Lab assignment #10: Random numbers, Monte Carlo integration

Instructor: Nicolas Grisouard (nicolas.grisouard@utoronto.ca)

Due Friday, November 27th 2020, 5 pm

First, try to sign up for room 1. If it is full, sign up for room 2 (and ask your partner to join you there if they are in room 1). Office hours will take place in room 1.

Room 1; office hour room Ahmed Rayyan (Marker, a.rayyan@mail.utoronto.ca),
URL: `https://gather.town/mIAKeWKElOnF4uL3/PHY407-AR`
PWD: `phy407ar`

Room 2 Mohamed Shaaban (m.shaaban@mail.utoronto.ca),
URL: `https://gather.town/xg1R1WVWH3wfgdCY/phy407-ms`
PWD: `phy407-2020-MS`

General Advice

- **Work with a partner!**
- Read this document and do its suggested readings to help with the pre-labs and labs.
- The topics of this lab revolve around basic methods to use random numbers, and Monte Carlo integration.
- Ask questions if you don't understand something in this background material: maybe we can explain things better, or maybe there are typos.
- Specific instructions regarding what to hand out are written for each question in the form:

  THIS IS WHAT IS REQUIRED IN THE QUESTION.

  Not all questions require a submission: some are only here to help you. When we do though, we are looking for "C^3" solutions, i.e., solutions that are **C**omplete, **C**lear and **C**oncise.
- An example of **C**larity: make sure to label all of your plot axes and include legends if you have more than one curve on a plot. Use fonts that are large enough. For example, when integrated into your report, the font size on your plots should visually be similar to, or larger than, the font size of the text in your report.
- Whether or not you are asked to hand in pseudocode, you **need** to strategize and pseudocode **before** you start coding. Writing code should be your last step, not your first step. Test your

code as you go, **not** when it is finished. The easiest way to test code is with `print()` statements. Print out values that you set or calculate to make sure they are what you think they are. Practice modularity. It is the concept of breaking up your code into pieces that are as independent as possible form each other. That way, if anything goes wrong, you can test each piece independently. One way to practice modularity is to define external functions for repetitive tasks. An external function is a piece of code that looks like this:

```
def MyFunc(argument):
    """A header that  explains the function
    INPUT:
    argument [float] is the angle in rad
    OUTPUT:
    res [float] is twice the argument"""
    res = 2.*argument
    return res
```

Place these functions in a separate file called e.g. `functions_labNN.py`, and call and use them in your answer files with:

```
import functions_labNN as fl  # make sure file is in same folder
ZehValyou = 4.
ZehDubble = fl.MyFunc(ZehValyou)
```

Physics Background

Diffusion-Limited Aggregation Most of the following info is from wikipedia.

Diffusion-limited aggregation (DLA) is the process whereby particles undergoing a random walk due to Brownian motion cluster together to form aggregates of such particles. This theory is applicable to aggregation in any system where diffusion is the primary means of transport in the system.

DLA is important in processes such as crystal growth (e.g., snowflakes), dielectric breakdown (e.g., lightning), electrodeposition (e.g., coating 1 metal with another by donating electrons to ions in a solution) and bacterial colony growth.

The clusters formed in DLA processes are called Brownian trees. See fig. 1 for cool pics.

Figure 1: Left: copper aggregate formed from a copper sulfate solution in an electrode position cell. Right: bacterial colony grown under starvation conditions.

Left: copper aggregate formed from a copper sulfate solution in an electrode position cell.
Right: bacterial colony grown under starvation conditions.

Hypersphere A hypersphere is the equivalent of a sphere in higher dimensions. In n dimensions, a hypersphere ("n-ball") of radius R with coordinate directions x_i, $i = 1, ...n$, is given by the points x_i such that

$$\sqrt{\sum_{i=1}^{n} x_i^2} = R. \tag{1}$$

Convince yourself that this makes sense for a 3D sphere (the one you are used to) as well as a 2D sphere (i.e., a circle).

There is actually an exact expression for the volume of an n-ball of radius R is

$$V_n(R) = \frac{R^n \pi^{n/2}}{\Gamma\left(\frac{n}{2}+1\right)}, \tag{2}$$

where Γ is the gamma function, i.e., an extension of the factorial operator to real numbers. For our purposes,

$$\forall x \in \mathbb{R}_{\geq 0},\ \Gamma(x+1) = x\Gamma(x) \quad \text{and} \quad \forall n \in \mathbb{N}^*,\ \Gamma(n) = (n-1)!. \tag{3}$$

Computational Background: Monte Carlo integration

Random number generators There is not one go-to python package for random number generation. We have seen a few already, and the following pages have a lot more info:

https://numpy.org/doc/stable/reference/random/index.html

https://docs.python.org/3/library/random.html

Monte Carlo integration Monte Carlo methods refer to methods that make use of random numbers in evaluating something. In this lab you will be using three Monte Carlo integration techniques:

- "hit or miss" Monte Carlo integration is described in section 10.2
- "mean value" Monte Carlo integration is described in section 10.2.1
- "importance sampling" Monte Carlo integration is described in section 10.2.3

All three methods have associated expressions for estimating errors.

Questions

1. **Brownian motion and diffusion limited aggregation (45% of the lab):**
   (a) Do exercise 10.3 on page 457.

HAND IN YOUR CODE AND A FIGURE SHOWING THE FULL TRAJECTORY OF YOUR RANDOM WALK.

Notes:

- *you can start from the sample program* `Brownian_start.py` *already provided.*
- *1 million steps will take forever. Try 5000 steps instead.*
- *The book refers to an animation method that uses* `vpython`. *Its use is not recommended, but you can use methods that you have used in previous labs instead. Or just plot the trajectory, as requested.*

(b) Do exercise 10.13 part (a) on pages 499-500 of the Newman text. Use 100 particles.

HAND IN YOUR CODE.

Notes:

- *You can start from the sample program* `DLA_start.py`, *along with the program you wrote for Q1a.*
- *Ignore all references to* `vpython`. *We just need to see a code that shows a good animation.*

(c) Do exercise 10.13 part (b) on pages 500-501. To modify your lattice into the 201×201 grid make take too much time (though you are free to do so), so just do 151×151 (which itself takes quite a while!). You do not need to make the anchored particles shown in different shades, but if you feel like it, knock yourself out.

HAND IN YOUR CODE AND A SNAPSHOT OF YOUR FINAL DLA DISTRIBUTION.

(d) You don't need to do part (c).

2. **Volume of a 10-dimensional Hypersphere (20% of the lab):** Do exercise 10.7 on pages 471-472 of the Newman text, and compute the value of the error associated with your calculation.

HAND IN YOUR CODE, THE VALUE FOR YOUR INTEGRAL AND THE VALUE FOR YOUR ERROR, ALONG WITH HOW YOU CALCULATED IT.

Hint: the volume of an n-dimensional hypercube of length L is L^n. You will want to use 1 million points to get a good estimate.

3. **Importance sampling (35% of the lab):**

(a) The integral

$$\int_0^1 \frac{x^{-1/2}}{1+e^x} dx \tag{4}$$

(eqn. 10.34 of the textbook) is an integral whose integrand has a divergence. Section 10.2.3 discusses using importance sampling to evaluate this integral. To learn about the regular mean value method and the importance sampling methods, evaluate the integral using both methods with 10,000 sample points, then repeat this calculation 100 times for each method. For the importance sampling approach, use the weighting function $w(x) = x^{-1/2}$, which removes the singularity. You will need to determine the

transformation needed to non-uniformly sample your region. The probability distribution you will be drawing from is

$$p(x) = \frac{1}{2\sqrt{x}} \tag{5}$$

Pages 458-459 of the text explain how to find the transformation for a given probability distribution, to help you understand where this $p(x)$ comes from. Make a histogram of the resulting integral values using 10 bins from 0.80 to 0.88. For an array of values (say it's called `I`), you do this with the command:

```
import matplotlib.pyplot as plt
# ...
plt.hist(I, 10, range=[0.8, 0.88])
plt.show()
```

Comment on what your histograms from the mean value method and the importance sampling method tell you about each method.

HAND CODE, PLOTS, EXPLANATORY NOTES.

(b) Importance sampling also helps evaluate rapidly varying integrands even if they aren't singular. For example, consider the integral

$$\int_0^{10} \exp(-2|x-5|)\,dx. \tag{6}$$

This is a function that is sharply peaked near $x = 5$, and we can easily find its exact value. If we calculate it using points uniformly distributed on the interval 0 <= x <= 10 we will be using a lot of points that don't contribute to the expectation value. To sample more points near $x = 5$ in order to increase the precision of the sampling method, choose an importance function that is Gaussian of the form

$$w(x) = \frac{1}{\sqrt{2\pi}} e^{-(x-5)^2/2} \tag{7}$$

that will lead you to draw from the normal distribution

$$p(x) = \frac{1}{\sqrt{2\pi}} e^{-(x-5)^2/2} \tag{8}$$

with mean of 5 and standard deviation of 1. You can use `numpy.random.normal` to do this.

Repeat part (a) using the mean value and importance sampling methods for this integral, and comment on the results.

HAND CODE, PLOTS, EXPLANATORY NOTES.