

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 représentant des objets (nombres, fonctions, etc.)
- Calcul numérique : flottants (erreurs d'arrondis)

<pre>1 import sympy as sb 2 import numpy as np</pre>	Calcul formel	Calcul numérique
	Représentation exacte	Flottant (norme IEEE 754)
Entiers & fractions	<code>sb.S(2)/5</code> → 2/5	2/5 → 0.4
Les nombres irrationnels		
<ul style="list-style-type: none"> ● Le nombre π ● Le nombre de Neper e ● $\sqrt{2}$ 	<pre>sb.pi sb.sin(sb.pi) → 0 sb.E sb.exp(sb.I*sb.pi) → -1 sb.sqrt(2) sb.sqrt(2)**2-2 → 0</pre>	<pre>np.pi → 3.14159265358979 np.sin(np.pi) → 1.224[...]²e-16 np.e → 2.71828182845905 np.exp(1j*np.pi) → (-1+1.224[...]²e-16j) np.sqrt(2) → 1.4142135623731 np.sqrt(2)**2-2 → 4.440[...]⁶e-16</pre>

[Voir aussi le fichier [UEF-MINI_CM_formel_vs_numerique.ipynb](#) fourni.]

Liens : [● fr.wikipedia.org/wiki/Calcul_formel](https://fr.wikipedia.org/wiki/Calcul_formel)
[● fr.wikipedia.org/wiki/IEEE_754](https://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	<code>a,b = sb.symbols("a,b") ((a+b)**2).expand()</code>	$a^2 + 2ab + b^2$
Factoriser	<code>(a**2+2*a-3).factor()</code>	$(a-1)(a+3)$
Simplifier	<code>(1-sb.cos(a)**2).simplify()</code>	$\sin(a)^2$
	<code>n = sb.symbols("n", integer=True) [sb.sin(sb.pi*n), sb.cos(sb.pi*n)]</code>	$[0, (-1)^n]$
	<code>A = sb.sqrt(a**2) [A, sb.refine(A, sb.Q.positive(a))]</code>	$\left[\sqrt{a^2}, a \right]$
Décomposer	<code>x = sb.symbols("x") ((2*x-7) / ((x-1)*(x**2+4))).apart()</code>	$\frac{x+3}{x^2+4} - \frac{1}{x-1}$

[*] « 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

Dériver	<pre>m, s = sb.symbols("m, sigma", real=True) G = sb.exp(-(x-m)/s)**2/2) G.diff(x).simplify()</pre>	$\frac{m-x}{\sigma^2} e^{-\frac{(m-x)^2}{2\sigma^2}}$
	<pre>sb.exp(-x**2/2).diff(x, 3)</pre>	$x(-x^2+3)e^{-\frac{x^2}{2}}$
Intégrer	<pre>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)))</pre>	$\sqrt{2}\sqrt{\pi}\sigma$
	<pre>t = sb.symbols("t", real=True) sb.integrate(sb.exp(-t**2), (t, 0, x))</pre>	$\frac{\sqrt{\pi}}{2} \operatorname{erf}(x)$
Sommer	<pre>k, n = sb.symbols("k, n", integer=True, positive=True) p = sb.symbols("p", positive=True) sb.summation(p**k, (k, 0, n))</pre>	$\begin{cases} n+1 & p=1 \\ \frac{1-p^{n+1}}{1-p} & \text{otherwise} \end{cases}$

Le calcul symbolique, *ou tout ce que ne fait pas le calcul numérique*

Fonctions indéfinies

<code>f = sb.Function("f")</code> <code>f(x).diff(x)</code>	$\frac{d}{dx} f(x)$
<code>f(a*x+t).diff(x)</code>	$a \frac{d}{d\xi_1} f(\xi_1) \Big _{\xi_1=ax+t}$
<code>f(sin(x)).diff(x)</code>	$\cos(x) \frac{d}{d\xi_1} f(\xi_1) \Big _{\xi_1=\sin(x)}$

Développements limités

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

Le calcul symbolique, ou tout ce que ne fait pas le calcul numérique

Matrices et vecteurs contenant des symboles

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

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 $a + x$.

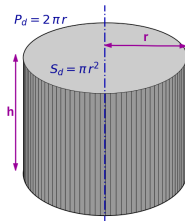
Réécriture d'expressions mathématiques par application de règles de substitution

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 : (π h r2, 2 π r (h + r))
```



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* :

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

- Avec *sympy* :

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

Exemple d'équation algébrique

```
X, p, q = sb.symbols("X, p, q", real=True)
P = X**2 + 2*p*X + q
sb.solve(P, X)  donne  $\left[-p - \sqrt{p^2 - q}, -p + \sqrt{p^2 - q}\right]$ 
sb.solve(P, p)  donne  $\left[-\left(X^2 + q\right) / (2X)\right]$ 
sb.solve(P, q)  donne  $[-X(X + 2p)]$ 
```

[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**

$$f''(t) + 2af'(t) + 3f(t) = 0, \quad \text{avec } f(0) = 1 \text{ et } f'(0) = 0$$

```
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)   donne f(t) = C1 * e^{t(-a - sqrt(a^2 - 3))} + C2 * e^{t(-a + sqrt(a^2 - 3))}
solu_gene = sol_edo.rhs # right-hand side
CI = [solu_gene.xreplace({t:0}) - 1, solu_gene.diff(t).xreplace({t:0})]
constantes = sb.solve(CI, ["C1", "C2"])   donne { -a/(2*sqrt(a^2-3)) + 1/2, a/(2*sqrt(a^2-3)) + 1/2 }
```

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

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

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

[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.)