Project 2: Binary Numbers and Computational Errors

Due Dates

Week 4, Monday at 4:59

Your submission should be a single zip file containing all the .py files containing your code as well as a Single typed PDF document with your test results and answers to the problems/questions. Word documents will not be accepted. Your zip file should be in the form <lastname>_project_< N >.zip where you replace <lastname> with your last name and <N> with the project number. Upload your zip file via the link provided on the project webpage found at http://urminsky.ca.

Documentation: *Make sure that your code is well enough commented so that it can be easily understood by almost anyone, including yourself, if you look at it again at the end of the course (when you've forgotten how you coded it.)*

Project Briefing

Part 1

(a) Suppose you have a primitive computer that stores non-negative integers in 8-bit words. The largest number that can be stored is $IMAX = 11111111$. Similarly, the smallest is $IMIN = 00000000$. A random example of another number that can be stored is 11010011. What are the base-10 values of these three numbers?

(b) (b) Plancks constant is $h = 6.626?10 - 34$ J·s. This time you have a slightly more advanced computer that stores floating-point numbers in 16-bit words. Can h be stored in the computer exactly as given above? If not, what is the mantissa and exponent of the binary (base-2) floating-point number that gives the closest approximation and what is the base-10 value of this approximation? Allow 1 bit each for the signs of the mantissa and exponent. Hint: Work out the binary form of the base-2 exponent first (recalling that the mantissa in binary needs to be between ? and 1) and determine how many bits you will need to store this exponent. Then work out the number of bits left over for the mantissa (remembering to allow 2 bits for the signs).

Part 2

This is an exercise designed to demonstrate both truncation and round-off errors. Consider the series

$$f_1(N) = \sum_{n=1}^{N} \frac{1}{n^2}$$

As $N \to \infty$, the series converges to $f_1(\infty) = \pi^2/6$. See (http://en.wikipedia.org/wiki/Basel_problem for proofs.)

Design, write, and test a PYTHON function called pi_series that uses this series to compute the value of π. The number of terms N should be supplied as an input parameter to the function. The function should return the estimated value of as an output parameter. You will need a for loop to calculate the sum of the terms. You then need to call the function to use it. For example

```
def pi_series(N):
    #this is where you compute the series and then return π
    ...
    ...

# now call the function
N = 1000
print pi_series(N)
```

Once your function is working, use it to make a table of the number of terms N needed to obtain a value for to within 10%, 1%, 0.1%, 0.01%, 0.001%, and 0.0001% of the exact value. Is there a relationship between the percentage error and N? (This tells you how rapidly your algorithm for computing converges to the correct value.) Explore this question by constructing a suitable graph, thinking carefully about (or experimenting with) the best way to plot the results so that you can most clearly see the relationship! (In particular, can you plot it in a way that gives you a straight line?)

Now consider the series where you sum down from N to 1 instead of up from 1 to N:

$$f_2(N) = \sum_{n=N}^{1} \frac{1}{n^2}$$

Since f_2 is mathematically equivalent to f_1, we would expect it to give the same value. Lets see if it does. Create a new function, pi_2series, by modifying the original so that it calculates π using f_2. Use the same value of N for both f_1 and f_2. Run for large values of N. (Start with $N = 10000$ and go up by, say, powers of ten from there.) Do the two values of π agree? If not, what is the problem?

Part 3

What if you want your result to converge faster to the correct value of π ? By considering the close relationship between sums and integrals, can you figure out how to approximate the remaining terms in the infinite series (i.e., the terms $N+1$ to ∞ that you are not including) by an integral that you can then obtain an analytic expression for? Implement this as a new function pi_series_better ; test and plot the relationship between the percentage error and N with this modified algorithm. How has the relationship changed? You should find that there is an optimal value to choose for the lower limit of your integral in order to get the fastest convergence. What is this value and how fast does the algorithm then converge?

Part 4

Consider the one dimensional mapping

$$x_{n+1} = \lambda x_n(1 - x_n)$$

1. Write a python script logistic1.py to compute the first 20 iterations of the map starting at $x_0 = 0.12$ and a second orbit $y_0 = 0.1200001$ for $\lambda = 1.1$. Your script should calculate the difference, $err_n = |x_n - y_n|$, between the two orbits at each time step and plot the difference err_n vs. n. Choose an appropriate scale for your axis and label your axis and plot.

2. Using your script logistic2.py, create a new script logistic2.pi in which $\lambda = 3.9$. Again, plot the difference between the values at each step on the orbit. Choose an appropriate method of plotting the information.

What do you notice is different between the two cases above? Can you suggest an explanation for the two different results?