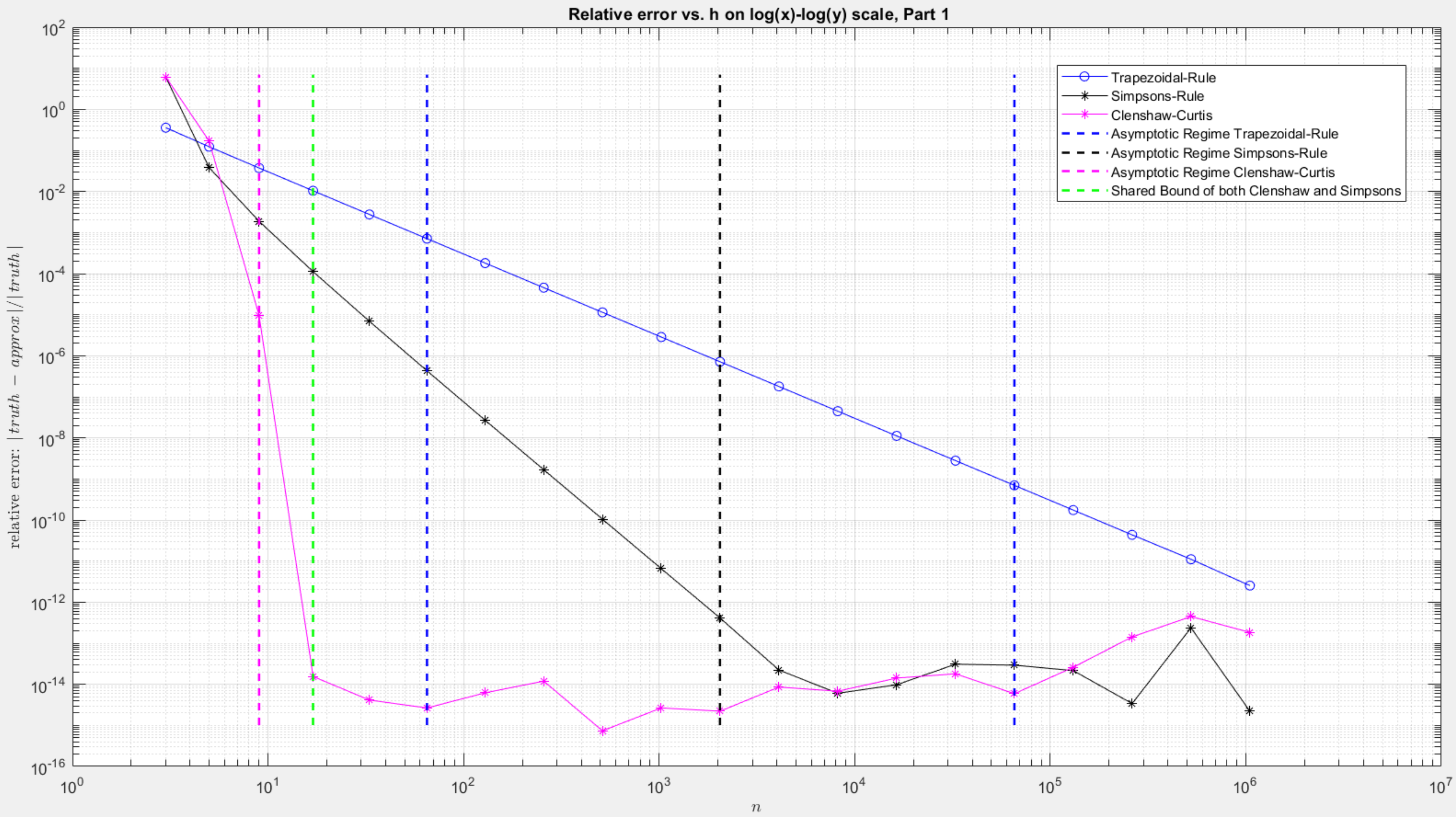
Part 1:

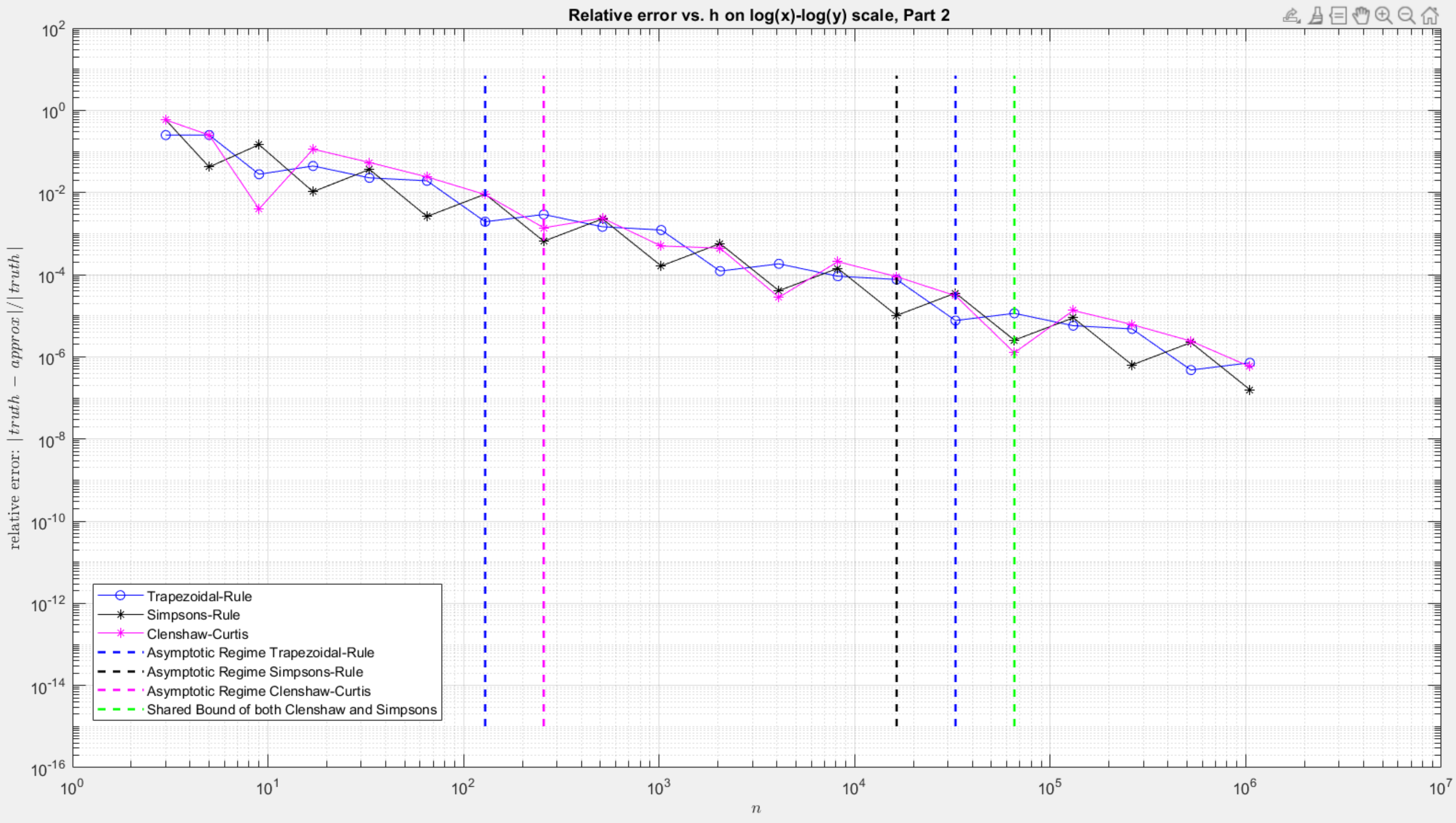
Method	Observed Convergence Rate
Trapezoidal-Rule	-2.00004
Simpsons-Rule	-4.05278
Clenshaw-Curtis	-31.83494

Part 2:

Method	Observed Convergence Rate
Trapezoidal-Rule	-1.00000
Simpsons-Rule	-1.00092
Clenshaw-Curtis	-1.26380

Using a numerical study similar to Problem one, the convergence rates for the second function remain similar for all methods around a value of -1. For clenshaw the error is bounded O(n^-1) for not smooth functions, in addition all methods struggle to accuratetly fit to the jump at 0.2 causing errors in all methods.>>


Relative error vs. h on log(x)-log(y) scale, Part 1
Trapezoidal-Rule
Simpsons-Rule
Clenshaw-Curtis
Asymptotic Regime Trapezoidal-Rule
Asymptotic Regime Simpsons-Rule
Asymptotic Regime Clenshaw-Curtis
Shared Bound of both Clenshaw and Simpsons
relative error: |truth − approx|/|truth|
n



Relative error vs. h on log(x)-log(y) scale, Part 2
relative error: |truth − approx|/|truth|
n
Trapezoidal-Rule
Simpsons-Rule
Clenshaw-Curtis
Asymptotic Regime Trapezoidal-Rule
Asymptotic Regime Simpsons-Rule
Asymptotic Regime Clenshaw-Curtis
Shared Bound of both Clenshaw and Simpsons

Table of Contents

Housekeeping

```
clear all; close all; clc;

%{
    CSCI 3656 HW11
    Author: Connor O'Reilly
    Email: coor1752@colorado.edu
    Last Edited: 12/6/21
 %}
```

Part 1

```
%Using calculus the computed definite integral is shown above in the
 provided pictures

%given function
f = @(x) sin(2*x) + cos(3*x);
int = [-1,1]; %interval of x

%computed definite integral value using calculus
def_int_calc = 0.094080005373245;

%initialization
k = 1:20;
n = 2.^k + 1;
est_trapz = zeros(20,1);
est_simp = est_trapz;
est_clen = est_trapz;
%estimations
for i = 1:20
    est_trapz(i) = trap_rule(f , int(1) , int(2) , n(i) );
    est_simp(i) = simp_rule(f , int(1) , int(2) , n(i) );
    [ x_curr , w_curr ] = clenshaw_curtis_rule( n(i), int(1) ,
 int(2) );
    est_clen(i) = clenshaw_solve( f , x_curr , w_curr );
end

%get rid of variables
clear x_curr w_curr
```

```
%compute the relative error against calculus
rel_trapz = abs(est_trapz - def_int_calc) / abs( def_int_calc );
rel_simp = abs(est_simp - def_int_calc) / abs( def_int_calc );
rel_clen = abs(est_clen - def_int_calc) / abs( def_int_calc );

%determine the asyptotic regime for all relative errors

%initialize log of arrays
log_n = log(n);
len_n = length(n);
log_trapz = log(rel_trapz);
log_simp = log(rel_simp);
log_clen = log(rel_clen);

%find change points
%used
%https://www.mathworks.com/help/signal/ref/findchangepts.html#bu3nws1-
ipt
%hopefully it works
ptst = findchangepts(log_trapz, 'MinThreshold', 2*len_n);
ptss = findchangepts(log_simp, 'MinThreshold', 2*len_n);
ptsc = findchangepts(log_clen, 'MinThreshold', 2*len_n);

%define regime for three methods
rnget = [ptst(1) , ptst(end)];
rnges = [ptss(1) , ptss(end)];
rngec = [ptsc(1) , ptsc(end)];


%find convergence rates

conv_t = ( log_trapz(rnget(2)) - log_trapz(rnget(1)) ) /
 ( log_n(rnget(2)) - log_n(rnget(1)) );
conv_s = ( log_simp(rnges(2)) - log_simp(rnges(1)) ) /
 ( log_n(rnges(2)) - log_n(rnges(1)) );
conv_c = ( log_clen(rngec(2)) - log_clen(rngec(1)) ) /
 ( log_n(rngec(2)) - log_n(rngec(1)) );
```

Part 2

```
%given function in functions section

%compute integral using calculus or area undercurve. im too lazy to do
 it
%on paper so hopefully this is right. f(x) = sign(x-0.2) + 1
truth_int = 1.6;
f_2 = @(x) sign(x-0.2) + 1;

int = [-1,1]; %interval of x

%initialization
est_trapz_p2 = zeros(20,1);
est_simp_p2 = est_trapz_p2;
```

```
est_clen_p2 = est_trapz_p2;

%estimations
for i = 1:20
    est_trapz_p2(i) = trap_rule(f_2 , int(1) , int(2) , n(i) );
    est_simp_p2(i) = simp_rule(f_2 , int(1) , int(2) , n(i) );
    [ x_curr , w_curr ] = clenshaw_curtis_rule( n(i), int(1) ,
 int(2) );
    est_clen_p2(i) = clenshaw_solve( f_2 , x_curr , w_curr );
end
%get rid of variables
clear x_curr w_curr

%compute the relative error against calculus
rel_trapz_p2 = abs(est_trapz_p2 - truth_int) / abs( truth_int );
rel_simp_p2 = abs(est_simp_p2 - truth_int) / abs( truth_int );
rel_clen_p2 = abs(est_clen_p2 - truth_int) / abs( truth_int );

%determine the asyptotic regime for all relative errors

%initialize log of arrays
log_trapz_p2 = log(rel_trapz_p2);
log_simp_p2 = log(rel_simp_p2);
log_clen_p2 = log(rel_clen_p2);

%find change points
%used
%https://www.mathworks.com/help/signal/ref/findchangepts.html#bu3nws1-
ipt
%hopefully it works
ptst_p2 = findchangepts(log_trapz_p2, 'MinThreshold', 2*len_n);
ptss_p2 = findchangepts(log_simp_p2, 'MinThreshold', 2*len_n);
ptsc_p2 = findchangepts(log_clen_p2, 'MinThreshold', 2*len_n);

%define regime for three methods
rnget_p2 = [ptst_p2(1) , ptst_p2(end)];
rnges_p2 = [ptss_p2(1) , ptss_p2(end)];
rngec_p2 = [ptsc_p2(1) , ptsc_p2(end)];


%find convergence rates

conv_t_p2 = ( log_trapz_p2(rnget_p2(2)) -
 log_trapz_p2(rnget_p2(1)) ) / ( log_n(rnget_p2(2)) -
 log_n(rnget_p2(1)) );
conv_s_p2 = ( log_simp_p2(rnges_p2(2)) - log_simp_p2(rnges_p2(1)) ) /
 ( log_n(rnges_p2(2)) - log_n(rnges_p2(1)) );
conv_c_p2 = ( log_clen_p2(rngec_p2(2)) - log_clen_p2(rngec_p2(1)) ) /
 ( log_n(rngec_p2(2)) - log_n(rngec_p2(1)) );
```

Display

```
%part 1
```

```
fprintf('\n
\n-------------------------------------------------------------------
\n')
fprintf('Part 1:')
fprintf('\n-------------------------------------------------------------------
\n\n')
fprintf('\n-------------------------------------------------------------------
\n')
fprintf('           Method             |    Observed Convergence
 Rate');
fprintf('      |
\n-------------------------------------------------------------------
\n')
fprintf(' Trapezoidal-Rule         |                %0.5f
          |', conv_t)
fprintf('\n-------------------------------------------------------------------
\n')
fprintf(' Simpsons-Rule            |                %0.5f
          |', conv_s)
fprintf('\n-------------------------------------------------------------------
\n')
fprintf(' Clenshaw-Curtis          |                %0.5f
        |', conv_c)
fprintf('\n-------------------------------------------------------------------
\n')


%part 2
fprintf('\n
\n-------------------------------------------------------------------
\n')
fprintf('Part 2:')
fprintf('\n-------------------------------------------------------------------
\n\n')
fprintf('\n-------------------------------------------------------------------
\n')
fprintf('           Method             |    Observed Convergence
 Rate');
fprintf('      |
\n-------------------------------------------------------------------
\n')
fprintf(' Trapezoidal-Rule         |                %0.5f
          |', conv_t_p2)
fprintf('\n-------------------------------------------------------------------
\n')
fprintf(' Simpsons-Rule            |                %0.5f
          |', conv_s_p2)
fprintf('\n-------------------------------------------------------------------
\n')
fprintf(' Clenshaw-Curtis          |                %0.5f
        |', conv_c_p2)
fprintf('\n-------------------------------------------------------------------
\n')
```

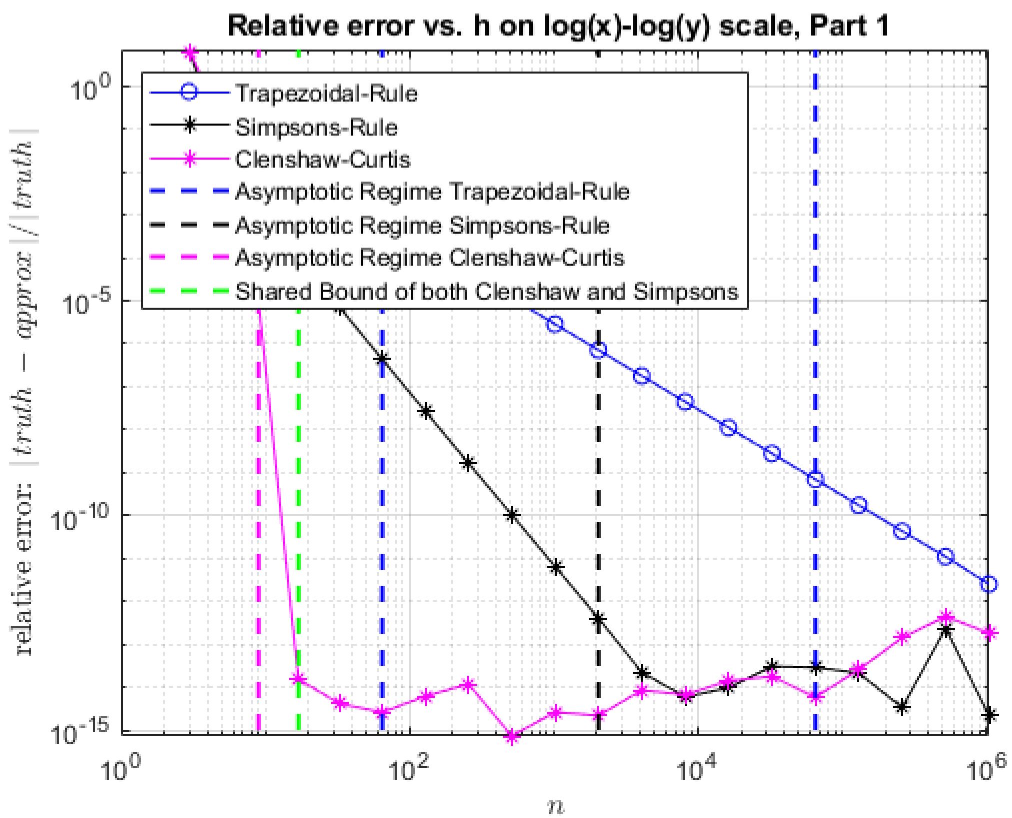
```
fprintf('Using a numerical study similar to Problem one, the
 convergence rates for the second function remain \nsimilar for all
 methods around a value of -1. For clenshaw the error is bounded
 O(n^-1) for not smooth functions, \n in addition all methods
 struggle to accurately fit to the jump at 0.2 causing errors in all
 methods.')
```

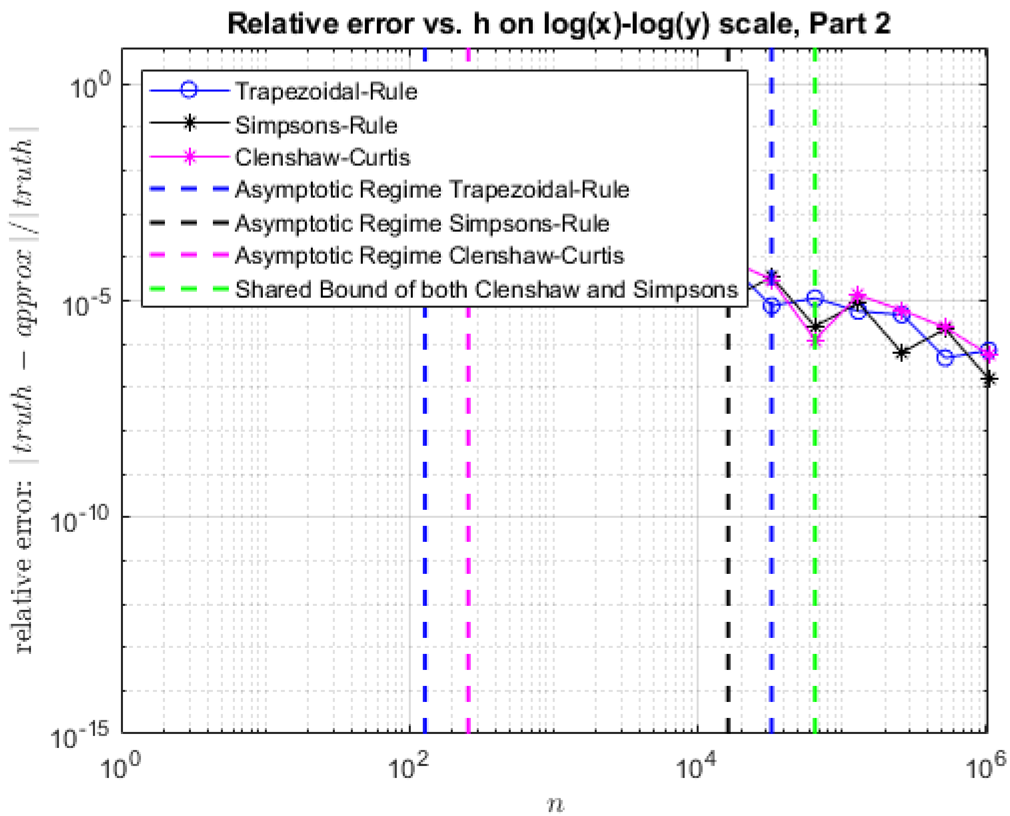
```
------------------------------------------------------------------
Part 1:
------------------------------------------------------------------


------------------------------------------------------------------
           Method                  |    Observed Convergence Rate      |
------------------------------------------------------------------
 Trapezoidal-Rule              |                 -2.00004
      |
------------------------------------------------------------------
 Simpsons-Rule                 |                 -4.05278
      |
------------------------------------------------------------------
 Clenshaw-Curtis               |                 -31.83494
     |
------------------------------------------------------------------


------------------------------------------------------------------
Part 2:
------------------------------------------------------------------


------------------------------------------------------------------
           Method                  |    Observed Convergence Rate      |
------------------------------------------------------------------
 Trapezoidal-Rule              |                 -1.00000
      |
------------------------------------------------------------------
 Simpsons-Rule                 |                 -1.00092
      |
------------------------------------------------------------------
 Clenshaw-Curtis               |                 -1.26380
    |
------------------------------------------------------------------
Using a numerical study similar to Problem one, the convergence rates
 for the second function remain
similar for all methods around a value of -1. For clenshaw the error
 is bounded O(n^-1) for not smooth functions,
 in addition all methods struggle to accurately fit to the jump at
 0.2 causing errors in all methods.
```

Plotting

```
%part1
% elative error as a function of n on a log-log scale
figure(1)
loglog(n , rel_trapz , 'bo-')
hold on
loglog(n , rel_simp , 'k*-')
loglog(n , rel_clen , 'm*-')

%add vertical lines for asymptotic range
%for trap error
loglog([n(rnget(1)) n(rnget(1))], [10^-15 7], '--b', 'Linewidth',1.5)
loglog([n(rnget(2)) n(rnget(2))], [10^-15 7],'--b', 'Linewidth',1.5)

%for simp error
loglog([n(rnges(1)) n(rnges(1))], [10^-15 7], '--k', 'Linewidth',1.5)
loglog([n(rnges(2)) n(rnges(2))], [10^-15 7],'--k', 'Linewidth',1.5)

%for simp error
loglog([n(rngec(1)) n(rngec(1))], [10^-15 7], '--m', 'Linewidth',1.5)
loglog([n(rngec(2)) n(rngec(2))], [10^-15 7],'--g', 'Linewidth',1.5)
grid on;
legend('Trapezoidal-Rule' , 'Simpsons-Rule' , 'Clenshaw-
Curtis','Asymptotic Regime Trapezoidal-Rule', '', 'Asymptotic Regime
 Simpsons-Rule','','Asymptotic Regime Clenshaw-Curtis','Shared Bound
 of both Clenshaw and Simpsons', 'Location', 'northwest' )
title(' Relative error vs. h on log(x)-log(y) scale, Part 1')
ylabel('relative error: $|\,truth\,-\,approx\,|/|\,truth\,|
$', 'interpreter', 'latex');
xlabel('$n$', 'interpreter', 'latex');
hold off;

%part2
% elative error as a function of n on a log-log scale
figure(2)
loglog(n , rel_trapz_p2 , 'bo-')
hold on
loglog(n , rel_simp_p2 , 'k*-')
loglog(n , rel_clen_p2 , 'm*-')

%add vertical lines for asymptotic range
%for trap error
loglog([n(rnget_p2(1)) n(rnget_p2(1))], [10^-15 7], '--
b', 'Linewidth',1.5)
loglog([n(rnget_p2(2)) n(rnget_p2(2))], [10^-15 7],'--
b', 'Linewidth',1.5)

%for simp error
loglog([n(rnges_p2(1)) n(rnges_p2(1))], [10^-15 7], '--
k', 'Linewidth',1.5)
loglog([n(rnges_p2(2)) n(rnges_p2(2))], [10^-15 7],'--
k', 'Linewidth',1.5)
```

```
%for simp error
loglog([n(rngec_p2(1)) n(rngec_p2(1))], [10^-15 7], '--
m', 'Linewidth',1.5)
loglog([n(rngec_p2(2)) n(rngec_p2(2))], [10^-15 7],'--
g', 'Linewidth',1.5)
grid on;
legend('Trapezoidal-Rule' , 'Simpsons-Rule' , 'Clenshaw-
Curtis','Asymptotic Regime Trapezoidal-Rule', '', 'Asymptotic Regime
 Simpsons-Rule','','Asymptotic Regime Clenshaw-Curtis','Shared Bound
 of both Clenshaw and Simpsons', 'Location', 'northwest' )
title(' Relative error vs. h on log(x)-log(y) scale, Part 2')
ylabel('relative error: $|\,truth\,-\,approx\,|/|\,truth\,|
$', 'interpreter', 'latex');
xlabel('$n$', 'interpreter', 'latex');
hold off;
```





Functions

```
%trapezodial rule
function [ int_est ] = trap_rule(f , a , b, n)
%{

    Purpose: Matlab Implementation of composite trapezoidal to esimate
 the
    definite integral of f(x) on the interval [ a , b ] with n points.

    Inputs:
        f: function to be integrated
        a,b: end points of the interval to be evaluated
        n: number of points used in the evaluation
    Outputs:
        int_est: estimate off the deginite integral using n points

%}
    %xo = a, xn = b

    %initilize interval and variables
    h = ( b - a ) / n;


    %compute right hand sum
```

```
    sum = 0;
    for i = 1 : 1 : ( n - 1 )
        sum = sum + f( a + i*h );
    end

    %compute esitmate
    int_est = (h/2) * (f(a) + f(b) + 2*sum);
end

%simpsons rule
function [ int_est ] = simp_rule(f , a , b, m2)
%{

    Purpose: Matlab Implementation of composite Simpson's to esimate
 the
    definite integral of f(x) on the interval [ a , b ] with m2
 points.

    Inputs:
        f: function to be integrated
        a,b: end points of the interval to be evaluated
        m2: number of points used in the evaluation
    Outputs:
        int_est: estimate off the deginite integral using n points

%}

%initialization
m = (m2-1)/2;
f_sum = 0;
s_sum = 0;
h = (b - a) / (2 * m);

%first summation
for i = 1 : m
    f_sum = f_sum + f(a + h*(2*i -1) ) ;
end

%second summation
for i = 1 : (m-1)
    s_sum = s_sum + f(a + h*(2*i));
end

%compute estimate
int_est = (h/3) * ( f(a) + f(b) + 4*f_sum + 2*s_sum);

end

%clenshaw curtis

function [x,w]=clenshaw_curtis_rule(n,a,b)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% fclencurt.m - Fast Clenshaw Curtis Quadrature
```

```
%
% [x,w]=fclencurt(N1,a,b);
%
% Compute the N nodes and weights for Clenshaw-Curtis
% Quadrature on the interval [a,b]. Unlike Gauss
% quadratures, Clenshaw-Curtis is only exact for
% polynomials up to order N, however, using the FFT
% algorithm, the weights and nodes are computed in linear
% time. This script will calculate for N=2^20+1 (1048577
% points) in about 5 seconds on a normal laptop computer.
%
% Written by: Greg von Winckel - 02/12/2005
% Contact: gregvw(at)chtm(dot)unm(dot)edu
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
N=n-1; bma=b-a;
c=zeros(n,2);
c(1:2:n,1)=(2./[1 1-(2:2:N).^2 ])'; c(2,2)=1;
f=real(ifft([c(1:n,:);c(N:-1:2,:)]));
w=bma*([f(1,1); 2*f(2:N,1); f(n,1)])/2;
x=0.5*((b+a)+N*bma*f(1:n,2));
end

%do simple clen
function [ int_est ] = clenshaw_solve(f , x , w )
%{

    Purpose: Matlab Implementation to compute estimate of definite
 integral
    using weights and nodes computed in provided code

    Inputs:
        f: function to be integrated
        x: nodes
        w: weights

    Outputs:
        int_est: estimate off the deginite integral using length(x)
 nodes

%}

    %initialize
    n = length(x);
    int_est = 0;

    %compute summation
    for i = 1 : n
        int_est = int_est + ( f( x(i) ) * w(i) );
    end

end
```

Published with MATLAB® R2021a