

JHU Department of Civil Engineering

CE 560.445: Advanced Structural Analysis

Spring 2002

MATLAB tutorial

1 Basics

1.1 Navigation

- `cd` – change directory. `cd c:\temp` moves you to the `temp` directory on the `c:` drive. `cd ..` moves up one level.
- `mkdir` – make new directory. `mkdir temp` makes a directory of name `temp`.
- `ls` – list contents of current directory. `ls *.m` lists all files ending in `.m`.
- `pwd` – prints current working directory.

1.2 Syntax and commands

- `whos` – lists currently active variables and their size and type. `whos x*` lists all current variables which start with `x`.
- `save` – save all data in the current workspace. `save mydata` saves all current variables in file `mydata.mat`. `save mydata x* -ascii` save all current variables starting with `x` in an ascii file called `mydata.mat`.
- `load` – loads data from a `.mat` file. `load mydata` loads variables from file `mydata.mat`.
- the semicolon. A semicolon at the end of a line suppresses the output of the line.
- `help` command lists the help information for command.
- `lookfor` keyword lists all commands the help files of which contain keyword.
- `edit filename` opens `filename.m` in the MATLAB editor, or creates the file if it does not exist.

2 Vectors and matrices

- A vector is simply a special case of a matrix.
- `m = [1 2; 3 2]`; creates the matrix

$$\begin{bmatrix} 1 & 2 \\ 3 & 2 \end{bmatrix}$$

- The series of commands `m(1,1) = 1; m(1,2) = 2; m(2,1) = 2; m(2,2) = 2;` gives the same matrix.
- `m(i,j)` is m_{ij} .
- `m(i:j,k:l)` is a submatrix of m which includes rows i, \dots, j and columns k, \dots, l .
- `m'` gives the transpose of m .
- `det(m)` gives the determinant of m .
- `inv(m)` gives the inverse of m .
- `y=a\b` solves the system of equations $b = ay$. This is equivalent to $y=inv(a)*b$ but uses a more efficient solution algorithm.
- `zeros(n,m)` – makes a matrix of dimension $n \times m$ in which every entry is a zero. `ones(n,m)` has the same syntax.
- `x = xmin:dx:xmax` makes a vector which spans from `xmin` to `xmax` with an increment dx .

Example 1: Solve the systems of equations

$$\begin{aligned} 1 &= x_1 - 2x_2 + x_3 \\ 2 &= 4x_1 + 3x_2 - 3x_3 \\ 3 &= x_1 - 2x_2 + 2x_3 \end{aligned}$$

and

$$\begin{aligned} 1 &= x_1 - 2x_2 \\ 2 &= 4x_1 + 3x_2 \end{aligned}$$

1. `>>y = [1 2 3]'`;
2. `>>m = [1 -2 1; 4 3 -3; 1 -2 2]`;
3. `>>xa = m\y`;
4. `>>xa` (to see the result to the first system)
5. `>>xb = m(1:2,1:2)\y(1:2)`;
6. `>>xb` (to see the result to the second system)

3 Graphics

- The most basic graphics operation is to plot two vectors, one against the other. `plot(x,y,'pattern')` plots y against x with a line of characteristics `pattern`.
- `figure` – creates a new window in which to display graphics.
- `xlabel('xname')`, `ylabel('yname')` labels the x and y axes of the plot.
- `title('titlename')` adds a title to the plot.
- `axes([x1 x2 y1 y2])` sets the upper and lower limits of the plot axes.
- `line([x1 x2],[y1 y2])` draws a line from point (x_1, y_1) to (x_2, y_2) .

Example 2: Plot $y = \sin(t)$ over the interval $t \in (5, 10)$ and add a line showing $y = 0$.

```
1. >>t = 5:.1:10;
2. >>y = sin(t);
3. >>figure;
4. >>plot(t,y,'r<--');
5. >>xlabel('time');ylabel('sin(t)');
6. >>axes([5 10 -1.2 1.2]);
7. >>line([5 10],[0 0]);
```

4 Control structures

- `for i = imin:di:imax; commands; end` executes a for loop, incrementing the counter i by the amount di after each loop. `commands` are carried out until i reaches $imax$.
- `if condition1; commands1; elseif condition2; commands2; else; commands3` executes `commands1` if `condition1` is met, and so on. The `elseif` and `else` parts are optional. The conditions are given in the form $x==10, x>=10, x<10, x\sim=10$, etc.

Example 3: Generate a plot of the function $y = \text{sign}(\sin(t)) \min\{|\sin(t)|, 1\}$ on the interval $t \in (0, 12\pi)$

```

t = 0:01:12*pi;
for i = 1:length(t)
    temp = abs(2*sin(t(i)));
    sgn = sign(sin(t(i)));
    if temp > 1
        y(i) = 1*sgn;
    else
        y(i) = sgn*temp;
    end
end
plot(t,y);

```

5 Functions

- User defined function live in files which are named `funcname.m` and which have as their first line `function [out_1,...,out_n] = funcname(in_1,...,in_n)` where `out_1,...,out_n` are the output variables, and `in_1,...,in_n` are the arguments, or input variables.
- More than one function can exist in a single file, but only the first can be called from the MATLAB command line. Others can be called only from within functions which come prior to them in the file.
- You can insert comments into the file by using the `%` character.

Example 4: Write a function which takes the length, moment of inertia, and elastic modulus of a fixed-fixed beam and the magnitude of a point load applied at its midpoint and returns the maximum positive and negative bending moments and the maximum displacement of the beam.

1. `edit beam` to create and open a file called `beam.m`
2. Enter the following code

```

function [moments,disp] = beam(L,I,E,P)

mmax = P*L/8;
mmin = -P*L/8;
moments(1) = mmax;
moments(2) = mmin;

disp = P*L^3/(192*E*I);

```

3. Save the file `beam.m`

4. To execute the function type, for example `[m,d] = beam(60,500,30000,330);` and `m,d` to see the results.

Exercise 1: Consider again the fixed-fixed beam with a point load applied at the midpoint. Assume that the elastic modulus and moment of inertia of the beam are known exactly, but that the length and applied load are free parameters which take values over some intervals $(L_1, L_2), (P_1, P_2)$. The minimum midpoint displacement will occur when $P = P_1$ and $L = L_1$ and the maximum midpoint displacement when $P = P_2$ and $L = L_2$. Write a matlab function or functions to which returns pairs (L, P) which result in a deflection greater than a specified value Δ_{max} .

The input for your function should be the fixed values of E, I , the maximum and minimum values of P, L and the increments dP, dL . Also, the allowable deflection Δ_{max} . You can use the code from above to do this task.