THE UNIVERSITY OF RHODE ISLAND

EGR 106
Foundations of Engineering II

Lecture 8 – Part B
Loops and User Defined Functions

THINK BIG WE DO℠

This Week's Topics

Programming in MATLAB (cont.)

- "for-end" loops (cont.)
- "while-end" loops
- "break" and "continue" commands
- Nested loops
- User defined functions

This Week's Examples – Lecture_8.m

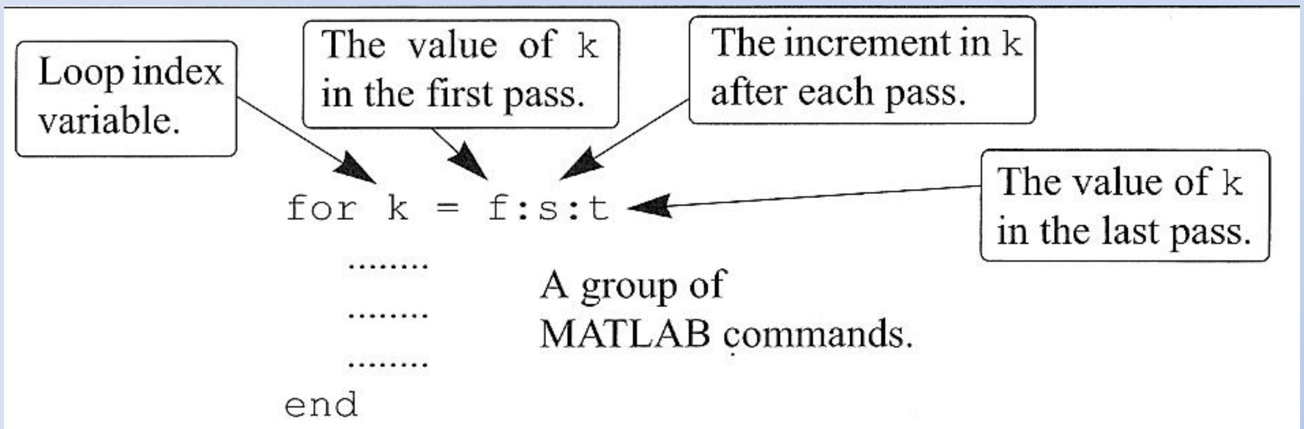
1. Compound interest using 'for' loop
2. Compound interest using 'while' loop
3. Compound interest using break
4. Constant acceleration problem
5. Taylor/Maclaurin series summation
6. Computing Fourier series with nested loops
7. User function to convert degrees to radians
8. User function to estimate sine

This Week’s Examples (cont.)

Lecture_8.m

```
Enter 1 for compound interest problem using for loop
      2 for compound interest problem using while loop
      3 for compound interest problem using 'break'
      4 for constant acceleration problem
      5 for Taylor / Maclaurin series
      6 for Fourier series wiith nested loops
      7 for user function: deg2rad
      8 for user function: Taylor_sin
   =>
```

The “for-end” Loop



Example 1 – Compound Interest

Calculating 5% interest compounded annually for 10 years:

```
year(1)=0;
value(1)=1000;
rate=.05;
for i=1:10
    year(i+1)=i;
    value(i+1)=value(i)*(1+rate);
    disp(['$ ' num2str(value(i+1)) ' after ' num2str(i) ' years.'])
end
```

```
$ 1050 after 1 years.
$ 1103 after 2 years.
$ 1158 after 3 years.
$ 1216 after 4 years.
$ 1276 after 5 years.
$ 1340 after 6 years.
$ 1407 after 7 years.
$ 1477 after 8 years.
$ 1551 after 9 years.
$ 1629 after 10 years.
```

while – end Command

```
while conditional expression
    . . .
    . . .
    . . .
end
```

Group of MATLAB commands

Loop will continue until as long as conditional expression is true

Be careful to avoid infinite loop

Example 2 – Compound Interest with while loop

Interest (5%) compounded until investment doubles:

```
year=0;
value=1000;
rate=.05;
while value<2000
    year=year+1;
    value=value*(1+rate);
    disp(['$ ' num2str(value) ' after ' num2str(i) ' years.'])
end
```

```
$ 1050 after 1 years.
$ 1103 after 2 years.
$ 1158 after 3 years.
$ 1216 after 4 years.
$ 1276 after 5 years.
$ 1340 after 6 years.
$ 1407 after 7 years.
$ 1477 after 8 years.
$ 1551 after 9 years.
$ 1629 after 10 years.
$ 1710 after 11 years.
$ 1796 after 12 years.
$ 1886 after 13 years.
$ 1980 after 14 years.
$ 2079 after 15 years.
```

Loop Controls

Loops contain sets of commands that you want to do repeatedly.

But you might want to:

- Skip commands in the current iteration
- Stop the loop itself

Why continue once you've found what you're looking for !!

Skipping Ahead: Continue

Continue – jumps to next loop iteration:

```
x=-3:6;
for k = 1:10
    disp(['Iteration ', num2str(k)])
    if  x(k) > 0
        continue
    end
    disp(['x(k) = ',num2str(x(k))])
end
```

skip to end & continue loop

```
Iteration 1
x(k) = -3
Iteration 2
x(k) = -2
Iteration 3
x(k) = -1
Iteration 4
x(k) = 0
Iteration 5
Iteration 6
Iteration 7
Iteration 8
Iteration 9
Iteration 10
```

THE UNIVERSITY OF RHODE ISLAND

Early Termination: Break Command

Break ends the loop:

```
x=-3:6;
for k = 1:10
    disp(['Iteration ', num2str(k)])
    if  x(k) > 0
        break
    end
    disp(['x(k) = ',num2str(x(k))])
end
```

go to commands beyond end

```
Iteration 1
x(k) = -3
Iteration 2
x(k) = -2
Iteration 3
x(k) = -1
Iteration 4
x(k) = 0
Iteration 5
```

Example 3 – using the Break command

Calculate interest until the amount doubles:

will calculate up to 1000 years, if necessary

```
value=1000;
for year=1:1000
    value=value*1.05;
    disp(['$ ',num2str(round(value)),' after ',...
        num2str(year),' years'])
    if value>=2000
        break
    end
end
```

if condition decides when to terminate loop

```
$ 1050 after 1 years
$ 1103 after 2 years
$ 1158 after 3 years
$ 1216 after 4 years
$ 1276 after 5 years
$ 1340 after 6 years
$ 1407 after 7 years
$ 1477 after 8 years
$ 1551 after 9 years
$ 1629 after 10 years
$ 1710 after 11 years
$ 1796 after 12 years
$ 1886 after 13 years
$ 1980 after 14 years
$ 2079 after 15 years
```

Example 4 – Constant Acceleration

Consider the deceleration a plane after landing:

Landing speed: 70 m/s

Acceleration: $a=-1.5\ m/s^2$

Exact solution after 40 seconds:

$$v(40)=v_0+a\,t = 70\ m/s - 1.5\ m/s^2\ (40\ s) = 10\ m/s$$

$$s(40)=s_0+v_0\,t + \tfrac{1}{2}\,a\,t^2 = 0\ m + 70\ m/s\ (40\ s) + \tfrac{1}{2}\,(-1.5\ m/s^2)(40\ s)^2$$

$$= 1600\ m$$

Example 4 – Constant Acceleration (cont.)

Numerical (approximate) solution:

Given $v = \frac{ds}{dt} = v_0 + a\,t$

where $\frac{ds}{dt} = \frac{s(t+dt) - s(t)}{dt}$

solving for s(t+dt) gives $s(t+dt) = s(t) + (v_0 + a\,t)\,dt$

with $s(0\ sec) = 0, v_0 = 70\,\frac{m}{s}, a = \ -1.5\ \frac{m}{s^2}$

determine $s(40\ sec) = ?$

Example 4 – Constant Acceleration (cont.)

Matlab solution:

$$s(0\,sec) = 0$$

$$v_0 = 70\frac{m}{s}$$

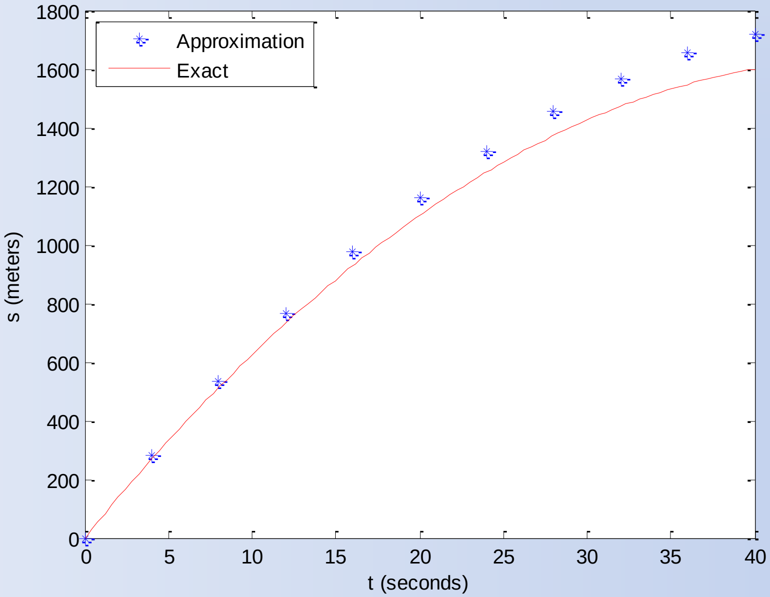
$$a = -1.5\,\frac{m}{s^2}$$

$$s(t + dt) = s(t) + (v_0 + a\,t)\,dt$$

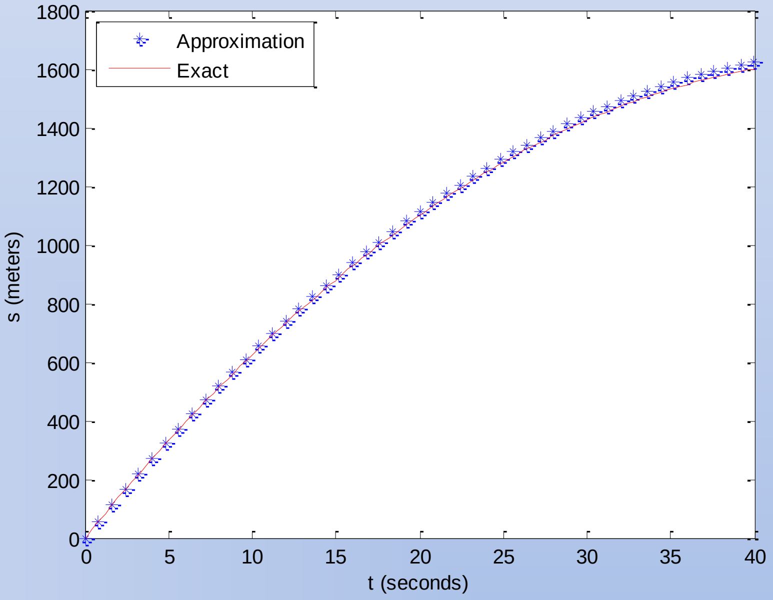
```
close all; clear; clc;
%
t(1)=0;
s(1)=0;
v0=70;
a=-1.5;
N=input('Enter number of time steps: ');
dt=40/N;
%
for i=1:N
    t(i+1)=t(i)+dt;
    s(i+1)=s(i)+(v0+a*t(i))*dt;
end
t_exact=linspace(0,40,100);
s_exact=v0*t_exact+.5*a*t_exact.^2;
plot(t,s,'b*',t_exact,s_exact,'r')
xlabel('t (seconds)')
ylabel('s (meters)')
legend('Approximation','Exact','Location','NorthWest')
```

Example 4 – Constant Acceleration (cont.)

N = 10 steps



N = 50 steps



THE UNIVERSITY OF RHODE ISLAND

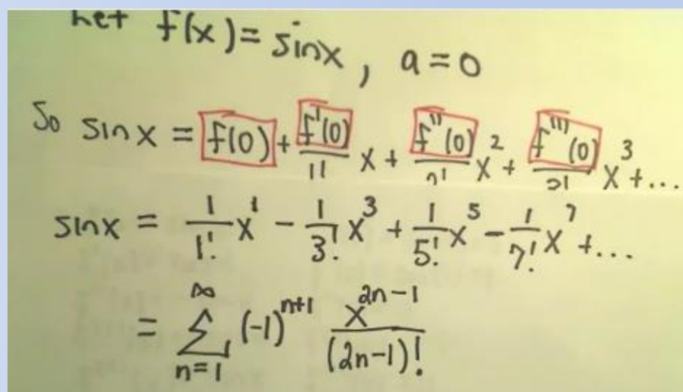
Example 5 – Taylor /Maclaurin Series for sin(x)

Express sin(x) in a Taylor/Maclaurin series expansion

$$\sin x = \frac{x}{1!} - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots = \sum_{n=1}^{\infty} (-1)^{n+1} \frac{x^{2n-1}}{(2n-1)!}$$

Aside – video on how to find Taylor/Maclaurin series expansion:

http://www.youtube.com/watch?v=dp2ovDuWhro



Example 5 – Taylor /Maclaurin Series for sin(x) (cont.)

Sum the series at x=pi/4:

Important – must first initialize sum

```
x=pi/4;
sum=0;
for n=1:5
    sum=sum+(-1)^(n+1)*x^(2*n-1)/factorial(2*n-1);
    error=abs(sum-sqrt(2)/2);
    disp([num2str(n),' terms, sum = ', num2str(sum,14),...
        ', error = ', num2str(error,3)])
end
```

$$\sum_{n=1}^{\infty}(-1)^{n+1}\frac{x^{2n-1}}{(2n-1)!}$$

```
1 terms, sum = 0.78539816339745, error = 0.0783
2 terms, sum = 0.70465265120917, error = 0.00245
3 terms, sum = 0.70714304577936, error = 3.63e-005
4 terms, sum = 0.70710646957518, error = 3.12e-007
5 terms, sum = 0.70710678293687, error = 1.75e-009
```

Note: $\sin(\pi/4)=\frac{\sqrt{2}}{2}=0.707106781186547$

Nested Loops – loops within loops

```
for  index1 = array1
    {outer loop commands}
    for index2 = array2
        {inner loop commands}
    end
    {more outer loop commands}
end
```

Note each “for” must have its own “end”

Can be more than 2 levels deep

Nested Loops - Example

```
for index1=1:4
    for index2=[3 6 9]
        [index1 index2]
    end
end
```

index1	index2
1	3
1	6
1	9
2	3
2	6
2	9
3	3
3	6
3	9
4	3
4	6
4	9

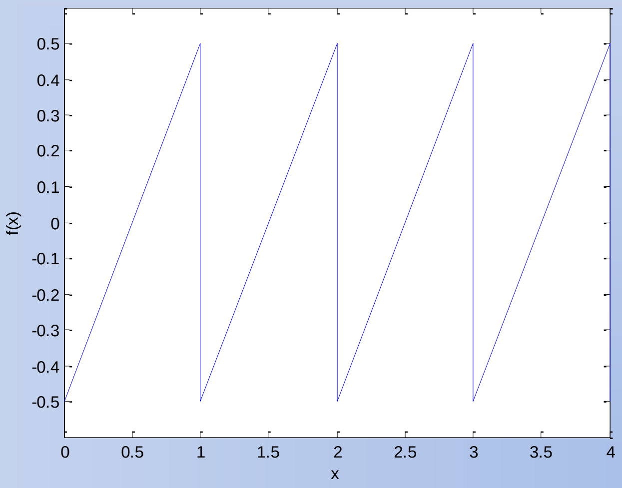
Example 6 - Computing Fourier Series with Nested Loops

Fourier Series:

any periodic function can be expressed as an infinite series of sines and cosines.

Example:

Sawtooth Wave

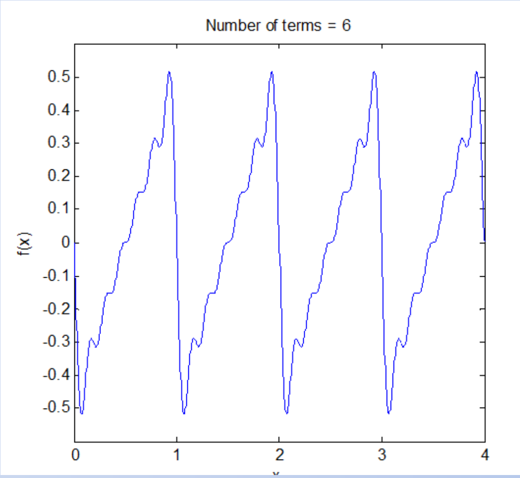


Example 6 - Fourier Series – Sawtooth Function

Fourier Series expansion

$$f(x) = \sum_{n=1}^{\infty} -\left(\frac{\sin(2n\pi x)}{\pi} \right)$$

Sum of first six terms:



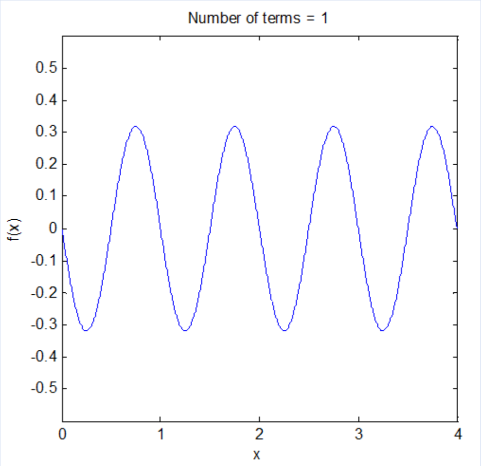
THE UNIVERSITY OF RHODE ISLAND

Example 6 - Fourier Series Example (cont)

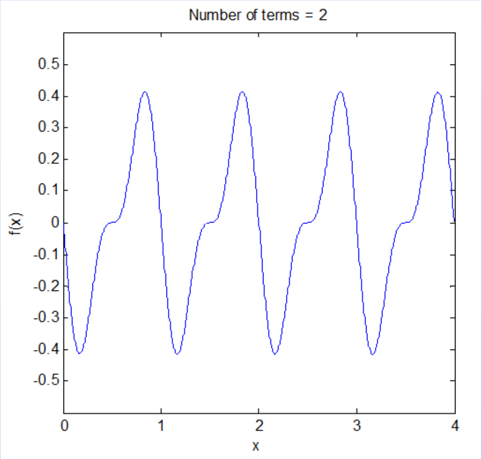
```
close all; clear; clc;
% define x and initialize f(x)=0
x=linspace(0,4,1000);
f=zeros(1,1000);
% for each value of n, compute and plot both new term added to series
%   and the summation of terms 1 to n:
figure('Position',[100, 100, 1000, 400])
for n=1:25
    for xi=1:1000
        fnew(xi)=-(1/pi)*(1/n)*sin(2*n*pi*x(xi));
        f(xi)=f(xi)+fnew(xi);
    end
    subplot(1,2,1)
    plot(x,fnew)
    title('New Term');
    xlabel('x');ylabel('f_n_e_w(x)');axis([0 4 -.6 .6]);
    subplot(1,2,2)
    plot(x,f)
    title(['Number of terms = ',num2str(n)]);
    xlabel('x');ylabel('f(x)');axis([0 4 -.6 .6]);
    pause
end
```

Fourier Series Example (cont)

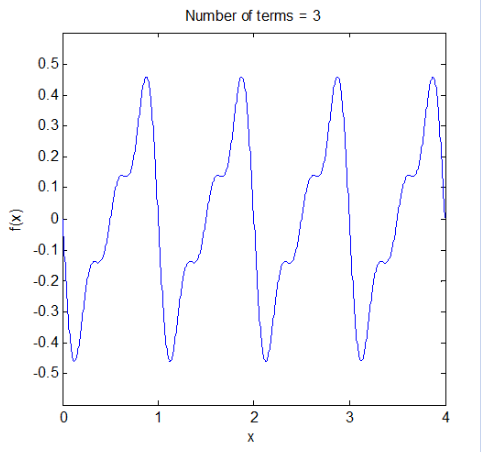
One term:



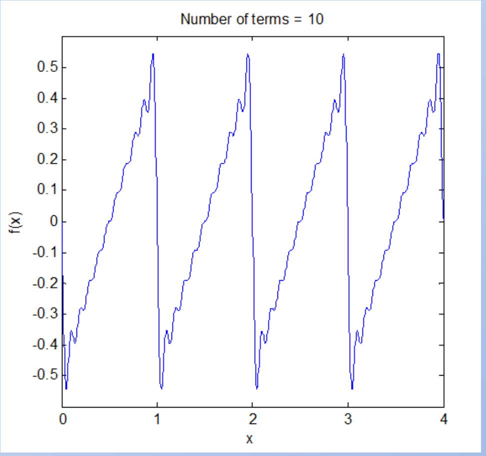
Two terms:



Three terms:



Ten terms:



Function Concept

So far:

Have used MATLAB's built-in functions; for example

sin(x), exp(x), abs(x), . . .

Function:

Reusable script

Sometimes called "user-defined function" or subprogram

Can be used just like built-in MATLAB functions

Useful as a building block for larger programs

Computes an output from an input

Building Block Concept

% Main Program
function

------
function #1

------
function #2

------
function #3

------

function #1

------

function #2

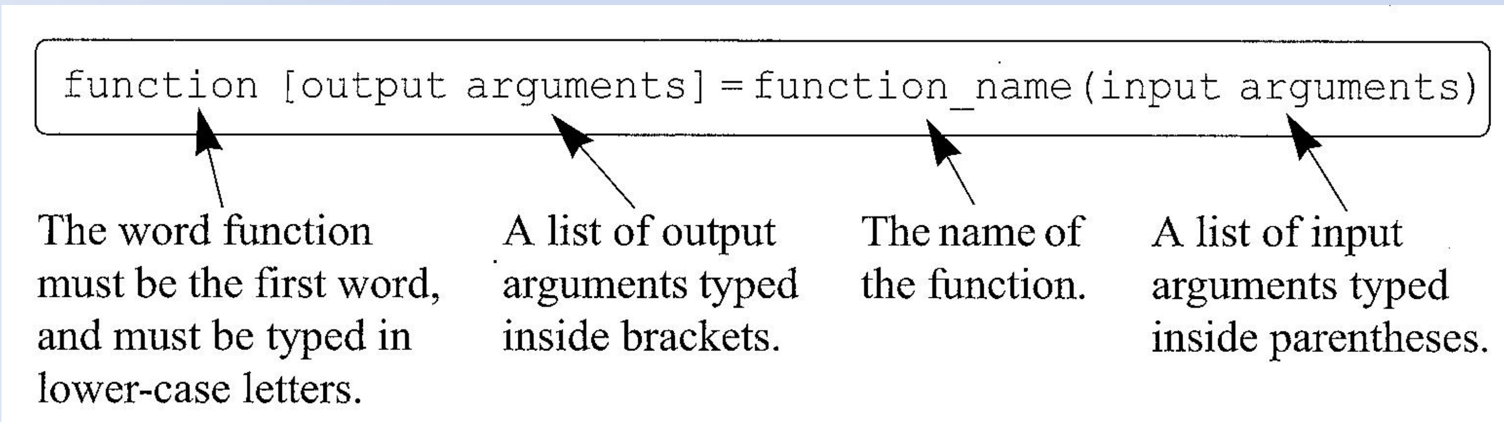
------

function #3

Functions can be in separate files with filename: "function_name.m" or can be in the same file as main program

Function File Format

First line of the file must be of the form:



If no input or output arguments are needed:

function function_name

Input to Functions

- Used to transfer data into the function from the workspace
  - Workspace variables are unavailable within the function
  - Any necessary variables must be brought in
- For multiple inputs:
  - Separate them by commas
  - Order is important
- Examples of built-in functions with inputs:

sum(x) **plot(x,y)**

Output from Functions

Used to transfer results back into the workspace from the function

For multiple outputs:

- Separate them by commas in brackets
- Order is important

Output variables must be assigned

Examples of built-in functions with outputs:

```
y = sum(x)
[value,location] = max(x)
```

Saving & Using Function Files

- Function files must be saved before they can be used.
- Files are saved with the same extension ".m" as used for script files.
- If saved separately, it is recommend that the file be saved with the same name as the function name.
- The function file can be called from: the Command Window, or a script file, or another function.
- To use a saved function, it must be in the current folder

Function File Structure

- Each function can be a single file
  - Convenient for large programs where tasks are broken into smaller building blocks that are tested independently
- Or, can stack functions in one file (as done in Lecture_7.m)
  - All blocks in file must be functions (including main program)
  - Useful for mailing and testing

THE UNIVERSITY OF RHODE ISLAND

Example 7 – Convert Degrees to Radians

```
function y=deg2rad(x)
% DEG2RAD converts degrees to radians
y=x*pi/180;
```

Functions are “called” just like a built-in functions
Application is independent of the variable
    names within the function (x,y)
Executed by typing function_name (input)

Example 7 – Convert Degrees to Radians (cont)

“Main” program

```
angle_0_deg_in_radians=deg2rad(0)
angle_45_deg_in_radians=deg2rad(45)
angle_90_deg_in_radians=deg2rad(90)
angle_135_deg_in_radians=deg2rad(135)
angle_180_deg_in_radians=deg2rad(180)
```

Command Window

```
angle_0_deg_in_radians =
     0
angle_45_deg_in_radians =
    0.7854
angle_90_deg_in_radians =
    1.5708
angle_135_deg_in_radians =
    2.3562
angle_180_deg_in_radians =
    3.1416
```

Example 8 – User function to approximate sin(x)

Truncated Taylor series (replace ∞ with N)

$$\sin x = \sum_{n=1}^{\infty} (-1)^{n+1} \frac{x^{2n-1}}{(2n-1)!} \approx \sum_{n=1}^{N} (-1)^{n+1} \frac{x^{2n-1}}{(2n-1)!}$$

User function

```
function y=Taylor_sin(x,N)
% Taylor_sin computes Taylor sine series at x up to N terms
sum=0;
n=1;
for i=1:N
    sum=sum+(-1)^(n+1)*x.^(2*n-1)/factorial(2*n-1);
    n=n+1;
end
y=sum;
```

Example 8 – User function to approximate sin(x) (cont)

"Main" program

```
sin(pi/4)
Taylor_sin(pi/4,1)
Taylor_sin(pi/4,2)
Taylor_sin(pi/4,3)
Taylor_sin(pi/4,4)
Taylor_sin(pi/4,5)
error=sin(pi/4)-Taylor_sin(pi/4,5)
```

Command Window

```
ans =
   0.707106781186547
ans =
   0.785398163397448
ans =
   0.704652651209168
ans =
   0.707143045779360
ans =
   0.707106469575178
ans =
   0.707106782936867
error =
    -1.750319666982136e-09
```