Scientific Computation

1 Displaying Numbers

1.1 Monte-Carlo method

Your first step here is to create a new project named `SIP_LAB9` in PyCharm Edu.

We will use the Monte-Carlo method to compute π. The principle of this method is to place points at random in a square, and count how many of them land in the inscribed circle. The probability for a given point to be in the circle is $S_{circle}/S_{square} = \pi/4$

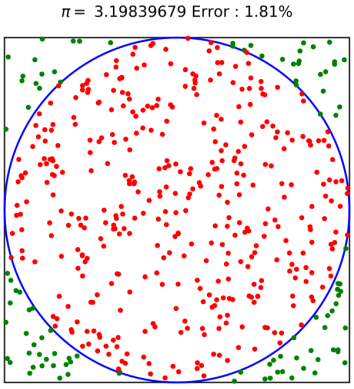
Question 1 Computing the Monte-Carlo method

Implement the Monte-Carlo method in Python. What is the order of magnitude of the error for 1000 points? 100 000?

Question 2 Displaying the Monte-Carlo method

This code will setup the figure for you.

```
1  import numpy as np
2  import matplotlib.pyplot as plt
3  import matplotlib.patches as patch
4  import random as rd
5  from math import pi
6
7  # ----Plotting the circle and the rectangle-------
8  x = np.linspace(-1, 1, 1000)
9  y = np.sqrt(1-x**2)  # Upper half of circle
10 neg_y = -y  # Lower half of circle
11 fig, ax = plt.subplots()
12
13 # painting circle boundary
14 ax.plot(x, y, 'blue')
15 ax.plot(x, neg_y, 'blue')
16
17 # painting square boundary
18 rect = patch.Rectangle((-1, -1), 2, 2, linewidth=1, fill=False)
19 ax.add_patch(rect)
20
21 # removing th axis
22 ax.axis('equal')
23 ax.axis('off')
```



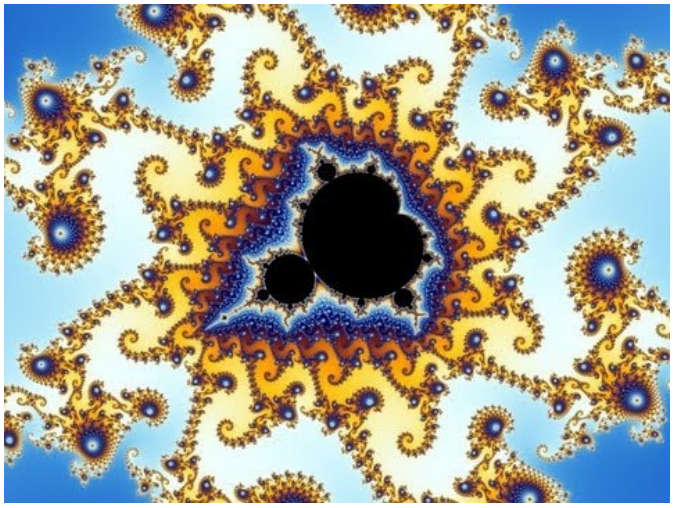
Using matplotlib and the code shown and the logic of the last question, display the circle, the points and the approximation.

Question 3 Animation (Optional)

With the command `plt.pause(seconds)` we can briefly stop the plot. Use this to create an animation of the Monte-Carlo Method

1.2 Mandelbrot's set

The Mandelbrot set is the set of complex numbers c for which the function $f_c(z) = z^2 + c$ does not diverge when iterated from $z = 0$, i.e., for which the sequence $f_c(0)$ $f_c(0)$, $f_c(f_c(0))$, etc., remains bounded in absolute value.



1.2.1 Complex numbers in Python

For this exercise we will be using complex numbers

To know : Complex Numbers

Quick Reminder on complex numbers:

- A complex number is of the form $z = x + iy$ with $i^2 = -1$ with x, y real number
- Real Part: $Re(z) = x$ / Imaginary Part: $Im(z) = y$
- $|z|$ module ("size" of the complex number): $|z| = \sqrt{x^2 + y^2}$
- $\overline{z}$ conjugate: $\overline{z} = x - iy$ / $z + \overline{z} = 2Re(z)$ / $z - \overline{z} = i * 2Im(z)$ / $z * \overline{z} = |z|^2$

In Python complex numbers are a specific type of variables. You can create them either by declaring them explicitly or by specifying x and y

```
>>> 3 + 4j
(3+4j)
>>> complex(3,4)
(3+4j)
>> 3 + 0j
(3+0j)
>>> complex(3,0)
(3+0j)
```

Remarks:

- Python uses j instead of i.
- Notice also that to create a real number as a complex explicitly you have to specify $0j$

You can simply operate on them like other numbers

```
>>> 1j
1j
>>> 1j + 1j
2j
>>> 1j * 1j
(-1+0j)
```

Even if the value of an operation is a real it will still be a complex (hence the $0j$ in the output)
Complex numbers come with properties

```
>>> z = 3 + 4j
>>> z.real
3
>>> z.imag
4
>>> z.conjugate()
(3 - 4j)
```

Question 4 Module of a complex number

Create two functions:

- module_real_and_img(z) that uses the imaginary and real parts of a complex to compute the module
- module_conjug(z) that uses the the conjugate to compute the module

Hint: the math module provides a `math.sqrt()` function.

Question 5 Testing for a point

Practically speaking since we are squaring complex numbers we will assume that when $|f_c(z)| > 2$ we say that f_c will diverge.

Plotting a Mandelbrot's set will be done for each complex $c = x + iy$ computing the value n_c that is the first n such that $|f_c^n(0)| > 2$.

Since this could take an infinite number of steps (namely for c such that $f_c^n(0)$ converges), we add a limit n_{max} of iterations, as follows:

$$n_c = \begin{cases} n_{max} & if \quad n_c \geq n_{max} \\ n_c & if \quad n_c \leq n_{max} \end{cases}$$

Create a function $mandelbrot(c, n\_max)$ that for a given point c will return n_z
(Remark: if it helps you can create the function $mandelfunc(c, z)$ that computes $f_c(z)$

Question 6 Plotting the Mandelbrot's set

Now that we have a function that computes the Mandelbrot's set for one point, we are going to plot the whole mandelbrot set. For that purpose we are going to use numpy arrays and matrices.

We will use $np.linspace(x, y, n)$ that creates an array of n values between x and y evenly spaced (interval of $(x-y)/n$ between each value)
Also, we will use $np.empty((x, y))$ that creates a matrix of size x and y. Coefficients are not initialized.

Create a function $mandelbrot\_set(xmin, xmax, ymin, ymax, xnum, ynum, maxiter)$ that creates two arrays numpy and one matrix:

- x: an array of $xnum$ evenly spaced values ranging from $xmin$ to $xmax$.
- y: an array of $ynum$ evenly spaced values ranging from $ymin$ to $ymax$.
- z: a matrix of size $xnum * ynum$ where $z_{j,k} = n_c$ with $c = x[j] + i * y[k]$.

Question 7 Displaying the set

Plot the Mandelbrot set with the following values:

```
xmin, xmax, xnum = -2.25, +0.75, 3000//2
ymin, ymax, ynum = -1.25, +1.25, 2500//2
```

2 Ranking the Web

2.1 Random Walk

A random walk is a random process describing a path constructed by a succession on random steps, each step only depending on the present state.

One of the most used example of this kind of process is the PageRank algorithm.

2.2 Problem

Let's imagine that we have n "Web Pages" that we want to rank. We will call p_{ij} the probability that if someone is at page j, he will go to page i because he has clicked on a link or something similar.

Thus, we have :

$$\forall j \in [\![0, n-1]\!], \sum_i p_{ij} = 1$$

We will consider that we have the same probability to start from any page : $\frac{1}{n}$

We have the following transition matrix :

$$M = \begin{pmatrix} p_{0,0} & p_{0,1} & \cdots & p_{0,n-1} \\ \vdots & \vdots & \ddots & \vdots \\ p_{n-1,0} & p_{n-1,1} & \cdots & p_{n-1,n-1} \end{pmatrix}$$

We search the "PageRank" of our set of pages, which is the state s representing the probability to be at any of the said pages after a long time. Thus, s such that :

$$Ms = s$$

To that end we define :

$$\begin{cases} s_0 = \begin{pmatrix} \frac{1}{n} & \cdots \frac{1}{n} \end{pmatrix}^\intercal \\ s_{n+1} = Ms_n \end{cases}$$

We will consider that s_n is close enough to s if :

$$|s_{n+1} - s_n| \leq \epsilon$$

2.3 Creating the data set

Question 8 Random sum of value 1

Write a function that takes in an argument n and returns a random subdivision of 1 of size n , i.e. :

$$(a_0, a_1, \cdots, a_{n-1}), \sum_i a_i = 1$$

Question 9 Creating our ranking matrix

Write a program that takes in an argument n and returns a random transition matrix M of size n (the matrix must verify the invariant that the values in each column sum up to 1; that kind of matrix is called a left stochastic matrix) using the function you defined above.

For that you will use the matrix structure from numpy:

```
>>> import numpy as np
>>> m = np.matrix([[1,2,3],[4,5,6],[7,8,9]])
>>> m
matrix([[1, 2, 3],
        [4, 5, 6],
        [7, 8, 9]])
>>> m.transpose()
matrix([[1, 4, 7],
        [2, 5, 8],
        [3, 6, 9]])

# We can calculate the product naively using this structure (power also)
>>> m*m
matrix([[ 30,  36,  42],
        [ 66,  81,  96],
        [102, 126, 150]])
>>> m**2
matrix([[ 30,  36,  42],
        [ 66,  81,  96],
        [102, 126, 150]])

# And if we create a vector using the same structure
>>> v = np.matrix([1,2,3]).transpose()
>>> m*v
matrix([[14],
        [32],
        [50]])
```

2.4 Convergence and Norm

Question 10 Convergence

Given a precision ϵ, what condition can we write to say that our sequence is a good enough approximation of s?

So now we have the problem of calculating the norm of a matrix using Python. Fortunately for us, the linalg subpackage of numpy provides us with just the right function :

```
>>> import numpy as np
>>> import numpy.linalg as la
```

```
>>> m = np.matrix([[1,2,3],[4,5,6],[7,8,9]])
>>> la.norm(m) # Frobenius norm
16.881943016134134
>>> la.norm(m, 2) # norm 2
16.84810335261421
>>> la.norm(m, np.inf) # infinity norm
24
```

Question 11 PageRank

Now write a program that, given the number of pages analyzed and the epsilon, will return the "PageRank" of the pages.

3 Prey-Predator system

3.1 Introduction

The prey-predator system (more precisely here the Lotka-Volterra system) is an equation system modeling the evolution of the population of two species interacting with one another: one as a prey, the other as a predator.

This model is based on an equation system of two non-linear differential equations :

$$\begin{cases} \frac{dx}{dt} = x(\alpha - \beta y) \\ \frac{dy}{dt} = y(\delta x - \gamma) \end{cases}$$

Where :

- x is the prey population
- y is the predator population
- $\frac{dx}{dt}$ and $\frac{dy}{dt}$ are the instant population growth rate
- α, β, δ and γ are positive real numbers describing the interaction

To begin with, let's examine these equation to understand how they work. We will suppose that the populations are separated (they do not interact with each other), which will lead to removing the y factor in the first equation and x in the second one.

Question 12 First, some math...

Resolve the system under these hypotheses. What can you infer about the parameters α and γ ? And about the β and δ parameters ?

3.2 Application

We will suppose two populations :

We will assume that every time an alien kills a human, he will first lay one egg inside of it that will hatch and give birth to a new alien, thus we have $\beta = \delta$.


Figure 1: Aliens and Humans

The iterative way to model the behaviour of $x(t)$ is to take a time subdivision $(t_n)_{n\in\mathbb{N}}$ and to model the function by a sequence $(x_n)_{n\in\mathbb{N}}$.

Thus, we can use the explicit euler scheme to write :

$$\begin{cases} \dfrac{dx}{dt} = \dfrac{x_{n+1}-x_n}{t_{n+1}-t_n} = f(x_n, y_n, \alpha, \beta) \\ \dfrac{dy}{dt} = \dfrac{y_{n+1}-y_n}{t_{n+1}-t_n} = f(y_n, x_n, -\gamma, -\delta) \end{cases} \quad \text{where } f(X, Y, A, B) = X(A - BY)$$

We will consider that $\forall n \in \mathbb{N}, t_{n+1} - t_n = h$ known constant.

Question 13 Our growth function

Write the function f defined above.

Question 14 Let's start growing

Using f, write a function *next_step* that returns z_{n+1} using the problems parameters (here z is either x or y, so you must write a function working for both).

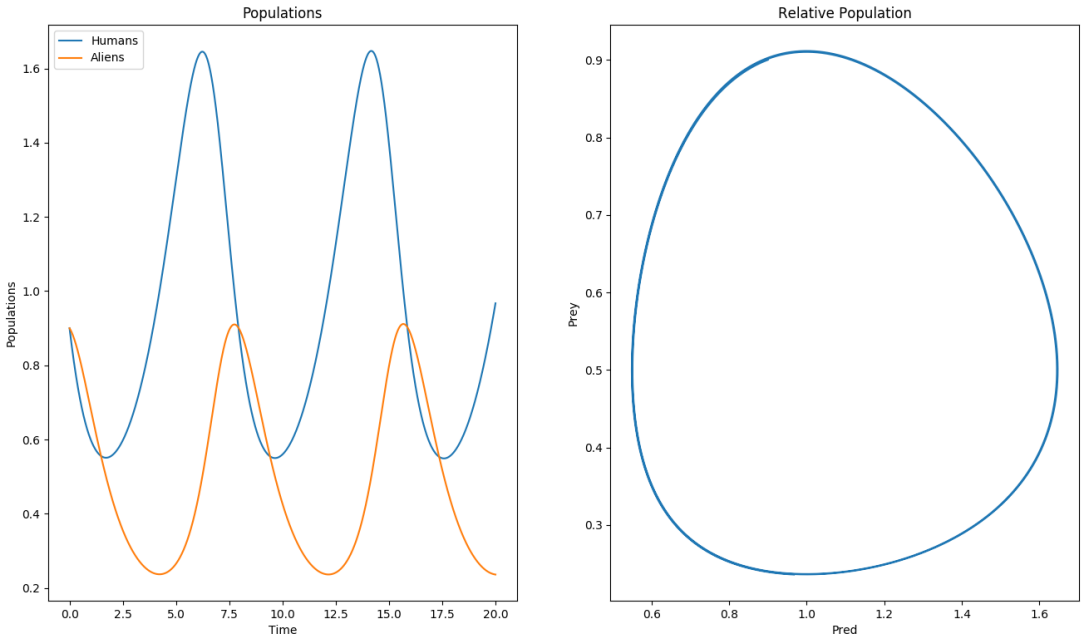
Question 15 Let the hunt begin...

Write a function that takes in argument $(x_0, y_0, \alpha, \beta, \delta, \gamma, h, n)$ with n the number of steps and returns the two lists containing $(x_n)_{n\in\mathbb{N}}$ and $(y_n)_{n\in\mathbb{N}}$.

Question 16 Will we survive ?

Define a function *display_populations* that will display the evolution of the population according to the time, andthe relative evolution of population in the same figure.
You can take the following values as example:

- $x_0 = 0.9, y_0 = 0.9$
- $\alpha = 2/3, \beta = 4/3, \gamma = \delta = 1$
- timestep = 0.001, time_max = 20, n = (time_max/timestep)


Populations
Humans
Aliens
Populations
Time
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
1.6
Relative Population
Prey
Pred
0.3
0.4
0.5
0.6
0.7
0.8
0.9
0.6
0.8
1.0
1.2
1.4
1.6