2. Nombres complexes

Maple permet aussi de faire des calculs dans le corps des complexes.

Le *i* complexe est représenté par `I` dans Maple

On peut aisément le vérifier :

```
>I^2;
```

$$-1$$

À partir de là, toutes les opérations vues avec les nombres réels peuvent être réalisées avec des nombres complexes.

```
>(4+3*I)+(7-2*I);
```

$$11 + I$$

```
>(1+I)*(2-3*I);
```

$$5 - I$$

Toutes les opérations classiques sur les complexes sont disponibles.

On peut isoler partie réelle et partie imaginaire à l'aide des fonctions `Re` et `Im` :

```
>Re(nombre complexe);
>Im(nombre complexe);
```

```
>Re(1+2*I);
```

$$1$$

```
>Im(1+2*I);
```

$$2$$

On peut prendre le conjugué :

```
>conjugate(4-3*I);
```

$$4 + 3\,I$$

On peut également entrer le nombre complexe sous sa forme polaire :

```
>polar(1,Pi/6);
```

$$\mathrm{polar}\left(1, \frac{1}{6}\pi\right)$$

On peut alors passer à la forme cartésienne de ce nombre :

```
>evalc(");
```

$$\frac{1}{2}\sqrt{3} + \frac{1}{2}\,I$$

On peut vérifier qu'il s'agit bien du nombre que l'on a entré. Tout d'abord déterminons son module :

```
>abs(");
```

$$1$$

Le module de ce nombre complexe est donc 1.

De même on vérifie que l'argument de ce nombre complexe est $\frac{\pi}{6}$:

```
>argument("");
```

$$\frac{1}{6}\pi$$

Profitons de cette allusion aux angles pour signaler que Maple ne connaît que les radians, seule unité mathématique d'angles. Il n'existe pas ici comme sur les calculatrices de « mode » degré et de « mode » radian. Au mieux, on peut convertir les degrés en radians grâce à la commande :

```
>convert(30*degrees,radians);
```

$$\frac{1}{6}\pi$$

On peut bien entendu mener l'opération inverse :

```
>convert(Pi/6,degrees);
```

$$30\ degrees$$

3. Fonctions numériques

D'abord, il convient de revenir sur la notion de fonction. Commençons par entrer une expression en mémoire (on rappelle que l'affectation[1] se fait à l'aide de **:=**) :

```
>a:=x^2+2*x+1;
```

$$a := x^2 + 2x + 1$$

On peut penser que, a priori, $a(1)$ peut renvoyer $4 (= 1^2 + 2 + 1)$. Mais en fait, la réponse de Maple est différente :

```
>a(1);
```

$$x(1)^2 + 2x(1) + 1$$

Pour définir une fonction en Maple, il faut employer :

```
>fonction:=var->expression(var);
```

On peut noter que cette syntaxe est assez proche de la définition mathématique d'une fonction.

Par exemple :

```
>f:=x->x^2+2*x+1;
```

$$f := x \to x^2 + 2x + 1$$

Là, par contre, on obtient bien le résultat voulu :

```
>f(1);
```

$$4$$

On peut aussi utiliser la fonction **unapply** pour définir une fonction à partir d'une expression :

```
>fonction:=unapply(expression,var);
```

Par exemple, on crée la même fonction f par :

```
>g:=unapply(a,x);
```

$$g := x \to x^2 + 2x + 1$$

Posons-nous les questions classiques : quel est l'ensemble de définition de cette fonction ?

Pour répondre, on va utiliser la fonction **singular** qu'il convient au préalable de charger avec **readlib(singular);**.

```
>singular(f(x));
```

$$\{x = \infty\}, \{x = -\infty\}$$

On peut cependant regretter que la réponse de Maple ne soit pas très explicite ! Maple dit ici que l'ensemble de définition est $]-\infty, +\infty[$. Mais on peut quand même trouver des points où la fonction n'est pas définie avec la fonction **singular** :

```
>singular(1/x);
```

$$\{x = 0\}$$

Ou encore :

```
>singular(tan(x));
```

$$\left\{x = \_N\pi + \frac{1}{2}\pi\right\}$$

On peut aussi tester la parité de f en vérifiant que $f(x) = f(-x)$:

```
>evalb(f(x)=f(-x));
```

false

On peut aussi utiliser une autre solution en testant le type[1] de f :

```
>type(f(x),evenfunc(x));
```

false

(On pourrait de même tester l'imparité de f avec **type(f(x),oddfunc(x));**).

Nous allons tester la continuité de la fonction grâce à la fonction **iscont** qu'il faut au préalable charger à l'aide de **readlib(iscont);**

```
>iscont(f(x),x=-infinity..infinity);
```

true

En revanche :

```
>iscont(tan(x),x=-infinity..infinity);
```

false

Le second argument de la fonction **iscont** est donc un intervalle, qui peut être plus restreint que ceux figurant dans les deux exemples précédents.

On peut alors chercher les points de discontinuité à l'aide de la fonction **discont** :

```
>discont(tan(x),x);
```

$$\{\pi\_Z1 \sim + \frac{1}{2}\pi\}$$

On détermine l'ensemble de définition d'une fonction f avec la fonction **singular** :

```
>singular(f(x));
```

On vérifie si une fonction f est continue sur $[a,b]$ avec la fonction **iscont** :

```
>isont(f(x),x=a..b);
```

Ces deux fonctions doivent au préalable être chargées à l'aide de **readlib(*fonction*);**.

3.1 *Dérivation*

Intéressons-nous désormais à la dérivation de fonctions. On n'emploiera pas la même syntaxe si l'objet que l'on doit dériver est une fonction (de type *function*) ou une expression.

Dans le cas d'une expression, on différencie avec la fonction **diff** :

>diff(*expression*, var_1,…, var_n);

On dérive l'*expression* par rapport aux variables $var_1..var_n$.

Par exemple :

```
>diff(2*x^2+3,x);
```

$$4x$$

Comme pour la fonction **Sum**, le fait d'entrer **Diff** (le nom de la fonction avec la première lettre en majuscule) permet d'obtenir la forme inerte :

```
>Diff(2*x^2+3,x);
```

$$\frac{\partial}{\partial x}(2x^2+3)$$

On peut dériver par rapport à plusieurs variables ou plusieurs fois par rapport à la même variable :

```
>diff(sin(x*y),x,y);
```

$$-\sin(xy)\,xy + \cos(xy)$$

dérive $\sin(xy)$ par rapport à x puis par rapport à y.

```
>diff(2*x^3+3*x^2+4,x,x,x);
```

$$12$$

dérive le polynôme trois fois par rapport à x.

Il existe aussi une autre méthode pour dériver plusieurs fois par rapport à la même variable, à l'aide de l'opérateur **$** :

```
>diff(2*x^3+3*x^2+4,x$3);
```

$$12$$

Pour dériver une fonction f on utilise la fonction **D** :

```
>D(f);
```

Ou pour une fonction à plusieurs variables :

```
>D[1,...,n](f);
```

Cette dernière ligne de commande dérive f par rapport aux variables $var_1,\ldots,var_n$ (dans le même ordre que lors de la définition de la fonction).

Par exemple pour la fonction f que nous allons définir :

```
>f:=x->x^2+2*x+1;
```

$$f := x \to x^2 + 2x + 1$$

```
>D(f)(x);
```

$$2x + 2$$

On peut alors très simplement calculer le nombre dérivé en un point :

```
>D(f)(2);
```

$$6$$

Pour différencier plusieurs fois la même fonction par rapport à une variable :

```
>(D@@2)(f)(x);
```

$$2$$

Intéressons-nous maintenant aux fonctions de plusieurs variables :

```
>g:=(x,y)->2*x^2*y^3+3*y*x+x*sin(y)+y+1;
```

$$g := (x,y) \to 2x^2y^3 + 3yx + x\sin(y) + y + 1$$

```
>D[1](g)(x,y);
```

$$4xy^3 + 3y + \sin(y)$$

Cette commande permet de dériver la fonction f par rapport à x. On peut aussi dériver par rapport à deux variables ou même plusieurs fois par rapport à la même variable. Comme le montrent respectivement les deux exemples suivants :

```
>D[1,2](g)(x,y);
```

$$12xy^2 + 3 + \cos(y)$$

```
>D[1$2,2](g)(x,y);
```

$$12y^2$$

On peut remarquer que, pour l'instant, on a demandé le nombre dérivé au point (x, y) ; le résultat est donc une expression. Par défaut, l'usage de la fonction **D** renvoie une fonction :

```
>D[1,2](g);
```

$$(x,y) \to 12xy^2 + 3 + \cos(y)$$

3.2 *Limites*

Intéressons-nous aux limites de cette fonction. Pour cela, il suffit d'utiliser la fonction **limit**.

On calcule la limite d'une expression en un point donné par :

```
>limit(expression(var),var=point,direction);
```

La direction est un argument optionnel qui peut être nécessaire, comme nous le verrons dans peu de temps. Prenons tout de suite un exemple :

```
>limit(f(x),x=infinity);
```

Parfois il est nécessaire de spécifier si l'on veut une limite à gauche ou à droite ; c'est ce que l'on va faire avec l'argument direction :

```
>limit(1/x,x=0);
```

undefined

Il faut alors entrer une direction : **right** ou **left**.

```
>limit(1/x,x=0,left);
```

$$-\infty$$

On obtient la forme inerte en entrant **Limit** (**limit** avec un L majuscule). Par exemple, on peut avoir :

```
>Limit(ln(x)/x,x=infinity)=limit(ln(x)/x,x=infinity);
```

$$\lim_{x\to\infty}\frac{\ln(x)}{x}=0$$