# CHAPTER 3

**3.1** The M-file can be written as

```
function coscomp(x,n)
i = 1;
tru = cos(x);
ser = 0;
fprintf('\n');
fprintf('order  true value     approximation     error\n');
while (1)
  if i > n, break, end
  ser = ser + (-1)^(i - 1) * x^(2*i-2) / factorial(2*i-2);
  er = (tru - ser) / tru * 100;
  fprintf('%3d  %14.10f  %14.10f  %12.8f\n',i,tru,ser,er);
  i = i + 1;
end
```

This function can be used to evaluate the test case,

```
>> coscomp(1.5,8)

order  true value     approximation     error
  1    0.0707372017    1.0000000000  -1313.68329030
  2    0.0707372017   -0.1250000000  276.71041129
  3    0.0707372017    0.0859375000  -21.48840776
  4    0.0707372017    0.0701171875    0.87650367
  5    0.0707372017    0.0707528251   -0.02208652
  6    0.0707372017    0.0707369341    0.00037823
  7    0.0707372017    0.0707372050   -0.00000469
  8    0.0707372017    0.0707372016    0.00000004
```

**3.2** The M-file can be written as

```
function futureworth(P, i, n)
nn=0:n;
F=P*(1+i).^nn;
y=[nn;F];
fprintf('\n  year   future worth\n');
fprintf('%5d %14.2f\n',y);
```

This function can be used to evaluate the test case,

```
>> futureworth(100000,0.06,7)

  year   future worth
    0      100000.00
    1      106000.00
    2      112360.00
    3      119101.60
    4      126247.70
    5      133822.56
    6      141851.91
    7      150363.03
```

**3.3** The M-file can be written as

```
function annualpayment(P, i, n)
nn = 1:n;
A = P*i*(1+i).^nn./((1+i).^nn-1);
y = [nn;A];
fprintf('\n  year   annual payment\n');
fprintf('%5d %14.2f\n',y);
```

This function can be used to evaluate the test case,

```
>> annualpayment(55000,.066,5)

  year   annual payment
    1       58630.00
    2       30251.49
    3       20804.86
    4       16091.17
    5       13270.64
```

**3.4** The M-file can be written as

```
function Ta = avgtemp(Tm, Tp, ts, te)
w = 2*pi/365;
t = ts:te;
T = Tm + (Tp-Tm)*cos(w*(t-205));
Ta = mean(T);
```

This function can be used to evaluate the test cases,

```
>> avgtemp(22.1,28.3,0,59)
ans =
   16.2148

>> avgtemp(10.7,22.9,180,242)
ans =
   22.2491
```

**3.5** The M-file can be written as

```
function Vol = tankvolume(R, d)
if d < R
  Vol = pi * d ^ 3 / 3;
elseif d <= 3 * R
  V1 = pi * R ^ 3 / 3;
  V2 = pi * R ^ 2 * (d - R);
  Vol = V1 + V2;
else
  Vol = 'overtop';
end
```

This function can be used to evaluate the test cases,

```
>> tankvolume(1.5,1)
ans =
    1.0472

>> tankvolume(1.5,2)
ans =
    7.0686

>> tankvolume(1.5,4.5)
ans =
   24.7400

>> tankvolume(1.5,4.6)
ans =
overtop
```

**3.6** The M-file can be written as

```
function [r, th] = polar(x, y)
r = sqrt(x .^ 2 + y .^ 2);
if x > 0
  th = atan(y/x);
elseif x < 0
  if y > 0
    th = atan(y / x) + pi;
  elseif y < 0
    th = atan(y / x) - pi;
  else
    th = pi;
  end
else
  if y > 0
    th = pi / 2;
  elseif y < 0
    th = -pi / 2;
  else
    th = 0;
  end
end
th = th * 180 / pi;
```

This function can be used to evaluate the test cases. For example, for the first case,

```
>> [r,th]=polar(1,0)
r =
     1
th =
     0
```

All the cases are summarized as

| $x$ | $y$ | $r$ | $\theta$ |
|---|---|---|---|
| 1 | 0 | 1.0000 | 0 |

| | | | |
|---|---|---|---|
| 1 | 1 | 1.4142 | 45 |
| 0 | 1 | 1.0000 | 90 |
| –1 | 1 | 1.4142 | 135 |
| –1 | 0 | 1.0000 | 180 |
| –1 | –1 | 1.4142 | –135 |
| 0 | 0 | 0.0000 | 0 |
| 0 | –1 | 1.0000 | –90 |
| 1 | –1 | 1.4142 | –45 |

**3.7** The M-file can be written as

```
function polar2(x, y)
r = sqrt(x .^ 2 + y .^ 2);
n = length(x);
for i = 1:n
  if x(i) > 0
    th(i) = atan(y(i) / x(i));
  elseif x(i) < 0
    if y(i) > 0
      th(i) = atan(y(i) / x(i)) + pi;
    elseif y(i) < 0
      th(i) = atan(y(i) / x(i)) - pi;
    else
      th(i) = pi;
    end
  else
    if y(i) > 0
      th(i) = pi / 2;
    elseif y(i) < 0
      th(i) = -pi / 2;
    else
      th(i) = 0;
    end
  end
  th(i) = th(i) * 180 / pi;
end
ou=[x;y;r;th];
fprintf('\n     x        y       radius     angle\n');
fprintf('%8.2f %8.2f %10.4f %10.4f\n',ou);
```

This function can be used to evaluate the test cases and display the results in tabular form,

```
>> x=[1 1 0 -1 -1 -1 0 0 1];
>> y=[0 1 1 1 0 -1 0 -1 -1];
>> polar2(x,y)

     x        y       radius     angle
    1.00     0.00     1.0000     0.0000
    1.00     1.00     1.4142    45.0000
    0.00     1.00     1.0000    90.0000
   -1.00     1.00     1.4142   135.0000
   -1.00     0.00     1.0000   180.0000
   -1.00    -1.00     1.4142  -135.0000
```

```
    0.00      0.00      0.0000      0.0000
    0.00     -1.00      1.0000    -90.0000
    1.00     -1.00      1.4142    -45.0000
```

**3.8** The M-file can be written as

```
function grade = lettergrade(score)
if score >= 90
  grade = 'A';
elseif score >= 80
  grade = 'B';
elseif score >= 70
  grade = 'C';
elseif score >= 60
  grade = 'D';
else
  grade = 'F';
end
```

This function can be tested with a few cases,

```
>> lettergrade(95)
ans =
A

>> lettergrade(45)
ans =
F

>> lettergrade(80)
ans =
B
```

3.9 The M-file can be written as

```
function Manning(A)
A(:,5)=sqrt(A(:,2))./A(:,1).*(A(:,3).*A(:,4)./(A(:,3)+2*A(:,4))).^(2/3);
fprintf('\n    n         S          B          H          U\n');
fprintf('%8.3f %8.4f %10.2f %10.2f %10.4f\n',A');
```

This function can be run to create the table,

```
>> A=[.035 .0001 10 2
.020 .0002 8 1
.015 .001 20 1.5
.03 .0007 24 3
.022 .0003 15 2.5];
>> Manning(A)

    n         S          B          H          U
   0.035   0.0001      10.00       2.00     0.3624
   0.020   0.0002       8.00       1.00     0.6094
   0.015   0.0010      20.00       1.50     2.5167
   0.030   0.0007      24.00       3.00     1.5809
```

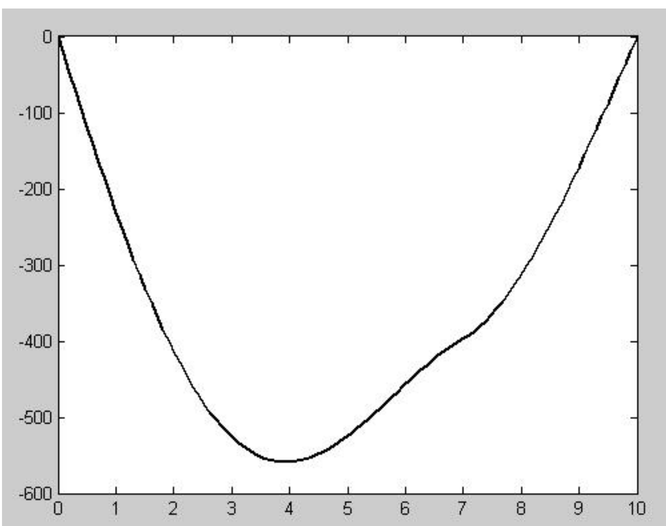
```
   0.022   0.0003      15.00       2.50     1.1971
```

**3.10** The M-file can be written as

```
function beam(x)
xx = linspace(0,x);
n=length(xx);
for i=1:n
  uy(i) = -5/6.*(sing(xx(i),0,4)-sing(xx(i),5,4));
  uy(i) = uy(i) + 15/6.*sing(xx(i),8,3) + 75*sing(xx(i),7,2);
  uy(i) = uy(i) + 57/6.*xx(i)^3 - 238.25.*xx(i);
end
plot(xx,uy)

function s = sing(xxx,a,n)
if xxx > a
  s = (xxx - a).^n;
else
  s=0;
end
```

This function can be run to create the plot,

```
>> beam(10)
```

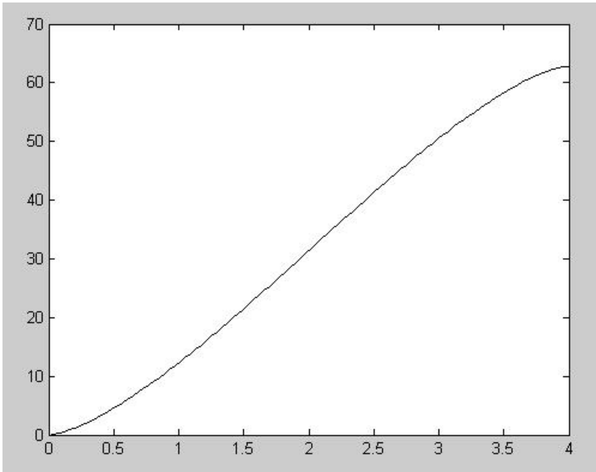



**3.11** The M-file can be written as

```
function cylinder(r, L)
h = linspace(0,2*r);
V = (r^2*acos((r-h)./r)-(r-h).*sqrt(2*r*h-h.^2))*L;
plot(h, V)
```

This function can be run to the plot,

```
>> cylinder(2,5)
```




**3.12** The unvectorized version generates the following values for t and v:

```
t =
     0     4     8    12    16    20
y =
   15.0000   11.5451    5.9549    5.9549   11.5451   15.0000
```

A vectorized version that generates the same results can be written as

```
tt=tstart:(tend-tstart)/ni:tend
yy=10+5*cos(2*pi*tt/(tend-tstart))
```

**3.13**

```
function s=SquareRoot(a,eps)
ind=1;
if a ~= 0
  if a < 0
    a=-a;ind=j;
  end
  x = a / 2;
  while(1)
    y = (x + a / x) / 2;
    e = abs((y - x) / y);
    x = y;
    if e < eps, break, end
  end
  s = x;
else
  s = 0;
end
s=s*ind;
```

The function can be tested:

```
>> SquareRoot(0,1e-4)
ans =
```

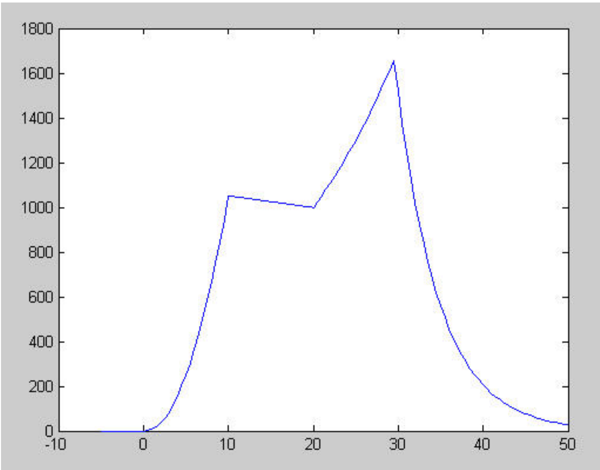
```
     0

>> SquareRoot(2,1e-4)
ans =
    1.4142

>> SquareRoot(4,1e-4)
ans =
     2

>> SquareRoot(-9,1e-4)
ans =
        0 + 3.0000i
```

**3.14** The following function implements the piecewise function:

```
function v = vpiece(t)
if t<0
  v = 0;
elseif t<10
  v = 11*t^2 - 5*t;
elseif t<20
  v = 1100 - 5*t;
elseif t<30
  v = 50*t + 2*(t - 20)^2;
else
  v = 1520*exp(-0.2*(t-30));
end
```

Here is a script that uses `vpiece` to generate the plot

```
k=0;
for i = -5:.5:50
  k=k+1;
  t(k)=i;
  v(k)=vpiece(t(k));
end
plot(t,v)
```



**3.15** This following function employs the round to larger method. For this approach, if the number is exactly halfway between two possible rounded values, it is always rounded to the larger number.

```
function xr=rounder(x, n)
if n < 0,error('negative number of integers illegal'),end
xr=round(x*10^n)/10^n;
```

Here are the test cases:

```
>> rounder(467.9587,2)
ans =
  467.9600
>> rounder(-467.9587,2)
ans =
 -467.9600
>> rounder(0.125,2)
ans =
    0.1300
>> rounder(0.135,2)
ans =
    0.1400
>> rounder(-0.125,2)
ans =
   -0.1300
>> rounder(-0.135,2)
ans =
   -0.1400
```

A preferable approach is called *banker's rounding* or *round to even*. In this method, a number exactly midway between two possible rounded values returns the value whose rightmost digit is even. Here is as function that implements banker's rounding along with the test cases that illustrate how it differs from `rounder`:

```
function xr=rounderbank(x, n)
if n < 0,error('negative number of integers illegal'),end
x=x*10^n;
if mod(floor(abs(x)),2)==0 & abs(x-floor(x))==0.5
  xr=round(x/2)*2;
else
  xr=round(x);
end
xr=xr/10^n;

>> rounderbank(0.125,2)
ans =
    0.1200
>> rounderbank(0.135,2)
ans =
    0.1400

>> rounderbank(-0.125,2)
ans =
```

```
   -0.1200
>> rounderbank(-0.135,2)
ans =
   -0.1400
```

**3.16**

```
function nd = days(mo, da, leap)
nd = 0;
for m=1:mo-1
  switch m
    case {1, 3, 5, 7, 8, 10, 12}
      nday = 31;
    case {4, 6, 9, 11}
      nday = 30;
    case 2
      nday = 28+leap;
  end
  nd=nd+nday;
end
nd = nd + da;

>> days(1,1,0)
ans =
     1
>> days(2,29,1)
ans =
    60
>> days(3,1,0)
ans =
    60
>> days(6,21,0)
ans =
   172
>> days(12,31,1)
ans =
   366
```

**3.17**

```
function nd = days(mo, da, year)
leap = 0;
if year / 4 - fix(year / 4) == 0, leap = 1; end
nd = 0;
for m=1:mo-1
  switch m
    case {1, 3, 5, 7, 8, 10, 12}
      nday = 31;
    case {4, 6, 9, 11}
      nday = 30;
    case 2
      nday = 28+leap;
  end
  nd=nd+nday;
end
nd = nd + da;
```

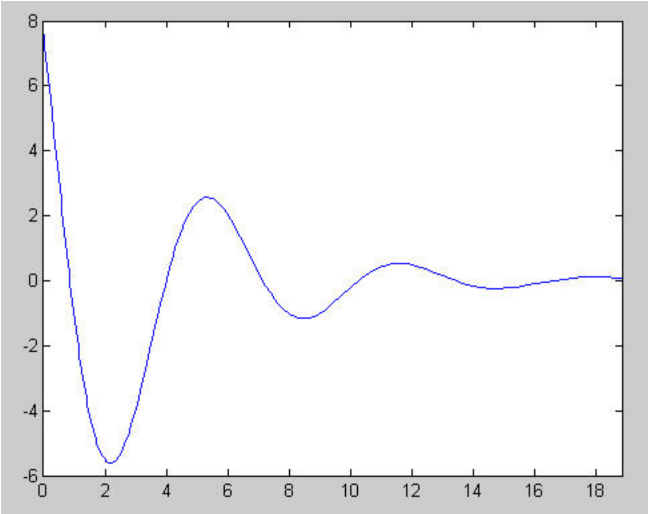
```
>> days(1,1,1999)
ans =
     1
>> days(2,29,2000)
ans =
    60
>> days(3,1,2001)
ans =
    60
>> days(6,21,2002)
ans =
   172
>> days(12,31,2004)
ans =
   366
```

**3.18**

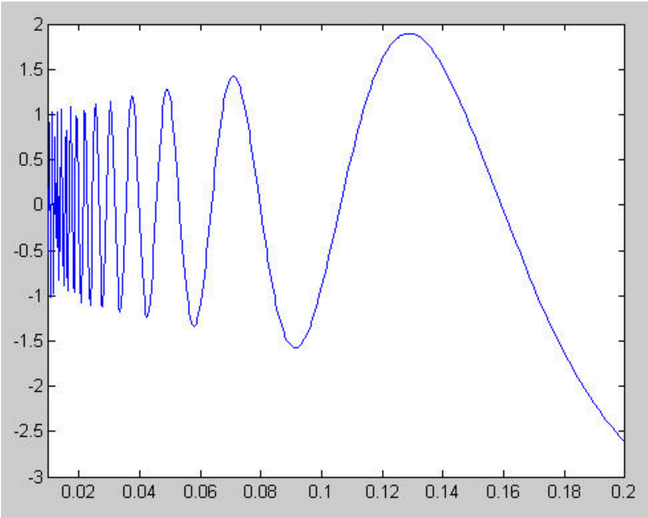
```
function fr = funcrange(f,a,b,n,varargin)
% funcrange: function range and plot
%   fr=funcrange(f,a,b,n,varargin): computes difference
%      between maximum and minimum value of function over
%      a range. In addition, generates a plot of the function.
% input:
%   f = function to be evaluated
%   a = lower bound of range
%   b = upper bound of range
%   n = number of intervals
% output:
%   fr = maximum - minimum
x = linspace(a,b,n);
y = f(x,varargin{:});
fr = max(y)-min(y);
fplot(f,[a b],varargin{:})
end
```

**(a)**

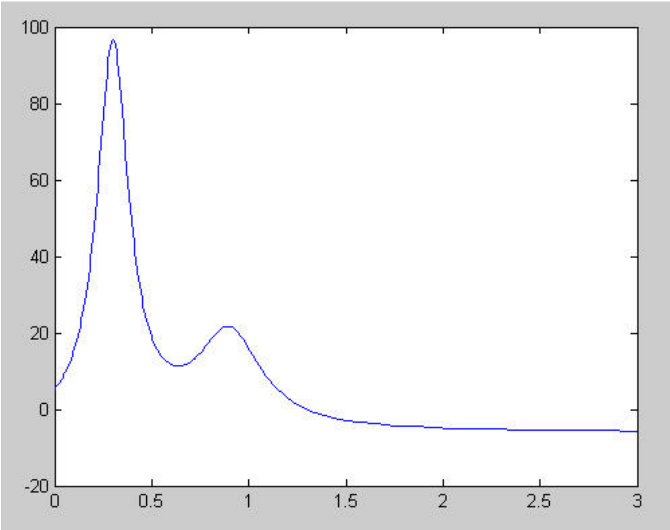
```
>> f=@(t) 10*exp(-0.25*t).*sin(t-4);
>> funcrange(f,0,6*pi,1000)
ans =
   13.1873
```




**(b)**
```
>> f=@(x) exp(5*x).*sin(1./x);
>> funcrange(f,0.01,0.2,1000)
ans =
    4.5031
```




**(c)**
```
>> funcrange(@humps,0,3,1000)
ans =
  102.1396
```




**3.19**

```
function yend = odesimp(dydt, dt, ti, tf, yi, varargin)
% odesimp: Euler ode solver
% yend = odesimp(dydt, dt, ti, tf, yi, varargin):
%   Euler's method solution of a single ode
% input:
%   dydt = function defining ode
%   dt = time step
%   ti = initial time
%   tf = final time
%   yi = initial value of dependent variable
% output:
%   yend = dependent variable at final time
t = ti; y = yi; h = dt;
while (1)
  if t + dt > tf, h = tf - t; end
  y = y + dydt(y,varargin{:}) * h;
  t = t + h;
  if t >= tf, break, end
end
yend = y;
```

<u>test run:</u>

```
>> dvdt=@(v,m,cd) 9.81-(cd/m)*v^2;
>> odesimp(dvdt,0.5,0,12,0,68.1,0.25)

ans =
   50.9259
```