Fiche Maple n°2

Dans cette fiche nous commençons à explorer quelques fonctionnalités un peu plus élaborées de Maple : tests, boucles et procédures.

1. Tests

Le principe est assez simple et basé sur l'emploi des fonction booléennes et de **if**

Les fonctions booléennes sont standard : **and**, **or**, **not**, **xor** (ou exclusif : prend la valeur **true** si l'un est vrai et l'autre est faux) et **implies**.

```
> (-1<-2) and (-1<0);
```

false

```
> (-1<-2) implies (-1<0);
```

true

evalb renvoie **false** ou **true** suivant la valeur de vérité de l'expression entre parenthèses.

```
> evalb(-1<-2);
```

false

Comme Maple utilise les mots **true** ou **false** comme valeur de vérité , ce qui n'est pas très pratique, c'est l'occasion de fabriquer une petite procédure qui renverra 0 pour **false** et 1 pour **true** (les commentaires ne sont pas à taper bien sûr).

```
> bool:=proc(expression)
```

On donne le nom que l'on veut à la procédure (ici **bool**) suivi de **:=** puis du mot réservé **proc(…)** sans le **;** habituel. Ici la variable passée en paramètre est appelée **expression**, mais on peut mettre autant de variables que l'on veut séparées par des virgules avec le nom que l'on veut. On peut déclarer le type de variable mais ce n'est pas nécessaire en général, Maple s'adaptant au contexte.

```
> local u:integer;
```

Lorsqu'on utilise une variable intermédiaire on utilise le mot **local**. Ceci permet de réutiliser cette variable (ici **u**) ailleurs sans crainte qu'elle ait pris une valeur bizarre au passage.

```
>     if evalb(expression)=true then u:=1
```

La syntaxe générale de **if** est : **if … then … ; end if ;** On peut rajouter **else** comme ici sans **;** après la première condition ni derrière le **else (**ne pas oublier le **end if;**). Entre la condition et le **endif;** on peut mettre autant d'instructions qu'on le souhaite.

```
>         else
>           u:=0;
```

Le résultat de la procédure est contenu dans la variable **u**. C'est ce résultat que l'on peut réutiliser par la suite.

```
>     end if;  #on peut mettre <fi;> (anciennes versions)
>     u;
```

On peut rappeler la variable contenant le résultat de la procédure si on a un doute.

```
> end;
```

Toutes les procédures doivent se finir par **end ;** ou **end :**. Les fonctions de débugage sont franchement limitées dans Maple, aussi le mieux est de tester en live puis de modifier ce qu'on a fait pour l'intégrer dans une procédure. Ici on n'a pas testé si l'expression est valide (FAIL), la procédure mériterait d'être améliorée.

```
>
```

```
bool := proc(expression)
local u;
    integer; if evalb(expression) = true then u := 1 else u := 0 end if
end proc
```

On réutilise la procédure précédente :

```
> bool((-1<-2));
> bool((-1<0));
> bool((-1<-2) and (-1<0));
> bool((-1<-2) or (-1<0));
>
```

0 1 0 1

Tout va bien. Essayons de tracer une fonction :

```
> f:=x->exp(-x)*bool(x>1)+cos(x)*bool(x<=1);
```

$$f := x \to \mathbf{e}^{(-x)}\,\mathrm{bool}(1 < x) + \cos(x)\,\mathrm{bool}(x \le 1)$$

Malheureusement lorsqu'on essaie de tracer f rien ne va plus… En fait le problème provient de l'interprétation de la condition dans la procédure. Maple ne comprend rien à $x<1$ ou $x>=1$ lorsqu'il y a une variable ! On va donc réécrire la procédure en utilisant le mot réservé **piecewise** :

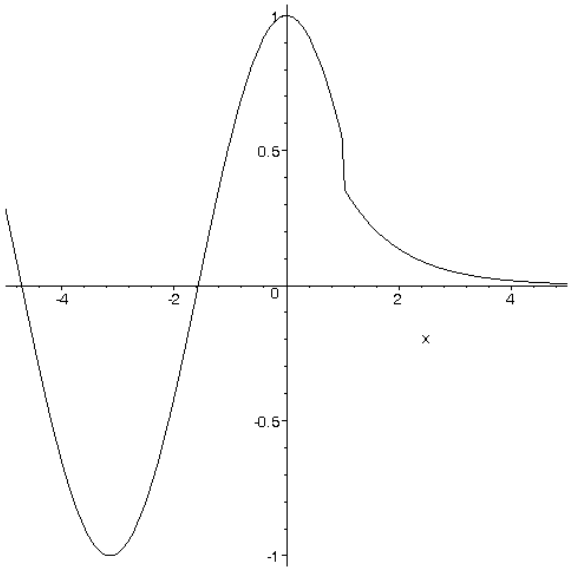
```
> bool_2:=proc(expression)
      piecewise(expression,1,0);
end;
```

```
bool_2 := proc(expression) local u; integer; piecewise(expression, 1, 0) end proc
```

```
> f:=x->exp(-x)*bool_2(x>1)+cos(x)*bool_2(x<=1);
```

$$f := x \to \mathbf{e}^{(-x)}\,\mathrm{bool\_2}(1 < x) + \cos(x)\,\mathrm{bool\_2}(x \le 1)$$

```
> plot(f(x), x=-5..5, color=black);
```



Là c'est bon.

2. Boucles

On utilise essentiellement les deux termes **for** (boucle sans condition d'arrêt) et **while** (avec arrêt). Les syntaxes sont les suivantes :

for (compteur) **while** (conditions) **do** (instructions) **; end do ;**

```
> total := 0;
```

On met la variable **total** à 0 pour éviter les mauvaises surprises.

```
for i from 11 by 2 while i < 100 do
```

i est le compteur sur lequel se fait la boucle, **from** indique que l'on commence à 11, **by** 2 que l'on incrémente de 2 à chaque tour, **while (i < 100)** : on boucle tant que i ne vaut pas 100. On peut omettre **from** et **by** et les remplacer par exemple par

```
for i from 11 to 100 do
```

```
   total := total + i
```

total est modifié pendant les instructions.

```
end do;
```

On clôture le **do**.

Nous allons étudier un autre exemple (plus sophistiqué) qui va montrer la puissance de Maple (notez les # qui permettent de mettre des commentaires…).

```
> u0:=0; n:=5;       #valeurs initiales
f:=x->(1+x^2)/2 ;   #fonction d'itération
```

$$u0 := 0$$

$$n := 5$$

$$f := x \to \frac{1}{2} + \frac{1}{2}x^2$$

```
> L:=NULL:
```

La variable **L** ne contient rien.

```
> uu:=u0:
nn :=n:
```

On stocke les valeurs initiales dans des variables provisoires (c'est utile si on veut transformer par la suite en procédure)

```
L:=[uu,0]:
```

L contient le premier point.

```
for k from 1 to nn do
```

On boucle jusqu'à n.

```
vv:=f(uu);
```

Calcul de u(n+1), stocké dans vv.

```
L:=L,[uu,vv],[vv,vv];
```

Construction de la figure sous forme d'un tableau de points (u_n, $f(u_n)$) puis (u_{n+1}, u_{n+1}).

```
uu:=vv;
```

Retour de u(n+1) dans uu.

```
od:  # on peut mettre <end do:>
```

Fin du **do ;** (vous remarquerez que le : interdit l'affichage de toutes les instructions intermédiaires).

```
d1:=plot([L]):
```

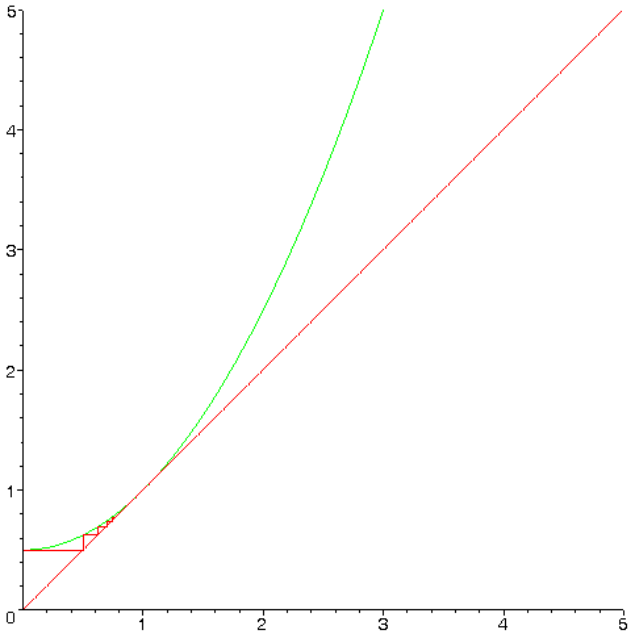
Stockage du tableau de points sous forme de dessin.

```
d2 :=plot({f(x),x},x=0..5,y=0..5):
```

Même chose pour la courbe de f et la droite y = x.

```
plots[display]({d1,d2});
```

On dessine.



3. Exercices

1. Résoudre le système d'équations différentielles de la radioactivité

(voir http://promenadesmaths.free.fr/desintegration.htm)

avec $N_1(0)=1$ et $N_k(0)=0, k>1$.

$$\begin{cases} \dfrac{dN_1}{dt} = -\lambda_1 N_1 \\ \dfrac{dN_{k+1}}{dt} = \lambda_k N_k - \lambda_{k+1} N_{k+1}, k \neq 1, n \\ \dfrac{dN_n}{dt} = \lambda_{n-1} N_{n-1} \end{cases}$$

2. Voilà le mail que j'ai reçu l'autre jour…

> Bonjour, vous avez l'air de bien savoir utiliser maple.
> es ce que vous sauriez faire, avec maple, une somme sur diviseurs par ex, je cherche a faire la convolution entre f et g : f*g (n) = somme (f(d)*g(n/d); où d parcours les diviseurs de n)
> merci de votre réponse
> cordialement

Que faire ?

3. La fonction phi d'Euler (φ) compte pour un nombre n quelconque le nombre de nombres p < n qui ne divisent pas n. Ecrire une procédure donnant φ et tracer φ. Vérifier également que si N est premier avec a, alors $a^{\varphi(N)} \equiv 1(N)$.

4. La fonction *zêta* de Riemann est définie par $\zeta(s) = \sum_{n=1}^{\infty} \frac{1}{n^s} = \prod_{p \text{ premier}} \frac{1}{1-\frac{1}{p^s}}$ pour s complexe.

Vérifier la deuxième égalité.