MA3: Multiprocessing, `git`, and Linux

This assignment primarily concerns parallel programming, higher order functions, and basic usage of `git` and Linux. The assignment consist of two major parts:

1. Parallel programming in Python
   - Use Monte Carlo to approximate π and the volume of a d-dimensional hypersphere
   - Comprehension, Lambda functions, `map`, `reduce`, `filter`, and `zip`
   - Plotting with `matplotlib`
   - Parallel programming in Python
2. `git` and Linux
   - Upload your solution to `git`
   - Test your parallel code on IT dept Linux machines

1. The exercises for the module can be found in the yellow frames below. There are **4 mandatory exercises** in this module.
2. When you complete them, present the solution to the last exercise to our teaching team.
3. Then, upload all scripts together with figures to the Studium.
4. For learning another way for accelerating your code, namely **integrating C++**, you are welcome to the first of 'Practice more' exercises in the Exam section.
5. As usual, each exercise has its own test file and there is tests.sh file to run all tests together. HighOrderFunctionChecker is needed for testing.
6. You will see that all results in this module have a random nature. That is, you may fail tests even your implementation is correct before you fix a random seed.

Important: You may collaborate with other students, but you must write and be able to explain your own code. You may not copy code neither from other students nor from the Internet except from the places explicitly pointed out in this lesson. Changing variable names and similar modifications does not count as writing your own code. Also, aiding somebody to cheat (for example, if somebody hands in your code as their own) is not permitted and will be reported.
Since the lesson tasks (MA's) are included as part of the examination, we are obliged to report failures to follow these rules.

1 Parallel programming in Python

In this part of the assignment you will first write a program that uses Monte Carlo to compute an **approximation of π**. Then you will modify this program so that the **computation is done in parallel**. Then, you will again modify the code so that you approximate the **volume of a d-dimensional hypersphere**.

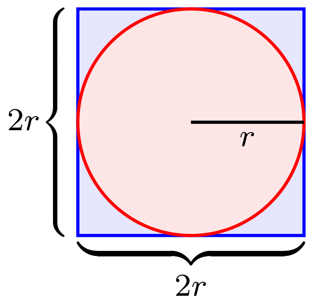
1.1 Monte Carlo approximation of π

Monte Carlo methods (`https://en.wikipedia.org/wiki/Monte_Carlo_method`) belong to a large and important class of methods and are used by many different algorithms to solve problems numerically; most importantly in optimization, integration, and probability theory.

One utilizes random tests to approximate the true underlying property. This is typically very useful and important for multidimensional problems where, for example, integration in all dimensions is impossible to do exactly with other conventional methods.

In this part of the assignment you will write a program that uses Monte Carlo to estimate the constant $\pi = 3.14159\ldots$

In Figure 1 is shown, in red, a circle with radius r and area $A_c = \pi r^2$. It is placed in a blue square, with sides $2r$, that has area $A_s = (2r)^2 = 4r^2$.



Figur 1: Circle with radius r inscribes in a square with sides $2r$.

If you divide the area A_c with A_s, you have,

$$\frac{A_c}{A_s} = \frac{\pi r^2}{4r^2} = \frac{\pi}{4},$$

or

$$\pi = 4\frac{A_c}{A_s}.$$

Now assume that $r = 1$ and that the center of the circle is at the origin, (0,0). Now create n uniformly distributed random coordinates $(x, y) \in [-1, 1] \times [-1, 1]$ in the square, and sort these n_c points that lie inside the circle and n_s that lie outside the circle $(n = n_c + n_s)$. One can then approximate π with

$$\pi \approx 4\frac{n_c}{n}.$$

because the ratio n_c/n will approximate A_c/A_k, and when $n \to \infty$ we will get equivalence.

In Figure 2 is shown four examples, for $n = \{50, 100, 200, 400\}$. The approximations of π are then $\pi \approx \{2.8, 3.16, 2.96, 2.97\}$. Note that it can happen that one gets a better approximation of π for an n smaller than another, but as $n \to \infty$ the approximation will be arbitrarily close to π.

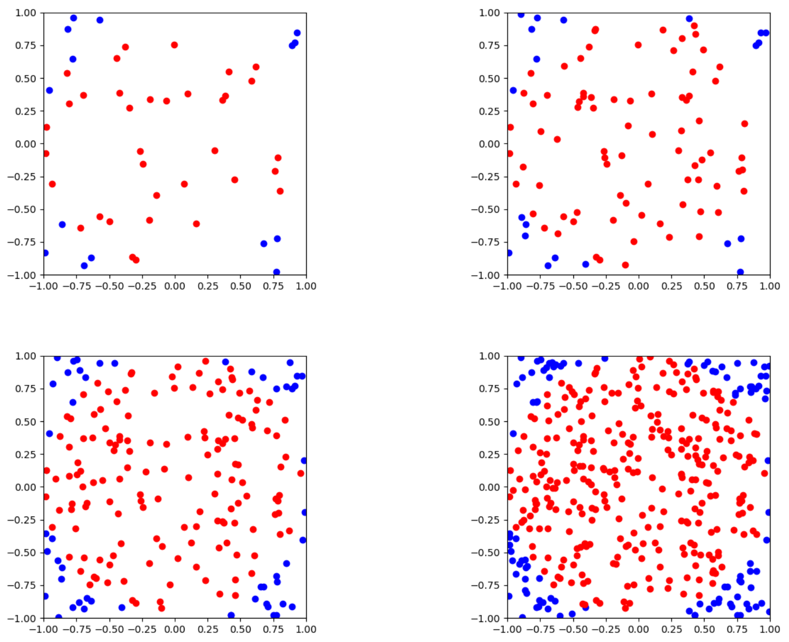
How do you decide if a point is inside the circle? In this simple case, when $r = 1$ and the center is in the origin, one can simply test if $x^2 + y^2 \leq 1$ (equivalent to $\sqrt{x^2 + y^2} \leq 1$). If one had a more complicated object, one can divide the domain into a mesh (for example a square grid or a triangulation) where each cell is either inside or outside the object.

Now write a program that has n, the number of random points that should be generated, as an argument and produces the following output:

1. Print the number of points n.

2. Print and return the approximation of $\pi \approx 4n_c/n$.

3. Produce a `png` file that shows all points inside the circle as red dots and points outside the circle as blue dots (like in Figure 2).



Figur 2: Approximation of π using Monte Carlo. Top: $n = 50, \pi \approx 2.8$ and $n = 100, \pi \approx 3.16$. Bottom: $n = 200, \pi \approx 2.96$ and $n = 400, \pi \approx 2.97$.

Tip: Use `random.uniform()` or `random.random()` to create the n random points. Also, use `matplotlib.pyplot` to create the figures (Note, do not take screenshots write the figure to disk.)

Exercise 1 Compute approximations of π for $n = \{1000, 10000, 100000\}$. Produce figures for these values of n.

1.2 Approximate the volume of a d-dimensional hypersphere

- In this part you will use higher order functions. See separate pdf in Studium (Module M3) or lecture notes for an introduction. You must use at least three of the concepts/ functionalities
  - Comprehension
  - Lambda function
  - `map()`
  - `functools.reduce()`
  - `filter()`
  - `zip()`

in this part of the assignment.

In this part of MA3 you should write a program that uses a Monte Carlo approximation of the volume $V_d(r)$ for a d-dimensional hypersphere radius r. A hypersphere is a generalization of the circle and sphere to higher dimensions than two and three. The method `sphere_volume()` should have two arguments, n which is the number of random points to generate, and d which is the number of dimensions. One can assume $r = 1$ and the center of the hypersphere is the origin. However, one may also include r as the third parameter.

The test to see if a point is inside the hypersphere is in this case,

$$x_1^2 + x_2^2 + \ldots + x_d^2 \leq 1.$$

The furmula for the volume $V_d(r)$ (`https://en.wikipedia.org/wiki/Volume_of_an_n-ball`), to verify your code, is

$$V_d(r) = \frac{\pi^{d/2}}{\Gamma(d/2+1)} r^d, \qquad V_d(1) = \frac{\pi^{d/2}}{\Gamma(d/2+1)}$$

where Γ is the Gamma function (`https://en.wikipedia.org/wiki/Gamma_function`). The Gamma function already exists in Python (in the module `math`) and you do not have to implement it yourselves. To solve this assignment you may use `numpy` if you want.

Exercise 2: Write a function `sphere_volume(n,d)` that computes the approximation for $V_d(1)$. Use at least three higher order functions. Write a method `hypersphere_exact` that calculates the exact volume. Compare (print) the results produced by the exact formula and the approximation for $(n, d) = (100000, 2)$ and $(n, d) = (100000, 11)$.

1.3 Parallel programming in Python

Read the separate pdf in Studium (Module M3) for an introduction to parallel programming in Python.

Exercise 3:

- Make a loop of 10 iterations for `sphere_volume`, with values $(n, d) = (10^5, 11)$ in each iteration. Calculate the average of the functions' outputs. Measure the time it takes to execute the code, e.g. with `time.perf_counter`.
- Write a function `sphere_volume_parallel1` where the loop is parallelized using `futures.ProcessPoolExecutor` (basically you divide iterations among parallel processes) and the average value is returned. Time the new code. Does it work faster?

The last exercise is similar to the previous. In Exercise 3, the same code is executed several times. It can be helpful if outputs of a program are random and mean with standard deviation of the results are required. In Exercise 4, execution of one code needs to be optimized.

Note that you can modify the file directly in linux machines using an editor e.g. nano (write `nano MA3.py` in the terminal) or use graphical interface via VSCode. If you have updated the git repository and want to update the files in the server, use `git pull`.

Exercise 4:

- Run the function `sphere_volume` with values $(n, d) = (10^6, 11)$ in a sequential manner. Here is no loop any more, we just want to estimate the volume accurately by setting a large value of n.
- Write `sphere_volume_parallel2` that speeds up `sphere_volume` using parallel programming. Here you divide data among parallel processes. So, you exhibit yourselves to two ways of parallelism: loop (ex 3) and data (ex 4). Compare the time between the two approaches (sequential and parallel).
- Run the code on the Linux servers provided by IT Department. See more information about the servers below. How much faster it was to run a parallel code on servers than on your local computer?

2 `git` and a bit of `Linux`

The last exercises should be tested on the Linux machines of the IT department machines. We expect you to use `git` there in order to get your files and to run the code on these machines.

Linux is a free UNIX-like operating system (`https://en.wikipedia.org/wiki/Linux`) that is extensively used and you will learn how to use some basic features. This is useful for example if you ever need to setup servers (databases, web servers, etc.) or run programs in the cloud (with virtual machines), which is important in data engineering and data science.

2.1 IT department Linux machines

Choose one of the following Linux machines:

```
arrhenius.it.uu.se
atterbom.it.uu.se
cronstedt.it.uu.se
enequist.it.uu.se
```

Note that you can freely switch between the machines between different logins, all files are stored on a central filesystem and is available on all machines. You can also login multiple sessions at the same time. **NB: a/ these machines are old and can be slow; b/ if you have issues connecting, let us know asap.**

The machines run the Ubuntu distribution of Linux and you do not have administrator (root) access, so you can't (and don't need to) use package manager to install any programs on the system.

To access a Linux machine you need to use a terminal program on your own machine (commands are run by writing them). You will use SSH (Secure Shell) to login to the Linux machines.

Some terminal programs (the recommended ones are highlighted in red), and you need to use one of them on your local computer.

Your OS	
Linux	xterm (preinstalled)
	(https://www.tecmint.com/linux-terminal-emulators/)
macOS	Terminal (preinstalled in /Applications/Utilities)
	iTerm2 (https://iterm2.com/)
Windows	If Windows 10 or later, the builtin Powershell
	WSL2 (search Microsoft Store)
	PuTTY (https://www.putty.org/)
	bitwise (https://www.bitvise.com/)

To login write (do not write the `$`, when it is in the beginning of a line it only indicates that you have a terminal started) in the terminal:

```
$ ssh abcde123@arrhenius.it.uu.se
```

Here `abcde123` is you UU account (same as you use to login to STUDIUM) and you should use you password A (the same password you use to login to STUDIUM, not 'A') when asked for it (once you press return/enter). `arrhenius.it.uu.se` is the hostname you chose in the list of linux machines. If you use PuTTY or bitwise you will have to enter this information in a GUI (username: `abcde123` and hostname: `arrhenius.it.uu.se`).

Here is a collection of some useful linux commands:

`ls`	List files in current directory (`ls -la` to show more information)
`pwd`	Show where you are (print working directory)
`cd abc`	Go into a directory named `abc`
`cd ..`	Go up one step in the file system (e.g.,`/` is the root directory, `/home` contains home directories and `/home/abcde123` is the home directory of user `abcde123`)
`cd`	Go to your home directory (where your personal files are, and this is where you are when you first log in)
`mkdir abc`	Create a directory called `abc`
`rm -fr abc`	Remove a file or directory called `abc`
`nano hej.txt`	Edit a file `hej.txt` with the editor `nano` (there are many different editors). Leave nano with `ctrl-x`
`python3 test.py`	Run the Python code in the file `test.py` with Python 3.x. Changing to `python test.py` would invoke Python 2.x on these linux machines (configurable so might be different elsewhere).

A tutorial with an introduction to linux

https://maker.pro/linux/tutorial/basic-linux-commands-for-beginners

Python is installed on the Linux machines, however `matplotlib` is not installed. You need to install `matplotlib` once, and it is easiest to do with:

```
$ pip install matplotlib
```

Note: you only do this once on the Linux machine.

To exectute a Python file `hello.py`, you run it as

```
$ python hello.py
```

You may specify what python version to use by writing python3.9 or similar (better not to use an older version).

If you want to try this assignment locally on your home computer you will need a C++ compiler (for example, on the Linux machines you will use `g++` which is part of `gcc`). On Windows, if you installed WSL2 `gcc` is installed by default (you could also use `https://visualstudio.microsoft.com/`). On macOS you install XCode through the App Store.

2.2 `git`

Get an account at `www.github.com`. You can follow the instructions below to have your private repository OR can joint `github classroom` and have your private repository there. I think `github classroom` is easier, see more here Section 2.3. However, instructions from below are still useful.

Assume that you have created a `git` repository at your `git` provider and it is called `prog 2`. Now clone it to your local computer. We also assume that files on your local computer are in `prog2/MA3/MA3_files` folder. Now add, commit, and push the files to the repository of your provider. Either do this with some specific `git` client, VSCode, or in the terminal,
In your local computer:

1. Navigate to your project directory by `cd path_name`
2. Initialize git to set up your repository

   `git init`
3. Add a folder or a file to the repository

   `git add 'foldername'`
4. Commit the files i.e.save a snapshot of current changes

   `git commit -m 'Your commit message here'`
5. Set up a remote repository i.e. GitHub

   `git remote add origin https://github.com/yourusername/your-repo.git`
6. Push the files to the remote repository

   `git push`

In a server:

Once you are logged into a Linux machine write

```
git clone https://github.com/yourusername/your-repo.git
```

where the url is modified to fit your `git` server provider, username, and repository name.

Note: If you use Github to handle your repository, it is required to use so-called *Personal access token* to login when you want to clone the repository (it wil not work with the password you use to login to www.github.com). Here is a description how to create an access token, and in point 8 you just check "repo".
Then you use the created access token instead of a password when `git` asks for it. To not be required to use it every time you access the repository, you can store it on the Linux machine, this can be achieved by doing the following once you have cloned the repository:

```
sveek137@atterbom:~/prog2$ git config credential.helper store
sveek137@atterbom:~/prog2$ git pull
Username for 'https://github.com': riakymch
Password for 'https://riakymch@github.com':
Already up to date.
```

Now write (modify paths if they are different for you)

```
$ cd prog2/MA3/MA3_files
$ ls
```

and you should now see the files described in the previous section.

To test your code run (it is assumed below that `approximate_pi` has print statements):

```
$ python3 MA3.py
estimated pi for 1000 dots = 3.232
estimated pi for 10000 dots = 3.1344
estimated pi for 100000 dots = 3.14596
```

You can also test the following: Add `#!/usr/bin/python3` in the beginning of your file. Read more about Shebang https://realpython.com/python-shebang/.

Now you can run python programs from terminal without specifying an interpreter. Then, in terminal:

```
$ ./MA3.py
$ ls -la
$ chmod 755 MA3.py
$ ls -la
$ ./MA3.py
```

The first time you run `$ ./MA3.py` you get an error message. Note the difference between the first and second time you execute `$ ls -la` for the file `MA3.py`. The command `chmod` changes the rights of the file, and 755 makes the file executable. When you have done this once you can execute the code by just typing `$ ./MA3.py` (until you change the rights of the file to something else).

2.3 `github classroom`

Instructions for submission to `github classroom`:

1. Click this link https://classroom.github.com/a/yQngq2Gq and login with your github account. If you do not have one, you need to create it. Then, you will have

an individual repository created for you. Save that link somewhere. The repository contains test files, so you just need to add your solutions in `MA3.py`.

2. Clone the repository to your local machine `git clone repo_name`

3. Place your solutions, in a single file `MA3.py`, in the folder of the cloned repo. This can be done from the command line using `cp file dest_folder`. For more advanced users, you may explore an option of duplicating github repositories if you have created one. Please keep all repositories as private.

4. Add your files to the repository and commit them. This can be done by

   (a) one command `git commit -m ‘write a meaningful message’ -a`

   (b) by two commands `git add file_name` and then `git commit -m ‘write a meaningful message’`. I prefer the second option since it is more clear.

5. Push your solution to the global repository `git push`