MTH 6060

Vance Turnewitsch

Exam 1 Part 1 Solutions

A. Below we present our code copy and pasted since it was much too long for a screenshot:

function [ MCintegral,MLBintegral ] = MCIntegration( fx )

%MCIntegration Prompts user for a one-dimensional function and bounds upon

%which Monte Carlo integration will be performed.

% MATH 6060 -- Exam 1 Part 1 -- Vance Turnewitsch

% A: Write a Matlab routine to calculate \int a\_b f(x) dx by Monte Carlo

% Method

% First, we get the function to be integrated.

% Now we get the bounds of integration

a = input('Enter the lower bound of integration>');

b = input('Enter the upper bound of integration>');

if a > b || abs(a-b) < 10E-5

disp('Bad bounds!');

exit();

end

accuracy = input('Enter number order of accuracy>');

MCintegral = 0;

% Get a "better" approximation from MatLab for this integral, for

% comparison

MLBintegral = integral(fx,a,b);

% We must find the width and height of the box we will use first.

pts = zeros(1,round((b-a)/1E-3)); % Divide interval into intervals of 1E-3 length

for i=1:round((b-a)/1E-3)

pts(1,i) = fx(a+1E-3\*i);

end

fxMax = max(pts);

pts = zeros(1,round((b-a)/1E-3)); % Divide interval into intervals of 1E-3 length

for i=1:round((b-a)/1E-3)

pts(1,i) = fx(a+1E-3\*i);

end

fxMin = min(pts);

% Now calculate the box width and height in which we generate points.

boxWidth = b-a;

boxHeight = abs(fxMax - fxMin);

boxArea = boxWidth \* boxHeight;

% Ok, time to integrate!

totalIterations = 0;

% These will keep track of the negative and positive integrals.

ptsInPos = 0;

ptsInNeg = 0;

ttlPtsPos = 0;

ttlPtsNeg = 0;

goOn = true;

% How long a batch of monte carlo tests should run before we test for

% convergence.

innerTestCount = 100;

while goOn

if (ttlPtsNeg == 0) ==false && (ttlPtsPos == 0) ==false

oldIntegral = boxArea\*ptsInPos/(ttlPtsPos+ttlPtsNeg) - boxArea\*ptsInNeg/(ttlPtsNeg+ttlPtsPos);

else

oldIntegral = 1E6; % Crazy integral!

end

% We do a certain number of MC tests before comparing the integrals

% for convergence

for i = 1:innerTestCount

% generate test point

testX = a + boxWidth\*rand(); % Generate uniformly distributed point on the interval a to b0

testY = fxMin + boxHeight\*rand();

% Generate point on curve

trueY = fx(testX);

% Now test the point!

if trueY <= 0

% We are looking at a negative area, curve below y=0

ttlPtsNeg = ttlPtsNeg + 1;

% Is point between curve and x axis?

if testY < 0 && testY > trueY

ptsInNeg = ptsInNeg + 1;

end

else

% We are looking at a positive area, curve above y=0

ttlPtsPos = ttlPtsPos + 1;

if testY > 0 && testY < trueY

ptsInPos = ptsInPos + 1;

end

end

end

totalIterations = totalIterations + innerTestCount;

% So, the batch of tests is done, let's see if we converged

newIntegral = boxArea\*ptsInPos/(ttlPtsPos+ttlPtsNeg) - boxArea\*ptsInNeg/(ttlPtsNeg+ttlPtsPos);

%disp( sprintf('The integral is>%d after %d iterations.',newIntegral,totalIterations) );

if abs(newIntegral - oldIntegral) < accuracy

goOn = false;

end

MCintegral = newIntegral;

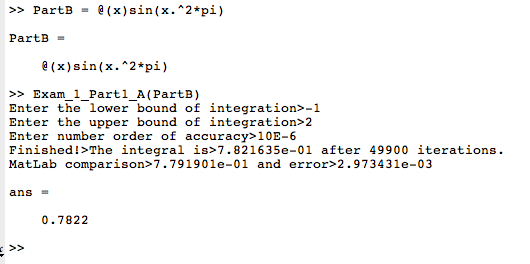
end

disp(sprintf('Finished!>The integral is>%d after %d iterations.',MCintegral,totalIterations));

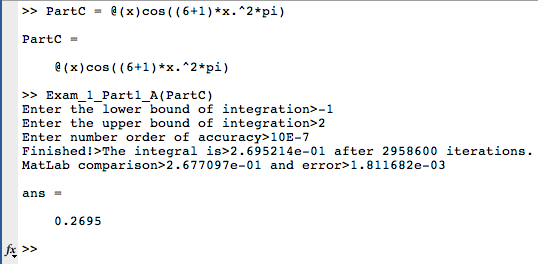
disp(sprintf('MatLab comparison>%d and error>%d',MLBintegral,abs(MLBintegral-MCintegral)));

end

B. Now we run the routine and present our results below:



C. We again run the routine with a different function using u=5



D. We attempt to find order of convergence; but have somewhat difficulty in interpreting the instructions. We use standard convergence of a sequence instead:

function [ MCintegral,MLBintegral ] = MCIntegration( fx )

%MCIntegration Prompts user for a one-dimensional function and bounds upon

%which Monte Carlo integration will be performed.

% MATH 6060 -- Exam 1 Part 1 -- Vance Turnewitsch

% A: Write a Matlab routine to calculate \int a\_b f(x) dx by Monte Carlo

% Method

% First, we get the function to be integrated.

% Now we get the bounds of integration

a = input('Enter the lower bound of integration>');

b = input('Enter the upper bound of integration>');

if a > b || abs(a-b) < 10E-5

disp('Bad bounds!');

exit();

end

accuracy = input('Enter number order of accuracy>');

MCintegral = 0;

% Get a "better" approximation from MatLab for this integral, for

% comparison

MLBintegral = integral(fx,a,b);

% We must find the width and height of the box we will use first.

pts = zeros(1,round((b-a)/1E-3)); % Divide interval into intervals of 1E-3 length

for i=1:round((b-a)/1E-3)

pts(1,i) = fx(a+1E-3\*i);

end

fxMax = max(pts);

pts = zeros(1,round((b-a)/1E-3)); % Divide interval into intervals of 1E-3 length

for i=1:round((b-a)/1E-3)

pts(1,i) = fx(a+1E-3\*i);

end

fxMin = min(pts);

% Now calculate the box width and height in which we generate points.

boxWidth = b-a;

boxHeight = abs(fxMax - fxMin);

boxArea = boxWidth \* boxHeight;

% Ok, time to integrate!

totalIterations = 0;

% These will keep track of the negative and positive integrals.

ptsInPos = 0;

ptsInNeg = 0;

ttlPtsPos = 0;

ttlPtsNeg = 0;

goOn = true;

% How long a batch of monte carlo tests should run before we test for

% convergence.

innerTestCount = 100;

% We continue to do MC until we achieve desired accuracy!

while abs(MCintegral-MLBintegral) > accuracy

if (ttlPtsNeg == 0) ==false && (ttlPtsPos == 0) ==false

oldIntegral = boxArea\*ptsInPos/(ttlPtsPos+ttlPtsNeg) - boxArea\*ptsInNeg/(ttlPtsNeg+ttlPtsPos);

else

oldIntegral = 1E6; % Crazy integral!

end

% We do a certain number of MC tests before comparing the integrals

% for convergence

for i = 1:innerTestCount

% generate test point

testX = a + boxWidth\*rand(); % Generate uniformly distributed point on the interval a to b0

testY = fxMin + boxHeight\*rand();

% Generate point on curve

trueY = fx(testX);

% Now test the point!

if trueY <= 0

% We are looking at a negative area, curve below y=0

ttlPtsNeg = ttlPtsNeg + 1;

% Is point between curve and x axis?

if testY < 0 && testY > trueY

ptsInNeg = ptsInNeg + 1;

end

else

% We are looking at a positive area, curve above y=0

ttlPtsPos = ttlPtsPos + 1;

if testY > 0 && testY < trueY

ptsInPos = ptsInPos + 1;

end

end

end

totalIterations = totalIterations + innerTestCount;

% So, the batch of tests is done, let's see if we converged

newIntegral = boxArea\*ptsInPos/(ttlPtsPos+ttlPtsNeg) - boxArea\*ptsInNeg/(ttlPtsNeg+ttlPtsPos);

MCintegral = newIntegral;

end

disp(sprintf('Finished!>The integral is>%d after %d iterations.',MCintegral,totalIterations));

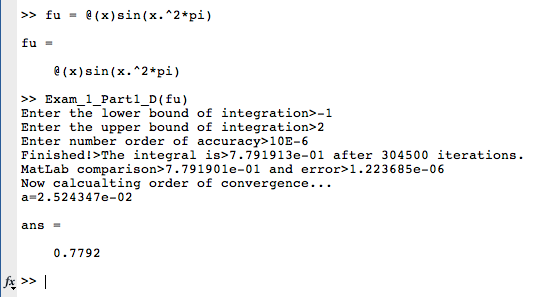
disp(sprintf('MatLab comparison>%d and error>%d',MLBintegral,abs(MLBintegral-MCintegral)));

disp('Now calcualting order of convergence...');

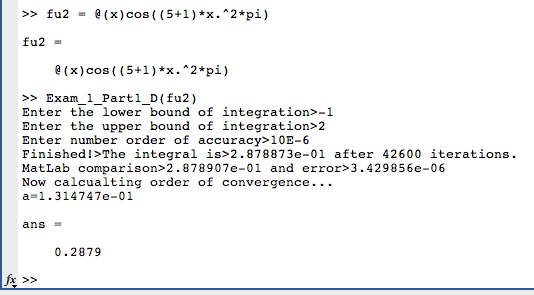
disp(sprintf('a=%d',log(MCintegral)/log(MCintegral-oldIntegral)));

end

We now display results of running the script:



E. We present the results for determining convergence using integrand from part c.



Unfortunately, the order of convergence is not very good. The order of convergence for Monte Carlo scheme is supposed to be approximately ½. But this is quite far from our convergence.