Math 306 - Numerical Methods

Introduction to MATLAB

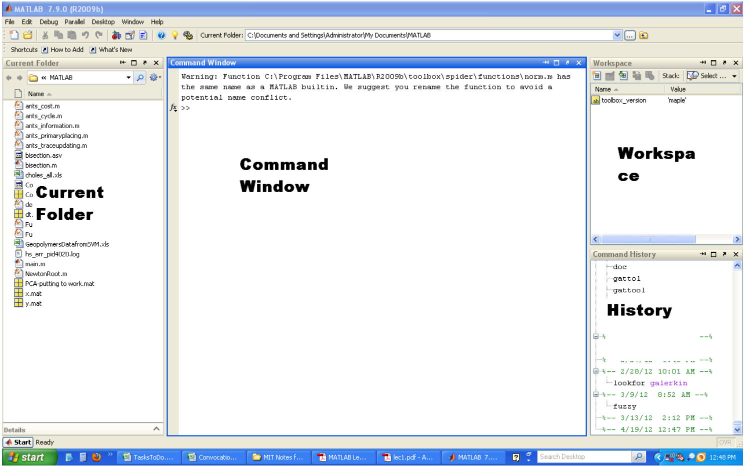
Sqn Ldr Dr Athar Kharal

Humanities and Science Department
College of Aeronautical Engineering
PAF Academy Risalpur

Outline

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

The MATLAB Desktop



Name	Value
toolbox_version	'maple'

```
Warning: Function C:\Program Files\MATLAB\R2009b\toolbox\spider\functions\norm.m has
the same name as a MATLAB builtin. We suggest you rename the function to avoid a
potential name conflict.
>>
```

```
doc
gattol
gattool
%                        --%
%                        --%
%-- 2/28/12 10:01 AM --%
lookfor galerkin
%-- 3/9/12  8:52 AM --%
fuzzy
%-- 3/13/12  2:12 PM --%
%-- 4/19/12 12:47 PM --%
```

Getting Started

Customization

File -> Preferences

Allows you personalize your MATLAB experience

MATLAB Basics

- MATLAB can be thought of as a super-powerful calculator

MATLAB Basics

- MATLAB can be thought of as a super-powerful calculator
  - With many many more buttons (built-in functions)

MATLAB Basics

- MATLAB can be thought of as a super-powerful calculator
  - With many many more buttons (built-in functions)
  - And graphing

MATLAB Basics

- MATLAB can be thought of as a super-powerful calculator
  - With many many more buttons (built-in functions)
  - And graphing
- It is not a symbolic processor

MATLAB Basics

- MATLAB can be thought of as a super-powerful calculator
  - With many many more buttons (built-in functions)
  - And graphing
- It is not a symbolic processor
- It is a programming language

MATLAB Basics

- MATLAB can be thought of as a super-powerful calculator
  - With many many more buttons (built-in functions)
  - And graphing
- It is not a symbolic processor
- It is a programming language
  - MATLAB is an interpreted language,

MATLAB Basics

- MATLAB can be thought of as a super-powerful calculator
    - With many many more buttons (built-in functions)
    - And graphing
- It is not a symbolic processor
- It is a programming language
    - MATLAB is an interpreted language,
    - Commands are executed line by line

MATLAB Basics

- MATLAB can be thought of as a super-powerful calculator
  - With many many more buttons (built-in functions)
  - And graphing
- It is not a symbolic processor
- It is a programming language
  - MATLAB is an interpreted language,
  - Commands are executed line by line
  - Like the computer languages Scheme and BASIC

Conversing with MATLAB

- who
  MATLAB replies with the variables in your workspace

Conversing with MATLAB

- who
  MATLAB replies with the variables in your workspace
- what
  MATLAB replies with the current directory and MATLAB files in the directory

Conversing with MATLAB

- who

  MATLAB replies with the variables in your workspace
- what

  MATLAB replies with the current directory and MATLAB files in the directory
- why

  See yourself!!

Conversing with MATLAB

- who

  MATLAB replies with the variables in your workspace
- what

  MATLAB replies with the current directory and MATLAB files in the directory
- why

  See yourself!!
- help

  The most important function for learning MATLAB on your own

Conversing with MATLAB

- who
  MATLAB replies with the variables in your workspace
- what
  MATLAB replies with the current directory and MATLAB files in the directory
- why
  See yourself!!
- help
  The most important function for learning MATLAB on your own
- More on help later

Variable Types

- MATLAB is a weakly typed language

Variable Types

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

Variable Types

- MATLAB is a weakly typed language
- No need to initialize variables!
- MATLAB supports various types, the most often used are

Variable Types

- MATLAB is a weakly typed language
- No need to initialize variables!
- MATLAB supports various types, the most often used are
    - »3.84

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'

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

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 arrays or matrices of doubles or chars

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 arrays or matrices of doubles or chars
- Other types are also supported: complex, symbolic, 16-bit and 8 bit integers, etc.

Naming variables

- To create a variable, simply assign a value to a name:

```
»var1=3.14
»myString='hello world'
```

Naming variables

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

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

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

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! (var1is different from Var1)

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! (var1is different from Var1)
- Built-in variables

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! (var1is different from Var1)
- Built-in variables
  - i and j can be used to indicate complex numbers

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! (var1is different from Var1)
- Built-in variables
  - i and j can be used to indicate complex numbers
  - pi has the value 3.1415926...

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! (var1is different from Var1)
- Built-in variables
  - 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)

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! (var1is different from Var1)
- Built-in variables
  - 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

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! (var1is different from Var1)
- Built-in variables
  - 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'

Hello World

- Here are several flavors of Hello World to introduce MATLAB

Hello World

- Here are several flavors of Hello World to introduce MATLAB
- MATLAB will display strings automatically

Hello World

- Here are several flavors of Hello World to introduce MATLAB
- MATLAB will display strings automatically
  - »'Hello 6.094'

Hello World

- Here are several flavors of Hello World to introduce MATLAB
- MATLAB will display strings automatically
  - »'Hello 6.094'
  - To remove "ans=", use disp()

Hello World

- Here are several flavors of Hello World to introduce MATLAB
- MATLAB will display strings automatically
  - »'Hello 6.094'
  - To remove "ans=", use disp()
    - »disp('Hello6.094')

Hello World

- Here are several flavors of Hello World to introduce MATLAB
- MATLAB will display strings automatically
  - »'Hello 6.094'
  - To remove "ans=", use disp()
    - »disp('Hello6.094')
  - sprintf() allows you to mix strings with variables
    »class=6.094;
    »disp(sprintf('Hello%g', class))

Hello World

- Here are several flavors of Hello World to introduce MATLAB
- MATLAB will display strings automatically
  - »'Hello 6.094'
  - To remove "ans=", use disp()
    - »disp('Hello6.094')
  - sprintf() allows you to mix strings with variables
    »class=6.094;
    »disp(sprintf('Hello%g', class))
    - The format is C-syntax

Scalars

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

Scalars

- A variable can be given a value explicitly
  »a = 10
  which shows up in workspace!
- Or as a function of explicit values and existing variables
  »c = 1.3*45-2*a

Scalars

- A variable can be given a value explicitly
  »a = 10
  which 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
  »m= 13/3;

Arrays

- Like other programming languages, arrays are an important part of MATLAB

Arrays

- Like other programming languages, arrays are an important part of MATLAB
- Two types of arrays

Arrays

- Like other programming languages, arrays are an important part of MATLAB
- Two types of arrays
    - Matrix of numbers (either double or complex)

Arrays

- Like other programming languages, arrays are an important part of MATLAB
- Two types of arrays
  - Matrix of numbers (either double or complex)
  - Cell array of objects (more advanced data structure)

Arrays

- Like other programming languages, arrays are an important part of MATLAB
- Two types of arrays
  - Matrix of numbers (either double or complex)
  - Cell array of objects (more advanced data structure)
- MATLAB makes vectors easy!That's its power!

Row & Column Vectors

- Row vector: comma or space separated values between brackets

  ```
  »row = [1 2 5.4 -6.6];
  »row = [1, 2, 5.4, -6.6];
  ```

Row & Column Vectors

- Row vector: comma or space separated values between brackets

  »row = [1 2 5.4 -6.6];

  »row = [1, 2, 5.4, -6.6];

- Column vector: semicolon separated values between brackets

  »column = [4;2;7;4];

Matrices

- Make matrices like vectors

Matrices

- Make matrices like vectors
  - Element by element
    »a= [1 2;3 4];

Matrices

- Make matrices like vectors
  - Element by element

    »a= [1 2;3 4];
  - By concatenating vectors or matrices (dimension matters)

    »a = [1 2];
    »b = [3 4];
    »c = [5;6];
    »d = [a;b];

Matrices

- Make matrices like vectors
  - Element by element
    »a= [1 2;3 4];
  - By concatenating vectors or matrices (dimension matters)
    »a = [1 2];
    »b = [3 4];
    »c = [5;6];
    »d = [a;b];
  - »e = [d c];

Matrices

- Make matrices like vectors
  - Element by element
    »a= [1 2;3 4];
  - 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];[ab a]];

Matrices

- Make matrices like vectors
  - Element by element
    »a= [1 2;3 4];
  - 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];[ab a]];
  - Workout the output for f ?

Matrices

- Make matrices like vectors
  - Element by element
    »a= [1 2;3 4];
  - 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];[ab a]];
  - Workout the output for f ?
  -

```
1   2   5   1   2   5
3   4   6   3   4   6
1   2   3   4   1   2
```

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.matfile is in the current directory

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.matfile is in the current directory
- Default working directory is

  »\MATLAB\work

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.matfile is in the current directory
- Default working directory is

  »\MATLAB\work
- Create own folder and change working directory to it

  »MyDocuments\6.094\day1

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.matfile is in the current directory
- Default working directory is
  »\MATLAB\work
- Create own folder and change working directory to it
  »MyDocuments\6.094\day1
- Use clear to remove variables from environment
  »clear a b
  Now look at workspace, the variablesa andb are gone

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.matfile is in the current directory
- Default working directory is
  »\MATLAB\work
- Create own folder and change working directory to it
  »MyDocuments\6.094\day1
- Use clear to remove variables from environment
  »clear a b
  Now look at workspace, the variablesa andb are gone
- Use load to load variable bindings into the environment
  »load myfile
  look at workspace, the variablesa andb are back

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.matfile is in the current directory
- Default working directory is
  »\MATLAB\work
- Create own folder and change working directory to it
  »MyDocuments\6.094\day1
- Use clear to remove variables from environment
  »clear a b
  Now look at workspace, the variablesa andb are gone
- Use load to load variable bindings into the environment
  »load myfile
  look at workspace, the variablesa andb are back
- Can do the same for entire environment
  »save myenv; clear all; load myenv;

Exercise: Variables

- Do the following 5 things on your notebooks:

Exercise: Variables

- Do the following 5 things on your notebooks:
    - Create the variable r as a row vector with values 1 4 7 10 13

Exercise: Variables

- Do the following 5 things on your notebooks:
  - Create the variable r as a row vector with values 1 4 7 10 13
  - Create the variable c as a column vector with values 13 10 7 4 1

Exercise: Variables

- Do the following 5 things on your notebooks:
  - Create the variable r as a row vector with values 1 4 7 10 13
  - Create the variable c as a column vector with values 13 10 7 4 1
  - Save these two variables to file varEx

Exercise: Variables

- Do the following 5 things on your notebooks:
  - Create the variable r as a row vector with values 1 4 7 10 13
  - Create the variable c as a column vector with values 13 10 7 4 1
  - Save these two variables to file varEx
  - clear the workspace

Exercise: Variables

- Do the following 5 things on your notebooks:
    - Create the variable r as a row vector with values 1 4 7 10 13
    - Create the variable c as a column vector with values 13 10 7 4 1
    - Save these two variables to file varEx
    - clear the workspace
    - load the two variables you just created

Exercise: Variables

- Do the following 5 things on your notebooks:
  - Create the variable r as a row vector with values 1 4 7 10 13
  - Create the variable c as a column vector with values 13 10 7 4 1
  - Save these two variables to file varEx
  - clear the workspace
  - load the two variables you just created
- And here are the answers:

```
»r=[1 4 7 10 13];
»c=[13; 10; 7; 4; 1];
»save varExr c
»clear r c
»load varEx
```

Basic Scalar Operations

- Arithmetic operations (+, -, *, /)

```
»7/45
»(1+i)*(2+i)
»1 / 0
»0 / 0
```

Basic Scalar Operations

- Arithmetic operations (+, -, *, /)

```
»7/45
»(1+i)*(2+i)
»1 / 0
»0 / 0
```

- Exponentiation (^)

```
»4^2
»(3+4*j)^2
```

Basic Scalar Operations

- Arithmetic operations (+, -, *, /)

  ```
  »7/45
  »(1+i)*(2+i)
  »1 / 0
  »0 / 0
  ```

- Exponentiation (^)

  ```
  »4^2
  »(3+4*j)^2
  ```

- Complicated expressions, use parentheses

  ```
  »((2+3)*3)^0.1
  ```

Basic Scalar Operations

- Arithmetic operations (+, -, *, /)

  ```
  »7/45
  »(1+i)*(2+i)
  »1 / 0
  »0 / 0
  ```

- Exponentiation (^)

  ```
  »4^2
  »(3+4*j)^2
  ```

- Complicated expressions, use parentheses

  ```
  »((2+3)*3)^0.1
  ```

- Multiplication is NOT implicit given parentheses

  »3(1+0.7) gives an error

Basic Scalar Operations

- Arithmetic operations (+, -, *, /)
  ```
  »7/45
  »(1+i)*(2+i)
  »1 / 0
  »0 / 0
  ```
- Exponentiation (^)
  ```
  »4^2
  »(3+4*j)^2
  ```
- Complicated expressions, use parentheses
  ```
  »((2+3)*3)^0.1
  ```
- Multiplication is NOT implicit given parentheses
  `»3(1+0.7)` gives an error
- To clear cluttered command window
  ```
  »clc
  ```

Built-in Functions

- MATLAB has an enormous library of built-in functions

Built-in Functions

- MATLAB has an enormous library of built-in functions
- Call using parentheses – passing parameter to function

```
»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);
```

Help/Docs

- To get info on how to use a function:
  »help sin

Help/Docs

- To get info on how to use a function:
  »help sin
- Help also contains related functions

Help/Docs

- To get info on how to use a function:
  »help sin
- Help also contains related functions
- To get a nicer version of help with examples and easy-to-read descriptions:
  »doc sin

Help/Docs

- To get info on how to use a function:
  »help sin
- Help also contains related functions
- 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
  »lookfor hyperbolic
  One-word description of what you're looking for

Exercise: Scalars

Verify that $e^{ix} = \cos(x) + i\sin(x)$ for a few values of x.

```
»x = pi/3;
»a = exp(i*x)
»b = cos(x)+ i*sin(x)
»a-b
```

size & length

- You can tell the difference between a row and a column vector by:

size & length

- You can tell the difference between a row and a column vector by:
  - Looking in the 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

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

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

transpose

- The transpose operators turns a column vector into a row vector and vice versa
  ```
  »a = [1 2 3 4]
  »transpose(a)
  ```

transpose

- The transpose operators turns a column vector into a row vector and vice versa
  »a = [1 2 3 4]
  »transpose(a)
- Can use dot-apostrophe as short-cut
  »a.’

transpose

- The transpose operators turns a column vector into a row vector and vice versa
  »a = [1 2 3 4]
  »transpose(a)
- Can use dot-apostrophe as short-cut
  »a.'
- The apostrophe gives the Hermitian-transpose, i.e. transposes and conjugates all complex numbers
  »a = [1+j 2+3*j]
  »a'
  »a.'

transpose

- The transpose operators turns a column vector into a row vector and vice versa

  ```
  »a = [1 2 3 4]
  »transpose(a)
  ```

- Can use dot-apostrophe as short-cut

  ```
  »a.'
  ```

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

  ```
  »a = [1+j 2+3*j]
  »a'
  »a.'
  ```

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

Addition and Subtraction

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

$$\begin{array}{cc} & [12 \quad 3 \quad 32 \quad -11] \\ + & [2 \quad 11 \quad -30 \quad 32] \\ \hline & [14 \quad 14 \quad 2 \quad 21] \end{array} \qquad \begin{bmatrix} 12 \\ 1 \\ -10 \\ 0 \end{bmatrix} - \begin{bmatrix} 3 \\ -1 \\ 13 \\ 33 \end{bmatrix} = \begin{bmatrix} 9 \\ 2 \\ -23 \\ -33 \end{bmatrix}$$

Addition and Subtraction

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

$$\begin{array}{cc} & [12 \quad 3 \quad 32 \quad -11] \\ + & [2 \quad 11 \quad -30 \quad 32] \\ \hline & [14 \quad 14 \quad 2 \quad 21] \end{array} \qquad \begin{bmatrix} 12 \\ 1 \\ -10 \\ 0 \end{bmatrix} - \begin{bmatrix} 3 \\ -1 \\ 13 \\ 33 \end{bmatrix} = \begin{bmatrix} 9 \\ 2 \\ -23 \\ -33 \end{bmatrix}$$

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

Addition and Subtraction

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

$$\begin{array}{rl} & [12 \;\; 3 \;\; 32 \;\; -11] \\ + & [2 \;\; 11 \;\; -30 \;\; 32] \\ \hline & [14 \;\; 14 \;\; 2 \;\; 21] \end{array} \qquad \begin{bmatrix} 12 \\ 1 \\ -10 \\ 0 \end{bmatrix} - \begin{bmatrix} 3 \\ -1 \\ 13 \\ 33 \end{bmatrix} = \begin{bmatrix} 9 \\ 2 \\ -23 \\ -33 \end{bmatrix}$$

- The following would give an error

  »c = row + column

- Use the transpose to make sizes compatible

  »c = row'+ column

  »c = row + column'

Addition and Subtraction

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

$$\begin{array}{cc} & [12\;\;3\;\;32\;\;-11] \\ + & [2\;\;11\;\;-30\;\;32] \\ \hline & [14\;\;14\;\;2\;\;21] \end{array} \qquad \begin{bmatrix} 12 \\ 1 \\ -10 \\ 0 \end{bmatrix} - \begin{bmatrix} 3 \\ -1 \\ 13 \\ 33 \end{bmatrix} = \begin{bmatrix} 9 \\ 2 \\ -23 \\ -33 \end{bmatrix}$$

- 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

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)];

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

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 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

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)!
  »a=[1 2 3];b=[4;2;1];
  »a.*b, a./b, a.^b → all errors
  »a.*b', a./b', a.^(b') → all valid

Operators: element-wise

- To do element-wise operations, use the dot. BOTH dimensions must match (unless one is scalar)!
  »a=[1 2 3];b=[4;2;1];
  »a.*b, a./b, a.^b → all errors
  »a.*b', a./b', a.^(b') → all valid
- $$\begin{bmatrix} 1 & 2 & 3 \end{bmatrix} \quad .* \quad \begin{bmatrix} 4 \\ 2 \\ 1 \end{bmatrix} \quad = \quad \mathit{ERROR}$$

Operators: element-wise

- To do element-wise operations, use the dot. BOTH dimensions must match (unless one is scalar)!
  »a=[1 2 3];b=[4;2;1];
  »a.*b, a./b, a.^b → all errors
  »a.*b', a./b', a.^(b') → all valid
- $$\begin{bmatrix} 1 & 2 & 3 \end{bmatrix} \quad .* \quad \begin{bmatrix} 4 \\ 2 \\ 1 \end{bmatrix} \quad = \quad \mathit{ERROR}$$
- $$\begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} \quad .* \quad \begin{bmatrix} 4 \\ 2 \\ 1 \end{bmatrix} \quad = \quad \begin{bmatrix} 4 \\ 4 \\ 3 \end{bmatrix}$$
  $$3 \times 1 \quad .* \quad 3 \times 1 \quad = \quad 3 \times 1$$

Operators: element-wise (cont'd)

$$\begin{bmatrix} 1 & 1 & 1 \\ 2 & 2 & 2 \\ 3 & 3 & 3 \end{bmatrix} \quad .* \quad \begin{bmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \\ 1 & 2 & 3 \end{bmatrix} \quad = \quad \begin{bmatrix} 1 & 2 & 3 \\ 2 & 4 & 6 \\ 3 & 6 & 9 \end{bmatrix}$$

$$3 \times 3 \quad .* \quad 3 \times 3 \quad = \quad 3 \times 3$$

Operators: standard

- Multiplication can be done in a standard way or element-wise

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!!

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 (ˆ) implicitly uses $*$

  Can only be done on square matrices or scalars

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 (^) implicitly uses $*$
  Can only be done on square matrices or scalars
- Left and right division (/ \) is same as multiplying by inverse
  Recommendation: just multiply by inverse (more on this later)

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 (^) implicitly uses $*$

  Can only be done on square matrices or scalars
- Left and right division (/ \) is same as multiplying by inverse

  Recommendation: just multiply by inverse (more on this later)
-

$$\begin{bmatrix} 1 & 2 & 3 \end{bmatrix} * \begin{bmatrix} 4 \\ 2 \\ 1 \end{bmatrix} = 11$$

$$1 \times 3 * 3 \times 1 = 1 \times 1$$

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 (^) implicitly uses $*$
  Can only be done on square matrices or scalars
- Left and right division (/ \) is same as multiplying by inverse
  Recommendation: just multiply by inverse (more on this later)
- $$\begin{bmatrix} 1 & 2 & 3 \end{bmatrix} * \begin{bmatrix} 4 \\ 2 \\ 1 \end{bmatrix} = 11$$
  $$1 \times 3 * 3 \times 1 = 1 \times 1$$
- $$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \hat{}2 = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} * \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

Operators: standard (cont'd)

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

$$3 \times 3 \quad * \quad 3 \times 3 \quad = \quad 3 \times 3$$

Exercise: Vector Operations

- Find the inner product between [1 2 3] and [3 5 4]

  ```
  »a=[1 2 3]*[3 5 4]'
  ```

Exercise: Vector Operations

- Find the inner product between [1 2 3] and [3 5 4]
  »a=[1 2 3]*[3 5 4]'
- Multiply the same two vectors element-wise
  »b=[1 2 3].*[3 5 4]

Exercise: Vector Operations

- Find the inner product between [1 2 3] and [3 5 4]
  »a=[1 2 3]*[3 5 4]'
- Multiply the same two vectors element-wise
  »b=[1 2 3].*[3 5 4]
- Calculate the natural log of each element of the resulting vector
  »c=log(b)

Automatic Initialization

- Initialize a vector of ones, zeros, or random numbers
  »o=ones(1,10)
  row vector with 10 elements, all 1

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

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])

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)

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:

$$var = zeros(M, N)$$

where M =Number of rows and N =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

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

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

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

Exercise: Vector Functions

- Make a vector that has $10,000$ samples of $f(x) = e^{-x}\cos(x)$, for x between 0 and 10.

Exercise: Vector Functions

- Make a vector that has 10,000 samples of $f(x) = e^{-x}\cos(x)$, for x between 0 and 10.
- Solution

Exercise: Vector Functions

- Make a vector that has 10,000 samples of $f(x) = e^{-x}\cos(x)$, for x between 0 and 10.
- Solution
  - »x = linspace(0,10,10000);

Exercise: Vector Functions

- Make a vector that has $10,000$ samples of $f(x) = e^{-x}\cos(x)$, for x between 0 and 10.
- Solution
  - »x = linspace(0,10,10000);
  - »f = exp(-x).*cos(x);

Vector Indexing

- MATLAB indexing starts with 1, not 0

Vector Indexing

- MATLAB indexing starts with 1, not 0
- a(n)

  returns the *n*th element

Vector Indexing

- MATLAB indexing starts with 1, not 0
- a(n)
  returns the *n*th element
- 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);
```

Vector Indexing

- MATLAB indexing starts with 1, not 0
- a(n)
  returns the *n*th element
- 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);
```

Vector Indexing

- MATLAB indexing starts with 1, not 0
- a(n)

  returns the *n*th element
- 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); → a=[13 5];

Vector Indexing

- MATLAB indexing starts with 1, not 0
- a(n)
  returns the *n*th element
- 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); → a=[13 5];
  »b=x(1:end-1);

Vector Indexing

- MATLAB indexing starts with 1, not 0
- a(n)
  returns the *n*th element
- 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); → a=[13 5];
  »b=x(1:end-1); → b=[12 13 5];

Matrix Indexing

- Matrices can be indexed in two ways

Matrix Indexing

- Matrices can be indexed in two ways
  - using subscripts (row and column)

$$\begin{array}{ccccc} b(1,1) & \rightarrow & \left[\begin{array}{cc} 14 & 33 \\ 9 & 8 \end{array}\right] & \leftarrow & b(1,2) \\ b(2,1) & \rightarrow & & \leftarrow & b(2,2) \end{array}$$

Matrix Indexing

- Matrices can be indexed in two ways
  - using subscripts (row and column)

$$\begin{array}{ccccc} b(1,1) & \rightarrow & \left[\begin{array}{cc} 14 & 33 \\ 9 & 8 \end{array}\right] & \leftarrow & b(1,2) \\ b(2,1) & \rightarrow & & \leftarrow & b(2,2) \end{array}$$

  - using linear indices (as if matrix is a vector)

$$\begin{array}{ccccc} b(1) & \rightarrow & \left[\begin{array}{cc} 14 & 33 \\ 9 & 8 \end{array}\right] & \leftarrow & b(3) \\ b(2) & \rightarrow & & \leftarrow & b(4) \end{array}$$

Matrix Indexing

- Matrices can be indexed in two ways
  - using subscripts (row and column)

$$\begin{array}{ccc} b(1,1) \rightarrow & \left[\begin{array}{cc} 14 & 33 \\ 9 & 8 \end{array}\right] & \leftarrow b(1,2) \\ b(2,1) \rightarrow & & \leftarrow b(2,2) \end{array}$$

  - using linear indices (as if matrix is a vector)

$$\begin{array}{ccc} b(1) \rightarrow & \left[\begin{array}{cc} 14 & 33 \\ 9 & 8 \end{array}\right] & \leftarrow b(3) \\ b(2) \rightarrow & & \leftarrow b(4) \end{array}$$

- Picking submatrices

Matrix Indexing

- Matrices can be indexed in two ways
  - using subscripts (row and column)

$$\begin{array}{ccccc} b\,(1,1) & \rightarrow & \left[\begin{array}{cc} 14 & 33 \\ 9 & 8 \end{array}\right. & \left.\begin{array}{c} \\ \\ \end{array}\right] \leftarrow & b\,(1,2) \\ b\,(2,1) & \rightarrow & & \leftarrow & b\,(2,2) \end{array}$$

  - using linear indices (as if matrix is a vector)

$$\begin{array}{ccccc} b\,(1) & \rightarrow & \left[\begin{array}{cc} 14 & 33 \\ 9 & 8 \end{array}\right] & \leftarrow & b\,(3) \\ b\,(2) & \rightarrow & & \leftarrow & b\,(4) \end{array}$$

- Picking submatrices
  - »A = rand(5) % shorthand for 5x5 matrix

Matrix Indexing

- Matrices can be indexed in two ways
  - using subscripts (row and column)

$$\begin{array}{ccccc} b\,(1,1) & \rightarrow & \left[\begin{array}{cc} 14 & 33 \\ 9 & 8 \end{array}\right. & \left.\right] & \leftarrow & b\,(1,2) \\ b\,(2,1) & \rightarrow & & & \leftarrow & b\,(2,2) \end{array}$$

  - using linear indices (as if matrix is a vector)

$$\begin{array}{ccccc} b\,(1) & \rightarrow & \left[\begin{array}{cc} 14 & 33 \\ 9 & 8 \end{array}\right. & \left.\right] & \leftarrow & b\,(3) \\ b\,(2) & \rightarrow & & & \leftarrow & b\,(4) \end{array}$$

- Picking submatrices
  - »A = rand(5) % shorthand for 5x5 matrix
  - »A(1:3,1:2) % specify contiguous submatrix

Matrix Indexing

- Matrices can be indexed in two ways
  - using subscripts (row and column)

$$\begin{matrix} b(1,1) & \rightarrow \\ b(2,1) & \rightarrow \end{matrix} \quad \begin{bmatrix} 14 & 33 \\ 9 & 8 \end{bmatrix} \quad \begin{matrix} \leftarrow & b(1,2) \\ \leftarrow & b(2,2) \end{matrix}$$

  - using linear indices (as if matrix is a vector)

$$\begin{matrix} b(1) & \rightarrow \\ b(2) & \rightarrow \end{matrix} \quad \begin{bmatrix} 14 & 33 \\ 9 & 8 \end{bmatrix} \quad \begin{matrix} \leftarrow & b(3) \\ \leftarrow & b(4) \end{matrix}$$

- 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

Advanced Indexing 1

- The index argument can be a matrix. In this case, each element is looked up individually, and returned as a matrix of the same size as the index matrix.

Advanced Indexing 1

- The index argument can be a matrix. In this case, each element is looked up individually, and returned as a matrix of the same size as the index matrix.
  - »a=[-1 10 3 -2];
    »b=a([1 2 4;3 4 2]);
    Workout!

Advanced Indexing 1

- The index argument can be a matrix. In this case, each element is looked up individually, and returned as a matrix of the same size as the index matrix.
  - »a=[-1 10 3 -2];
    »b=a([1 2 4;3 4 2]);
    Workout!
  - $b = \begin{bmatrix} -1 & 10 & -2 \\ 3 & -2 & 10 \end{bmatrix}$

Advanced Indexing 1

- The index argument can be a matrix. In this case, each element is looked up individually, and returned as a matrix of the same size as the index matrix.
  - »a=[-1 10 3 -2];
    »b=a([1 2 4;3 4 2]);
    Workout!
  - $b = \begin{bmatrix} -1 & 10 & -2 \\ 3 & -2 & 10 \end{bmatrix}$
- To select rows or columns of a matrix, use the :

Advanced Indexing 1

- The index argument can be a matrix. In this case, each element is looked up individually, and returned as a matrix of the same size as the index matrix.
    - »a=[-1 10 3 -2];
      »b=a([1 2 4;3 4 2]);
      Workout!
    - $b = \begin{bmatrix} -1 & 10 & -2 \\ 3 & -2 & 10 \end{bmatrix}$
- To select rows or columns of a matrix, use the :
- $c = \begin{bmatrix} 12 & 5 \\ -2 & 13 \end{bmatrix}$

Advanced Indexing 1

- The index argument can be a matrix. In this case, each element is looked up individually, and returned as a matrix of the same size as the index matrix.
  - »a=[-1 10 3 -2];
    »b=a([1 2 4;3 4 2]);
    Workout!
  - $b = \begin{bmatrix} -1 & 10 & -2 \\ 3 & -2 & 10 \end{bmatrix}$
- To select rows or columns of a matrix, use the :
- $c = \begin{bmatrix} 12 & 5 \\ -2 & 13 \end{bmatrix}$
- »d=c(1,:); $\Rrightarrow$ d=[12 5];

Advanced Indexing 1

- The index argument can be a matrix. In this case, each element is looked up individually, and returned as a matrix of the same size as the index matrix.
  - »a=[-1 10 3 -2];
    »b=a([1 2 4;3 4 2]);
    Workout!
  - $b = \begin{bmatrix} -1 & 10 & -2 \\ 3 & -2 & 10 \end{bmatrix}$
- To select rows or columns of a matrix, use the :
- $c = \begin{bmatrix} 12 & 5 \\ -2 & 13 \end{bmatrix}$
- »d=c(1,:); $\Rrightarrow$ d=[12 5];
- »e=c(:,2); $\Rrightarrow$ e=[5;13];

Advanced Indexing 1

- The index argument can be a matrix. In this case, each element is looked up individually, and returned as a matrix of the same size as the index matrix.
  - »a=[-1 10 3 -2];
    »b=a([1 2 4;3 4 2]);
    Workout!
  - $b = \begin{bmatrix} -1 & 10 & -2 \\ 3 & -2 & 10 \end{bmatrix}$
- To select rows or columns of a matrix, use the :
- $c = \begin{bmatrix} 12 & 5 \\ -2 & 13 \end{bmatrix}$
- »d=c(1,:); $\Rightarrow$ d=[12 5];
- »e=c(:,2); $\Rightarrow$ e=[5;13];
- »c(2,:)=[3 6]; %replaces second row of c

Advanced Indexing (cont'd)

- There are functions to help you find desired values within a vector or matrix e.g.
  »vec= [1 5 3 9 7]

Advanced Indexing (cont'd)

- There are functions to help you find desired values within a vector or matrix e.g.
  »vec= [1 5 3 9 7]
- To get the minimum value and its index:»[minVal,minInd] = min(vec);

Advanced Indexing (cont'd)

- There are functions to help you find desired values within a vector or matrix e.g.
  »vec= [1 5 3 9 7]
- To get the minimum value and its index:»[minVal,minInd] = min(vec);
- To get the maximum value and its index:»[maxVal,maxInd] = max(vec);

Advanced Indexing (cont'd)

- There are functions to help you find desired values within a vector or matrix e.g.
  »vec= [1 5 3 9 7]
- To get the minimum value and its index:»[minVal,minInd] = min(vec);
- To get the maximum value and its index:»[maxVal,maxInd] = max(vec);
- To find any of the indices of specific values or ranges:
  »ind= find(vec== 9);
  »ind= find(vec> 2 & vec< 6);

Advanced Indexing (cont'd)

- There are functions to help you find desired values within a vector or matrix e.g.
  »vec= [1 5 3 9 7]
- To get the minimum value and its index:»[minVal,minInd] = min(vec);
- To get the maximum value and its index:»[maxVal,maxInd] = max(vec);
- To find any of the indices of specific values or ranges:
  »ind= find(vec== 9);
  »ind= find(vec> 2 & vec< 6);
- To convert between subscripts and indices, useind2sub, andsub2ind. Look up help to see how to use them.

Exercise: Vector Indexing

- Evaluate a sine wave at 1,000 points between 0 and 2*pi.

Exercise: Vector Indexing

- Evaluate a sine wave at 1,000 points between 0 and 2*pi.
    - What's the value at
      Index 55
      Indices 100 through 110

Exercise: Vector Indexing

- Evaluate a sine wave at 1,000 points between 0 and 2*pi.
  - What's the value at
    Index 55
    Indices 100 through 110
  - Find the index of
    the minimum value,
    the maximum value, and
    values between -0.001 and 0.001

Exercise: Vector Indexing

- Evaluate a sine wave at 1,000 points between 0 and 2*pi.
    - What's the value at
      Index 55
      Indices 100 through 110
    - Find the index of
      the minimum value,
      the maximum value, and
      values between -0.001 and 0.001
- Workout!

Exercise: Vector Indexing

- Evaluate a sine wave at 1,000 points between 0 and 2*pi.
  - What's the value at
    Index 55
    Indices 100 through 110
  - Find the index of
    the minimum value,
    the maximum value, and
    values between -0.001 and 0.001
- Workout!
-

```
»x = linspace(0,2*pi,1000);
»y=sin(x);
»y(55)
»y(100:110)
»[minVal,minInd]=min(y)
»[maxVal,maxInd]=max(y)
»inds=find(y>-0.001 & y<0.001)
```

BONUS Exercise: Matrices (Workout)

- Make a 3x100 matrix of zeros, and a vector x that has 100 values between 0 and 10

BONUS Exercise: Matrices (Workout)

- Make a 3x100 matrix of zeros, and a vector x that has 100 values between 0 and 10
  - »mat=zeros(3,100);

BONUS Exercise: Matrices (Workout)

- Make a 3x100 matrix of zeros, and a vector x that has 100 values between 0 and 10
  - »mat=zeros(3,100);
  - »x=linspace(0,10,100);

BONUS Exercise: Matrices (Workout)

- Make a 3x100 matrix of zeros, and a vector x that has 100 values between 0 and 10
  - »mat=zeros(3,100);
  - »x=linspace(0,10,100);
- Replace the first row of the matrix with cos(x)

BONUS Exercise: Matrices (Workout)

- Make a 3x100 matrix of zeros, and a vector x that has 100 values between 0 and 10
  - »mat=zeros(3,100);
  - »x=linspace(0,10,100);
- Replace the first row of the matrix with cos(x)
  - »mat(1,:)=cos(x);

BONUS Exercise: Matrices (Workout)

- Make a 3x100 matrix of zeros, and a vector x that has 100 values between 0 and 10
    - »mat=zeros(3,100);
    - »x=linspace(0,10,100);
- Replace the first row of the matrix with cos(x)
    - »mat(1,:)=cos(x);
- Replace the second row of the matrix with log((x+2)^2)

BONUS Exercise: Matrices (Workout)

- Make a 3x100 matrix of zeros, and a vector x that has 100 values between 0 and 10
  - »mat=zeros(3,100);
  - »x=linspace(0,10,100);
- Replace the first row of the matrix with cos(x)
  - »mat(1,:)=cos(x);
- Replace the second row of the matrix with log((x+2)^2)
  - »mat(2,:)=log((x+2).^2);

BONUS Exercise: Matrices (Workout)

- Make a 3x100 matrix of zeros, and a vector x that has 100 values between 0 and 10
  - »mat=zeros(3,100);
  - »x=linspace(0,10,100);
- Replace the first row of the matrix with cos(x)
  - »mat(1,:)=cos(x);
- Replace the second row of the matrix with log((x+2)^2)
  - »mat(2,:)=log((x+2).^2);
- Replace the third row of the matrix with a random vector of the correct size

BONUS Exercise: Matrices (Workout)

- Make a 3x100 matrix of zeros, and a vector x that has 100 values between 0 and 10
  - »mat=zeros(3,100);
  - »x=linspace(0,10,100);
- Replace the first row of the matrix with cos(x)
  - »mat(1,:)=cos(x);
- Replace the second row of the matrix with log((x+2)^2)
  - »mat(2,:)=log((x+2).^2);
- Replace the third row of the matrix with a random vector of the correct size
  - »mat(3,:)=rand(1,100);

BONUS Exercise: Matrices (Workout)

- Make a 3x100 matrix of zeros, and a vector x that has 100 values between 0 and 10
  - »mat=zeros(3,100);
  - »x=linspace(0,10,100);
- Replace the first row of the matrix with cos(x)
  - »mat(1,:)=cos(x);
- Replace the second row of the matrix with log((x+2)^2)
  - »mat(2,:)=log((x+2).^2);
- Replace the third row of the matrix with a random vector of the correct size
  - »mat(3,:)=rand(1,100);
- Use the sum function to compute row and column sums of mat (see help)

BONUS Exercise: Matrices (Workout)

- Make a 3x100 matrix of zeros, and a vector x that has 100 values between 0 and 10
  - »mat=zeros(3,100);
  - »x=linspace(0,10,100);
- Replace the first row of the matrix with cos(x)
  - »mat(1,:)=cos(x);
- Replace the second row of the matrix with log((x+2)^2)
  - »mat(2,:)=log((x+2).^2);
- Replace the third row of the matrix with a random vector of the correct size
  - »mat(3,:)=rand(1,100);
- Use the sum function to compute row and column sums of mat (see help)
  - »rs= sum(mat,2);

BONUS Exercise: Matrices (Workout)

- Make a 3x100 matrix of zeros, and a vector x that has 100 values between 0 and 10
  - »mat=zeros(3,100);
  - »x=linspace(0,10,100);
- Replace the first row of the matrix with cos(x)
  - »mat(1,:)=cos(x);
- Replace the second row of the matrix with log((x+2)^2)
  - »mat(2,:)=log((x+2).^2);
- Replace the third row of the matrix with a random vector of the correct size
  - »mat(3,:)=rand(1,100);
- Use the sum function to compute row and column sums of mat (see help)
  - »rs= sum(mat,2);
  - »cs= sum(mat); % default dimension is 1

Plotting Vectors

- Example

  ```
  »x=linspace(0,4*pi,10);
  »y=sin(x);
  ```

Plotting Vectors

- Example

```
»x=linspace(0,4*pi,10);
»y=sin(x);
```

- Plot values against their index

```
»plot(y);
```

Plotting Vectors

- Example

  ```
  »x=linspace(0,4*pi,10);
  »y=sin(x);
  ```

- Plot values against their index

  ```
  »plot(y);
  ```

- Usually we want to plot y versus x

  ```
  »plot(x,y);
  ```

Plotting Vectors

- Example

  ```
  »x=linspace(0,4*pi,10);
  »y=sin(x);
  ```
- Plot values against their index

  ```
  »plot(y);
  ```
- Usually we want to plot y versus x

  ```
  »plot(x,y);
  ```

Plotting Vectors

- Example

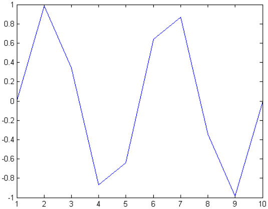
```
»x=linspace(0,4*pi,10);
»y=sin(x);
```

- Plot values against their index

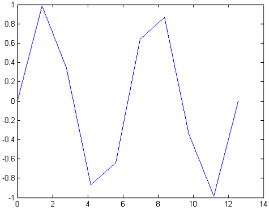
```
»plot(y);
```

- Usually we want to plot y versus x

```
»plot(x,y);
```



Index based



x-values based

What does plot do?

- plot generates dots at each(x,y) pair and then connects the dots with a line

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

  ```
  »x=linspace(0,4*pi,1000);
  »plot(x,sin(x));
  ```

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

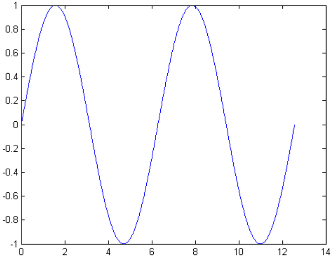
```
»x=linspace(0,4*pi,1000);
»plot(x,sin(x));
```

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

  »x=linspace(0,4*pi,1000);

  »plot(x,sin(x));



- x and y vectors must be same size or else there is an error

  plot([1 2], [1 2 3])

Plot Options

- Can change the line color, marker style, and line style by adding a string argument

```
»plot(x,y,'g.-');
```

Plot Options

- Can change the line color, marker style, and line style by adding a string argument
  »plot(x,y,'g.-');
- Can plot without connecting the dots by omitting line style argument
  »plot(x,y,'.')

Plot Options

- Can change the line color, marker style, and line style by adding a string argument
  »plot(x,y,'g.-');
- Can plot without connecting the dots by omitting line style argument
  »plot(x,y,'.')
- Look at help document ofplot for a full list of colors, markers, and linestyles

Other Useful plot Commands

- To plot two lines on the same graph

  »hold on;

Other Useful plot Commands

- To plot two lines on the same graph
  »hold on;
- To plot on a new figure
  »figure;
  »plot(x,y);

Other Useful plot Commands

- To plot two lines on the same graph
  »hold on;
- To plot on a new figure
  »figure;
  »plot(x,y);
- Play with the figure GUI to learn more on how to
  add axis labels
  add a title
  add a grid
  zoom in/zoom out

Exercise: Plotting

- Plotf(x) = e^x*cos(x) on the intervalx = [0 10]. Use a red solid line with a suitable number of points to get a good resolution.

Exercise: Plotting

- Plot$f(x) = e\hat{}x*cos(x)$ on the interval$x = [0\ 10]$. Use a red solid line with a suitable number of points to get a good resolution.
- Work it out

Exercise: Plotting

- Plot$f(x) = e\hat{}x*\cos(x)$ on the interval$x = [0\ 10]$. Use a red solid line with a suitable number of points to get a good resolution.
- Work it out
-
  ```
  »x=0:.01:10;
  »plot(x,exp(x).*cos(x),'r');
  ```

Thank you