Introduction to Computer Programming for the Physical Sciences

Joseph F. Hennawi

Spring 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.**

☐ For parts of problems that require analytical solutions you can perform your calculations using a pencil and paper. Then photograph your work and convert the photograph to a .pdf file using an online

tool. Homework assignments can only be submitted as a single .pdf file, so you will also need to figure out how to concatenate your photo .pdf file and your notebook .pdf file into a single .pdf file that you can submit. Online websites can do this for you. Alternatively, you can code up the analytical solution to your problems in a notebook Markdown cell using the LaTeX mathematical rendering language. This is harder but a chatbot can help you learn it.

Homework 2

Problem 1: 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 $(2, 3, 3)$.

In [ ]:
```
# Your solution here.
def CompareNumber(a,b,c):
    if(a>b):
        i=a
        a=b
        b=i
    # if a>b switch a and b
    if(a>c):
        i=a
        a=c
        c=i
    # if a>c switch a and c
    if(b>c):
        i=b
        b=c
        c=i
    #if b>c swith b and c
    return a,b,c

print(CompareNumber(2,3,1))
```

```
(1, 2, 3)
```

Problem 2: 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 (ϵ_m) that you just derived. What is the difference between the smallest representable floating point number and the machine epsilon?

c) Consider 32bit (binary32) 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 16bit (binary16) floating point numbers.

In [ ]:
```
# Your solution here.
# a) shows as following:
def FindEpsilon():
    epsilon = 1.0
    # set the initial value of epsilon
    while(float(1.0 + 0.5*epsilon) != float(1.0)):
        #use the float() to make sure that they are use float to compare
        #will not stop until the epsilon is found
        epsilon = epsilon*0.5
        #reduce the epsilon if it is not the one we need
    return epsilon
    # return the value of epsilon we find

print(FindEpsilon())
```

```
2.220446049250313e-16
```

b)

It is impossible for every variable to have the max accuracy. If doing so, it may take up too much space in ram, which is not necessary.

c)

for binary32 the machine epsilon should be the squre root of binary 64, which should be approximately 1.19e-7. The smallest number should be -3.4028234664e38 and largestest should be 3.4028234664e38 for binary16 the

machine epsilon should be the square root of binary 32, which should be approximately 9.773e-4. The smallest number should be -65504, and largest should be 65504

Problem 3: 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 evauate 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 $h = 10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}, \ldots$. 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

a)

$f'(x) = 3x^2 - 2x$

$f'(1.0) = 1$

In [ ]:
```
# Your solution here please
# b) is shown as below
def fx(x):
    return float(x**3-x**2)
# write a function of f(x), to reduce work.
def dfdxWithForward(x,h):
    return float((fx(x+h)-fx(x))/(h))
# use the formula given to return the value we need
# c) is shown as below
def dfdxWtihSymmetric(x,h):
    return float((fx(x+h)-fx(x-h))/(2*h))
# do as the instruction asks
# d) shown as below
for i in range(16):
    # I choose 15 at max just because it looks really bad since h = 1e-16
    h = 10**(-i)
    print(h,dfdxWithForward(1.0,h),dfdxWtihSymmetric(1.0,h))
```

```
1 4.0 2.0
0.1 1.2100000000000022 1.010000000000001
0.01 1.0201000000000127 1.0001000000000038
0.001 1.0020010000000301 1.0000010000000281
0.0001 1.0002000099995634 1.0000000100002238
1e-05 1.0000200000970239 1.000000000095369
1e-06 1.00000199987349 0.9999999999177334
1e-07 1.0000002004240116 1.0000000000287557
1e-08 1.0000000161269895 0.9999999994736442
1e-09 1.000000082740371 1.0000000272292198
1e-10 1.000000082740371 1.000000082740371
1e-11 1.000000082740371 1.000000082740371
1e-12 1.000088900582341 1.0000333894311098
1e-13 0.9992007221626409 0.9997558336749535
1e-14 0.9992007221626409 0.9992007221626409
1e-15 1.1102230246251565 1.0547118733938987
```

e)

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

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

for the method shows in b)

$$f'(x) \approx f'_{(b)}(x) = \frac{f(x+h) - f(x)}{h} = \frac{hf'(x) + \frac{h^2}{2}f''(x) + \frac{h^3}{3!}f'''(x) + \frac{h^4}{4!}f''''(x}{h}$$

for the method shows in c)

$$f'(x) \approx f'_{(c)}(x) = \frac{f(x+h) - f(x-h)}{2h} = \frac{2hf'(x) + \frac{2h^3}{3!}f'''(x) + \frac{2h^5}{5!}f'''''(x) +}{2h}$$

f)

Thus, we can find the difference between the approximation in order to find out which has more factors.

$$f'_{(b)} - f'_{(c)} = \frac{h}{2}f''(x) + \frac{h^3}{4!}f''''(x) + \cdots$$

Thus, we can find out that $f'_{(b)}$ has more factors. And it is reasonalbe to state that since $f'_{(b)}$ has more facotrs than $f'_{(c)}$, $f'_{(c)}$ is a better approximation compared to $f'_{(b)}$.

g)

Since all the values stored in computer is not the acutual value, it is always an approximation with precision of ϵ_m. Thus, when h is very small, the f(x) and f(x+h) will have such a small difference that the result may not be trusted.