```
clc
clf
clear all

%**********************************************************************
%
% INPUTS
%
% Click the run button and refer to the command window
% These are the inputs that can be modified by the user
%
% f(x), the function to integrate

  f= @(x) 2000*log(1400/21./x)-9.8*x ;

% a, the lower limit of integration

  a=8 ;

% b, the upper limit of integration

  b=12 ;

% n, the number of segments. Note that this number must be even.

  n=12 ;

%**********************************************************************

disp(sprintf('\n\nSimulation of the Simpson''s 1/3rd Rule'))
disp(sprintf('University of South Florida'))
disp(sprintf('United States of America'))
disp(sprintf('kaw@eng.usf.edu\n'))

disp(sprintf↙
('\n*******************************Introduction*********************************'))
disp('Trapezoidal rule was based on approximating the integrand by a first order')
disp('polynomial, and then integrating the polynomial in the interval of integration.')
disp('Simpson''s 1/3 rule is an extension of Trapezoidal rule where the integrand is')
disp('now approximated by a second order polynomial.')

disp(sprintf('\n\n*******************************Input↙
Data*********************************\n'))
disp(sprintf('     f(x), integrand function'))
disp(sprintf('     a = %g, lower limit of integration ',a))
disp(sprintf('     b = %g, upper limit of integration ',b))
disp(sprintf('     n = %g, number of subdivisions (must be even)',n))
format short g

disp(sprintf('\nFor this simulation, the following parameter is constant.\n'))
% calculate the spacing parameter.
h=(b-a)/n ;
```

```
disp(sprintf('     h = ( b - a ) / n '))
disp(sprintf('       = ( %g - %g ) / %g ',b,a,n))
disp(sprintf('       = %g',h))

disp(sprintf↙
('\n*****************************Simulation*********************************\n'))
sum=0 ;
disp('The approximation is expressed as')
disp(' ')
disp('     approx = h/3 * ( f(a) + 4*Sum(i=1,n-1,odd) f(a+i*h) +')
disp('                            2*Sum(i=2,n-2,even) f(a+i*h) + f(b) )')

disp(' ')
disp('1) Begin summing odd function values at points between a and b not')
disp('   including a and b. Multiply this by 4.')
disp(' ')
disp('       4*Sum (i=1,n-1,odd) f(a+i*h)')
disp(' ')
% sum of functions evaluated at odd spacings
for i=1:2:n-3
  disp(sprintf('          f(%g)',a+i*h))
end
disp(sprintf('       +  f(%g)',a+(n-1)*h))
disp(sprintf('       ------------'))
for i=1:2:n-3
  sum=sum+f(a+i*h) ;
  disp(sprintf('          %g',f(a+i*h)))
end
sum=sum+f(a+(n-1)*h) ;
disp(sprintf('       +  %g',f(a+(n-1)*h)))
disp(sprintf('       ------------'))
disp(sprintf('       4* %g = %g\n',sum,4*sum))
sum=4*sum ;

sum2=0 ;
disp(' ')
disp('2) Continue by summing even function values at points between a and')
disp('   b not including a and b. Multiply this sum by 2.')
disp(' ')
disp('       2*Sum (i=1,n-1,even) f(a+i*h)')
disp(' ')
for i=2:2:n-4
  disp(sprintf('          f(%g)',a+i*h))
end
disp(sprintf('       +  f(%g)',a+(n-2)*h))
disp(sprintf('       ------------'))
% sum of functions evaluated at even spacings
for i=2:2:n-4
  sum2=sum2+f(a+i*h) ;
  disp(sprintf('          %g',f(a+i*h)))
end
sum2=sum2+f(a+(n-2)*h) ;
```

```
disp(sprintf('       +  %g',f(a+(n-2)*h)))
disp(sprintf('       ------------'))
disp(sprintf('       2* %g = %g\n',sum2,2*sum2))
sum2=2*sum2 ;

disp('3) Add f(a) and f(b) to the sums from 1) and 2)')
disp(' ')
disp(sprintf('     %g + %g + %g + %g = %g',f(a),sum,sum2,f(b),f(a)+sum+sum2+f(b)))
sum=sum+sum2+f(a)+f(b) ;

disp(' ')
disp('4) Multiply this by h/3 to get the approximation for the integral.')
disp(' ')
disp(sprintf('     approx = h/3 * %g',sum))
disp(sprintf('     approx = %g * %g',h/3,sum))
approx=h/3*sum ;
disp(sprintf('     approx = %g',approx))

disp(sprintf('\n\n************************Results*****************************'))

% The following finds what is called the 'Exact' solution
exact = quad(f,a,b) ;

disp(sprintf('\n   Approximate = %g',approx))
disp(sprintf('   Exact       = %g',exact))
disp(sprintf('\n   True Error = Exact - Approximate'))
disp(sprintf('              = %g - %g',exact,approx))
disp(sprintf('              = %g',exact-approx))
disp(sprintf('\n   Absolute Relative True Error Percentage'))
disp(sprintf('              = | ( Exact - Approximate ) / Exact | * 100'))
disp(sprintf('              = | %g / %g | * 100',exact-approx,exact))
disp(sprintf('              = %g',abs( (exact-approx)/exact )*100))

disp(sprintf('\nThe Simpson 1/3rd rule can be more accurate if we made our'))
disp(sprintf('segment size smaller (that is, increasing the number of segments).\n\n'))


% The following code sets up the graphical depiction of the method.
x(1)=a ;
y(1)=f(a) ;
hold on
for i=1:n
   x(i+1)=a+i*h ;
   y(i+1)=f(x(i+1)) ;
end
% polyfit is used to fit quadratics to each interval of 3 points.
% Simpson's rule is equivalent to these approximations integrated.
for i=1:2:n
    bg = i ;
    ed = i + 2 ;
    coeffs = polyfit(x(bg:ed), y(bg:ed), 2);
    poly_x1 = [x(bg):(x(ed) - x(bg))/1000:x(ed)];
```

```
    poly_y1 = coeffs(1)*poly_x1.^2 + coeffs(2)*poly_x1 + coeffs(3);
    poly_x1 = [poly_x1(1) poly_x1 poly_x1(end)];
    poly_y1 = [0 poly_y1 0];
    fill(poly_x1, poly_y1, 'y')
end
xrange=a:(b-a)/1000:b;
plot(xrange,f(xrange),'k','Linewidth',2)
title('Integrand function and Graphical Depiction of Simpson''s 1/3rd Rule')
```