Bachelor of Ecole Polytechnique
Computational Mathematics, year 2, semester 1
Lecturer: Lucas Gerin (send mail) (mailto:lucas.gerin@polytechnique.edu)

Symbolic computing 1: Proofs with SymPy

$$\pi \approx 384\sqrt{-\sqrt{\sqrt{\sqrt{\sqrt{\sqrt{\sqrt{\sqrt{3}+2}+2}+2}+2}+2}+2}+2}+2$$

Table of contents


```
# execute this part to modify the css style
from IPython.core.display import HTML
def css_styling():
    styles = open("./style/custom2.css").read()
    return HTML(styles)
css_styling()
```

```
## loading python libraries

# necessary to display plots inline:
%matplotlib inline

# load the libraries
import matplotlib.pyplot as plt # 2D plotting library
import numpy as np              # package for scientific computing
from pylab import *

from math import *              # package for mathematics (pi, arctan, sqrt, factorial
import sympy as sympy             # package for symbolic computation
from sympy import *
```

Introduction to SymPy

Using python library `SymPy` we can perform *exact* computations. For instance, run and compare the following scripts:

```
print('With Numpy: ')
print('root of two is '+str(np.sqrt(2))+'')
print('the square of (root of two) is '+str(np.sqrt(2)**2)+'')
print('---------')
print('With SymPy: ')
print('root of two is '+str(sympy.sqrt(2))+'')
print('the square of (root of two) is '+str(sympy.sqrt(2)**2)+'')
```

```
With Numpy:
root of two is 1.4142135623730951
the square of (root of two) is 2.0000000000000004
---------
With SymPy:
root of two is sqrt(2)
the square of (root of two) is 2
```

One can expand or simplify expressions:

```
print('Simplification of algebraic expressions:')
print('the square root of 40 is '+str(sympy.sqrt(40))+'')
print('(root(3)+root(2))**20 is equal to '+str(expand((sympy.sqrt(3)+sympy.sqrt(2))**20
#
print('----------------')
print('Simplification of symbolic expressions:')
var('x') # We declare a 'symbolic' variable
Expression=(x**2 - 2*x + 1)/(x-1)
print(str(Expression) + ' simplifies into '+str(simplify(Expression)))
```

```
Simplification of algebraic expressions:
the square root of 40 is 2*sqrt(10)
(root(3)+root(2))**20 is equal to 4517251249 + 1844160100*sqrt(6)
----------------
Simplification of symbolic expressions:
(x**2 - 2*x + 1)/(x - 1) simplifies into x - 1
```

With `Sympy` one can also obtain Taylor expansions of functions with `series` :

```
# Real variable
var('x')
Expression=cos(x)
print('Expansion of cos(x) at x=0: '+str(Expression.series(x,0)))

# integer variable
var('n',integer=True)
Expression=cos(1/n)
print('Expansion of cos(1/n) when n -> +oo: '+str(Expression.series(n,oo)))    # oo mean
```

```
Expansion of cos(x) at x=0: 1 - x**2/2 + x**4/24 + O(x**6)
```

SymPy can also compute with "big O's". (By default $\mathcal{O}(x)$ is considered for $x \to 0$.)

```
var('x')
simplify((x+O(x**3))*(x+x**2+O(x**3)))
```

$x^2 + x^3 + O\left(x^4\right)$

> **Remark.** A nice feature of `Sympy` is that you can export formulas in LaTeX. You also can get nice displays with `display`.

```
var('x y')
formula=simplify((cos(x+y)-sin(x+y))**2)
print(formula)
print(latex(formula))
display(formula)
```

```
2*cos(x + y + pi/4)**2
2 \cos^{2}{\left(x + y + \frac{\pi}{4} \right)}
```

$2\cos^2\left(x + y + \frac{\pi}{4}\right)$

Warning: Fractions such as $1/4$ must be introduced with `Rational(1,4)` to keep `Sympy` from evaluating the expression. For example:

```
print('(1/4)^3 = '+str((1/4)**3))
print('(1/4)^3 = '+str(Rational(1,4)**3))
```

Let SymPy do the proofs

Exercise 1. A warm-up

> **Do it yourself.**
> Set $\phi = \frac{1+\sqrt{5}}{2}$. Use `SymPy` to simplify $F = \frac{\phi^4 - \phi}{1+\phi^7}$.

Exercise 2. A simple (?) recurrence

We will see how to use SymPy to prove a mathematical statement. Our aim is to make as rigorous proofs as possible, as long as we trust SymPy.

Do it yourself.

Let a, b be two real numbers, we define the sequence $(u_n)_{n\geq 0}$ as follows: $u_0 = a, u_1 = b$ and for $n \geq 2$

$$u_n = \frac{1 + u_{n-1}}{u_{n-2}}.$$

1. Write a short program which returns the 15 first values of u_n in terms of symbolic variables a, b. The output should be something like:

```
u_0 = a
u_1 = b
u_2 = (b + 1)/a
...
```

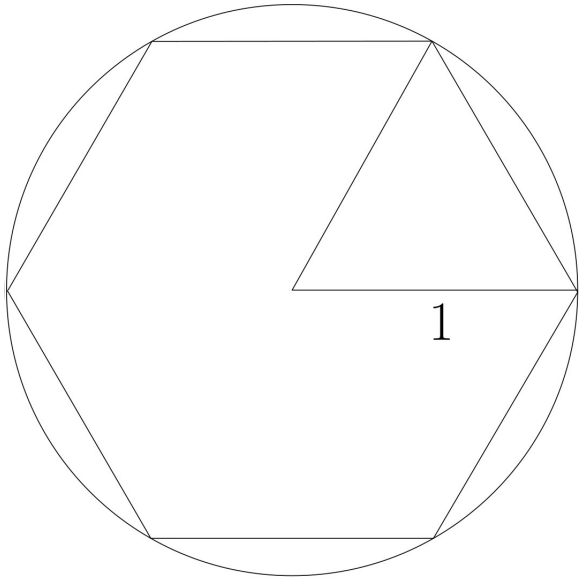
2. Use `SymPy` to make and prove a conjecture for the asymptotic behaviour of the sequence (u_n), for every reals a, b.

Answers.

1.
2.

Exercise 3. What if Archimedes had known `Sympy` ?

For $n \geq 1$, let $\mathcal{P}_n$ be a regular 3×2^n-gon with radius 1. Here is $\mathcal{P}_1$:

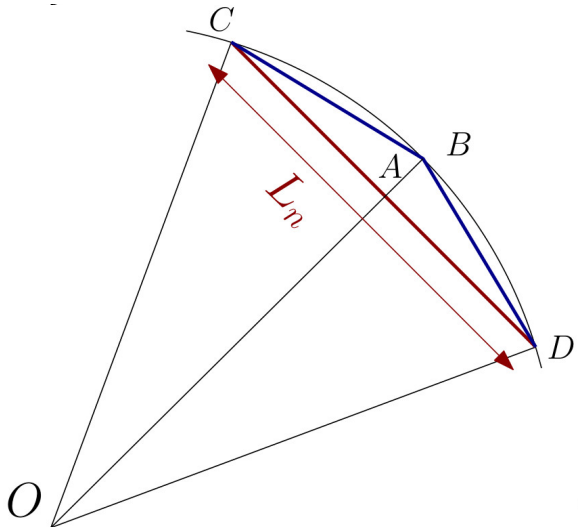


Archimedes (IIIrd century BC) used the fact that $\mathcal{P}_n$ gets closer and closer to the unit circle to

obtain good approximations of π.
We will use SymPy to deduce nice formulas for approximations of π.

Do it yourself. Let L_n be the length of any side of $\mathcal{P}_n$. Compute L_1 and use the following picture to write L_{n+1} as a function of L_n:

- O is the center of the circle, $OC = 1$.
- (OB) is the bisector of $\widehat{DOC}$.
- $\widehat{OAC}$ is a right angle.



Answers.

Do it yourself. 1. Write a script which computes exact expressions for the first values $L_1, L_2, \dots, L_n$. (First try for small n's.)

2. Find a sequence a_n such that $a_n L_n$ converges to π (we don't ask for a proof). Deduce some good algebraic approximations of π. Export your results in `Latex` in order to get nice formulas.

(In order to check your formulas, you may compute numerical evaluations. With `SymPy`*, a numerical evaluation is obtained with* `N(expression)`*.)*

Answers.

Solving equations with SymPy

One can solve equations with Sympy. The following script shows how to solve $x^2 = x + 1$:

```
var('x') # we declare the variable
SetOfSolutions=solve(x**2-x-1,x)
print(SetOfSolutions)
```

Exercise 4. Solving equations with Sympy: the easy case

Let $n \geq 1$ be an integer, we are interested in solving the equation

$$x^3 + nx = 1. \qquad (\star)$$

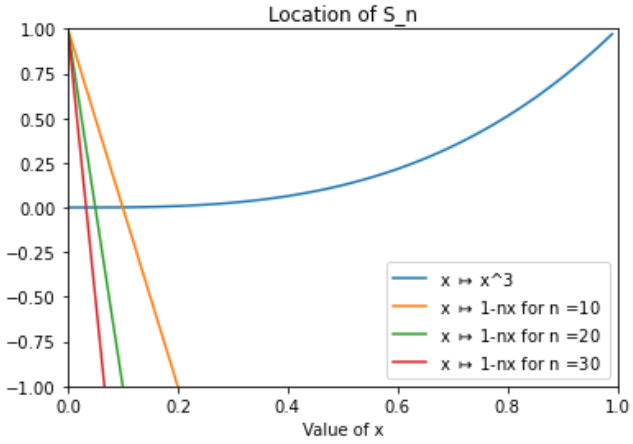
In the script below we plot $x \mapsto x^3$, and $x \mapsto 1 - nx$ for $0 \leq x \leq 1$ and for several (large) values of n. This suggests that Equation $(\star)$ has a unique real solution in the interval $[0, 1]$, that we will denote by S_n.

```
RangeOf_x=np.arange(0,1,0.01)

plt.plot(RangeOf_x,RangeOf_x**3,label='x $\mapsto$ x^3')

for n in [10, 20, 30]:
    f=[1-n*x for x in RangeOf_x]
    plt.plot(RangeOf_x,f,label='x $\mapsto$ 1-nx for n ='+str(n)+' ')

plt.xlim(0, 1),plt.ylim(-1, 1)
plt.xlabel('Value of x')
plt.legend()
plt.title('Location of S_n')
plt.show()
```



Do it yourself. **(Theory)**

1. Prove that indeed for every $n \geq 1$, Equation $(\star)$ has a unique real solution in the interval $[0, 1]$.
2. According to the plot, what can we conjecture for the limit S_n?

Answers.

1.
2.

Do it yourself.

1. Use the function `solve()` to compute the exact expression of S_n.
2. Use `SymPy` and the function `series` to get the asymptotic expansion of S_n (up to $\mathcal{O}(1/n^5)$). Check your previous conjecture.

Answers.

1.
2.

Exercise 5. Solving equations: when SymPy needs help

We consider the following equation:

$$X^5 - 3\varepsilon X^4 - 1 = 0, \qquad (\star)$$

where ε is a positive parameter. As in previous exercise a quick analysis shows that if $\varepsilon > 0$ is small enough then $(\star)$ has a unique real solution, that we denote by S_ε.

The degree of this equation is too high to be solved by `SymPy` :

```
var('x')
var('e')
solve(x**5-3*e*x**4-1,x)
```

```
[]
```

Indeed, `SymPy` needs help to handle this equation.

In the above script we plotted the function $f(x) = x^5 - 3\varepsilon x^4 - 1$ for some small ε. This suggests that $\lim_{\varepsilon \to 0} S_\varepsilon = 1$.

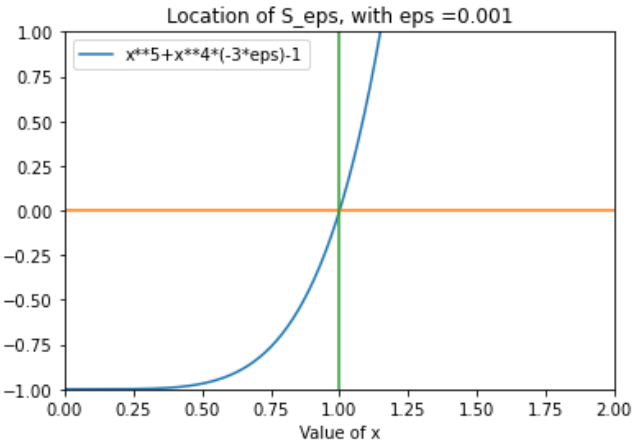
```
RangeOf_x=np.arange(0,2,0.01)

def ToBeZero(x,eps):
    return x**5+x**4*(-3*eps) -1

eps=0.001
plt.plot(RangeOf_x,[ToBeZero(x,eps) for x in RangeOf_x],label='x**5+x**4*(-3*eps)-1')

plt.xlim(0, 2)
plt.ylim(-1, 1)
plt.plot([-2,2],[0,0])
```

```
plt.plot([1,1],[-2,2])
plt.xlabel('Value of x')
plt.title('Location of S_eps, with eps ='+str(eps))
plt.legend()
plt.show()
```



Do it yourself.

We admit that S_ε can be written as

$$S_\varepsilon = 1 + r\varepsilon + s\varepsilon^2 + \mathcal{O}(\varepsilon^3),$$

for some real r, s. Use `SymPy` to find r, s.

(You can use any `SymPy` function already seen in this notebook.)

Answers.