (or impossible) to find the analytical solution to an ODE. So we use the fact that $y(t+\Delta t) \approx y(t) + \frac{dy}{dt}\Delta t$ to "draw out" the solution to our DE.

Let's walk through solving a first order ODE, with the goal of plotting y vs x.

Consider the following set of ODEs (from Lab 3):

$$\frac{dx(t)}{dt} = x'(t) = ax(t) + by(t) \quad x(0) = x_0$$
$$\frac{dy(t)}{dt} = y'(t) = cx(t) + dy(t) \quad y(0) = y_0$$

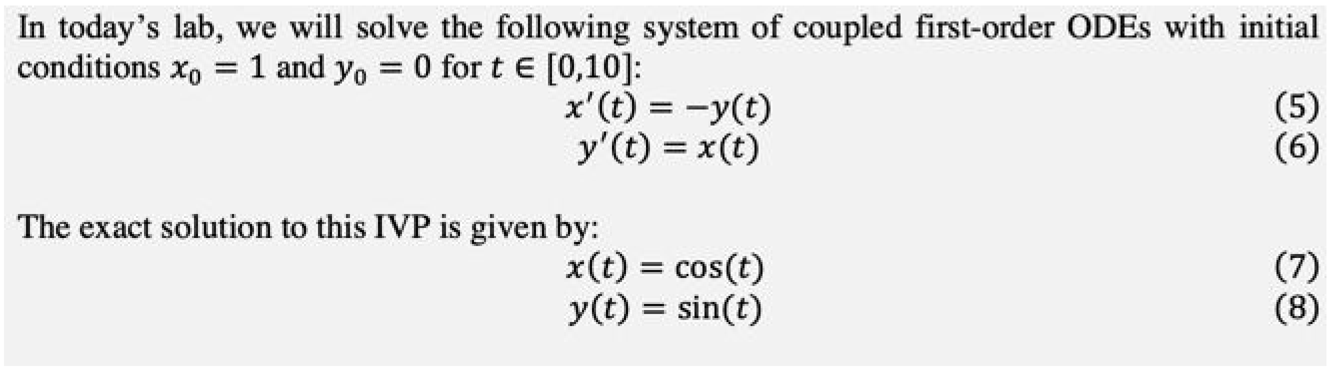
The motivation behind using matrices and vectors to store the above information is that it makes it compact (note to self: read up about the thingy). In particular, we say that $\begin{bmatrix} a & b \\ c & d \end{bmatrix}\begin{bmatrix} x(t) \\ y(t) \end{bmatrix} = \begin{bmatrix} x'(t) \\ y'(t) \end{bmatrix}$. Let's give each part of that equation a name, because we'll refer to them often.

Call $Z = \begin{bmatrix} x(t) \\ y(t) \end{bmatrix}$, $Z' = \begin{bmatrix} x'(t) \\ y'(t) \end{bmatrix}$, $A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$, and $Z_0 = \begin{bmatrix} x_0 \\ y_0 \end{bmatrix}$. So our equation can be written as $Z' = AZ$ (note Z and Z' are vectors. I should denote them with vector symbols but I am sticking to the format used in the lab handout because it's easier to typeset :D).

To plot y on x, use your initial conditions to determine A (*if necessary*). Then use the fact $y(t+\Delta t) \approx y(t) + \frac{dy}{dt}\Delta t$. Applied to our problem, $Z(t+\Delta t) = Z(t) + Z'(t)\Delta t$ which after substituting $Z' = AZ$ becomes $Z(t+\Delta t) = Z(t) + AZ(t)\Delta t$. The problem with using the equation just derived is that we do not know $Z(t)$. At best, we know $Z(t-\Delta t)$ (this comes from knowing the initial condition, which is why having an initial value is important, or we'd have an infinite set of graphs!!). Rather, we say $Z(t-\Delta t) = Z(t-\Delta t) + AZ(t-\Delta t)\Delta t$.

This process of estimating the current value of a function, based on previous values and slope, is known as **Euler's Method**.

Now we just have to program it!!! A small note: it's easier to program Z_{n-1} as the previous value rather than $Z(t-\Delta t)$, but that's just an implementation consideration.



In today's lab, we will solve the following system of coupled first-order ODEs with initial conditions $x_0 = 1$ and $y_0 = 0$ for $t \in [0,10]$:

$$x'(t) = -y(t) \tag{5}$$

$$y'(t) = x(t) \tag{6}$$

The exact solution to this IVP is given by:

$$x(t) = \cos(t) \tag{7}$$

$$y(t) = \sin(t) \tag{8}$$

```
A = [0 -1; 1 0];
Z0 = [1; 0];
resolution = 100000; % tells you how many steps to take in the given
interval
T = 10;
dt = T/resolution
```

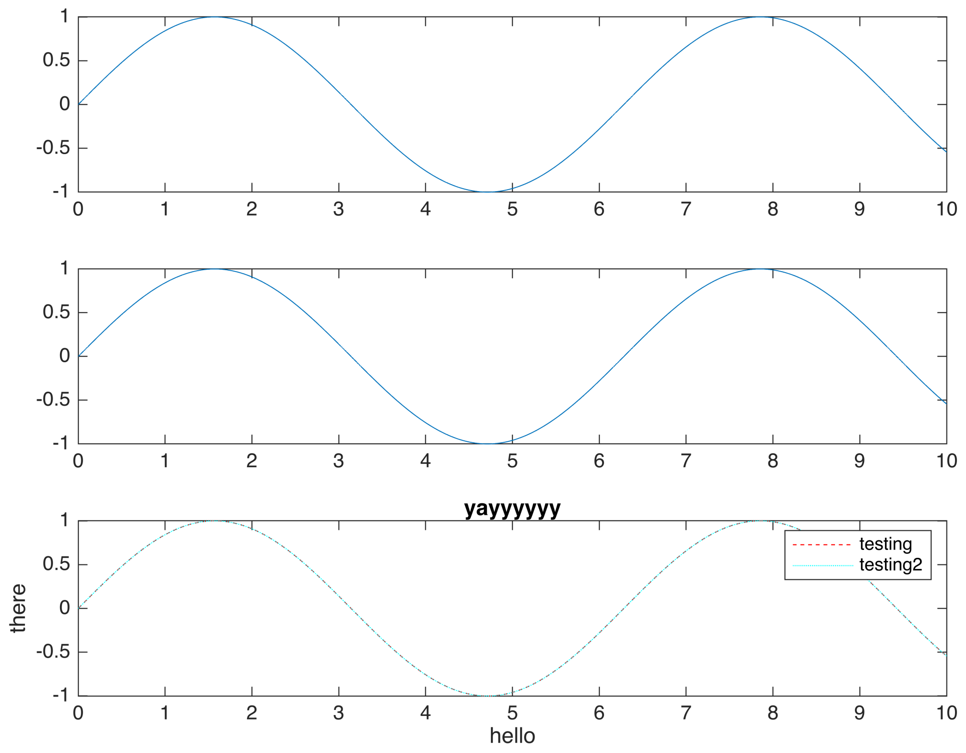
```
dt =
1.0000e-04
```

```
t = 0:dt:T
```

```
t = 1×100001
         0    0.0001    0.0002    0.0003    0.0004    0.0005    0.0006    0.0007 ···
```

```
% Perform Euler's method
Z = NaN(2, length(t)); % define an empty solution vector
% define initial conditions
Z(1,1) = 1; % x = 1
Z(2,1) = 0; % y = 0
for n=2:length(t)
    Z(:,n) = Z(:,n-1) + A*Z(:,n-1)*dt; % Z(:,n) means select ALL rows in
the nth column
end
% plot the solution
% compare it to sin x
x_set = 0:dt:T
x_set = 1×100001
         0    0.0001    0.0002    0.0003    0.0004    0.0005    0.0006    0.0007 ···
```

```
subplot(3,1,1)
plot(t, Z(2,:))
subplot(3,1,2)
plot(x_set, sin(x_set))
subplot(3,1,3)
plot(x_set, sin(x_set), "--r")
hold on;
plot(x_set, Z(2,:), ":c")
legend('testing', 'testing2')
xlabel("hello")
ylabel("there")
title("yayyyyyy")
hold off;
```



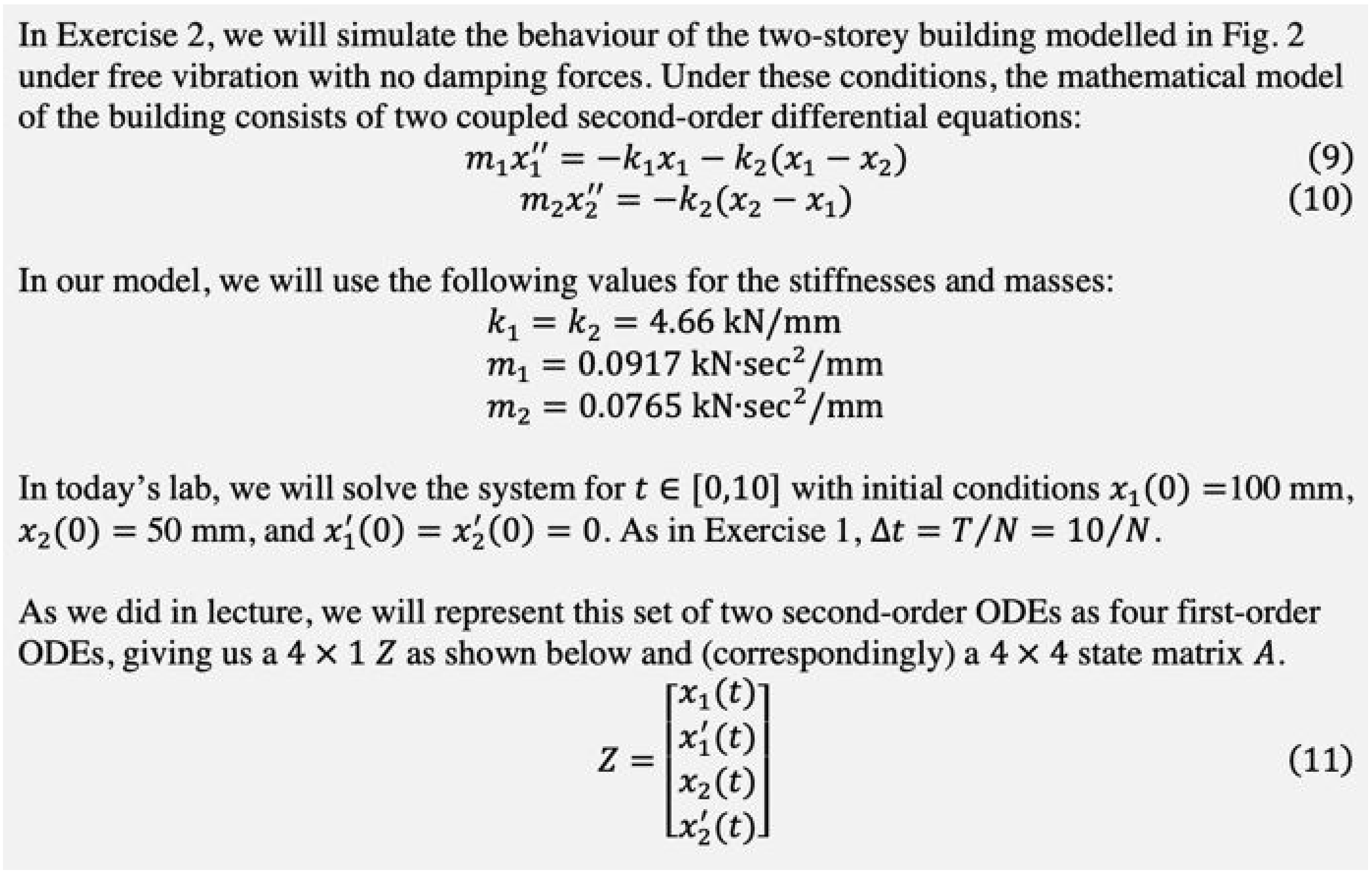
You can improve the efficency by using **IEM**, which just says it's better to take the average slope between the current value and previous value than to take only the slope of the previous value. To do this, we need to calculate Z'_n:

(i) calculate Z_n, $Z_n = Z_{n-1} + AZ_{n-1}\Delta t$

(ii) use Z_n to get Z'_n.Then you can better estimate Z_n. i.e. $Z_n = Z_{n-1} + \frac{\Delta t}{2}A(Z_{n-1} + Z_n)$.

Example 2: Second Order ODE

Same idea, just a bit more complicated to construct A. I'm using the example given in Lab 3 Q2.



In Exercise 2, we will simulate the behaviour of the two-storey building modelled in Fig. 2 under free vibration with no damping forces. Under these conditions, the mathematical model of the building consists of two coupled second-order differential equations:

$$m_1 x_1'' = -k_1 x_1 - k_2(x_1 - x_2) \tag{9}$$

$$m_2 x_2'' = -k_2(x_2 - x_1) \tag{10}$$

In our model, we will use the following values for the stiffnesses and masses:

$$k_1 = k_2 = 4.66 \text{ kN/mm}$$

$$m_1 = 0.0917 \text{ kN}\cdot\text{sec}^2\text{/mm}$$

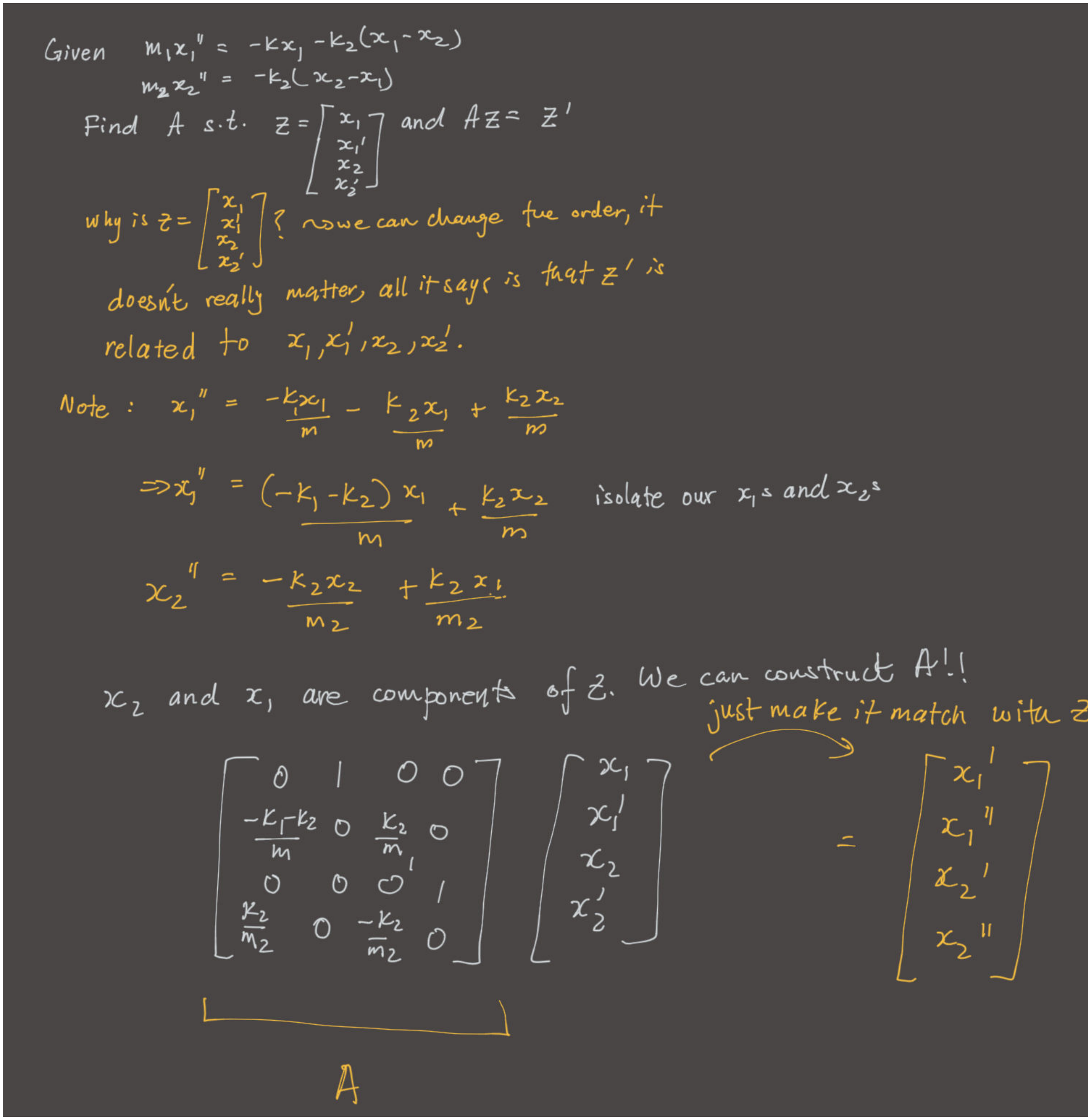
$$m_2 = 0.0765 \text{ kN}\cdot\text{sec}^2\text{/mm}$$

In today's lab, we will solve the system for $t \in [0,10]$ with initial conditions $x_1(0) = 100$ mm, $x_2(0) = 50$ mm, and $x_1'(0) = x_2'(0) = 0$. As in Exercise 1, $\Delta t = T/N = 10/N$.

As we did in lecture, we will represent this set of two second-order ODEs as four first-order ODEs, giving us a 4×1 Z as shown below and (correspondingly) a 4×4 state matrix A.

$$Z = \begin{bmatrix} x_1(t) \\ x_1'(t) \\ x_2(t) \\ x_2'(t) \end{bmatrix} \tag{11}$$

The image below shows how you can construct A (I didn't want to typeset this it's too late rahhh)



Given $m_1 x_1'' = -k x_1 - k_2(x_1 - x_2)$

$m_2 x_2'' = -k_2(x_2 - x_1)$

Find A s.t. $z = \begin{bmatrix} x_1 \\ x_1' \\ x_2 \\ x_2' \end{bmatrix}$ and $Az = z'$

why is $z = \begin{bmatrix} x_1 \\ x_1' \\ x_2 \\ x_2' \end{bmatrix}$? no we can change the order, it doesn't really matter, all it says is that z' is related to x_1, x_1', x_2, x_2'.

Note : $x_1'' = \frac{-k_1 x_1}{m} - \frac{k_2 x_1}{m} + \frac{k_2 x_2}{m}$

$\Rightarrow x_1'' = \frac{(-k_1 - k_2)\, x_1}{m} + \frac{k_2 x_2}{m}$ isolate our x_1s and x_2s

$x_2'' = \frac{-k_2 x_2}{m_2} + \frac{k_2 x_1}{m_2}$

x_2 and x_1 are components of z. We can construct A!!

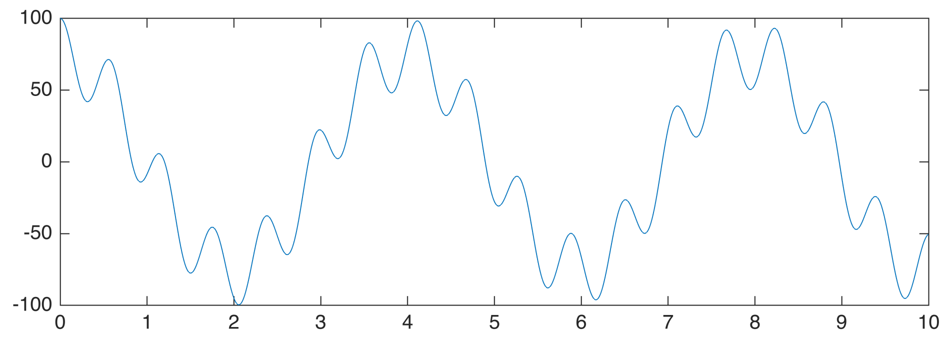
just make it match with z

$$\underbrace{\begin{bmatrix} 0 & 1 & 0 & 0 \\ \frac{-k_1-k_2}{m} & 0 & \frac{k_2}{m_1} & 0 \\ 0 & 0 & 0 & 1 \\ \frac{k_2}{m_2} & 0 & \frac{-k_2}{m_2} & 0 \end{bmatrix}}_{A} \begin{bmatrix} x_1 \\ x_1' \\ x_2 \\ x_2' \end{bmatrix} = \begin{bmatrix} x_1' \\ x_1'' \\ x_2' \\ x_2'' \end{bmatrix}$$

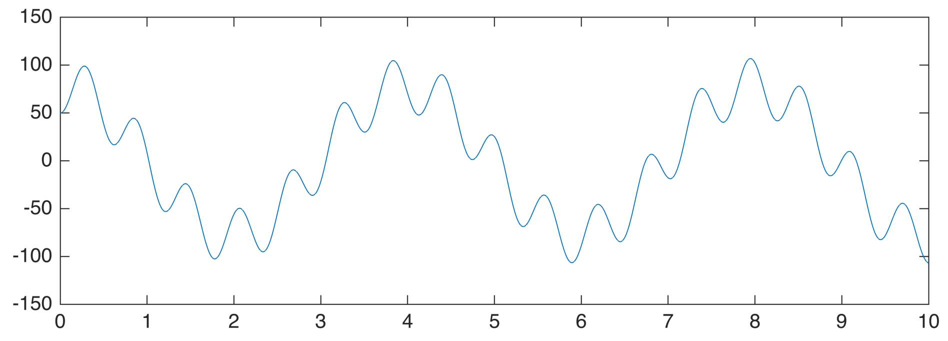
The Matlab coding process is essentially the same. IEM is implemented below.

```
k1 = 4.66;
k2 = 4.66;
m1 = 0.0917;
m2 = 0.0765;
```

```
A = [0 1 0 0 ; (-k1-k2/m1) 0 k2/m1 0; 0 0 0 1 ; k2/m2 0 -k2/m2 0];
T = 10;
resolution = 10000;
dt = T/resolution;
Z = NaN(4, resolution);
Z(1,1) = 100;
Z(2,1) = 0;
Z(3,1) = 50;
Z(4,1) = 0;
for n=2:resolution
    currVal = Z(:, n-1) + A*Z(:,n-1)*dt;
    % update Z
    Z(:,n) = Z(:,n-1) + 0.5*dt*A*(Z(:,n-1)+currVal);
end
figure
% I think something might've gone wrong here. Double check later.
subplot(2,1,1)
plot(0:dt:T-dt, Z(1,:))
subplot(2,1,2)
plot(0:dt:T-dt, Z(3,:))
```



100
50
0
-50
-100
0
1
2
3
4
5
6
7
8
9
10



150
100
50
0
-50
-100
-150
0
1
2
3
4
5
6
7
8
9
10