

# L1 - Calcul Scientifique



Youssef Chahir  
[youssef.chahir@unicaen.fr](mailto:youssef.chahir@unicaen.fr)

Université de Caen Normandie

1

## Organisation du cours

- Objectif de ce cours :
  - Donner les méthodes permettant de résoudre numériquement des problèmes scientifiques
  - Introduire des librairies Python permettant de faciliter ces calculs
- 10 x 1,25h CM
- 10 TP de 1H15 + 1 TP noté à la fin
- Questions sur les groupes de TP : François Rioult ([francois.rioult@unicaen.fr](mailto:francois.rioult@unicaen.fr))
- Diapositives et sujet de TP sur la plateforme **ecampus** (<https://ecampus.unicaen.fr/>)

2

## Contrôle des connaissances

- Note finale = (Contrôle continu + Contrôle Terminal)
- Contrôle Terminal : épreuve écrite en fin de semestre
- Contrôle continu :
  - 6 exercices notés en fin de séances.
  - 1 TP noté à la fin
- Présence aux TP obligatoire : des exercices notés sont demandés en fin de chaque séance de TP, présence exigée.

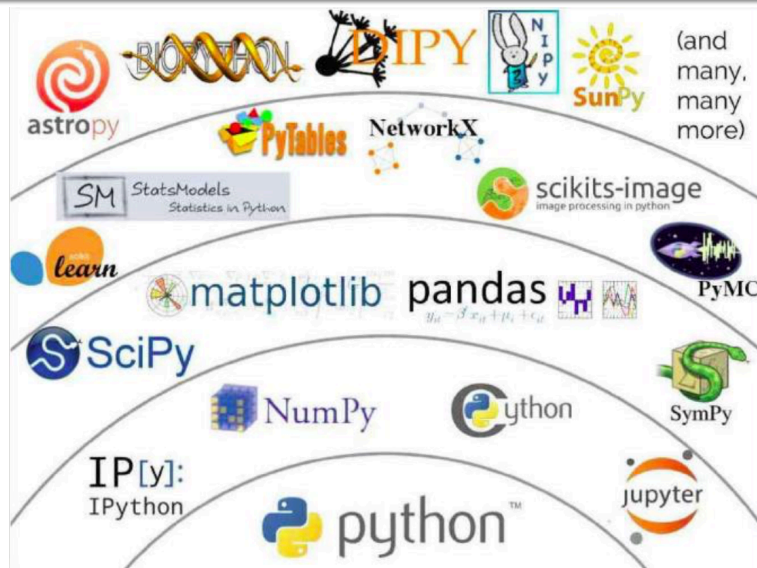
3

## Introduction

- Le calcul scientifique et numérique a pris un essor considérable dans le domaine de l'ingénierie, de la recherche scientifique ou de l'analyse de données (big data)
- Le langage Python est devenu au fil du temps un langage incontournable pour le calcul scientifique
  - Syntaxe claire et facile à lire
  - Python est utilisé comme un langage liant le haut niveau et le bas niveau
  - Trouver le bon compromis entre temps d'exécution et temps de développement
  - Les librairies de calcul sont écrites en langage C, puis utilisées dans Python qui a l'avantage d'être un langage interprété (pas de nécessité de compiler les programmes)
  - Écosystème au sein duquel tout est interopérable
  - Librairies numpy, scipy, matplotlib, etc.

4

## Introduction (suite)



5

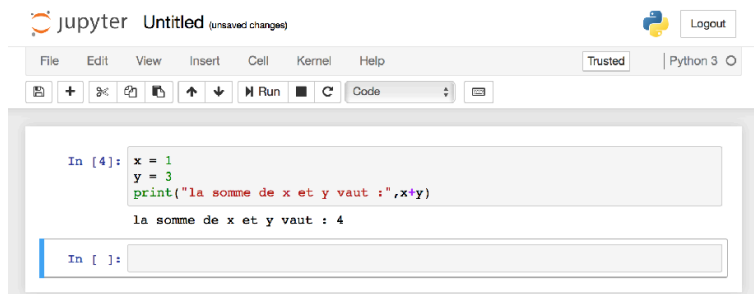
## Notebook Jupyter

- Jupyter est une application Web qui regroupe deux fonctionnalités très différentes :
  - Un outil qui permet de créer des documents multimédia intégrant du texte, des formules mathématiques, des graphiques, des images, voire des animations et des vidéos.
  - Une interface qui permet d'exécuter du code informatique.
    - Pour cela Jupyter s'appuie sur des programmes indépendants capables d'interpréter le langage dans lequel est écrit ce code.
    - Dans la terminologie de Jupyter ces interpréteurs sont appelés des noyaux (kernel en anglais).
- Remarque :** Nous utiliserons le noyau pour le langage Python 3.
- Un document Jupyter est appelé un notebook. Un fichier notebook est reconnaissable par son extension **.ipynb**.
- Le notebook est constitué d'une succession de cellules comportant
  - soit du texte en Markdown
  - soit du code comme dans la cellule suivante (Python pour nous)
- Les cellules peuvent être dans le mode commande ou le mode édition :
  - mode commande : permet de se déplacer d'une cellule à l'autre et d'exécuter les cellules
  - mode édition : permet de modifier le contenu d'une cellule.

7

## Jupyter

- Nous utiliserons pour le cours et les TPs les 'notebooks' Jupyter.
- Pour lancer la session (et un navigateur), saisir la commande : **jupyter lab** ou **jupyter notebook** ou **ipython notebook** soit dans un terminal, soit dans une invite de commande sous Windows. Ou lancer iPython : C:\ProgramData\Anaconda3\Scripts\ipython3
- Pour essayer : <https://try.jupyter.org/> (pour de petits tests uniquement)



6

## Mode commande

- Pour entrer dans le mode commande de la cellule sélectionnée, il suffit de presser la touche Esc.
- Quand vous êtes dans le mode commande, vous pouvez ajouter ou supprimer des cellules mais vous ne pouvez pas saisir de texte dans une cellule.
- Voici les raccourcis principaux disponibles en mode commande :
  - A et B: Insèrent une nouvelle cellule, respectivement au-dessus ou au-dessous de la cellule sélectionnée.
  - M : Transforme la cellule en une cellule de type Markdown.
  - Y: Transforme la cellule en une cellule de type Code.
  - C et V: Copie et colle des cellules.
  - DD: Supprime la cellule sélectionnée.
  - Z: Annule la dernière suppression de cellule.
  - II: Interrompt l'exécution du code.
  - 00: Redémarre l'interpréteur. Il se retrouve alors dans son état initial.
  - H: Affiche la liste de tous les raccourcis clavier.

8

## Mode édition

- Markdown est un système d'édition et de formatage de texte.
- Pour entrer dans le mode édition de la cellule sélectionnée, il suffit de presser la touche Enter ou de double-cliquer à l'intérieur de la cellule. Quand une cellule est en édition vous pouvez saisir du texte comme dans un éditeur classique.
- Lorsque le curseur est en début de ligne ou lorsque vous avez sélectionné du texte, l'appui sur la touche Tab (respectivement Shift TAB) indente (respectivement désindente) les lignes correspondantes.
- Voici d'autres raccourcis clavier :
  - Ctrl A: Sélectionne tout le texte de la cellule.
  - Ctrl Z: Annule les dernières saisies de texte.
  - Ctrl Enter: Exécute la cellule.
  - Shift Enter: Exécute la cellule et sélectionne la cellule suivante. L'appui répété de cette touche permet ainsi d'exécuter pas à pas toutes les cellules du notebook.
  - Alt Enter: Exécute la cellule et insère une nouvelle cellule juste en dessous.
  - ESC: Passe dans le mode commande

9

## Mode édition : la cellule Code

- Pour nous, cellule de code python
- Aide en ligne : commande précédée par ?. Ex : `?abs`
- Commande système : précédée par !. Ex : `!ls -l`
- Commande magiques de IPython : précédée par %
  - `%autosave 300` : Sauvegarde automatiquement le notebook tous les 300 secondes (=5 minutes)
  - `%who`, `%whos` affichent les objets qui sont actuellement définis dans l'interpréteur Python.
  - `%reset` efface les variables définies. `reset -f` efface les variables définies sans demander confirmation.

11

## Mode édition : la cellule Markdown

- Titres: en ajoutant des dièses (#)
- Paragraphe: laisser une ligne vide.
- Les caractères spéciaux ayant un sens en markdown doivent être échappés (précéder avec antislash \).
- Mise en forme:
  - Entourer par un signe unique (\* ou \_ (underscore) passe en *italique* et par un double signe en **gras**.
  - Un double tildes ~~ permettent de barrer le texte.
  - Citation: commencez le paragraphe par >.
  - Liste non ordonnée : commencez la ligne par une astérisque \*, un moins - ou un plus +.
  - Liste ordonnée : commencez la ligne par un nombre suivi d'un point.
  - Bloc de code: sautez deux lignes comme pour un paragraphe, puis indentez avec 4 espaces ou une tabulation. Pour afficher du code dans une ligne, il faut l'entourer par des guillemets simples (').
  - Lien: `<http://www.google.com>` ou `[google](http://www.google.com)`
  - Image:
    - `![jupyter logo](https://upload.wikimedia.org/wikipedia/commons/3/38/Jupyter_logo.svg)`
    - ``
  - Tableau :

```
| Titre1 | Titre2 | Titre3 |
| --- | --- | --- |
| Ligne1 | 1 | A |
| Ligne2 | 22 | B |
```
- Les équations peuvent être directement données au format Latex.
  - formule dans texte entre `$.` . Ex: `$x_{12}$`
  - formule centrée: `$$..$$`. Ex: `$$\sqrt{1+x}$$`

10

## Références

- <https://jupyter.org/>
- [https://jupyter-notebook.readthedocs.io/en/stable/examples/Notebook/examples\\_index.html](https://jupyter-notebook.readthedocs.io/en/stable/examples/Notebook/examples_index.html)
- Tutoriels: <https://www.youtube.com/playlist?list=PLRJx8WOUx5XcDMOxSOegCJUjTJePTIF9Z>

12

## Les modules python utilisés

- **sympy** : module pour le calcul arithmétique formel basique, algèbre, mathématiques différentielles, physique, de mécanique classique ou quantique.
- **numpy** : module pour la manipulation des matrices ou tableaux multidimensionnels et fonctions mathématiques opérant sur les tableaux. Il est la base de **SciPy**, regroupement de bibliothèques Python autour du calcul scientifique;
- **matplotlib** : tracer et visualiser des données sous formes de graphiques. Se combine avec NumPy et SciPy.

13

## Travaux pratiques

1. Introduction aux Notebooks
2. Calcul symbolique (sympy)
3. Prise en main numpy et matplotlib
4. Approfondissement python et matplotlib
5. Recherche de racines
6. Interpolation, ajustement
7. Splines, Intégration, dérivées

14

## Chapitre I : SymPy une bibliothèque Python pour le calcul formel/symbolique

---

## Plan

- Historique
- Aide en ligne
- Données numériques exacts ou approchées
- Les Symboles SymPy
- Les expressions de la bibliothèque SymPy

16

# SymPy

- SymPy est une librairie Python de mathématiques symboliques et un outil de calcul formel.
- SymPy offre un ensemble riche de fonctions documentées qui facilite la modélisation de problèmes mathématiques :
  - Arithmétique, calcul symbolique, résolution d'équations, recherche de racines, dessin de fonctions, limites et séries, calcul différentiel et intégral, algèbre linéaire.
- La librairie offre aussi des fonctionnalités plus avancées en :
  - Mathématiques : Géométrie différentielle et algébrique, intégration numérique, théorie des catégories
  - Physique : Optique, mécanique classique, mécanique et informatique quantique.
- Pour plus d'informations sur SymPy:
  - Aperçu de fonctionnalités de SymPy : <http://www.sympy.org/fr/features.html>
  - Documentation complète sur SymPy (anglais): <http://docs.sympy.org/>

17

# Le module SymPy

- La librairie SymPy permet de faire du calcul symbolique avec Python (équivalent de Maple ou Mathematica).
- SymPy permet de définir des symboles et de faire du calcul sur des expressions contenant des symboles et non des nombres.
  - Utilisation d'expression en Python :

```
x=1
y=x+x+1
print(y)
```

3

```
import sympy
x= sympy.symbols('x')
y= x+x+1
print(y)
```

2\*x + 1

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

- Vous pouvez essayer le module sympy en temps réel en allant sur le site : <http://live.sympy.org/>

19

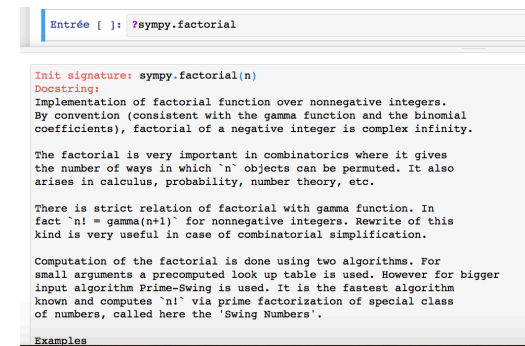
## Bref historique du calcul formel

- 1950 Algorithmes de calcul de dérivée d'une fonction
- 1970 Premiers systèmes de calcul formel : Reduce, Macsyma écrits en LISP
- 1980 Systèmes modernes (avec GUI) : Maple, Mathematica écrits en C
- 2000 Calcul formel dans le libre : sage (<http://www.sagemath.org>) utilise Python (et autres langages) ou SymPy un module de Python

18

## Aide

- Pour obtenir de l'aide sur une fonction  $f$ , il suffit d'écrire **?f** ou **f?** ou **help(sympy.f)**. Le menu propose des tutoriels et de l'aide. Il permet d'accéder aux pages web des différents modules (SymPy, NumPy,...)
- Exemple : **?sympy.factorial** ou **sympy.factorial?** ou **help(sympy.factorial)**



Entrée [ ]: ?sympy.factorial

```
Init signature: sympy.factorial(n)
Docstring:
Implementation of factorial function over nonnegative integers.
By convention (consistent with the gamma function and the binomial
coefficients), factorial of a negative integer is complex infinity.

The factorial is very important in combinatorics where it gives
the number of ways in which 'n' objects can be permuted. It also
arises in calculus, probability, number theory, etc.

There is strict relation of factorial with gamma function. In
fact 'n! = gamma(n+1)' for nonnegative integers. Rewrite of this
kind is very useful in case of combinatorial simplification.

Computation of the factorial is done using two algorithms. For
small arguments a precomputed look up table is used. However for bigger
input algorithm Prime-Swing is used. It is the fastest algorithm
known and computes 'n!' via prime factorization of special class
of numbers, called here the 'Swing Numbers'.
```

Exemples

20

# Numérique

## ► Déclarer ces variables

```
In[2] : t
(...)
NameError : name 't' is not defined

In[3] : t = symbols('t'); type(t)
Out[4] : sympy.core.symbol.Symbol

In[5] : x, y = Symbols('x y'); expr = x**2 + y**2
```

## ► Valeur numérique

```
In[7] : math.sqrt(20)
Out[8] : 4.47213595499958

In[9] : float(sqrt(20))
Out[10] : 4.47213595499958
```

## ► Valeur symbolique

```
In[11] : import sympy
In[12] : sympy.sqrt(20)
Out[13] : 2*sqrt(5)
```

!1

# Nombre rationnel

- On construit des nombres rationnels à l'aide de la fonction [Rational](#)
- Une autre façon plus courte est d'utiliser la fonction raccourcie [S](#) qui traduit des types de Python comme une chaîne de caractères ou un entier en des nombres entiers ou rationnels de SymPy.

```
: sympy.Rational(4,6) | : sympy.S("5/2")

2/3 | : sympy.S(5)/2 | : sympy.S("5/2")
# traduction d'une chaîne de
# caractères en un rationnel

5/2 | : sympy.S(5)/2
# S(5) transforme 5 (type int)
# en un nombre entier de sympy

: sympy.Rational(1,4) + sympy.Rational(1,3)

7/12
```

23

# Affichage de résultats exacts ou approchés

- Avec SymPy on travaille avec des valeurs exactes :

```
12**100
828179745220145502584084235957368498016122811853894435464201864103254919330121223037770283296858019385573376

: sympy.sqrt(3)
sqrt(3)
```

- On peut obtenir une valeur approchée avec les fonctions `.n()` ou `.evalf()` :

```
: sympy.sqrt(3).n() | : sympy.sqrt(3).evalf()
1.73 | : sympy.sqrt(3).n() | : sympy.sqrt(3).evalf()
1.73205080756888 | : sympy.sqrt(3).n() | : sympy.sqrt(3).evalf()
1.73205080756888
```

- Quelques constantes pi, e (tapé E), i (tapé I), l'infini (oo)

```
: sympy.pi | : sympy.E | : sympy.I | : sympy.oo
pi | e | i | infinity
```

22

# Les symboles

- Les symboles sont les éléments de base du calcul algébrique ou calcul littéral
- La définition d'un symbole se fait avec la fonction `symbols` : `x=sympy.Symbol('x')` ou `x = sympy.symbols('x')`
- Attention à bien distinguer le nom du symbole et celui de la variables : La variable x (à gauche) contient le symbole x (à droite)
- Possibilité de définir plusieurs symboles d'un coup : `x,y = sympy.symbols('x,y')`
- 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)`

24

## Les symboles (suite)

from sympy import \*

- Pour définir un **symbol** **x**, on utilise la commande: **Symbol**  
`x = Symbol('x')` # notez le S en majuscule  
ou  
`import sympy as sy`  
`x = sy.Symbol('x')`

- Plusieurs symbols sympy en même temps, on utilise la commande:  
**symbols**  
`x, y, z = symbols('x y z')` # notez le 's' en minuscule
- dir(x)**: commande python qui vous donne un aperçu des  
\_\_méthodes\_\_ que possède l'objet (ici le symbole) \$x\$.

25

## Les symboles (suite)

- Nom du symbole contenu dans une variable :

```
1 x = sympy.symbols("x")
2 x.name
'x'
```

```
1 u = sympy.symbols("x")
2 u.name
'x'
```

- Utilisation des symboles dans des calculs arithmétiques simples :

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

```
1 z
x*y + 2*x
```

27

## Les symboles (suite)

- La définition d'un symbole se fait aussi avec la fonction **Symbol**
- On peut vérifier le type d'un symbole

```
: x = sympy.Symbol('x', real=True)
: x.is_imaginary
False
: y = Symbol('y', positive=True)
: y>0
True
```

```
: n = sympy.Symbol('n')
: sympy.Sum(1/n**2, (n, 1, 10))
```

$$\sum_{n=1}^{10} \frac{1}{n^2}$$

26

## Les symboles (suite)

- Sympy fait automatiquement des simplifications dans des cas simples :

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

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

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

$(x + 2)(x + 3)$

```
z=x**3 + 3*x**2 + 3*x
z
```

$x^3 + 3x^2 + 3x$

```
print(z)
```

$x**3 + 3*x**2 + 3*x$

28



## Factorisation et développement

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

<pre>1 x,y=sympy.symbols('x,y') 2 z = (x+2)*(x+3) 3 z = z.expand() 4 print(z)</pre>	OU	<pre>1 x,y=sympy.symbols('x,y') 2 z = (x+2)*(x+3) 3 z = sympy.expand(z) 4 print(z)</pre>
$x^2 + 5x + 6$		$x^2 + 5x + 6$

- Et une autre permettant de factoriser : `factor`

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

$x^2 + 5x + 6 \rightarrow (x + 2)(x + 3)$

Si pas de factorisation possible, l'expression reste identique

29

## Affichage d'une expression

```
e= sp.sqrt(2)
print("avec print :")
print("e=",e)

print("\navec sp.pprint :")
sp.pprint(e)

print("\navec display :")
display(e)
```

avec print :  
e= sqrt(2)

avec sp.pprint :  
 $\sqrt{2}$

avec display :  
 $\sqrt{2}$

31

## Affichage d'une expression

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

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

$x^2 + 5x + 6 \rightarrow (x + 2)(x + 3)$   
 $x^2 + 5x + 6$

- La fonction `init_printing()` permet aussi de visualiser les résultats de manière plus jolie
- Les termes sont arrangés par ordre des puissances décroissantes par défaut, mais cela peut être modifié (inversé)

30

## Construction d'une série

- Il est possible de construire des fonctions littérales en utilisant des structures itératives
- Par exemple, construisons la série :  $x + \frac{x^2}{2} + \frac{x^3}{3} + \frac{x^4}{4} + \dots + \frac{x^n}{n}$ .
- Pour cela, nous pouvons écrire la fonction `serie(n)` où n est la valeur du dernier indice de la série :

```
: def serie(n):
    x=sympy.symbols('x')
    s=x
    for i in range(2,n+1):
        s=s+x**i/i
    return(s)
print(serie(5))
sympy.pprint(serie(5))
```

$x^5/5 + x^4/4 + x^3/3 + x^2/2 + x$   
 $\frac{x^5}{5} + \frac{x^4}{4} + \frac{x^3}{3} + \frac{x^2}{2} + x$

32



## Substitution de valeurs

- Pour remplacer les symboles par des valeurs numériques, on utilise la fonction `subs`

Evaluation pour  $x=1$  et  $y=2$  :

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

$$x^2 + 2x + y^2$$

Substitution partielle :

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

7

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

$y^2 + 3$

- L'argument de la fonction `subs` est un *dictionnaire* python<sub>33</sub>

## Substitution de valeurs (suite)

- Il est également possible de substituer des symboles par d'autres symboles ou expressions.
- Par exemple, si  $x = 1 - y$  alors nous pouvons écrire :

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

$$x^2 + 2x + y^2$$

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

$y^2 - 2y + (-y + 1)^2 + 2$

## Simplification des expressions

- Sympy possède une fonction permettant de simplifier des expressions : `simplify`
- Par exemple, dans l'exemple précédent, le résultat peut être simplifié :

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

$2y^2 - 4y + 3$

- Attention de bien utiliser les fonctions mathématique de sympy
- Sympy possède aussi un parser permettant de saisir les expressions :

```
1 e = input('entrer une expression mathématique :')
2 es = sympy.sympify(e)
3 2*es
```

entrer une expression mathématique :  $3x^2 - 7x + 3$

$6x^2 - 14x + 6$

## Résolution d'équations

- La fonction `solve(expr)` permet de trouver les solutions de l'expression `expr` et retourne une liste car plusieurs valeurs peuvent être solutions.
- Par défaut, `solve` s'attend à résoudre des expressions de type  $f(x)=0$
- Exemple :

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

$[-4, -1]$

- Les solutions peuvent être des complexes :

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

$[-1/2 - \sqrt{3}I/2, -1/2 + \sqrt{3}I/2]$

## Résolution d'équations (suite)

- La fonction `Eq(expr1,expr2)` permet de résoudre une équation de type égalité complète ( $\text{expr1}=\text{expr2}$ ):

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

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

- On peut 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)}]
```

37

## Résolution de systèmes d'équations

- Supposons que nous cherchons les solutions de :

$$2x + 3y = 6$$

$$3x + 2y = 12$$

- Nous pouvons écrire :

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

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

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

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

0

39

## 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  $b$ .

```
1 x,a,b,c = sympy.symbols('x,a,b,c')
2 expr = a*x**2 + b*x + c
3 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 :

```
1 x,a,b,c = sympy.symbols('x,a,b,c')
2 expr = a*x**2 + b*x + c
3 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)}]
```

38

## Synthèse : Expression sympy

- On peut faire des opérations sur les expressions comme factorisation, développement, simplification ...

### 1. Développer une expression à l'aide de la méthode `expand()`

```
expr = (x - y)**3
print(expr.expand()) #affiche: x**3 - 3*x**2*y + 3*x*y**2 - y**3
```

### 2. Simplification via la méthode `simplify()`

```
expr1 = (x + 1)**2
expr2 = x**2 + 2*x + 1
expr3 = sy.simplify( expr2 - expr1 )
print("expr2 - expr1 = ", expr3) # affiche: expr2 - expr1 = 0
```

### 3. Factoriser une expression à l'aide de la méthode `factor()`

```
P = x**2 - 2*x + 1
fact = sy.factor(P)
print(fact) # affiche: (x - 1)**2
```

40

# Expressions sympy

## 4. Evaluation d'une expression à l'aide de la méthode evalf()

```
print(pi.evalf()) # affiche: 3.14159265358979
print(pi.evalf(3)) # affiche: 3.14
```

## 5. Substitution des symbols dans sympy grâce à la méthode subs().

```
expr = x**2 + x + 3
# substitution de x par 1
print(expr.subs(x, 1)) # affiche 5
expr = x**2 + x*y + 3*x + y
# substitution de x par y (remplacement de x par y)
print(expr.subs(x, y)) # affiche: 2*y**2 + 4*y
```

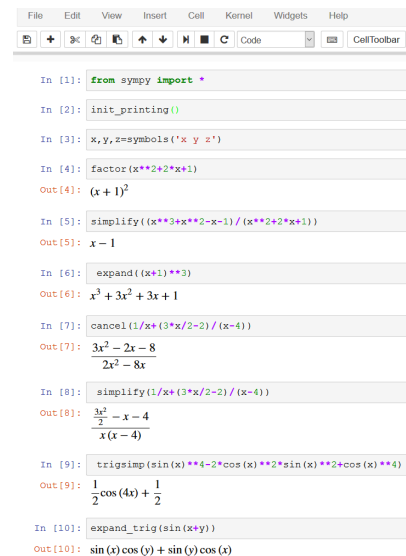
## 6. Résoudre une équation avec la méthode sympy.solve()

```
# résoudre l'équation: x**2 - 3 = 0
print(sy.solve(sy.Eq(x**2, 3))) # affiche: [-sqrt(3), sqrt(3)]
```

41

# Jupyter (application web)

Sur l'image ci-dessous, on voit que l'affichage des résultats fournis par sympy est très lisible



43

# Exemple: polynôme

```
1 P=x**3+4*x**2+5*x+2
2 display(P)
3 solve(P,x)
```

$$x^3 + 4x^2 + 5x + 2$$
$$[-2, -1]$$

```
1 factor(P)
```

$$(x + 1)^2 (x + 2)$$

```
1 f=(x**2+2*x+3)/(x**3+4*x**2+5*x+2)
2 display(f)
3 apart(f)
```

$$\frac{x^2 + 2x + 3}{x^3 + 4x^2 + 5x + 2}$$
$$\frac{3}{x+2} - \frac{2}{x+1} + \frac{2}{(x+1)^2}$$

On peut la décomposer en somme de fractions rationnelles à l'aide de la fonction apart de SymPy

42

## SymPy Cheatsheet (<http://sympy.org>)

<b>Basics</b> SymPy help: <code>help(function)</code> Declare symbol: <code>x = Symbol('x')</code> Substitution: <code>expr.subs(old, new)</code> Numerical evaluation: <code>expr.evalf()</code> Expanding: <code>expr.expand()</code> Common denominator: <code>ratsimp(expr)</code> Simplify expression: <code>simplify(expr)</code>	<b>Geometry</b> Points: <code>a = Point(xcoord, ycoord)</code> Lines: <code>l = Line(pointA, pointB)</code> Circles: <code>c = Circle(center, radius)</code> Triangles: <code>t = Triangle(a, b, c)</code> Areas: <code>object.area</code> Intersection: <code>intersection(a, b)</code> Checking tangency: <code>c.is_tangent(l)</code>	<b>Examples</b> Find 100 digits of $\pi^*$ : <code>(pi**E).n(100)</code> Expand $(x+y)^2(x-y)(x^2+y)$ : <code>((x+y)**2 * (x-y) * (x**2+y)).expand()</code> Simplify $\frac{1}{2} + \frac{x \sin x - 1}{x^2 - 1}$ : <code>simplify(1/2 + (x*sin(x) - 1)/(x**2 - 1))</code> Check if line passing through points (0, 1) and (1, 1) is tangent to circle with center at (5, 5) and radius 3: <code>Circle(Point(5,5), 3).is_tangent(Line(Point(0,1), Point(1,1)))</code> Find roots of $x^4 - 4x^3 + 2x^2 - x = 0$ : <code>solve(x**4 - 4*x**3 + 2*x**2 - x, x)</code> Solve the equations system: $x + y = 4, xy = 3$ : <code>solve([x + y - 4, x*y - 3], [x, y])</code> Calculate limit of the sequence $\sqrt[n]{n}$ : <code>limit(n**(1/n), n, oo)</code> Calculate left-sided limit of the function $\frac{\ln x}{x}$ in 0: <code>limit(abc(x)/x, x, 0, dir='-')</code> Calculate the sum $\sum_{n=0}^{100} n^2$ : <code>summation(n**2, (n, 0, 100))</code> Calculate the sum $\sum_{n=0}^{\infty} \frac{1}{2^n}$ : <code>summation(1/n**2, (n, 0, oo))</code> Calculate the integral $\int \cos^3 x dx$ : <code>integrate(cos(x)**3, x)</code> Calculate the integral $\int_0^{\infty} \frac{e^{-x}}{x} dx$ : <code>integrate(1/x**2, (x, 1, oo))</code> Find 10 terms of series expansion of $\frac{1}{1-2x}$ at 0: <code>(1/(1-2*x)).series(x, 0, 10)</code> Solve the differential equation $f''(x) + 9f(x) = 1$ : <code>dsolve(f(x).diff(x, 2) + 9*f(x) - 1, f(x))</code>
<b>Constants</b> $\pi$ : <code>pi</code> $e$ : <code>E</code> $\infty$ : <code>oo</code> $i$ : <code>I</code>	<b>Numbers types</b> Integers (Z): <code>Integer(x)</code> Rationals (Q): <code>Rational(p, q)</code> Reals (R): <code>Float(x)</code>	
<b>Basic functions</b> Trigonometric: <code>sin cos tan cot</code> Cyclometric: <code>asin acos atan acot</code> Hyperbolic: <code>sinh cosh tanh coth</code> Area hyperbolic: <code>asinh acosh atanh acoth</code> Exponential: <code>exp(x)</code> Square root: <code>sqrt(x)</code> Logarithm (log, a): <code>log(a, b)</code> Natural logarithm: <code>log(a)</code> Gamma (Γ(x)): <code>gamma(x)</code> Absolute value: <code>abs(x)</code>	<b>Plotting</b> Plot: <code>Plot(f, [a, b])</code> Zoom: <code>+/ -</code> 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	
<b>Calculus</b> $\lim_{x \rightarrow a} f(x)$ : <code>limit(f, x, a)</code> $\lim_{x \rightarrow \infty} f(x)$ : <code>limit(f, x, a, dir='+')</code> $\lim_{x \rightarrow -\infty} f(x)$ : <code>limit(f, x, a, dir='-')</code> $\frac{d}{dx} f(x)$ : <code>diff(f, x)</code> $\frac{d^2}{dx^2} f(x, y)$ : <code>diff(f, x)</code> $\int f(x) dx$ : <code>integrate(f, x)</code> $\int_a^b f(x) dx$ : <code>integrate(f, (x, a, b))</code> Taylor series (at a, deg n): <code>f.series(x, a, n)</code>	<b>Discrete math</b> Factorial (n!): <code>factorial(n)</code> Binomial coefficient $\binom{n}{k}$ : <code>binomial(n, k)</code> Sum $\sum_{n=a}^b expr$ : <code>summation(expr, (n, a, b))</code> Product $\prod_{n=a}^b expr$ : <code>product(expr, (n, a, b))</code>	
<b>Equations</b> Equation $f(x) = 0$ : <code>solve(f, x)</code> System of equations: <code>solve([f, g], [x, y])</code> Differential equation: <code>dsolve(equation, f(x))</code>	<b>Linear algebra</b> Matrix definition: <code>m = Matrix([[a, b], [c, d]])</code> Determinant: <code>m.det()</code> Inverse: <code>m.inv()</code> Identity matrix $n \times n$ : <code>eye(n)</code> Zero matrix $n \times n$ : <code>zeros(n)</code> Ones matrix $n \times n$ : <code>ones(n)</code>	
	<b>Printing</b> l <sup>A</sup> T <sub>E</sub> X print: <code>print latex()</code> Python print: <code>print python()</code> Pretty print: <code>pprint()</code>	

44