`Assignment 6 --- QMB 3311 --- Spring 2025 --- Due: March 31, 2025`

Instructions:

Complete this assignment within the space on your group's GitHub repo in a folder called `Assignment_06`. In this folder, save your answers to a file called `my_A6_module.py`.

For all of the exercises, use your examples to test the functions you defined. Since the examples are all contained within the docstrings of your functions, you can use the `doctest.testmod()` function within the doctest module to test your functions automatically.

Don't worry about false alarms: if there are some "failures" that are only different in the smaller decimal places, then your function is good enough. It is much more important that your function runs without throwing an error.

1. A *Taylor Series* is the sum of a sequence of terms that approximates the value of a function in the neighborhood of a particular point. For the natural logarithm function, $\ln(z)$, the Taylor series expansion around the point z_0 is

$$\ln(z) = \ln(1) + \sum_{k=1}^{\infty}(-1)^{k-1}\frac{1}{k}(z-1)^k$$

   and since $\ln(1) = 0$, the approximation can be calculated without use of the `math.log` function. Write a function `ln_taylor(z, n)` that calculates the first n terms of this approximation. In crafting your examples, you will find that this approximation of the `math.log` function is more accurate for values of z close to 1 or when a large number of terms n is used.

2. Another way to solve for the value of the $\ln(z)$ function is to transform it into a root-finding problem: solve for the value of x such that:

$$e^x = z \text{ or } e^x - z = 0$$

   which is true when $x = \ln(z)$, for a given z. Write a function `exp_x_diff(x, z)` that returns the value of $e^x - z$.

3. Now solve the function $f(x) =$ `exp_x_diff(x, z) == 0` for the root of x^*. Write a function `ln_z_bisect(z, a_0, b_0, num_iter)` that calculates the root using the bisection method. Essentially, this will produce an algorithm for calculating the natural logarithm of `z`. Follow the algorithm used in the lecture that recursively splits the interval in half and, in each iteration, assigns the midpoint `m_i` to the endpoint for which `exp_x_diff()` has the same sign as `exp_x_diff(m_i, z)`. It should start with the interval (a_0, b_0) and perform `num_iter` iterations. To guarantee a solution, your function should first check that the signs of $f(x)$ differ at the end points, i.e. $f(a_0) \times f(b_0) < 0$

4. Next, solve for the roots using Newton's method. Before doing this, you will nedd a function that returns the derviative. Write a function `exp_x_diff_prime(x, z)` $(= f'(x))$ that returns the derivative of `exp_x_diff(x, z)` with respect to `x`. Note that `z` is a constant in this function.

5. Now, use Newton's method to find the natural log of z. Use the recurrence relation

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

repeatedly until it reaches a value of x_n such that $|f(x_n)| < \varepsilon$ where ε is a small number reprented by `tol`. Write a function `ln_z_newton(z, x0, tol num_iter)` that solves for the natural logarithm of z by using the inital value `x0`, up to a level of tolerance `tol`, and a maximum number of iterations `num_iter`. It should print a warning message if it reaches the maximum number of iterations before the `exp_x_diff(x_i, z)` function value is less than `tol`.

6. In the next exercise, you will use the fixed-point method to find a root of this function. A *fixed point* of a function $g(x)$ is a value of x^* such that $g(x^*) = x^*$. Consider the function

$$g(x) = \frac{1}{2}(z - e^x + 2x)$$

Note that $x^* = \ln(z)$ is a fixed point of $g(x)$; that is, $g(\ln(z)) = \ln(z)$. Write a function `exp_x_fp_fn(x, z)` that returns the value $g(x)$ for a given value of z.

7. Finally, use the fixed-point method to find the natural logarithm of z. THat is, use the recurrence relation $x_{i+1} = g(x_i)$ repeatedly until it reaches a value x_n such that $|g(x_n) - x_n| < \varepsilon$, where ε is a small number represented by `tol`. Write this algorithm within a function `ln_z fixed_pt(z, x0, tol, num_iter)`.