Outline

1 Basics
   • Introduction
   • Arithmetics

2 Variables
   • Vectors
   • Matrices

3 Operators and operations
   • Element-wise Operators
   • Automatic initialization
   • Indexing

4 Plotting
   • Plot properties
   • 3D Line Plots
   • surf and contour

5 Statistics

6 Scripting

7 Linear Algebra

8 Debugging
Outline

1 Basics
   - Introduction
   - Arithmetics

2 Variables
   - Vectors
   - Matrices

3 Operators and operations
   - Element-wise Operators
   - Automatic initialization
   - Indexing

4 Plotting
   - Plot properties
   - 3D Line Plots
   - surf and contour

5 Statistics

6 Scripting

7 Linear Algebra

8 Debugging

Waqar Qureshi (AIT)
Modeling for WREE
February 11, 2014

Basics

What is a MATLAB?

- MATLAB is a numerical computer environment.
- MATLAB allows
  - Matrix manipulation.
  - Plotting of functions and data.
  - Implementation of algorithms.
  - Interfacing with other programs written in other language.
- The name MATLAB is derived from MATrix LABoratory
Matlab interface layout

1. File
2. Edit
3. Current Folder
4. Command Window
5. Start
6. Workspace
7. Command History
Introduction to Matlab windows

<table>
<thead>
<tr>
<th>Window</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command Window</td>
<td>Main window, enters variables, runs programs.</td>
</tr>
<tr>
<td>Figure Window</td>
<td>Contains output from graphic commands.</td>
</tr>
<tr>
<td>Editor Window</td>
<td>Creates and debugs script and function files.</td>
</tr>
<tr>
<td>Help Window</td>
<td>Provides help information.</td>
</tr>
<tr>
<td>Launch Pad Window</td>
<td>Provides access to tools, demos, and documentation.</td>
</tr>
<tr>
<td>Command History Window</td>
<td>Logs commands entered in the Command Window.</td>
</tr>
<tr>
<td>Workspace Window</td>
<td>Provides information about the variables that are used.</td>
</tr>
<tr>
<td>Current Directory Window</td>
<td>Shows the files in the current directory.</td>
</tr>
</tbody>
</table>
Outline

1 Basics
   • Introduction
   • Arithmetics

2 Variables
   • Vectors
   • Matrices

3 Operators and operations
   • Element-wise Operators
   • Automatic initialization
   • Indexing

4 Plotting
   • Plot properties
   • 3D Line Plots
   • surf and contour

5 Statistics

6 Scripting

7 Linear Algebra

8 Debugging
Basics
Arithmetics (Mathematical operations)

- You can enter a simple arithmetics commands by just typing

```plaintext
>> 2 + 3
>> (2 + 3) / 3
```

- Mathematical operators
  +, -, /, *, ^

- Semicolon is used to hide the output display

```plaintext
>> 2 + 3;
```
- Use your up arrow key to bring up the older commands.
- Use down arrow key to bring back the cursor to enter new command.
Basics
Display Formats

>> 101 * (3 + 3^2 + 27^(1/3))/7 % display format = short
ans =
216.4286
>> ans % display format = long
ans =
2.164285714285714e+002
>> ans % display format = short e
ans =
2.1643e+002
>> ans % display format = long e
ans =
2.164285714285714e+002
>> ans % display format = short g
ans =
216.43
### Display Formats

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>format short</td>
<td>Fixed-point with 4 decimal digits for:</td>
<td>&gt;&gt; 290/7</td>
</tr>
<tr>
<td></td>
<td>0.001 ≤ number ≤ 1000 Otherwise display format short e.</td>
<td>ans = 41.4286</td>
</tr>
<tr>
<td>format long</td>
<td>Fixed-point with 14 decimal digits for:</td>
<td>&gt;&gt; 290/7</td>
</tr>
<tr>
<td></td>
<td>0.001 ≤ number ≤ 100 Otherwise display format long e.</td>
<td>ans = 41.42857142857143</td>
</tr>
<tr>
<td>format short e</td>
<td>Scientific notation with 4 decimal digits.</td>
<td>&gt;&gt; 90/7</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ans = 4.1429e+001</td>
</tr>
<tr>
<td>format long e</td>
<td>Scientific notation with 15 decimal digits.</td>
<td>&gt;&gt; 250/7</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ans = 4.142857142857143e+001</td>
</tr>
<tr>
<td>format short g</td>
<td>Best of 5-digit fixed or floating point.</td>
<td>&gt;&gt; 290/7</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ans = 41.429</td>
</tr>
<tr>
<td>format long g</td>
<td>Best of 15-digit fixed or floating point.</td>
<td>&gt;&gt; 290/7</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ans = 41.4285714285714</td>
</tr>
<tr>
<td>format bank</td>
<td>Two decimal digits.</td>
<td>&gt;&gt; 290/7</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ans = 41.40</td>
</tr>
</tbody>
</table>
## Build-in math functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>sqrt(x)</code></td>
<td>Square root.</td>
<td><code>&gt;&gt; sqrt(81)</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>ans = 9</code></td>
</tr>
<tr>
<td><code>exp(x)</code></td>
<td>Exponential ($e^x$).</td>
<td><code>&gt;&gt; exp(5)</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>ans = 148.4132</code></td>
</tr>
<tr>
<td><code>abs(x)</code></td>
<td>Absolute value.</td>
<td><code>&gt;&gt; abs(-24)</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>ans = 24</code></td>
</tr>
<tr>
<td><code>log(x)</code></td>
<td>Natural logarithm. Base e logarithm (ln).</td>
<td><code>&gt;&gt; log(1000)</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>ans = 6.9078</code></td>
</tr>
<tr>
<td><code>log10(x)</code></td>
<td>Base 10 logarithm.</td>
<td><code>&gt;&gt; log10(1000)</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>ans = 3.0000</code></td>
</tr>
</tbody>
</table>
| `factorial(x)` | The factorial function $x!$  
(x must be a positive integer.) | `>> factorial(5)`        |
|              |                                                          | `ans = 120`              |

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>sin(x)</code></td>
<td>Sine of angle $x$ ($x$ in radians).</td>
<td><code>&gt;&gt; sin(pi/6)</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>ans = 0.5000</code></td>
</tr>
<tr>
<td><code>cos(x)</code></td>
<td>Cosine of angle $x$ ($x$ in radians).</td>
<td><code>&gt;&gt; cos(pi/6)</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>ans = 0.8660</code></td>
</tr>
<tr>
<td><code>tan(x)</code></td>
<td>Tangent of angle $x$ ($x$ in radians).</td>
<td><code>&gt;&gt; tan(pi/6)</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>ans = 0.5774</code></td>
</tr>
<tr>
<td><code>cot(x)</code></td>
<td>Cotangent of angle $x$ ($x$ in radians).</td>
<td><code>&gt;&gt; cot(pi/6)</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>ans = 1.7321</code></td>
</tr>
</tbody>
</table>
## Round functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>round(x)</code></td>
<td>Round to the nearest integer.</td>
<td><code>&gt;&gt; round(17/5)</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>ans =</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>3</code></td>
</tr>
<tr>
<td><code>fix(x)</code></td>
<td>Round towards zero.</td>
<td><code>&gt;&gt; fix(13/5)</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>ans =</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>2</code></td>
</tr>
<tr>
<td><code>ceil(x)</code></td>
<td>Round towards infinity.</td>
<td><code>&gt;&gt; ceil(11/5)</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>ans =</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>3</code></td>
</tr>
<tr>
<td><code>floor(x)</code></td>
<td>Round towards minus infinity.</td>
<td><code>&gt;&gt; floor(-9/4)</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>ans =</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>-3</code></td>
</tr>
<tr>
<td><code>rem(x,y)</code></td>
<td>Returns the remainder after x is divided by y.</td>
<td><code>&gt;&gt; rem(13,5)</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>ans =</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>3</code></td>
</tr>
<tr>
<td><code>sign(x)</code></td>
<td>Signum function. Returns 1 if x &gt; 0, -1 if x &lt; 0, and 0 if x = 0.</td>
<td><code>&gt;&gt; sign(5)</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>ans =</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>1</code></td>
</tr>
</tbody>
</table>
Outline

1 Basics
   • Introduction
   • Arithmetics

2 Variables
   • Vectors
   • Matrices

3 Operators and operations
   • Element-wise Operators
   • Automatic initialization
   • Indexing

4 Plotting
   • Plot properties
   • 3D Line Plots
   • surf and contour

5 Statistics

6 Scripting

7 Linear Algebra

8 Debugging

Waqar Qureshi (AIT)
Variables

- Variables is a temporary storage for data. To create a variable and assign a value:

  >>> var = 4.5
  >>> name = 'Ahmed'
  >>> a = 2 * 3

- To display variable value, just type variable name:

  >>> var
  >>> a;

- Some build-in function:
  - `pi` has a value 3.1415926...
  - `ans` stores the last unassigned value (like on calculator)
  - `i` and `j` can be sued to indicate complex numbers
  - `Inf` and `-Inf` are positive and negative infinity
  - `NaN` represents 'Not a Number'
Like other programming languages, array are an important part of MATLAB. Two types of arrays:

- Matrix of numbers
- Cell of array of objects (more advanced data structures)

MATLAB makes vectors easy. That’s its power.
Outline

1 Basics
   - Introduction
   - Arithmetics

2 Variables
   - Vectors
   - Matrices

3 Operators and operations
   - Element-wise Operators
   - Automatic initialization
   - Indexing

4 Plotting
   - Plot properties
   - 3D Line Plots
   - surf and contour

5 Statistics

6 Scripting

7 Linear Algebra

8 Debugging
Variables

Vectors

- Vector is a column or row array structure. Row vector is comma or space separated between brackets

```matlab
>> v = [1 2 5.5 -6.6];
```

- See the stored structure in workspace
Column vectors

- Semicolon separated values between brackets

```matlab
>> c = [4; 2; 7; 4];
```

- Access an element in vector
  ```matlab
  <vector name> (<element index in vector>);
  ```

```matlab
>> c(2);
```

- Function `length()` gives you total elements in a vector
- Function `size()` gives you the dimensions of a vector
Outline

1 Basics
   • Introduction
   • Arithmetics

2 Variables
   • Vectors
   • Matrices

3 Operators and operations
   • Element-wise Operators
   • Automatic initialization
   • Indexing

4 Plotting
   • Plot properties
   • 3D Line Plots
   • surf and contour

5 Statistics

6 Scripting

7 Linear Algebra

8 Debugging
A matrix is a rectangular array of numbers or other mathematical objects. We define matrix as

\[ b = \begin{bmatrix} 3 & 4 & 5 \\ 6 & 7 & 8 \end{bmatrix} \]

\[ c = \begin{bmatrix} 6 & 8 & 8 \\ 7 & 8 & 9 \end{bmatrix} \]
Matrix: Accessing elements

- We need to mention the **index** to access the elements of matrix.

```matlab
>> a = [2 3 4; 5 6 7; 7 8 9]
a =
   2 3 4
   5 6 7
   7 8 9
>> a(2,3)
ans =
   7
>> a(2)
ans =
   5
>> a(:,2)
ans =
   3
   6
```
To mention the range we use (:) symbol.

\[
\begin{align*}
\text{A} & = [2 \ 6 \ 9 \ 3 \ 5 \ 1] \\
\text{B} & = [2 \ 6 \ 9; \ 4 \ 2 \ 8; \ 3 \ 5 \ 1]
\end{align*}
\]
Outline

1 Basics
   • Introduction
   • Arithmetics

2 Variables
   • Vectors
   • Matrices

3 Operators and operations
   • Element-wise Operators
   • Automatic initialization
   • Indexing

4 Plotting
   • Plot properties
   • 3D Line Plots
   • surf and contour

5 Statistics

6 Scripting

7 Linear Algebra

8 Debugging
Operators and operations

Some useful functions

- **length(A)**: Returns the length of an object (vector). For matrix, the length returns no of rows or cols, whichever is greater.

- **size(A, dim)**: Returns no of rows and columns in A. If dim is provided, it will return the output for that specified dimension.

```matlab
B = [2 1 4 9 5];
length(B);
ans = 5

A = [2 1 4; 2 1 3];
size(A);
ans = 2 3
```
Test function `length()` and `size()` on the following arrays.

\[
A = [3 \ 4 \ 2 \ 4] \\
B = [2 \ 4 \ 1; \ 3 \ 4 \ 5] \\
C = [5 \ 2 \ 3 \ 4; \ 3 \ 4 \ 2 \ 1; \ 1 \ 3 \ 4 \ 5]
\]
Operators and operations

Transpose

- Interchange each row and the corresponding column.

```matlab
>> A = [ 1 2 3; 4 2 1]
ans =
    1  2  3
    4  2  1
>> transpose(A);
>> A'
ans =
    1  4
    2  2
    3  1
```
All the functions that work on scalars also work on vectors

```matlab
>> t = [2 4 9];
>> f = sqrt(t);

is same as

```matlab
>> f = [sqrt(2) sqrt(4) sqrt(9)];
```

- Operators ( * / ^)
  - Element-wise
  - Standard
Outline

1 Basics
   - Introduction
   - Arithmetics

2 Variables
   - Vectors
   - Matrices

3 Operators and operations
   - Element-wise Operators
   - Automatic initialization
   - Indexing

4 Plotting
   - Plot properties
   - 3D Line Plots
   - surf and contour

5 Statistics

6 Scripting

7 Linear Algebra

8 Debugging
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
```

```
<table>
<thead>
<tr>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
</tbody>
</table>


3x1 .* 3x1 = 3x1
```

```
[1 1 1] .* [1 2 3] = [1 2 3]
[2 2 2] .* [1 2 3] = [2 4 6]
[3 3 3] .* [1 2 3] = [3 6 9]
3x3 .* 3x3 = 3x3
```

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

Can be any dimension
Operators and operations

Operators: standards

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

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

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

- Multiply the same two vectors element-wise

- Calculate the natural log of each element of the resulting vector
Exercise: Vector Operations

• Find the inner product between $[1 \ 2 \ 3]$ and $[3 \ 5 \ 4]$
  
  » $a = [1 \ 2 \ 3] \ast [3 \ 5 \ 4]'$

• Multiply the same two vectors element-wise
  
  » $b = [1 \ 2 \ 3] \ast [3 \ 5 \ 4]$

• Calculate the natural log of each element of the resulting vector
  
  » $c = \log(b)$
Outline

1 Basics
   • Introduction
   • Arithmetics

2 Variables
   • Vectors
   • Matrices

3 Operators and operations
   • Element-wise Operators
   • Automatic initialization
   • Indexing

4 Plotting
   • Plot properties
   • 3D Line Plots
   • surf and contour

5 Statistics

6 Scripting

7 Linear Algebra

8 Debugging
Initialize a vector of \texttt{ones}, \texttt{zeros}, or \texttt{rand}om numbers

\texttt{>>ones(1, 10)}: row vector with 10 elements, all 1
\texttt{>>zeros(23, 1)}: column vector with 23 elements, all 0
\texttt{>>rand(1, 45)}: row vector with 45 elements (uniform \([0, 1]\))
\texttt{>>nan(1, 69)}: row vector of NaNs (useful for representing uninitialized variables)
Operators and operations

Automatic initialization

- To initialize a linear vector of values use `linspace(S, E, n)`

```plaintext
>> a = linspace(0, 10, 5)
```

start at 0, ends at 10 (inclusive), 5 values

- We can also use operator (:)  

```plaintext
>> b = 0:2:10
```

- Starts at 0, increments by 2, and ends at or before 10
- increment can be decimal or negative

```plaintext
>> c = 1:5
```

if increment isn’t specified, default is 1

- To initialize logarithmically spaced values use `logspace` similar to `linspace`
Outline

1 Basics
   • Introduction
   • Arithmetics

2 Variables
   • Vectors
   • Matrices

3 Operators and operations
   • Element-wise Operators
   • Automatic initialization
   • Indexing

4 Plotting
   • Plot properties
   • 3D Line Plots
   • surf and contour

5 Statistics

6 Scripting

7 Linear Algebra

8 Debugging
**Indexing**

**Vector Indexing**

- `a(n)` return 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);  \rightarrow a=[13 5];
» b=x(1:end-1);  \rightarrow b=[12 13 5];
```

- Keyword **end** mentions index of last element in array.
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 sub-matrices

```
>> A = rand(5)  % shorthand for 5 x 5 matrix
>> A(1:3, 1:2)  % specify contiguous sub-matrix
>> A([1 5 3], [1 4])  % specify rows and columns
```
MATLAB contains functions to help you find desired values within a vector or matrix. To get the minimum value and its index

```matlab
>> vec = [1 5 3 9 7]
>> [minVal, minInd] = min(vec);
```

To get the maximum value and its index

```matlab
>> [maxVal, maxInd] = max(vec);
```
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

```matlab
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)
```
Save, load and clear data

- 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 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`
  ➢ 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;`
Conversing with MATLAB

- **who / whos**: MATLAB replies with the variables in your workspace
- **what**: MATLAB replies with the current directory and MATLAB files in the directory
- **help**: The most important function for learning MATLAB on your own
• To get info on how to use a function:
  » help sin
  ➢ Help contains related functions
• To get a nicer version of help with examples and easy-to-read descriptions:
  » doc sin
Plotting

Simplest and easiest plot

- Plotting is a fun?
- What can be a simplest plot? Plotting values of $y$ corresponding to $x$. I need some values of $x$ and some values of $y$ to plot, in other words.

```matlab
>> x = [0 1 2 3 4];
>> x = 0:4
>> y = x.*2;
```

- Now plot $x$ and $y$

```matlab
>> plot(x,y);
```
We need to generate points to plot them

\[ x = \text{linspace}(0, 4\pi, 10); \]
\[ y = \sin(x); \]

Plot values against their index

\[ \text{plot}(x); \]

Usually we want to plot \( y \) versus \( x \)

\[ \text{plot}(x, y); \]
Plotting

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:](image1.png)

![1000 x values:](image2.png)
Plotting
Plot Options

- Can change the line color, marker style, and line style by adding a string argument
  \[
  \text{plot}(x,y,'k.-');
  \]
  \hspace{0.5cm}
color \quad marker \quad line-style

- Can plot without connecting the dots by omitting line style argument
  \[
  \text{plot}(x,y,'.' )
  \]

- Look at \texttt{help plot} for a full list of colors, markers, and linestyles
Plotting

Other useful plot commands

- Much more on this in Lecture 2, for now some simple commands

- To plot two lines on the same graph
  ```matlab
  hold on;
  ```

- To plot on a new figure
  ```matlab
  figure;
  plot(x,y);
  ```

- Play with the figure GUI to learn more
  - add axis labels
  - add a title
  - add a grid
  - zoom in/zoom out
Plot $f(x) = e^x \cdot \cos(x)$ on the interval $x = [0 \ 10]$. Use a red solid line with a suitable number of points to get a good resolution.
Exercise: Plotting

```matlab
x = 0:.01:10;
plot(x, exp(x).*cos(x), 'r');
```
Plotting
Playing with the plot

to select lines and delete or change properties

to zoom in/out
to slide the plot around

to see all plot tools at once
Plotting
Line and Marker Options

- Everything on a line can be customized
  ```matlab
  » plot(x,y,'--rs','LineWidth',2,...
      'MarkerEdgeColor','k',...
      'MarkerFaceColor','g',...
      'MarkerSize',10)
  ```

- See `doc line` for a full list of properties that can be specified
### Line Style Specifier

<table>
<thead>
<tr>
<th>Specifier</th>
<th>LineStyle</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-</code></td>
<td>Solid line (default)</td>
</tr>
<tr>
<td><code>–</code></td>
<td>Dashed line</td>
</tr>
<tr>
<td><code>:</code></td>
<td>Dotted line</td>
</tr>
<tr>
<td><code>-.</code></td>
<td>Dash-dot line</td>
</tr>
<tr>
<td>Specifier</td>
<td>Marker Type</td>
</tr>
<tr>
<td>-----------------------</td>
<td>------------------------------------</td>
</tr>
<tr>
<td>`'+'</td>
<td>Plus sign</td>
</tr>
<tr>
<td>`'o'</td>
<td>Circle</td>
</tr>
<tr>
<td>`'*'</td>
<td>Asterisk</td>
</tr>
<tr>
<td>`.'</td>
<td>Point</td>
</tr>
<tr>
<td>`'x'</td>
<td>Cross</td>
</tr>
<tr>
<td>`'square' or 's'</td>
<td>Square</td>
</tr>
<tr>
<td>`'diamond' or 'd'</td>
<td>Diamond</td>
</tr>
<tr>
<td>`'^'</td>
<td>Upward-pointing triangle</td>
</tr>
<tr>
<td>`'v'</td>
<td>Downward-pointing triangle</td>
</tr>
<tr>
<td>`'&gt;$'</td>
<td>Right-pointing triangle</td>
</tr>
<tr>
<td>`'&lt;''</td>
<td>Left-pointing triangle</td>
</tr>
<tr>
<td>`'pentagram' or 'p'</td>
<td>Five-pointed star (pentagram)</td>
</tr>
<tr>
<td>`'hexagram' or 'h'''</td>
<td>Six-pointed star (hexagram)</td>
</tr>
<tr>
<td>Specifier</td>
<td>Color</td>
</tr>
<tr>
<td>-----------</td>
<td>-----------</td>
</tr>
<tr>
<td>r</td>
<td>Red</td>
</tr>
<tr>
<td>g</td>
<td>Green</td>
</tr>
<tr>
<td>b</td>
<td>Blue</td>
</tr>
<tr>
<td>c</td>
<td>Cyan</td>
</tr>
<tr>
<td>m</td>
<td>Magenta</td>
</tr>
<tr>
<td>y</td>
<td>Yellow</td>
</tr>
<tr>
<td>k</td>
<td>Black</td>
</tr>
<tr>
<td>w</td>
<td>White</td>
</tr>
</tbody>
</table>
Plot the sine function over three different ranges using different line styles, colors, and markers.

Figure: Reprinted from: http://www.mathworks.com/help/matlab/ref/linespec.html
figure

```matlab
  t = 0:pi/20:2*pi;
  plot(t,sin(t),'-.r*')
  hold on
  plot(t,sin(t-pi/2),'-mo')
  plot(t,sin(t-pi),':bs')
  hold off
```
Plotting

Labels

- Last time we saw how to add titles and labels using the GUI. Can also do it command-line:
  ```
  » title('Stress-Strain');
  » xlabel('Force (N)');
  ```

- For multiple lines, add a legend entry for each line
  ```
  » legend('Steel','Aluminum','Tungsten');
  ```

- Can specify font and size for the text
  ```
  » ylabel('Distance (m)','FontSize',14,...
     'FontName','Helvetica');
  ```
  ▶ use ... to break long commands across multiple lines

- To put parameter values into labels, need to use `num2str` and concatenate:
  ```
  » str = ['Strength of ' num2str(d) 'cm diameter rod'];
  » title(str)
  ```
• A grid makes it easier to read values
  » grid on

• xlim sets only the x axis limits
  » xlim([-pi pi]);
• ylim sets only the y axis limits
  » ylim([-1 1]);

• To specify both at once, use \texttt{axis}:
  » axis([-pi pi -1 1]);
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hspace{1cm}
  \hs space{1cm}
  \hs space{1cm}
  \hs space{1cm}
  \hs space{1cm}
  \hs space{1cm}
  \hs space{1cm}
  \hs space{1cm}
  \hs space{1cm}
  \hs space{1cm}
  \hs space{1cm}
  \hs space{1cm}
  \hs space{1cm}
  \hs space{1cm}
  \hs space{1cm}
  \hs space{1cm}
  \hs space{1cm}
  \hs space{1cm}
  \hs space{1cm}
  \hs space{1cm}
  \hs space{1cm}
- Built-in axis modes
  - \texttt{axis square}
    - makes the current axis look like a box
  - \texttt{axis tight}
    - fits axes to data
  - \texttt{axis equal}
    - makes $x$ and $y$ scales the same
  - \texttt{axis xy}
    - puts the origin in the bottom left corner (default)
  - \texttt{axis ij}
    - puts the origin in the top left corner (for viewing matrices)
Plotting

Multiple Plots

- Use the figure command to open a new figure
  » `figure`

- or activate an open figure
  » `figure(1)`

- To have multiple axes in one figure
  » `subplot(2,3,1)` or `subplot(231)`
    ➢ makes a figure with 2 rows and three columns of axes, and activates the first axis for plotting
    ➢ each axis can have labels, a legend, and a title
  » `subplot(2,3,4:6)`
    ➢ activating a range of axes fuses them into one

- To close existing figures
  » `close([1 3])`
    ➢ closes figures 1 and 3
  » `close all`
    ➢ closes all figures (useful in scripts/functions)
Figures can be pasted into other apps (word, ppt, etc)
- Edit \rightarrow copy options \rightarrow figure copy template
  - Change font sizes, line properties; presets for word and ppt
- Edit \rightarrow copy figure to copy figure
- Paste into document of interest
Figures can be saved in many formats. The common ones are:

- `.fig` preserves all information
- `.bmp` uncompressed image
- `.eps` high-quality scaleable format
- `.pdf` compressed image

Courtesy of The MathWorks, Inc. Used with permission.
Open a figure and plot a sine wave over two periods with data points between 0 and 4\pi. Use black squares as markers and a dashed red line of thickness 2 at the line.
```matlab
» figure
» plot(0:pi/4:4*pi,sin(0:pi/4:4*pi), 'rs--', ...
   'LineWidth', 2, 'MarkerFaceColor', 'k');
```

- Save the figure as a pdf
- View with pdf viewer.
Outline

1 Basics
   • Introduction
   • Arithmetics

2 Variables
   • Vectors
   • Matrices

3 Operators and operations
   • Element-wise Operators
   • Automatic initialization
   • Indexing

4 Plotting
   • Plot properties
   • 3D Line Plots
   • surf and contour

5 Statistics

6 Scripting

7 Linear Algebra

8 Debugging

Waqar Qureshi (AIT)
Modeling for WREE
February 11, 2014
We can plot in 3 dimensions just as easily as in 2

```matlab
>> time=0:0.001:4*pi;
>> x=sin(time);
>> y=cos(time);
>> z=time;
>> plot3(x,y,z,'k','LineWidth',2);
>> zlabel('Time');
```

- Use tools on figure to rotate it
- Can set limits on all 3 axes
  ```matlab
  >> xlim, ylim, zlim
  ```
Plotting
Surface Plots

- It is more common to visualize surfaces in 3D

- Example:
  \[ f(x, y) = \sin(x)\cos(y) \]
  \[ x \in [-\pi, \pi]; y \in [-\pi, \pi] \]

- **surf** puts vertices at specified points in space \( x, y, z \), and connects all the vertices to make a surface

- The vertices can be denoted by matrices \( X, Y, Z \)

- How can we make these matrices
  - loop (DUMB)
  - built-in function: **meshgrid**
Outline

1 Basics
   • Introduction
   • Arithmetics

2 Variables
   • Vectors
   • Matrices

3 Operators and operations
   • Element-wise Operators
   • Automatic initialization
   • Indexing

4 Plotting
   • Plot properties
   • 3D Line Plots
   • surf and contour

5 Statistics

6 Scripting

7 Linear Algebra

8 Debugging

Waqar Qureshi (AIT)
Plotting

surf

- Make the x and y vectors
  » \( x=-\pi:0.1:\pi; \)
  » \( y=-\pi:0.1:\pi; \)

- Use meshgrid to make matrices (this is the same as loop)
  » \([X,Y]=\text{meshgrid}(x,y);\)

- To get function values, evaluate the matrices
  » \( Z=\sin(X) .* \cos(Y); \)

- Plot the surface
  » \( \text{surf}(X,Y,Z) \)
  » \( \text{surf}(x,y,Z); \)
- See `help surf` for more options
- There are three types of surface shading
  - shading faceted
  - shading flat
  - shading interp
- You can change colormaps
  - `colormap(gray)`
You can make surfaces two-dimensional by using `contour`

```matlab
contour(X,Y,Z,'LineWidth',2)
```
- Takes same arguments as `surf`
- Color indicates height
- Can modify linestyle properties
- Can set colormap

```matlab
hold on
```

```matlab
mesh(X,Y,Z)
```
Plotting

Exercise: 3D plots

- Plot $\exp(-.1(x^2+y^2))\sin(xy)$ for $x,y$ in $[-2\pi,2\pi]$ with interpolated shading and a hot colormap:
x = -2*pi:0.1:2*pi;
y = -2*pi:0.1:2*pi;
[X,Y] = meshgrid(x,y);
Z = exp(-.1*(X.^2+Y.^2)).*sin(X.*Y);
surf(X,Y,Z);
shading interp
colormap hot
Specialized Plotting Functions

- MATLAB has a lot of specialized plotting functions
  - **polar** - to make polar plots
    ```matlab
    polar(0:0.01:2*pi,cos((0:0.01:2*pi)*2))
    ```
  - **bar** - to make bar graphs
    ```matlab
    bar(1:10,rand(1,10));
    ```
  - **quiver** - to add velocity vectors to a plot
    ```matlab
    [X,Y]=meshgrid(1:10,1:10);
    quiver(X,Y=rand(10),rand(10));
    ```
  - **stairs** - plot piecewise constant functions
    ```matlab
    stairs(1:10,rand(1,10));
    ```
  - **fill** - draws and fills a polygon with specified vertices
    ```matlab
    fill([0 1 0.5],[0 0 1],'r');
    ```
  - see help on these functions for syntax
  - **doc specgraph** – for a complete list
End
Outline

1 Basics
   • Introduction
   • Arithmetics

2 Variables
   • Vectors
   • Matrices

3 Operators and operations
   • Element-wise Operators
   • Automatic initialization
   • Indexing

4 Plotting
   • Plot properties
   • 3D Line Plots
   • surf and contour

5 Statistics

6 Scripting

7 Linear Algebra

8 Debugging
Statistics

- Whenever analyzing data, you have to compute statistics
  
  » scores = 100*rand(1,100);

- Built-in functions
  
  ➢ mean, median, mode

- To group data into a histogram
  
  » hist(scores,5:10:95);
  
  ➢ makes a histogram with bins centered at 5, 15, 25...95
  
  » N=histc(scores,0:10:100);
  
  ➢ returns the number of occurrences between the specified bin edges 0 to <10, 10 to <20...90 to <100.
Many probabilistic processes rely on random numbers.

MATLAB contains the common distributions built in:

- rand
  - draws from the uniform distribution from 0 to 1
- randn
  - draws from the standard normal distribution (Gaussian)
- random
  - can give random numbers from many more distributions
  - see doc random for help
  - the docs also list other specific functions

You can also seed the random number generators:

- rand('state',0)
We can alter the given distributions

```matlab
y = rand(1, 100) * 10 + 5;
% gives 100 uniformly distributed numbers between 5 and 15
y = floor(rand(1, 100) * 10 + 6);
% gives 100 uniformly distributed integers between 10 and 15. floor or ceil is better to use here than round
```

```matlab
y = randn(1, 1000)
```

```matlab
y2 = y * 5 + 8
% increases std to 5 and makes the mean 8
```
Understanding of mean, median and mode

Median
(middle or average of middle values)
\[
\frac{4.8 + 5.6}{2}
\]

Mode
(most common value)
9.6

Mean
(Average Value)
\[
\frac{3.4 + 4.8 + 8.4 + 9.6 + 2.3 + 9.6 + 5.6 + 9.6 + 4.8 + 2.2}{10}
\]
Following command is used to calculate the mean of a matrix

\[ X = \begin{bmatrix} 2, & 3, & 4; & 4 & 9 & 1 \end{bmatrix} \]
\[ M = \text{mean}(X, \text{dim}) \]

dim = 1 returns the mean of each column and dim = 2 returns the mean of each row of the matrix. In case of using a vector instead of a matrix, the command is used as

\[ X = [2, 7, 3, 9/2, 11, 12] \]
\[ M = \text{mean}(X) \]
Command below is used for calculating median.

\[ M = \text{median}(X, \text{dim}) \]

and the command for mode

\[ M = \text{mode}(X, \text{dims}) \]
Variance and Standard deviation

- Command below is used for calculating median.

$$V = \text{var}(X, w, \text{dim})$$
$$S = \text{std}(X, w, \text{dim})$$

$w = 0$ uses $n - 1$, and $w = 1$ uses $n$ for calculation. More by typing help std/var.
Range

- Range is a static that represents the difference between maximum and minimum value.

\[ R = \text{range}(X, \text{dim}) \]

- Apply mean, median, mode and range on data.txt
Skewness and Kurtosis are calculated in MATLAB using command

\[
\begin{align*}
Sk &= \text{Skewness}(x, \text{flag}, \text{dim}) \\
K &= \text{Kurtosis}(x, \text{flag}, \text{dim})
\end{align*}
\]

flag = 0 is used to correct the calculation for a sample from a population. Otherwise flag = 1 is used to calculate the above parameters for a population.
Matlab provides a command `polyfit` to fit a line to your data. Load two data files `Housesx.dat` and `pricesy.dat` and fit line.

```matlab
>> x = load('housesx.dat');
>> y = load('pricesy.dat');
```

To fit a line we have command.

```matlab
>> p = polyfit(x, y, 1);
```

that return two coefficient fitting line

```matlab
r = a_1 * x + a_2
```
compute a new vector that has matching data points in $x$

$$r = p(1) \times x + p(2);$$

Now plot both the points in $y$ and the curve fit in $r$

```matlab
plot(x, y, 'x', 'MarkerSize', 10);
hold on;
plot(x, r, '-');
hold off;
```
Outline

1 Basics
   - Introduction
   - Arithmetics

2 Variables
   - Vectors
   - Matrices

3 Operators and operations
   - Element-wise Operators
   - Automatic initialization
   - Indexing

4 Plotting
   - Plot properties
   - 3D Line Plots
   - Surf and contour

5 Statistics

6 Scripting

7 Linear Algebra

8 Debugging
Copy the data files to your working folder from the following link.
https://sites.google.com/site/abdulbasitkhan/

You should have the following files
- Temp.dat
- housesx.dat
- pricesy.dat
- statistic.dat
Save, load and clear data

- **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 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`
  > - 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;`
Scripts: Overview

- Scripts are
  - written in the MATLAB editor
  - saved as MATLAB files (.m extension)
  - evaluated line by line

- To create an MATLAB file from command-line
  » edit myScript.m

- or click
Scripts: The Editor

* Means that it's not saved

Line numbers

MATLAB file path

Debugging tools

Help file

Comments

```matlab
% coinToss.m
% a script that flips a fair coin and displays the output

if rand < 0.5 % if random number is less than 0.5 say heads
disp('HEADS');
else % if greater than 0.5, say tails
disp('TAILS');
end
```
• Take advantage of "smart indent" option

• Keep code clean
  ➢ Use built-in functions
  ➢ Vectorize, vectorize, vectorize
  ➢ When making large matrices, allocate space first
    ➢ Use nan or zeros to make a matrix of the desired size

• Keep constants at the top of the MATLAB file

• 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
A student has taken three exams. The performance on the exams is random (uniform between 0 and 100)

- The first exam is worth 20%, the second is worth 30%, and the final is worth 50% of the grade
- Calculate the student's overall score
- Save script as practiceScript.m and run a few times
A student has taken three exams. The performance on the exams is random (uniform between 0 and 100).
The first exam is worth 20%, the second is worth 30%, and the final is worth 50% of the grade.
Calculate the student's overall score.
Save script as practiceScript.m and run a few times.

```matlab
scores=rand(1,3)*100;
weights=[0.2 0.3 0.5];
overall=scores*weights'
```
User-defined Functions

- Functions look exactly like scripts, but for **ONE** difference
  - Functions must have a function declaration

```
% stats: computes the average, standard deviation, and range
% of a given vector of data
%
% [avg, sd, range]=stats(x)
% avg - the average (arithmetic mean) of x
% sd - the standard deviation of x
% range - a 2x1 vector containing the min and max values in x
% x - a vector of values

function [avg, sd, range]=stats(x)
  avg=mean(x);
  sd=std(x);
  range=[min(x); max(x)];
```
User-defined Functions - Cond.

- Some comments about the function declaration

  \[
  \text{function } [x, y, z] = \text{funName}\text{(in1, in2)}
  \]

  Inputs must be specified

  Must have the reserved word: function

  If more than one output, must be in brackets

- No need for return: MATLAB returns the variables whose names match those in the function declaration

- Variable scope: Any variables created within the function but not returned disappear after the function stops running

- Can have variable input arguments (see help varargin)
Functions: Exercise

- Take the script we wrote to calculate the student's overall score and make it into a function.

- The inputs should be:
  - the scores row vector
  - the weight row vector, with the same length as scores

- The output should be:
  - A scalar: the overall score

- Assume the user knows the input constraints (no need to check if the inputs are in the correct format/size)

- Name the function `overallScore.m`
We're familiar with
  » zeros
  » size
  » length
  » sum

Look at the help file for size by typing
  » help size

The help file describes several ways to invoke the function
  ➢ D = SIZE(X)
  ➢ [M,N] = SIZE(X)
  ➢ [M1,M2,M3,...,MN] = SIZE(X)
  ➢ M = SIZE(X,DIM)
Functions

- MATLAB uses mostly standard relational operators
  - equal
    - `==`
  - not equal
    - `~=`
  - greater than
    - `>`
  - less than
    - `<`
  - greater or equal
    - `>=`
  - less or equal
    - `<=`

- Logical operators
  - And
    - `normal &`
    - `bitwise &&`
  - Or
    - `|`
    - `||`
  - Not
    - `~`
    - `xor`
    - `all`
  - Xor
  - All true
  - Any true
  - Any

- Boolean values: zero is false, nonzero is true
- See help . for a detailed list of operators
Functions

- Basic flow-control, common to all languages
- MATLAB syntax is somewhat unique

**IF**
```plaintext
if cond
  commands
end
```

**ELSE**
```plaintext
if cond
  commands1
else
  commands2
end
```

**ELSEIF**
```plaintext
if cond1
  commands1
elseif cond2
  commands2
else
  commands3
end
```

- No need for parentheses: command blocks are between reserved words
- **for loops**: use for a definite number of iterations
- **MATLAB syntax**:

```
for n=1:100
commands
end
```

- **The loop variable**
  - Is defined as a vector
  - Is a scalar within the command block
  - Does not have to have consecutive values

- **The command block**
  - Anything between the `for` line and the `end`
• The while is like a more general for loop:
  ➢ Don't need to know number of iterations

```
WHILE
  while cond
  commands
  end
```

• The command block will execute while the conditional expression is true
• Beware of infinite loops!
Exercise: Control Flow

- Write a function to calculate the factorial of an integer $N$ using a loop (you can use a for or while loop). If the input is less than 0, return NaN. Test it using some values.

```matlab
function a = factorial(N)
    if N<0,
        a=nan,
    else
        a = 1;
        for k=1:N
            a = a*k;
        end
    end
```

- But note that factorial() is already implemented! You should see if there are built-in functions before implementing something yourself.
  
```matlab
  which factorial
  ```
Vectorization

- Avoid loops whenever possible
  - This is referred to as vectorization
- Vectorized code is more efficient for MATLAB
- Use indexing and matrix operations to avoid loops
- For example:
  ```matlab
  a = rand(1,100);
b = zeros(1,100);
for n = 1:100
  if n == 1
    b(n) = a(n);
  else
    b(n) = a(n-1) + a(n);
  end
end
  ```
  - Efficient and clean
- Slow and complicated
Vectorization: Exercise

- Alter your factorial program to work WITHOUT a loop. Use `prod`

  ```
  » function a=factorial(N)
  » a=prod(1:N);
  ```

- You can tic/toc to see how much faster this is than the loop!

- **BUT**...Don’t ALWAYS avoid loops
  - Over-vectorizing code can obfuscate it, i.e. you won’t be able to understand or debug it later
  - Sometime a loop is the right thing to do, it is clearer and simple
Outline

1 Basics
   • Introduction
   • Arithmetics

2 Variables
   • Vectors
   • Matrices

3 Operators and operations
   • Element-wise Operators
   • Automatic initialization
   • Indexing

4 Plotting
   • Plot properties
   • 3D Line Plots
   • surf and contour

5 Statistics

6 Scripting

7 Linear Algebra

8 Debugging
Systems of Linear Equations

- Given a system of linear equations
  - $x + 2y - 3z = 5$
  - $-3x - y + z = -8$
  - $x - y + z = 0$
- Construct matrices so the system is described by $Ax = b$
  - $A = [1 \ 2 \ -3; \ -3 \ -1 \ 1; \ 1 \ -1 \ 1];$
  - $b = [5; -8; 0];$
- And solve with a single line of code!
  - $x = A \backslash b;$
  - $x$ is a $3 \times 1$ vector containing the values of $x, y,$ and $z$
- The \ will work with square or rectangular systems.
- Gives least squares solution for rectangular systems. Solution depends on whether the system is over or underdetermined.
More Linear Algebra

- Given a matrix
  ```matlab
  mat=[1 2 -3;-3 -1 1;1 -1 1];
  ```

- Calculate the rank of a matrix
  ```matlab
  r=rank(mat);
  ```
  ➢ the number of linearly independent rows or columns

- Calculate the determinant
  ```matlab
  d=det(mat);
  ```
  ➢ mat must be square
  ➢ if determinant is nonzero, matrix is invertible

- Get the matrix inverse
  ```matlab
  E=inv(mat);
  ```
  ➢ if an equation is of the form $A\times x=b$ with $A$ a square matrix,
  $x=A\backslash b$ is the same as $x=inv(A)\times b$
Matrix Decompositions

- MATLAB has built-in matrix decomposition methods

- The most common ones are
  - \([V,D] = \text{eig}(X)\)
    - Eigenvalue decomposition
  - \([U,S,V] = \text{svd}(X)\)
    - Singular value decomposition
  - \([Q,R] = \text{qr}(X)\)
    - QR decomposition
• Solve the following systems of equations:

- System 1:
  - $x + 4y = 34$
  - $-3x + y = 2$

  ```matlab
  A = [1 4; -3 1];
  b = [34; 2];
  rank(A)
  x = inv(A) * b;
  ```

- System 2:
  - $2x - 2y = 4$
  - $-x + y = 3$
  - $3x + 4y = 2$

  ```matlab
  A = [2 -2; -1 1; 3 4];
  b = [4; 3; 2];
  rank(A)
  x1 = A \ b;
  
  ➤ rectangular matrix
  
  ➤ gives least squares solution
  
  A * x1
  ```
Debugging

- When debugging functions, use `disp` to print messages
  
  ```
  » disp('starting loop')
  » disp('loop is over')
  
  » disp prints the given string to the command window
  ```

- It's also helpful to show variable values
  
  ```
  » disp(strcat(['loop iteration ',num2str(n)]));
  
  » strcat concatenates the given strings
  
  » Sometimes it's easier to just remove some semicolons
Debugging

- To use the debugger, set breakpoints
  - Click on – next to line numbers in MATLAB files
  - Each red dot that appears is a breakpoint
  - Run the program
  - The program pauses when it reaches a breakpoint
  - Use the command window to probe variables
  - Use the debugging buttons to control debugger

```matlab
% coinToss.m
% a script that flips a fair coin and displays the output

if rand < 0.5
    disp('HEADS');    % Two breakpoints
else
    disp('TAILS');    % Where the program is now
end
```
Exercise: Debugging

```matlab
% buggyCode
x = 1:10;
mat = rand(3,10);
vec = rand(3,1);
an1 = x.*(mat.^2)';
an2 = an1 + (vec./rand(1,3))';
an3 = an2.*(mat.*x);
```
What are Toolboxes?

• Toolboxes contain functions specific to a particular field
  ➢ for example: signal processing, statistics, optimization

• It's generally more efficient to use MATLAB's toolboxes rather than redefining the functions yourself
  ➢ saves coding/debugging time
  ➢ some functions are compiled, so they run faster
  ➢ HOWEVER there may be mistakes in MATLAB’s functions and there may also be surprises
End