Nicholas Bush

Math 50

Homework 4

I will both attach the mat lab files and paste them in here.

Problem 1:

function [ geomean ] = geometricmean( vector\_of\_pos\_nums )

%UNTITLED2 Summary of this function goes here

% Detailed explanation goes here

geomean = nthroot(prod(vector\_of\_pos\_nums),length(vector\_of\_pos\_nums));

end

Problem 2:

function [ one\_norm, two\_norm, inf\_norm ] = three\_norms( a )

%UNTITLED4 Summary of this function goes here

% Detailed explanation goes here

one\_norm = sum(abs(a));

two\_norm = sqrt(sum(a.^2));

inf\_norm = max(abs(a));

end

Problem 3:

function [ A ] = heron\_area( a, b, c )

%UNTITLED5 Summary of this function goes here

% Detailed explanation goes here

s = (a+b+c)/2;

A = sqrt(s\*(s-a)\*(s-b)\*(s-c));

end

Problem 4:

n=10;

A=zeros(n,n);

A(:, 1) = 1;

j=2;

k=2;

while (j<=n)

k=2;

 while (k<=j)

A((j),(k)) =(A((j-1),(k-1)) +A((j-1),(k)));

k=k+1;

 end

j=j+1;

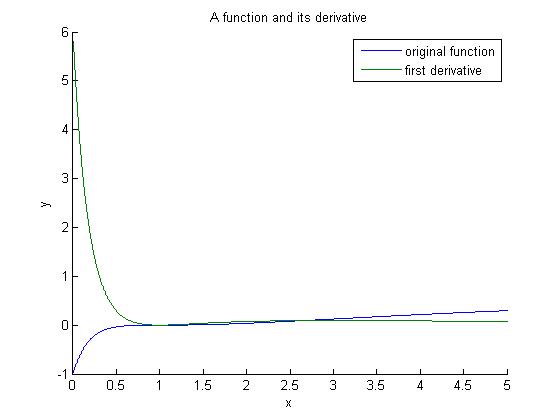
end

B = rem(A,2);

spy(B);

%the spy command plots all non zero elements in a matrix and allows for one

%to examine patterns., as we increase the number n, we start to see more triangles in %the matrix and a lot more open space, the only real thing in the end that maintains the %triangles are the outer edges, everywhere else becomes white space

Problem 5: 

function [ fprime ] = FD\_Left( f, a, b, n )

% Nic

% 2-10-2014

% Math 50

% Lecturer: Derek

%

% INPUTS: function f, interval [a,b], number of grid points n

% OUTPUT: vector of approximate derivative values

%

% This function will use the left-hand finite difference scheme to

% approximate derivatives.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% initialization

h = (b-a)/n;

fprime = zeros(1,n); %derivative array

x = linspace(a,b,n); %makes n points between a and b

% im not sure by the wording of the question but i might have been supposed

% to make this work with arrays that i could plug into functions, however

% they would have had lower accuracy and have to be done pre program...

% this just seemed easier

y = f(x);

%original function this for some reason didnt work

%when i plugged it in down bellow so i just used the formula instead of

%this bellow, the graph is still done using this though

for j = 2:n %calculate all but the first value

fprime(j) = f( x(j) ) - f( x(j-1) ); % fixed the values so it would be

%left hand

end

fprime(1) = f(x(2)) - f(x(1));%as the loop will not define what fprime(1)

%is it must be done outside of the loop

fprime = fprime ./ h;

% display results

hold on;

plot(x,y,x,fprime)

legend('original function','first derivative')

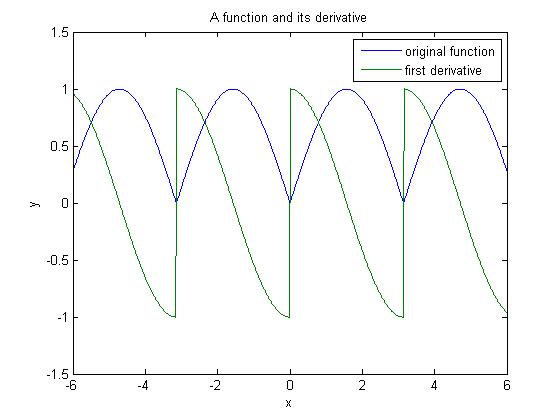
title('A function and its derivative')

xlabel('x')

ylabel('y')

end

Problem6: abs(sin(x)) and deriv



function [ fprime ] = Two\_Step( f, a, b, n )

% Nic

% 2-10-2014

% Math 50

% Lecturer: Derek

%

% INPUTS: function f, interval [a,b], number of grid points n

% OUTPUT: vector of approximate derivative values

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% initialization

h = (b-a)/n; %step size

fprime = zeros(1,n); %derivative array

x = linspace(a,b,n); %makes n points between a and b

y = f(x); %original function

for j = 2:n-1 %calculate for all but last values

fprime(j) = f( x(j+1) ) - f( x(j-1) );

end

% right & left boundry

fprime = fprime ./ (2\*h);

fprime(n) = (f( x(j) ) - f( x(j-1) ))/h;

fprime(1) = (f(x(2)) - f(x(1)))/h;

% display results

plot(x,y,x,fprime)

legend('original function','first derivative')

title('A function and its derivative')

xlabel('x')

ylabel('y')

end

Problem 7:

function [ A ] = hw4\_trap\_comp( func, a, b, n )

%does the trap rule to integrate

% trap rule takes in values like a/ the initial b/ the final, n/ the number of terms, and func the fuction which is attempting to be integrated.

A=0;

x = linspace(a,b,n);

m=1;

dist = (b-a)/n;

while(m<n)

A= A + dist\*((func(x(m))+func(x(m+1)))/2);

m=m+1;

end

end

Answer: 0.1882 which matches what I got when I plugged it into a calculator

Problem 8:

function [ A ] = AdaptiveQuadrature( func, a, b, n, tol )

% This recusive function will either return a Trapezoidal Rule

% approximation, or the sum of two Trapezoidal Rule approximations.

hold on;

% If using twice as many Trapezoids ...

A = hw4\_trap\_comp(func,a,b,n);

B = hw4\_trap\_comp(func,a,b,2\*n);

% ... hardly changes the result, then return the current approximation

if abs(A - B) < tol

% The following section simply draws the Trapezoids

h = (b-a)/n;

x = a:h:b;

y = func(x);

line([x(1) x(1)], [0 y(1)], 'Color', 'k'); %left edge

for i = 2:1:n

line([x(i) x(i)], [0 y(i)], 'Color', 'k'); %vertical edges

line([x(i-1) x(i)], [y(i-1) y(i)], 'Color', 'k'); %diagonal edges

end

pause(1) %for pseudo-animation

else %split at midpoint

m = (a + b) / 2;

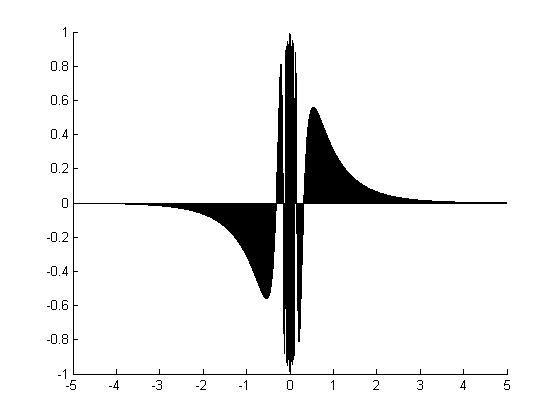
A = AdaptiveQuadrature( func, a, m, n, tol ) + ...

AdaptiveQuadrature( func, m, b, n, tol );

end

end

f(x) = exp(-abs(x)).\*sin(1./x)



Answer: technically zero but according to mat lab 7.7213e-18.