Due Fri Sept 9

AM 111 Problem Set 01 at 5pm

There are some problems that will require programming. For other problems, you are welcome to do them by hand or to write a program to complete them.

Use a single python notebook or file for all programming work on the problem set. Submit that source file on Canvas (in case the source is needed during grading). Print to pdf and submit the pdf on Gradescope. Use Gradescope's problem tagging to tag the locations in the pdf that correspond to particular problems.

- As part of completing the assignment, fill out the online cover sheet on Canvas to name your collaborators, list resources you consulted, and estimate the time you spent on the assignment.

  For coding related resources (looking up Python commands, or syntax, etc), include the references as comments within your code instead of adding them to the cover sheet.

  You may copy snippets of code from other sources so long as there is a comment indicating the source of the code.

- Submit your work on this problem set via Gradescope (find the link on Canvas).

- Collaboration is encouraged on all assignments. Individual written work for this class should be your own. You are encouraged to discuss the mathematics and to work out the math together, then put away or erase joint work before writing up your solution.

  If you believe your work is incorrect, please do show it to your classmates and the teaching staff. If you believe your solution to be correct, I encourage you to discuss or describe your solution, **without actually showing your written work to others.**

The late work policy for problem sets is available in the syllabus.

---

1. (binary and floating point)
   (a) (Sauer §0.2: 1ac, 2ac): Find the binary representation of the base 10 numbers:
      i. 64
      ii. 79
      iii. $1/8$
      iv. $35/16$

   (b) (Sauer §0.3: 2ac): Express the following base 10 numbers as floating point numbers. Use the Rounding to Nearest Rule.

   Example notation for expressing the floating point number (have 52 bits in the box):

   $$\text{fl}(\frac{1}{3}) = +1.\boxed{0101010101010101010101010101010101010101010101010101}\times 2^{-2}$$

      i. 9.5
      ii. 100.2

   (c) (Sauer §0.3: 9)

   Explain why you can determine machine epsilon on a computer using floating point numbers with rounding to nearest by calculating $(7/3 - 4/3) - 1$.

Does $(4/3 - 1/3) - 1$ also give ϵ_{mach}?

To provide your explanation, convert to floating point numbers and carry out the machine arithmetic.

2. (condition number) (based on Greenbaum and Chartier §6.3: 5)

Suppose a_0 dollars are deposited in a bank paying 5% interest per year. When the interest compounds n times per year, the amount in the account after the first compounding will be $a_0\left(1+\frac{0.05}{n}\right)$.

At the next compounding, the interest is paid on $a_0\left(1+\frac{0.05}{n}\right)$ so the amount will be

$$\left(a_0\left(1+\frac{0.05}{n}\right)\right)\left(1+\frac{0.05}{n}\right) = a_0\left(1+\frac{0.05}{n}\right)^2.$$

After one year (n compoundings), the amount will be $a_0\left(1+\frac{0.05}{n}\right)^n$.

Let $\mathcal{I}_n(x) = \left(1+\frac{x}{n}\right)^n$, the *compound interest formula*.

(a) Find an expression for the relative condition number, $\kappa_{\mathcal{I}_n}(x)$, for $\mathcal{I}_n(x)$.

For $x = 0.05$ would you consider the problem to be well conditioned? Ill conditioned? Does your assessment vary with n? Provide a brief justification/explanation.

(b) For fixed x, $\lim_{n\to\infty} \mathcal{I}_n(x) = e^x$

Use the Python code below to see values of $\mathcal{I}_n(x)$ for $n = 1, 10, 100, 1000, ..., 10^{19}$.

(type this code in or find it on Canvas/github/FAS On Demand)

```
n = 1
x = 0.05
interest_of_x = []
for val in range(0, 20):
    interest_of_x.append(['{:.3e}'.format(n), (1+x/n)**n])
    n = 10*n
# print formatting help from
# https://stackoverflow.com/questions/58722579/python-print-list-of-numbers-in-sci

print(*interest_of_x, sep='\n')
# more print formatting help from
# https://www.geeksforgeeks.org/print-lists-in-python-4-different-ways/
```

Compare these values to

```
import numpy as np

np.exp(x)
```

For which n are $\mathcal{I}_n(x)$ and e^x closest?

Does your calculation converge to e^x as n increases?

(c) For the value of n you chose as yielding the closest value to e^x, treat $\left(1+\frac{x}{n}\right)^n$ as an approximation to e^x.

For this n, compute the condition number from your values:

$x = 0.05$,

$y = e^x$,

$\hat{y} = \left(1+\frac{x}{n}\right)^n$,

and $e^{\hat{x}} = \hat{y}$.

Compare this to your condition number from part (a).

For $n = 18$ or $n = 19$, what happens to the calculation in part(b)? Can this error be explained by the condition number? If not, provide an alternate explanation.

3. (more floating point) (from Ciocanel Spring 2022)

(a) Write a program to determine the machine error of either a `float16`, `float32`, or `float64` in Python.

To do this, `import numpy as np` and set `epsilon = np.float64(1.0)`. (Use `float16` or `float32` for all math if you prefer).

Create a loop, and each time through the loop assign `x = np.float64(1.0) + epsilon`. Then divide `epsilon` by 2 (`epsilon = epsilon/np.float64(2)`).

Continue looping until the value of `np.float64(1.0)+epsilon` is no longer greater than 1.

You might use a `while` loop for this.

Use your results to determine the number of bits in the mantissa of the data type.

Each division by 2 is shifting the decimal/binary/radix point of your binary number, so counting these divisions could help. `np.log2` could also help.

(b) Write a program to determine the overflow of a float of your choice.

Initialize a variable `r = np.float32(1.0)` (or `float16` or `float32`).

Create a loop that prints out a value for `r = np.float32(1.0)` and for `np.log2(r)` and then doubles `r` each time through the loop (be sure `r` remains the correct type of float when you double it). Repeat until the value of `r` becomes infinite. Find the value `k`, where `r = 2**k` for the largest `r` before overflow.

(c) To check your work, compare your results to information about the data type in Python. You can find that info using following commands

```
print(np.finfo(np.float16()))
print(np.finfo(np.float32()))
print(np.finfo(np.float64()))
```

4. (Calculus review) (Sauer §0.5) Review the Fundamentals of Calculus section in Sauer 2017.

Find a copy on Canvas.

(a) Complete Sauer §0.5: 3ab

(b) Complete Sauer §0.5: 5ab

5. (Numerical error in a stock index) *(Thomas Fai, AM 111 Spring 2017. See example in Greenbaum and Chartier)*

   Read this problem in its entirety before beginning work on it.

   A famous example of roundoff error was a short-lived index devised at the Vancouver stock exchange[1]. The index contained 1400 stocks listed on the exchange, and each stock was weighted equally in determining the value of the index (most other indexes are weighted so that large companies count more than small ones). At the time the index was started in January 1982, the sum of the initial selling prices of all 1400 stocks (the baseline sum) was rescaled to give the index an initial value of 1000.

   Taken together, the stocks in this index underwent changes in price a total of 2800 times per day. Each time one of the stocks changed its price, the index was updated as follows:

$$\text{New Index} = \text{Old Index} + (\text{Change in Stock Price})\frac{1000}{\text{Baseline Sum}}$$

   Then, after each change, the index was truncated after the third decimal place. For example, if after a change in stock price the index stood at 735.32567, the computer simply dropped the last two digits, making it 735.325.

   (a) Write two functions:

   - function `truncate(x)` that returns `trunc_x`, a truncation of the number x after the third decimal place.
   - function `update_index(old_index, delta, baseline_sum)` that returns `new_index`, as defined above. `delta` is the change in stock price of a single stock, `old_index` is the previous value of the index, and `baseline_sum` is the original sum of the stock prices

   Find commenting conventions for Python here.

   (b) Write a program that simulates the evolution of this index over time.

   (c) List any assumptions you made about how the market works as part of your simulation.

   (d) Plot the evolution of the index over the first day, and over a 22 month period (you can assume all months have the same number of trading days).

   Write a brief description of what you see in your plot.

   (e) Given the truncation procedure used in updating the index, by how many points, on average, would you expect the index to drop from one change of stock price? What about in 1 day? Or in 22 months?

   (f) After 22 months, the actual index stood at 524.881. What can you infer about the evolution of the market during this time: was it a bull market or a bear market? Explain your thinking.

   (g) Make a suggestion for how to modify the procedure used to update the index. Implement your modification (you will need to write a new function to replace the function truncate from part (a)). Plot the evolution of your modified index over 1 day and over 22 months. Compare your modified index to the evolution of the actual index (compare the indices using identical values of the change in stock price at all time points). What do you observe?

(h) If you have spent more than 9 hours on this problem set, write a note to that effect and skip this problem or parts of this problem.

This problem is provided without information about the Python functions you might use to complete it, and without guidance for some of the assumptions you'll need to make about the index.

If you'd like more information/guidance, ask us on the Ed discussion board

References

[1] Kevin Quinn. Ever had problems rounding off figures? this stock exchange has. *Wall Street Journal*, page 37–37, Nov 1983.