PHYS 331 – Introduction to Numerical Techniques in Physics

Homework 2: Number Representation and Root Finding

Due Friday, Sept. 8, 2017, at 11:59pm.

Problem 1 – Accumulation of Numerical Error (10 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 *HW2p1template.py* template file 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 $1/\Delta$ 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) Notice that an error message is printed to the screen if $\Delta = 0$. What would happen if this error case were not checked?

b) In the provided template file, write code in a 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) For a working precision of 16-bits, 32-bits, and 64-bits, below what value of Δ causes the error message described above to be displayed? Find the threshold within the nearest power of 10 for each precision.

<u>Summary of deliverables:</u> Parts (a), (b), and (c) all require free-form answers in your *HW2.pdf* file. Part (b) requires changes to the template that can then be submitted as your *HW2p1.py* file.

Problem 2 – One-Dimensional Root Finding Using the Bisection Method (15 points)
Implement a root-finding algorithm in the provided *HW2p2template.py* file that uses the bisection method to find a root of a one-dimensional function. *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!* Your solution should be implemented as the function `rf_bisect`, which takes the following five input parameters:

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

$$f_1(x) = 3x + \sin(x) - \exp(x) \tag{1}$$

$$f_2(x) = x^3 \tag{2}$$

$$f_3(x) = \sin\left(\frac{1}{x + 0.01}\right) \tag{3}$$

$$f_4(x) = \frac{1}{x - 1/2} \tag{4}$$

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; if it is not possible to do so for a given function using the standard bisection method, explain why. An example of how you could 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, although you can further automate and revise it for readability if you wish. You should also consider what maximum number of iterations should be used.

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.

<u>Summary of deliverables:</u> Free-form answers to the above questions go in your *HW2.pdf* file. Your *HW2p2.py* file should contain your modified template, and be able to be executed to show all of the results requested (plots and values of roots for each function).

Problem 3 – Visualization of Bisection / Developing Diagnostic Tools (15 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 a new file named *HW2p3.py*,

1. Copy your implementation of `rf_bisect` from Problem 2 into the notebook for this problem.
2. 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. Modify `main` for each function $f_i(x)$ to plot the (x_{mid}, f_{mid}) sequence over the plot of $f_i(x)$, for each $f_i(x)$ for which `rf_bisect` does not fail; this only needs to be done for the finest tolerance value (10^{-12}). Use appropriate marker styles to visualize the individual data points of (x_{mid}, f_{mid}).
4. Display a second plot for each $f_i(x)$ which shows the error at each iteration (assuming the final value is the “true” value).

Problem 4 – Solving a Real-World Numerical Problem (10 points)
Do problem 18: Bernoulli’s equation in Chapter 4 of the textbook, using the bisection method. Use `rf_bisect` from problem 2. Develop your own style – write functions as needed; but make sure that (1) you show a plot of the function for which you are finding the roots, (2) you compute all roots that are physically plausible, (3) your code is well-commented, and (4) it runs and displays all results (plots, roots, etc) automatically. Choose a tolerance that is sufficient for the problem in a real-world scenario.