

SymPy par la pratique

Exemple et exercice avancée

K.I.A Derouiche



Contents

0.1	Avant-Propos	9
0.2	Calcul formel	11
0.2.1	Logiciel de système de calcul formel	11
0.2.2	Quelques logiciels de calcul formel	11
0.2.3	Pourquoi choisir SymPy?	12
I	Premier pas vers SymPy	13
1	Premier pas vers SymPy	15
1.1	La bibliothèque SymPy	15
1.1.1	Le cas de la bibliothèque SymPy	15
1.1.2	Travaillez avec SymPy	16
1.1.3	Installation de SymPy	16
1.1.4	isympy	16
1.1.5	SymPyGamma	16
1.1.6	SymPyLive	16
1.2	SymPy comme calculatrice	16
1.2.1	Premier calculs	17
1.2.2	Structure de données dans SymPy	17
1.2.3	Variable et affectation	18
1.2.4	Substitution	18
1.2.5	Contrôle du flux d'instructions	18

2	Analyse et Algèbre	19
2.1	Expressions symboliques et simplification	19
2.1.1	Expressions symboliques	19
2.1.2	Transformation d'expressions	19
2.1.3	Fonctions mathématiques usuelles	19
2.2	Équations	19
2.2.1	Résolution explicite	20
2.2.2	Équations sans solution explicite	20
2.3	Inéquations	20
2.4	Analyse	20
2.4.1	Les Fonctions	20
2.4.2	Sommes	21
2.4.3	Limites	21
2.4.4	Suites	21
2.4.5	Développements limités	21
2.4.6	Séries	21
2.4.7	Dérivation	21
2.4.8	Dérivées partielles	21
2.4.9	Intégration	21
2.5	Calcul matriciel	21
2.5.1	Résolution de systèmes linéaires	21
2.5.2	Calcul vectoriel	21
2.5.3	Calcul matriciel	21
2.5.4	Réduction d'une matrice carrée	21
3	Graphiques	23
3.1	Courbes en 2D	23
3.1.1	Représentation graphique de fonctions	23
3.1.2	Courbe paramétrée	23
3.1.3	Courbe en coordonnées polaires	23
3.1.4	Courbe définie par une équation implicite	23
3.1.5	Tracé de données	23
3.1.6	Tracé de solution d'équation différentielle	23
3.1.7	Développée d'une courbe	23
3.2	Courbes en 3D	23
II	Logique, ensemble et catégorie	25
4	Logique	27

5	Théorie des ensemble	29
5.1	Un petit rappel pour les ensembles dans Python	29
5.2	Théorie des ensembles	29
5.2.1	Logique	30
5.2.2	Ensembles	30
6	Théorie des catégories	33
6.1	Note historique	33
6.2	Introduction	33
6.2.1	Quels applications pour l'informatiques	34
III	Algèbre et théorie des nombres	35
7	Anneaux et corps finis	37
7.1	Anneau des entiers modulo n	37
7.2	Corps finis	37
7.3	Quelques problèmes élémentaires de théorie des nombres	37
7.3.1	Théorème de Wilson	37
8	Polynômes	39
8.1	Anneaux et polynômes	39
8.1.1	Introduction	39
8.1.2	Construction d'anneaux de polynômes	39
8.2	Polynômes	39
8.2.1	Création et arithmétique de base	39
8.2.2	Vue d'ensemble des opérations sur les polynômes	39
8.2.3	Changement d'anneau	39
8.2.4	Itération	40
8.3	Arithmétique euclidienne	40
8.3.1	Divisibilité	40
8.3.2	Idéaux et quotients	40
8.3.3	Idéaux	40
8.4	Factorisation et racines	40
8.4.1	Factorisation	40
8.4.2	Recherche de racines	40
8.4.3	Résultant	40
8.4.4	Groupe de Galois	40
8.5	Fractions rationnelles	40
8.5.1	Construction et propriétés élémentaires	40

8.6 Séries formelles	40
8.6.1 Opérations sur les séries tronquées	41
8.6.2 Développement de solutions d'équations	41
9 Algèbre linéaire	43
 9.1 Constructions et manipulations élémentaires	43
9.1.1 Espaces de vecteurs, de matrices	43
9.1.2 Construction des matrices et des vecteurs	43
9.1.3 Manipulations de base et arithmétique sur les matrices	43
9.1.4 Opérations de base sur les matrices	44
 9.2 Calculs sur les matrices	44
9.2.1 Élimination de Gauss, forme échelonnée	45
9.2.2 Résolution de systèmes ; image et base du noyau	45
9.2.3 Valeurs propres, forme de Jordan et transformations de similitude	45
10 Systèmes polynomiaux	49
 10.1 Polynômes à plusieurs indéterminées	49
10.1.1 Les anneaux $A[x_1, \dots, x_n]$	49
10.1.2 Polynômes	50
 10.2 Systèmes polynomiaux et idéaux	50
10.2.1 Un premier exemple	50
10.2.2 Qu'est-ce que résoudre ?	50
10.2.3 Idéaux et systèmes	51
 10.3 Bases de Gröbner	51
10.3.1 Ordres monomiaux	51
10.3.2 Division par une famille de polynômes	51
10.3.3 Propriétés des bases de Gröbner	51
11 Équations différentielles	53
 11.1 Équations différentielles	53
11.1.1 Introduction	53
11.1.2 Équations différentielles ordinaires d'ordre 1	54
11.1.3 Équations d'ordre 2	54
11.1.4 Transformée de Laplace	54
11.1.5 Systèmes différentiels linéaires	54
12 suites définies par une relation de récurrence	55
 12.1 Suites définies par $u_{n+1} = f(u_n)$	55
 12.2 Suites récurrentes linéaires	55
 12.3 Suites récurrentes « avec second membre »	55

IV Géométrie	57
13 Géométrie plan	61
14 Géométrie Différentielle	63
15 Groupe de Lie	65
15.1 Résolution d'équations par groupe de Lie	65
16 Convexity	67
16.0.1 Cone	67
16.1 Convex Functions	68
16.1.1 Epigraph	68
17 Support Function	71
17.1 Operations Preserve Convexity of Functions	72
17.2 Remarks	73
17.3 Corollaries	73
17.4 Propositions	73
17.4.1 Several equations	73
17.4.2 Single Line	74
17.4.3 Paragraph of Text	74
17.5 Exercises	74
17.6 Problems	74
17.7 Vocabulary	74
V Calcul numérique et discret	75
18 Nombres à virgule flottante	77
19 Intégration numérique	79
19.1 Examples	79
19.1.1 Equation and Text	79
20 Équations non linéaires	81
20.1 Équations algébriques	81
20.2 Figure	82

VI Combinatoire	83
21 Permutations	87
22 Semi-groupe numérique	89
22.1 Introduction	89
VII Physique	91
23 Chaos	95
23.1 Pendule simple	95
23.1.1 Pendule à deux bras	95
23.1.2 Mouvements d'un robot	95
24 Mécanique et information quantique	97
25 Le modèle ϕ^4	99
25.0.1 LES DIAGRAMMES DE FEYNMAN	99
VIII Annexe	101
26 Programmation Orientée Objet	103
27 Décorateurs	105
28 Optimisation du code	107
28.0.1 Cython	108
28.0.2 Theano	109
29 Interface graphique	111
29.1 Bibliographie	111
29.2 Index	111

0.1 Avant-Propos

Ce livre traite de SymPy, une bibliothèque de calcul symbolique entièrement écrite en Python un langage de programmation de haut niveau, orienté objet, totalement libre, conçu pour produire du code de qualité, portable et facile à intégrer. Ainsi la conception d'un programme scientifique ou symbolique avec SymPy et Python est très rapide et offre au développeur une bonne productivité. En tant que bibliothèque pythonienne elle repose sur un langage dynamique, très souple d'utilisation et constitue un complément idéal à des langages compilés. Elle reste une bibliothèque complète et autosuffisante, pour des petits scripts fonctionnels de quelques lignes, comme pour des applicatifs complexes de plusieurs centaines de modules.

Pourquoi ce livre ?

Il n'existe pas beaucoup d'ouvrages qui traitent du calcul symbolique en générale par rapport aux calculs numériques ou des ouvrages consacrés aux bibliothèques symboliques écrites en Python. En particulier, gravitent autour de SymPy mis à part un livre de 50 pages, quelques chapitres ou des lignes de codes cités à titre d'exemples. Citons le livre de référence de Svein Linge et Hans Petter Langtangen *Programming for Computations – Python A Gentle Introduction to Numerical Simulations with Python*, aux éditions Springer, ou encore une version du livre de 50 pages *Instant SymPy Starter* de Ronan Lamy, aux éditions Packt Publishing Limited. Le livre est *Instant SymPy Starter* de Ronan Lamy, c'est un guide de démarrage rapide, La documentation en ligne de SymPy est bonne, mais il serait plus facile de commencer avec ce livre. Alors, pourquoi ce livre ?

Si ce livre présente comme celui de Ronan Lamy les notions de la bibliothèque, celui-ci ajoute des exemples originaux, des choix dans la présentation des classes, et une approche globale particulière et détaillée, il tente également d'ajouter à ce socle des éléments qui participent de la philosophie de la programmation en Python scientifique, aller plus loin dans le développement non scientifique, mettre en valeur l'intérêt et l'importance, à savoir :

- des conventions de codage ;
- combiné l'approche symbolique et numérique ;
- des bonnes pratiques de programmation et des techniques d'optimisation ;

Même si chacun de ces sujets pourrait à lui seul donner matière à des ouvrages entiers, les réunir dans un seul et même livre contribue à fournir une vue complète de ce qu'un développeur d'application scientifique en particulier et Python averti et son chef de projet mettent en œuvre quotidiennement.

A qui s'adresse l'ouvrage?

Cet ouvrage s'adresse bien sûr aux développeurs de tous horizons mais également aux étudiants, chercheurs, enseignants et chefs de projets. Ils ne trouveront pas dans ce livre de bases de programmation; une pratique minimale préalable est indispensable de Python, quel que soit le langage utilisé. Il n'est pour autant pas nécessaire de maîtriser la programmation orientée objet et la connaissance d'un langage impératif est suffisante. Les développeurs Python débutants – ou les développeurs avertis ne connaissant pas encore cette bibliothèque – trouveront dans cet ouvrage des techniques et sujets avancés, les patterns efficaces et l'application de certains design patterns objet, topologie, théorie des catégories, machine learning. Les étudiants et enseignants trouveront un ouvrage ouvert sur l'apprentissage par l'exercice résolus et une interprétation d'exercices mathématiques les chercheurs trouveront un outil léger et efficace à travers des approches poussées liées aux questions récentes en connections avec les mathématiques pures, appliquées et la physique

théorique. Les chefs de projets trouveront des éléments pratiques pour augmenter l'efficacité de leurs équipes pluridisciplinaires, notamment la présentation des principaux modules à la fois issues de la bibliothèque standard, graphique et numérique.

Indication

Quand un théorème est exposé ce dernier sera démontrai

0.2 Calcul formel

Le calcul formel, ou parfois calcul symbolique, est le domaine des mathématiques et de informatique qui s'intéresse aux algorithmes opérant sur des objets de nature mathématique par le biais de représentations finies et exactes. Ainsi, un nombre entier est représenté de manière finie et exacte par la suite des chiffres de son écriture en base 2. Étant données les représentations de deux nombres entiers, le calcul formel se pose par exemple la question de calculer celle de leur produit.

Le calcul formel est en général considéré comme un domaine distinct du calcul scientifique, cette dernière appellation faisant référence au calcul numérique approché à l'aide de nombres en virgule flottante, là où le calcul formel met l'accent sur les calculs exacts sur des expressions pouvant contenir des variables ou des nombres en précision arbitraire (en). Comme exemples d'opérations de calcul formel, on peut citer le calcul de dérivées ou de primitives, la simplification d'expressions, la décomposition en facteurs irréductibles de polynômes, la mise sous formes normales de matrices, ou encore la résolution des systèmes polynomiaux.

Sur le plan théorique, on s'attache en calcul formel à donner des algorithmes avec la démonstration qu'ils terminent en temps fini et la démonstration que le résultat est bien la représentation d'un objet mathématique défini préalablement. Autant que possible, on essaie de plus d'estimer la complexité des algorithmes que l'on décrit, c'est-à-dire le nombre total d'opérations élémentaires qu'ils effectuent. Cela permet d'avoir une idée a priori du temps d'exécution d'un algorithme, de comparer l'efficacité théorique de différents algorithmes ou encore éclairer la nature même du problème.

0.2.1 Logiciel de système de calcul formel

Dans cette section en va exposer les systèmes de calcul formel, leur intérêt qui à vue un renouveau ces dernières années à cause de l'émergence de technique, technologie et nouvelle approche de programmation pour le domaine scientifique et industriel, hormis le fait que le logiciel de calcul formel en soient sont un outil pédagogique indispensable pour les scientifiques et les ingénieurs

Definition 0.2.1 Un logiciel de système formel est un outil qui facilite le calcul symbolique. La partie principale de ce système est la manipulation des expressions mathématiques sous leur forme symbolique.

■ **Example 0.1** soit $G = \{x \in \mathbb{R}^2 : |x| < 3\}$ et noté par: $x^0 = (1, 1)$; en considère la fonction:

$$f(x) = \begin{cases} e^{|x|} & \text{si } |x - x^0| \leq 1/2 \\ 0 & \text{si } |x - x^0| > 1/2 \end{cases} \quad (1)$$

The function f has bounded support, we can take $A = \{x \in \mathbb{R}^2 : |x - x^0| \leq 1/2 + \varepsilon\}$ for all $\varepsilon \in]0; 5/2 - \sqrt{2}[$. ■

cet exemple se traduit en forme symbolique avec la bibliothèque SymPy:

0.2.2 Quelques logiciels de calcul formel

qui exprime ce qui nous permet notre choix pour un CAS qui possède des caractéristiques techniques et sur le plan du coût très important quand peut résumer dans les points suivants:

1. Leger et
2. S'appuie sur le langage de programmation Python
3. Portabilité dans toute transparence

L'un des systèmes qui peut nous permettre d'écrire cette exemple avec un ordinateurs avec SymPy qui semble mieux intégré

0.2.3 Pourquoi choisir SymPy?

Part I

Premier pas vers SymPy

1. Premier pas vers SymPy

Ce chapitre d'introduction présente la tournure d'esprit de la bibliothèque mathématique SymPy. Les autres chapitres de cette partie développent les notions de base de SymPy: effectuer des calculs numériques ou symboliques en analyse, opérer sur des vecteurs et des matrices, écrire des programmes, manipuler des listes de données, construire des graphiques, etc. Les parties suivantes de cet ouvrage approfondissent quelques branches des mathématiques dans lesquelles l'informatique fait preuve d'une grande efficacité.

1.1 La bibliothèque SymPy

1.1.1 Le cas de la bibliothèque SymPy

Dans un cas plus simple l'exemple 1.1 se formule beaucoup plus dans un outil comme SymPy est une bibliothèque de calcul formel elle est aussi un environnement pour l'apprentissage de l'algèbre, l'analyse, géométrie, combinatoire, cryptographie, mécanique classique et quantique pour le lycée et l'université mais aussi un environnement de développement et de recherche. SymPy écrit entièrement en Python un langage de programmation facile à apprendre et adapté à l'apprentissage, elle fournit aux étudiant *SymPyGamma* une application web notamment des primitives générales de traitement des expressions algébriques (développement, factorisation, ...), des aides à l'organisation des objets mathématiques intervenant dans la résolution d'un problème ainsi qu'une assistance à la preuve. Il permet au professeur de préparer et de suivre le travail de l'élève. Différentes maquettes ont été développées et testées auprès d'élèves. Dans la plus récente, nous nous sommes attachés à explorer une nouvelle forme d'activité algébrique. Alors que le calcul en papier crayon et les logiciels standards considèrent les expressions de façon isolée, l'environnement que nous développons organise en réseau les différentes expressions intervenant dans la résolution d'un problème. L'ordinateur peut facilement mettre à jour ce réseau quand l'utilisateur modifie certains de ses éléments. Il devient ainsi possible, pour aborder un problème générique, d'explorer facilement des cas particuliers et de conduire une généralisation. Les relations entre expressions algébriques sont mieux mises en évidence du fait de leur invariance dans les modifications du réseau. De façon très concise, Casyopée peut être défini

1.1.2 Travaillez avec SymPy

1.1.3 Installation de SymPy

1.1.4 isympy

```
IPython console for SymPy 1.3 (Python 3.6.7-64-bit) (ground types: python)
These commands were executed:
>>> from __future__ import division
>>> from sympy import *
>>> x, y, z, t = symbols('x y z t')
>>> k, m, n = symbols('k m n', integer=True)
>>> f, g, h = symbols('f g h', cls=Function)
>>> init_printing()
```

Documentation can be found at <http://docs.sympy.org/1.3/>

1.1.5 SymPyGamma

Est une interface onWeb marche avec un navigateur contient plusieurs catégorie liée de calcul, dynamique. L'Intérêt de cette outil qu'il est facilement partageable adapté pour l'enseignement et surtout l'auto-apprentissage



1.1.6 SymPyLive

SymPy Live est SymPy qui s'exécute sur Google App Engine. Ceci est juste un shell Python standard, avec les commandes suivantes exécutées par défaut

1.2 SymPy comme calculatrice

Contrairement à Sage[], Maple, Octave et les autres logiciel de calcul formel, SymPy, effectue des calculs directe numériques de deux manière différents en passant par la méthode,

1.2.1 Premier calculs

Dans la suite du livre, nous présentons les calculs sous la forme suivante, qui imite l'allure d'une session de SymPyLive à travers la ligne de commande isympy:

```
In [1]: from sympy import *
In [2]: 1+1
Out[2]: 2
```

Variables Python

Lorsque l'on veut conserver le résultat d'un calcul, on peut l'affecter à une variable :

Variables Symboliques

Les objets mathématiques manipulés par SymPy sont symboliques ils sont représentés exactement loin de toute approximation numérique, SymPy permet une manipulation avec des expressions contenant des variables, comme $x^2 + zy^3 + z^2$ ou encore $\sin(x) - \exp(x)$. Les variables symboliques du mathématicien x, y, z apparaissant dans ces expressions diffèrent, avec SymPy, des variables du développeur $\sin(2) = 0.9092974268256817$ que nous manipulons sous Python section précédente. SymPy diffère notamment, sur ce point, d'autres systèmes de calcul formel comme Maple ou Maxima, Sage c'est inspiré de SymPy sur ce point.

La documentation officiel présente la différence entre valeur numérique gérer par la bibliothèque standard Python math à travers l'exemple de la racine carré $\sqrt{8}$ sans évaluation, posons $x = 8$

```
In [1]: import math
In [2]: math.sqrt(x)
Out[2]: 2.8284271247461903
```

Les variables symboliques doivent être explicitement déclarées avant d'être employées

Dans cet exemple, la commande SR.var('z') « construit » et renvoie une variable symbolique dont le nom est z. Cette variable symbolique est un objet Sage comme un autre : elle n'est pas traitée différemment d'expressions plus complexes comme $\sin(x) + 1$. Ensuite, cette variable symbolique est affectée à la variable « du programmeur » z, ce qui permet de s'en servir comme de n'importe quelle expression pour construire des expressions plus complexes.

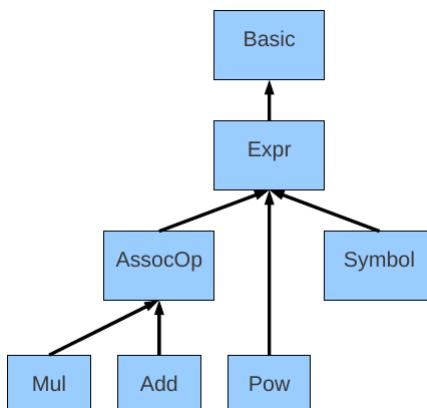
```
In [1]: from sympy import *
In [2]: x = symbols('x')
In [3]: type(x)
Out[3]: sympy.core.symbol.Symbol
```

```
In [4]: x+3
Out[4]: x + 3
```

1.2.2 Structure de données dans SymPy

Le moteur symbolique de SymPy tire parti de l'orientation des objets (notamment l'héritage) pour créer une base de code facilement extensible. Toutes les classes dérivent des fonctionnalités, telles que la possibilité de se comparer à d'autres objets, à partir de méthodes de la super-classe Basic. Les

objets pouvant faire l'objet d'opérations algébriques acquièrent cette capacité grâce à un ensemble de méthodes d'une classe appelée `Expr`. Ces objets `Expr` peuvent être conservés dans des objets conteneur (qui contiennent également la sous-classe `Expr`) `Mul`, `Add` et `Pow`; les objets conteneur sont instanciés à l'aide de l'opérateur Python, tel que la fonction de surcharge, qui permet au constructeur de la classe conteneur d'être appelé chaque fois que l'opérateur binaire approprié est utilisé (* pour `Mul`, + pour Ajouter et ** pour `Pow`). De cette manière, des objets supplémentaires peuvent être ajoutés en créant simplement une sous-classe qui hérite des fonctionnalités de la classe `Expr`. Ces sous-classes bénéficient gratuitement de certaines fonctionnalités, telles que la possibilité de comparer, de multiplier, d'ajouter, etc. Voici comment SymPy crée un environnement modifiable, maintenable, et donc facile à étendre. Grâce à la possibilité d'hériter des propriétés de classes supérieures, la quantité de code nécessaire pour développer, par exemple, un système modélisant la mécanique quantique et la notation Dirac décroissant de manière significative.



1.2.3 Variable et affectation

Exercise 1.1 Affectez les variables temps t et distance d par les valeurs 6.892 et 19.7. Calculez et affichez la valeur de la vitesse. Améliorez l'affichage en imposant un chiffre après le point décimal.

Solution 1.1 Pour, affectez des variables est les rendre symbolique comme c'est décrit dans le mémo ou il sera expliquer temps t et distance d par les valeurs 6.892 et 19.7. Calculez et affichez la valeur de la vitesse. Améliorez l'affichage en imposant un chiffre après le point décimal.

1.2.4 Substitution

Une des choses les plus courantes que vous pourriez vouloir faire avec une expression mathématique est la substitution. La substitution remplace toutes les occurrences de quelque chose dans une expression par quelque chose d'autre. SymPy utilisant la méthode `subs`.

1.2.5 Contrôle du flux d'instructions

This is a theorem consisting of just one line.

Exercise 1.2 A set $\mathcal{D}(G)$ in dense in $L^2(G)$, $|\cdot|_0$.

Solution 1.2

2. Analyse et Algèbre

Ce chapitre présente à travers des exemples simples les fonctions de base utiles en analyse et en algèbre. Les lycéens et étudiants trouveront matière à remplacer le crayon et le papier par le clavier et l'écran tout en relevant le même défi intellectuel de compréhension des mathématiques. Cet exposé des principales commandes de calcul avec Sage se veut accessible aux élèves de lycée.

2.1 Expressions symboliques et simplification

2.1.1 Expressions symboliques

La bibliothèque SymPy permet d'effectuer toutes sortes de calculs d'analyse à partir d'expressions symboliques combinant des nombres, des variables symboliques, les quatre opérations, et des fonctions usuelles comme `sqrt`, `exp`, `log`, `sin`, `cos`, etc. Une expression symbolique peut être représentée par un arbre comme ceux de la. Il est important de comprendre qu'une expression symbolique est une formule et non pas une valeur ou une fonction mathématique. Ainsi, SymPy ne reconnaît pas que les deux expressions suivantes sont égales

2.1.2 Transformation d'expressions

2.1.3 Fonctions mathématiques usuelles

La plupart des fonctions mathématiques se retrouvent dans SymPy, en particulier les fonctions trigonométriques, le logarithme et l'exponentiel : elles sont rassemblées dans le tableau

2.2 Équations

Ceci est la documentation officielle du module `solveset` dans les solveurs. Il contient les questions fréquemment posées sur notre nouveau module pour résoudre des équations.

2.2.1 Résolution explicite

2.2.2 Équations sans solution explicite

2.3 Inéquations

2.4 Analyse

Dans cette section, nous présentons succinctement les fonctions couramment utiles en analyse réelle. Pour une utilisation avancée ou des compléments, on renvoie aux chapitres suivants notamment ceux qui traitent de l'intégration numérique (ch. 14), de la résolution des équations non linéaires (ch. 12), et des équations différentielles (ch. 10)

2.4.1 Les Fonctions

Il sert également de constructeur pour les classes de fonctions non définies.

```
from sympy import Function, Symbol
```

Exercise 2.1 Écrire une fonction cube qui retourne le cube de son argument



Exercise 2.2 Écrire une fonction *volumeSphere* qui calcule le volume d'une sphère de rayon *r* fourni en argument et qui utilise la fonction cube . Tester la fonction *volumeSphere* par un appel dans le programme principal.



Exercise 2.3 Écrire une fonction maFonction qui retourne $f(x) = 2x^3 + x - 5$



Exercise 2.4 Écrire une fonction tabuler avec quatre paramètres : *fonction* , *borneInf* , *borneSup* et *nbPas* . Cette procédure affiche les valeurs de *fonction* , de *borneInf* à *borneSup* , tous les *nbPas* . Elle doit respecter *borneInf* < *borneSup*. Tester cette fonction par un appel dans le programme principal après avoir saisi les deux bornes dans une floatbox et le nombre de pas dans une integerbox (utilisez le module easyguiB).



Exercise 2.5 Écrire une fonction *volMasse* Ellipsoïde qui retourne le volume et la masse d'un ellipsoïde grâce à un tuple. Les paramètres sont les trois demi-axes et la masse volumique. On donnera à ces quatre paramètres des valeurs par défaut.

On donne: $v = \frac{3}{4}\pi abc$

Tester cette fonction par des appels avec différents nombres d'arguments.



Exercise 2.6 Une fonction $f(x)$ est linéaire et a une valeur de 29 à $x = -2$ et 39 à $x = 3$. Trouver sa valeur à $x = 5$.



Exercise 2.7 Pour l'ensemble N de nombres naturels et une opération binaire $f : NxN \rightarrow N$, on appelle un élément $z \in N$ une identité pour f , si $f(a, z) = a = f(z, a)$, pour tout $a \in N$. Lesquelles des opérations binaires suivantes ont une identité?:

1. $f(x, y) = x + y - 3$
2. $f(x, y) = \max(x, y)$
3. $f(x, y) = x^y$

Solution 2.1 le deuxième et le troisième

2.4.2 Sommes

2.4.3 Limites

2.4.4 Suites

2.4.5 Développements limités

2.4.6 Séries

On peut utiliser les commandes précédentes pour effectuer des calculs sur les séries. Donnons quelques exemples.

2.4.7 Dérivation

La fonction derivative (qui a pour alias diff) permet de dériver une expression symbolique ou une fonction symbolique

2.4.8 Dérivées partielles

La commande diff permet également de calculer des dérivées n-ièmes ou des dérivées partielles.

2.4.9 Intégration

2.5 Calcul matriciel

Dans cette section, on décrit les fonctions de base utiles en algèbre linéaire :opérations sur les vecteurs, puis sur les matrices. Pour plus de détails, on renvoie au chapitre 8 pour le calcul matriciel symbolique et au chapitre 13 pour le calcul matriciel numérique.

2.5.1 Résolution de systèmes linéaires

2.5.2 Calcul vectoriel

Les fonctions de base utiles pour la manipulation des vecteurs sont résumées dans le tableau 2.5. On peut se servir de ces fonctions pour traiter l'exercice suivant.

2.5.3 Calcul matriciel

2.5.4 Réduction d'une matrice carrée



3. Graphiques

La visualisation de fonctions d'une ou deux variables, d'une série de données, facilite la perception de phénomènes mathématiques ou physiques et permet de conjecturer des résultats efficacement. Dans ce chapitre, on illustre sur des exemples les capacités graphiques de SymPy.

3.1 Courbes en 2D

Une courbe plane peut être définie de plusieurs façons : comme graphe d'une fonction d'une variable, par un système d'équations paramétriques, par une équation en coordonnées polaires, ou par une équation implicite. Nous présentons ces quatre cas, puis donnons quelques exemples de tracés de données.

- 3.1.1 Représentation graphique de fonctions
- 3.1.2 Courbe paramétrée
- 3.1.3 Courbe en coordonnées polaires
- 3.1.4 Courbe définie par une équation implicite
- 3.1.5 Tracé de données
- 3.1.6 Tracé de solution d'équation différentielle
- 3.1.7 Développée d'une courbe
- 3.2 Courbes en 3D

Part II

Logique, ensemble et catégorie

4. Logique

La bibliothèque fournit un module pour faire de la logique mathématique indépendamment vue comme extension des classes fournit par le builtins de Python.

5. Théorie des ensembles

5.1 Un petit rappel pour les ensembles dans Python

La théorie d'ensemble existe sous Python, il suffit simplement de tapez

```
ens = set()
```

ou écrire simplement

```
ens = {1, 2, 4}
```

5.2 Théorie des ensembles

La notion d'objet immuable en Python est fondamentale, une structure qui rappelle les ensembles en mathématiques que soit fini ou infini est *set*, importante, bien que dans le cadre de SymPy elle s'appuie entièrement sur Python avec certaines modifications, avec la collection d'objet.

La fonction set accepte donc en argument un objet de type quelconque et s'efforce de le traduire dans un ensemble. Lorsqu'on ne passe aucun argument à set (option 2), ou qu'on lui passe une liste vide, set renvoie naturellement un ensemble vide; on aurait pu utiliser aussi bien, de la même manière, set(), set(), ou même set("") pour arriver au même résultat.

Exercise 5.1 Définir deux ensembles $X = \{a, b, c, d\}$ et $Y = \{s, b, d\}$, puis affichez les résultats suivants :

1. les ensembles initiaux.
2. le test d'appartenance de l'élément c à X .
3. le test d'appartenance de l'élément a à Y .
4. les ensembles $X - Y$ et $Y - X$.
5. l'ensemble $X \cup Y$ (union).
6. l'ensemble $X \cap Y$ (intersection).

Solution 5.1 Il faut noter qu'il existe une solution qui se base sur le Python builtins en utilisant la structure de donnée *sets*. Mais comme en n'est dans la logique en utilise

```
from sympy import FiniteSet

X = FiniteSet('a', 'b', 'c', 'd')
Y = FiniteSet('s', 'b', 'd')

class MyClass(Yourclass):
    def __init__(self, my, yours):
        bla = '5 1 2 3 4'
        print bla

class MyClass(Yourclass):
    def __init__(self, my, yours):
        bla = '5 1 2 3 4'
        print bla
```

5.2.1 Logique

Exercise 5.2 Dans la carte de Karnaugh ci-dessous, X indique un terme sans intérêt. Quelle est la forme minimale de la fonction représentée par la carte de Karnaugh?

5.2.2 Ensembles

La notion d'objet immuable en Python est fondamentale, une structure qui rappel les ensembles en mathématiques que soit fini ou infini est *set*, importante, bien que dans le cadre de SymPy elle s'appuie entièrement sur Python avec certain modification, avec la collection d'objet.

La fonction set accepte donc en argument un objet de type quelconque et s'efforce de le traduire dans un ensemble. Lorsqu'on ne passe aucun argument à set (option 2), ou qu'on lui passe une liste vide, set renvoie naturellement un ensemble vide; on aurait pu utiliser aussi bien, de la même manière, set(()), set(), ou même set("") pour arriver au même résultat.

Exercise 5.3 Définir deux ensembles $X = \{a, b, c, d\}$ et $Y = \{s, b, d\}$, puis affichez les résultats suivants :

1. les ensembles initiaux.
2. le test d'appartenance de l'élément c à X .
3. le test d'appartenance de l'élément a à Y .
4. les ensembles $X - Y$ et $Y - X$.
5. l'ensemble $X \cup Y$ (union).
6. l'ensemble $X \cap Y$ (intersection).

Solution 5.2 Il faut noter qu'il existe une solution qui se base sur le Python builtins en utilisant la structure de donnée *sets*. Mais comme en n'est dans la logique en utilise

```
from sympy import FiniteSet

X = FiniteSet('a', 'b', 'c', 'd')
Y = FiniteSet('s', 'b', 'd')

class MyClass(Yourclass):
    def __init__(self, my, yours):
        bla = '5 1 2 3 4'
        print bla

class MyClass(Yourclass):
    def __init__(self, my, yours):
        bla = '5 1 2 3 4'
        print bla
```



6. Théorie des catégories

6.1 Note historique

Les catégories¹ sont utilisées dans la plupart des branches mathématiques et dans certains secteurs de l'informatique théorique et en mathématiques de la physique. Elles forment une notion unificatrice. Cette théorie a été mise en place par Samuel Eilenberg et Saunders Mac Lane en 1942-1945, en lien avec la topologie algébrique, et propagée dans les années 1960-1970 en France par Alexandre Grothendieck, qui en fit une étude systématique. À la suite des travaux de William Lawvere, la théorie des catégories est utilisée depuis 1969 pour définir la logique et la théorie des ensembles ; la théorie des catégories peut donc, comme la théorie des ensembles ou la théorie des types, avec laquelle elle a des similarités, être considérée comme fondement des mathématiques. Une élaboration fine est développé avec l'ajout de la théorie d'homotopie pour en donnée de la théorie des types homotopiques qui sera pas traité dans ce chapitre ni d'une catégorie élaboré tel-que: 2-catégorie ou ∞ -catégorie

6.2 Introduction

Dans le langage des catégories tout commence avec la relation des objets entre-eux à travers des flèches ou morphismes. Il existe une pycategories² bibliothèque Python pour la théorie des catégories, différence avec SymPy

Application à l'informatique.

Pour commencer nous définissons des objets et des flèches qui met en relation avec ces objets

■ **Example 6.1** (Produit cartésien) Comment définir le produit cartésien $A \times B = \{(a, b) | a \in A \wedge b \in B\}$ sans faire intervenir \in , en considérant uniquement des transformations(fonctions) entre objects(ensembles)?.

- premier projection: $\pi_1 : A \times B \rightarrow A$

¹<https://prezi.com/chlftwema-yy/intro-a-la-theorie-des-categories>

²<https://pypi.org/project/pycategories/>

- deuxième projection: $\pi_2 : A \times B \rightarrow B$
- la structure $(A \times B, \pi_1, \pi_2)$ est optimale
- quelques soient $f : C \rightarrow A$ et $g : C \rightarrow B$ on peut construire $\langle f, g \rangle : C \rightarrow A \times B$ définie par et $\langle f, g \rangle(x) = (f(x), g(x))$

$$\begin{array}{ccc} X & \xrightarrow{f} & Y \\ & \searrow h & \downarrow g \\ & V & \end{array}$$

Le code python correspondant.

```
from sympy.categories import Object, NamedMorphism
from sympy import init_printing(pretty_print=True)
A = Object("A")
B = Object("B")
```

■

6.2.1 Quels applications pour l'informatiques

“This paper tries to explain why and how category theory is useful in computing science, by giving guidelines for applying seven basic categorical concepts : category, functor, natural transformation, limit, adjoint, colimit and comma category. Some examples, intuition, and references are given for each concept, but completeness is not attempted.”

- théorie des graphes (catégorie \approx algèbre des chemins)
- théorie des automates (systèmes et comportement, bisimulation)
- théorie des types (polymorphisme)
- programmation fonctionnelle (modèles du λ -calcul, effets)
- substitutions de variables et unification
- systèmes de réécriture

Part III

Algèbre et théorie des nombres

7. Anneaux et corps finis

7.1 Anneau des entiers modulo n

7.2 Corps finis

7.3 Quelques problèmes élémentaires de théorie des nombres

7.3.1 Théorème de Wilson

¹ énoncé du théorème

Theorem 7.3.1 Un entier p strictement plus grand que 1 est un nombre premier si et seulement s'il divise $(p-1)! + 1$, c'est-à-dire si et seulement si $(p-1)! + 1 \equiv 0 \pmod{p}$

Indication. Quatre démonstrations de ce résultat de Wilson. L'idée directrice des deux premières démonstrations est de remplacer ce calcul de congruence $(\mathbb{Z}/p\mathbb{Z})$ par un calcul dans \mathbb{F}_p ce qui va permettre d'utiliser les propriétés d'un corps. L'idée de la troisième démonstration est d'utiliser les théorèmes de Sylow dans le groupe symétrique σ_p , la quatrième est plutôt combinatoire qui repose sur l'identité algébrique $\sum_{i=0}^n (-1)^i C_n^p (x-i)^n = n!$ donnée en théorème l'auteur² ainsi le théorème sera simplement comme corollaire en remplaçant n par $p-1$

Proof. A venir! ■

Alain Connes dans son article "Autour du théorème de Wilson"³, donne une approximation du nombre π en somme de \sin

¹**Note Historique.** Le théorème de Wilson a été découvert à la fin du dixième siècle par le mathématicien arabe Ibn al-Haytham (965 – 1040). Le résultat ressurgit, sans démonstration, à la fin du dix-huitième siècle dans les écrits de Edward Waring qui l'attribue en 1770 à son élève John Wilson. L'année suivante, Lagrange en donne deux démonstrations dans son article [LAG]. En fait, Leibniz (1646 – 1716) connaissait déjà le résultat et sa démonstration mais ne les avait pas publiés (voir [RAS] pour de plus amples considérations historiques).

²<https://arxiv.org/pdf/math/0406086.pdf>

³Alain Connes, An essay on the Riemann Hypothesis, Open Problems in Mathematics. John Forbes Nash, Jr. Michael Th. Rassias Editors

R Contrairement au petit théorème de Fermat, le théorème de Wilson est une condition nécessaire et suffisante pour tester la primalité. Toutefois, cela conduirait à un test très lent informatiquement, car le calcul de $(p-1)!$ nécessite beaucoup d'opérations.

```
import sympy

def isPrime(n):
    if n == 4: return
    return bool(math.factorial(n>>1)%n)
```



8. Polynômes

8.1 Anneaux et polynômes

8.1.1 Introduction

Nous avons vu au chapitre 2 comment effectuer des calculs sur des expressions formelles, éléments de « l’anneau symbolique ». Dans ce chapitre en va manipuler le module dédié, `sympy.polys`, permettant de calculer des algèbres polynomiales sur divers domaines de coefficients implémentant un grand nombre de méthodes allant d’outils simples comme la division polynomiale à des concepts avancés comprenant les bases de Gröbner et la factorisation multivariée sur des domaines de nombres algébrique:

8.1.2 Construction d’anneaux de polynômes

En SymPy, les polynômes, comme beaucoup d’autres objets algébriques, sont en général à coefficients dans un anneau commutatif. C’est le point de vue que nous adoptons, mais la plupart de nos exemples concernent des polynômes sur un corps. Dans tout le chapitre, les lettres `A` et `K` désignent respectivement un anneau commutatif et un corps quelconques. La première étape pour mener un calcul dans une structure algébrique est souvent de construire `R` elle-même. On construit $\mathbb{Q}[x]$

8.2 Polynômes

8.2.1 Crédation et arithmétique de base

8.2.2 Vue d’ensemble des opérations sur les polynômes

8.2.3 Changement d’anneau

¹ Changement d’anneau. La liste exacte des opérations disponibles, leur effet et leur efficacité dépendent fortement de l’anneau de base. Par exemple, les polynômes de `ZZ['x']` possèdent une méthode `content` qui renvoie leur contenu, c’est-à-dire le pgcd de leurs coefficients ; ceux de `QQ['x']`

¹Contrairement à Sage, il peut y être que dans SymPy certain fonctionnalité ne soit pas directement accessible que par passage à la programmation

non, l'opération étant triviale. La méthode `factor` existe quant à elle pour tous les polynômes mais déclenche une exception `NotImplementedError` pour un polynôme à coefficients dans `SR`(le cas de Sage) ou dans $\mathbb{Z}/4\mathbb{Z}$. Par exemple Cette exception signifie que l'opération n'est pas disponible dans Sage pour ce type d'objet bien qu'elle ait un sens mathématiquement. Il est donc très utile de pouvoir jongler avec les différents anneaux de coefficients sur lesquels on peut considérer un « même » polynôme. Appliquée à un polynôme de $A[x]$, la méthode `change_ring` renvoie son image dans $B[x]$, quand il y a une façon naturelle de convertir les coefficients. La conversion est souvent donnée par un morphisme canonique de A dans B : notamment, `change_ring` sert à étendre l'anneau de base pour disposer de propriétés algébriques supplémentaires. Ici par exemple, le polynôme p est irréductible sur les entiers, mais se factorise sur R :

8.2.4 Itération

8.3 Arithmétique euclidienne

8.3.1 Divisibilité

8.3.2 Idéaux et quotients

8.3.3 Idéaux

8.4 Factorisation et racines

8.4.1 Factorisation

8.4.2 Recherche de racines

8.4.3 Résultant

8.4.4 Groupe de Galois

Par défaut le calcul de groupe de Galois n'est pas disponible dans SymPy, ce qui nous amène encore une fois de programmer en ajoutant des modules. Le groupe de Galois d'un polynôme irréductible $p \in \mathbb{Q}[x]$ est un objet algébrique qui décrit certaines « symétries » des racines de p . Il s'agit d'un objet central de la théorie des équations algébriques. Notamment, l'équation $p(x) = 0$ est résoluble par radicaux, c'est-à-dire que ses solutions s'expriment à partir des coefficients de p au moyen des quatre opérations et de l'extraction de racine n -ième, si et seulement si le groupe de Galois de p est résoluble.

8.5 Fractions rationnelles

8.5.1 Construction et propriétés élémentaires

La division de deux polynômes (sur un anneau intègre) produit une fraction rationnelle. Son parent est le corps des fractions de l'anneau de polynômes, qui peut s'obtenir par `Frac(R)` :

8.6 Séries formelles

Une série formelle est une série enGroupes de matrices.tière vue comme une simple suite de coefficients, sans considération de convergence. Plus précisément, si A est un anneau commutatif, on appelle séries formelles (en anglais formal power series) d'indéterminée à coefficients dans les sommes formelles $\sum_{n=0}^{\infty} a_n x^n$ où (a_n) est une suite quelconque d'éléments de A . Munies des opérations d'addition et de multiplication naturelles

$$\sum_{n=0}^{\infty} a_n x^n + \sum_{n=0}^{\infty} b_n x^n = \sum_{n=0}^{\infty} (a_n + b_n) x^n$$

,

$$\left(\sum_{n=0}^{\infty} a_n x^n \right) \left(\sum_{n=0}^{\infty} b_n x^n \right) = \sum_{n=0}^{\infty} \left(\sum_{i+j=n} a_i b_j \right) x^n$$

, les séries formelles forment un anneau noté $A[[x]]$.

les séries formelles forment un anneau noté Dans un système de calcul formel, ces séries sont utiles pour représenter des fonctions analytiques dont on n'a pas d'écriture exacte. Comme toujours, l'ordinateur fait les calculs, mais c'est à l'utilisateur de leur donner un sens mathématique. À lui par exemple de s'assurer que les séries qu'il manipule sont convergentes.

8.6.1 Opérations sur les séries tronquées

8.6.2 Développement de solutions d'équations

Face à une équation différentielle dont les solutions exactes sont trop compliquées à calculer ou à exploiter une fois calculées, ou tout simplement qui n'admet pas de solution en forme close, un recours fréquent consiste à chercher des solutions sous forme de séries. On commence habituellement par déterminer les solutions de l'équation dans l'espace des séries formelles, et si nécessaire, on conclut ensuite par un argument de convergence que les solutions formelles construites ont un sens analytique. SymPy peut être d'une aide précieuse pour la première étape. Considérons par exemple l'équation différentielle

■ **Example 8.1**

$$(x) = \sqrt{1+x^2}$$

Exercise 8.1 Considérons le polynôme $p(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3$, où $a_i \neq 0 \forall i$. Le nombre minimum de multiplications nécessaires pour évaluer p sur une entrée x est:



9. Algèbre linéaire

Ce chapitre traite de l'algèbre linéaire exacte et symbolique, c'est-à-dire sur des anneaux propres au calcul formel, tels que \mathbb{Z} , des corps finis, des anneaux de polynômes. Nous présentons les constructions sur les matrices et leurs espaces ainsi que les opérations de base, puis les différents calculs possibles sur ces matrices, regroupés en deux thèmes : ceux liés à l'élimination de Gauss et aux transformations par équivalence à gauche, et ceux liés aux valeurs et espaces propres et aux transformations de similitude.

9.1 Constructions et manipulations élémentaires

9.1.1 Espaces de vecteurs, de matrices

De la même façon que pour les polynômes, les vecteurs et les matrices sont manipulés comme des objets algébriques appartenant à un espace. Si les coefficients appartiennent à un corps K , c'est un espace vectoriel sur K ; s'ils appartiennent à un anneau, c'est un K -module libre. **Groupes de matrices.** On pourra par ailleur définir des sous-groupes de l'espace total des matrices. Ainsi le constructeur

9.1.2 Construction des matrices et des vecteurs

Les matrices et les vecteurs peuvent naturellement être générés comme des éléments d'un espace en fournissant la liste des coefficients en arguments. Pour les matrices, ceux-ci seront lus par ligne :

9.1.3 Manipulations de base et arithmétique sur les matrices

Indices et accès aux coefficients. L'accès aux coefficients ainsi qu'à des sous-matrices extraites se fait de façon unifiée par l'opérateur crochet $A[i, j]$, selon les conventions usuelles de Python. Les indices de ligne i et de colonne j peuvent être des entiers (pour l'accès à des coefficients) ou des intervalles sous la forme $1 : 3$ (on rappelle que par convention, en Python les indices commencent à 0, et les intervalles sont toujours inclusifs pour la borne inférieure et exclusifs pour la borne supérieure). L'intervalle «`:`» sans bornes correspond à la totalité des indices possibles dans la

dimension considérée. La notation $a : b : k$ permet d'accéder aux indices compris entre a et $b - 1$ par pas de k . Enfin, les indices négatifs sont aussi valides, et permettent de parcourir les indices à partir de la fin. Ainsi $A[-2, :]$ correspond à l'avant dernière ligne. L'accès à ces sous-matrices se fait aussi bien en lecture qu'en écriture. On peut par exemple modifier une colonne donnée de la façon suivante :

9.1.4 Opérations de base sur les matrices

Les opérations arithmétiques sur les matrices se font avec les opérateurs usuels $+, -, *, \cdot$. L'inverse d'une matrice A peut s'écrire aussi bien A^{-1} que A . Lorsque a est un scalaire et A une matrice, l'opération $a * A$ correspond à la multiplication externe de l'espace de matrices. Pour les autres opérations où un scalaire a est fourni en lieu et place d'une matrice (par exemple l'opération $a+A$), il est considéré comme la matrice scalaire correspondante aI_n si $a \neq 0$ et les dimensions le permettent. Le produit élément par élément de deux matrices s'effectue avec l'opération `elementwise_product`.

9.2 Calculs sur les matrices

En algèbre linéaire, les matrices peuvent être utilisées pour représenter aussi bien des familles de vecteurs, des systèmes d'équations linéaires, des applications linéaires ou des sous-espaces. Ainsi, le calcul d'une propriété comme le rang d'une famille, la solution d'un système, les espaces propres d'une application linéaire, ou la dimension d'un sous-espace se ramènent à des transformations sur ces matrices révélant cette propriété. Ces transformations correspondent à des changements de base, vus au niveau.

Ces transformations correspondent à des changements de base, vus au niveau matriciel comme des transformations d'équivalence : $B = PAQ^{-1}$, où P et Q sont des matrices inversibles. Deux matrices sont dites équivalentes s'il existe une telle

transformation pour passer de l'une à l'autre. On peut ainsi former des classes d'équivalence pour cette relation, et l'on définit des formes normales, permettant de caractériser de manière unique chaque classe d'équivalence. Dans ce qui suit, nous présentons l'essentiel des calculs sur les matrices disponibles avec SymPy, sous l'angle de deux cas particuliers de ces transformations :

- Les transformations d'équivalence à gauche, de la forme $B = UA$, qui révèlent les propriétés caractéristiques pour les familles de vecteurs, telles que le rang (nombre de vecteurs linéairement indépendants), le déterminant (volume du parallélépipède décrit par la famille de vecteurs), le profil de rang (premier sous-ensemble de vecteurs formant une base), . . . L'élimination de Gauss est l'outil central pour ces transformations, et la forme échelonnée réduite (forme de Gauss-Jordan dans un corps ou forme de Hermite dans \mathbb{Z}) est la forme normale. En outre, ces transformations servent à la résolution des systèmes linéaires.
- Les transformations de similitude, de la forme $B = UAU^{-1}$, qui révèlent les propriétés caractéristiques des matrices représentant des endomorphismes, comme les valeurs propres, les espaces propres, les polynômes minimal et caractéristique, . . . La forme de Jordan ou la forme de Frobenius, selon les domaines de calcul, seront les formes normales pour ces transformations.

La forme de Gram-Schmidt est une autre décomposition basée sur les transformations d'équivalence à gauche, transformant une matrice en un ensemble de vecteurs orthogonaux.

9.2.1 Élimination de Gauss, forme échelonnée

Élimination de Gauss et équivalence à gauche. L'élimination de Gauss est l'une des opérations fondamentales en algèbre linéaire car elle permet d'accéder à une représentation de la matrice à la fois plus adaptée au calcul, comme la résolution de systèmes, et révélant certaines de ses propriétés fondamentales, comme le rang, le déterminant, le profil de rang, etc. Les opérations de base pour l'élimination sont les opérations élémentaires sur les lignes :

Élimination de Gauss-Jordan. La transformation de Gauss-Jordan est similaire à celle de Gauss, en ajoutant à $G_{x,k}$ les transvections correspondant aux lignes d'indice $i < k$; cela revient à éliminer les coefficients d'une colonne au-dessus et au-dessous du pivot. Si de plus on divise chaque ligne par son pivot, on obtient alors une forme échelonnée dite réduite encore appelée forme de Gauss-Jordan. Pour toute classe d'équivalence de matrices, il existe une unique matrice sous cette forme ; il s'agit donc d'une forme normale.

Forme échelonnée dans les anneaux euclidiens.

9.2.2 Résolution de systèmes ; image et base du noyau

9.2.3 Valeurs propres, forme de Jordan et transformations de similitude

Lorsque l'on interprète une matrice carrée comme un opérateur linéaire (un endomorphisme), elle n'en est que la représentation dans une base donnée. Tout changement de base correspond à une transformation de similitude $B = U^{-1}AU$ de la matrice. Les deux matrices A et B sont alors dites semblables. Ainsi les propriétés de l'opérateur linéaire, qui sont indépendantes de la base, sont révélées par l'étude des invariants de similitude de la matrice. Par l'étude des invariants de similitude de la matrice. Parmi ces invariants, les plus simples sont le rang et le déterminant. En effet les matrices U et U^{-1} étant inversibles, le rang de $U^{-1}AU$ égale le rang de A . De plus $\det(U^{-1}AU) = \det(U^{-1})\det(A)\det(U) = \det(U^{-1}U)\det(A) = \det(A)$. De la même façon, le polynôme caractéristique de la matrice A , défini par est aussi invariant par transformation de similitude :

Par conséquent, les valeurs caractéristiques d'une matrice, définies comme les racines du polynôme caractéristique dans son corps de décomposition, sont donc aussi des invariants de similitude. Par définition, un scalaire λ est une valeur propre d'une matrice A si il existe un vecteur non nul u tel que $Au = \lambda u$. L'espace propre associé à une valeur propre λ est l'ensemble des vecteurs u tels que $Au = \lambda u$. C'est un sous-espace vectoriel défini par $E_\lambda = \text{Ker}(\lambda Id - A)$.

Les valeurs propres coïncident avec les valeurs caractéristiques :

$$\det(\lambda Id - A) = 0 \Leftrightarrow \dim(\text{Ker}(\lambda Id - A)) > 1 \Leftrightarrow \exists u \in \mathbb{C}^n : \lambda u - Au = 0.$$

Ces deux points de vue correspondent respectivement à l'approche algébrique et géométrique des valeurs propres. Dans le point de vue géométrique, on s'intéresse à l'action de l'opérateur linéaire A sur les vecteurs de l'espace avec plus de précision que dans le point de vue algébrique. En particulier on distingue les notions de multiplicité algébrique, correspondant à l'ordre de la racine dans le polynôme caractéristique, de la multiplicité géométrique, correspondant à la dimension du sous-espace propre associé à la valeur propre. Pour les matrices diagonalisables, ces deux notions sont équivalentes. Dans le cas contraire, la multiplicité géométrique est toujours inférieure à la multiplicité algébrique. Le point de vue géométrique permet de décrire plus en détail la structure de la matrice. Par ailleurs, il donne des algorithmes beaucoup plus rapides pour le calcul des valeurs et espaces propres, et des polynômes caractéristique et minimal.

Espaces invariants cycliques, et forme normale de Frobenius. Soit A une matrice $n \times n$ sur

un corps K et u un vecteur de K^n . La famille de vecteurs $u, Au, A^2u, \dots, A^n u$, appelée suite de Krylov, est liée (comme famille de $n+1$ vecteurs en dimension n). Soit d tel que $A^d u$ soit le premier vecteur de la séquence linéairement dépendant avec ses prédécesseurs $u, Au, \dots, A^{d-1}u$. On écrira

$$A^d u = \sum_{i=0}^{d-1} \alpha_i A^i u$$

cette relation de dépendance linéaire. Le polynôme $\varphi_{A,u}(x) = x^d - \sum_{i=0}^{d-1} \alpha_i x^i$, qui vérifie $\varphi_{A,u}(A)u = 0$ est donc un polynôme unitaire annulateur de la suite de Krylov et de degré minimal. On l'appelle le polynôme minimal du vecteur u (sous entendu, relativement à la matrice A). L'ensemble des polynômes annulateurs de u forme un idéal de $K[X]$, engendré par $\varphi_{A,u}$. Le polynôme minimal de la matrice A est défini comme le polynôme unitaire $\varphi_A(x)$ de plus petit degré annulant la matrice A : $\varphi_A(A) = 0$. En particulier, en appliquant $\varphi_A(A)$ au vecteur u , on constate que $\varphi_A(A)$ est un polynôme annulateur de la suite de Krylov. Il est donc nécessairement un multiple du polynôme minimal de u . On peut en outre montrer (cf. exercice) qu'il existe un vecteur \bar{u} tel que

$$\varphi_{A,\bar{u}} = \varphi_A. \quad (9.1)$$

Lorsque le vecteur u est choisi aléatoirement, la probabilité qu'il satisfasse l'équation (1.1) est d'autant plus grande que la taille du corps est grande (on peut montrer qu'elle est au moins $1 - \frac{n}{|K|}$)

Exercise 9.1 Montrons qu'il existe toujours un vecteur u dont le polynôme minimal coïncide avec le polynôme minimal de la matrice.

1. coïncide avec le polynôme minimal de la matrice.
1. Soit (e_1, \dots, e_n) une base de l'espace vectoriel. Montrer que φ_A coïncide avec le ppcm des φ_{A,e_i} .
2. Dans le cas particulier où φ_A est une puissance d'un polynôme irréductible, montrer qu'il existe un indice i_0 tel que $\varphi_A = \varphi_{A,e_{i_0}}$.
- 3.
- 4.
- 5.

Facteurs invariants et invariants de similitude. Une propriété importante relie les invariants de similitude et les facteurs invariants vus dans la section Les invariants de similitude d'une matrice à coefficients dans

Theorem 9.2.1 Les invariants de similitude d'une matrice A à coefficients dans un corps correspondent aux facteurs invariants de sa matrice caractéristique $xId - A$.

Valeurs propres, vecteurs propres. Si l'on décompose le polynôme minimal en facteurs irréductibles, $\varphi_1 = \psi_1^{m_1}, \psi_2^{m_2}, \psi_3^{m_3}, \dots, \psi_s^{m_s}$ alors tous les facteurs invariants s'écrivent sous la forme.

Forme de Jordan. Lorsque le polynôme minimal est scindé mais ayant des facteurs avec des multiplicités supérieures à 1, la forme intermédiaire (8.3) n'est pas diagonale. On montre alors qu'il n'existe pas de transformation de similitude la rendant diagonale, la matrice initiale n'est donc pas diagonalisable. On peut en revanche la trigonaliser, c'est-à-dire la rendre triangulaire supérieure, telle que les valeurs propres apparaissent sur la diagonale. Parmi les différentes matrices triangulaires possibles, la plus réduite de toutes est la forme normale de Jordan.

Forme normale primaire. Pour être complet, il faut mentionner une dernière forme normale qui généralise la forme de Jordan dans le cas quelconque où le polynôme minimal n'est pas scindé. Pour un polynôme irréductible P de degré k , on définit le bloc de Jordan de multiplicité m comme la matrice $J_{P,m}$ de dimension $km \times km$ vérifiant.

unique à une permutation des blocs diagonaux près. L'unicité de ces formes normales permet en particulier de tester si deux matrices sont semblables, et par la même occasion de produire une matrice de passage entre l'une et l'autre.

Exercise 9.2 Écrire un programme qui détermine si deux matrices A et B sont semblables et renvoie la matrice U de passage telle que $A = U^{-1}BU$ (on pourra renvoyer None dans le cas où les matrices ne sont pas semblables). ▀

10. Systèmes polynomiaux

Ce chapitre prolonge les deux précédents. Les objets sont des systèmes d'équations à plusieurs variables, comme ceux du chapitre 8. Ces équations, dans la lignée du chapitre 7, sont polynomiales. Par rapport aux polynômes à une seule indéterminée, ceux à plusieurs indéterminées présentent une grande richesse mathématique mais aussi des difficultés nouvelles, liées notamment au fait que l'anneau $K[x_1, x_2, \dots, x_n]$ n'est pas principal. La théorie des bases de Gröbner fournit des outils pour contourner cette limitation. Au final, on dispose de méthodes puissantes pour étudier les systèmes polynomiaux, avec d'innombrables applications qui couvrent des domaines variés.

Une bonne partie du chapitre ne presuppose que des connaissances de base sur les polynômes à plusieurs indéterminées. Certains passages sont cependant du niveau d'un cours d'algèbre commutative de L3 ou M1. Pour une introduction moins allusive et en français à la théorie mathématique des systèmes polynomiaux, accessible au niveau licence, le lecteur pourra se reporter au chapitre [FSED09] de Faugère et Safey El Din. On trouvera un traitement plus avancé dans le livre de Elkadi et Mourrain [EM07]. Enfin, en anglais cette fois, le livre de Cox, Little et O'Shea [CLO07] est à la fois accessible et fort complet.

10.1 Polynômes à plusieurs indéterminées

10.1.1 Les anneaux $A[x_1, \dots, x_n]$

Nous nous intéressons ici aux polynômes à plusieurs indéterminées, dits aussi — anglicisme commun dans le domaine du calcul formel — multivariés. Comme pour les autres structures algébriques disponibles dans SymPy, avant de pouvoir construire des polynômes, il nous faut définir une famille d'indéterminées, vivant toutes dans un même anneau. La syntaxe est pratiquement la même qu'en une variable :

Exercise 10.1 Définir l'anneau $\mathbb{Q}[x_2, x_3, \dots, x_{37}]$ dont les indéterminées sont indexées par les nombres premiers inférieurs à 40, ainsi que des variables x_2, x_3, \dots, x_{37} pour accéder aux indéter-

minées.

Il peut enfin s'avérer utile, dans quelques cas, de manipuler des polynômes à plusieurs indéterminées en représentation récursive, c'est-à-dire comme éléments d'un anneau de polynômes à coefficients eux-mêmes polynomiaux.

10.1.2 Polynômes

10.2 Systèmes polynomiaux et idéaux

Nous abordons à présent le sujet central de ce chapitre. Les sections 9.2.1 et 9.2.2 offrent un panorama des manières de trouver et de comprendre les solutions d'un système d'équations polynomiales avec l'aide de SymPy La section 9.2.3 est consacrée aux idéaux associés à ces systèmes. Les sections suivantes reviennent de façon plus détaillée sur les outils d'élimination algébrique et de résolution de systèmes.

10.2.1 Un premier exemple

Considérons une variante du système polynomial de la section 2.2,

$$\begin{cases} x^2yz = 18 \\ xy^3z = 24 \\ xyz^4 = 0,5 \end{cases} \quad (10.1)$$

Simplifier le système. Une approche différente est possible. Plutôt que de chercher les solutions, essayons de calculer une forme plus simple du système lui-même. Les outils fondamentaux qu'offre Sage pour ce faire sont la décomposition triangulaire et les bases de Gröbner. Nous verrons plus loin ce qu'ils calculent exactement ; essayons déjà de les utiliser sur cet exemple :

10.2.2 Qu'est-ce que résoudre ?

Un système polynomial qui possède des solutions en a souvent une infinité. L'équation toute simple $x^2 - y = 0$ admet une infinité de solutions dans \mathbb{Q}^2 , sans parler de \mathbb{R}^2 ou \mathbb{C}^2 . Il n'est donc pas question de les énumérer. Le mieux qu'on puisse faire est décrire l'ensemble des solutions « aussi explicitement que possible », c'est-à-dire en calculer une représentation dont on puisse facilement extraire des informations intéressantes. La situation est analogue à celle des systèmes linéaires, pour lesquels (dans le cas homogène) une base du noyau du système est une bonne description de l'espace des solutions.

Dans le cas particulier où les solutions sont en nombre fini il devient possible de « les calculer ». Mais même dans ce cas, cherche-t-on à énumérer les solutions dans \mathbb{Q} , ou encore dans un corps fini \mathbb{F}_n ? À trouver des approximations numériques des solutions réelles ou complexes ? Ou encore, comme dans l'exemple de la section précédente, à représenter ces dernières à l'aide de nombres algébriques, c'est-à-dire par exemple à calculer les polynômes minimaux de leurs coordonnées ?

Ce même exemple illustre que d'autres représentations de l'ensemble des solutions peuvent être bien plus parlantes qu'une simple liste de points, surtout quand les solutions sont nombreuses. Ainsi, les énumérer n'est pas forcément la chose la plus pertinente à faire même quand c'est possible. In fine, on ne cherche pas tant à calculer les solutions qu'à calculer avec les solutions, pour en déduire ensuite, suivant le problème, les informations auxquelles on s'intéresse vraiment. La suite de ce chapitre explore différents outils utiles pour ce faire.

10.2.3 Idéaux et systèmes

Si s polynômes $p_1, \dots, p_s \in K[x]$ s'annulent en un point x à coordonnées dans K ou dans une extension de K , tout élément de l'idéal qu'ils engendrent s'annule aussi en x . Il est donc naturel d'associer au système polynomial

$$p_1(x) = p_2(x) = \dots = p_s(x)$$

l'idéal $J = \langle p_1, \dots, p_s \rangle \subset K[x]$. Deux systèmes polynomiaux qui engendent le même idéal sont équivalents au sens où ils ont les mêmes solutions. Si L est un corps contenant K , on appelle sous-variété algébrique de L^n associée à J l'ensemble

$$V_L(J) = \{x \in L^n \mid \forall p \in J, p(x) = 0\} = \{x \in L^n \mid p_1(x) = \dots = p_s(x) = 0\}$$

des solutions à coordonnées dans L du système. Des idéaux différents peuvent avoir la même variété associée. Par exemple, les équations $x = 0$ et $x^2 = 0$ admettent la même unique solution dans \mathbb{C} , alors que l'on a $\langle x^2 \rangle \subsetneq \langle x \rangle$. Ce que l'idéal engendré par un système polynomial capture est plutôt la notion intuitive de « solutions avec multiplicités ». Ainsi, les deux systèmes suivants expriment chacun l'intersection du cercle unité et d'une courbe d'équation $\alpha x^2 y^2 = 1$, réunion de deux hyperboles équilatères

10.3 Bases de Gröbner

Nous avons jusqu'ici utilisé les fonctionnalités d'élimination algébrique et de résolution de systèmes polynomiaux qu'offre SymPy comme des boîtes noires. Cette section introduit quelques-uns des outils mathématiques et algorithmiques sous-jacents. Le but est à la fois d'y recourir directement et de faire un usage avisé des fonctions de plus haut niveau présentées auparavant. Les techniques employées par Sage pour les calculs sur les idéaux et l'élimination reposent sur la notion de base de Gröbner. On peut voir celle-ci, entre autres, comme une extension à plusieurs indéterminées de la * représentation par générateur principal des idéaux de $K[x]$. Le problème central de cette section est de définir et calculer une forme normale pour les éléments des algèbres quotients de $K[x]$. Notre point de vue reste celui de l'utilisateur : nous définissons les bases de Gröbner, montrons comment en obtenir avec Sage et à quoi cela peut servir, mais nous n'abordons pas les algorithmes utilisés pour faire le calcul.

10.3.1 Ordres monomiaux

10.3.2 Division par une famille de polynômes

10.3.3 Propriétés des bases de Gröbner

Les bases de Gröbner servent à implémenter les opérations étudiées dans la section 9.2. On les utilise notamment afin de calculer des formes normales pour les idéaux d'anneaux de polynômes et les éléments des quotients par ces idéaux, d'éliminer des variables dans les systèmes polynomiaux, ou encore de déterminer des caractéristiques des solutions telles que leur dimension.



11. Équations différentielles

11.1 Équations différentielles

11.1.1 Introduction

Si la méthode de George Pólya semble peu efficace, on peut faire appel à SymPy même si le domaine de la résolution formelle des équations différentielles demeure une faiblesse de nombreux logiciels de calcul. SymPy est en pleine évolution cependant et progresse à chaque version un peu plus en élargissant son spectre de résolution.

On peut, si on le souhaite, invoquer Sage afin d'obtenir une étude qualitative : en effet, ses outils numériques et graphiques guideront l'intuition. C'est l'objet de la section 14.2 du chapitre consacré au calcul numérique. Des outils d'étude graphique des solutions sont donnés à la section 4.1.6. Des méthodes de résolution à l'aide de séries se trouvent à la section 7.5.2. On peut préférer résoudre les équations différentielles exactement. Sage peut alors parfois y aider en donnant directement une réponse formelle comme nous le verrons dans ce chapitre.

Dans la plupart des cas, il faudra passer par une manipulation savante de ces équations pour aider SymPy. Il faudra veiller à garder en tête que la solution attendue d'une équation différentielle est une fonction dérivable sur un certain intervalle mais que SymPy, lui, manipule des expressions sans domaine de définition. La machine aura donc besoin d'une intervention humaine pour aller vers une solution rigoureuse.

Nous étudierons d'abord les généralités sur les équations différentielles ordinaires d'ordre 1 et quelques cas particuliers comme les équations linéaires, les équations à variables séparables, les équations homogènes, une équation dépendant d'un paramètre (§10.1.2) ; puis de manière plus sommaire les équations d'ordre 2 ainsi qu'un exemple d'équation aux dérivées partielles (§10.1.3). Nous terminerons par l'utilisation de la transformée de Laplace (§10.1.4) et enfin la résolution de certains systèmes différentiels (§10.1.5).

On rappelle qu'une équation différentielle ordinaire (parfois notée EDO, ou ODE en anglais) est une équation faisant intervenir une fonction (inconnue) d'une seule variable, ainsi qu'une ou plusieurs dérivées, successives ou non, de la fonction.

Dans l'équation $y'(x) + xy(x) = e^x$ la fonction inconnue y est appelée la variable dépendante et la variable x (par rapport à laquelle y varie) est appelée la variable indépendante. Une équation aux dérivées partielles (notée parfois EDP, ou PDE en anglais) fait intervenir plusieurs variables indépendantes ainsi que les dérivées partielles de la variable dépendante par rapport à ces variables indépendantes. Sauf mention contraire, on considérera dans ce chapitre des fonctions d'une variable réelle.

11.1.2 Équations différentielles ordinaires d'ordre 1

■ **Definition 11.1.1** Une équation différentielle ordinaire

```
from sympy import symbols, Function
x = symbols('x')
y = Function("y")(x)
```

Équations du premier ordre pouvant être résolues directement par SymPy. Nous allons étudier dans cette section comment résoudre avec SymPy Équations différentielles ordinaires d'ordre 1 les équations linéaires, les équations à variables séparables, les équations de Bernoulli, les équations homogènes, les équations exactes, ainsi que les équations de Riccati, Lagrange et Clairaut.
Équations linéaires. il s'agit d'équations du type:

$$y' + P(x)y = Q(x)$$

ou P et Q sont des fonctions continues sur des intervalles données.

■ **Example 11.1**

$$y' + 3y = e^x$$

■

11.1.3 Équations d'ordre 2

Équations linéaires à coefficients constants. Résolvons maintenant une équation du second ordre linéaire à coefficients constants, par exemple :

$$y'' + 3y = x^2 - 7x + 31$$

11.1.4 Transformée de Laplace

La transformée de Laplace permet de convertir une équation différentielle avec des conditions initiales en équation algébrique et la transformée inverse permet ensuite de revenir à la solution éventuelle de l'équation différentielle.

Pour mémoire ensuite de revenir à la solution éventuelle de l'équation différentielle. Pour mémoire, si f est une fonction définie sur \mathbb{R} en étant identiquement nulle sur $]-\infty; 0[$, on appelle transformée de Laplace de f la fonction F définie, sous certaines conditions, par :

$$\mathcal{L}(f(x)) = F(s) = \int_0^{+\infty} e^{-sx} f(x) dx$$

11.1.5 Systèmes différentiels linéaires

12. suites définies par une relation de récurrence

Il y a pas de classe pour la construction et le calcul de relation de récurrence de suite de fonction, le seul moyen est de faire intervenir le module `from.sympy.Function`

12.1 Suites définies par $u_{n+1} = f(u_n)$

Definition 12.1.1 Considérons une suite définie par une relation $u_{n+1} = f(u_n)$ avec $u_0 = a$. On peut définir la suite naturellement à l'aide d'un algorithme récursif. Prenons par exemple une suite logistique (suite définie par une récurrence de la forme $x_{n+1} = rx_n(1 - x_n)$) :

Exercise 12.1

1. Étudier la suite définie par récurrence par $u_0 = a$ et $u_{n+1} = \cos u_n$, où a est un nombre réel donné.
2. Étudier la suite définie pour $n \geq 1$ par $u_n = \underbrace{\cos(\cos(\cos(\cdots(\cos n)\cdots)))}_{n \text{ fois cos}}$

12.2 Suites récurrentes linéaires

Definition 12.2.1 Une suite récurrente linéaire est défini de $\mathbb{N} \rightarrow \mathbb{R}$ est de la forme

$$a_k u_{n+k} + a_{k-1} u_{n+k-1} + a_{k-2} u_{n+k-2} + \dots + a_1 u_{n+1} + a_0 u_0 = 0$$

avec $(a)_{0 \leq i \leq k}$ une famille de scalaire

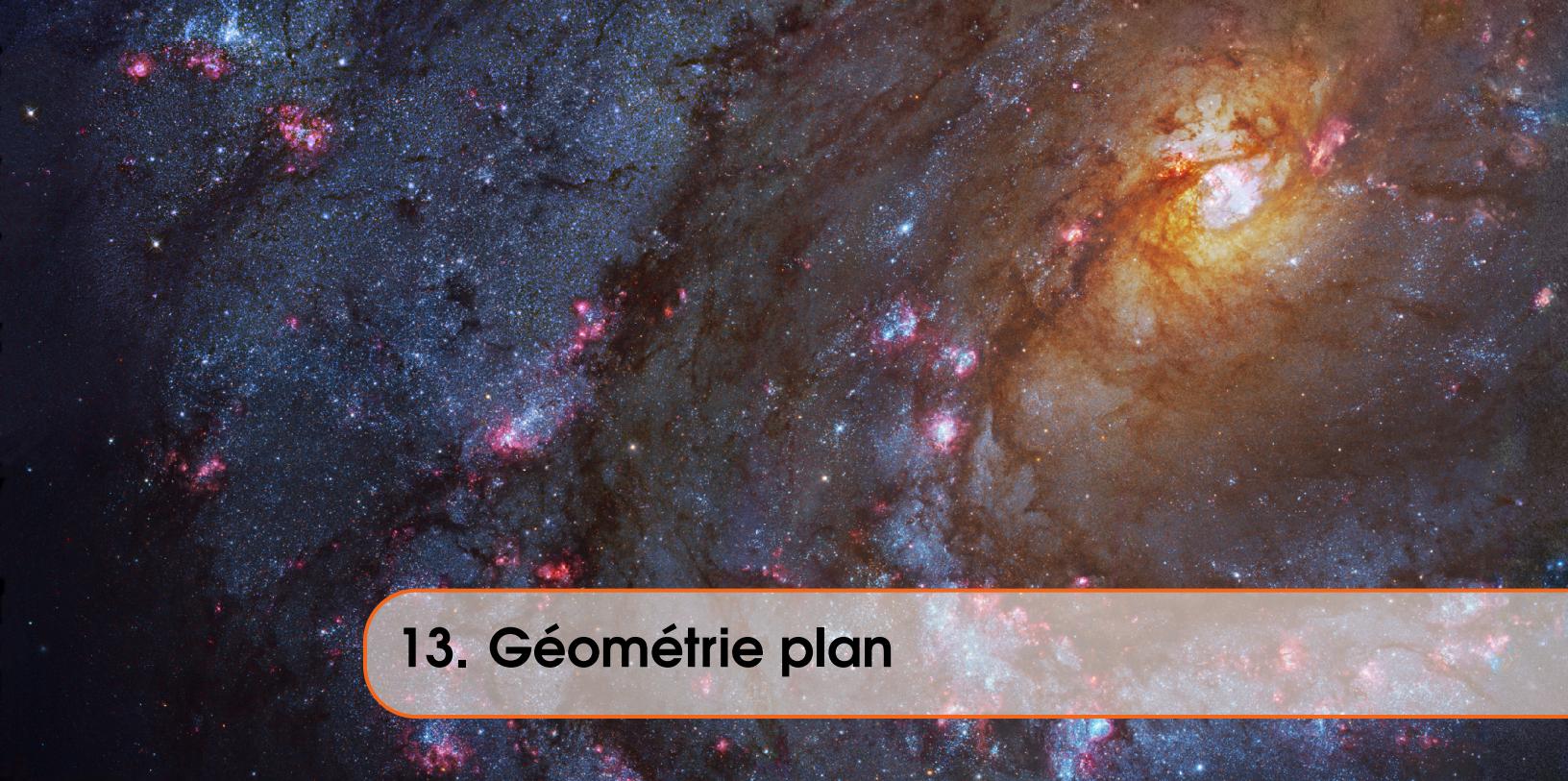
■ **Example 12.1**

12.3 Suites récurrentes « avec second membre »

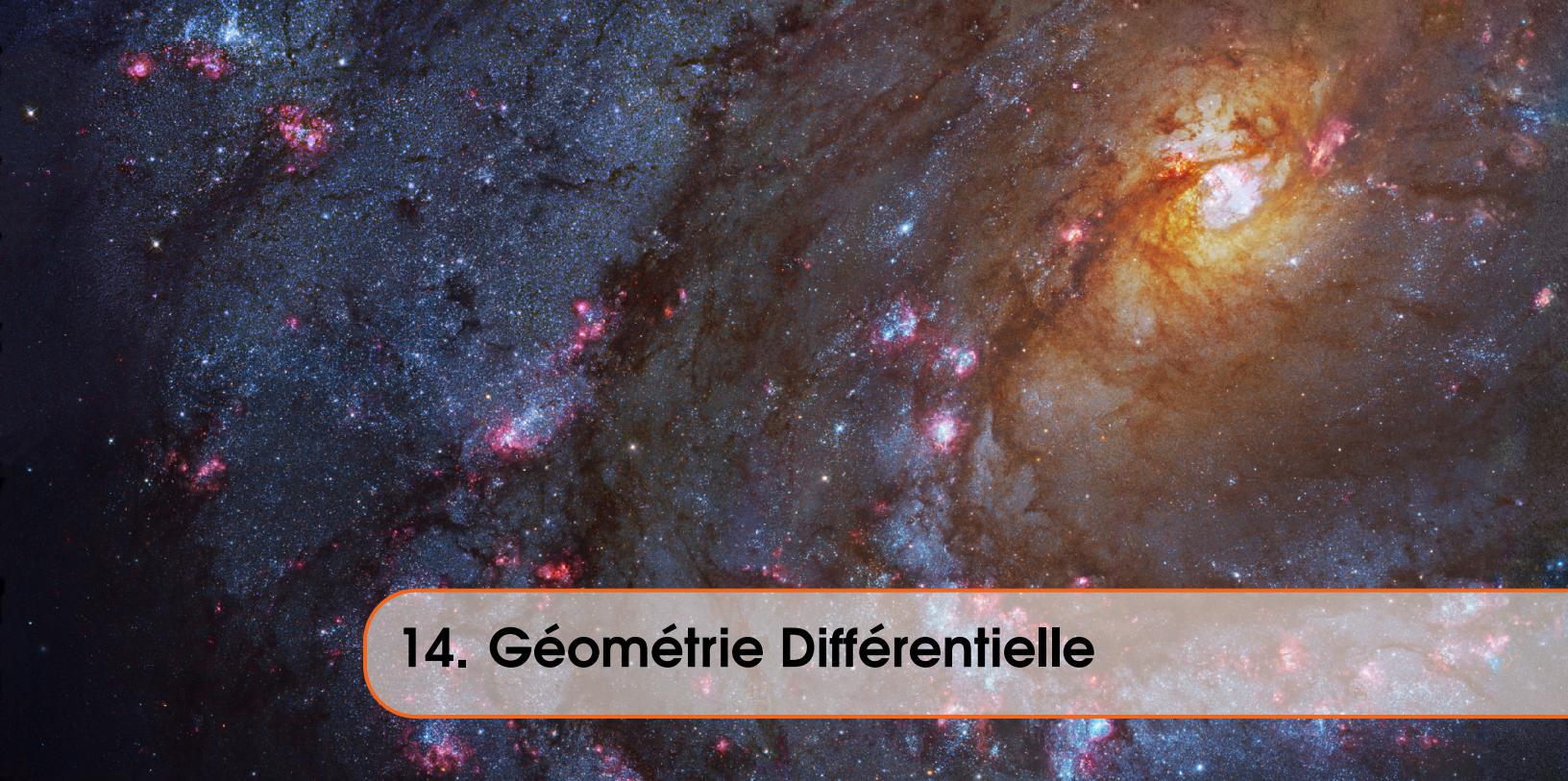
Part IV

Géométrie

Le module de géométrie pour SymPy permet de créer des entités géométriques bidimensionnelles, telles que des lignes et des cercles, et une requête d'informations sur ces entités. Cela peut inclure de demander l'aire d'une ellipse, de vérifier la colinéarité d'un ensemble de points ou de rechercher l'intersection de deux lignes. Le cas d'utilisation principal du module implique des entités avec des valeurs numériques, mais il est également possible d'utiliser des représentations symboliques.



13. Géométrie plan



14. Géométrie Différentielle



15. Groupe de Lie

15.1 Résolution d'équations par groupe de Lie

Cette astuce implémente la méthode du groupe de Lie pour résoudre les équations différentielles du premier ordre. Le but est de convertir l'équation différentielle donnée du système de coordonnées donné en un autre système de coordonnées où elle devient invariante avec le groupe de translations de Lie à un paramètre. L'ODE converti est en quadrature et peut être résolu facilement. Il utilise la `sympy.solvers.ode.infinitesimals()` fonction qui retourne les infinitésimaux de la transformation.

16. Convexity

16.0.1 Cone

Definition 16.0.1 — Cone. A set $K \in \mathbb{R}^n$, when $x \in K$ implies $\alpha x \in K$.

A non convex cone can be hyper-plane.

For convex cone $x + y \in K, \forall x, y \in K$.

Cone don't need to be "pointed". e.g.

Direct sums of cones $C_1 + C_2 = \{x = x_1 + x_2 | x_1 \in C_1, x_2 \in C_2\}$.

■ **Example 16.1** $S_1^n \{X | X = X^n, \lambda(x) \geq 0\}$

A matrix with positive eigenvalues.

euclid

Operations preserving convexity

Intersection $C \cap_{i \in \mathbb{I}} C_i$

Linear map Let $A : \mathbb{R}^n \rightarrow \mathbb{R}^m$ be a linear map. If $C \in \mathbb{R}^n$ is convex, so is $A(C) = \{Ax | x \in C\}$

Inverse image $A^{-1}(D) = \{x \in \mathbb{R}^n | Ax \in D\}$

Operations that induce convexity

Convex hull on $S = \cap \{C | S \in C, C \text{ is convex}\}$

■ **Example 16.2** $Co\{x_1, x_2, \dots, x_m\} = \{\sum_{i=1}^m \alpha_i x_i | \alpha \in \delta_m\}$

For a convex set $x \in C \Rightarrow x = \sum \alpha_i x_i$.

Theorem 16.0.1 — Carathéodory's theorem. If a point $x \in \mathbb{R}^d$ lies in the convex hull of a set P , there is a subset P' of P consisting of $d+1$ or fewer points such that x lies in the convex hull of P' . Equivalently, x lies in an r -simplex with vertices in P .

16.1 Convex Functions

Definition 16.1.1 — Convex function. Let $C \in \mathbb{R}^n$ be convex, $f : C \rightarrow \mathbb{R}$ is convex on f if $x, y \in C \times C$. $\forall \alpha \in (0, 1)$, $f(\alpha x + (1 - \alpha)y) \leq f(\alpha x) + f((1 - \alpha)y)$

Definition 16.1.2 — Strictly Convex function. Let $C \in \mathbb{R}^n$ be convex, $f : C \rightarrow \mathbb{R}$ is strictly convex on f if $x, y \in C \times C$. $\forall \alpha \in (0, 1)$, $f(\alpha x + (1 - \alpha)y) < f(\alpha x) + f((1 - \alpha)y)$

Definition 16.1.3 — Strongly convex. $f : C \rightarrow \mathbb{R}$ is strongly convex with modulus $u \geq 0$ if $f - \frac{1}{2}u\|\cdot\|^2$ is convex.

Interpretation: There is a convex quadratic $\frac{1}{2}u\|\cdot\|^2$ that lower bounds f .

■ **Example 16.3** $\min_{x \in C} f(x) \leftrightarrow \min \bar{f}(x)$ Useful to turn this into an unconstrained problem.

$$\bar{f}(x) = \begin{cases} f(x) & \text{if } x \in C \\ \infty & \text{elsewhere} \end{cases}$$

Definition 16.1.4 A function $f : \mathbb{R}^n \rightarrow \mathbb{R} \cup \infty \bar{\mathbb{R}}$ is convex if $x, y \in \mathbb{R}^n \times \mathbb{R}^n$, $\forall x, y, \bar{f}(\alpha x + (1 - \alpha)y) \leq f(\alpha x) + f((1 - \alpha)y)$

Definition 1 is equivalent to definition 2 if $f(x) = \infty$.

■ **Example 16.4** $f(x) = \sup_{j \in J} f_j(x)$

16.1.1 Epigraph

Definition 16.1.5 — Epigraph. For $f : \mathbb{R}^n \rightarrow \bar{\mathbb{R}}$, its epigraph $epi(f) \in \mathbb{R}^{n+1}$ is the set $epi(f) = \{(x, \alpha) | f(x) \leq \alpha\}$

Next: a function is convex i.f.f. its epigraph is convex.

Definition 16.1.6 A function $f : C \rightarrow \mathbb{R}$, $C \in \mathbb{R}^n$ is convex if $\forall x, y \in C$, $f(ax + (1 - a)x) \leq af(x) + (1 - a)f(y) \quad \forall a \in (0, 1)$.

Strict convex: $x \neq y \Rightarrow f(ax + (1 - a)x) < af(x) + (1 - a)f(y)$

(R) f is convex $\Rightarrow -f$ is concave.

Level set: $S_\alpha f = \{x | f(x) \leq \alpha\}$.

$S_\alpha f$ is convex $\Leftrightarrow f$ is convex.

Definition 16.1.7 — Strongly convex. $f : C \rightarrow \mathbb{R}$ is strongly convex with modulus μ if $\forall x, y \in C$, $\forall \alpha \in (0, 1)$, $f(ax + (1 - \alpha)y) \leq af(x) + (1 - \alpha)f(y) - \frac{1}{2\mu}\alpha(1 - \alpha)\|x - y\|^2$.

(R)

- f is 2nd-differentiable, f is convex $\Leftrightarrow \nabla^2 f(x) \succ 0$.
- f is strongly convex $\Leftrightarrow \nabla^2 f(x) \succ \mu I \Leftrightarrow x \geq \mu$

Definition 16.1.8 — 2. $f : \mathbb{R}^n \rightarrow \bar{\mathbb{R}}$ is convex if $x, y \in \mathbb{R}$, $\alpha \in (0, 1)$, $f(\alpha x + (1 - \alpha)y) \leq \alpha f(x) + (1 - \alpha)f(y)$.

The effective domain of f is $\text{dom } f = \{x | f(x) < +\infty\}$

■ **Example 16.5 — Indicator function.** $\delta_C(x) = \begin{cases} 0 & x \in C \\ +\infty & \text{elsewhere} \end{cases}$.
 $\text{dom } \delta_C(x) = C$

Definition 16.1.9 — Epigraph. The epigraph of f is $\text{epif} = \{(x, \alpha) | f(x) \leq \alpha\}$

The graph of epif is $\{(x, f(x)) | x \in \text{dom } f\}$.

Definition 16.1.10 — III. A function $f : \mathbb{R}^n \rightarrow \bar{\mathbb{R}}$ is

Theorem 16.1.1 $f : \mathbb{R}^n \rightarrow \bar{\mathbb{R}}$ is convex $\iff \forall x, y \in \mathbb{R}^n, \alpha \in (0, 1), f(\alpha x + (1 - \alpha)y) \leq \alpha f(x) + (1 - \alpha)f(y)$.

Proof. \Rightarrow take $x, y \in \text{dom } f$, $(x, f(x)) \in \text{epif}, (y, f(y)) \in \text{epif}$. ■

■ **Example 16.6 — Distance.** Distance to a convex set $d_c(x) = \inf\{\|z - x\| | z \in C\}$. Take any two sequences $\{y_k\}$ and $\{\bar{y}_k\} \subset C$ s.t. $\|y_k - x\| \rightarrow d_c(x)$, $\|\bar{y}_k - \bar{x}\| \rightarrow d_c(\bar{x})$. $z_k = \alpha y_k + (1 - \alpha)\bar{y}_k$.

$$\begin{aligned} d_c(\alpha x + (1 - \alpha)\bar{x}) &\leq \|z_k - \alpha x - (1 - \alpha)\bar{x}\| \\ &= \|\alpha(y_k - x) + (1 - \alpha)(\bar{y}_k - \bar{x})\| \\ &\leq \alpha\|y_k - x\| + (1 - \alpha)\|\bar{y}_k - \bar{x}\| \end{aligned}$$

Take $k \rightarrow \infty$, $d_c(\alpha x + (1 - \alpha)\bar{x}) \leq \alpha d_c(x) + (1 - \alpha)d_c(\bar{x})$ ■

■ **Example 16.7 — Eigenvalues.** Let $X \in S^n := \{n \times \text{nsymmetric matrix}\}$. $\lambda_1(x) \geq \lambda_2(x) \geq \dots \geq \lambda_n(x)$.
 $f_k(x) = \sum_1^n \lambda_i(x)$.
Equivalent characterization

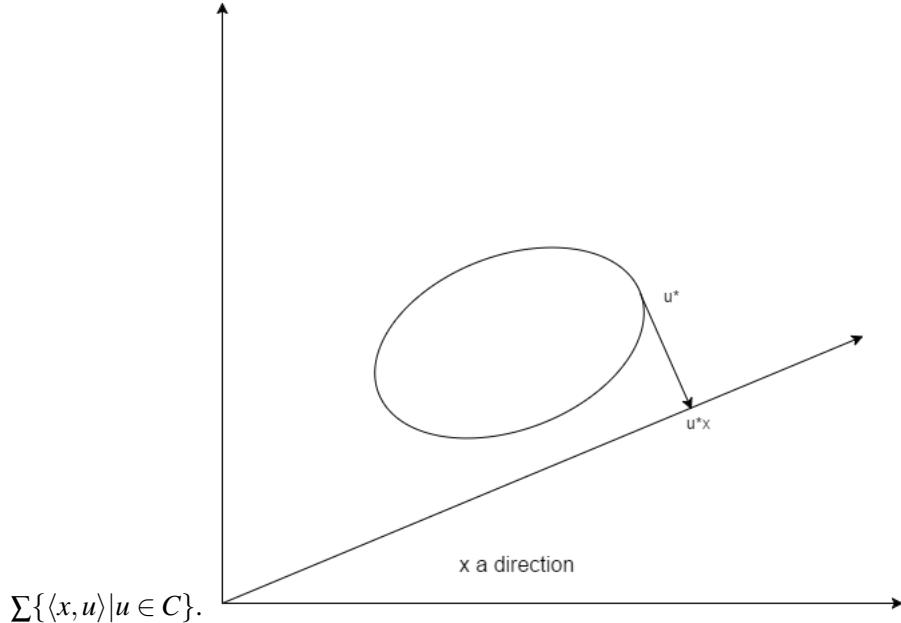
$$\begin{aligned} f_k(x) &= \max\left\{\sum_i v_i^T X v_i \mid v_i \perp v_j, i \neq j\right\} \\ &= \max\{tr(V^T X V) \mid V^T V = I_k\} \\ &= \max\{tr(VV^T X)\} \text{ by circularity} \end{aligned}$$

Note $\langle A, B \rangle = \text{tr}(A^T B)$ is true for symmetric matrix.

$$\langle A, A \rangle = |A|_F^2 = \sum_i A_{ii}^2$$

17. Support Function

Take a set $C \in \mathbb{R}^n$, not necessarily convex. The support function is $\sigma_C = \mathbb{R}^n \rightarrow \bar{\mathbb{R}}$. $\sigma_C(x) =$



Fact 17.0.1 The support function binds the supporting hyper-plane.

Supporting functions are

- Positively homogeneous

$$\sigma_C(\alpha x) = \alpha \sigma_C(x) \forall \alpha > 0$$

$$\sigma_C(\alpha x) = \sup_{u \in C} \langle \alpha x, u \rangle = \alpha \sup_{u \in C} \langle x, u \rangle = \alpha \sigma_C(x)$$

- Sub-linear (a special case of convex, linear combination holds $\forall \alpha$.)

$$\sigma_C(\alpha x + (1 - \alpha)y) = \sup_{u \in C} \langle \alpha x + (1 - \alpha)y, u \rangle \leq \alpha \sup_{u \in C} \langle x, u \rangle + (1 - \alpha) \sup_{u \in C} \langle y, u \rangle$$

■ **Example 17.1 — L2-norm.** $\|x\| = \sup_{u \in C} \{\langle x, u \rangle, u \in \mathbb{R}^n\}$.

$$\|x\|_p = \sup\{\langle x, u \rangle, u \in B_q\} \text{ where } \frac{1}{p} + \frac{1}{q} = 1. B_q = \{\|x\|_q \leq 1\}.$$

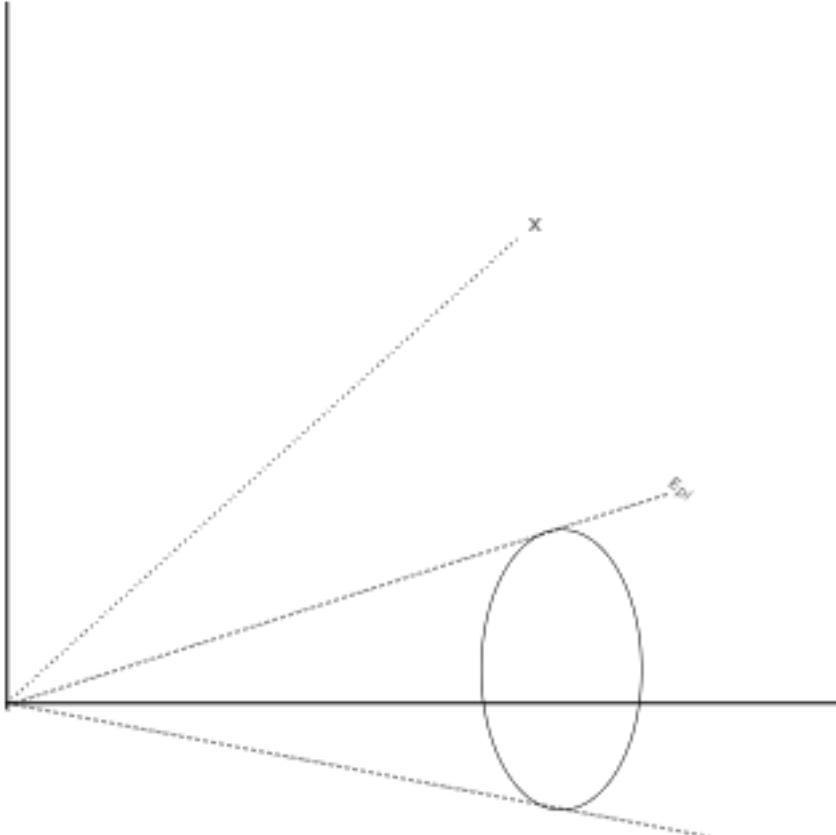
section The norm is

- Positive homogeneous
- sub-linear
- If $0 \in C$, σ_C is non-negative.
- If C is central-symmetric, $\sigma_C(0) = 0$ and $\sigma_C(x) = \sigma_C(-x)$

■

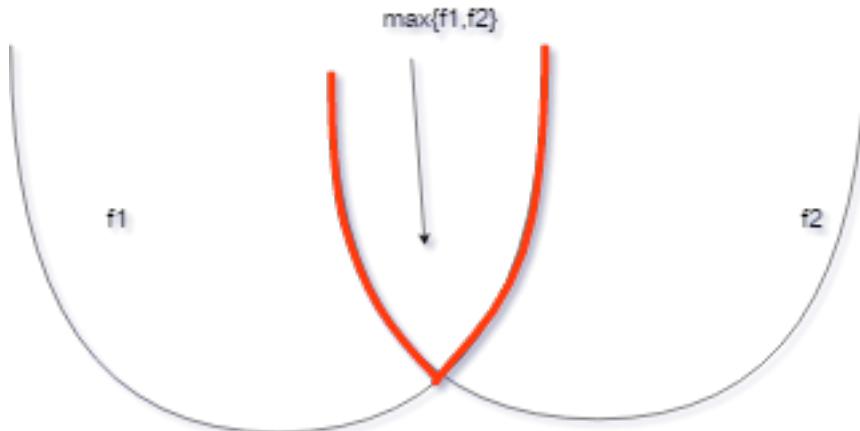
Fact 17.0.2 — Epigraph of a support function. $epi\sigma_C = \{(x, t) | \sigma_C(x) \leq t\}$. Suppose $(x, t) \in epi\sigma_C$. Take any $\alpha > 0$. $\alpha(x, t) = (\alpha x, \alpha t)$.

$$\alpha\sigma_C(x) = \alpha\sigma_C(x) \leq \alpha t. \alpha(x, c) \in epi\sigma_C$$



17.1 Operations Preserve Convexity of Functions

- Positive affine transformation
 $f_1, f_2, \dots, f_k \in cvx\mathbb{R}^n$.
 $f = \alpha_1 f_1 + \alpha_2 f_2 + \dots + \alpha_k f_k$
- Supremum of functions. Let $\{f_i\}_{i \in I}$ be arbitrary family of functions. If $\exists x \sup_{j \in J} f_j(x) < \infty \Leftrightarrow f(x) = \sup_{j \in J} f_j(x)$



- Composition with linear map.
 $f \in \text{cvx}\mathbb{R}^n, A : \mathbb{R}^n \rightarrow \mathbb{R}^m$ is a linear map. $f \circ A(x) = f(Ax) \in \text{cvx}\mathbb{R}^n$

$$\begin{aligned} f \circ A(x) &= f(A(\alpha x + (1 - \alpha)y)) \\ &= f(A\alpha x + (1 - \alpha)Ay) \\ &\leq \alpha f(Ax) + (1 - \alpha)f(Ay) \end{aligned}$$

17.2 Remarks

This is an example of a remark.

R The concepts presented here are now in conventional employment in mathematics. Vector spaces are taken over the field $\mathbb{K} = \mathbb{R}$, however, established properties are easily extended to $\mathbb{K} = \mathbb{C}$.

17.3 Corollaries

This is an example of a corollary.

Corollary 17.3.1 — Corollary name. The concepts presented here are now in conventional employment in mathematics. Vector spaces are taken over the field $\mathbb{K} = \mathbb{R}$, however, established properties are easily extended to $\mathbb{K} = \mathbb{C}$.

17.4 Propositions

This is an example of propositions.

17.4.1 Several equations

Proposition 17.4.1 — Proposition name. It has the properties:

$$||\mathbf{x}|| - ||\mathbf{y}|| \leq ||\mathbf{x} - \mathbf{y}|| \quad (17.1)$$

$$||\sum_{i=1}^n \mathbf{x}_i|| \leq \sum_{i=1}^n ||\mathbf{x}_i|| \quad \text{where } n \text{ is a finite integer} \quad (17.2)$$

17.4.2 Single Line

Proposition 17.4.2 Let $f, g \in L^2(G)$; if $\forall \varphi \in \mathcal{D}(G)$, $(f, \varphi)_0 = (g, \varphi)_0$ then $f = g$.

17.4.3 Paragraph of Text

■ **Example 17.2 — Example name.** Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris. ■

17.5 Exercises

This is an example of an exercise.

Exercise 17.1 This is a good place to ask a question to test learning progress or further cement ideas into students' minds. ■

17.6 Problems

Problem 17.1 What is the average airspeed velocity of an unladen swallow?

17.7 Vocabulary

Define a word to improve a students' vocabulary.

Vocabulary 17.1 — Word. Definition of word.

Part V

Calcul numérique et discret



18. Nombres à virgule flottante

Dans les chapitres suivants, les nombres à virgule flottante sont au cœur des calculs ; il convient de les étudier car leur comportement suit des règles précises. Comment représenter des nombres réels en machine ? Comme ces nombres ne peuvent pas en général être codés avec une quantité finie d'information, ils ne sont pas toujours représentables sur un ordinateur : il faut donc les approcher avec une quantité de mémoire finie. Un standard s'est dégagé autour d'une approximation des nombres réels avec une quantité fixe d'information : la représentation à virgule flottante. Dans ce chapitre, on trouve : une description sommaire des nombres à virgule flottante et des différents types de ces nombres disponibles dans SymPy, et la démonstration de quelques-unes de leurs propriétés. Quelques exemples montreront certaines des difficultés qu'on rencontre en calculant avec les nombres à virgule flottante, quelques astuces pour arriver parfois à les contourner, en espérant développer chez le lecteur une prudence bien nécessaire ; en conclusion, nous essayons de donner quelques propriétés que doivent posséder les méthodes numériques pour pouvoir être utilisées avec ces nombres.

19. Intégration numérique

Ce chapitre traite le calcul numérique d'intégrales (§14.1) ainsi que la résolution numérique d'équations différentielles ordinaires (§14.2) avec SymPy. Nous rappelons des bases théoriques des méthodes d'intégration, puis nous détaillons les fonctions disponibles et leur usage (§14.1.1). Le calcul symbolique d'intégrales avec SymPy a été traité précédemment (§2.3.8), et ne sera que mentionné rapidement ici comme une possibilité de calculer la valeur numérique d'une intégrale. Cette approche « symbolique puis numérique », lorsqu'elle est possible, constitue une des forces de SymPy et est à privilégier car le nombre de calculs effectués, et donc d'erreurs d'arrondi, est en général moindre que pour les méthodes d'intégration numérique. Nous donnons une rapide introduction aux méthodes classiques de résolution d'équations différentielles, puis le traitement d'un exemple (§14.2.1) débutera l'inventaire des fonctions disponibles en SymPy (§14.2.2).

19.1 Examples

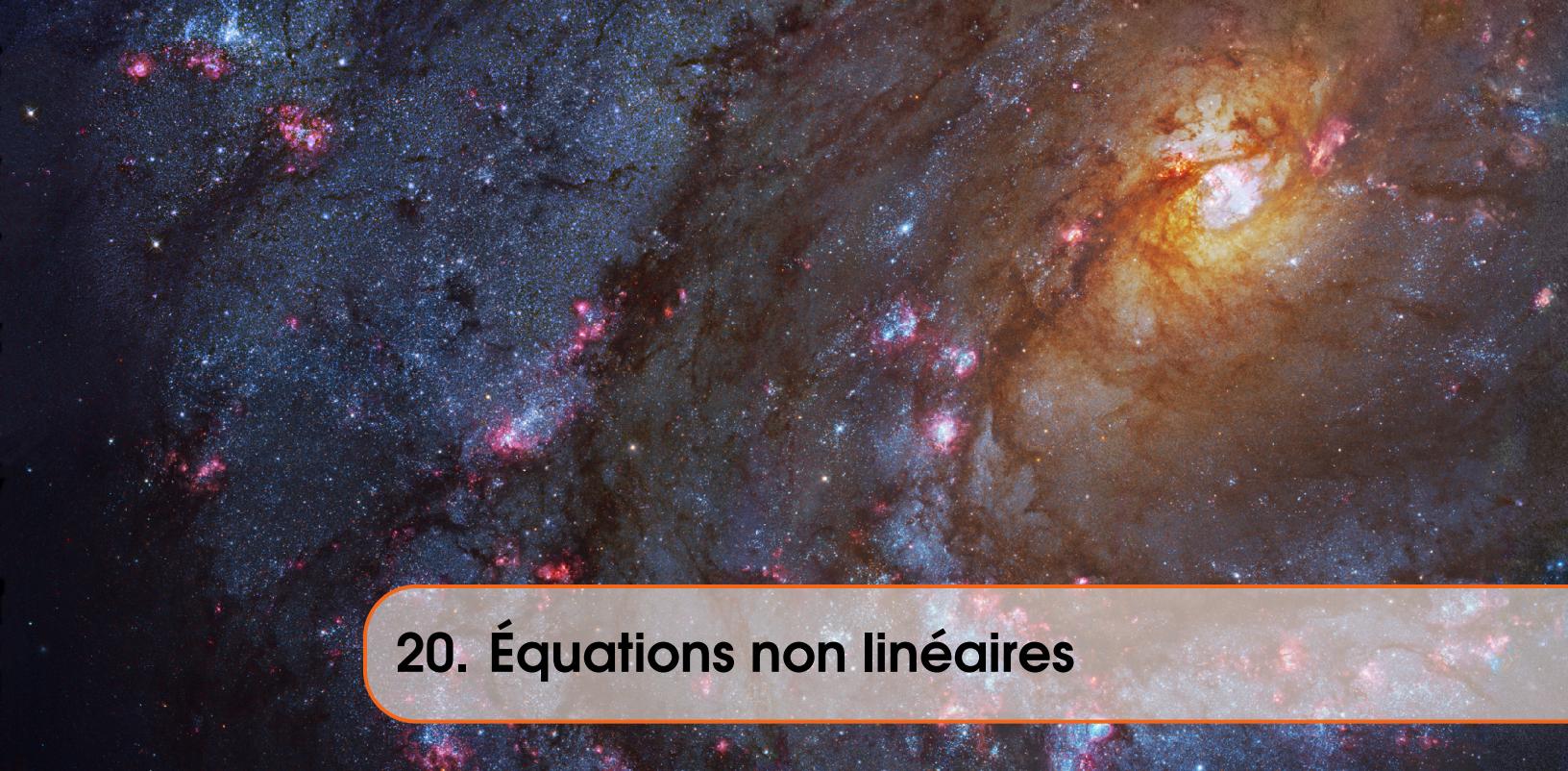
This is an example of examples.

19.1.1 Equation and Text

■ **Example 19.1** Let $G = \{x \in \mathbb{R}^2 : |x| < 3\}$ and denoted by: $x^0 = (1, 1)$; consider the function:

$$f(x) = \begin{cases} e^{|x|} & \text{si } |x - x^0| \leq 1/2 \\ 0 & \text{si } |x - x^0| > 1/2 \end{cases} \quad (19.1)$$

The function f has bounded support, we can take $A = \{x \in \mathbb{R}^2 : |x - x^0| \leq 1/2 + \varepsilon\}$ for all $\varepsilon \in]0; 5/2 - \sqrt{2}[$. ■



20. Équations non linéaires

Ce chapitre explique comment résoudre une équation non linéaire avec SymPy. Dans un premier temps on étudie les équations polynomiales et on montre les limitations de la recherche de solutions exactes. Ensuite on décrit le fonctionnement de quelques méthodes classiques de résolution numérique. Au passage on indique quels sont les algorithmes de résolution numérique implémentés dans SymPy. Qu'est ce que non-linéaire et qu'est ce que une équation algébrique

Une équation algébrique est un polynôme de la forme $P(x)$

$$\exp(-x) \sin(x) = \cos(x) \tag{20.1}$$

20.1 Équations algébriques

20.2 Figure

Treatments	Response 1	Response 2
Treatment 1	0.0003262	0.562
Treatment 2	0.0015681	0.910
Treatment 3	0.0009271	0.296

Table 20.1: Table caption

Part VI

Combinatoire

Les sujets de ce chapitre sont du néanmoins axées sur des questions ou l'approche mathématique et physique et demandé



21. Permutations

Une permutation, également appelée "numéro d'agencement" ou "ordre", est un agencement des éléments d'une liste ordonnée dans un mappage un-à-un avec lui-même. La permutation d'un arrangement donné est donnée en indiquant les positions des éléments après le réarrangement. Par exemple, si on commençait par les éléments $[x, y, a, b]$ (dans cet ordre) et qu'ils étaient réordonnés sous la forme $[x, y, b, a]$, la permutation serait alors $[0, 1, 3, 2]$. Notez que (dans SymPy) le premier élément est toujours désigné par 0 et que la permutation utilise les indices des éléments dans l'ordre d'origine, et non les éléments (a, b , etc.) eux-mêmes.



22. Semi-groupe numérique

22.1 Introduction

Part VII

Physique

Les sujets de ce chapitre sont du néanmoins axées sur des questions où l'approche mathématique et physique est demandée



23. Chaos

Prenons une pause dans l'apprentissage de nouvelles techniques et algorithmes informatiques pour un peu, et passer du temps en utilisant ce que nous avons appris jusqu'à présent pour enquêter sur quelque chose d'intéressant. Nous allons commencer avec quelque chose de familier: le simple pendule.

23.1 Pendule simple

Le pendule simple figure

23.1.1 Pendule à deux bras

23.1.2 Mouvements d'un robot

Qu'est ce qu'il faut savoir quand on veut modélisé le comportement d'un robot?. Et bien la réponse est tout simplement des mathématiques



24. Mécanique et information quantique



25. Le modèle ϕ^4

25.0.1 LES DIAGRAMMES DE FEYNMAN

Part VIII

Annexe

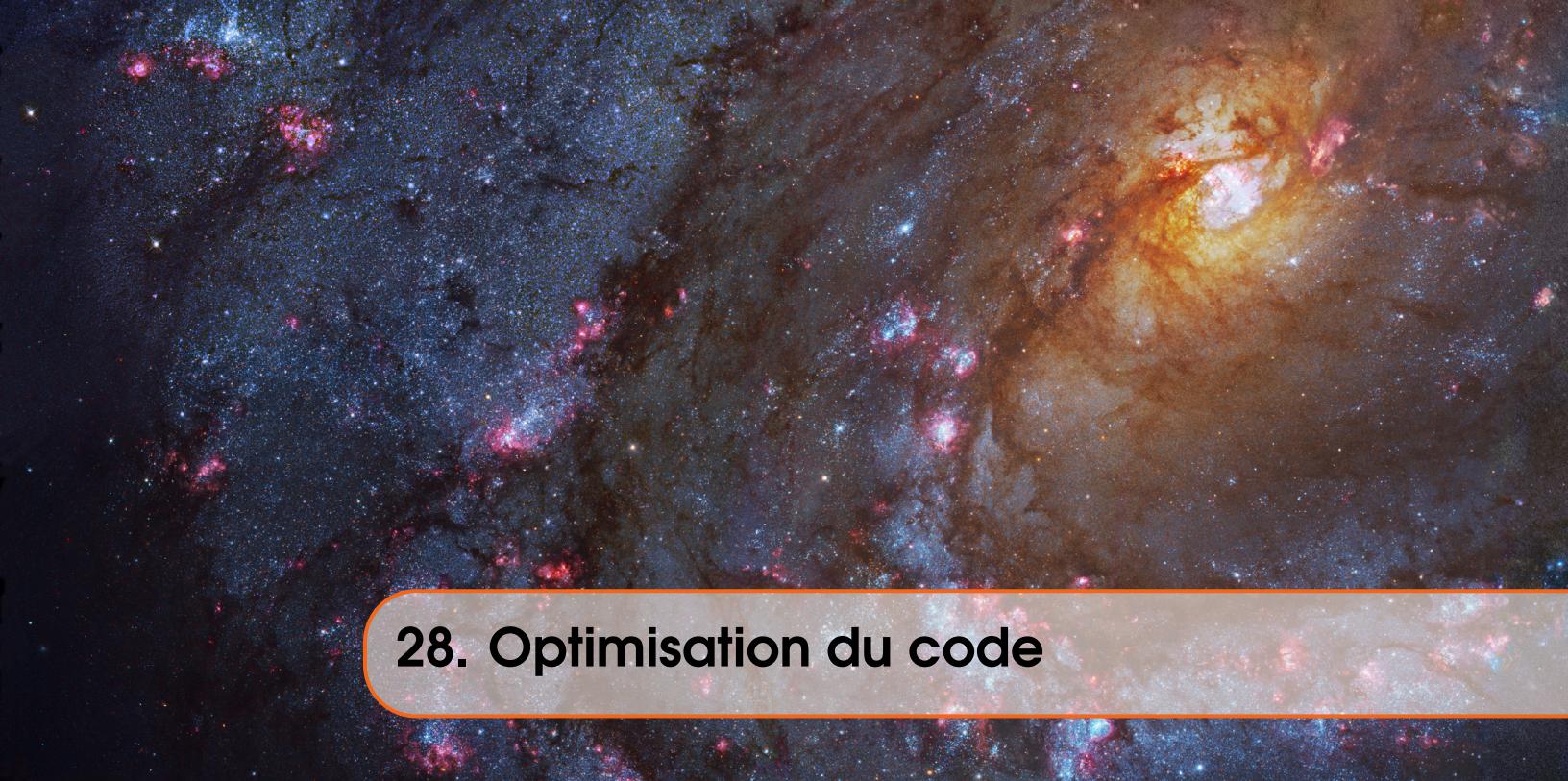


26. Programmation Orientée Objet



27. Décorateurs

Les décorateurs un mécanisme incontournable pour écrire de très bon code et purement lisible et portable



28. Optimisation du code

Sans aucun doute l'usage de la programmation symbolique avec ce que en a vue plus haut, ralentisse grandement l'exécution du programme, donc en gagne sur le coté sureté, élégance et maintenance du code et d'autre part en perd complètement la vitesse; penser à des centaine de ligne de code si vous voulez programmé un robot, voiture ou des objets connectés qui implémente des algorithmes mathématiques et qui de demande beaucoup de ressource est un temps de retour très élevées

28.0.1 Cython

Cython (<http://www.cython.org/>) est un métalangage qui permet de combiner du code Python et des types de données C, pour concevoir des extensions compilables pour Python. Dans un module Cython, il est possible de définir des variables C directement dans le code Python et de définir des fonctions C qui prennent en paramètre des variables C ou des objets Python. Cython contrôle ensuite de manière transparente la génération de l'extension C, en transformant le module en code C par le biais des API C de Python. Toutes les fonctions Python du module sont alors automatiquement publiées. Le gain de temps dans la conception introduit par Cython est considérable : toute la mécanique habituellement mise en œuvre pour créer un module d'extension est entièrement gérée par Cython. Ainsi, la fonction max() du module calculs.c précédemment présentée devient :

Les fichiers Cython ont par convention l'extension pyx, en référence à l'ancien nom.

setup.py pour calculs.pyx

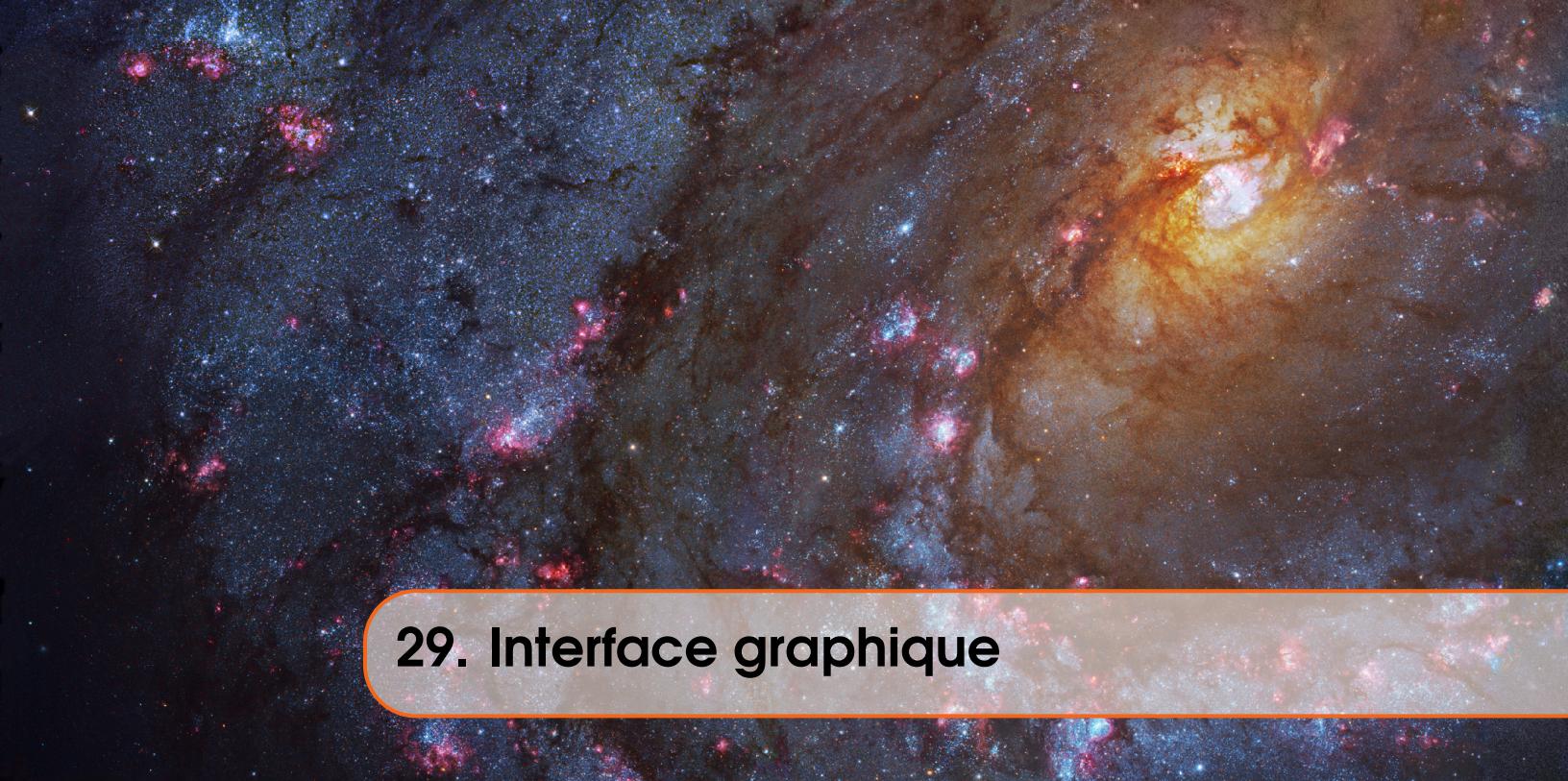
```
from distutils.core import setup
from distutils.extension import Extension
from Cython.Distutils import build_ext

extension = Extension("calculs", ["calculs.pyx"])

setup(name="calculs", ext_modules=[extension], cmdclass={'build_ext': build_ext})
```

28.0.2 Theano

Theano est une bibliothèque pour l'accélération du code lent en Python, très importante et intéressante elle offre une syntaxe très particulière.



29. Interface graphique

Quelles bibliothèque Python pour développer des applications scientifiques graphiques, le choix est difficile. D'autant qu'il y en a plusieurs pour ne cité que les plus populaires: Tkinter, Gtk, Qt, wx il existe encore d'autre bibliothèques qui sont moins conçus: Ftk

Dans cette section nous allons exposés les bibliothèques les plus populaires en mettant l'accent plus particulièrement sur deux d'entre eux: Qt et ipywidgets.

29.1 Bibliographie

29.2 Index