L1 — Calcul Scientifique - Chapitre 1

Frédéric Jurie (frederic.jurie@unicaen.fr)
Université de Caen Normandie
Année universitaire 2019-2020

Le calcul symbolique avec la librairie ‘sympy’

La librairie sympy

- Permet de faire du calcul sur des expressions contenant des symboles et non des nombres
- Par exemple : factoriser l'expression $x^3 + 3x^2 + 3x + 1$
- La librairie sympy permet de définir des symboles et de faire des opérations dessus
- Utilisation d'expression en Python :

```
In [1]:  x = 1
         x + x + 1
Out[1]: 3
```

- Ne donne pas $2x + 1$!
- Avec sympy:

```
import sympy
x = sympy.symbols("x")
2*x+1
```

```
2*x + 1
```

Les symboles

- En sympy, les symboles sont les éléments de base du calcul algèbrique ou calcul littéral
- La définition d'un symbole se fait avec la fonction symbols :
  `x = sympy.symbols("x")`
- Par défaut un `symbol` représente un réel, mais possible de forcer le type :
  `z = sympy.symbols("z",imaginary=True)`
  `n = sympy.symbols("n",integer=True)`
- Possibilité de définir plusieurs symboles d'un coup
  `x,y = sympy.symbols("x,y")`
- Attention à bien distinguer le nom du symbole et celui de la variables :
  `u = sympy.symbol("x")`
  La variable u contient le symbole x

```
import sympy
u = sympy.symbols("x")
2*u+1
```

```
2*x + 1
```

Les symboles (suite) et calculs élémentaires

- Il est possible de connaitre le nom du symbole contenu dans une variable de type symbolique :

```
x = sympy.symbols("x")
x.name
```

```
'x'
```

```
u = sympy.symbols("x")
u.name
```

```
'x'
```

- Une fois les symboles définis, il est possible de les utiliser dans des calculs arithmétiques simples :

```
x,y=sympy.symbols('x,y')
z = 2*x+y*x
```

```
z
```

```
x*y + 2*x
```

Les symboles (suite) et calculs élémentaires

- Sympy fait automatiquement des simplifications dans des cas simples :

```
x,y=sympy.symbols('x,y')
z = x*y+x*y
```

```
z
```

```
2*x*y
```

- Mais dans les autres cas les expressions sont conservées telles que données par l'utilisateur :

```
x,y=sympy.symbols('x,y')
z = (x+2)*(x+3)
```

```
z
```

```
(x + 2)*(x + 3)
```

Factorisation et développement

- Sympy possède une fonction permettant de développer les expressions (expand):

```
x,y=sympy.symbols('x,y')
z = (x+2)*(x+3)
z = z.expand()
print(z)
```

```
x**2 + 5*x + 6
```

Ou :

```
x,y=sympy.symbols('x,y')
z = (x+2)*(x+3)
z = sympy.expand(z)
print(z)
```

```
x**2 + 5*x + 6
```

- Et une autre permettant de factoriser (factor):

```
z = x**2 + 5*x + 6
print(z,'->',z.factor())
```

```
x**2 + 5*x + 6 -> (x + 2)*(x + 3)
```

- Si pas de factorisation possible, l'expression reste identique

Affichage ‘joli’

- Sympy possède une fonction permettant d’afficher des expressions mathématique de manière plus jolie qu’un simple print :

```
z = x**2 + 5*x + 6
print(z,'->',z.factor())
sympy.pprint(z)
```

```
x**2 + 5*x + 6 -> (x + 2)*(x + 3)
 2
x  + 5·x + 6
```

- Les termes sont arrangés par ordre des puissances décroissantes par défaut, mais cela peut être modifié (inversé)

Construction d'une série

- Il est possible de construire des fonctions littérales en utilisant des structures itératives
- Par exemple, construisons la série :

$$x + \frac{x^2}{2} + \frac{x^3}{3} + \frac{x^4}{4} + \ldots + \frac{x^n}{n}.$$

- Pour cela, nous pouvons écrire la fonction `serie(n)` où `n` est la valeur du dernier indices de la série :

```python
def serie(n):
    x = sympy.symbols('x')
    s = y
    for i in range (2,n+1):
        s = s + x**i/i
    return s

print(serie(5))
sympy.pprint(serie(5))
```

```
x**5/5 + x**4/4 + x**3/3 + x**2/2 + y
 5    4    3    2
x    x    x    x
── + ── + ── + ── + y
5    4    3    2
```

Substitution de valeurs

- Nous pouvons vouloir, à certains moments, remplacer les symboles par des valeurs numériques
- Prenons l'exemple de : $x^2 + 2xy + y^2$
- Que nous voulons évaluer pour x= 1 et y = 2

```
x,y = sympy.symbols('x,y')
f = x**2+y**2+2*x
sympy.pprint(f)
```

```
 2           2
x  + 2⋅x + y
```

```
res = f.subs({x:1,y:2})
print(res)
```

```
7
```

Substitution partielle

```
res = f.subs({x:1})
print(res)
```

```
y**2 + 3
```

- L'argument de la fonction subs est un dictionnaire python

Substitution de valeurs (suite)

- Il est également possible de substituter des symboles par d'autres symboles ou expressions
- Par exemple, dans le cas précédent, si nous savons que x = 1 - y
- Nous pouvons écrire :

```python
g = f.subs({x:1-y})
print(g)
```

```
y**2 - 2*y + (-y + 1)**2 + 2
```

Simplification des expressions

- sympy possède des méthodes permettant la simplification des expressions
- Par exemple, dans l'exemple précédent, le résultat peut être simplifié

```
e = y**2 - 2*y + (-y + 1)**2 + 2
e_s = e.simplify()
print(e_s)
```

```
2*y**2 - 4*y + 3
```

- Pratique lorsque l'on a oublié les formules trigonométriques :

```
e = y**2 - 2*y + (-y + 1)**2 + 2
e_s = e.simplify()
print(e_s)
```

```
2*y**2 - 4*y + 3
```

- Attention de bien utiliser les fonctions mathématique de sympy

Conversion chaines de caractères en expressions mathématiques

- En plus de la méthode précédemment vue pour introduire des expressions mathématiques littérales, sympy possède un parser permettant de saisir les expressions plus simplement :

```
e = input('entrer une expression mathématique :')
es = sympy.sympify(e)
2*es
```

```
entrer une expression mathématique :3*x**2-7*x+3

6*x**2 - 14*x + 6
```

Résolution d'équations

- La fonction `solve(expr)` permet de trouver les solutions de `expr`
- Exemple :

```python
x = sympy.symbols('x')
expr = x**2 + 5*x + 4
print(sympy.solve(expr))
```

```
[-4, -1]
```

- La fonction retourne une liste car plusieurs valeurs peuvent être solutions.
- Les solutions peuvent être des complexes. Exemples :

```python
x = sympy.symbols('x')
expr = x**2 + x + 1
print(sympy.solve(expr))
```

```
[-1/2 - sqrt(3)*I/2, -1/2 + sqrt(3)*I/2]
```

Résolution d'équations (suite)

- Par défaut, `solve` s'attend à résoudre des expressions de type f(x)=0
- Possibilité de donner une égalité complète :
-

```
e = sympy.Eq(x*x,2)
print(e)
print(sympy.solve(e))
```

```
Eq(x**2, 2)
[-sqrt(2), sqrt(2)]
```

- Et de récupérer un dictionnaire en sortie :

```
e = sympy.Eq(x*x,2)
print(e)
print(sympy.solve(e,dict=True))
```

```
Eq(x**2, 2)
[{x: -sqrt(2)}, {x: sqrt(2)}]
```

Résolution d'équations à plusieurs variables

- Prenons le cas de l'expressions $ax^2 + bx + c = 0$
- Il est possible de considérer a et b comme variables symboliques et de calculer les solutions en fonction de a et b.

```
x,a,b,c = sympy.symbols('x,a,b,c')
expr = a*x**2 + b*x + c
print(sympy.solve(expr,x))
```

```
[(-b + sqrt(-4*a*c + b**2))/(2*a), -(b + sqrt(-4*a*c + b**2))/(2*a)]
```

- Idem avec réponse sous la forme d'un dictionnaire :

```
x,a,b,c = sympy.symbols('x,a,b,c')
expr = a*x**2 + b*x + c
print(sympy.solve(expr,x,dict=True))
```

```
[{x: (-b + sqrt(-4*a*c + b**2))/(2*a)}, {x: -(b + sqrt(-4*a*c + b**2))/(2*a)}]
```

plus facile à utiliser pour d'autres calculs

Résolution de systèmes d'équations

- Supposons que nous cherchons les solutions de :

$$2x + 3y = 6$$

$$3x + 2y = 12$$

- Nous pouvons écrire :

```
x,y=sympy.symbols('x,y')
e1 = 2*x + 3*y - 6
e2 = 3*x + 2*y - 12
sympy.solve((e1,e2),dict=True)
```

```
[{x: 24/5, y: -6/5}]
```

- Il est possible de vérifier que la 1ere équation est vérifiée :

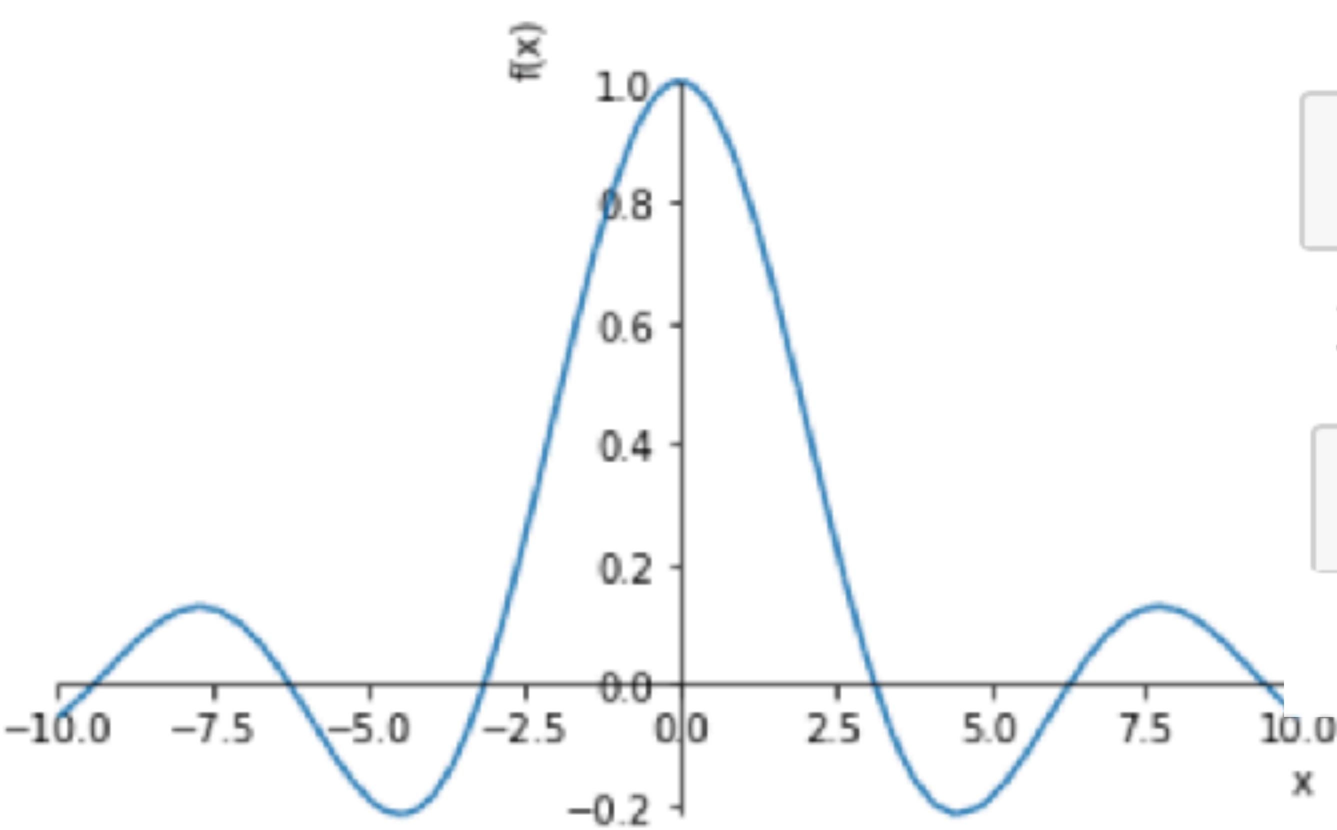
```
sol =sympy.solve((e1,e2),dict=True)
e1.subs({x:sol[0][x],y:sol[0][y]})
```

```
0
```

Limites d'une fonction

- La fonction '`limit`' de sympy permet de trouver les limites des fonctions
- Par exemple la fonction sinus cardinal :

```
%matplotlib inline
import matplotlib.pyplot as plt
x = sympy.symbols('x')
e = sympy.sin(x)/x
sympy.plot(e)
```




```
sympy.limit(e,x,0)
```

1

```
sympy.limit(e,x,sympy.oo)
```

0

Limites d'une fonction (suite)

- Possibilité de calculer les limites à droite et à gauche
- Exemple :

```
sympy.limit(1/x,x,0, dir = "+")
```

oo

```
sympy.limit(1/x,x,0, dir = "-")
```

-oo

Dérivées

- La fonction `diff` permet de calculer les dérivées de manière littérale

```
x = sympy.symbols('x')
sympy.diff(sympy.sin(x),x)
```

```
cos(x)
```

```
x = sympy.symbols('x')
sympy.diff(sympy.sin(x)*sympy.exp(x)/x,x)
```

```
exp(x)*sin(x)/x + exp(x)*cos(x)/x - exp(x)*sin(x)/x**2
```

- Possibilité de calculer la dérivée n-ème :

```
x = sympy.symbols('x')
sympy.diff(sympy.sin(x),x,2)
```

```
-sin(x)
```

- Ou encore les dérivées partielles :

```
x,y = sympy.symbols('x,y')
f = x**2+y**2
sympy.diff(f,x)
```

```
2*x
```

Intégration littérale

- Aussi simple à faire que le calcule des dérivées, grace à la fonction `integrate`
- Exemples

```
x = sympy.symbols('x')
sympy.integrate(sympy.sin(x)*sympy.cos(x))
```

```
sin(x)**2/2
```

- Il est également possible de calculer la valeur de l'intégrale entre deux bornes (attention à la manière de définir les bornes) :

```
x = sympy.symbols('x')
sympy.integrate(sympy.sin(x),(x,0,sympy.pi/2))
```

```
1
```

Intégrales multiples

- Exemple :

```
x,y = sympy.symbols('x,y')
sympy.integrate(x**2+y**2,x,y)
```

```
x**3*y/3 + x*y**3/3
```

```
sympy.integrate(x**2+y**2,(x,0,1),(y,0,1))
```

```
2/3
```

Développement limité

- Le développement limité d'une fonction, au voisinage d'un point à un ordre n, donne la fonction polynomiale de degré inférieur égal à n qui approxime le mieux la fonction au voisinage du point considéré.
- La fonction `series` permet d'obtenir ces développements limités de manière littérale
- Exemple :

```
x = sympy.symbols('x')
sympy.series(sympy.cos(x),n=4,x0=0)
```

```
1 - x**2/2 + O(x**4)
```

Transformation des fonctions littérales -> Python

- Les fonctions crées avec sympy sont littérales
- Possible d'évaluer la fonction numériquement avec méthode 'subs' mais appels à la fonction python impossible :

```python
x,y,z = sympy.symbols('x,y,z')
f = x**2+y**2+z**2
f(1,1)
```

```
---------------------------------------------------------------------------
TypeError                                 T
call last)
<ipython-input-91-0b558ca22ba3> in <module>
      1 x,y,z = sympy.symbols('x,y,z')
      2 f = x**2+y**2+z**2
----> 3 f(1,1)

TypeError: 'Add' object is not callable
```

- Problématique lors que l'on veut utiliser ces fonctions avec numpy (ou scipy ...)

Transformation des fonctions littérales -> Python (suite)

- Il est possible de construire automatiquement des fonctions python correspondant aux fonctions sympy :

```
f_p = sympy.lambdify('x,y,z',f)
f_p(1,1,1)
```

```
3
```

- La fonction est ensuite connue de python :

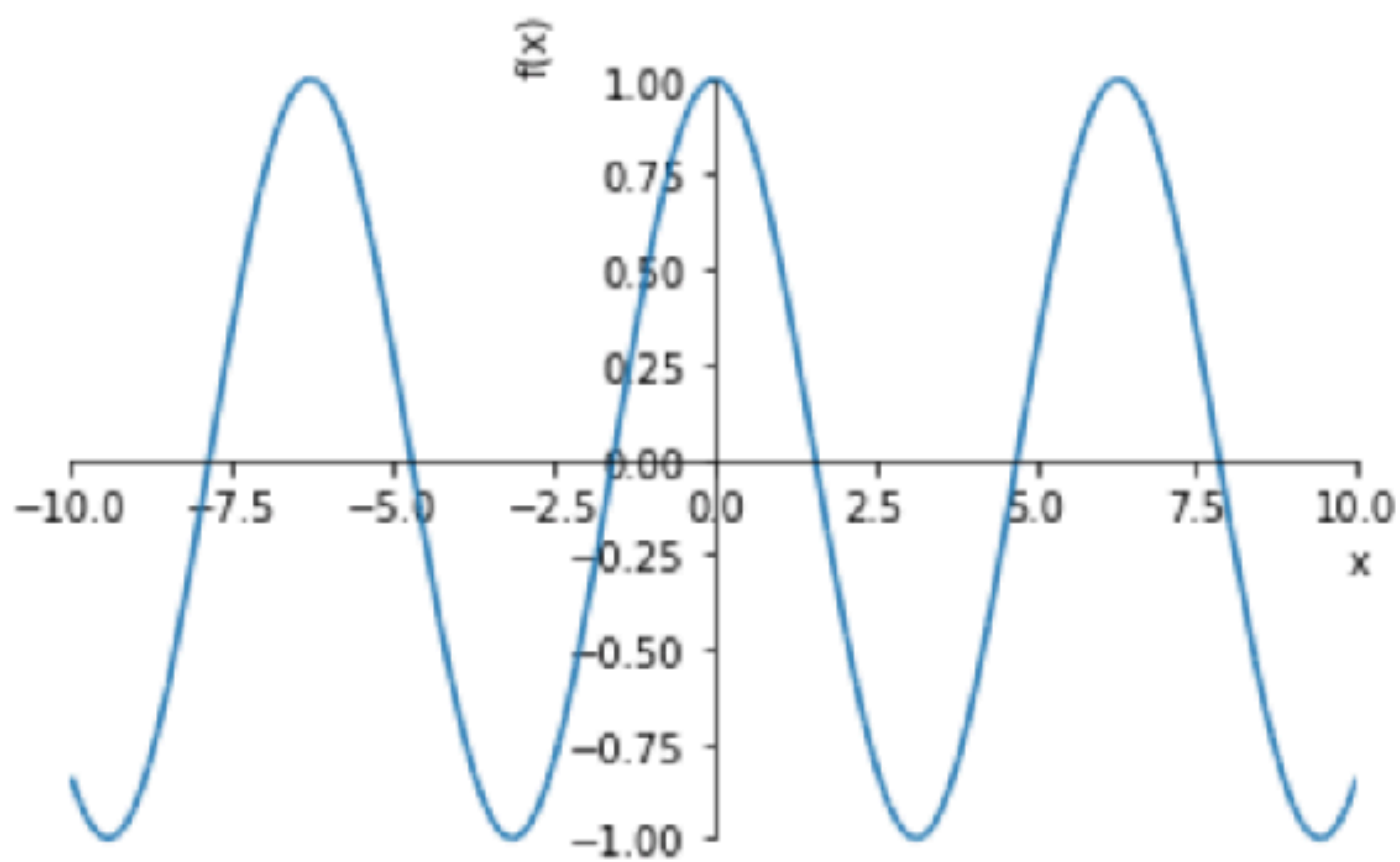
```
In [93]: f_p?
```

```
Signature: f_p(x, y, z)
Docstring:
Created with lambdify. Signature:

func(arg_0)

Expression:

x**2 + y**2 + z**2
File:      Dynamically generated function. No source code available.
Type:      function
```

Tracé de courbes avec sympy

- Il est possible de tracer des graphes à partir de fonctions littérales avec la fonction sympy `‘plot’`
- Exemple :

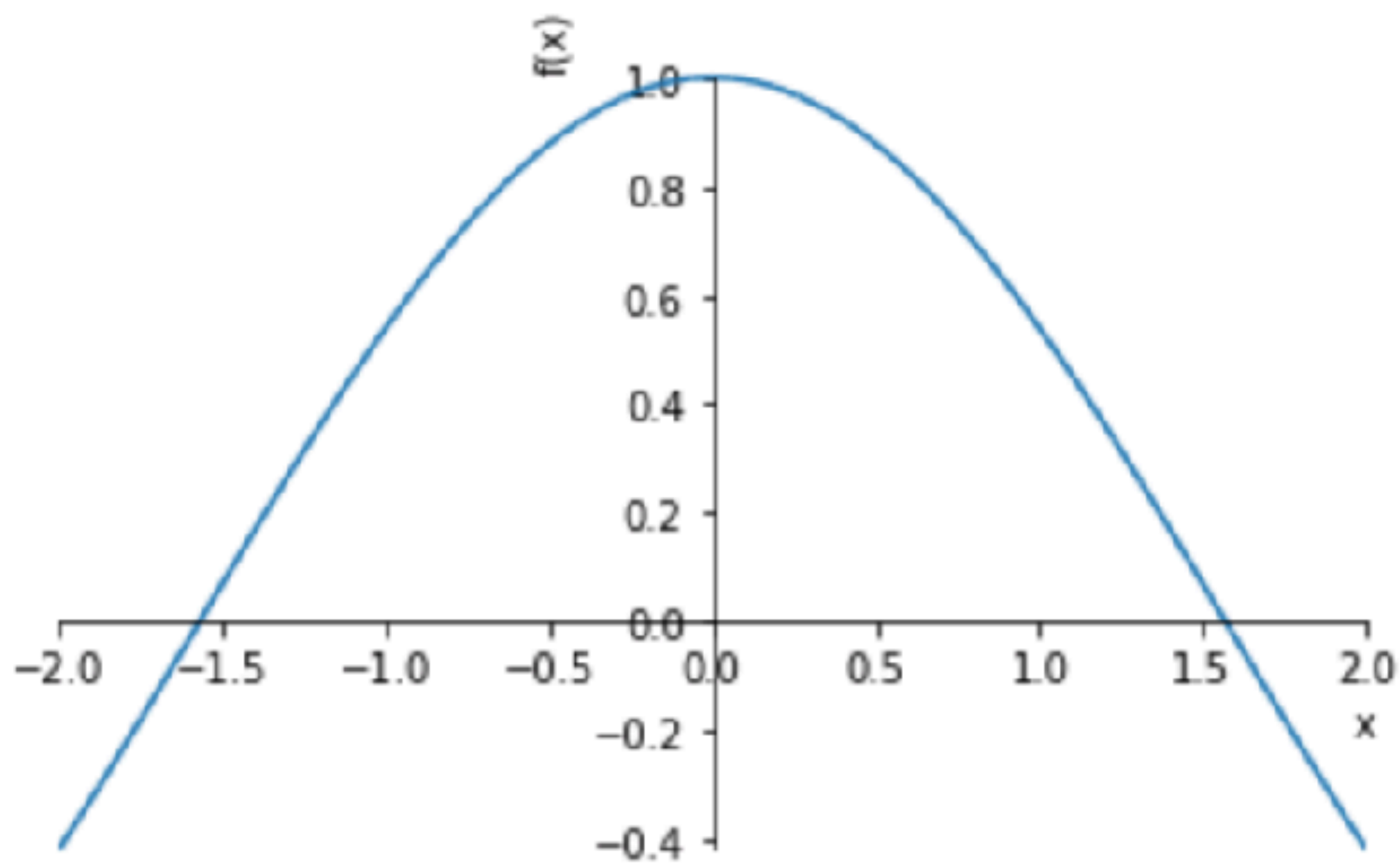
```
x = sympy.symbols('x')
sympy.plot(sympy.cos(x))

```



Tracé de courbes avec sympy (suite)

- Les plages de valeurs des variables littérales peuvent être spécifiées :
- Example :

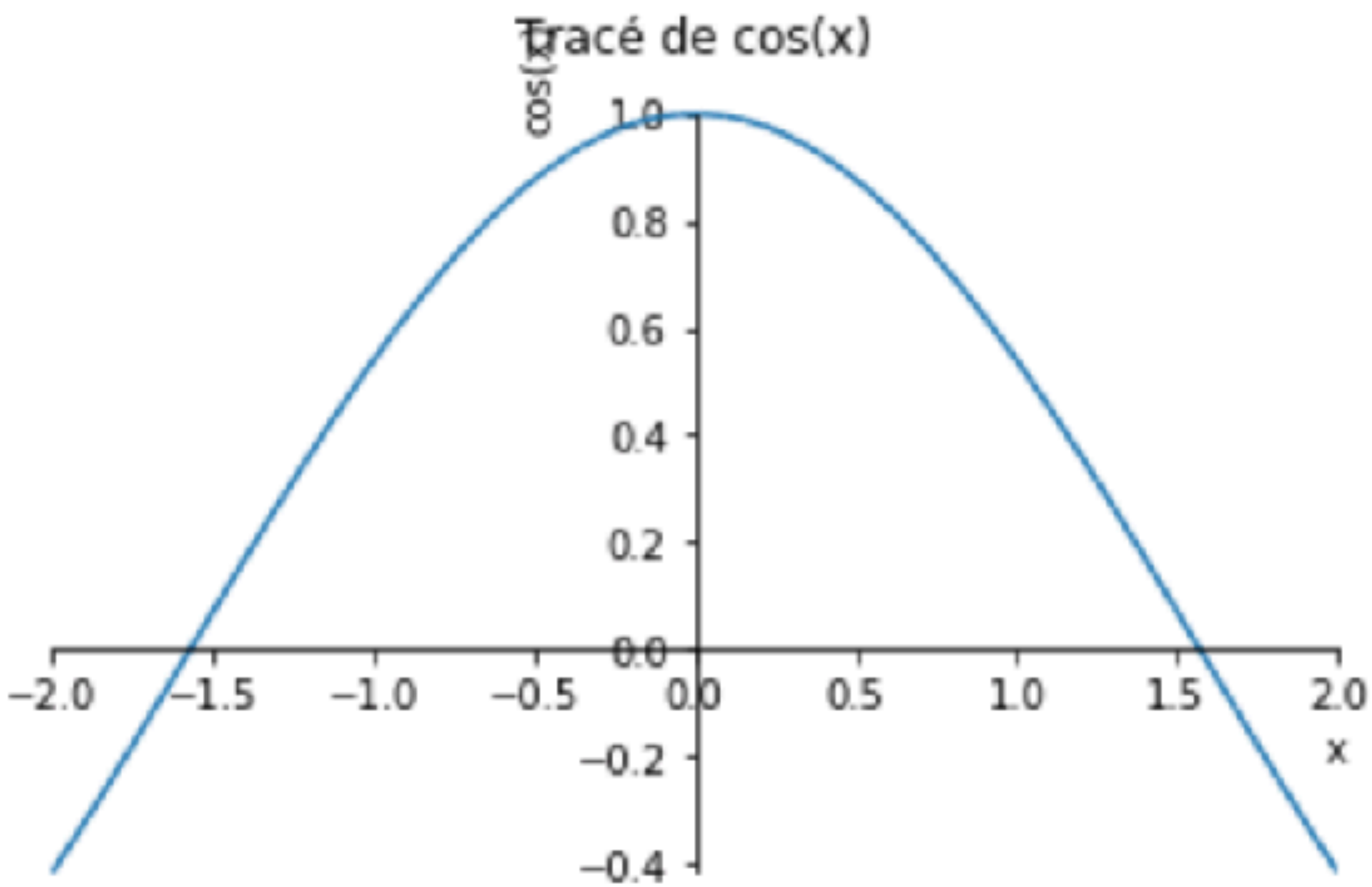
```
x = sympy.symbols('x')
sympy.plot(sympy.cos(x),(x,-2,2))

```



Tracé de courbes avec sympy (suite)

- Possibilité de ‘décorer’ les courbes et de les sauver
- Example :

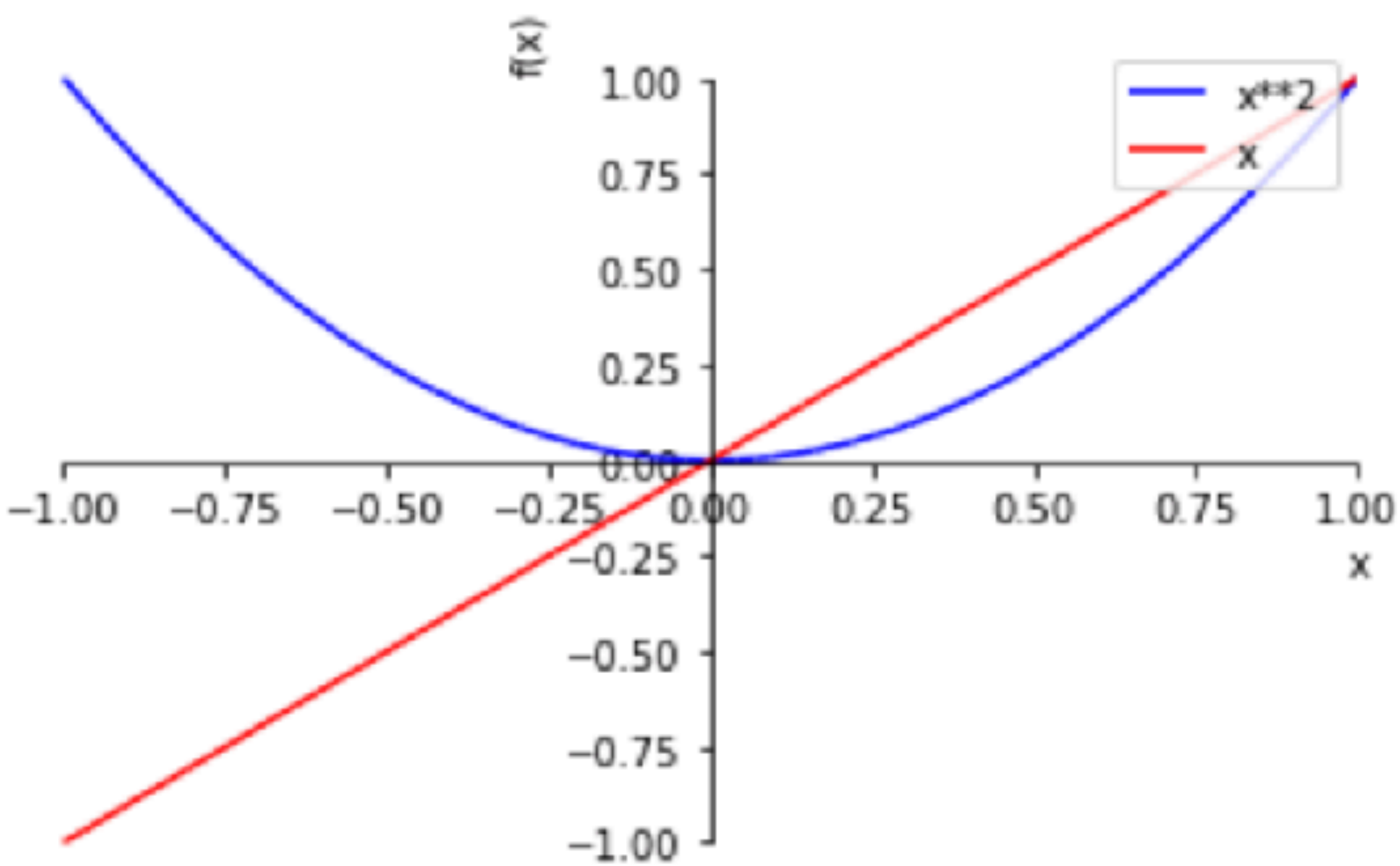
```
ymbols('x')
sympy.cos(x),(x,-2,2),title='Tracé de cos(x)',xlabel='x',ylabel='cos(x)')
```



Tracé de courbes avec sympy (suite)

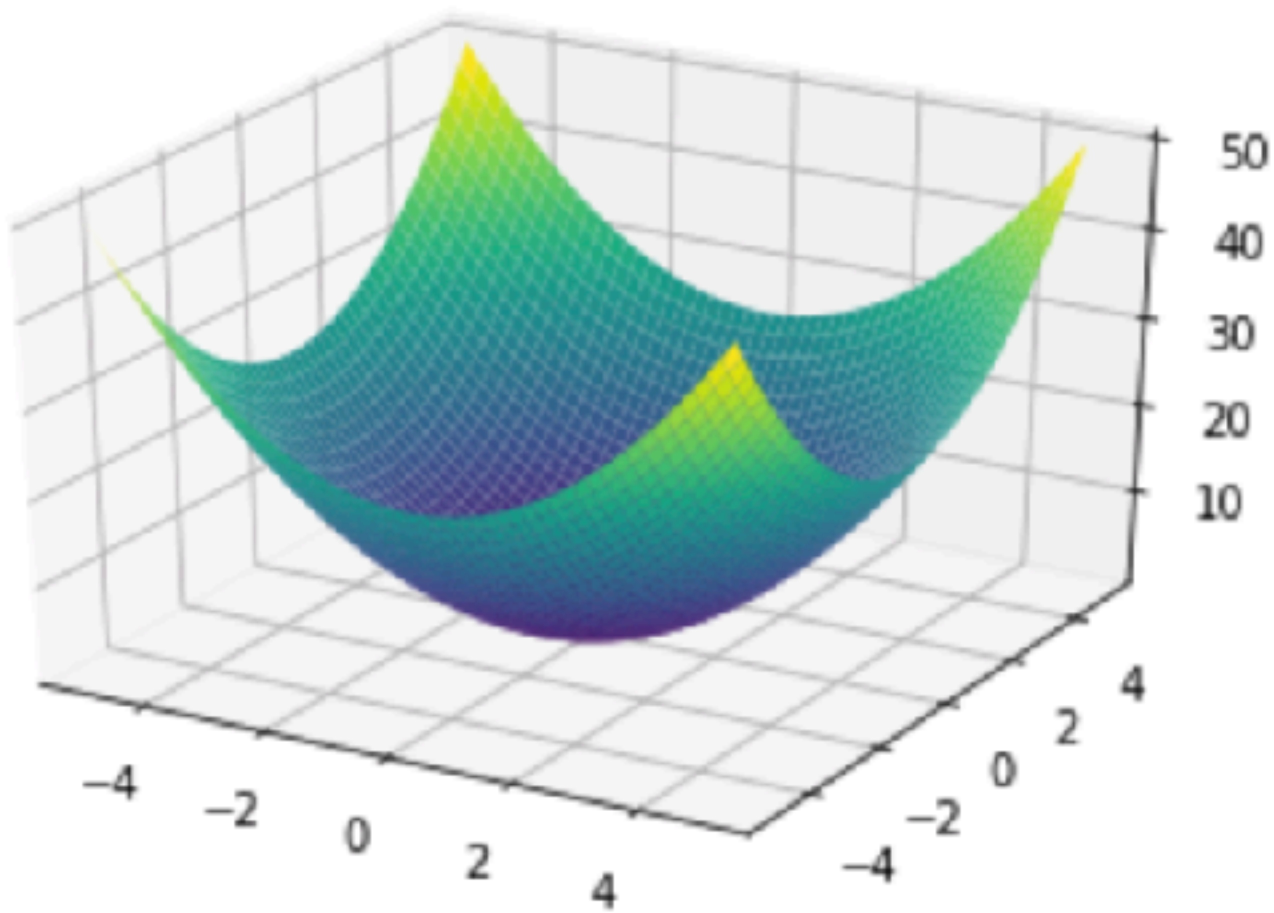
- Possibilité de tracer plusieurs courbes par graphes et de fixer les propriétés de chaque courbe
- Example :

```
p = sympy.plot(x**2,x,(x,-1,1),legend=True,show=False)
p[0].line_color='b'
p[1].line_color='r'
p.show()
```



Tracé de surfaces en 3D

```
x,y = sympy.symbols('x,y')
sympy.plotting.plot3d(x**2+y**2,(x,-5,5),(y,-5,5))
```



SymPy Cheatsheet (http://sympy.org)

Basics

Sympy help:	`help(function)`
Declare symbol:	`x = Symbol('x')`
Substitution:	`expr.subs(old, new)`
Numerical evaluation:	`expr.evalf()`
Expanding:	`expr.expand()`
Common denominator:	`ratsimp(expr)`
Simplify expression:	`simplify(expr)`

Constants

π:	`pi`
e:	`E`
∞:	`oo`
i:	`I`

Numbers types

Integers ($\mathbb{Z}$):	`Integer(x)`
Rationals ($\mathbb{Q}$):	`Rational(p, q)`
Reals ($\mathbb{R}$):	`Float(x)`

Basic funtions

Trigonometric:	`sin cos tan cot`
Cyclometric:	`asin acos atan acot`
Hyperbolic:	`sinh cosh tanh coth`
Area hyperbolic:	`asinh acosh atanh acoth`
Exponential:	`exp(x)`
Square root:	`sqrt(x)`
Logarithm ($\log_b a$):	`log(a, b)`
Natural logarithm:	`log(a)`
Gamma ($\Gamma(x)$):	`gamma(x)`
Absolute value:	`abs(x)`

Calculus

$\lim_{x\to a} f(x)$:	`limit(f, x, a)`
$\lim_{x\to a_-} f(x)$:	`limit(f, x, a, dir='-')`
$\lim_{x\to a_+} f(x)$:	`limit(f, x, a, dir='+')`
$\frac{d}{dx} f(x)$:	`diff(f, x)`
$\frac{\partial}{\partial x} f(x,y)$:	`diff(f, x)`
$\int f(x)\,dx$:	`integrate(f, x)`
$\int_a^b f(x)\,dx$:	`integrate(f, (x, a, b))`
Taylor series (at a, deg n)	`f.series(x, a, n)`

Equations

Equation $f(x) = 0$:	`solve(f, x)`
System of equations:	`solve([f, g], [x, y])`
Differential equation:	`dsolve(equation, f(x))`

Geometry

Points:	`a = Point(xcoord, ycoord)`
Lines:	`l = Line(pointA, pointB)`
Circles:	`c = Circle(center, radius)`
Triangles:	`t = Triangle(a, b, c)`
Area:	`object.area`
Intersection:	`intersection(a, b)`
Checking tangency:	`c.is_tangent(l)`

Plotting

Plot:	`Plot(f, [a, b])`
Zoom: +/−:	`R/F or PgUp/PgDn or Numpad +/-`
Rotate X,Y axis:	`Arrow Keys or WASD`
Rotate Z axis:	`Q and E or Numpad 7 and 9`
View XY:	`F1`
View XZ:	`F2`
View YZ:	`F3`
View Perspective:	`F4`
Axes Visibility:	`F5`
Axes Colors:	`F6`
Screenshot:	`F8`
Exit plot:	`ESC`

Discrete math

Factorial ($n!$):	`factorial(n)`
Binomial coefficient $\binom{n}{k}$:	`binomial(n, k)`
Sum ($\sum_{n=a}^{b} expr$):	`summation(expr, (n, a, b))`
Product ($\prod_{n=a}^{b} expr$):	`product(expr, (n, a, b))`

Linear algebra

Matrix definition:	`m = Matrix([[a, b], [c, d]])`
Determinant:	`m.det()`
Inverse:	`m.inv()`
Identity matrix $n \times n$:	`eye(n)`
Zero matrix $n \times n$:	`zeros(n)`
Ones matrix $n \times n$:	`ones(n)`

Printing

LaTeX print:	`print latex()`
Python print:	`print python()`
Pretty print:	`pprint()`

Examples

Find 100 digits of π^e:
```
(pi**E).n(100)
```

Expand $(x+y)^2(x-y)(x^2+y)$:
```
((x + y)**2 * (x - y) * (x**2 + y)).expand()
```

Simplify $\frac{1}{x} + \frac{x \sin x - 1}{x^2 - 1}$:
```
simplify((1/x) + (x * sin(x) - 1)/(x**2 - 1))
```

Check if line passing through points $(0,1)$ and $(1,1)$ is tangent to circle with center at $(5,5)$ and radius 3:
```
Circle(Point(5,5), 3).is_tangent(
Line(Point(0,1), Point(1,1)))
```

Find roots of $x^4 - 4x^3 + 2x^2 - x = 0$:
```
solve(x**4 - 4*x**3 + 2*x**2 - x, x)
```

Solve the equations system: $x + y = 4$, $xy = 3$:
```
solve([x + y - 4, x*y - 3], [x, y])
```

Calculate limit of the sequence $\sqrt[n]{n}$:
```
limit(n**(1/n), n, oo)
```

Calculate left-sided limit of the function $\frac{|x|}{x}$ in 0:
```
limit(abs(x)/x, x, 0, dir='-')
```

Calculate the sum $\sum_{n=0}^{100} n^2$:
```
summation(n**2, (n, 0, 100))
```

Calculate the sum $\sum_{n=0}^{\infty} \frac{1}{n^2}$:
```
summation(1/n**2, (n, 0, oo))
```

Calculate the integral $\int \cos^3 x\,dx$:
```
integrate(cos(x)**3, x)
```

Calculate the integral $\int_1^{\infty} \frac{dx}{x^2}$:
```
integrate(1/x**2, (x, 1, oo))
```

Find 10 terms of series expansion of $\frac{1}{1-2x}$ at 0:
```
(1/(1 - 2*x)).series(x, 0, 10)
```

Solve the differential equation $f''(x) + 9f(x) = 1$:
```
dsolve(f(x).diff(x, x) + 9*f(x) - 1, f(x))
```

Références

Ouvrages utilisés pour préparer ce cours