Assignment 7

Table of Contents

Note: Problem 2 appears at the very end, due to the function call.

Problem 3 - Numerical Integration

```
clc; close all; clear all;

x0 = 0; xn = 30;
force = @(x) 1.6*x - 0.045*x^2;
theta = @(x) 0.8 + 0.125*x - 0.009*x^2 + 0.0002*x^3;
work = @(x) force(x)*cos(theta(x));

% Part a) Trapezoidal Rule
% 4 segments
n = 4;
sum = 0; x = x0 + (xn-x0)/n;
for i = 1:(n-1)
    sum = sum + work(x);
    x = x + (xn-x0)/n;
end
integral_f = (xn - x0)/(2*n)*(work(x0) + work(xn) + 2*sum);
fprintf('Using trapezoidal rule with %d segments, the work done by
 friction is about %.4f.\n', n, integral_f);

% 8 segments
n = 8; x0 = 0; xn = 30;
sum = 0; x = x0 + (xn-x0)/n;
for i = 1:(n-1)
    sum = sum + work(x);
    x = x + (xn-x0)/n;
end
integral_f = (xn - x0)/(2*n)*(work(x0) + work(xn) + 2*sum);
fprintf('Using trapezoidal rule with %d segments, the work done by
 friction is about %.4f.\n', n, integral_f);

% 16 segments
n = 16; x0 = 0; xn = 30;
sum = 0; x = x0 + (xn-x0)/n;
for i = 1:(n-1)
    sum = sum + work(x);
    x = x + (xn-x0)/n;
end
integral_f = (xn - x0)/(2*n)*(work(x0) + work(xn) + 2*sum);
```

```
fprintf('Using trapezoidal rule with %d segments, the work done by
 friction is about %.4f.\n\n', n, integral_f);


% Part b) Simpson's 1/3 Rule
% 4 segments
n = 4; x0 = 0; xn = 30; h = (xn-x0)/n;
sum = 0; x2 = x0 + (xn-x0)/n;
for i = 1:(n-1)
    sum = sum + (x2-x0)/6*(work(x0) + work(x2) + 4*work((x2+x0)/2));
    x0 = x2;
    x2 = x0 + h;
end
fprintf('Using Simpson''s rule with %d segments, the work done by
 friction is about %.4f.\n', n, sum);

% 8 segments
n = 8; x0 = 0; xn = 30; h = (xn-x0)/n;
sum = 0; x2 = x0 + h;
for i = 1:(n-1)
    sum = sum + (x2-x0)/6*(work(x0) + work(x2) + 4*work((x2+x0)/2));
    x0 = x2;
    x2 = x0 + h;
end
fprintf('Using Simpson''s rule with %d segments, the work done by
 friction is about %.4f.\n', n, sum);

% 16 segments
n = 16; x0 = 0; xn = 30; h = (xn-x0)/n;
sum = 0; x2 = x0 + (xn-x0)/n;
for i = 1:(n-1)
    sum = sum + (x2-x0)/6*(work(x0) + work(x2) + 4*work((x2+x0)/2));
    x0 = x2;
    x2 = x0 + h;
end
fprintf('Using Simpson''s rule with %d segments, the work done by
 friction is about %.4f.\n\n', n, sum);


% Part c) 2-point Gauss Quadrature Rule
% 4 segments
n = 4; x0 = 0; xn = 30; h = (xn-x0)/n;
sum = 0; x1 = x0 + h;
for i = 1:(n-1)
    sum = sum + h/2*(work(.5*(-1/sqrt(3)*(x1-x0) + (x1+x0))) +
 work(.5*(1/sqrt(3)*(x1-x0)+(x1+x0))));
    x0 = x1;
    x1 = x0 + h;
end
fprintf('Using 2-point Gauss Quadrature rule with %d segments, the
 work done by friction is about %.4f.\n', n, sum);

% 8 segments
n = 8; x0 = 0; xn = 30; h = (xn-x0)/n;
```

```
sum = 0; x1 = x0 + h;
for i = 1:(n-1)
    sum = sum + h/2*(work(.5*(-1/sqrt(3)*(x1-x0) + (x1+x0))) +
 work(.5*(1/sqrt(3)*(x1-x0)+(x1+x0))));
    x0 = x1;
    x1 = x0 + h;
end
fprintf('Using 2-point Gauss Quadrature rule with %d segments, the
 work done by friction is about %.4f.\n', n, sum);

% 16 segments
n = 16; x0 = 0; xn = 30; h = (xn-x0)/n;
sum = 0; x1 = x0 + (xn-x0)/n;
for i = 1:(n-1)
    sum = sum + h/2*(work(.5*(-1/sqrt(3)*(x1-x0) + (x1+x0))) +
 work(.5*(1/sqrt(3)*(x1-x0)+(x1+x0))));
    x0 = x1;
    x1 = x0 + h;
end
fprintf('Using 2-point Gauss Quadrature rule with %d segments, the
 work done by friction is about %.4f.\n', n, sum);
```

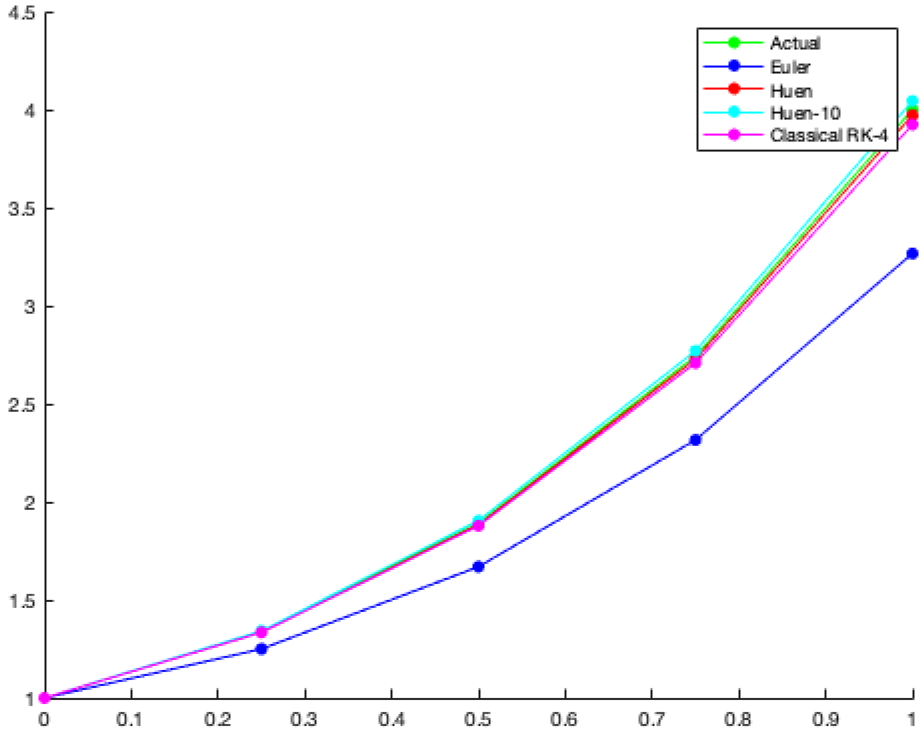
```
Using trapezoidal rule with 4 segments, the work done by friction is
 about 58.7168.
Using trapezoidal rule with 8 segments, the work done by friction is
 about 64.8995.
Using trapezoidal rule with 16 segments, the work done by friction is
 about 66.4193.

Using Simpson's rule with 4 segments, the work done by friction is
 about 61.6363.
Using Simpson's rule with 8 segments, the work done by friction is
 about 69.4781.
Using Simpson's rule with 16 segments, the work done by friction is
 about 69.5822.

Using 2-point Gauss Quadrature rule with 4 segments, the work done by
 friction is about 61.7621.
Using 2-point Gauss Quadrature rule with 8 segments, the work done by
 friction is about 69.4790.
Using 2-point Gauss Quadrature rule with 16 segments, the work done by
 friction is about 69.5820.
```

Problem 4 - Solving ODEs

```
clc; close all; clear all;
x0 = 0; x1 = 1; h = 0.25; y0 = 1;
slope = @(x,y) (1 + 2*x)*(sqrt(y));

% Part b) - Euler's Method
euler = [y0]; x_e = x0; y_e = y0;
while x_e < x1
    y_e = slope(x_e, y_e)*h + y_e;
```

```
    euler = [euler, y_e];
    x_e = x_e + h;
end
euler

% Part c) - Huen's Method, 1 iteration
huen1 = [y0]; x_h = x0; y_old = y0; y_p = 1; y_c = 0;
while x_h < x1
    y_p = slope(x_h, y_old)*h + y_old;
    y_c = y_old + h/2*(slope(x_h, y_old) + slope(x_h + h, y_p));
    huen1 = [huen1, y_c];
    x_h = x_h + h;
    y_old = y_c;
end
huen1

% Part d) - Huen's Method, 10 iterations
huen10 = [y0]; x_h = x0; y_old = y0; y_p = 1;
while x_h < x1
    y_p = slope(x_h, y_old)*h + y_old;
    y_c = y_p;
    for i = 1:10
        y_c = y_old + h/2*(slope(x_h, y_old) + slope(x_h + h, y_c));
    end
    huen10 = [huen10, y_c];
    x_h = x_h + h;
    y_old = y_c;
end
huen10

% Part e) - 4th Order RK
rk4 = [y0]; x_rk = 0; y_rk = y0;
while x_rk < x1
    k1 = slope(x_rk, y_rk);
    k2 = slope(x_rk + .5*h, y_rk + .5*k1*h);
    k3 = slope(x_rk + .5*h, y_rk + .5*k2*h);
    k4 = slope(x_rk + h, y_rk + h);
    y_rk = y_rk + 1/6*(k1 + 2*k2 + 2*k3 +k4)*h;
    rk4 = [rk4, y_rk];
    x_rk = x_rk + h;
end
rk4

hold all
x = linspace(x0, x1, 5);
plot(x, (.5*(x.^2 + x) + 1).^2, '.-g', 'MarkerSize', 15);
plot(x, euler, '.-b', 'MarkerSize', 15);
plot(x, huen1, '.-r', 'MarkerSize', 15);
plot(x, huen10, '.-c', 'MarkerSize', 15);
plot(x, rk4, '.-m', 'MarkerSize', 15);

legend('Actual', 'Euler', 'Huen', 'Huen-10', 'Classical RK-4');
```

```
euler =

    1.0000    1.2500    1.6693    2.3153    3.2663


huen1 =

    1.0000    1.3346    1.8836    2.7277    3.9710


huen10 =

    1.0000    1.3422    1.9045    2.7695    4.0437


rk4 =

    1.0000    1.3345    1.8781    2.7078    3.9235
```



Problem 2 - 3-Point Gauss Quadrature

```
clc; clear all; close all;

x0 = [.5, 1, .5, 1, 0, -1];
```

```
% Initial values guessed based off first 2 equations and expected
 values
% from the textbook.
options =
 optimoptions('fsolve','Display','iter','MaxFunctionEvaluations',100000,'MaxIterat
[x,fval] = fsolve(@Function_F, x0, options);
x
% Note, the indices in x store c0, c1, c2, x0, x1, and x2 respectively

function F = Function_F(x)
F = [];
F(1) = x(1)*x(4)^0 + x(2)*x(5)^0 + x(3)*x(6)^0 - 2;
F(2) = x(1)*x(4)^1 + x(2)*x(5)^1 + x(3)*x(6)^1 - 0;
F(3) = x(1)*x(4)^2 + x(2)*x(5)^2 + x(3)*x(6)^2 - 2/3;
F(4) = x(1)*x(4)^3 + x(2)*x(5)^3 + x(3)*x(6)^3 - 0;
F(5) = x(1)*x(4)^4 + x(2)*x(5)^4 + x(3)*x(6)^4 - 2/5;
F(6) = x(1)*x(4)^5 + x(2)*x(5)^5 + x(3)*x(6)^5 - 0;
end
```

```
                                             Norm of      First-order
 Trust-region
 Iteration  Func-count     f(x)             step          optimality
 radius
     0           7         0.471111                              1.53
        1
     1          14        0.0171979          0.20548            0.182
        1
     2          21      0.000143506         0.200506          0.00746
        1
     3          28      4.98455e-07        0.0559041          0.00093
        1
     4          35      4.27607e-14      0.000751786         2.77e-07
        1

Equation solved.

fsolve completed because the vector of function values is near zero
as measured by the value of the function tolerance, and
the problem appears regular as measured by the gradient.


x =

    0.5556    0.8889    0.5556    0.7746   -0.0000   -0.7746
```

Published with MATLAB® R2020b