MTH 3300
Spring 2019
E. Fink
March 5, 2019

Homework 5

Directions: Download the template files I have provided on Blackboard. Then open Spyder, load these template files, and write the following programs. Submit your source code to me via Blackboard, in `.py` format; do NOT send any other files. READ THE INSTRUCTIONS on how to submit your work in the Course Documents section of Blackboard.

Be sure to read the SPECIFICATIONS carefully for all problems! And write comments!

---

1) simpsons.py

If you've taken Calculus II, you've probably encountered Simpson's rule for approximating definite integrals:

$$\int_a^b f(x)dx \approx \frac{\Delta x}{3}\left(1f(x_0) + 4f(x_1) + 2f(x_2) + 4f(x_3) + 2f(x_4) + \ldots + 2f(x_{N-2}) + 4f(x_{N-1}) + 1f(x_N)\right)$$

where

$$\Delta x = \frac{b-a}{N}$$

and

$$x_i = a + i\Delta x$$

Here, N is the number of pieces you break the interval $[a, b]$ into, which *must be even*; the more pieces, the better. Beware of the coefficients: they are 1 for the first term and the last term; all the rest alternate: 4,2,4,2,4,..., 4,2,4. (If you want to know why this is a good approximation, we can talk, or consult a textbook).

So, for example: if I want to approximate $\int_1^2 x^4 dx$ with $N = 6$ steps, I have:

$$\Delta x = \frac{2-1}{6} = \frac{1}{6}$$

$$x_0 = 1,\ x_1 = 1 + \frac{1}{6} = \frac{7}{6},\ x_2 = 1 + \frac{2}{6} = \frac{8}{6},\ x_3 = 1 + \frac{3}{6} = \frac{9}{6},\ x_4 = 1 + \frac{4}{6} = \frac{10}{6},\ x_5 = 1 + \frac{5}{6} = \frac{11}{6},\ x_6 = 1 + \frac{6}{6} = 2$$

$$\int_1^2 x^4 dx \approx \frac{1/6}{3}\left(1(1)^4 + 4\left(\frac{7}{6}\right)^4 + 2\left(\frac{8}{6}\right)^4 + 4\left(\frac{9}{6}\right)^4 + 2\left(\frac{10}{6}\right)^4 + 4\left(\frac{11}{6}\right)^4 + 1(2)^4\right) \approx 6.2001028807$$

The exact answer is of course $\frac{31}{5} = 6.2$, so we're pretty close.

I want you to create a program that asks the user to enter in A, B and an even integer N, and uses Simpson's rule with N intervals to approximate the value of

$$\int_A^B \frac{1}{\sqrt{2\pi}} e^{-x^2/2} dx.$$

For instance, if $A = -1.0$, $B = 1.0$ and $N = 10$, this should be 0.6826982201754326 – look carefully at your coefficients if you don't match at least the first 8 digits after the decimal place! Also, if $A = -2.0$, $B = 2.0$ and $N = 10000$, this should be 0.954500; if $A = -3.0$, $B = 3.0$ and $N = 10000$, this should be 0.997300. If those numbers sound vaguely familiar, that's because the given integral gives the area under the standard bell curve, between the z-values A and B.

You should write a Python function to represent the mathematical function $f(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2}$. (That seems pretty reasonable, right?) Warning – it's probably a bad idea to have this function do anything other take a value of x as input, and return $f(x)$.

Don't be intimidated by this problem – it's just a big sum! But do be careful about the details, like initializing your sum, using the right number of terms, and providing the right coefficient for each term.

And finally, **make sure you understand Simpson's rule well enough to perform some calculations by hand**. It can be tedious, sure, but if you can't perform Simpson's rule by hand, you'll never be able to code it.

Specifications: your program must

- ask the user for values of A, B and N, where N is an even integer (you may assume the user complies), and A and B should be `floats`.

- print an estimate for the value of $\int_A^B \frac{1}{\sqrt{2\pi}} e^{-x^2/2} dx$ using Simpson's rule with N steps.
- use a Python function to represent the function $f(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2}$.

***Challenge**: instead of asking the user for an integer N, ask the user to enter an error tolerance. Then, have your program determine an N on its own, so that the error between the Simpson's rule estimate and the true value of the integral is no more than the given error tolerance. There is an inequality related to Simpson's rule that can help you be sure about whether your N is sufficient.*

---

2) scrooge.py

Scrooge McDuck is letting you into his vault full of pennies, nickels, dimes and quarters. He will let you play the following game: you can pick 10 coins at random, and if they sum to at least a dollar, you win, and can keep your money! But if you lose, you don't get to keep the money, AND have to pay Scrooge a dollar. (Each coin you pick is equally likely to be a penny, nickel, dime, or quarter.)

Your job: using a simulation, estimate

1. the probability that you win, and
2. if you play this game over and over, the average amount you will win (in particular, does this game have a positive expected payoff?).

Here is a bit more direction: you should simulate 100,000 plays of this game. Each simulated play should involve picking 10 different random numbers – each one should be .01, .05, .10, or .25, with each value equally likely – which represent the 10 coins being chosen; these should be summed each time.

As you proceed through these 100,000 simulated plays, you should keep track of the number of simulations where you win: the probability of winning should be approximately equal to $\frac{\text{number of winning simulations}}{\text{total number of simulated games}}$. You should also keep track of the average amount of you've won per game (which could be negative).

After the simulations are complete, print out your program's estimates for both.

Specifications: your program must

- NOT ask the user for any input.
- print out an estimate of the probability that 10 values randomly chosen (independently and uniformly) from .01, .05, .10, and .25 sum to a value greater than or equal to 1.
- also print out an estimate of the average amount of winnings per game, where winnings are calculated by either the sum or -1, the latter when the sum of the 10 values is less than 1.
- **use 100,000 simulations with random numbers – no credit for theoretical solutions**. Your answers should **not** be exactly the true probability/average, and you should **not** even get the same answers each time you run it. If this point isn't obvious, please ask for clarification!

***Challenge**: do the same problem, except change it so that instead of all the coins being equally likely, each coin has a 52% chance of being a quarter, a 26% chance of being a dime, a 2% chance of being a nickel, and a 20% chance of being a penny. Then, instead of just printing out the average and probability, print out a histogram showing the frequencies of with which each winning value occurs. (I encourage you to do the first part of the challenge by thinking rather than Googling – the second part should probably be Googled.)*