The answers to Questions 1.2 and 1.3 are provided on the webpage to serve as a template. Please turn in answers to Questions 1.1 and 1.4; and, if you like, 1.5.

**Question 1.1:** Derive the Simpson rule and prove that it has an $O(h^4)$ error.

*Hint:* To determine the coefficients in the rule, consider the formula
\[
\int_{-h}^{h} f(x) \, dx \approx af(-h) + bf(0) + cf(h).
\]

Making the formula exact for $f(x) = 1$, $f(x) = x$, $f(x) = x^2$ yields three equations for three unknowns. Then to estimate the error, notice that if $f$ is any function with four continuous derivatives, then
\[
f(x) = f(0) + x f'(0) + \frac{1}{2} x^2 f''(0) + \frac{1}{6} x^3 f^{(3)}(0) + \frac{1}{24} x^4 f^{(4)}(\xi)
\]
for some $\xi$ such that $|\xi| \leq |x|$.

**Question 1.2:** The purpose of this question is to demonstrate the importance of avoiding loops when writing Matlab programs. Consider the trapezoidal rule
\[
I \approx h \left( \frac{1}{2} f(x_0) + \sum_{j=1}^{n-1} f(x_j) + \frac{1}{2} f(x_n) \right),
\]
where $n$ is the number of quadrature points and
\[
h = \frac{b-a}{n}, \quad x_j = a + h j.
\]

Consider the following two Matlab functions:

```matlab
function I = trapezoidal1(f,a,b,n)
    h = (b-a)/n;
    I = 0.5*h*(f(a) + f(b));
    for icount = 1:(n-1)
        I = I + h*f(a+icount*h);
    end
    return
end

function I = trapezoidal2(f,a,b,n)
    h = (b-a)/n;
    w = h*[0.5,ones(1,n-1),0.5];
    x = linspace(a,b,n+1);
    I = sum(w.*f(x));
    return
end
```
Let \( t_j = t_j(n) \) denote the time required for Matlab to execute “version \( j \)” of the program. Show via numerical examples that the ratios \( t_j(n)/n \) converge to some numbers \( c_j \) as \( n \) grows, and estimate these numbers.

*Hint:* The timing functions `tic` and `toc` will not give accurate answers when the interval between them is very short. A way around this is to execute the function you want to time (say) 1000 times, and then divide the measured time by 1000.

**Question 1.3:** Implement the trapezoidal rule and the Simpson rule efficiently. Then measure the convergence order of your functions for a variety of different (You know them, of course, but let us measure them as an exercise.) In this exercise, you may use the built-in function

\[
I = \text{quad}(f,a,b,tol)
\]

to compute the “exact” answer. Use your functions to estimate the integral

\[
I = \int_0^1 \cos\left(\frac{1}{0.01 + x^2}\right) \, dx.
\]

Produce (loglog) graphs that plot the error in your function versus \( h \), where \( h = 1/n \) and

\[n = 100, 200, 400, 800, 1600, \ldots, 51200.\]

How would you estimate the convergence order if you do not have a “reference” function to compute the “exact” answer?

**Question 1.4:** Repeat Question 3, but now use Gaussian quadrature. In other words, construct a function with calling sequence

\[
\text{function } I = \text{gaussquad}(f,a,b,n,p)
\]

that uses Gaussian quadrature on \( n \) panels, with \( p \) points on each panel. Estimate computationally the “order” of the method as \( n \) and \( p \) increase. Produce a graph that plots the accuracy of the Simpson rule versus the accuracy of the Gaussian rule for \( p = 5, 10, 15, 20 \) (for the same number of function evaluations).

*Hint:* I believe that there is no built-in Matlab function for generating Gaussian quadrature nodes and weights. On the webpage you’ll find a simple function `lgwt.m` that does this task.

**Question 1.5:** Write a function for estimating the integral

\[
I = \int_c^d \int_a^b f(x_1, x_2) \, dx_1 \, dx_2.
\]

You may use either Gaussian or Newton-Cotes quadrature. Then estimate the integrals

\[
I_1(k) = \int_0^1 \int_0^1 \cos(k x_1 x_2^2) \frac{1}{1 + \cos(x_1)} \, dx_1 \, dx_2,
\]

\[
I_2(k) = \int_0^1 \int_0^1 |\cos(k x_1 x_2^2)| \frac{1}{1 + \cos(x_1)} \, dx_1 \, dx_2,
\]

for different values of the positive integer \( k \). Roughly how large can \( k \) be for you to quickly (say within a second) get 10 accurate digits?