Arts et Métiers — Sciences et Technologies

Calcul formel et calcul numérique

UEF MINI – Outils mathématiques et informatiques pour l'ingénieur

Jean-Luc CHARLES
Éric DUCASSE
Thomas MILCENT

Qu’est-ce que le calcul formel ?

Éléments de réponse :

Calcul formel = Calcul littéral = Calcul symbolique

1. **Calcul exact / calcul numérique**
2. **Le calcul symbolique, ou tout ce que ne fait pas le calcul numérique**
3. **Réécriture d'expressions mathématiques par application de règles de substitution**
4. **Évaluation immédiate / évaluation différée**
5. **Équations : définition et résolution formelle**

Calcul exact / calcul numérique

- Calcul exact : entiers, symboles représentants des objets (nombres, fonctions, etc.)
- Calcul numérique : flottants (erreurs d'arrondis)

`import sympy as sb` `import numpy as np`	**Calcul formel** Représentation exacte	**Calcul numérique** Flottant (norme IEEE 754)
Entiers & fractions	`sb.S(2)/5` → 2/5	`2/5` → 0.4
Les nombres irrationnels • Le nombre π	`sb.pi`	`np.pi` → `3.14159265358979`
	`sb.sin(sb.pi)` → 0	`np.sin(np.pi)` → `1.224[···]2e-16`
• Le nombre de Neper $\mathbb{e}$	`sb.E`	`np.e` → `2.71828182845905`
	`sb.exp(sb.I*sb.pi)` → -1	`np.exp(1j*np.pi)` → `(-1+1.224[···]2e-16j)`
• $\sqrt{2}$	`sb.sqrt(2)`	`np.sqrt(2)` → `1.4142135623731`
	`sb.sqrt(2)**2-2` → 0	`np.sqrt(2)**2-2` → `4.440[···]6e-16`

[*Voir aussi le fichier* `UEF-MINI_CM_formel_vs_numerique.ipynb` *fourni.*]

Liens :
- `fr.wikipedia.org/wiki/Calcul_formel`
- `fr.wikipedia.org/wiki/IEEE_754`

Le calcul symbolique,

ou tout ce que ne fait pas le calcul numérique

Type de calcul symbolique	*Exemples*	
	Instruction	*Résultat*
Manipulation d'expressions mathématiques, algébriques[*] ou non		
Développer	`a,b = sb.symbols("a,b")` `((a+b)**2).expand()`	$a^2 + 2\,a\,b + b^2$
Factoriser	`(a**2+2*a-3).factor()`	$(a-1)(a+3)$
Simplifier	`(1-sb.cos(a)**2).simplify()`	$\sin(a)^2$
	`n = sb.symbols("n", integer=True)` `[sb.sin(sb.pi*n),sb.cos(sb.pi*n)]`	$\left[0, (-1)^n\right]$
	`A = sb.sqrt(a**2)` `[A,sb.refine(A,sb.Q.positive(a))]`	$\left[\sqrt{a^2}, a\right]$
Décomposer	`x = sb.symbols("x")` `((2*x-7)/((x-1)*(x**2+4))).apart()`	$\dfrac{x+3}{x^2+4} - \dfrac{1}{x-1}$

[*] « *Une expression algébrique est une expression contenant des nombres, des lettres représentant des nombres inconnus, des parenthèses et des symboles opératoires.* » `[mathplace.fr]`.

[*Pour la déclaration de symboles, voir aussi le fichier* `UEF-MINI_CM_declaration_symboles.ipynb` *fourni.*]

Le calcul symbolique,

ou tout ce que ne fait pas le calcul numérique

Dérivation, intégration, sommation d'expressions

	Code	Résultat

Dériver	`m,s = sb.symbols("m,sigma", real=True)`
	`G = sb.exp(-((x-m)/s)**2/2)`
	`G.diff(x).simplify()`
		$\frac{m-x}{\sigma^2} e^{-\frac{(m-x)^2}{2\sigma^2}}$

	`sb.exp(-x**2/2).diff(x,3)`	$x\left(-x^2+3\right) e^{-\frac{x^2}{2}}$

Intégrer	`s = sb.symbols("sigma", positive=True)`
	`H = sb.exp(-((x-m)/s)**2/2)`
	`sb.integrate(H,(x,-sb.oo,sb.oo))`
	(ou `H.integrate( (x,-sb.oo,sb.oo) )`)
		$\sqrt{2}\sqrt{\pi}\,\sigma$

	`t = sb.symbols("t", real=True)` `sb.integrate(sb.exp(-t**2),(t,0,x))`	$\frac{\sqrt{\pi}}{2}\operatorname{erf}(x)$

Sommer	`k,n = sb.symbols("k,n", integer=True, positive=True)`
	`p = sb.symbols("p", positive=True)`
	`sb.summation(p**k,(k,0,n))`
		$\begin{cases} n+1 & p=1 \\ \frac{1-p^{n+1}}{1-p} & \text{otherwise} \end{cases}$

Le calcul symbolique,

ou tout ce que ne fait pas le calcul numérique

Fonctions indéfinies

Code	Résultat
`f = sb.Function("f")` `f(x).diff(x)`	$\frac{\mathrm{d}}{\mathrm{d}x} f(x)$
`f(a*x+t).diff(x)`	$a \frac{\mathrm{d}}{\mathrm{d}\xi_1} f(\xi_1)\Big\vert_{\xi_1 = a\,x + t}$
`f(sb.sin(x)).diff(x)`	$\cos(x) \frac{\mathrm{d}}{\mathrm{d}\xi_1} f(\xi_1)\Big\vert_{\xi_1 = \sin(x)}$

Développements limités

Code	Résultat
`sb.exp(x).series(x,0,4)`	$1 + x + \frac{x^2}{2} + \frac{x^3}{6} + \mathcal{O}(x^4)$
`(1/sb.sin(3*x)).series(x,0,5)`	$\frac{1}{3\,x} + \frac{x}{2} + \frac{21\,x^3}{40} + \mathcal{O}(x^5)$
`a,h = sb.symbols("a,h", real=True)` `f(a+h).series(h,0,3).doit()`	$f(a) + h \frac{\mathrm{d}}{\mathrm{d}a} f(a) + \frac{h^2}{2} \frac{\mathrm{d}^2}{\mathrm{d}a^2} f(a) + \mathcal{O}(h^3)$

Le calcul symbolique,

ou tout ce que ne fait pas le calcul numérique

Matrices et vecteurs contenant des symboles

	Code	Résultat
Déterminant	`P = sb.Matrix([[1,1,1], [-2,-1,0], [0,1,a]])` `(P, P.det())`	$\left(\begin{bmatrix} 1 & 1 & 1 \\ -2 & -1 & 0 \\ 0 & 1 & a \end{bmatrix}, a-2\right)$
Inverse	`(a-2) * P.inv()`	$\begin{bmatrix} -a & -a+1 & 1 \\ 2\,a & a & -2 \\ -2 & -1 & 1 \end{bmatrix}$
Produits	`D = sb.diag(2,1,a)` `M=((a-2)*P@D@P.inv()).expand()` `M.simplify() ; M`	$\begin{bmatrix} -2\,a & 2-2\,a & a \\ 2\,a & 3\,a-4 & -2 \\ 2a(1-a) & a(1-a) & a^2-2 \end{bmatrix}$
	`u = (1,-1,x) ; P.dot(u)`	$[x, -1, a\,x-1]$ (vecteur)
	`U = sb.Matrix(u) ; P@U`	*idem* (matrice-colonne)
	`V = P.row(2) ; V.dot(u)`	$a\,x-1$
Valeurs propres	`A = sb.Matrix([[a,2,0], [1,a,1], [0,2,a]])` `A.eigenvals()`	$\{a:1,\ a-2:1,\ a+2:1\}$

Réécriture d'expressions mathématiques par application de règles de substitution

En calcul formel, on peut avoir besoin de remplacer dans une expression mathématique un symbole par une expression symbolique ou à une valeur numérique, sans pour autant avoir recours à une affectation. [*Voir le fichier* `UEF-MINI_CM_reecriture_application_numerique.ipynb` *fourni.*]

- **Programmation standard**

```
x,y = sb.symbols("x,y")
a = x*(y-1)
b = (a+x).expand()
```

`[a,b]` donne : $[x\,(y-1)), x\,y]$

À la fin, `b` ne désigne plus la somme $a+x$ mais son calcul en remplaçant a par $x\,(y-1)$.

- **Utilisation d'une règle de substitution**

```
a,x,y = sb.symbols("a,x,y")
regle = {a:x*(y-1)}
b = a+x
```

`b.xreplace(regle).expand()` donne : $x\,y$

`[a,b]` donne : $[a, a+x]$

À la fin, `a` reste le symbole a et `b` reste tel qu'il a été défini symboliquement : la somme $a+x$.

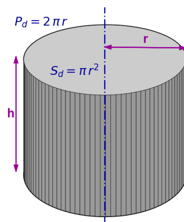
Réécriture d'expressions mathématiques par application de règles de substitution

En calcul formel, on peut avoir besoin de remplacer dans une expression mathématique un symbole par une expression symbolique ou à une valeur numérique, sans pour autant avoir recours à une affectation. [*Voir le fichier* **UEF-MINI_CM_reecriture_application_numerique.ipynb** *fourni.*]

- **Exemple d'application**
  1. **Calcul littéral**

```
r,h = sb.symbols("r,h", real=True, positive=True)
V = sb.pi*r**2 * h
S = ( 2 * sb.pi*r**2 + 2*sb.pi*r * h ).factor()
```

(V,S) donne : $\left(\pi h r^2, 2\pi r (h+r)\right)$



  2. **Application(s) numérique(s)**

```
AN1,AN2 = {r:2e-3, h:3e-2}, {r:2.5e-3, h:2e-2}
for regle in (AN1, AN2) :
    print([float(e.xreplace(regle)) for e in (r,h,V,S)])
```

	Rayon r	Hauteur h	Volume V	Surface S
Cas n°1	2.0 mm	30.0 mm	$\approx 377\ \text{mm}^3$	$\approx 402\ \text{mm}^2$
Cas n°2	2.5 mm	20.0 mm	$\approx 393\ \text{mm}^3$	$\approx 353\ \text{mm}^2$

Évaluation immédiate et évaluation différée

Sur l'exemple :

```
x,u,v = sb.symbols("x,u,v")
n = sb.symbols("n", integer=True, positive=True)
somme = sb.summation(v**n/sb.factorial(n),(n,0,sb.oo))
```

→ **somme** désigne $\exp(v)$, obtenu par le calcul de $\sum_{n=0}^{\infty} \frac{v^n}{n!}$.

On veut obtenir ce résultat pour une **liste** d'objets **sympy**.

- **Évaluation différée**, ou ce qui se passe en programmation standard

  Boucle, avec calcul à chaque itération :

  ```
  L1 = [ sb.summation(u**n/sb.factorial(n), (n,0,sb.oo)) for u in liste ]
  ```

  Idem avec la définition d'une fonction :

  ```
  def s_differee(t) :
      return sb.summation(t**n/sb.factorial(n), (n,0,sb.oo))
  L2 = [ s_differee(u) for u in liste ]
  ```

- **Évaluation immédiate**, faite une fois pour toutes

  ```
  L3 = [ somme.replace(v,u) for u in liste ]
  ```

  ```
  s_immediate = sb.lambdify(x,sb.summation(x**n/sb.factorial(n),
                                            (n,0,sb.oo)), "sympy")
  L4 = [ s_immediate(u) for u in liste ]
  ```

  [*Voir le fichier* `UEF-MINI_CM_evaluations_immediate_et_differee.ipynb` *fourni.*]

Équations : définition et résolution formelle

Dans la plupart des langages récents : **=** désigne l'affectation, **==** désigne l'égalité.

- En *Python* :

`a = b`	On donne le nom **a** à l'objet de nom **b**
`a == b`	→ booléen indiquant si les valeurs de **a** et **b** sont égales
`a is b`	→ booléen indiquant si **a** et **b** désignent le même objet

- Avec **sympy** :

`a == b` **[à éviter !]**	→ booléen indiquant si les valeurs de **a** et **b** sont identiques
`a.equals(b)`	→ booléen indiquant si les expressions littérales **a** et **b** sont égales
`sb.Eq(a,b)`	→ **Équation** (objet non évalué *a priori*)
Exemple :	`x,y = sb.symbols("x,y") ; sb.Eq(x,y)` → $x = y$; `sb.Eq(x,x)` → `True` ; `sb.Eq(x,x+1)` → `False`.

Exemple d'équation algébrique

```
X,p,q = sb.symbols("X,p,q", real=True)
eq_alg = sb.Eq( X**2 + 2*p*X + q , 0 )
```

`sb.solve(eq_alg,X)` donne $\left[-p - \sqrt{p^2 - q}, -p + \sqrt{p^2 - q}\right]$

`sb.solve(eq_alg,p)` donne $\left[-\left(X^2 + q\right)/(2X)\right]$

`sb.solve(eq_alg,q)` donne $[-X\,(X + 2\,p)]$

[*Voir le fichier* `UEF-MINI_CM_equations_solveurs.ipynb` *fourni.*]

- **Remarque** De nombreuses équations n'admettent *pas de solutions exactes* et on utilise alors des *méthodes numériques de résolution*

Exemples de problèmes différentiels

- **Équation différentielle ordinaire avec conditions initiales**

  $f''(t) + 2\,a\,f'(t) + 3\,f(t) = 0,$ avec $f(0) = 1$ et $f'(0) = 0$

```
t,a = sb.symbols("t,a", real=True) ; f = sb.Function("f")
edo = sb.Eq(f(t).diff(t,2)+2*a*f(t).diff(t)+3*f(t),0)
sol_edo = sb.dsolve(edo, f(t))
```

donne $f(t) = C_1\,e^{t\left(-a-\sqrt{a^2-3}\right)} + C_2\,e^{t\left(-a+\sqrt{a^2-3}\right)}$

```
solu_gene = sol_edo.rhs # right-hand side
CI = [ sb.Eq( solu_gene.xreplace({t:0}), 1), \
       sb.Eq( solu_gene.diff(t).xreplace({t:0}), 0) ]
constantes = sb.solve(CI, "C1,C2")
```

donne $\left\{C_1\colon -\frac{a}{2\sqrt{a^2-3}} + \frac{1}{2},\ C_2\colon \frac{a}{2\sqrt{a^2-3}} + \frac{1}{2}\right\}$

- **Système différentiel ordinaire avec conditions aux limites**

$$\begin{cases} T'(x) = -\phi(x)/k(x) \\ \phi'(x) = (2\,x-1)^2 \end{cases} \quad \text{avec} \quad \begin{cases} \phi(0) = h\,(T_\text{ext} - T(0)) \\ \phi(1) = h\,(T(1) - T_\text{ext}) \end{cases} \quad \text{et} \quad k(x) = 1+x$$

Solution : $T(x) = T_\text{ext} - \frac{A}{h} + \left(\frac{1300}{3} - A\right)\log(x+1) - \frac{100\,x}{9}\left(4\,x^2 - 15\,x + 39\right)$

[*Pour le détail, voir le fichier* **UEF-MINI_CM_equations_solveurs.ipynb** *fourni.*]

- **Remarque** De nombreuses équations différentielles n'admettent *pas de solutions exactes* et on utilise alors des *méthodes numériques de résolution* (méthodes pas-à-pas, différences finies, etc.)