Chapitre 4

Calcul symbolique dans MATLAB

variables en virgule flottante et variables symboliques

Les variables sont en virgule flottante, vous devez indiquer explicitement qu'une variable est destinée à être symbolique. Une façon de le faire est d'utiliser la commande syms, qui indique à MATLAB qu'une ou plusieurs variables sont symboliques. Par exemple, la commande suivante définit a et b comme des variables symboliques :

syms a b

On peux former des expressions symboliques impliquant ces variables :

2*a*b

Remarque 1 *le résultat ci-dessus est symbolique, pas numérique comme ce serait le cas si les variables étaient des variables à virgule flottante. De plus, le calcul ci-dessus n'entraîne pas d'erreur, même si a et b n'ont pas de valeurs. Une autre façon de créer une variable symbolique consiste à attribuer une valeur symbolique à un nouveau symbole. Puisque les nombres sont, par défaut, en virgule flottante, il est nécessaire d'utiliser la fonction sym pour dire à MATLAB qu'un nombre est symbolique*

On peux faire des calculs symboliques :

a=sqrt(c)

Vous devriez remarquer la différence entre le résultat ci-dessus et ce qui suit :

a=sqrt(2)

Même si a été déclaré être une variable symbolique, une fois que je lui assigne une valeur en virgule flottante, il devient numérique. Cet exemple souligne également que sym doit être utilisé avec des constantes littérales si elles doivent être interprétées comme symboliques et non numériques :

a=sqrt(sym(2))

En guise d'exemple supplémentaire, considérez les deux commandes suivantes :

sin(sym(pi))

sin(pi)

Dans le premier cas, puisque π est symbolique, MATLAB remarque que le résultat est exactement nul ; dans la seconde, à la fois π et le résultat sont représentés en virgule flottante, donc le résultat n'est pas exactement zéro (l'erreur est juste arrondie Erreur).

En utilisant des variables symboliques, on peux effectuer des manipulations algébriques.

syms x

p=(x-1)*(x-2)*(x-3) expand(p)

expand(p)

Les entiers peuvent également être factorisés, bien que la forme du résultat dépende du fait que l'entrée est numérique ou symbolique :

factor(144)

factor(sym(144))

Pour une entrée numérique (entière), le résultat est une liste des facteurs, répétés selon la multiplicité.

Pour une entrée symbolique, la sortie est une expression symbolique.
Une commande importante pour travailler avec des expressions symboliques est simplify, qui essaye de réduire une expression à une expression plus simple égale à l'original.
syms x a b c
$p = (x-1)*(a*x^2+b*x+c)+x^2+3*x+a*x^2+b*x$
simplify(p)
Puisque le concept de simplification n'est pas défini avec précision (ce qui est plus simple, un polynôme sous forme factorisée ou sous forme développée $a+bx+cx2+\cdots$), MATLAB dispose d'un certain nombre de commandes de simplification spécialisées. On a déjà utilisé deux d'entre eux, **factoriser** et **développer** . Un autre est **collect** , qui "rassemble les termes similaires" :
collect(p,x)
D'ailleurs, l'affichage de la sortie symbolique peut être rendu plus mathématique en utilisant la commande pretty :
Pretty(ans)

Manipuler des fonctions dans MATLAB

Les expressions symboliques peuvent être traitées comme des fonctions. Voici un exemple :
syms x
$p = x/(1+x^2)$
En utilisant la commande subs, on peux évaluer la fonction p pour une valeur donnée de x.
Le calcul est automatiquement vectorisé :
» x=sym(linspace(-5,5,101)) ;
» z=subs(p,x) ;
» plot(x,z)
L'une des caractéristiques les plus puissantes du calcul symbolique dans MATLAB est que certaines opérations de calcul, notamment l'intégration et la différenciation, peuvent être effectuées symboliquement. Ces capacités seront expliquées plus tard.
Ci-dessus, on a vu comment évaluer une fonction définie par une expression symbolique. Il est également possible de définir explicitement une fonction sur la ligne de commande. Voici un exemple :
f=@(x)x^2
La fonction f peut être évaluée de la manière attendue, et l'entrée peut être flottante ou symbolique :

```
>> f(1)
>> f(sqrt(pi))
>> f(sqrt(sym(pi)))
```

L'opérateur @ peut également être utilisé pour créer une fonction de plusieurs variables :

```
>> g=@(x,y)x^2*y
>> g(1,2)
```

Une fonction définie par l'opérateur @ n'est pas automatiquement vectorisée :

```
>> x=linspace(0,1,11)';
>> y=linspace(0,1,11)';
>> g(x,y)
```

La fonction peut être vectorisée lorsqu'elle est définie :

```
>> g=@(x,y)x.^2.*y
>> g(x,y)
```

Utilisation du calcul symbolique dans des équations différentielles

```
>> syms x
>> f=sin(x^2)
>> diff(f,x)
>> syms x y
>> q=x^2*y^3*exp(x)
>> pretty(q)
>> diff(q,y)
>> syms a t
>> u=exp(a*t)
>> diff(u,t)-a*u
>> v=a*t
```

Il est facile de vérifier si une fonction de plusieurs variables est la solution d'une EDP. Par exemple, $w(x,y) = sin(\pi x) + sin(\pi y)$ une solution de l'équation différentielle

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial x^2} = 0?$$

```
>> syms x y
>> w=sin(pi*x)+sin(pi*y)
>> diff(w,x,2)+diff(w,y,2)
>> simplify(ans)
```

On peut calculer des dérivées supérieures d'une expression. Par exemple, diff(w,x,5)
dérivée mixte

```
>> syms x y
>> w=x^2*exp(y)+x*y^2
>> diff(diff(w,x),y)
```

MATLAB peut calculer des intégrales indéfinies et définies. La commande de calcul d'une intégrale indéfinie (antidérivative) est exactement analogue à celle du calcul d'une dérivée :

```
>> syms x
>> f=x^2
>> f=x^2
>> int(f,x)
```

Remarque 2 *MATLAB n'ajoute pas la "constante d'intégration". Il retourne simplement un anti-dérivé (si possible). Si l'intégrande est trop compliquée, MATLAB retourne juste l'intégrale non évaluée, et imprime un message d'avertissement*

```
>> int(exp(cos(x)),x)
```

Pour calculer une intégrale définie, nous devons spécifier les limites de l'intégration :

```
>> int(x^2,x,0,1)
```

MATLAB possède des commandes pour estimer la valeur d'une intégrale définie qui ne peut pas être calculée analytiquement. Soit :

```
>> int(exp(cos(x)),x)
```

Étant donné que Matlab ne pouvait pas trouver un antidérivé explicite, je peux utiliser la fonction quad pour estimer l'intégrale définie. La commande quad prend, en entrée, une fonction plutôt qu'une expression (comme le fait int). Par conséquent, je dois d'abord créer une fonction :

```
>> f = @(x)exp(cos(x))
>> quad(f,0,1)
```

Exercice 1 *Vérifier que* $\frac{d}{dx}\int_a^b f(x,y)dy = \int_a^b \frac{d}{dx}f(x,y)dy$ *pour* $f = xy^3 + x^2y$

Résolution de problèmes aux limites simples par l'intégration

Exemple 1 *Soit le BVP suivant :*

$$\begin{cases} -\frac{\partial^2 u}{\partial x^2} = x+1 & 0 < x < 1 \\ u(0) = 2, & u(1) = 0 \end{cases}$$

MATLAB n'ajoute pas de constante d'intégration, donc je le fais explicitement :

```
clear
syms x C1 C2
r1=int(-(1+x),x)+C1
r2=int(r1,x)+C2
```

La commande MATLAB solve

Avant de terminer l'exemple précédent, j'expliquerai la commande solve sur des exemples plus simples. Supposons que je souhaite résoudre l'équation linéaire $ax + b = 0$ pour x. Je peux considérer cela comme un problème de recherche de racine : je veux la racine de la fonction $f(x) = ax + b$. La commande solve trouve la racine d'une fonction par rapport à une variable indépendante donnée :

```
>> syms f x a b
>> f=a*x+b
solve(f,x)
```

```
f=x^2-3*x+2;
solve(f,x)
```

solve peut aussi être utilisé pour résoudre un système d'équations en plusieurs variables. Dans ce cas, les équations sont listées en premier, suivies des inconnues. Par exemple

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

```
>> syms x y
>> s=solve(x+y-1,2*x-y-1,x,y)
>> s.x
>> s.y
```

```
>> syms x y
>> s=solve(x^2+y^2-1,y-x^2,x,y)
>> pretty(s.x(1))
>> pretty(s.y(1))
>> pretty(s.x(2))
>> pretty(s.y(2))
```

Vous remarquerez peut-être que la deuxième solution est complexe (au moins, la valeur de x est complexe). Cela pourrait être plus facile à voir à partir de la valeur numérique des expressions, que nous pouvons voir avec la commande double (qui convertit une expression symbolique en une valeur à virgule flottante, si possible) double(s.x(2)), double(s.y(2)), double(s.x(3)), double(s.y(3)), double(s.x(4))) double(s.y(4)).
Enfin, il existe une autre façon d'appeler 'solve'. »L'équation et les inconnues peuvent être données sous forme de chaînes, ce qui signifie qu'il n'est pas nécessaire de définir des variables symboliques avant d'appeler solve :

```
>> solve('x^2-2*x-1=0','x')
```

Retour à l'exemple

```
>> clear
>> syms x C1 C2
>> r1=int(-(1+x),x)+C1
>> r2=int(r1,x)+C2
>> s=solve(subs(r2,x,0)-2,subs(r2,x,1),C1,C2)
>> s.C1
>> s.C2
>> u=subs(r2,{C1,C2},{s.C1,s.C2})
```

Exemple 2 *Considérons le BVP à coefficients non constant : :*

$$\begin{cases} -\frac{d}{dx}\left[\left(1+\frac{x}{2}\right)\frac{du}{dx}\right]=0 & 0<x<1 \\ u(0)=20, & u(1)=25 \end{cases}$$

nous intégrons une fois

$$\frac{du}{dx}=\frac{C1}{1+x/2}$$

(Il est facile d'effectuer ce calcul manuellement que de demander à MATLAB d'intégrer) on intègre une deuxième fois :

```
>> clear
>> syms C1 C2 x
>> u=int(C1/(1+x/2),x)+C2
```

On utilise la commande d'appeler 'solve' pour trouver C1 et C2 :

```
>> s=solve(subs(u,x,0)-20,subs(u,x,1)-25,C1,C2)
>> u=subs(u,{C1,C2},{s.C1,s.C2})
```

Remarque 3 *Remarquez que la réponse est inattendue ; le problème est que MATLAB n'a pas interprété les constantes des équations (0, 1, 20, 25) comme des quantités symboliques, mais plutôt comme des valeurs numériques (virgule flottante). En conséquence. La solution peut être trouvée en précisant que les différents nombres doivent être traités comme symboliques, en écrivant :*

```
>> s=solve(subs(u,x,sym(0))-sym(20),subs(u,x,sym(1))-sym(25),C1,C2)
>> u=subs(u,{C1,C2},{s.C1,s.C2})
```

Scripts et fonctions

Il est possible de créer et d'enregistrer une séquence d'instructions dans un fichier (appelé un M-file) et de les faire exécuter par MATLAB. On distingue 2 types de M-file, les fichiers de scripts et les fichiers de fonctions. Un script est un ensemble d'instructions MATLAB, que l'on éxécute dans la fenêtre MATLAB en tapant son nom. Les fichiers de fonctions permettent à l'utilisateur de définir ces propres fonctions qui ne figurent pas parmi les fonctions incorporées de MATLAB et de les utiliser de la même manière que ces dernières (ces fonctions sont nommées fonctions utilisateur). On définit la fonction de la manière suivante :

```
function [vars1, ..., vars_m] ← nom_fonc(vare_1, ..., vare_n)
suite d'instructions
```

où

- $vars_1, ..., vars_m$ sont les variables de sortie de la fonction et peuvent être de n'importe quel type
- $vare_1, ..., vare_n$ sont les variables d'entrée de la fonction et peuvent être de n'importe quel type
- séquence d'instructions est le corps de la fonction

Remarque 4

— Un nom de fonction, devra aussi être le nom du fichier dans lequel sera écrit la fonction
— Les paramètres d'entrée de la fonction, sont les valeurs des éléments dont aura besoin la fonction pour réaliser son calcul. Il est important de comprendre que la fonction ne peut pas lire les variables de la zone des variable de Matlab : c-à-d une fonction ne connaît que ses variables internes, celles déclarées en paramètres d'entrée ou déclarées dans le code même de la fonction. s'il n'y a pas de paramètre d'entrée, on peut omettre les parenthèses.
— Les paramètres en sortie de la fonction, sont les valeurs calculées par la fonction, et que l'on souhaite récupérer une fois la fonction terminée. Une fois la fonction terminée, toutes ses variables internes seront détruites. S'il n'y a qu'une valeur de sortie, les crochets peuvent être omis.
— Si on veut à ce que certaines variables de la fonction soient visibles et accessibles à l'extérieur de celle-ci, on peut les rendre "globales" en les définissant comme telles dans la fonction, avant qu'elles ne soient utilisées, par une déclaration global variable(s). Il faudra aussi faire une telle déclaration dans le workspace (et avant d'utiliser la fonction !) si l'on veut pouvoir accéder à ces variables dans workspace principal ou ultérieurement dans d'autres scripts !

Ecriture des données formatées : fprintf

Exemple 3

```
x = 1:0.1:100;
y = [x; log(x)];
fid = fopen('log.txt', 'wt');
fprintf(fid, '%6.2f %12.8f\n', y);
fclose(fid)
```