Calcul symbolique et utilisation de SymPy

Du concept mathématique à l'outil pratique

Participants :

DONGMO TCHOUMENE ANITA BELVIANE 22W2184

DONFACK SYNTHIA CALORINE 22U2073

BOKOU-BOUNA-ANGE-LARISSA 22W2188

JIATSA ROMMEL JUNIOR 22T2906

Superviseur :

Pr Paulin MELATAGIA

Année académique : 2024 – 2025

Plan de l'exposé

- Mise en situation et définition du calcul symbolique
- Importance et applications
- Outils de calcul symbolique
- Introduction à SymPy et philosophie
- Premiers pas : installation et exemples guidés
- Fonctionnalités clés : dérivation, intégration, résolution d'équations, systèmes, matrices, optimisation
- Passage du symbolique au numérique et workflow complet
- Bonnes pratiques et limitations
- Cas d'usage réel (ex. : Machine Learning)
- Récapitulatif et ressources pour aller plus loin
- Message final et conclusion
- Questions

Une question simple...

Quelle est la racine carrée de 8 ?

Avec une calculatrice

$$\sqrt{8} = 2.828427124...$$

En mathématiques

$$\sqrt{8} = 2\sqrt{2}$$

Deux approches différentes : numérique vs symbolique

Un problème plus concret

Situation

Vous devez résoudre : $x^2 - 2 = 0$

Approche numérique :

- Essai-erreur : $x \approx 1.414213562...$
- Résultat approximatif
- Erreur d'arrondi possible

Approche symbolique :

- Solution exacte : $x = \pm\sqrt{2}$
- Pas d'approximation
- Forme mathématique pure

$\Rightarrow$ Le calcul symbolique manipule des formules, pas des nombres !

Qu'est-ce que le calcul symbolique ?

Definition

Le **calcul symbolique** (ou calcul formel) est la manipulation d'expressions mathématiques sans les évaluer numériquement.

Principe

On travaille avec des **symboles** (lettres, formules) comme en algèbre classique :

- $(x+y)^2 = x^2 + 2xy + y^2$ (développement)
- $x^2 - 1 = (x-1)(x+1)$ (factorisation)
- $\frac{d}{dx}(x^2) = 2x$ (dérivation)

Opposé au calcul numérique

Le calcul numérique remplace immédiatement par des valeurs approchées (flottants)

Analogie : l'architecte et le constructeur

Calcul symbolique

= **L'architecte**

- Dessine les plans
- Manipule des formules
- Vision d'ensemble
- Résultats exacts

Calcul numérique

= **Le constructeur**

- Exécute les calculs
- Utilise des mesures
- Cas concrets
- Résultats approchés

Les deux sont nécessaires et complémentaires !

Pourquoi avons-nous besoin du calcul symbolique ?

1. **Pour l'exactitude**
   - En mathématiques, $\sqrt{2}$ est **exactement** $\sqrt{2}$, pas 1.414...
2. **Pour les démonstrations**
   - Prouver que $(a+b)^2 = a^2 + 2ab + b^2$ pour tous a et b
3. **Pour dériver des formules générales**
   - Calculer $\frac{d}{dx}(x^n) = nx^{n-1}$ une fois pour toutes
4. **Pour l'optimisation et l'IA**
   - Calculer des gradients de fonctions complexes automatiquement
   - Simplifier des modèles mathématiques

Applications concrètes

1. **Enseignement** : vérifier les calculs d'étudiants
2. **Recherche** : démonstrations automatisées, exploration mathématique
3. **Physique** : résoudre des équations différentielles analytiquement
4. **Machine Learning** : calculer les gradients de fonctions de coût
5. **Ingénierie** : conception de circuits, analyse de systèmes
6. **Finance** : modèles d'évaluation d'options, calculs actuariels

Les outils de calcul symbolique

Systèmes de Calcul Formel (CAS)

Logiciels capables de manipuler des expressions mathématiques symboliques

Outils commerciaux :

- Mathematica (Wolfram)
- Maple (Maplesoft)
- MATLAB Symbolic Toolbox

Outils open-source :

- SymPy (Python) ← **Notre sujet !**
- Maxima
- SageMath

Et SymPy dans tout ça ?

Definition

SymPy = Bibliothèque Python pour le calcul symbolique

Pourquoi SymPy ?

1. **Gratuit et open-source** : accessible à tous
2. **En Python** : langage populaire en Data Science
3. **Léger** : pure Python, pas de dépendances lourdes
4. **Intégré** : fonctionne avec NumPy, Matplotlib, Jupyter...
5. **Extensible** : on peut ajouter nos propres fonctions

La philosophie de SymPy

Apporter le calcul symbolique à tous, gratuitement, en Python

Avantages

- Syntaxe lisible
- Courbe d'apprentissage douce
- Communauté active
- Documentation riche

Limites

- Moins rapide que Mathematica
- Certains domaines moins développés
- Performance sur très gros calculs

$\Rightarrow$ Idéal pour l'apprentissage et la plupart des applications !

Premiers pas : installation

Installation simple

```
pip install sympy
```

Premier programme

```python
import sympy as sp

# Créer un symbole (une variable)
x = sp.symbols('x')

# Créer une expression
expr = x**2 + 2*x + 1

# Factoriser
resultat = sp.factor(expr)
print(resultat)  # Affiche: (x + 1)**2
```

C'est tout ! Vous faites du calcul symbolique !

Exemple guidé 1 : Développement et factorisation

Problème : Développer $(x+3)^2$ puis factoriser le résultat

```
import sympy as sp
x = sp.symbols('x')

# Développer
developpement = sp.expand((x + 3)**2)
print(developpement)
# Résultat: x**2 + 6*x + 9

# Factoriser
factorisation = sp.factor(x**2 + 6*x + 9)
print(factorisation)
# Résultat: (x + 3)**2
```

Observation

SymPy retrouve la forme originale ! Il manipule les structures mathématiques.

Exemple guidé 2 : Simplification

Problème : Simplifier $\frac{x^2-1}{x-1}$

```
x = sp.symbols('x')
expr = (x**2 - 1) / (x - 1)

# Simplifier
simplifie = sp.simplify(expr)
print(simplifie)
# Résultat: x + 1
```

Explication :

- SymPy reconnaît que $x^2 - 1 = (x-1)(x+1)$
- Il simplifie : $\frac{(x-1)(x+1)}{x-1} = x + 1$

Comme vous le feriez sur papier, mais automatiquement !

La puissance du calcul symbolique : la dérivation

En cours de maths : Dériver $f(x) = x^3 \sin(x)$
Vous appliquez la règle du produit, triez les termes...

Avec SymPy :

```
x = sp.symbols('x')
f = x**3 * sp.sin(x)

# Dériver
df = sp.diff(f, x)
print(df)
# Résultat: x**3*cos(x) + 3*x**2*sin(x)
```

Plus fort encore

Dérivée seconde ? `sp.diff(f, x, 2)`
Dérivée n-ième ? `sp.diff(f, x, n)`

L'intégration symbolique

Problème : Calculer $\int xe^x \, dx$

```
x = sp.symbols('x')
f = x * sp.exp(x)

# Intégrale indéfinie (primitive)
primitive = sp.integrate(f, x)
print(primitive)
# Résultat: (x - 1)*exp(x)
```

Vérification : Dérivons la primitive

```
verif = sp.diff(primitive, x)
print(verif)
# Résultat: x*exp(x)  -> C'est bien f !
```

SymPy trouve la primitive quand elle existe en forme close

Résoudre des équations

Problème : Résoudre $x^2 - 5x + 6 = 0$

```
x = sp.symbols('x')
equation = sp.Eq(x**2 - 5*x + 6, 0)

# Résoudre
solutions = sp.solve(equation, x)
print(solutions)
# Résultat: [2, 3]
```

Équations plus complexes :

```
# Équation avec racines
eq = sp.Eq(x**2 - 2, 0)
sol = sp.solve(eq, x)
print(sol)
# Résultat: [-sqrt(2), sqrt(2)]
```

Forme exacte, pas d'approximation !

Systèmes d'équations

Problème : Résoudre le système

$$\begin{cases} 2x + y = 5 \\ x - y = 1 \end{cases}$$

```
x, y = sp.symbols('x y')
eq1 = sp.Eq(2*x + y, 5)
eq2 = sp.Eq(x - y, 1)

# Résoudre le système
solution = sp.solve([eq1, eq2], [x, y])
print(solution)
# Résultat: {x: 2, y: 1}
```

Application

Utile pour : intersection de courbes, systèmes dynamiques, optimisation sous contraintes...

Algèbre linéaire symbolique

Créer une matrice avec des symboles :

```
x, y = sp.symbols('x y')
M = sp.Matrix([[1, x],
               [y, 1]])

# Déterminant
det = M.det()
print(det)  # Résultat: 1 - x*y

# Valeurs propres
valeurs_propres = M.eigenvals()
print(valeurs_propres)
```

Pourquoi c'est utile ?

- Étudier la stabilité de systèmes (valeurs propres)
- Analyse paramétrique (selon x et y)
- Démonstrations théoriques

Application : Optimisation

Problème : Minimiser $f(x, y) = (x - 1)^2 + (y + 2)^2$

```
x, y = sp.symbols('x y')
f = (x - 1)**2 + (y + 2)**2

# Gradient (dérivées partielles)
grad_x = sp.diff(f, x)  # 2*(x - 1)
grad_y = sp.diff(f, y)  # 2*(y + 2)

# Point critique (gradient = 0)
point_critique = sp.solve([grad_x, grad_y], [x, y])
print(point_critique)
# Résultat: {x: 1, y: -2}
```

Interprétation

Le minimum est atteint en $(1, -2)$ avec $f(1, -2) = 0$

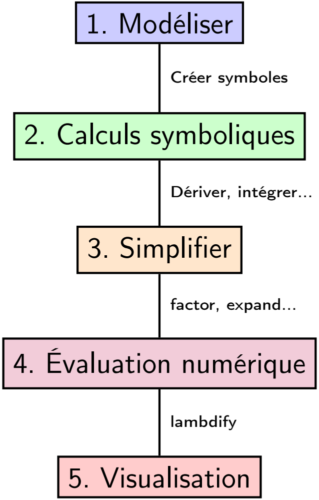
Pont entre symbolique et numérique : lambdify

Problème : J'ai une formule symbolique, je veux l'évaluer rapidement

```
import numpy as np
import matplotlib.pyplot as plt

x = sp.symbols('x')
f = sp.sin(x) * sp.exp(-x**2)

# Convertir en fonction numérique
f_num = sp.lambdify(x, f, 'numpy')

# Tracer la courbe
x_vals = np.linspace(-3, 3, 500)
y_vals = f_num(x_vals)

plt.plot(x_vals, y_vals)
plt.show()
```

Le meilleur des deux mondes : exactitude + vitesse !

Workflow typique avec SymPy



Exemple

Modèle $\rightarrow$ Gradient $\rightarrow$ Simplification $\rightarrow$ Optimisation numérique $\rightarrow$

Bonnes pratiques

1. **Déclarer les propriétés des symboles**
   - `x = sp.symbols('x', real=True, positive=True)`
   - Aide SymPy à mieux simplifier
2. **Simplifier régulièrement**
   - Les expressions peuvent devenir complexes
   - Utiliser `simplify()`, `factor()`, `expand()`
3. **Utiliser lambdify pour l'évaluation intensive**
   - Gain de vitesse $\times$ 100 à $\times$ 1000
4. **Documenter avec LaTeX**
   - `sp.latex(expr)` génère le code LaTeX
   - Parfait pour les rapports

Limitations à connaître

1. **Certains problèmes n'ont pas de solution en forme close**
   - Exemple : $\int e^{x^2} dx$ (pas de primitive élémentaire)
2. **Performance sur gros calculs**
   - Expressions très complexes peuvent être lentes
   - Solution : simplifier ou passer au numérique
3. **Moins complet que Mathematica/Maple**
   - Certains domaines avancés moins développés
   - Mais suffisant pour 95% des cas !

Conseil

Comprendre quand utiliser symbolique vs numérique est une compétence clé

Cas d'usage réel : Machine Learning

Contexte : Calculer le gradient d'une fonction de coût

Sans SymPy :

- Dériver à la main (erreurs possibles)
- Approximations numériques (imprécis)

Avec SymPy :

- Définir la fonction de coût symboliquement
- Calculer le gradient automatiquement
- Vérifier les formules avant implémentation
- Générer le code optimisé

Résultat

Gain de temps + réduction d'erreurs + meilleure compréhension

Récapitulatif : Le parcours conceptuel

1. **Calcul symbolique** = manipuler des formules, pas des nombres
2. **Pourquoi ?** Exactitude, démonstrations, formules générales
3. **SymPy** = outil Python gratuit pour le calcul symbolique
4. **Applications** : dérivation, intégration, équations, optimisation...
5. **Pont** : lambdify relie symbolique et numérique
6. **Workflow** : modéliser $\rightarrow$ simplifier $\rightarrow$ évaluer

SymPy = le langage mathématique rencontre la programmation

Pour aller plus loin

Documentation officielle :

- `https://docs.sympy.org/latest/tutorial/`
- Tutoriels interactifs

Pratiquer :

- Jupyter Notebooks + SymPy
- SymPy Live : `https://live.sympy.org`
- Exemples sur GitHub

Communauté :

- Forum : SymPy Google Group
- Stack Overflow (tag : sympy)
- GitHub : contributions bienvenues !

Message final

Le calcul symbolique n'est pas réservé aux mathématiciens théoriciens.

C'est un outil pratique pour quiconque travaille avec des formules.

SymPy le rend accessible à tous !

Conclusion

Ce que vous devez retenir

- Le calcul symbolique manipule des **structures mathématiques**
- SymPy est un **CAS gratuit en Python**
- Applications : enseignement, recherche, Data Science, ingénierie
- Workflow : symbolique $\rightarrow$ simplification $\rightarrow$ numérique
- `lambdify` est la clé pour la performance

Expérimentez ! La meilleure façon d'apprendre est de pratiquer.

Questions ?

Merci pour votre attention !

Master I Data Science
Université de Yaoundé I

Superviseur : Pr Paulin MELATAGIA