PHYS 331 – Numerical Techniques for the Sciences I
Homework 2: Number Representation and Root Finding

Posted Tuesday, September 5, 2023
Due Friday, September 22, 2023.

Problem 1 – Accumulation of Numerical Error (5 points)
This problem will help you explore the limitations of precision of numerical data types of different sizes. The NumPy library provides functions for creating half-precision (16-bit), single-precision (32-bit), and double-precision (64-bit) floating point numbers. These functions are named `float16`, `float32`, and `float64`, respectively.

Open the provided notebook `problem1.ipynb` and examine it. Three functions are provided for you – `calculateSum_16bit`, `calculateSum_32bit`, and `calculateSum_64bit`. Each of these functions accepts an argument `delta` (which is a typical double-precision floating point number in Python), and repeatedly adds the 16-, 32-, or 64-bit representation of `delta` to a variable. The operation is repeated `1/delta` times, such that the returned result should be exactly 1 (since clearly $\Delta \cdot 1/\Delta = 1$). Convince yourself that these three functions perform this task.

(a) *(1 point)* Notice that an error message is printed to the screen if $\Delta = 0$. What would happen if this error case were not checked?

(b) *(2 points)* In the provided template file, write code in the function `main` which prints the result of each function for the parameter values $\Delta = 10^{-1}$, 10^{-2}, 10^{-3}, 10^{-4}, and 10^{-5}. Describe your observations.

(c) *(2 points)* For a working precision of 16-bits, 32-bits, and 64-bits, what value of Δ causes the error message described above to be displayed?

Problem 2 – One-Dimensional Root Finding Using the Bisection Method (19 points)
Implement a root-finding algorithm in the provided template notebook `problem2.ipynb` that uses the bisection method to find a root of a one-dimensional function. Your solution should be implemented as the function `rf_bisect`, which takes the following five parameters:

1. The function $f(x)$ for which the algorithm should find a root. This argument should refer to a Python function that accepts `x` as a parameter and returns a floating-point number `y`$= f(x)$. Note that four examples of this function are provided in `problem2.ipynb`.

2. The lower limit of the initial bracket to search.

3. The upper limit of the initial bracket to search.

4. The numerical tolerance the returned root should achieve.
5. The maximum number of iterations the bisection algorithm may take.

The function `rf_bisect` should return a tuple `(root, iters)` where `root` refers to the computed root, and `iters` refers to the number of actual iterations the algorithm took to complete the computation. The function signature and return statement are provided for you in the template.

Test your root-finding algorithm with the following four functions on the interval $x = [0, 1]$, with accuracy tolerances of 10^{-3}, 10^{-6}, and 10^{-12}:

$$\begin{aligned} f_1(x) &= 3x + \sin(x) - \exp(x) \quad (1)\\ f_2(x) &= x^3 \quad (2)\\ f_3(x) &= \sin\left(\frac{1}{x + 0.01}\right) \quad (3)\\ f_4(x) &= \frac{1}{x - 1/2} \quad (4) \end{aligned}$$

Modify the function `main` to perform the following:

1. plot the function being passed to `rf_bisect`,
2. find and output the roots for each of the four functions at each of these tolerances. An example of how you should output the result for the function $f_2(x) = x^3$ on the interval $[-1, 1]$ with a maximum of 25 iterations is provided for you.

For which of the four functions listed above is `rf_bisect` bound to fail on the given interval? Which solution is meaningless? Suggest remedies to resolve the cause of failure. Submit your assignment with plots for all functions which do not cause the root-finding algorithm to fail.

Note: Do not simply copy any part of the example provided in the textbook – the point is to learn how to write this algorithm yourself!

Problem 3 – Visualization of Bisection (6 points)
In the previous problem, you computed the roots of various functions using the bisection method. Some of these functions posed different challenges when finding roots using this algorithm. In this problem you will create a "diagnostic" tool to visualize when and how different cases fail as the bisection algorithm progresses.

In the provided template file `problem3.ipynb`,

1. Copy your implementation of `rf_bisect` from Problem 2 into the notebook for this problem as indicated;

2. *(2 points)* Modify `rf_bisect` to instead return a tuple of two NumPy vectors containing the sequence of x_{mid} and f_{mid} (i.e. the sequence of middle points calculated by the algorithm and the corresponding values of the function);

3. *(2 points)* Add code to `main` for each function $f_i(x)$ to plot the $(x_{\text{mid}}, f_{\text{mid}})$ sequence over the plot of $f_i(x)$, for each $f_i(x)$ for which `rf_bisect` does not fail;

4. *(2 points)* Display a second plot for each $f_i(x)$ which shows the error at each iteration versus the number of iterations.