1. **a.** Define a 2-point Gaussian quadrature for computing \( I(f) = \int_{-1}^{1} f(x) \, dx \). Find the nodes, weights and the degree of exactness.  
*Hint: use the Legendre polynomials.*

**b.** Use the Peano Kernel Theorem to find the Peano Kernel and the expression for the error for this quadrature rule.  
*Notes on the Peano Kernel Theorem are linked to my web site.*

2. **a.** Define an \( n \)-point Gaussian quadrature rule for computing \( I(f) = \int_{-\infty}^{\infty} f(x) e^{-x} \, dx \). Use the Laguerre polynomials and the Golub-Welsch algorithm to find the nodes and weights.  
The orthogonality relationships for the Laguerre polynomials:

\[
xP_0 = a_0 P_1 + b_0 P_0,
\]
\[
xP_k = a_k P_{k+1} + b_k P_k + c_k P_{k-1}, \quad k = 1, 2, \ldots
\]
\[
a_k = -(k+1), \quad b_k = 2k + 1, \quad k \geq 0, \quad c_k = -k, \quad k \geq 1.
\]

Make sure to set the matrix \( J \) properly (recall HW6). Find its eigenvectors and eigenvalues in Matlab using the command \([\text{Evecs, Evals}] = \text{eig}(J)\).

Use your quadrature to compute the Gamma-function \( \Gamma(z) = \int_{0}^{\infty} t^{z-1} e^{-t} \, dt \) on \( z \in [0, 1, 4] \) and plot its graph. Check that at the integer values of \( z \) you get the exact value: \( \Gamma(n) = (n-1)! \).

**b.** Use your quadrature rule to compute \( \int_{0}^{\infty} \cos x e^{-x} \, dx \). Compare the results of integration for different numbers of nodes \( n \) with the exact result that you can find via integration by parts. Plot the graph of \( \log(\text{Error}(n)) \).

3. Write a Matlab code that performs Romberg integration with 10 stages. Use it to compute \( I = \int_{0}^{2\pi} \cos x e^{-x} \, dx \). Set \( h_0 = \frac{\pi}{8} \) and \( q = 2 \). Find the exact value of this integral and compare it with the results of Romberg integration. Plot the graph of \( \log(|T_{m,k} - I|) \) versus \( k \).

4. Download the c-code for the Adaptive Simpson Rule from my website. You can run it on Linux or Mac from the terminal window as follows: open terminal, type `gcc AdaptiveSimpson.c -lm -g` to compile it, and then type `./a.out` to run it.

**a.** Integrate \( \exp \) from 0 to 1 using this routine. (You can find out how to do it from the comments at the beginning of the code. The lines with comments are marked with `//`.) Find out how many nodes you need to meet the error tolerance of \( 10^{-p} \) for \( p = 4, \ldots, 12 \). Compare your results with the exact answer. Check that the true error corresponds to the tolerance set. How are the nodes arranged along the \( x \)-axis?

**b.** Invent a function for which the need for adaptive integration would be essential (e.g. \( f(x) = (x + 0.05)^{-1} \sin \frac{1}{x + 0.05} \)). Set tolerances of \( 10^{-p} \) for \( p = 4, \ldots, 12 \). See how many nodes do you need for each of them, and how are these nodes arranged. Plot the graph of the function and the nodes for one of the tolerances for which the nodes are still distinguishable, but the pattern is clear.