Introduction to Modeling and Simulation

Dr. Nhut Ho
ME584
Agenda

- Dynamic Systems
- Modeling of Dynamic Systems
- Introduction to Matlab

- Active learning: Pair-share questions, Exercises in class
Dynamic Systems
Static V. Dynamic Systems

• Static
  » Output does not change with time
  » Output at any time depends on input at that time only

• Dynamic
  » Output is not instantaneously proportional to input or disturbance, may continue after input held constant
  » Present output depends on past inputs
Dynamic Systems in Engineering
Disciplines

- Mechanical systems
- Electrical systems
- Fluid systems
- Thermal systems
- Mixed systems
  - Electro-Mechanical
  - Fluid-Mechanical
  - Thermo-Mechanical
  - Electro-Thermal

Name an example and describe its dynamic response
Modeling

• What is a model?
  » Physical models (e.g., scale model)
  » Graphs or plots (e.g., time-dependent behavior)
  » Mathematical models

• Modeling
  » Identifying physical dynamic effects
  » Writing differential equations using conservation and property laws
  » Expressing in differential equations forms
Modeling of Dynamic Systems
Modeling Steps

- Inaccuracies propagate in each step (e.g., linearization, ignoring higher dynamics)
- Iterative modifications needed to get required output
- Example: modeling steps for your favorite sport device
Representing Dynamic Systems

Diagram showing the flow of inputs, disturbances, initial conditions, dynamic system, outputs, transient response, and settling time.
# Mathematical Representation and Solution Methods

<table>
<thead>
<tr>
<th>System Equations</th>
<th>Solution Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classical differential equation ( \ddot{x} + 2\dot{x} + 3x = f )</td>
<td>Analytical Solution</td>
</tr>
<tr>
<td>Transfer function ( X(s)s^2 + 2sX(s) + 3X(s) = F(s) )</td>
<td>Laplace Transform</td>
</tr>
</tbody>
</table>
| State space \[
\begin{bmatrix}
\dot{x} \\
\dot{z}
\end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -3 & -2 \end{bmatrix} \begin{bmatrix} x \\ z \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} f
\]
| Digital/Analog Simulation                             |                      |
Temperature distribution in a plate. (a) Distributed-parameter representation. (b) Lumped-parameter representation using three elements.

\[ f(T, \frac{\partial T}{\partial t}, \frac{\partial^2 T}{\partial x^2}, \frac{\partial^2 T}{\partial y^2}, \frac{\partial^2 T}{\partial z^2}) = 0 \]  

\[ f(T, \frac{\partial T}{\partial t}) = 0 \]
Model Classification Tree

- Increasing ease of analysis
- Increasing realism

Static

- Increasing realism
- Stochastic

Dynamic

- Deterministic
- Distributed parameter

Lumped parameter

Linear

- Nonlinear

Variable coefficient

Constant coefficient

- Continuous time
- Discrete time

First-order

Second-order

... (Higher-order)

(Coupled first-order)
Let $u(t)$ be input and $y(t)$ be output, the system is linear if

- **Additivity**: Response to $u_1 + u_2$ is $y_1 + y_2$
- **Homogeneity**: Response to $au_1$ is $ay_1$

Example: Show that $y(t) = 2x(t) + 3$ is not linear

- If $x_1 = 2$ and $x_2 = 3$, then
  - $y_1 = 2*2 + 3 = 7$, and $y_2 = 2*3 + 3 = 9$
- But for $x_3 = x_1 + x_2 = 5$, then
  - $y_3 = 2*5 + 3 = 13$
  - And $y_3 \neq y_1 + y_2 = 16$
- System is not linear
Pair-share exercise: Linear System Example

- Consider this system: \( y(t) = t \, u(t) \)
- Is this system linear?
- Consider two *arbitrary* inputs \( u_1 \) and \( u_2 \)
  - \( y_1 = tu_1 \)
  - \( y_2 = tu_2 \)
- Let \( u_3 = au_1 + bu_2 \), where \( a \) and \( b \) are *arbitrary* scalar constants
  - \( y_3 = tu_3 = t \, (au_1 + bu_2) = atu_1 + btu_2 = ay_1 + by_2 \)
- System is linear
Introduction to Matlab
(1) Getting Started
(2) Scripts
(3) Making Variables
(4) Manipulating Variables
(5) Basic Plotting
Getting Started

- Open up MATLAB for Windows
  - Through the START Menu
  - Or Click on Matlab icon
Making Folders

• Use folders to keep your programs organized

• To make a new folder, click the ‘Browse’ button next to ‘Current Directory’

• Click the ‘Make New Folder’ button, and change the name of the folder. **Do NOT use spaces** in folder names. In the MATLAB folder, make a new folder: ME584\MatlabIntro

• Highlight the folder you just made and click ‘OK’
• The current directory is now the folder you just created
• To see programs outside the current directory, they should be in the Path. Use File-> Set Path to add folders to the path
Customization

- File → Preferences
  - Allows you personalize your MATLAB experience
MATLAB Basics

- MATLAB can be thought of as a super-powerful graphing calculator
  - Remember the TI-83 from calculus?
  - With many more buttons (built-in functions)

- In addition it is a programming language
  - MATLAB is an interpreted language, like Java
  - Commands executed line by line
• **help**
  - **The most** important function for learning MATLAB on your own

• To get info on how to use a function:
  - » **help sin**
    - Help lists related functions at the bottom and links to the doc

• To get a nicer version of help with examples and easy-to-read descriptions:
  - » **doc sin**

• To search for a function by specifying keywords:
  - » **doc** + Search tab
(1) Getting Started

(2) Scripts

(3) Making Variables

(4) Manipulating Variables

(5) Basic Plotting
Scripts: Overview

• Scripts are
  - collection of commands executed in sequence
  - written in the MATLAB editor
  - saved as m-files (.m extension)

• To create an m-file from command-line
  » edit helloWorld.m

• or click
Scripts: the Editor

- Line numbers
- m-file path
- Debugging tools
- Real-time error check
- Help file
- Comments
- Possible breakpoints

* Means that it's not saved
Scripts: Some Notes

• **COMMENT!**
  - Anything following a `#` is seen as a comment
  - The first contiguous comment becomes the script's help file
  - Comment thoroughly to avoid wasting time later

• Note that scripts are somewhat static, since there is no input and no explicit output

• All variables created and modified in a script exist in the workspace even after it has stopped running
Exercise: Scripts

Make a \texttt{helloWorld} script

- When run, the script should display the following text:
  
  \begin{verbatim}
  Hello World!
  I am going to learn MATLAB!
  \end{verbatim}

- **Hint:** use \texttt{disp} to display strings. Strings are written between single quotes, like \texttt{'This is a string'}
Exercise: Scripts

Make a `helloWorld` script

- When run, the script should display the following text:
  
  Hello World!
  I am going to learn MATLAB!

- **Hint:** use `disp` to display strings. Strings are written between single quotes, like 'This is a string'

- Open the editor and save a script as `helloWorld.m`. This is an easy script, containing two lines of code:
  
  ```matlab
  » % helloWorld.m
  » % my first hello world program in MATLAB

  » disp('Hello World!');
  » disp('I am going to learn MATLAB!');
  ```
Variable Types

• MATLAB is a weakly typed language
  ➢ No need to initialize variables!

• MATLAB supports various types, the most often used are
  » 3.84
    ➢ 64-bit double (default)
  » ‘a’
    ➢ 16-bit char

• Most variables you’ll deal with will be vectors or matrices of doubles or chars

• Other types are also supported: complex, symbolic, 16-bit and 8 bit integers, etc. You will be exposed to all these types through the homework
Naming variables

• To create a variable, simply assign a value to a name:
  » var1=3.14
  » myString='hello world'

• Variable names
  ➢ first character must be a LETTER
  ➢ after that, any combination of letters, numbers and _
  ➢ CASE SENSITIVE! (var1 is different from Var1)

• Built-in variables. Don’t use these names!
  ➢ i and j can be used to indicate complex numbers
  ➢ pi has the value 3.1415926...
  ➢ ans stores the last unassigned value (like on a calculator)
  ➢ Inf and -Inf are positive and negative infinity
  ➢ NaN represents ‘Not a Number’
Scalars

• A variable can be given a value explicitly
  » a = 10
    ➢ shows up in workspace!

• Or as a function of explicit values and existing variables
  » c = 1.3*45-2*a

• To suppress output, end the line with a semicolon
  » cooldude = 13/3;
Arrays

• Like other programming languages, arrays are an important part of MATLAB
• Two types of arrays
  
  (1) matrix of numbers (either double or complex)
  
  (2) cell array of objects (more advanced data structure)

MATLAB makes vectors easy!
That’s its power!
Row Vectors

• Row vector: comma or space separated values between brackets
  
  » row = [1 2 5.4 -6.6]
  » row = [1, 2, 5.4, -6.6];

• Command window:

  >> row=[1 2 5.4 -6.6]

  row =

  1.0000   2.0000   5.4000  -6.6000

• Workspace:

  ![Workspace screenshot]

<table>
<thead>
<tr>
<th>Name</th>
<th>Size</th>
<th>Bytes</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>row</td>
<td>1x4</td>
<td>32</td>
<td>double array</td>
</tr>
</tbody>
</table>
Column Vectors

- Column vector: semicolon separated values between brackets
  
  » column = [4;2;7;4]

- Command window:  

  >> column=[4;2;7;4]

  
  
  column =

  4
  2
  7
  4

- Workspace:
size & length

- You can tell the difference between a row and a column vector by:
  - Looking in the workspace
  - Displaying the variable in the command window
  - Using the size function

\[
\text{size}(	ext{row}) \quad \text{size}(	ext{column})
\]

\[
\begin{array}{c}
\text{ans} = \\
1 \quad 4
\end{array}
\quad \quad
\begin{array}{c}
\text{ans} = \\
4 \quad 1
\end{array}
\]

- To get a vector's length, use the length function

\[
\text{length}(	ext{row}) \quad \text{length}(	ext{column})
\]

\[
\begin{array}{c}
\text{ans} = \\
4
\end{array}
\quad \quad
\begin{array}{c}
\text{ans} = \\
4
\end{array}
\]
Matrices

- Make matrices like vectors
- Element by element
  - \( a = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \)
- By concatenating vectors or matrices (dimension matters)
  - \( a = [1 \ 2] \)
  - \( b = [3 \ 4] \)
  - \( c = [5;6] \)
  - \( d = [a; b] \)
  - \( e = [d \ c] \)
  - \( f = [[e \ e]; [a \ b \ a]] \)
  - \( \text{str} = ['Hello, I am ' 'John'] \)
  - Strings are character vectors
save/clear/load

• Use **save** to save variables to a file
  » `save myFile a b`
  ➢ saves variables a and b to the file myfile.mat
  ➢ myfile.mat file is saved in the current directory
  ➢ Default working directory is
    » `\MATLAB`
  ➢ Make sure you’re in the desired folder when saving files. Right now, we should be in:
    » `MATLAB\ME584\MatlabIntro`

• Use **clear** to remove variables from environment
  » `clear a b`
  ➢ look at workspace, the variables a and b are gone

• Use **load** to load variable bindings into the environment
  » `load myFile`
  ➢ look at workspace, the variables a and b are back

• Can do the same for entire environment
  » `save myenv; clear all; load myenv;`
Exercise: Variables

Get and save the current date and time

• Create a variable `start` using the function `clock`
• What is the size of `start`? Is it a row or column?
• What does `start` contain? See `help clock`
• Convert the vector `start` to a string. Use the function `datestr` and name the new variable `startString`
• Save `start` and `startString` into a mat file named `startTime`
Exercise: Variables

Get and save the current date and time
- Create a variable `start` using the function `clock`
- What is the size of `start`? Is it a row or column?
- What does `start` contain? See `help clock`
- Convert the vector `start` to a string. Use the function `datestr` and name the new variable `startString`
- Save `start` and `startString` into a mat file named `startTime`

```matlab
» help clock
» start=clock;
» size(start)
» help datestr
» startString=datestr(start);
» save startTime start startString
```
Exercise: Variables

Read in and display the current date and time
• In helloWorld.m, read in the variables you just saved using `load`

• Display the following text:
  I started learning Matlab on *start date and time*

• **Hint:** use the `disp` command again, and remember that strings are just vectors of characters so you can join two strings by making a row vector with the two strings as sub-vectors.
Read in and display the current date and time

- In helloWorld.m, read in the variables you just saved using `load`

- Display the following text:
  
  I started learning Matlab on *start date and time*

- **Hint:** use the `disp` command again, and remember that strings are just vectors of characters so you can join two strings by making a row vector with the two strings as sub-vectors.

```matlab
» load startTime
» disp(['I started learning Matlab on ' ... startString]);
```
Outline

(1) Getting Started
(2) Scripts
(3) Making Variables
(4) Manipulating Variables
(5) Basic Plotting
Basic Scalar Operations

- Arithmetic operations (+, -, *, /)
  - \( \frac{7}{45} \)
  - \((1+i) \times (2+i)\)
  - \(1 / 0\)
  - \(0 / 0\)

- Exponentiation (^)
  - \(4^2\)
  - \((3+4j)^2\)

- Complicated expressions, use parentheses
  - \((2+3) \times 3)^{0.1}\)

- Multiplication is NOT implicit given parentheses
  - \(3(1+0.7)\) gives an error

- To clear command window
  - clc
Built-in Functions

- MATLAB has an enormous library of built-in functions
- Call using parentheses – passing parameter to function
  ```matlab
  » sqrt(2)
  » log(2), log10(0.23)
  » cos(1.2), atan(-.8)
  » exp(2+4*i)
  » round(1.4), floor(3.3), ceil(4.23)
  » angle(i); abs(1+i);
  ```
Exercise: Scalars

You will learn MATLAB at an exponential rate! Add the following to your helloWorld script:

- Your learning time constant is **1.5 days**. Calculate the number of **seconds** in 1.5 days and name this variable `tau`
- This class lasts 5 days. Calculate the number of seconds in 5 days and name this variable `endOfClass`
- This equation describes your knowledge as a function of time $t$:

$$ k = 1 - e^{-t/\tau} $$

- How well will you know MATLAB at `endOfClass`? Name this variable `knowledgeAtEnd`. (use `exp`)
- Using the value of `knowledgeAtEnd`, display the phrase:

  At the end of 6.094, I will know X% of Matlab

- **Hint:** to convert a number to a string, use `num2str`
Exercise: Scalars

» secPerDay=60*60*24;
» tau=1.5*secPerDay;
» endOfClass=5*secPerDay
» knowledgeAtEnd=1-exp(-endOfClass/tau);
» disp(['At the end of 6.094, I will know ' ... 
num2str(knowledgeAtEnd*100) '% of Matlab'])
Transpose

- The transpose operators turn a column vector into a row vector and vice versa
  
  ```matlab
  a = [1 2 3 4+i]
  transpose(a)
  a'
  a.'
  ```

- The `'` gives the Hermitian-transpose, i.e. transposes and conjugates all complex numbers

- For vectors of real numbers `. '` and `' ` give same result

Chap1 48
Addition and Subtraction

- Addition and subtraction are element-wise; sizes must match (unless one is a scalar):

\[
\begin{bmatrix}
12 & 3 & 32 & -11 \\
+ [ & 2 & 11 & -30 & 32 ] \\
\hline
= [ & 14 & 14 & 2 & 21 ]
\end{bmatrix}
\]

\[
\begin{bmatrix}
12 \\
1 \\
-10 \\
0
\end{bmatrix}
- 
\begin{bmatrix}
3 \\
-1 \\
13 \\
33
\end{bmatrix}
= 
\begin{bmatrix}
9 \\
2 \\
-23 \\
-33
\end{bmatrix}
\]

- The following would give an error
  
  » `c = row + column`

- Use the transpose to make sizes compatible
  
  » `c = row’ + column`
  
  » `c = row + column’`

- Can sum up or multiply elements of vector
  
  » `s=sum(row);`
  
  » `p=prod(row);`
Element-Wise Functions

• All the functions that work on scalars also work on vectors
  » \( t = [1 \ 2 \ 3] \);
  » \( f = \exp(t) \);
  ➢ is the same as
  » \( f = [\exp(1) \ \exp(2) \ \exp(3)] \);

• If in doubt, check a function’s help file to see if it handles vectors elementwise

• Operators \((*/^)\) have two modes of operation
  ➢ element-wise
  ➢ standard
Operators: element-wise

- To do element-wise operations, use the dot: . (.* , ./ , .\^).
  BOTH dimensions must match (unless one is scalar)!
  ```matlab
  » a=[1 2 3];b=[4;2;1];
  » a.*b, a./b, a.^b → all errors
  » a.*b', a./b', a.^(b') → all valid
  ```

```
[1 2 3].*[4]
[1] 2
[4]
1
3
3×1.*3×1=3×1
```

```
[1 1 1]
[1 2 3]
[2 2 2].*[1 2 3] = [2 4 6]
[3 3 3]
[1 2 3]
3×3.*3×3=3×3
```

```
[1 2]
[3 4].^2=\[1^2 2^2\]
[3^2 4^2]
```

Can be any dimension
Operators: standard

- Multiplication can be done in a standard way or element-wise
- Standard multiplication (*) is either a dot-product or an outer-product
  - Remember from linear algebra: inner dimensions must MATCH!!
- Standard exponentiation (^) can only be done on square matrices or scalars
- Left and right division (/ \) is same as multiplying by inverse
  - Our recommendation: just multiply by inverse (more on this later)

\[
\begin{bmatrix}
1 & 2 & 3
\end{bmatrix} \times \begin{bmatrix}
4
\end{bmatrix} = 11
\]

\[
\begin{bmatrix}
1 & 2 \\
3 & 4
\end{bmatrix}
\begin{bmatrix}
1 & 2
\end{bmatrix} = \begin{bmatrix}
1 & 2
\end{bmatrix}
\begin{bmatrix}
1 & 2
\end{bmatrix}
\]

\[
\begin{bmatrix}
1 & 1 & 1 \\
2 & 2 & 2 \\
3 & 3 & 3
\end{bmatrix}
\begin{bmatrix}
1 & 2 & 3
\end{bmatrix} = \begin{bmatrix}
3 & 6 & 9 \\
6 & 12 & 18 \\
9 & 18 & 27
\end{bmatrix}
\]

Must be square to do powers

\[
\begin{bmatrix}
1 & 2 \\
3 & 4
\end{bmatrix}
\begin{bmatrix}
1 & 2 \\
3 & 4
\end{bmatrix}
\begin{bmatrix}
1 & 2 \\
3 & 4
\end{bmatrix}
\]

\[
\begin{bmatrix}
1 & 2 & 3
\end{bmatrix}
\begin{bmatrix}
1 & 2 & 3
\end{bmatrix}
\begin{bmatrix}
1 & 2 & 3
\end{bmatrix}
\]
Calculate how many seconds elapsed since the start of class

- In helloWorld.m, make variables called `secPerMin`, `secPerHour`, `secPerDay`, `secPerMonth` (assume 30.5 days per month), and `secPerYear` (12 months in year), which have the number of seconds in each time period.
- Assemble a row vector called `secondConversion` that has elements in this order: `secPerYear`, `secPerMonth`, `secPerDay`, `secPerHour`, `secPerMinute`, 1.
- Make a `currentTime` vector by using `clock`
- Compute `elapsedTime` by subtracting `currentTime` from `start`
- Compute $t$ (the elapsed time in seconds) by taking the dot product of `secondConversion` and `elapsedTime` (transpose one of them to get the dimensions right)
Exercise: Vector Operations

```matlab
» secPerMin=60;
» secPerHour=60*secPerMin;
» secPerDay=24*secPerHour;
» secPerMonth=30.5*secPerDay;
» secPerYear=12*secPerMonth;
» secondConversion=[secPerYear secPerMonth ... 
  secPerDay secPerHour secPerMin 1];
» currentTime=clock;
» elapsedTime=currentTime-start;
» t=secondConversion*elapsedTime';
```
Exercise: Vector Operations

Display the current state of your knowledge

• Calculate `currentKnowledge` using the same relationship as before, and the `t` we just calculated:

\[ k = 1 - e^{-t/\tau} \]

• Display the following text:

At this time, I know X% of Matlab
Exercise: Vector Operations

Display the current state of your knowledge

• Calculate \( \text{currentKnowledge} \) using the same relationship as before, and the \( t \) we just calculated:

\[
k = 1 - e^{-t/\tau}
\]

• Display the following text:

At this time, I know \( X\% \) of Matlab

```matlab
>> \text{currentKnowledge}=1-\exp(-t/\tau);
>> \text{disp}(['At this time, I know ' ... 
                      \text{num2str(currentKnowledge*100)} '\% of Matlab']);
```
Automatic Initialization

• Initialize a vector of **ones, zeros, or random numbers**
  » `o=ones(1,10)`  
    - row vector with 10 elements, all 1  
  » `z=zeros(23,1)`  
    - column vector with 23 elements, all 0  
  » `r=rand(1,45)`  
    - row vector with 45 elements (uniform [0,1])  
  » `n=nan(1,69)`  
    - row vector of NaNs (useful for representing uninitialized variables)

The general function call is:

```matlab
var=zeros(M,N);
```

Number of rows  Number of columns
Automatic Initialization

• To initialize a linear vector of values use **linspace**
  
  » \( a=linspace(0,10,5) \)
  
  ➢ starts at 0, ends at 10 (inclusive), 5 values

• Can also use colon operator (\( : \))
  
  » \( b=0:2:10 \)
  
  ➢ starts at 0, increments by 2, and ends at or before 10
  ➢ increment can be decimal or negative

  » \( c=1:5 \)
  
  ➢ if increment isn’t specified, default is 1

• To initialize logarithmically spaced values use **logspace**
  
  ➢ similar to **linspace**, but see **help**
Exercise: Vector Functions

Calculate your learning trajectory

• In helloWorld.m, make a linear time vector \( tVec \) that has 10,000 samples between 0 and \( \text{endOfClass} \)

• Calculate the value of your knowledge (call it \( \text{knowledgeVec} \)) at each of these time points using the same equation as before:

\[
k = 1 - e^{-t/\tau}
\]
Exercise: Vector Functions

Calculate your learning trajectory

- In `helloWorld.m`, make a linear time vector `tVec` that has 10,000 samples between 0 and `endOfClass`.
- Calculate the value of your knowledge (call it `knowledgeVec`) at each of these time points using the same equation as before:

\[
k = 1 - e^{-t/\tau}
\]

```matlab
» tVec = linspace(0,endOfClass,10000);
» knowledgeVec=1-exp(-tVec/tau);
```
Vector Indexing

• Matlab indexing starts with 1, not 0
  ➢ We will not respond to any emails where this is the problem.
• a(n) returns the n\textsuperscript{th} element
  \[ a = [13 \ 5 \ 9 \ 10] \]

• The index argument can be a vector. In this case, each element is looked up individually, and returned as a vector of the same size as the index vector.
  » x=[12 13 5 8];
  » a=x(2:3); \rightarrow a=[13 5];
  » b=x(1:end-1); \rightarrow b=[12 13 5];
Matrix Indexing

- Matrices can be indexed in two ways
  - using **subscripts** (row and column)
  - using linear **indices** (as if matrix is a vector)
- Matrix indexing: **subscripts** or **linear indices**

- Picking submatrices
  
  » `A = rand(5)` % shorthand for 5x5 matrix
  
  » `A(1:3,1:2)` % specify contiguous submatrix
  
  » `A([1 5 3], [1 4])` % specify rows and columns
To select rows or columns of a matrix, use the :

\[ c = \begin{bmatrix} 12 & 5 \\ -2 & 13 \end{bmatrix} \]

- \( d = c(1,:) \); \( d = [12 \ 5] \);
- \( e = c(:,2) \); \( e = [5;13] \);
- \( c(2,:) = [3 \ 6] \); % replaces second row of \( c \)
Advanced Indexing 2

• MATLAB contains functions to help you find desired values within a vector or matrix
  » vec = [5 3 1 9 7]

• To get the minimum value and its index:
  » [minVal, minInd] = min(vec);
  ➢ max works the same way

• To find any the indices of specific values or ranges
  » ind = find(vec == 9);
  » ind = find(vec > 2 & vec < 6);
  ➢ find expressions can be very complex, more on this later

• To convert between subscripts and indices, use ind2sub, and sub2ind. Look up help to see how to use them.
Exercise: Indexing

When will you know 50% of Matlab?

• First, find the index where $\text{knowledgeVec}$ is closest to 0.5. Mathematically, what you want is the index where the value of $|\text{knowledgeVec} - 0.5|$ is at a minimum (use $\text{abs}$ and $\text{min}$).

• Next, use that index to look up the corresponding time in $\text{tVec}$ and name this time $\text{halfTime}$.

• Finally, display the string: I will know half of Matlab after X days. Convert $\text{halfTime}$ to days by using $\text{secPerDay}$.
Exercise: Indexing

When will you know 50% of Matlab?

- First, find the index where `knowledgeVec` is closest to 0.5. Mathematically, what you want is the index where the value of \( |knowledgeVec - 0.5| \) is at a minimum (use `abs` and `min`).
- Next, use that index to look up the corresponding time in `tVec` and name this time `halfTime`.
- Finally, display the string: I will know half of Matlab after X days
  Convert `halfTime` to days by using `secPerDay`

```matlab
[val,ind]=min(abs(knowledgeVec-0.5));
halfTime=tVec(ind);
disp(['I will know half of Matlab after ' ... num2str(halfTime/secPerDay) ' days']);
```
Outline

(1) Getting Started
(2) Scripts
(3) Making Variables
(4) Manipulating Variables
(5) Basic Plotting
Plotting

• Example
  » \texttt{x=linspace(0,4*\pi,10)};
  » \texttt{y=sin(x)};

• Plot values against their index
  » \texttt{plot(y)};

• Usually we want to plot \( y \) versus \( x \)
  » \texttt{plot(x,y)};

MATLAB makes visualizing data fun and easy!
What does plot do?

- **plot** generates dots at each \((x,y)\) pair and then connects the dots with a line.
- To make plot of a function look smoother, evaluate at more points:
  ```matlab
  x=linspace(0,4*pi,1000);
  plot(x,sin(x));
  ```
- \(x\) and \(y\) vectors must be same size or else you’ll get an error:
  ```matlab
  plot([1 2], [1 2 3])
  ```
  ➢ error!!

10 x values:

1000 x values:
Exercise: Plotting

Plot the learning trajectory

- In helloWorld.m, open a new figure (use `figure`)
- Plot the knowledge trajectory using `tVec` and `knowledgeVec`. When plotting, convert `tVec` to days by using `secPerDay`
- Zoom in on the plot to verify that `halfTime` was calculated correctly
Exercise: Plotting

Plot the learning trajectory

- In helloWorld.m, open a new figure (use `figure`)
- Plot the knowledge trajectory using `tVec` and `knowledgeVec`. When plotting, convert `tVec` to days by using `secPerDay`
- Zoom in on the plot to verify that `halfTime` was calculated correctly

```matlab
» figure
» plot(tVec/secPerDay, knowledgeVec);
```
Matlab Tutorial

- Matlab tutorials
  - http://www.engin.umich.edu/group/ctm/basic/basic.html
  - http://www.engin.umich.edu/group/ctm/model/model.html
- Tutorials included in Matlab
Homework 1: chapter 1

- 1.1
- 1.3
- 1.7
- 1.10
References

- Palm, W. J., Modeling, Analysis, and Control of Dynamic Systems
- Matlab slides are from: Lecture 1: Variables, Scripts, and Operations, by Danilo Šćepanović, IAP 2010 Course, MIT