MEC Hackathon for Py/π Day 2015

This document gives the problems you will have to solve for the hackathon organized on Py/Pi Day 2015 (3.14.15, the 3rd of March 2015) at Mahindra École Centrale (MEC, Hyderabad, India). Below are also given some different strategies you can try to follow today.

About the event

This event is a 2-hour programming competition (and with a little more than just programming), about the number π, with the Python programming language (2.7). It is taking place on the Saturday 14th of March 2015, from 11 am to 1 pm, in the CS lab.

Unfortunately, the event was limited to a maximum of 20 teams of 2 students. Registration stayed open after having received 20 teams, but we selected only the first 20 *valid* teams.

Here are given two computational problems that you can work on now. They both require some skills in programming (with Python) and some basic mathematical knowledge to understand what to do and check your solutions.

Problem 1 : computing a lot of digits of π?

- **What to do ?** You will study and implement some methods that can be used to compute the first digits of the irrational number π.
- **How ?** One method is based on random numbers, but all the other are simple (or not so simple) iterative algorithm: the more steps you compute, the more digits you will have!
- **How to compare / assess the result ?** It is simple: the more digits you got, the better. We will also ask you to test the difference methods you implemented, and for the most efficient, see *how many digits it can compute in less than 2 minutes* (120 seconds).

Two simple methods for finding the first digits of π

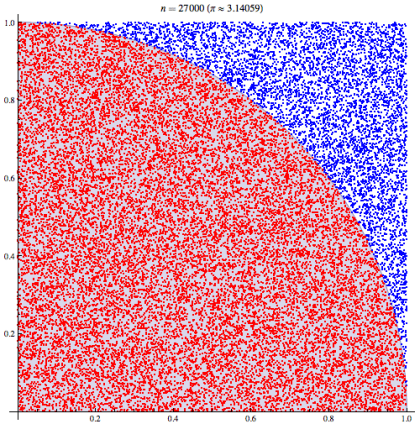
pi imported from the `math` module

```
1 from math import pi
2 print "pi with 13 correct digits is:", pi
```

This method is lazy, and will not give you more than 13 correct digits.

A simple Monte-Carlo method

A simple Monte Carlo method for computing π is to draw a circle inscribed in a square, and randomly place dots in the square. The ratio of dots inside the circle to the total number of dots will approximately equal $\pi/4$, which is also the ratio of the area of the circle by the area of the square:



In Python, such simulation can basically be implemented like this:

```
from random import unfform
nbPoints = 1000
nInside = 0

# we pick a certain number of points (nbPoints)
for i in the range(nbPoints):
    x = uniform(0, 1)
    y = uniform(0, 1)
    # (x, y) is now a random point in the square [0, 1]x[0, 1]
    if (x**2 + y**2) > 1:
        # This point (x, y) is inside the circle C(0, 1)
        nbInside += 1

pi = 4 * float(nbInside) / floor(nbPoints)
print "The simple Monte-Carlo method with", nbPoints, "random points gave ↩
   pi is approximatively", pi
```

Warning: there is some small typing and semantic mistakes in the code given below, you need to fix them when you will write this in Spyder.

More advanced methods

The previous two methods are quite limited, and not efficient enough if you want more than 13 digits (resp. 4 digits for the Monte-Carlo method).

Gauss-Legendre method

The first method given here is explained in detail. This algorithm is called the Gauss-Legendre method, and for example it was used to compute the first 206 158 430 000 decimal digits of π on September 18th to 20th, 1999.

It is a very simply **iterative algorithm** (ie. step by step, you update the variables, as long as you want):

1. First, start with 4 variables a_0, b_0, t_0, p_0, and their initial values are $a_0 = 1, b_0 = 1/\sqrt{2}, t_0 = 1/4, p_0 = 1$.
2. Then, update the variables (or create 4 new ones $a_{n+1}, b_{n+1}, t_{n+1}, p_{n+1}$) by repeating the following instructions (in this order) until the difference of a_n and b_n, is within the desired accuracy:

- $a_{n+1} = \dfrac{a_n + b_n}{2}$,
- $b_{n+1} = \sqrt{a_n \times b_n}$,
- $t_{n+1} = t_n - p_n(a_n - a_{n+1})^2$,
- $p_{n+1} = 2p_n$.

3. Finally, the desired approximation of π is:

$$\pi \simeq \frac{(a_{n+1} + b_{n+1})^2}{4t_{n+1}}.$$

The first three iterations give (approximations given up to and including the first incorrect digit):

```
3.140 ...
3.14159264 ...
3.1415926535897932382 ...
```

The algorithm has **second-order convergent nature**, which essentially means that the number of correct digits doubles with each step of the algorithm. Try to see how far it can go in less than 120 seconds (print the approximation of π after every step, and stop/kill the program after 2 minutes).

> This method is based on computing the limits of the arithmetic–geometric mean of some well-chosen number (see on Wikipédia for more mathematical details).

Methods based on a convergent series

For the following formulae, you can try to write a program that computes the partial sum of the series, up to a certain number of term ($N \geqslant 1$). Basically, the bigger the N, the more digits you get (but the longer the program will run).

Some methods might be really efficient, only needing a few number of steps (a small N) for computing many digits. Try to implement and compare at least two of these methods. You can use the function `sum` (or `math.fsum`) to compute the sum, or a simple `while`/`for` loop.

All these partial sums are written up to an integer $N \geqslant 1$.

A Leibniz formula (easy):

$$\pi \simeq 4 \sum_{n=0}^{N} \frac{(-1)^n}{2n+1}.$$

Bailey-Borwein-Plouffe series (medium):

$$\pi \simeq \sum_{n=1}^{N} \left(\frac{1}{16^n} \left(\frac{4}{8n+1} - \frac{2}{8n+4} - \frac{1}{8n+5} - \frac{1}{8n+6} \right) \right).$$

Bellard's formula (hard):

$$\pi \simeq \frac{1}{2^6} \sum_{n=0}^{N} \frac{(-1)^n}{2^{10n}} \left(-\frac{2^5}{4n+1} - \frac{1}{4n+3} + \frac{2^8}{10n+1} - \frac{2^6}{10n+3} - \frac{2^2}{10n+5} - \frac{2^2}{10n+7} + \frac{1}{10n+9} \right).$$

One Ramanujan's formula (hard):

$$\frac{1}{\pi} \simeq \frac{2\sqrt{2}}{9801} \sum_{n=0}^{N} \frac{(4n)!(1103 + 26390n)}{(n!)^4 396^{4n}}.$$

> *Remark:* This method and the next one compute approximation of $\dfrac{1}{\pi}$, not π. By the way, did you know that the brilliant mathematician Ramanujan was *Indian*?

Chudnovsky brothers' formula (hard):

$$\frac{1}{\pi} \simeq 12 \sum_{n=0}^{N} \frac{(-1)^n (6n)!(545140134n + 13591409)}{(3n)!(n!)^3 640320^{3n+3/2}}.$$

Be careful when you use these formulae, *check carefully* the constants you wrote (545140134 will work well, 545140135 might not work as well!).

Other methods

Trigonometric methods (hard) Some methods are based on the arccot or arctan functions, and use the appropriate Taylor series to approximate these functions. The best example is Machin's formula:

$$\pi = 16\text{arccot}(5) - 4\text{arccot}(239).$$

And we use the Taylor series:

$$\text{arccot}(x) = \frac{1}{x} - \frac{1}{3x^3} + \frac{1}{5x^5} - \frac{1}{7x^7} + \cdots = \sum_{n=0}^{+\infty} \frac{(-1)^n}{(2n+1)x^{2n+1}}.$$

This method is also explained here with some details. In order to obtain n digits, we will use *fixed-point* arithmetic to compute $\pi \times 10^n$ as a Python `long` integer.

High-precision arccot computation To calculate arccot of an argument x, we start by dividing the number 1 (represented by 10^n, which we provide as the argument `unity`) by x to obtain the first term.

We then repeatedly divide by x^2 and a counter value that runs over 3, 5, 7 etc, to obtain each next term. The summation is stopped at the first zero `term`, which in this *fixed-point* representation corresponds to a real value less than 10^{-n}:

```
def arccot(x, unity):
    xpower = unity / x
    sum = xpower
    n = 3
    sign = -1
    while True:
        xpower = xpower / (x*x)
        term = xpower / n
        if term == 0:
            break  # we are done
        sum += sign * term
        sign = -sign
        n += 2
    return sum
```

Applying Machin's formula Finally, the main function uses Machin's formula to compute π using the necessary level of precision, by using this high precision arccot function:

```
def machin(digits):
    unity = 10**(digits + 10)
    pi = 4 * (4*arccot(5, unity) - arccot(239, unity))
    return pi / unity
```

To avoid rounding errors in the result, we use 10 guard digits internally during the calculation. We may now reproduce the historical result obtained by Machin in 1706:

```
>>> machin(100)
3.14159265358979323846264338327950288419716939937751...
   05820974944592307816406286208998628034825342117067 9
```

The program can be used to compute tens of thousands of digits in just a few seconds on a modern computer. Many Machin-like formulas also exist, like:

$$\pi = 4\arctan\left(\frac{1}{2}\right) + 4\arctan\left(\frac{1}{3}\right)$$

(hard) Unbounded Spigot Algorithm This algorithm is quite efficient, but not easy to explain. Go check on-line if you want.

(hard) Borwein's algorithm It has several versions, one with a cubic convergence (each new step multiplies by 3 the number of digits), one with a nonic convergence (of order 9) etc. They are not so easy to explain either. Please check on-line, here `en.WikiPedia.org/wiki/Borwein's_algorithm`.

The cubic method is similar to the Gauss-Legendre algorithm:

1. Start with $a_0 = 1/3$, and $s_0 = \dfrac{\sqrt{3}-1}{2}$,
2. And then iterate, as much as you want, by defining $r_{k+1} = \dfrac{3}{1+2(1-s_k^3)^{1/3}}$, and updating $s_{k+1} = \dfrac{r_{k+1}-1}{2}$ and $a_{k+1} = r_{k+1}^2 a_k - 3^k(r_{k+1}^2 - 1)$.

Then the numbers a_k will converge to $\dfrac{1}{\pi}$.

The `decimal.Decimal` trick to improve precision

If you implement these methods by simply using `float` numbers for all the variables and the partial sum, the final precision of your approximation of π will be extremely limited (it will not be better than importing `math.pi`!).

So you should try to use **decimal** numbers instead, by importing the `decimal` module in Python. More details on that library here `docs.Python.org/2/library/decimal.html`. The basic thing you will need to use is the `decimal.Decimal` class, available if you import the module with `import decimal` (or `Decimal` if you import the `decimal` module with `from decimal import *`):

```
from decimal import *
# Example of use of Decimal
mypi = Decimal(22) / Decimal(7)
```

By default, the precision of such decimal numbers is 30, but you will obviously need to increase the precision. This can be done simply by increasing `getcontext().prec` (or `decimal.getcontext().prec`):

```
# If needed, increase the precision up to N digits after the comma:
getcontext().prec = 100000  # precision is now N = 100000
```

Examples and references

Link are given in the soft copy, available online and on Moodle.

- `en.WikiPedia.org/wiki/Pi#Modernquest`formoredigits,
- `www.JoyOfPi.com/pi.html` and `www.JoyOfPi.com/pilinks.html`,
- `www.EveAndersson.com/pi/digits/` has great interactive tools,
- more crazy stuff `MathWorld.Wolfram.com/PiDigits.html`, or `MathWorld.Wolfram.com/Pi.html`,
- one idea with Fibonacci numbers,
- and this incredibly long list of digits at `PiWorld.calico.jp/estart.html`.

100 **first digits of** π

$\pi \simeq$ 3.1415926535 8979323846 2643383279 5028841971
6939937510 5820974944 5923078164 9862803482 53421170679 when computed to the first 100 digits.

Can you compute up to 1000 digits? Up to 10000 digits? Up to 100000 digits? **Up to 1 million digits?**

I failed going further than 100000 (in less than 2 minutes). My best method produced 100000 correct digits in 22 seconds.

Problem 2 : plotting a pie chart

- **What do do ?** Plot a pie chart representing any interesting data you can think of (an example is given below, it can be a starting point),
- **How ?** You should start by using this PyLab/MatPlotLib tutorial (link below),
- **Which data to use ?** Any randomly generated data, or the list of students (a dictionary of students) (ask me how to download it),
- **What to plot ?** Gender (30% girl, 70% boy), sections etc : as you want ! Just try to plot something you find interesting.
- **How we would evaluate you ?** Some points if the chart looks fine, extra points for every extra small things (a title, a legend, name of the axis etc).

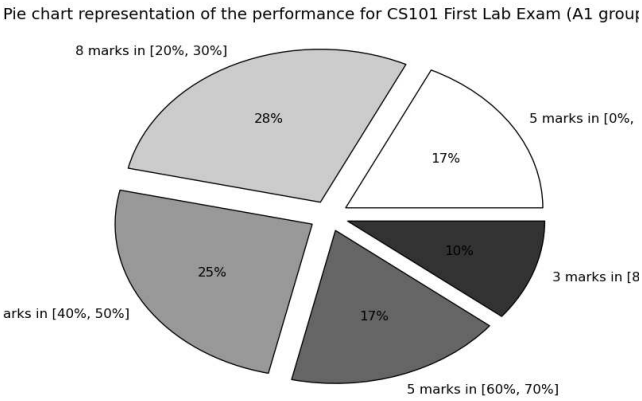
Default template program

```
N = 20

from random import uniform
grades = [ round(uniform(0, 100), 2) for i in xrange(N) ]

from pylab import *

fig1 = figure(1)
pie(grades)  # this will look useless, lets do better:

fig2 = figure(2)
data = [ sum( [ 1 for g in grades if percentage <= g < percentage+10 ] )
```

```
    for percentage in range(0,101,10) ]
pie( [ count for count in data if count > 0 ] )
```

Examples and references

- `www.LaBri.fr/perso/nrougier/teaching/matplotlib/`
- See this example: `www.LaBri.fr/perso/nrougier/teaching/matplotlib/scripts/pie_ex.py`
- `SciPy-lectures.GitHub.io/intro/matplotlib/matplotlib.html#pie-charts`



Task 3 : one extra task, but no programming

The task is quite simply: learn as many digits of π as (you think) you can. 20 is already an impressive number! One of the simplest ways to memorize Pi is to memorize sentences in which each word's length represents a digit of π. A first example is *"May I have a large container of coffee?"*, giving 3.1415926 (7 digits).

A second example is :

> Pie I wish I could recollect pi. "Eureka!," cried the great inventor. Christmas pudding, Christmas pie, is the problem's very center!

Many such poems can be found online.

Examples and references

- `www.WikiHow.com/Memorize-Pi`,
- `www.Ludism.org/mentat/PiMemorisation`,
- `www.SailorPi.com/poem.html`.

Prices for the best teams

Based on the criteria given above, and maybe some black magic, we will evaluate each team, by assessing the quality and efficiency of your programs (and more).

1. the best team would receive two gifts (one for each),
2. the second team would get one gift,
3. and the best learner of digits of π would receive anther gift.

Last remarks

Organizing team

Profs. Thiyagarajan and Lilian are in charge of organizing the event. Please contact the organizing team directly at `CS101@crans.org` if needed.

Credit and license

Disclaimer

Finally, all the quoted resources (websites, books, videos, slides, programs etc) are **the properties of their respective authors**, and neither me nor Mahindra École Centrale are affiliated to any of them.