//AUTHOR: DANIEL CRISP

//DATE: 11/15/2016

//DESC: TRAPEZOID

QUESTION 1: CHAPTER 7 -- #8

MATLAB CODE:

function [ data ] = rolldice( printresults, sides, dice, N )

% rolldice simulates 'N' rolls of two fair dice. print results=true

% will display textual output per roll, false will display results in

% bar chart along with probabilities based on those rolls.

% make printresults into something easier to use

switch printresults

case 'true'

printIt = 1;

case 'false'

printIt = 0;

end

% Create randomized data for each roll of each dice.

rng('shuffle'); %seed random number generator with time

data = rand(N, dice);

for i=1:N

for j=1:dice

data(i,j) = ceil(sides\*data(i,j));

if j == dice

% Put sum of dice values into a new column at the given roll

% index.

data(i,dice+1) = sum(data(i,:));

end

end

end

% Create array num for holding number of times each possible side-total was rolled

roll\_sums(dice\*1:dice\*sides) = [0];

% Add each time a roll-total was rolled to the correct element of 'num'.

for i=1:N

rollSum = data(i,dice+1);

roll\_sums(rollSum) = roll\_sums(rollSum) + 1;

if printIt==1 display\_line( data(i,:)); end

end

if printIt==0 bar\_prob(roll\_sums, N); end

fprintf('\n\n');

end

function display\_line( data )

%display\_line takes an array. 2 nums 1:6, and their sum. and displays a

%line of output.

sum = data(3);

str = string(sum);

if sum==2

str = 'snake eyes';

elseif sum==3

str = 'ace-deuce';

elseif sum==11

str = 'yo';

elseif sum==12

str = 'boxcars';

elseif sum==7

str = 'natural';

elseif data(1)==data(2)

if data(1)==2

str = 'hard four';

elseif data(1)==3

str = 'hard six';

end

end

fprintf('\nPlayer rolls a %i and a %i : %s',data(1),data(2),str);

end

function bar\_prob(roll\_sums, N)

% Calculate probability from simulation

prob(:) = 100\*roll\_sums(:)/N;

% Display bar graphs of roll total results, and probabilities.

subplot(2,1,1); bar(roll\_sums);

title('Dice Simulation'); xlabel('Roll Total'); ylabel('# Times Rolled');

grid on

subplot(2,1,2); bar(prob);

xlabel('Roll Total'); ylabel('% Probability');

grid on

end

MATLAB OUTPUT:

>>rolldice('true',6,2,10);

Player rolls a 2 and a 6 : 8

Player rolls a 6 and a 2 : 8

Player rolls a 6 and a 3 : 9

Player rolls a 2 and a 4 : 6

Player rolls a 1 and a 4 : 5

Player rolls a 1 and a 1 : snake eyes

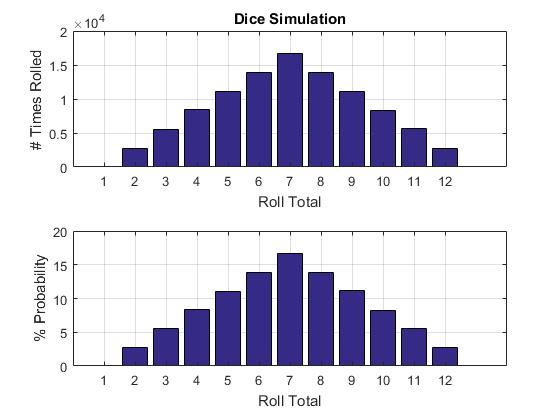
Player rolls a 1 and a 5 : 6

Player rolls a 2 and a 5 : natural

Player rolls a 3 and a 4 : natural

Player rolls a 5 and a 3 : 8

>>rolldice('false',6,2,1\*10^5);



QUESTION 2: CHAPTER 7 -- #13

MATLAB CODE:

function vout=clipVec( v , vmin , vmax )

%clipVec copies vector v to vout, but 'clips' values between

%vmin and vmax.

for i=1:length(v)

if v(i)>vmax

vout(i)=vmax;

elseif v(i)<vmin

vout(i)=vmin;

else

vout(i)=v(i);

end

end

end

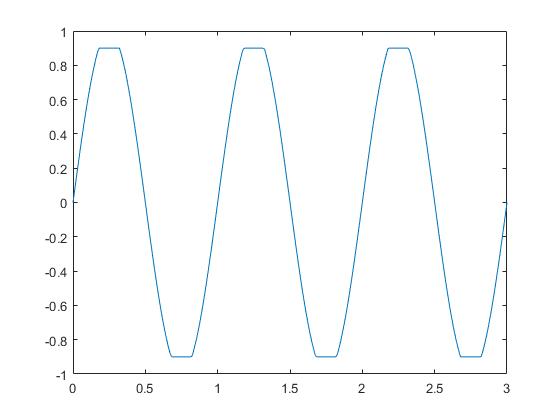
MATLAB OUTPUT:

>>x=linspace(0, 3, 300);

y=sin(2\*pi\*x);

plot(x,clipVec(y, -0.9, 0.9));

axis([0, 3, -1, 1]);



QUESTION 3a: TrapizoidRule

MATLAB CODE:

TrapizoidRule.m

function [vargout] = TrapizoidRule(X,fun,n)

%trap takes function handle fun, and n steps over X to estimate the

%integrand of fun over all X using the trapezoid metod.

Xlen = length(X);

range = X(Xlen)-X(1);

step = range/n;

% Create a corresponding Y space to match with X

Y = fun(X);

%Initialize variables for first pass through loop.

raw = 0;

a = X(1);

fa = fun(a);

for i=1:n

b = a + step;

fb = fun(b);

raw = raw + 0.5\*(b-a)\*(fb+fa);

if i<n

a = b;

fa = fb;

end

end

% Use trapz(X,Y) as reference to 'raw' estimate

estimated = trapz(X,Y);

error = abs((raw-estimated)/estimated)\*100;

% Aggregate values for 'raw', 'estimated', and 'error' into vargout.

vargout=[raw, estimated, error];

fprintf('\nThe raw calculation found the integral to be %f.\nThe trapz() estimate was %f.\nAnd the percent error between the two is %f.\n\n',vargout(:));

% Plot function

figure; plot(X,Y); title('Y = f(x)'),xlabel('X'); ylabel('Y');

end

MATLAB OUTPUT: (NOTE: output is my raw calculation, the trapz() output, and the %error between the two.)

>>X = -2:6/100:4; fun = @(x) 1-x-4\*x.^3 + 2\*x.^5;

>> n=1; [result1] = TrapizoidRule(X,fun,n);

The raw calculation found the integral to be 5280.000000.

The trapz() estimate was 1104.676774.

And the percent error between the two is 377.967866.

>> n=2; [result2] = TrapizoidRule(X,fun,n);

The raw calculation found the integral to be 2634.000000.

The trapz() estimate was 1104.676774.

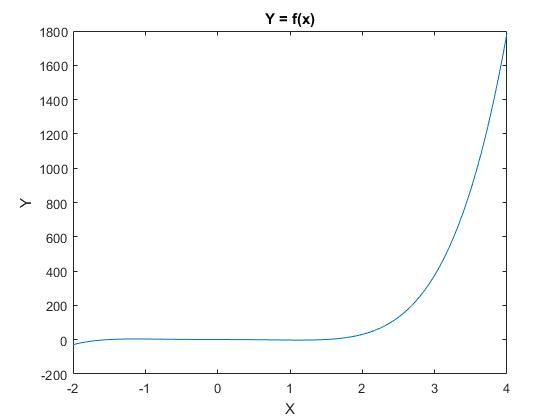
And the percent error between the two is 138.440788.

>> n=4; [result4] = TrapizoidRule(X,fun,n);

The raw calculation found the integral to be 1516.875000.

The trapz() estimate was 1104.676774.

And the percent error between the two is 37.313922.



QUESTION 3b: SimpsonRule

MATLAB CODE:

SimpsonRule.m

Write a function called SimpsonRule (3/8 rule) and implement the integration for the same integral as in 3a.

function [vargout] = SimpsonRule( plotfun, method , X , fun , n )

%SimpsonRule takes function handle fun, and n steps over X to estimate the

%integrand of fun over all X using either 1/3 or 3/8 simpson methods.

%Specify which--1/3 or 3/8--simpson method to use by stating either 0 or 1.

%When plotfun=0, the function will not be plotted.

if mod(n,2)

isodd = 1; %isodd is 1, or true, when n is odd.

else

isodd = 0;

end

if method==0 && isodd %method==0 for requesting 1/3 rule.

fprintf('\n"Vargin" n must be even if we are to have an odd number of points for Simpsons 1/3 Rule.');

fprintf('\nYet, %i is odd. Switching the method argument to 1 and continuing with the 3/8 Rule.\n\n',n);

method=1;

end

if method==1 && n<3

fprintf('\n"vargin" n must be 3 or greater to use Simpsons 3/8 rule.');

fprintf('\nYet, %i is not. Breaking from function "SimpsonRule".',n);

fprintf('\n\nIf n=2, try using method=0 for Simpsons 1/3, or make n equal 3 or greater\n\n');

vargout(1:5) = 0;

return;

end

% Determine step size using range of 'X' and number of steps 'n'

a = X(1);

b = X(length(X));

h = (b-a)/n;

% Initialize variables for first pass through loop.

f\_xa = fun(a);

f\_xb = fun(b);

if method==0

A=h\*(1/3);

else

A=h\*(3/8);

end

estimated = A\*(f\_xa+f\_xb); %Both first, and last points

for i=1:n-1

xi = a + (h\*i);

switch method

case 0 % Simpson's 1/3... 1-4-2-4-2-4-1

if mod(i,2)==0

% IF i is even

newval = A\*2\*fun(xi);

else

% IF i is odd

newval = A\*4\*fun(xi);

end

case 1 % Simpson's 3/8... 1-3-3-2-3-3-2-3-3-1 (or ...2-4-1 when n is odd)

if mod(i,3)==0

% IF i is divisible by 3

newval = A\*2\*fun(xi);

elseif isodd && i==n-2

% IF n is odd, and i equals n-2

newval = A\*4\*fun(xi);

else

% OTHERWISE

newval = A\*3\*fun(xi);

end

end

estimated = estimated + newval;

end

% Plot function

if plotfun == 1

figure; ezplot(fun); title('Y = f(x)'),xlabel('X'); ylabel('Y');

end

% Use trapz(X,Y) as reference to 'real' estimate.

real\_trapz = trapz(X,fun(X));

error\_trapz = ((real\_trapz-estimated)/real\_trapz)\*100;

% Using simps.m from Garcia, Damien to get more accurate estimation of

% integral, at least more-so than from trapz.

% Even though it is an extension of matlab's trapz(). Code was found at...

% https://www.mathworks.com/matlabcentral/fileexchange/25754-simpson-s-rule-for-numerical-integration/content/simps.m

if exist('simps.m','file')

real\_simps = simps(X,fun(X));

error\_simps = ((real\_simps-estimated)/real\_simps)\*100;

else

fprintf('\nsimps.m not found. real\_simps and error\_simps set to 0.\n');

real\_simps = 0;

error\_simps = 0;

end

% Calculate error bounds using err function. Change what point you use

% to evaluate the error by using evalAt. Had defaulted to a+h..

%error\_limit = errorlim(a,b,a+h,X,fun,n);

% Aggregate values for 'raw', 'estimated', and 'error' into vargout.

vargout=[estimated, real\_trapz, error\_trapz, real\_simps, error\_simps];

% Display

display( vargout, method);

end

function display( vargout , method )

if method==0

fprintf('\nUsing Simpsons 1/3, the integral was estimated to be %f.',vargout(1));

else

fprintf('\nUsing Simpsons 3/8, the integral was estimated to be %f.',vargout(1));

end

fprintf('\nThe function trapz() found the real value of the integral to be %f;\n\t...implying a %f%% error.',vargout(2:3));

fprintf('\nAnd function simps(), a function from Daimen Garcia on mathworks file exchange, finds the real value of the integral to be %f;\n\t...implying a %f%% error.\n\n',vargout(4:5));

end

function [error\_limit] = errorlim(a,b,evalAt,X,fun,n)

syms X;

error\_function = [-1/(180\*n^4)]\*[(b-a)^5]\*diff(fun(X),X,4);

error\_limit = vpa(subs(error\_function,X,evalAt));

end

>>X = -2:6/100:4; fun = @(x) 1-x-4\*x.^3 + 2\*x.^5;

>>results=SimpsonRule(0,1,X,fun,1);

Cannot compute simpson 1/3 or 3/8 with n<2

>>results=SimpsonRule(0,1,X,fun,2);

Cannot compute simpson 3/8 with n<3, calculating for 1/3

Using Simpsons 1/3, the integral was estimated to be 1752.000000.

The function trapz() found the real value of the integral to be 1104.676774;

...implying a -58.598428% error.

And function simps(), a function from Daimen Garcia on mathworks file exchange, finds the real value of the integral to be 1104.000104;

...implying a -58.695637% error.

>>results=SimpsonRule(0,1,X,fun,4);

Using Simpsons 3/8, the integral was estimated to be 1137.621094.

The function trapz() found the real value of the integral to be 1104.676774;

...implying a -2.982259% error.

And function simps(), a function from Daimen Garcia on mathworks file exchange, finds the real value of the integral to be 1104.000104;

...implying a -3.045379% error.

#QUESTION 4

MATLAB CODE: (didn't use function as before, could not adapt it in time to work for both arrays and function handles) h was not re-calculated. I think the lack of dynamic h in my script provides the largest error.

SimpsonScript.m

% Clean workspace and console

clear; clc;

% Re-set scriptname for re-run.

script = 'SimpsonScript.m';

% Instantiate variables

t=[1,2,3.25,4.5,6,7,8,8.5,9.3,10];

v=[5,6,5.5,7,8.5,8,6,7,7,5];

X = t; Xlen = length(X);

Y = v;

n = Xlen-1; %Using number of steps equal to length of array - 1.

if mod(n,2)

isodd = 1; %isodd is 1, or true, when n is odd.

else

isodd = 0;

end

% Determine step size using range of 'X' and number of steps 'n'

a = X(1);

b = X(Xlen);

h = (b-a)/n;

% Initialize variables for first pass through loop.

A=h\*(3/8);

f\_xa = Y(1);

vals(1) = A\*f\_xa;

f\_xb = Y(Xlen);

vals(Xlen) = A\*f\_xb;

estimated = vals(1) + vals(Xlen); %Both first, and last points

for i=2:Xlen-1

% Simpson's 3/8... 1-3-3-2-3-3-2-3-3-1 (or ...2-4-1 when n is odd)

if mod(i,3)==0

% IF i is divisible by 3

vals(i) = A\*2\*Y(i);

elseif isodd && i==n-2

% IF n is odd, and i equals n-2

vals(i) = A\*4\*Y(i);

else

% OTHERWISE

vals(i) = A\*3\*Y(i);

end

estimated = estimated + vals(i);

end

% Use trapz(X,Y) as reference to 'real' estimate

real\_trapz = trapz(X,Y);

error\_trapz = ((real\_trapz-estimated)/real\_trapz)\*100;

% Using simps.m from Garcia, Damien to get more accurate estimation of

% integral, at least more-so than from trapz.

real\_simps = simps(X,Y);

error\_simps = ((real\_simps-estimated)/real\_simps)\*100;

vargout=[estimated, real\_trapz, error\_trapz, real\_simps, error\_simps];

% Plot function

figure; plot(X,Y); title('Y = f(x)'),xlabel('X'); ylabel('Y');

fprintf('\nUsing Simpsons 3/8, the integral was estimated to be %f.',vargout(1));

fprintf('\nThe function trapz() found the real value of the integral to be %f;\n\t...implying a %f%% error.',vargout(2:3));

fprintf('\nAnd function simps(), a function from Daimen Garcia on mathworks file exchange, finds the real value of the integral to be %f;\n\t...implying a %f%% error.\n\n',vargout(4:5));

MATLAB OUTPUT:

Using Simpsons 3/8, the integral was estimated to be 60.187500.

The function trapz() found the real value of the integral to be 60.425000;

...implying a 0.393049% error.

And function simps(), a function from Daimen Garcia on mathworks file exchange, finds the real value of the integral to be 61.318056;

...implying a 1.843756% error.

MEANING:

A distance of about 60.5 was traveled (units not stated in question).