Question #1. - 8 marks

(a) Create a working copy of the MATLAB function Euler in your installation of MATLAB. DELIVERABLES: A copy of the M-FILE in your pdf.

```
function Euler(m,c,g,t0,v0,tn,n)
%print headings and initial conditions
fprintf('values of t approximations v(t)\n')
fprintf('%8.3f',t0),fprintf('%19.4f\n',v0)

%compute step size h
h = (tn - t0) / n

%set t,v to the initial values
t=t0;
v=v0;

%compute v(t) over n time steps using Eurler's method
for i=1:n
   v=v+(g-c/m*v)*h;
   t=t+h;
   fprintf('%8.2f',t),fprintf('%19.4f\n',v)
end
```

(b) Use Euler to solve the differential equation using m = 86.2, c = 12.5 and initial conditions v(0) = 0 on the time interval [0, 12] using 15 time steps and g = 9.81. DELIVERABLES: Your answer should include the function call to Euler and the resulting output.

```
>> Euler (86.2,12.5,9.81,0,0,12,15)
values of t approximations v(t)
  0.000           0.0000

h =

    0.8000

    0.80           7.8480
    1.60          14.7856
    2.40          20.9183
    3.20          26.3396
    4.00          31.1319
    4.80          35.3684
    5.60          39.1133
    6.40          42.4238
    7.20          45.3502
    8.00          47.9372
    8.80          50.2240
    9.60          52.2456
   10.40          54.0326
```

```
11.20     55.6123
12.00     57.0088
```

(c) Use Euler for a falling parachutist with the same parameters as Q1b but with a gravitational constant of 3.71 (as would be the case if the parachutist was falling on Mars). DELIVERABLES: Your answer should include the function call to Euler and the resulting output. 3

```
>> Euler (86.2,12.5,3.71,0,0,12,15)
values of t approximations v(t)
  0.000       0.0000

h =

   0.8000

   0.80       2.9680
   1.60       5.5917
   2.40       7.9110
   3.20       9.9612
   4.00      11.7737
   4.80      13.3758
   5.60      14.7921
   6.40      16.0441
   7.20      17.1508
   8.00      18.1292
   8.80      18.9940
   9.60      19.7585
  10.40      20.4343
  11.20      21.0318
  12.00      21.5599
```

(d) Use MATLAB to compute the relative error |εt | in the computed approximation at time t = 12, using the constants from Q1b. To do this, use the true (exact) solution: v(t) = gm c (1 − e − ct m) (1) DELIVERABLES: A copy of the commands and output. Note that in MATLAB e x is computed as exp(x) and |x| as abs(x).

```
>> c=12.5;
>> m=86.2;
>> g=9.81;
>> t=12;
>> v=g*m/c*(1-exp(-c*t/m));
>> v

v =

  55.7775
>> E=abs((55.7775-57.0088)/55.7775)

E =

   0.0221 ≈ 2.21%
```

Question #2 - 6 Marks.

In our mathematical model of a falling parachutist, instead of assuming that air resistance is linearly proportional to velocity (that is, FU = −cv), you might choose to model the upward force on the parachutist as a second-order relationship,

FU = −kv2 where k is a second-order drag coefficient.

This leads to the following differential equation

dv dt = g − k m v 2

(a) Modify the MATLAB function Euler in Question 1 so that it will use Eulers method to solve this differential equation. Use the function header
Euler2(m , k , g, t0 , v0 , tn , n)

DELIVERABLES: A copy of the M-FILE in your pdf.

```
function Euler2(m,k,g,t0,v0,tn,n)
%print headings and initial conditions
fprintf('values of t approximations v(t)\n')
fprintf('%8.3f',t0),fprintf('%19.4f\n',v0)

%compute step size h
h = (tn - t0) / n

%set t,v to the initial values
t=t0;
v=v0;

%compute v(t) over n time steps using Eurler's method
for i=1:n
   v=v+(g-((k/m)*v^2))*h;
   t=t+h;
   fprintf('%8.2f',t),fprintf('%19.4f\n',v)

end
```

(b) Use Euler2 to compute a numerical approximation to the above differential equation using m = 73.5, k = 0.234 and initial condition v(0) = 0 on the time interval [0, 18] using 72 time steps.
DELIVERABLES: The function call to Euler2 and the resulting output.

```
>> Euler2(73.5,0.234,3.71,0,0,18,72)
values of t approximations v(t)
  0.000           0.0000

h =

   0.2500

   0.25          0.9275
   0.50          1.8543
   0.75          2.7791
```

1.00	3.7004
1.25	4.6170
1.50	5.5276
1.75	6.4307
2.00	7.3253
2.25	8.2101
2.50	9.0840
2.75	9.9458
3.00	10.7946
3.25	11.6293
3.50	12.4492
3.75	13.2533
4.00	14.0410
4.25	14.8116
4.50	15.5645
4.75	16.2992
5.00	17.0152
5.25	17.7123
5.50	18.3901
5.75	19.0484
6.00	19.6871
6.25	20.3061
6.50	20.9055
6.75	21.4851
7.00	22.0452
7.25	22.5859
7.50	23.1074
7.75	23.6099
8.00	24.0937
8.25	24.5592
8.50	25.0066
8.75	25.4364
9.00	25.8490
9.25	26.2446
9.50	26.6239
9.75	26.9873
10.00	27.3351
10.25	27.6679
10.50	27.9861
10.75	28.2902
11.00	28.5807
11.25	28.8581
11.50	29.1227
11.75	29.3752
12.00	29.6159
12.25	29.8453
12.50	30.0638
12.75	30.2719
13.00	30.4701

13.25	30.6586
13.50	30.8380
13.75	31.0086
14.00	31.1708
14.25	31.3250
14.50	31.4715
14.75	31.6106
15.00	31.7428
15.25	31.8684
15.50	31.9875
15.75	32.1006
16.00	32.2080
16.25	32.3098
16.50	32.4065
16.75	32.4981
17.00	32.5850
17.25	32.6674
17.50	32.7456
17.75	32.8196
18.00	32.8898

(c) Use the fact the exact analytic solution of this problem is v(t) = r gm k tanh r gk m t ! to compute (either in MATLAB or using your calculator) the relative error in the computed solution at t = 18.

```
>> m=73.5;
>> k=0.234;
>> g=3.71;
>> t=18;
>> vt=sqrt((g*m)/k)*tanh(sqrt((g*k/m))*t);
>> vt

vt =

  32.7987

>> E=abs((32.7987-32.8898)/32.7987)

E =

   0.0028 ≈ 0.28%
```

Question #3 - 4 Marks

The function e −x can be approximated by its McLaurin series expansion as follows (note the alternating + and −): e −x ≈ 1 − x + x 2 2! − x 3 3! + . . . ± x n n! 4 Alternatively, note that e −x = 1 e x . Thus, e −x can also be approximated by 1 over the McLaurin series expansion of e x . That is,

e −x ≈ 1 1 + x + x2 2! + x3 3! + . . . + xn n!

Approximate e −2 using both approaches above for n = 1, 2, 3, 4 and 5. Note, n is the degree of the polynomial not the number of terms. So here you use 2 terms, then 3 terms, ..., and finally 6 terms. Compare each approximation to the true value of e −2 = 0.135335..., using the true relative error. What conclusions can you make about the two approaches?

```
function MSE(x)
%UNTITLED3 Summary of this function goes here
%   Detailed explanation goes here
sign = 1;
equation = 1;
n=1;
while n < 6
    equation = equation - ((x^n)/factorial(n))*sign;
    sign = sign*-1;
    error = abs((0.135335-equation)/0.135335);
    fprintf('n: %7.5f ',n),fprintf(' e^-x= %7.5f\n', equation)
    fprintf('Relative error: %7.5f\n',error)
    n=n+1;

end
>> MSE(2)
n: 1.00000  e^-x= -1.00000
Relative error: 8.38907
n: 2.00000  e^-x= 1.00000
Relative error: 6.38907
n: 3.00000  e^-x= -0.33333
Relative error: 3.46302
n: 4.00000  e^-x= 0.33333
Relative error: 1.46302
n: 5.00000  e^-x= 0.06667
Relative error: 0.50740

function MSE2(x)
equation = 1;
n=1;
while n<6
    equation = 1/(equation + (x^n/factorial(n)));
    error = abs((0.135335-equation)/0.135335);
    fprintf('n: %7.5f ',n),fprintf(' e^-x= %7.5f\n', equation)
    fprintf('Relative error: %7.5f\n',error)
    n=n+1;
    equation= 1/equation;
end
>> MSE2(2)
n: 1.00000  e^-x= 0.33333
Relative error: 1.46302
n: 2.00000  e^-x= 0.20000
Relative error: 0.47781
n: 3.00000  e^-x= 0.15789
Relative error: 0.16670
n: 4.00000  e^-x= 0.14286
Relative error: 0.05558
n: 5.00000  e^-x= 0.13761
```

Relative error: 0.01684

Conclusion: For approach 1, at n=5, the relative error is over 50%, whereas for approach 2 the relative error is only 1.68%. Hence, we can conclude approach 2 gives a more accurate approximation.