INF4127 :
TPE — Calcul symbolique et utilisation avec SymPy

ESSUTHI MBANGUE ANGE ARMEL — Matricule : 24F2456
TAGNE TALLA IDRISS CHANEL — Matricule : 19M2351
DJATCHE NKAMGANG SYLVANO — Matricule : 22W2163
GOUJOU GUIMATSA ZIDANE — Matricule : 21T2899

Université de Yaoundé I
Département d'Informatique
INF4127 — Optimisation 2

2 octobre 2025

Encadrement et Contexte du Cours

- **Encadreur : Professeur MELATAGIA**
- **Cours** : INF4127 — **Optimisation 2**

Résumé / Objectif de l'exposé

- Définir et contextualiser le **calcul symbolique** (CAS), incluant son **historique** et son évolution.
- Expliquer la distinction fondamentale entre le calcul symbolique et le calcul numérique, illustrant le **workflow combiné**.
- Présenter **SymPy** : librairie Python pure, ses fonctionnalités clés et son rôle dans l'écosystème scientifique.
- Détailler des applications concrètes, avantages et limites.

Plan de la Présentation

1.1. Introduction : Le Calcul Symbolique

Définition : Le calcul symbolique, ou **algèbre informatique (CAS)**, manipule des **expressions mathématiques** et des symboles ($x, y, \pi, \sqrt{2}$) de façon **exacte**. Il travaille sur la formule elle-même, et non sur des approximations de ses valeurs.

Le Cœur de la Différence : Résoudre $x^2 - 2 = 0$ donne les symboles $\sqrt{2}$ et $-\sqrt{2}$, garantissant la précision analytique. **Intérêt principal :**

- Obtenir des **formules analytiques exactes** (pour les preuves, les identités, l'analyse de systèmes).
- Automatiser la manipulation d'expressions complexes (dérivation, intégration, simplification).

Les Grandes Étapes des Systèmes d'Algèbre Informatique (CAS)

1960s
Fondations
(FORMAC, LISP)

1970s-1980s
Premiers Géants
(Macsyma, Maple, Reduce)

1990s-Présent
Intégration & Open Source
(Mathematica, **SymPy**, SageMath)

SymPy : Pure Python, Libre, pour l'écosystème scientifique.

Conclusion : SymPy s'inscrit dans la tendance moderne à rendre les outils symboliques transparents et interopérables.

Différences clés (Détaillé)

- **Représentation** : symbolique = expressions mathématiques (arbres syntaxiques) ; numérique = nombres flottants.
- **Précision** : symbolique = **exacte** ; numérique = limitée par la précision machine (erreurs d'arrondi ou de troncature).
- **Résultat** : symbolique = une **formule** ou une preuve analytique ; numérique = une **valeur** estimée.
- **Vitesse** : numérique (ex : NumPy) est souvent plus rapide pour les calculs intensifs sur de grands ensembles de données. Le symbolique est coûteux pour les manipulations d'expressions très complexes.

2.2. Schéma : Le rôle des deux calculs

Le symbolique établit la formule exacte ; le numérique assure l'efficacité du calcul final.

CALCUL SYMBOLIQUE
(SymPy : Formule exacte)

lambdify →

← Vérification / Retour

CALCUL NUMÉRIQUE
(NumPy : Performance rapide)

Rôle de `lambdify` : Générer une fonction Python optimisée (avec les structures NumPy) directement à partir de l'expression symbolique simplifiée. C'est le pont essentiel pour la performance.

2.3 Illustration — dérivée symbolique vs approximation numérique

```
from sympy import symbols, diff, sin
# Le symbole x repr sente une variable math matique
x = symbols('x')
f = sin(x)*x

# D riv e symbolique: r sultat exact (formule)
f_prime_sym = diff(f, x)    # -> x*cos(x) + sin(x)
print(f"D riv e exacte: {f_prime_sym}")
print(f"Valeur en x=1: {f_prime_sym.subs(x, 1).evalf()
    }")

# D riv e num rique (approximation) par
    diff rences finies
import math
def f_num(t): return (t * math.sin(t))
h = 1e-6
t0 = 1.0
```

3.1. Domaines d'application du calcul symbolique

Le calcul symbolique est essentiel chaque fois qu'une solution **analytique** ou une **preuve formelle** est nécessaire.

- **Optimisation** : Calcul exact du **Gradient** et de la **Matrice Hessienne** pour les algorithmes d'optimisation (Newton, descente de gradient).
- **Analyse** : Calcul différentiel (dérivées partielles) et intégral symbolique (intégrales définies et indéfinies).
- **Équations** : Résolution de certains systèmes d'équations (polynômes, différentielles ordinaires).
- **Algèbre Linéaire** : Calculs exacts sur les matrices (déterminants, inverse) avec des coefficients symboliques.
- **Génération de Code** : Traduction d'expressions complexes en fonctions numériques optimisées pour d'autres bibliothèques (NumPy, C/Fortran).

3.2. Présentation Détaillée de SymPy

- **SymPy** est une bibliothèque **Python open-source** pour le calcul symbolique.
- Elle est entièrement écrite en **Python pur** (ne dépend que de `mpmath` pour l'arithmétique de haute précision).
- **Avantage Clé** : Utiliser la puissance d'un CAS dans le langage de programmation scientifique le plus populaire (Python).
- **Intégration** : Fonctionne parfaitement avec **Jupyter** (pour le rendu $\LaTeX$), **NumPy** et **Matplotlib** (via `lambdify`).
- **Philosophie** : Être un CAS complet tout en conservant un code simple et extensible.

4.1. Symboles, fractions exactes et évaluation

Clé de SymPy : Utiliser `symbols` et `Rational` pour garantir une représentation exacte.

```
from sympy import symbols, Rational, sqrt
x = symbols('x')

# Rational(a, b) cr e une fraction exacte a/b
expr = (x + 1)**2 / Rational(2, 3) + sqrt(2)

print(expr)
# -> 3*(x + 1)**2/2 + sqrt(2) (affichage symbolique)
#  valuation num rique  15 d cimales
print(expr.subs(x, 1).evalf(15))
# -> 7.41421356237310
```

4.2. Manipulation algébrique

Les fonctions manipulent l'arbre d'expression pour appliquer les règles algébriques.

```python
from sympy import expand, factor, simplify
x, y = symbols('x y')
# D veloppement d'une expression
expr_dev = (x + 1)**3
print(f"D velopp : {expand(expr_dev)}")
# -> x**3 + 3*x**2 + 3*x + 1

# Factorisation d'un polyn me
expr_fac = x**3 + 3*x**2 + 3*x + 1
print(f"Factoris : {factor(expr_fac)}")
# -> (x + 1)**3

# Simplification d'une expression rationnelle
print(f"Simplifi : {simplify((x**2 - 1)/(x - 1))}")
# -> x + 1
```

4.3. Dérivation exacte et Gradient

Calcul du gradient (vecteur des dérivées partielles) — essentiel en Optimisation 2.

```
from sympy import symbols, diff, cos
x, y = symbols('x y')
# Fonction de co t (exemple)
f = x**2 * y**3 + cos(x)

# D riv e partielle par rapport   x ( f / x )
df_dx = diff(f, x)
print(f" f / x  (Gradient composante x) = {df_dx}")
# -> 2*x*y**3 - sin(x)

# D riv e partielle par rapport   y ( f / y )
df_dy = diff(f, y)
print(f" f / y  (Gradient composante y) = {df_dy}")
# -> 3*x**2*y**2
```

4.4. Le Pont Numérique : lambdify

Le mécanisme pour transformer le résultat exact en code numérique rapide (NumPy).

```
from sympy import lambdify, sin
import numpy as np
# x a  t  d fini pr c demment comme symbols('x')
f_sym = x**2 + sin(x) / x # Expression symbolique

# lambdify convertit l'expression symbolique en une fonction NumPy
f_num = lambdify(x, f_sym, 'numpy')

#  valuation  rapide avec un tableau NumPy
x_vals = np.linspace(0.1, 10, 5) #  viter  x=0
y_vals = f_num(x_vals) # Calcul vectoriel ultra-rapide
print(f"R sultat du calcul vectoris  (array): {y_vals}")
```

5. Avantages et limites de SymPy

Avantages

- **Exactitude** : Fournit des résultats mathématiques exacts, cruciaux pour la vérification ou la modélisation analytique.
- **Code Python** : Gratuit, open-source, facile à intégrer dans l'écosystème Python existant (NumPy, SciPy).
- **Pédagogie** : Excellent pour l'enseignement car il montre les étapes et la structure des formules.

Limites

- **Performance** : Moins rapide que les systèmes commerciaux pour des expressions géantes (problème d'explosion combinatoire de la simplification).
- **Complexité** : Le concept de "forme la plus simple" peut nécessiter l'utilisation de fonctions de simplification spécifiques.
- **Résolvabilité** : Ne peut pas garantir une solution analytique pour toutes les équations mathématiques complexes.

6. Conclusion et Perspectives

- Le calcul symbolique, avec **SymPy**, est un outil fondamental garantissant la **précision** des modèles mathématiques.
- Le workflow **Symbolique $\rightarrow$ Numérique** permet de bénéficier à la fois de l'exactitude des formules et de l'efficacité de l'exécution numérique.
- **Perspectives** : Utiliser SymPy comme base pour la **dérivation automatique** des fonctions coût en Optimisation, et explorer ses capacités d'intégration de code optimisé.

Références Approfondies

1. **Documentation Officielle SymPy** : Le guide essentiel pour toutes les fonctions et modules.
   https://docs.sympy.org/
2. **SymPy Live Shell (Environnement Interactif)** : Pour tester SymPy directement dans votre navigateur.
   https://live.sympy.org/
3. **Article Scientifique Clé (2017)** : Aaron Meurer et al., "SymPy : symbolic computing in Python", *PeerJ Computer Science*.
   https://doi.org/10.7717/peerj-cs.103
4. **Outils Similaires** : Wolfram Research (*Mathematica*) et Waterloo Maple Inc. (*Maple*).

Défis pour la salle :

- Factoriser le polynôme $x^4 - y^4$.
- Calculer la limite de $\frac{1-\cos(x)}{x^2}$ lorsque $x \to 0$.

```python
from sympy import factor, symbols, limit, cos
x, y = symbols('x y')

# 1. Factoriser
print(factor(x**4 - y**4))
# -> (x - y)*(x + y)*(x**2 + y**2)

# 2. Calculer la limite
print(limit((1 - cos(x))/x**2, x, 0))
# -> 1/2
```

Merci pour votre attention !

Questions ?