Matlab tutorial

Nathanael Fillmore
September 2009

Outline

1. Matlab as a glorified calculator.

Outline

1. Matlab as a glorified calculator.
2. Matlab as a glorified graphing calculator.

Outline

1. Matlab as a glorified calculator.
2. Matlab as a glorified graphing calculator.
3. Matlab as a programming language.

Outline

1. Matlab as a glorified calculator.
2. Matlab as a glorified graphing calculator.
3. Matlab as a programming language.
4. Demo.

Outline

1. Matlab as a glorified calculator.
2. Matlab as a glorified graphing calculator.
3. Matlab as a programming language.
4. Demo.

Note: 1 “$\subset$” 2 “$\subset$” 3.

Matlab as a glorified calculator: scalars

Whatever you think should work will probably work:

- Arithmetic:

```
>> (2+3*4)/6
ans =
    2.3333
```

Matlab as a glorified calculator: scalars

Whatever you think should work will probably work:

- Arithmetic:

```
>> (2+3*4)/6
ans =
    2.3333
```

- Trigonometric:

```
>> sin(pi/2)
ans =
     1
```

Matlab as a glorified calculator: scalars

Whatever you think should work will probably work:

- Arithmetic:

```
>> (2+3*4)/6
ans =
    2.3333
```

- Trigonometric:

```
>> sin(pi/2)
ans =
     1
```

- Exponential:

```
>> log(exp(1))
ans =
     1
```

Matlab as a glorified calculator: vectors

You can also define vectors:

- Row vectors:

```
>> x=[2 4 8]
x =
     2     4     8
```

Matlab as a glorified calculator: vectors

You can also define vectors:

- Row vectors:

```
>> x=[2 4 8]
x =
     2     4     8
```

- Column vectors:

```
>> x=[2;4;8]
x =
     2
     4
     8
```

Matlab as a glorified calculator: vectors

You can also define vectors:

- Row vectors:

```
>> x=[2 4 8]
x =
     2     4     8
```

- Column vectors:

```
>> x=[2;4;8]
x =
     2
     4
     8
```

OR

```
>> x=[2 4 8]'
x =
     2
     4
     8
```

Matlab as a glorified calculator: vectors

You can also define vectors:

- Row vectors:

```
>> x=[2 4 8]
x =
     2     4     8
```

- Column vectors:

```
>> x=[2;4;8]
x =
     2
     4
     8
```

OR

```
>> x=[2 4 8]'
x =
     2
     4
     8
```

And access their elements:

```
>> x(1)
ans =
     2
```

```
>> x(2)
ans =
     4
```

```
>> x(3)
ans =
     8
```

Matlab as a glorified calculator: vectors

The following syntax is often useful:

```
>> 1:5
ans =
     1     2     3     4     5
>> 1:2:10
ans =
     1     3     5     7     9
```

Matlab as a glorified calculator: matrices

Matrices are defined just like vectors:

```
>> [1 2 3; 4 5 6; 7 8 9]
ans =
     1     2     3
     4     5     6
     7     8     9
```

Matlab as a glorified calculator: matrices

Matrices are defined just like vectors:

```
>> [1 2 3; 4 5 6; 7 8 9]
ans =
     1     2     3
     4     5     6
     7     8     9

>> [1 2 3; 4 5 6; 7 8 9]'
ans =
     1     4     7
     2     5     8
     3     6     9
```

Matlab as a glorified calculator: matrices

Matrices are defined just like vectors:

```
>> [1 2 3; 4 5 6; 7 8 9]
ans =
     1     2     3
     4     5     6
     7     8     9

>> [1 2 3; 4 5 6; 7 8 9]'
ans =
     1     4     7
     2     5     8
     3     6     9

>> [1:3; 4:6; 7:9]
ans =
     1     2     3
     4     5     6
     7     8     9
```

Matlab as a glorified calculator: matrices

Elements of matrices are accessed just like elements of vectors:

```
>> A=[1 2 3; 4 5 6; 7 8 9]
ans =
     1     2     3
     4     5     6
     7     8     9
>> A(1,1)
ans =
     1
>> A(1,2)
ans =
     2
>> A(2,1)
ans =
     4
>> A(2,3)
ans =
     6
```

Matlab as a glorified calculator: matrices

Elements of matrices are accessed just like elements of vectors:

```
>> A=[1 2 3; 4 5 6; 7 8 9]
ans =
     1     2     3
     4     5     6
     7     8     9
>> A(1,1)
ans =
     1
>> A(1,2)
ans =
     2
>> A(2,1)
ans =
     4
>> A(2,3)
ans =
     6
```

You can also select submatrices:

```
>> A(1,:)
ans =
     1     2     3

>> A(:,1)
ans =
     1
     4
     7

 >> A(1:2,2:3)
 ans =
      2     3
      5     6
```

Matlab as a glorified calculator: matrices

Addition, subtraction, and scalar multiplication work as expected:

```
>> A=[1 2; 3 4];
>> B=[2 4; 6 8];
```

```
>> A+B
ans =
     3     6
     9    12
```

```
>> 2*A-B
ans =
     0     0
     0     0
```

Matlab as a glorified calculator: matrices

Addition, subtraction, and scalar multiplication work as expected:

```
>> A=[1 2; 3 4];
>> B=[2 4; 6 8];
```

```
>> A+B
ans =
     3     6
     9    12
```

```
>> 2*A-B
ans =
     0     0
     0     0
```

Multiplication is matrix multiplication:

```
>> A*B
ans =
    14    20
    30    44
```

Matlab as a glorified calculator: matrices

Addition, subtraction, and scalar multiplication work as expected:

```
>> A=[1 2; 3 4];
>> B=[2 4; 6 8];
```

```
>> A+B
ans =
     3     6
     9    12
```

```
>> 2*A-B
ans =
     0     0
     0     0
```

Multiplication is matrix multiplication:

```
>> A*B
ans =
    14    20
    30    44
```

Elementwise multiplication is also available:

```
>> A.*B
ans =
     2     8
    18    32
```

Matlab as a glorified calculator: matrices

Many functions exist to create special matrices:

```
>> eye(3)
ans =
     1     0     0
     0     1     0
     0     0     1
```

```
>> ones(2,3)
ans =
     1     1     1
     1     1     1
```

```
>> zeros(3,2)
ans =
     0     0
     0     0
     0     0
```

Matlab as a glorified calculator: matrices

Many functions exist to create special matrices:

```
>> eye(3)
ans =
     1     0     0
     0     1     0
     0     0     1
```

```
>> ones(2,3)
ans =
     1     1     1
     1     1     1
```

```
>> zeros(3,2)
ans =
     0     0
     0     0
     0     0
```

```
>> rand(2,4)
ans =
    0.9649    0.9706    0.4854    0.1419
    0.1576    0.9572    0.8003    0.4218
```

Matlab as a glorified calculator: matrices

Many functions exist to create special matrices:

```
>> eye(3)
ans =
     1     0     0
     0     1     0
     0     0     1
```

```
>> ones(2,3)
ans =
     1     1     1
     1     1     1
```

```
>> zeros(3,2)
ans =
     0     0
     0     0
     0     0
```

```
>> rand(2,4)
ans =
    0.9649    0.9706    0.4854    0.1419
    0.1576    0.9572    0.8003    0.4218
```

Many functions also exist to manipulate matrices:

```
A =
     1     2
     3     4
```

```
>> det(A)
ans =
    -2
```

```
>> inv(A)
ans =
   -2.0000    1.0000
    1.5000   -0.5000
```

Matlab as a glorified calculator: matrices

Many functions exist to create special matrices:

```
>> eye(3)
ans =
     1     0     0
     0     1     0
     0     0     1
```

```
>> ones(2,3)
ans =
     1     1     1
     1     1     1
```

```
>> zeros(3,2)
ans =
     0     0
     0     0
     0     0
```

```
>> rand(2,4)
ans =
    0.9649    0.9706    0.4854    0.1419
    0.1576    0.9572    0.8003    0.4218
```

Many functions also exist to manipulate matrices:

```
A =
     1     2
     3     4
```

```
>> det(A)
ans =
    -2
```

```
>> inv(A)
ans =
   -2.0000    1.0000
    1.5000   -0.5000
```

```
>> mean(A)
ans =
     2     3
```

```
>> mean(A,2)
ans =
    1.5000
    3.5000
```

Matlab as a glorified graphing calculator

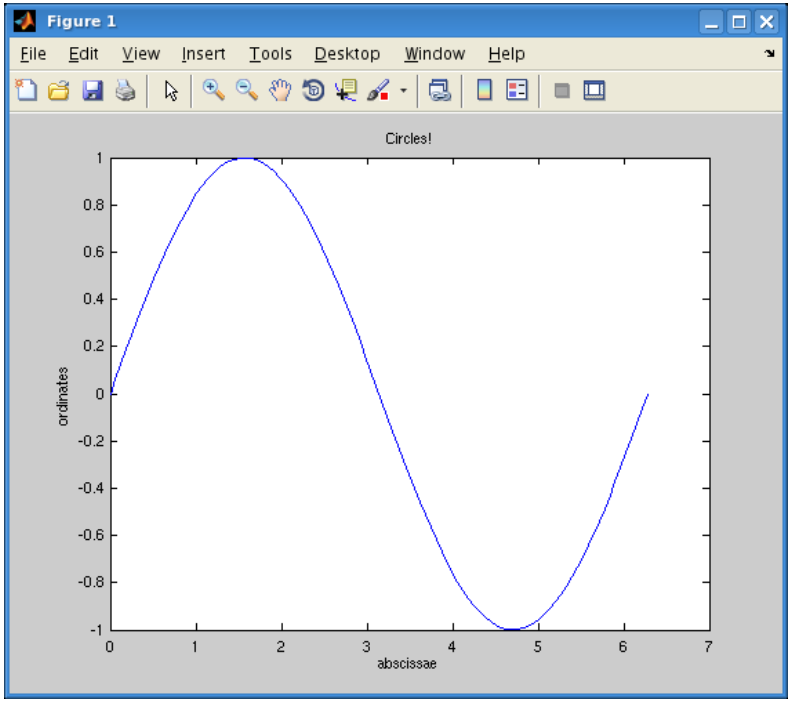
Plotting is easy:

```
>> x=linspace(0,2*pi);
>> y=sin(x);
>> plot(x,y)
>> xlabel('abscissae')
>> ylabel('ordinates')
>> title('Circles!')
```

Matlab as a glorified graphing calculator

Plotting is easy:

```
>> x=linspace(0,2*pi);
>> y=sin(x);
>> plot(x,y)
>> xlabel('abscissae')
>> ylabel('ordinates')
>> title('Circles!')
```



Matlab as a glorified graphing calculator

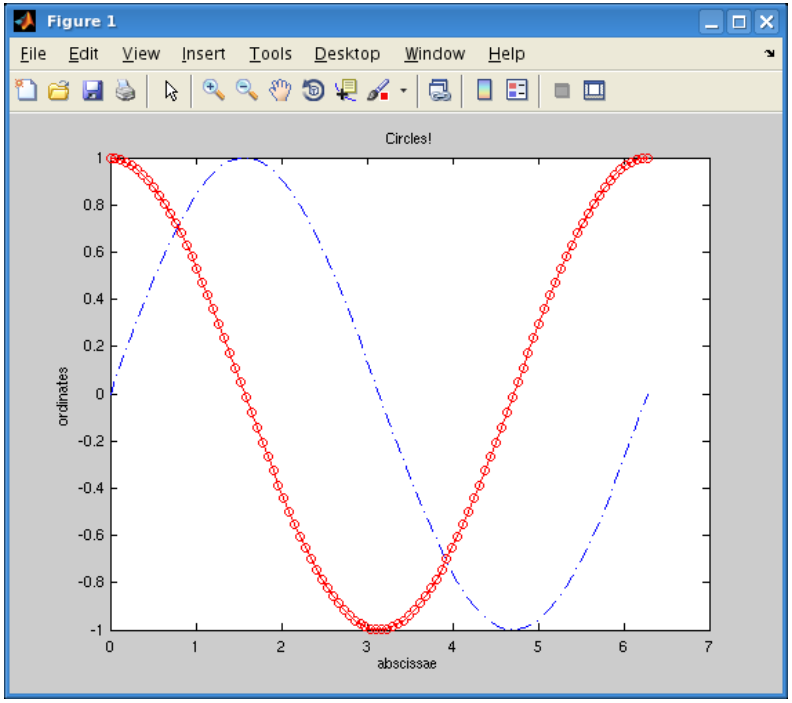
Multiple functions can be shown
on the same plot:

```
>> x=linspace(0,2*pi);
>> y=sin(x);
>> z=cos(x);
>> plot(x,y,'b-.',x,z,'r-o')
>> xlabel('abscissae')
>> ylabel('ordinates')
>> title('Circles!')
```

Matlab as a glorified graphing calculator

Multiple functions can be shown on the same plot:

```
>> x=linspace(0,2*pi);
>> y=sin(x);
>> z=cos(x);
>> plot(x,y,'b-.',x,z,'r-o')
>> xlabel('abscissae')
>> ylabel('ordinates')
>> title('Circles!')
```



Matlab as a programming language: control flow

For loops:

```
>> x=1;
>> for i=1:4
     x=x+2*i;
   end
>> x
x =
    21
```

Matlab as a programming language: control flow

For loops:

```
>> x=1;
>> for i=1:4
     x=x+2*i;
   end
>> x
x =
    21
```

While loops:

```
>> x=10;
>> while x>=0
     x=x-1;
   end
>> x
x =
    -1
```

Matlab as a programming language: control flow

Conditionals:

```
>> x=4;
>> if x<5
     disp('less');
   elseif x>5
     disp('greater');
   else
     disp('equal');
   end
less
```

```
>> x=6;
>> if x<5
     disp('less');
   elseif x>5
     disp('greater');
   else
     disp('equal');
   end
greater
```

```
>> x=5;
>> if x<5
     disp('less');
   elseif x>5
     disp('greater'
   else
     disp('equal');
   end
equal
```

Matlab as a programming language: control flow

Conditionals:

```
>> x=4;
>> if x<5
     disp('less');
   elseif x>5
     disp('greater');
   else
     disp('equal');
   end
less
```

```
>> x=6;
>> if x<5
     disp('less');
   elseif x>5
     disp('greater');
   else
     disp('equal');
   end
greater
```

```
>> x=5;
>> if x<5
     disp('less');
   elseif x>5
     disp('greater'
   else
     disp('equal');
   end
equal
```

Matlab's logical operators include:
==, ~= ("does not equal"), <, >, <=, >=, &&, ||, ~ ("not").

Matlab as a programming language: functions

Each function is placed in a separate file.
The name of the file is the name of the function.

Matlab as a programming language: functions

Each function is placed in a separate file.
The name of the file is the name of the function.

Example: Place the following in modadd.m:

```
function z=modadd(x,y,n)
z=mod(x+y,n);
```

Matlab as a programming language: functions

Each function is placed in a separate file.
The name of the file is the name of the function.

Example: Place the following in modadd.m:

```
function z=modadd(x,y,n)
z=mod(x+y,n);
```

Then, at the console:

```
>> modadd(1,6,7)
ans =
     0
```

Matlab as a programming language: functions

Each function is placed in a separate file.
The name of the file is the name of the function.

Example: Place the following in modadd.m:

```
function z=modadd(x,y,n)
z=mod(x+y,n);                % <-- semicolon suppresses output
```

Then, at the console:

```
>> modadd(1,6,7)
ans =
     0
```

Matlab as a programming language: functions

Each function is placed in a separate file.
The name of the file is the name of the function.

Example: Place the following in modadd.m:

```
function z=modadd(x,y,n)
z=mod(x+y,n);                % <-- semicolon suppresses output
```

Then, at the console:

```
>> modadd(1,6,7)
ans =
     0
```

Note 1: Functions must be located in the current working directory (`pwd`) or in a directory listed in `path`.

Matlab as a programming language: functions

Each function is placed in a separate file.
The name of the file is the name of the function.

Example: Place the following in modadd.m:

```
function z=modadd(x,y,n)
z=mod(x+y,n);                % <-- semicolon suppresses output
```

Then, at the console:

```
>> modadd(1,6,7)
ans =
     0
```

Note 1: Functions must be located in the current working directory (`pwd`) or in a directory listed in `path`.

Note 2: Matlab includes a text editor; type e.g. `edit modadd.m` to open it.

Matlab as a programming language: scripts

Each script is placed in a separate file.
The name of the file is the name of the script.

Matlab as a programming language: scripts

Each script is placed in a separate file.
The name of the file is the name of the script.

Example: Place the following in modtable.m:

```
x=(0:3)'*ones(1,4);
z=modadd(x,x',4)
```

Matlab as a programming language: scripts

Each script is placed in a separate file.
The name of the file is the name of the script.

Example: Place the following in modtable.m:

```
x=(0:3)'*ones(1,4);
z=modadd(x,x',4)
```

Then, at the console:

```
>> modtable
z =
     0     1     2     3
     1     2     3     0
     2     3     0     1
     3     0     1     2
```

Matlab as a programming language: scripts

Each script is placed in a separate file.
The name of the file is the name of the script.

Example: Place the following in modtable.m:

```
x=(0:3)'*ones(1,4);
z=modadd(x,x',4)
```

Then, at the console:

```
>> modtable
z =
     0     1     2     3
     1     2     3     0
     2     3     0     1
     3     0     1     2
```

Variables defined in a script are available outside the script:

```
>> x
x =
     0     0     0     0
     1     1     1     1
     2     2     2     2
     3     3     3     3
```

Demo