

# SymPy par la pratique(DRAFT(wip))

Exemple et exercice avancée

K.I.A Derouiche





# Contents

<b>0.1</b>	<b>Avant-Propos</b>	<b>5</b>
<b>0.2</b>	<b>Calcul formel</b>	<b>7</b>
0.2.1	Logiciel de système de calcul formel .....	7
0.2.2	Quelques logiciels de calcul formel .....	7
0.2.3	Pourquoi choisir SymPy? .....	8
<b>1</b>	<b>Premier pas vers SymPy .....</b>	<b>9</b>
<b>1.1</b>	<b>La bibliothèque SymPy</b>	<b>9</b>
1.1.1	Le cas de la bibliothèque SymPy .....	9
1.1.2	Travaillez avec SymPy .....	10
1.1.3	Installation de SymPy .....	10
1.1.4	isympy .....	10
1.1.5	SymPyGamma .....	10
1.1.6	SymPyLive .....	10
<b>1.2</b>	<b>SymPy comme calculatrice</b>	<b>10</b>
1.2.1	Premier calculs .....	11
1.2.2	Structure de données dans SymPy .....	11
1.2.3	Variable et affectation .....	12
1.2.4	Contrôle du flux d'instructions .....	12
<b>1.3</b>	<b>Les Fonctions</b>	<b>12</b>
<b>2</b>	<b>Algèbre et calcul formel .....</b>	<b>15</b>
2.0.1	Ensembles .....	15
2.0.2	Logique .....	16
2.0.3	Ensembles .....	16
2.0.4	Polynômes .....	17

2.0.5	Nombres parfaits et nombres chanceux . . . . .	17
<b>2.1</b>	<b>Examples</b>	<b>17</b>
2.1.1	Equation and Text . . . . .	17
<b>2.2</b>	<b>Programmation Orientée Objet</b>	<b>17</b>
2.2.1	POO . . . . .	17
2.2.2	Notions de COO et d'encapsulation . . . . .	18
<b>3</b>	<b>Problème non linéaire</b> . . . . .	<b>19</b>
<b>3.1</b>	<b>Chaos</b>	<b>20</b>
3.1.1	Pendule simple . . . . .	20
3.1.2	Pendule à deux bras . . . . .	20
3.1.3	Mouvements d'un robot . . . . .	20
<b>3.2</b>	<b>Mécanique et information quantique</b>	<b>20</b>
<b>3.3</b>	<b>Le modèle <math>\phi^4</math></b>	<b>20</b>
3.3.1	LES DIAGRAMMES DE FEYNMAN . . . . .	20
<b>3.4</b>	<b>Solution non linéaire d'équation algébrique</b>	<b>20</b>
<b>4</b>	<b>Mathématique pures</b> . . . . .	<b>21</b>
<b>4.1</b>	<b>La théorie de catégorie</b>	<b>22</b>
<b>4.2</b>	<b>Transport optimal</b>	<b>22</b>
<b>4.3</b>	<b>Figure</b>	<b>23</b>
<b>5</b>	<b>Annexe</b> . . . . .	<b>25</b>
<b>5.1</b>	<b>Programmation Orientée Objet</b>	<b>25</b>
<b>5.2</b>	<b>Décorateurs</b>	<b>25</b>
5.2.1	Optimisation du code . . . . .	25
5.2.2	Cython . . . . .	26
5.2.3	Theano . . . . .	27
<b>5.3</b>	<b>Interface graphique</b>	<b>28</b>
<b>5.4</b>	<b>Bibliographie</b>	<b>28</b>
<b>5.5</b>	<b>Index</b>	<b>28</b>

## 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 autosuffisant, 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 symbolique écrite en Python est en particulier gravitent autour de SymPy mis à part un livre de 50 pages, quelques chapitres ou des lignes de codes cité 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.

## 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?

# 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 fourni 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

---

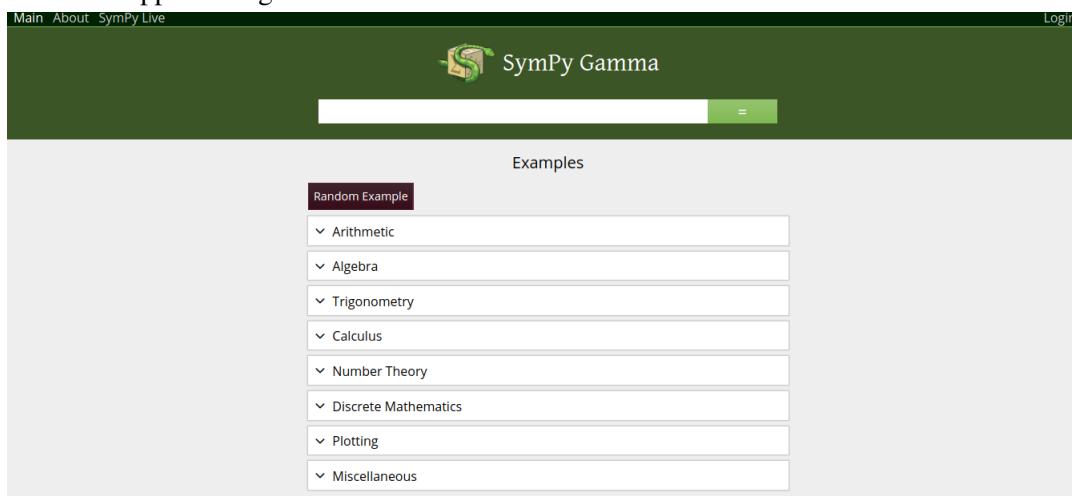
```
IPython console for SymPy 1.3 (Python 3.6.7-64-bit) (ground types: python)
These commands were executed:
>>> from __future__ import division
>>> from sympy import *
>>> x, y, z, t = symbols('x y z t')
>>> k, m, n = symbols('k m n', integer=True)
>>> f, g, h = symbols('f g h', cls=Function)
>>> init_printing()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

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 calcul

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

---

```
In [1]: import math
In [2]: math.sqrt(8)
Out[2]: 2.8284271247461903
```

---

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

---

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

---

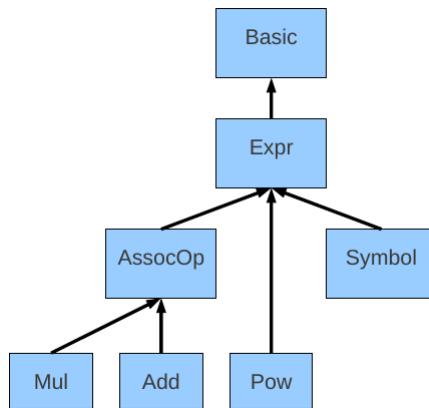
```
In [4]: x+3
Out[4]: x + 3
```

---

### 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 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**

## 1.3 Les Fonctions

This is an example of a definition. A definition could be mathematical or it could define a concept.

**Exercise 1.3** Écrire une fonction cube qui retourne le cube de son argument

**Exercise 1.4** 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 1.5** Écrire une fonction maFonction qui retourne  $f(x) = 2x^3 + x - 5$

**Exercise 1.6** 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  $\text{borneInf} < \text{borneSup}$ . 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 1.7** Écrire une fonction *volMasse* Ellipsoide 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 1.8** 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 1.9** 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 1.3** le deuxième et le troisième



## 2. Algèbre et calcul formel

### 2.0.1 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 2.1** 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 2.1** Il faut noter qu'il existe une solution qui se base sur les fonctions built-in en utilisant la structure de donnée *sets*. Mais comme en n'est pas 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

```

---

## 2.0.2 Logique

**Exercise 2.2** 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? ■

## 2.0.3 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 2.3** 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 2.2** 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 pas 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
```

---

## 2.0.4 Polynômes

**Exercise 2.4** Considérons le polynôme  $p(x) = a_0 + a_1x + a_2x^2 + a_3x^3$ , où  $a_i \neq 0 \forall i$ . Le nombre minimum de multiplications nécessaires pour évaluer  $p$  sur une entrée  $x$  est:

## 2.0.5 Nombres parfaits et nombres chanceux

**Exercise 2.5**  $\sqrt{12}$

## 2.1 Examples

This is an example of examples.

### 2.1.1 Equation and Text

■ **Example 2.1** Let  $G = \{x \in \mathbb{R}^2 : |x| < 3\}$  and denoted by:  $x^0 = (1, 1)$ ; consider the function:

$$f(x) = \begin{cases} e^{|x|} & \text{si } |x - x^0| \leq 1/2 \\ 0 & \text{si } |x - x^0| > 1/2 \end{cases} \quad (2.1)$$

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

## 2.2 Programmation Orientée Objet

**Notation 2.1.** Given an open subset  $G$  of  $\mathbb{R}^n$ , the set of functions  $\varphi$  are:

1. Bounded support  $G$ ;

2. Infinitely differentiable;

a vector space is denoted by  $\mathcal{D}(G)$ .

### 2.2.1 POO

**Exercise 2.6** Définir une classe Vecteur2D avec un constructeur fournissant les coordonnées par défaut d'un vecteur du plan (par exemple :  $x = 0$  et  $y = 0$ ). Dans le programme principal, instanciez un Vecteur2D sans paramètre, un Vecteur2D avec ses deux paramètres, et affichez-les.

**Solution 2.3** en utilise le module sympy.geometry ce module fait appel à tout les outils et theories qui peuvent être utilisés dans le cadre de la géométrie dans le Plan.

---

```
from sympy . geometry
```

---

**Exercise 2.7** Enrichissez la classe Vecteur2D précédente en lui ajoutant une méthode d'affichage et une méthode de surcharge d'addition de deux vecteurs du plan. Dans le programme principal, instanciez deux Vecteur2D , affichez-les et affichez leur somme. ■

**Solution 2.4**

### 2.2.2 Notions de COO et d'encapsulation



### **3. Problème non linéaire**

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

### 3.1 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.

#### 3.1.1 Pendule simple

Le pendule simple figure

#### 3.1.2 Pendule à deux bras

#### 3.1.3 Mouvements d'un robot

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

### 3.2 Mécanique et information quantique

### 3.3 Le modèle $\phi^4$

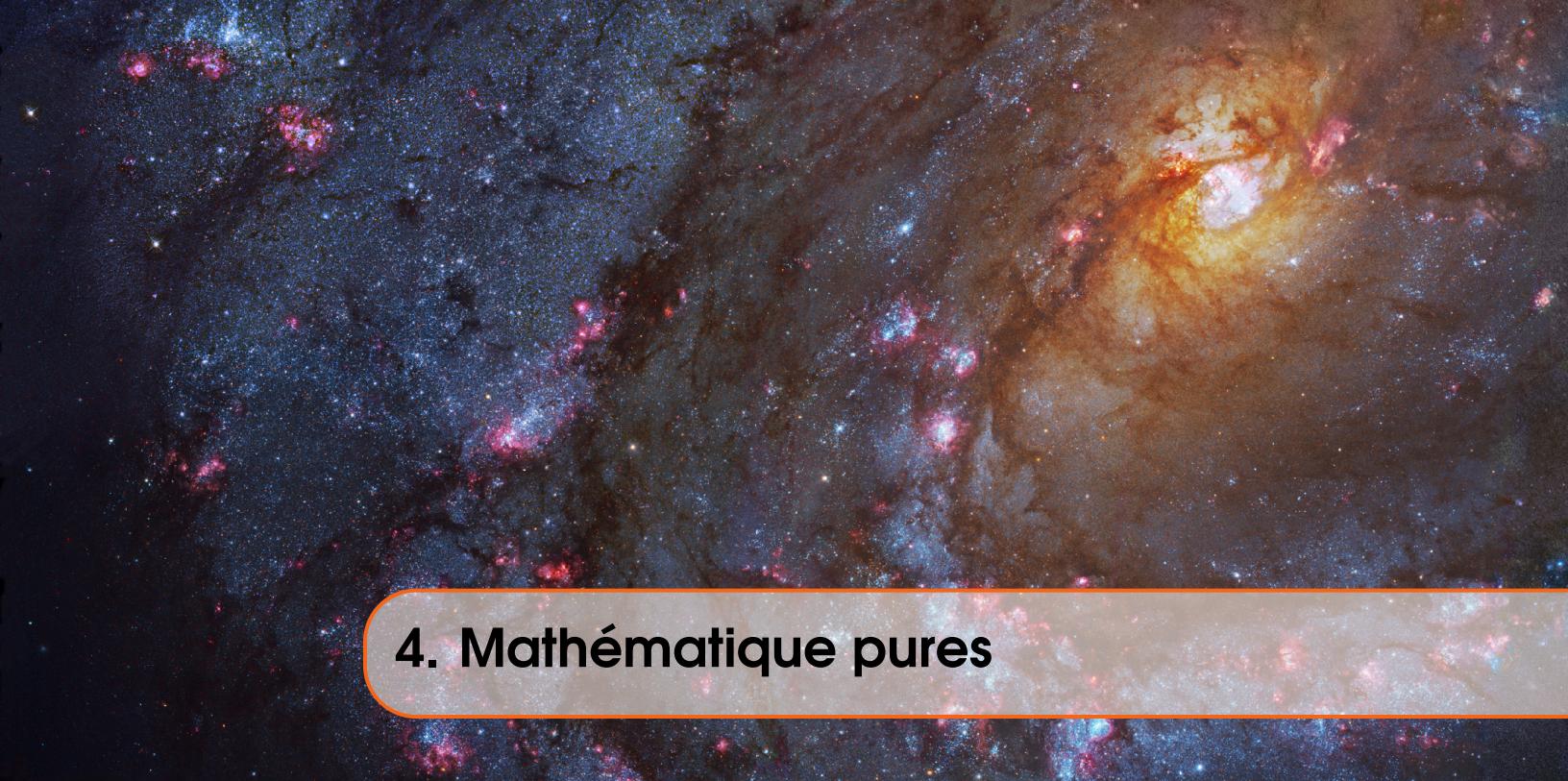
#### 3.3.1 LES DIAGRAMMES DE FEYNMAN

### 3.4 Solution non linéaire d'équation algébrique

Qu'est ce que non-linéaire et qu'est ce que une équation algébrique

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

$$\exp(-x) \sin(x) = \cos(x) \tag{3.1}$$



## 4. Mathématique pures

## 4.1 La théorie de catégorie

L'Intérêt de la théorie de catégorie dans les mathématiques modernes et l'informatique théorique.

---

```
from sympy.categories import Object, NamedMorphism
A = Object("A")
B = Object("B")
f = NamedMorphism(A, B, "f")
f.codomain
```

---

## 4.2 Transport optimal

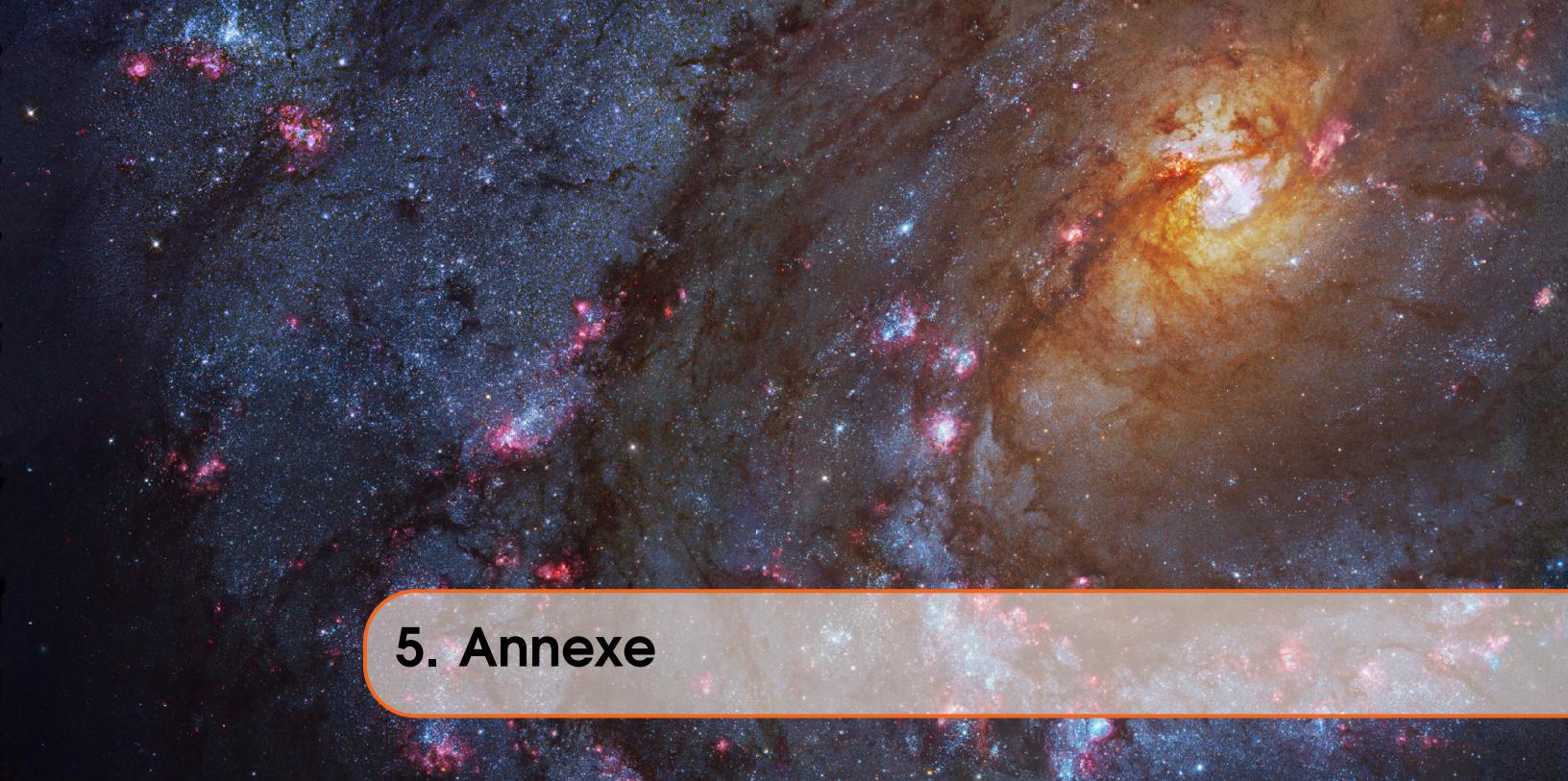
C'est quoi le *transport optimal*?, exemple simple..., le domaine du transport optimal est très

**4.3 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 4.1: Table caption





## 5. Annexe

### 5.1 Programmation Orientée Objet

### 5.2 Décorateurs

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

#### 5.2.1 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 sûreté, élégance et maintenance du code et d'autre part en perd complètement la vitesse; penser à des centaines de lignes de code si vous voulez programmer un robot, voiture ou des objets connectés qui implémentent des algorithmes mathématiques et qui demandent beaucoup de ressource est un temps de retour très élevées

### 5.2.2 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})
```

---

### 5.2.3 Theano

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

### 5.3 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'autres 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 ipywdigets.

### 5.4 Bibliographie

### 5.5 Index