Homework #6

Table of Contents

Ian Blackstone, Helena-Nikolai Fujishin
Math 365, Fall 2016

Problems

```
function hmwk1()
hmwk_problem(@prob1,'prob1');
hmwk_problem(@prob2,'prob2');
hmwk_problem(@prob3,'prob3');
hmwk_problem(@prob4,'prob4');
hmwk_problem(@prob5,'prob5');
end
function hmwk_problem(prob,msg)
try
    prob()
    fprintf('% s : Success!\n',msg);
catch me
    fprintf('% s : Something went wrong.\n',msg);
    fprintf('% s\n',me.message);
end
fprintf('\n');
end
```

Problem #1 : Degree of Precision

```
function prob1()
% Part A

% Set bounds of integration
a=0;
b=1;

% set maximum loops.
n = 10;

% Our function to integrate
f = @(x,k) x^k;

% The exact integral.
```

```
exct = @(x,k) 1/(k+1)*x^(k-1);

% Loop to find the highest value of k that gives an accurate answer.
for k = 0:n
T = (b-a)/2*(f(a,k) + f(b,k));
err = abs(exct(b,k)-T);
if err >= 1E-14
 fprintf('m for part a = % .i \n',k-1)
 break
end
end

% Part B: Simpson's rule
c =(a+b)/2;
h = (b-a);

% Loop to find the highest value of k that gives an accurate answer.
for k = 0:n
    M = h*f((a+b)/2,k);
    T = h*(f(a,k)+f(b,k))/2;
S = ((2/3)*M)+((1/3)*T);
err = abs(exct(b,k)-S);
if err >= 1E-14
 fprintf('m for part b = % .i \n',k-1)
 break
end
end

end
```

```
m for part a =  1
m for part b =  3
prob1 : Success!
```

Problem #2 : Gaussian Quadrature

```
function prob2()

% Part a

% Our given evaluation points and weights.
xi = [-0.96816023950763 -0.83603110732664 -0.61337143270059
  -0.32425342340381 0 0.32425342340381 0.61337143270059
 0.83603110732664 0.96816023950763]';
ci = [0.081274388361574 0.18064816069486 0.26061069640294
 0.31234707704 0.33023935500126 0.31234707704 0.26061069640294
 0.18064816069486 0.081274388361574]';

% Define our functions
f1 = @(x) ((x-1).^2).*exp(-(x.^2));
f2 = @(x) 2*(1./(1+x.^2));
```

```
% Calculate the Gaussian quadrature of each function, the exact
 integral, and the absolute error.
GQ1 = sum(ci.*f1(xi));
exact1 = integral(f1,-1,1);
error1 = abs(exact1-GQ1);

GQ2 = sum(ci.*f2(xi));
exact2 = integral(f2,-1,1);
error2 = abs(exact2-GQ2);

% Report the errors.
fprintf('The error for the Gaussian quadrature of function 1 is %.3e.
 \n',error1)
fprintf('The error for the Gaussian quadrature of function 2 is %.3e.
 \n',error2)

% Part b

% For this poart we will call a function stored in an external file:
% function x = simps(a,b,n,f)
% % The function implements the composite Simpson's rule

% h = (b-a)/n;
% x = zeros(1,n+1);
% x(1) = a;
% x(n+1) = b;
% p = 0;
% q = 0;

% % Define the x-vector
% for i = 2:n
%     x(i) = a + (i-1)*h;
% end

% % Define the terms to be multiplied by 4
% for i = 2:((n+1)/2)
%     p = p + (f(x(2*i -2)));
% end

% % Define the terms to be multiplied by 2
% for i = 2:((n-1)/2)
%     q = q + (f(x(2*i -1)));
% end

% % Calculate final output
% x = (h/3)*(f(a) + 2*q + 4*p + f(b));

% end

% Let us set-up the Simpson's Rule.
S1 = simps(-1,1,8,f1);
errorS1 = abs(exact1-S1);
S2 = simps(-1,1,8,f2);
errorS2 = abs(exact2-S2);
```

```
% Report the errors.
fprintf('The error for the Simpsons approximation of function 1 is
 %.3e. \n',errorS1)
fprintf('The error for the Simpsons approximation of function 2 is
 %.3e. \n',errorS2)

% The simpsons rule is less accurate for estimating an integral than
 Gaussian quadrature is.

% Part c

a=-1;
b=1;

% set maximum loops.
n = 20;

% Our function to integrate
f = @(x,k) x.^k;

% The exact integral.
exact = @(x,k) 1/(k+1).*x.^(k-1);

% Loop to find the highest value of k that gives an accurate answer.
for k = 0:n
GQ = sum(ci.*f(xi,k));
err = abs(exact(b,k) - exact(a,k) - GQ);
if err >= 1E-14
 fprintf('The degree of precision for Gausssian quadtrature, m = %.i
 \n',k-1)
 break
end
end

end
```

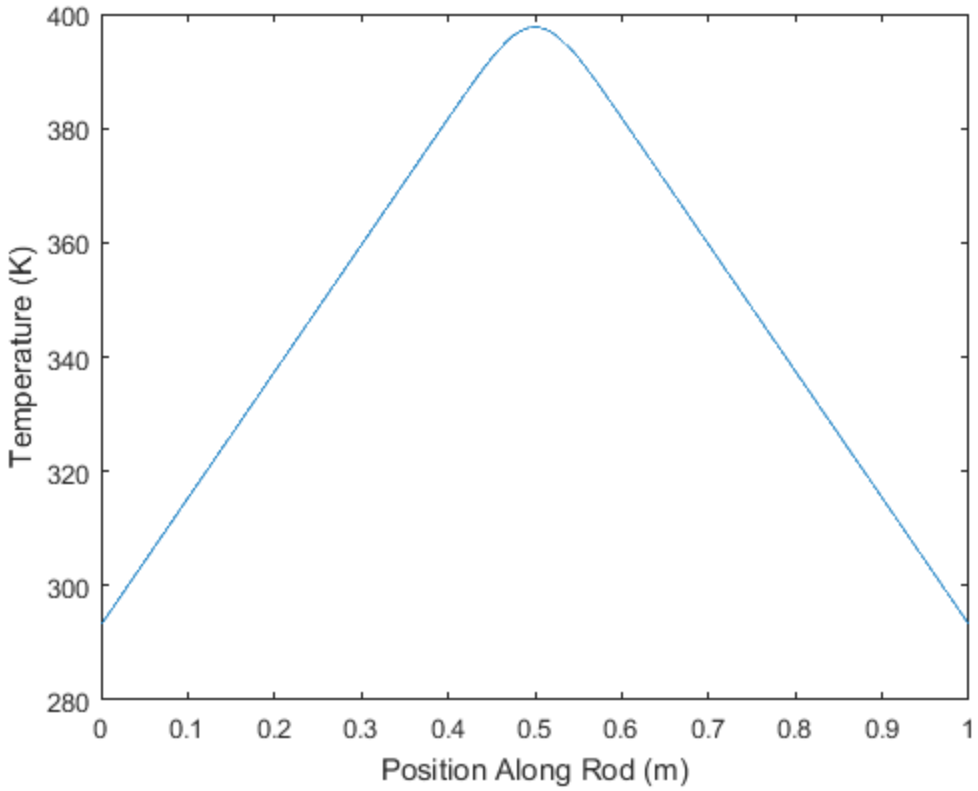
```
The error for the Gaussian quadrature of function 1 is 1.519e-10.
The error for the Gaussian quadrature of function 2 is 6.583e-07.
The error for the Simpsons approximation of function 1 is 4.418e-02.
The error for the Simpsons approximation of function 2 is 6.934e-01.
The degree of precision for Gausssian quadtrature, m = 17
prob2 : Success!
```

Problem #3 : Integral Equations

```
function prob3()

% Part a

% Define parameters and conditions.
N = 1000;
```

```
h = 1/N;
j = 0:N;
a = 293.15;
b = 293.15;
B = 1e-3;
S = 1e4;
Cp = 200;
rho = 10;
eps = 5e-2;

% Generate a list of zeros and set the first and last values to our
 end point conditions.
uj = zeros(N,1);
uj(1) = a;
uj(end) = b;

% Define a function for each integral in u.
f1 = @(x) x.*(-S/(rho*Cp*B)).*(exp(-(((x-(1/2))/(eps)).^2)));
f2 = @(x) (x-1).*(-S/(rho*Cp*B)).*(exp(-(((x-(1/2))/(eps)).^2)));

% Define a function to integrate over to find the temperature at that
 point.
u = @(x) (a-quad(f1,0,x))*(1-x) + (b+quad(f2,x,1))*x;

% Loop over each value of x to find the temperature at that point.
for x = j
 uj(x+1) = u(h.*x);
end

% Plot the data.
figure()
plot(h.*j,uj)
xlabel('Position Along Rod (m)')
ylabel('Temperature (K)')

% Part b

% Define a function that will be zero at the point we will burn
 ourselves.
u2 = @(x) (a-quad(f1,0,x))*(1-x) + (b+quad(f2,x,1))*x - 343.15;

% Use fzero and an initial guess of 0.25 to find a zero.
burn = fzero(u2,0.25);
fprintf('You will burn yourself starting at %.2f meters along the rod.
\n',burn)

end
```

```
You will burn yourself starting at 0.23 meters along the rod.
prob3 : Success!
```



Problem #4 : Quasi-Monte Carlo Integration

```
function prob4()

% Part a

% The exact answer is -512/pi^6 as given by the following code:
syms a b c d e f
exact = int(int(int(int(int(int(sin(pi/2*(a + b + c + d + e +
 f)),a,0,1),b,0,1),c,0,1),d,0,1),e,0,1),f,0,1);

% Part b

% Set the number of points and generate the random points based on a
 seed.
tic;
N = 10000;
rng(11032016);
MCpoints = rand(N,6);

% Define our function.
g = @(a,b,c,d,e,f) sin(pi/2*(a + b + c + d + e + f));

% Set the points for each variable.
a = MCpoints(:,1);
```

```
b = MCpoints(:,2);
c = MCpoints(:,3);
d = MCpoints(:,4);
e = MCpoints(:,5);
f = MCpoints(:,6);

% Calculate the integral using the random points.
MC = sum(g(a,b,c,d,e,f))/N;
MCtime = toc;
fprintf('The Monte Carlo method gives us %.5f, while the exact answer
 is %.5f. The error is %.5f. This calculation took %.5f seconds.
 \n',MC,exact,exact-MC, MCtime)

% Part c

% Generate points from the Halton set.
tic;
HSpoints = net(haltonset(6),N);

% Set each variable to our points from the Halton set
a = HSpoints(:,1);
b = HSpoints(:,2);
c = HSpoints(:,3);
d = HSpoints(:,4);
e = HSpoints(:,5);
f = HSpoints(:,6);

% Calculate the integral based on the Halton set.
HS = sum(g(a,b,c,d,e,f))/N;
HStime = toc;
fprintf('The Halton set gives us %.5f, while the exact answer
 is %.5f. The error is %.5f. This calculation took %.5f seconds.
 \n',HS,exact,exact-HS,HStime)

% Part d

% Define number of points and dimensionality.
tic;
n = 10;
h = 1/n;
d = 6;

% Trapezoidal rule weights in one dimension.
w1 = h*[0.5;ones(n-1,1);0.5];
w = w1;

% Generate the trapezoidal rule weights in 6 dimensions.
for j=1:d-1
 w = bsxfun(@times,w,reshape(w1,[ones(1,j) n+1]));
end

% Grid of points
[x1,x2,x3,x4,x5,x6] = ndgrid(linspace(0,1,n+1));
m = w.*g(x1,x2,x3,x4,x5,x6);
```

```
Qtrap = sum(m(:));
Ttime = toc;
fprintf('The trapezoidal method gives us %.5f, while the exact answer
 is %.5f. The error is %.5f. This calculation took %.5f seconds.
 \n',Qtrap,exact,exact-Qtrap,Ttime)

% Part e

% The trapezoidal rule gives a reasonably accurate answer with several
 orders of magnitude fewer points however the code takes longer than
 the random set to run.
% The Halton set takes more time to run than random points but is an
 order of magnitude more accurate.

end
```

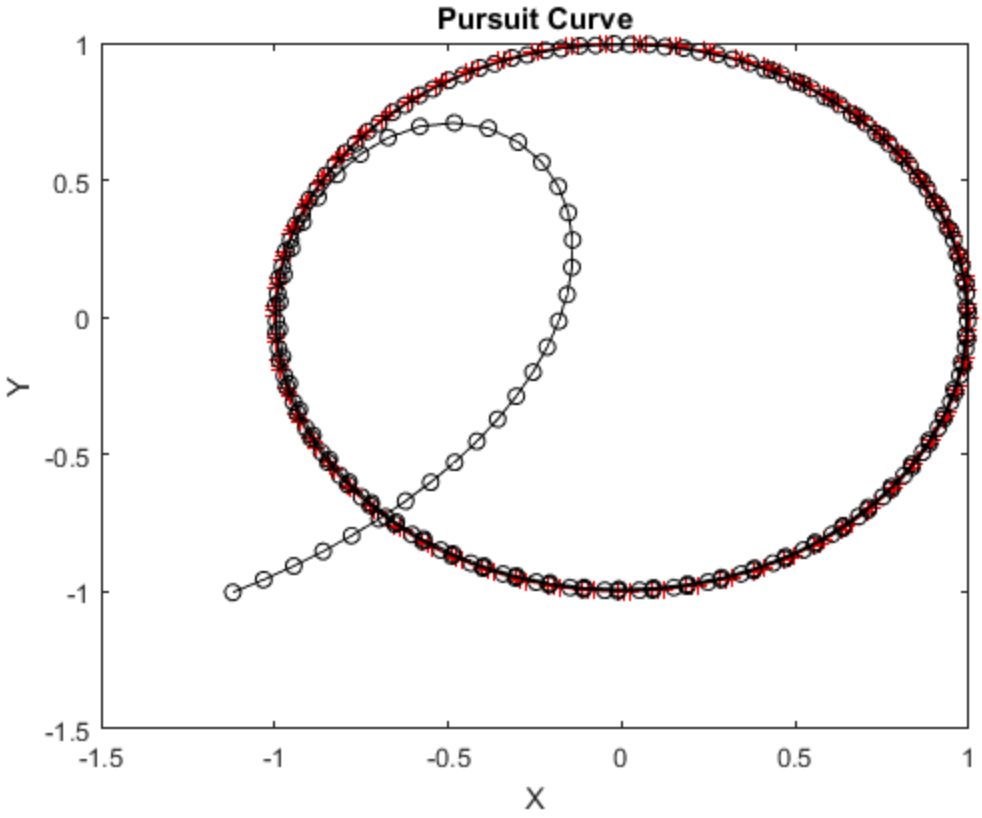
```
The Monte Carlo method gives us -0.53436, while the exact answer
 is -0.53256. The error is 0.00179. This calculation took 0.00295
 seconds.
The Halton set gives us -0.53280, while the exact answer is -0.53256.
 The error is 0.00024. This calculation took 0.01103 seconds.
The trapezoidal method gives us -0.52602, while the exact answer
 is -0.53256. The error is -0.00654. This calculation took 0.04407
 seconds.
prob4 : Success!
```

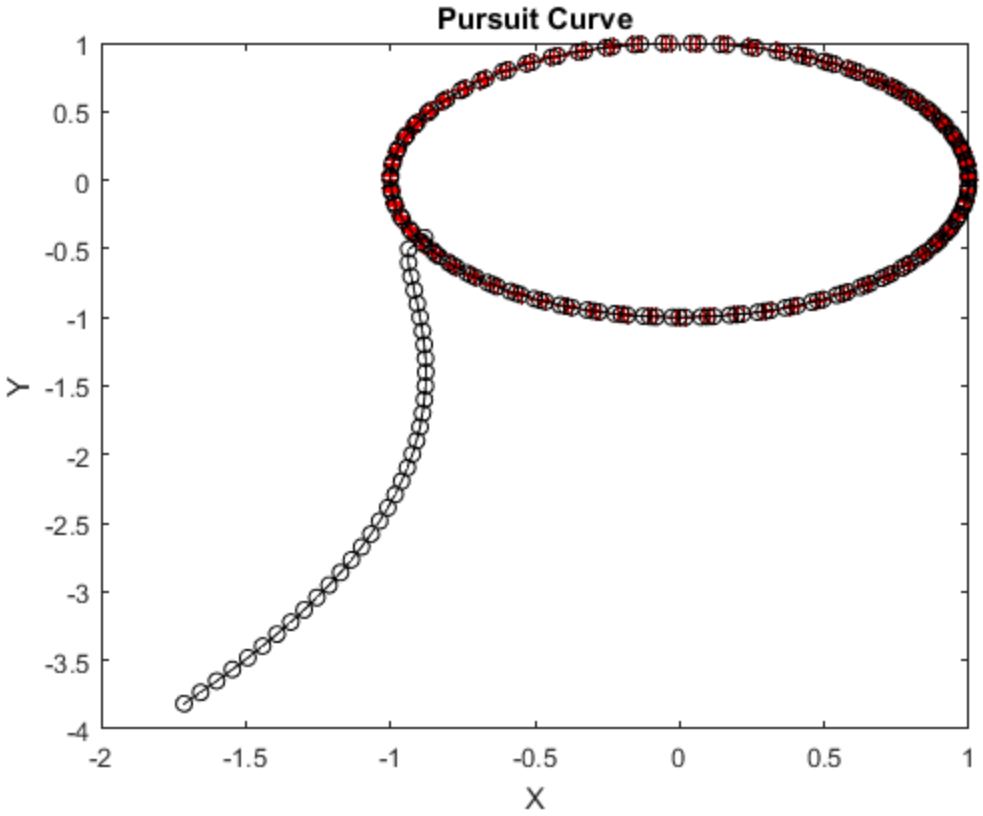
Problem #5 : Pursuit Curves

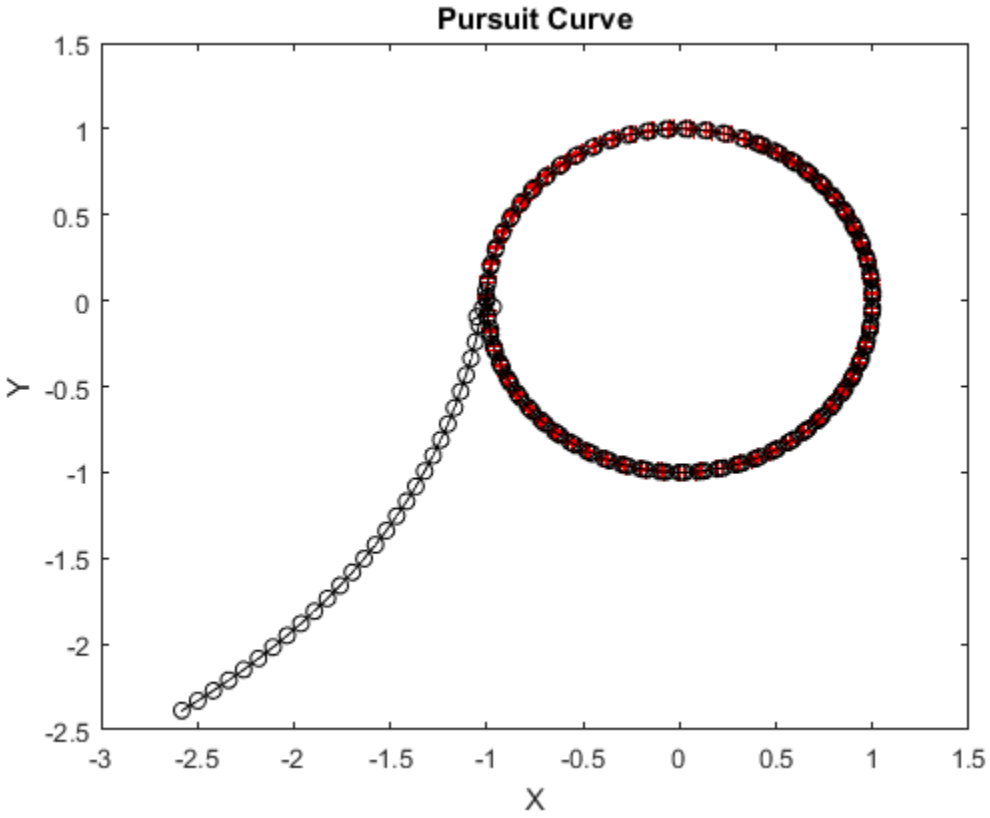
```
function prob5()

% Determine duration and step size.
Tmax = 20;
h = 0.1;

% Loop to generate three plots.
for x = 0:2
 % Initial fox starting location.
 F = [4*rand() - 4,4*rand() - 4];

 % create two lists to hold each set of points.
 Rpoints = zeros(Tmax/h,2);
 Fpoints = zeros(Tmax/h,2);

 % calculate the path in steps of h.
 for t = 0:Tmax/h
  R = [cos(t*h),sin(t*h)];
  dR = [-sin(t*h),cos(t*h)];
  dF = norm(dR,2)*(R-F)/norm(R-F,2);
  F = F + h*dF;
  Rpoints(t+1,:) = R;
  Fpoints(t+1,:) = F;
 end
```

```
 % Plot the results.
 figure()
 plot(Rpoints(:,1),Rpoints(:,2),'r-+',Fpoints(:,1),Fpoints(:,2),'k-o')
    xlabel('X')
    ylabel('Y')
    title('Pursuit Curve')
end

end
```

prob5 : Success!




Pursuit Curve
1
0.5
0
-0.5
-1
-1.5
-2
-2.5
-3
-3.5
-4
Y
-2
-1.5
-1
-0.5
0
0.5
1
X


Pursuit Curve
1.5
1
0.5
0
-0.5
-1
-1.5
-2
-2.5
Y
-3
-2.5
-2
-1.5
-1
-0.5
0
0.5
1
1.5
X

Published with MATLAB® R2016a