

# Introduction to Matlab

SAMSI Undergraduate Workshop

May 15, 2006

## The things you need to know about Matlab

- How arrays and matrices are stored
- How to use operators on arrays and matrices
- How to write a .m file
- How to display your results
- Some programming syntax
- The Statistical Toolbox
- ODE Solvers
- Optimization routines
- Uploading data

How arrays and matrices are stored

Row Vector

```
>>r = [1,2,3,4];
```

or

```
>>r = [1 2 3 4];
```

Column Vector

```
>>c = [1;2;3;4];
```

Array and Matrices con't

Matrices are Row Vectors Separated by a ;

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

or

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

## Miscellaneous Built-in Matrices that come in handy

- `>>zeros(m,n)`
- `>>ones(m,n)`
- `>>eye(n)`

**Note:** To see description and documentation for a Matlab built-in function type *help functionname*, e.g. `help ones`

## Operators on Matrices

- $+$ ,  $-$ ,  $*$ ,  $/$ ,  $^$  perform exactly how you would expect
- However “ $.*$ ”, “ $./$ ”, “ $.^$ ” act element-by-element for each matrix.

Example

$$A = B = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

$$A * B = \begin{bmatrix} 7 & 10 \\ 15 & 22 \end{bmatrix}$$

$$A.^ * B = \begin{bmatrix} 1 & 4 \\ 9 & 16 \end{bmatrix}$$

## Using the Colon Operator on Matrices in Matlab

If A is a matrix, then

```
>>A(:,j)
```

returns the  $j^{th}$  column of A

```
>>A(i,:)
```

returns the  $i^{th}$  row of A

Let

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

Then

$$A(:,1) = \begin{bmatrix} 1 \\ 3 \end{bmatrix}$$

And,

$$A(2, :) = \begin{bmatrix} 3 & 4 \end{bmatrix}$$

Using the Colon Operator to create vectors

>> [j:k]

is the same as  $[j, j + 1, j + 2, \dots, k]$

>> [j:i:k]

is the same as  $[j, j + i, j + 2i, \dots, k]$

Let

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

$$B = \begin{bmatrix} 2 & 4 \\ 9 & 16 \end{bmatrix}$$

Compute the following

- $A'$
- $B./A$
- $A./B$
- $\exp(A)$
- Create  $[.2, .35, .50, .65, .80, \dots, 1.55]$  using the colon operator

## Writing .m files

**What are they?** Function files saved as *functionname.m* that contain a sequence of Matlab lines.

### Structure

```
function out_variables = functionname(in_variables)
%comment lines
```

Body of Matlab code to execute

We will write two .m files as examples

1. fexp.m will evaluate  $y = \frac{1}{2}e^{\frac{1}{2}x}$  where  $x$  is an input  
function `y=fexp(x)`
2. quadratic.m will evaluate  $y = ax^2 + bx + c$  where  $a, b, c$   
and  $x$  are inputs  
function `y=quadratic(x, constants)`  
where  $constants = [a, b, c]$

Now let's use these functions

1. Let  $constants = [a, b, c] = [2, -3, 4]$  and let  $x = [-4 : .01 : 4]$ . Compute  $y1 = quadratic(x, constants)$
2. Let  $x \in [-3, 7]$  with a step size of .01. Compute  $y2 = fexp(x)$

## Displaying your results

- `plot(X,Y)` plots the *vector* Y versus the *vector* X  
example: `plot(t,sin(t))` where t is a vector
- Plotting multiple graphs  
syntax: `plot(t, sin(t), t, cos(t))`
- To create labels  
syntax: `xlabel('label'), ylabel('label')`
- `title` works the same as `xlabel`
- `legend(string1, string2, ...)` creates a legend
- type `help plot`

## Some Programming Syntax

- The “if” loop

```
if x == 3
```

```
    y = 2;
```

```
elseif x~1
```

```
    y = 3;
```

```
else
```

```
    y = 0;
```

```
end
```

Other operators include  $>$ ,  $>=$ ,  $<$ ,  $<=$

- The “for” loop

```
x=zeros(100,100)
```

```
for i=1:100
```

```
    for j=1:100
```

```
        x(i,j) = cos(i)*sin(j);
```

```
    end
```

```
end
```

- The “while” loop

```
count=0;
tol=10;

while (tol>1e-10) && (count <= 500)
    fox = 2*xold*sin(xold); %evaluate the function y=2x*sin(x)
    fprimeofx = 2*xold*cos(xold)+2*sin(xold);

    xnew = xold-fox/fprimeofx; %Newton Step
    tol = abs(xold-xnew); %How close are the two values
    count = count + 1; %increment count
    xold = xnew; %redeclare

end
```

## The Statistical Toolbox

- `mean(x)`
- `var(x)`
- `std(x)`

Note: If you think Matlab should have a certain function, they probably do.

## ODE Solvers

### 1. Main Syntax:

$[t, y] = \text{ode45}(@\text{fun}, [t_0, t_f], y_0, \text{options}, a_1, \dots, a_N);$

- **Inputs**

- @fun = Name of the function computing the ode
- $[t_0, t_f]$  = timespan
- $y_0$  = initial conditions
- options = ODE solver options (type **help ode45**)
- $a_1, \dots, a_N$  = parameter to pass to @fun

- **Outputs**

- $t$  = the variable timesteps
- $y$  = the function values at those timesteps

### 2. Requirements for the function computing the ode

- **Syntax:**

function  $dy = \text{fun}(t, y, \text{flag}, a_1, \dots, a_N)$

- $dy =$  (place the evaluation of the ODE here)
- Rewrite your ODE as  $y' = f(t, y(t))$
  - *flag* is a [placeholder](#) for *options* in the solver command
  - $a_1, \dots, a_N$  are other parameters that the ODE needs

## A simple example

Let  $y = f(t) = e^{\frac{1}{2}t}$ . Then

$$y' = f'(t) = \frac{1}{2}e^{\frac{1}{2}t} = \frac{1}{2}y$$
$$y(0) = 1$$

Hence, we write the following function to evaluate the ODE above

```
function dy = myode(t, y)
```

```
dy = .5*y;
```

And we call ODE45 as follows

```
>> [t, y] = ode45(@fun, [t0, tf], y0, options, a1, ..., aN);
```

```
>>[t y] = ode45(@myode, [0, 20], 1);
```

```
>>plot(t,y)
```

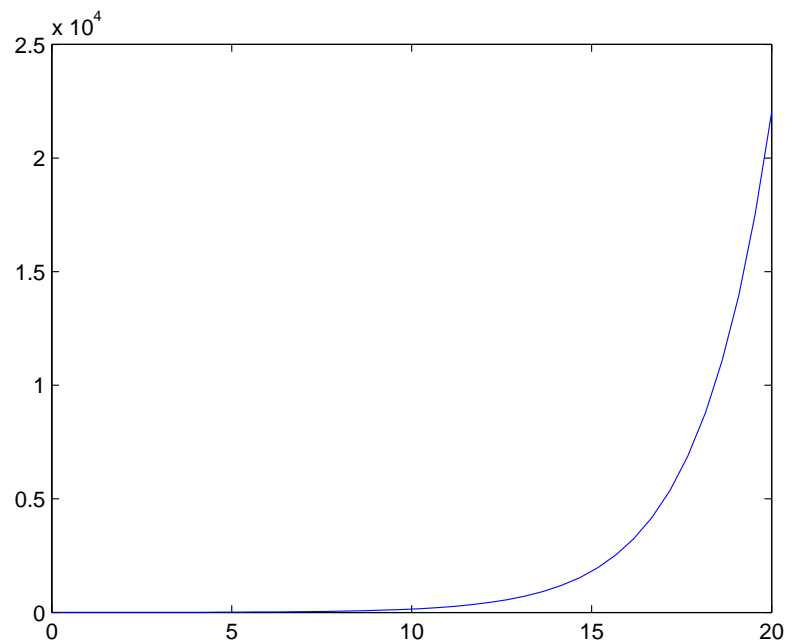


Figure 1:

ODE of  $2^{nd}$  order or higher

$$my''(t) + cy'(t) + ky(t) = \cos(t)$$

where  $y(0) = 0, y'(0) = 0$ .

Then, let

$$y_1 = y$$

$$y_2 = \dot{y}_1$$

$$\Rightarrow \dot{y}_2 = \ddot{y}_1 = \ddot{y}$$

which means we can rewrite our original ODE as

$$\begin{bmatrix} \dot{y}_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -\frac{k}{m} & -\frac{c}{m} \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{1}{m} \end{bmatrix} \cos(t).$$

## Other ODE Solvers

- ODE23
- ODE113
- ODE15S
- ODE23S
- ODE23T
- ODE23TB

## Optimization Routines(*fminsearch*)

- Syntax `x = fminsearch(FUN,X0)`

The Details: *fminsearch* starts at X0 and attempts to find a local minimizer X of the function FUN. FUN accepts input X and returns a scalar function value F evaluated at X. X0 can be a scalar, vector or matrix.

### Example

Compute the minimum of

$$f(x) = 3x^2 - 4x + 2$$

Then, *constants* = [3, -4, 2]

and

```
>>x = fminsearch(@(x)quadratic(x,constants), 3);
```

```
x = 0.6667
```

## Uploading Data

Here is the syntax:

```
>>load('filename')
```

where 'filename' is the name of the file that contains the data.

type **help load**