TPE : Calcul Symbolique

Groupe
BONO MBELLE AURELIEN — Matricule 24F2459
TAPAH NGASSA CLAUDIA — Matricule 20V2342
BITA ANGO'O WILLIAL MARRION — Matricule 18T2779

Filière Informatique — Option: Data Science

Université de Yaoundé I
Superviseur : Pr. Paulin MELATAGIA

2 octobre 2025

Sommaire

Présentation du calcul scientifique

Définition : branche des mathématiques appliquées utilisant des méthodes numériques et symboliques pour résoudre des problèmes (EDO, optimisation, modélisation). En optimisation, il permet de : vérifier la convexité (2 dérivée / Hessienne), obtenir les gradients/Hessiennes exacts, convertir en fonctions numériques (lambdify) et lancer des expériences de descente

Exemple : résoudre $x^2 - 2 = 0 \Rightarrow$ résultats exacts : $\pm\sqrt{2}$, pas seulement une approximation ± 1.414...

Applications :

- Simplification d'expressions.
- Résolution d'équations algébriques et différentielles.
- Calcul formel : dérivées, intégrales, limites.
- Algèbre linéaire symbolique.
- Utilisations : physique, optimisation, ingénierie...

Théorèmes et corollaires utiles

Convexité (1D)

Si $f : \mathbb{R} \to \mathbb{R}$ est C^2 et $f''(x) \geq 0 \ \forall x$ alors f est convexe.

Hessienne (nD)

Si $f : \mathbb{R}^n \to \mathbb{R}$ est C^2 et sa Hessienne $H_f(x)$ est semi-définie positive $\forall x$, alors f est convexe.

Corollaire

Obtenir un ∇f exact (symbolique) réduit les erreurs d'approximation lors d'une descente de gradient.

Dériver fonctions coût & appliquer la descente de gradient

- Ex. : coût MSE univarié : $J(a, b) = \frac{1}{n} \sum_i (y_i - (ax_i + b))^2$.
  Calcul symbolique : $\nabla J = \left(\frac{\partial J}{\partial a}, \frac{\partial J}{\partial b}\right)$.
- Descente : $\theta \leftarrow \theta - \eta \nabla J(\theta)$ (avec gradient obtenu symboliquement).
- Avantage : exactitude théorique des formules ; puis conversion en numérique pour exécution.

Présentation de Sympy

- Bibliothèque Python de calcul symbolique (CAS).
- Entièrement en Python, simple et extensible.
- Installation : `pip install sympy`.
- Objet de base : le symbole (`Symbol()`).
- Manipulation d'expressions : $+$, $-$, $\times$, $\div$.
- Calculs : simplification, dérivées, équations, matrices.
- `sympify()` : conversion en expression SymPy.
- `evalf()` : évaluation numérique haute précision.
- `lambdify()` : conversion en fonction Python/NumPy.
- Utilisation en script, terminal ou Jupyter Notebook (graphiques).

Théorèmes utiles et application avec Sympy(1/2)

- **Théorème de dérivation** : $\frac{d}{dx}[f(x)+g(x)] = f'(x)+g'(x)$, $\frac{d}{dx}[f(x)g(x)] = f'(x)g(x)+f(x)g'(x)$.
  SymPy : `diff(expr, x)` applique ces règles.
- **Convexité** : En 1D, $f''(x) \geq 0 \Rightarrow f$ convexe.
  En nD, $H_f(x) \succeq 0 \Rightarrow f$ convexe.
  SymPy : `diff(expr, x, 2)`, `hessian(expr, vars)`.
- **Gradient nul (point critique)** : Si f atteint un extremum local en x^*, alors $\nabla f(x^*) = 0$.
  SymPy : `grad = [diff(f,v) for v in vars]`, puis `solve(grad, vars)`.

Théorèmes utiles et application avec SymPy (2/2)

- **Test de la Hessienne (2D)** : $\det(H) > 0, f_{xx} > 0 \Rightarrow$ min local. $\det(H) > 0, f_{xx} < 0 \Rightarrow$ max local. $\det(H) < 0 \Rightarrow$ point selle.
  SymPy : calcul direct via dérivées secondes.
- **Inégalité de Jensen** : $f(\lambda x + (1-\lambda)y) \leq \lambda f(x) + (1-\lambda)f(y)$.
  SymPy : substitution symbolique + évaluation numérique.
- **Descente de gradient (corollaire)** : Si f convexe et gradient Lipschitz, convergence vers min global.
  SymPy : calcul du gradient + `lambdify()` pour optimiser numériquement.
- **Transformation symbolique → numérique** : Toute expression → fonction équivalente.
  SymPy : `lambdify(expr, vars)`.

Exemples de base avec SymPy

- `from sympy import *` → importer la bibliothèque
- `x, y = symbols('x y')` → définir des symboles
- `expr = x**2 + 2*x + 1` → créer une expression
- `diff(expr, x)` → dérivée par rapport à x
- `solve(expr, x)` → résoudre l'équation $expr = 0$
- `integrate(expr, (x, 0, 1))` → intégrale de 0 à 1
- `expr.evalf()` → évaluation numérique
- `f = lambdify(x, expr, 'numpy')` → conversion en fonction

Exemples de base avec SymPy

```
import sympy as sp

x = sp.Symbol('x')
expr = x**2 - 2*x + 1

sp.expand(expr)      # Développe
sp.factor(expr)      # Factorise
sp.simplify(expr)    # Simplifie

f = sp.sin(x)*sp.exp(x)
sp.diff(f, x)          # Dérivée
sp.integrate(f, x)     # Intégrale
```

Exemple de base avec Sympy

```
# Résolution d'équations
eq = sp.Eq(x**2 - 2, 0)
solutions = sp.solve(eq, x)
```

Résultat : [-sqrt(2), sqrt(2)]

```
# Algèbre linéaire
A = sp.Matrix([[1, 2], [3, 4]])
A.det()          # Déterminant
A.eigenvals()    # Valeurs propres
```

Exemple illustratif avec SymPy

Exemple : Soit $f(x,y) = x^2 + y^2$.

- **Gradient :** $\nabla f(x,y) = (2x,\, 2y)$
  SymPy : `grad = [diff(f,v) for v in (x,y)]`
- **Hessienne :** $H_f(x,y) = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}$ *SymPy* : `hessian(f, (x,y))`
- **Gradient nul (point critique)** :
  $\nabla f(x,y) = (2x, 2y) = 0 \quad \Rightarrow \quad (x^*, y^*) = (0,0)$.
  SymPy : `solve([diff(f,x), diff(f,y)], (x,y))`.
  $\nabla f(x,y) = 0 \ \Rightarrow \ (0,0)$ est un **minimum global**.
- **Théorème de dérivation** : $\frac{\partial f}{\partial x} = 2x, \quad \frac{\partial f}{\partial y} = 2y$.
  SymPy : `diff(f, x)`, `diff(f, y)`.

Exemple illustratif avec Sympy

- **Convexité** : $\frac{\partial^2 f}{\partial x^2} = 2 \geq 0, \quad \frac{\partial^2 f}{\partial y^2} = 2 \geq 0$. Donc f est convexe.
  $H_f \succ 0 \implies f$ est strictement convexe.
- **Inégalité de Jensen** : Pour $x = 1$, $y = 3$, $\lambda = 0.5$:
  $f(2) = 4 \leq 0.5f(1) + 0.5f(3) = 5$.
  SymPy : substitution numérique.
- **Descente de gradient** : Mise à jour : $x_{k+1} = x_k - \eta \nabla f(x_k)$.
  SymPy : calcul de ∇f, puis conversion avec `lambdify()` pour exécution numérique.
- **Transformation symbolique → numérique** : $f(x, y) \to$ fonction Python/NumPy avec `lambdify()`.

Conclusion

- Le **calcul symbolique** est un outil essentiel qui relie rigueur mathématique et implémentation pratique, permettant de manipuler des expressions de façon exacte plutôt que numérique.
- Les **théorèmes clés** (linéarité, règles de dérivation/intégration, propriétés algébriques) servent de fondations pour automatiser ces manipulations.
- Des bibliothèques comme **Sympy** rendent ces opérations accessibles : simplification, résolution d'équations, dérivées, intégrales et algèbre linéaire.
- Perspectives : comparaison avec auto-diff (TensorFlow/PyTorch), optimisation à grande échelle.

En résumé : le calcul symbolique est un pont entre la théorie mathématique et son application informatique.