homework2

January 18, 2024

Introduction to Computer Programming for the Physical Sciences

Joseph F. Hennawi

Winter 2024

Open a new Jupyter notebook

Name your notebook with your name and Homework 1

Open a Markdown cell at the top and write your name and Homework 1

Open a Markdown cell before each problem and write e.g. Problem 1, Problem 2(a), etc.

Please abide by the Policy and Guidelines on Using AI Tools

Once you finish the problems: 1) Restart the Python kernel and clear all cell outputs. 2) Rerun the notebook from start to finish so that all answers/outputs show up. 3) Save your notebook as a single .pdf file and upload it to Gradescope on Canvas by the deadline. No late homeworks will be accepted except for illness accompanied by a doctor's note.

1 Homework 2

1.1 Problem 1: Range of a Projectile

Write a Python function that computes the range of a projectile that is launched from height y_0 above the ground at speed v_0 at angle α above the horizontal direction. By *range*, I mean the horizontal distance traveled before reaching the ground. Neglect air resistance and Coriolis effects. For $y_0 = 11$ m and $v_0 = 13$ m s^{-1}, which of the following angles gives the biggest range: $\alpha = 20°, 30°, 40°, 50°, 60°$?

[4]:
```
# Your solution here.
```

1.2 Problem 2: Fun with Conditional Statments

Using `if`, `elif`, `else` statements, write a Python function that takes any three distinct real input numbers a, b, and c, and returns the same values in a tuple in order of smallest to largest. For example, if $a = 3$, $b = 1$, and $c = 2$, then the function should return the tuple $(1, 2, 3)$. If $a = 3$, $b = 2$, and $c = 3$, then the function should return the tuple $(2, 3, 3)$.

[5]:
```
# Your solution here.
```

1.3 Problem 3: Machine Epsilon

In the lecture we discussed the limited precision of floating point numbers and floating point arithmetic. An important value to quantity is floating-point accuracy which is referred to as the *machine epsilon*. Please read this Wikipedia article on the machine epsilon to learn more about this important concept.

The machine epsilon is defined as the smallest number ϵ_m such that $1 + \epsilon_m > 1$. According to the Wikipedia article, the machine epsilon in python can be estimated to within a factor of two via the algorithm:

```
epsilon_m = 1.0
while (1.0 + 0.5*epsilon_m) != 1.0:
    epsilon_m /= 2.0
```

a) Write a python function that implements this algorithm and returns the machine epsilon. Which float-type is used in Python (see the table of the Wikipedia article)?

b) In lecture it was argued that in Python the smallest number that can be represented in python is about `1e-308`, which is many orders of magnitude smaller than the (_m) that you just derived. What is the difference between the smallest representable floating point number and the machine epsilon?

c) Consider 32bit floating point numbers, or so called single-precision. To within an order of magnitude estimate the machine epsilon, the smallest number that can be represented, and the largest number that can be represented. Repeat your estimates for 8bit floating point numbers.

[6]:
```
# Your solution here.
```

1.4 Problem 4: Numerical Derivatives

In this problem we will explore the accuracy of numerical derivatives. Consider the function $f(x) = x^2(x-1)$

a) Analytically compute $f'(x)$ and evaluate it at $x = 1.0$.

b) Write a python function that estimates the derivative of $f(x)$ numerically using the forward difference formula:

$$f'(x) \approx \frac{f(x+h) - f(x)}{h}$$

where h is a small number. **c)** Write another python function that estimates the derivative of $f(x)$ numerically using the symmetric difference formula:

$$f'(x) \approx \frac{f(x+h) - f(x-h)}{2h}$$

d) Calculate $f'(1.0)$ using your two numerical derivative functions for $h = 10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}, ...$ until something *really bad happens* (see below). Print out both h and $f'(1.0)$, as h becomes smaller and smaller. Format the output of $f'(1.0)$ to show 16 digits after the decimal point. Do the calculation using the built-in python float data type (nothing fancy please!)

e) Based on your outputs from above, you should see that the symmetric difference formula is always more precise at a given value of h. To understand why this is the case we need a bit of

calculcus. The Taylor expansion of $f(x)$ around x is given by:

$$f(x+h) = f(x) + hf'(x) + \frac{h^2}{2}f''(x) + \frac{h^3}{3!}f'''(x) + \frac{h^4}{4!}f''''(x) + \cdots$$

which states that for small values of h, the function can be expanded as a sum of powers of h and higher order derivatives of $f(x)$.

Derive expressions for $f'(x)$ using the Taylor expansion above for the two numerical derivative formulas that we employed in part (b) and (c).

f) Based on your answers to part (e), explain why the symmetric difference formula is always more precise than the forward difference formula in the limit $h \to 0$. Note that the amount of computational work for both of the derivative estimators is the same, i.e. the function is evaluated at two locations, and then division by h or $2h$ is performed. Hence, this problem illustrates that numerical derivatives should always be calculated using symmetric differences whenever possible.

g) As h becomes *too small* the precision of both of the derivative estimators starts to degrade. This is because when h is extremely small, taking the difference of $f(x+h)$ and $f(x-h)$ (or $f(x+h)$ and $f(x)$) becomes problematic as you are subtracting two numbers that are very close to each other. Read this Wikipedia article on catastrophic cancellation and describe in your own words why the numerical preision degrades when h becomes too small.

It can be shown that catastrophic cancellation starts to degrade results when $h \approx x\sqrt{\epsilon_m}$ (forward difference estimator) or $h \approx x\epsilon_m^{2/3}$ (symmetric difference estimator), where $\epsilon_m \simeq 2 \times 10^{-16}$ is the machine epsilon that we derived in Problem 3. These formulas are reliable provided that x is not too close to zero (in our problem $x = 1$ so we are okay). For more background on where these scalings come from see Chapter 5.7 of Numerical Recipes

[7]:
```
# Your solution here please
```