Benjamin COLLAS
benjamin.collas@telaxo.com

Mi-101 **Mathematica** 2004-2005

Leçon 5 : Les applications

Objectifs :
- créer des fonctions simples et définies par morceaux
- étudier des fonctions de $\mathbb{R}$ dans $\mathbb{R}$
- utiliser un régionnement du plan afin de déterminer des expressions booléennes équivalentes
- étudier sommairement des suites de fonctions

Création de fonctions

■ Affectation de variables

Dans les leçons précédentes, nous avons déja donné des nom à des expressions afin de simplifier leur manipulation.
Pour donner un nom à une expression on utilise le symbole =.
Pour assigner une valeur à une variable, on utilise aussi le symbole =.
De manière générale, l'expression *x=10* peut être traduite par la phrase "mettre 10 dans x"; de même que l'expression *lis=Table[n,{n,1,10,2}]* serait "donner le nom *lis* à l'objet *Table[n,{n,1,10,2}]*".

```
lis = Range[10]
```

On peut alors utiliser directement sur *lis* les fonctions *Mathematica* que l'on utiliserai sur *Range[10]*.

```
lis[[2]]
Length[lis]
```

Un nom de variable doit commencer par une lettre mais peut être composé de lettres et de chiffres. Sa longueur n'est pas limitée.
Par convention, les variables que vous créerez ne devront pas commencer par une lettre majuscule. Seules les fonctions et variables prédéfinies dans *Mathematica* auront cette propriété.

■ Exercice:

Voici une petite subtilité dans l'assignation de variables. Tapez les lignes suivantes et prévoyez les résultats :


```
a = x^2
x = 2
a

x = 1
a
z = x^2
x = 3
z
```

Toutes les variables et expressions que vous introduisez au cours des TD sont gardées en mémoire durant l'utilisation de Mathematica, et ce même si vous effacez leur écriture du notebook.
Pour avoir des informations à leur propos, on utilise le *? :*

```
?a
```

Afin de libérer de la mémoire, on peut effacer les définitions des variables (mais pas leur nom) en utilisant la commande *Clear*.
Lors de la construction de fonctions (qui sont des variables un peu particulières) on est souvent amené à retaper plusieurs fois la définition de celle-ci. Il est alors très utile d'effacer l'ancienne définition avant d'en donner une nouvelle : cela permet d'être certain qu'il n'y aura pas de confusion quant à sa signification.
Pour effacer le nom d'une variable, on utilise la commande *Remove*.

```
Clear[a, x, z]

Remove[a, x, z]
```

■ Fonctions à une ou plusieurs variables

■ Fonction anonyme :

Nous avons déja défini des fonctions anonymes en utilisant la syntaxe :

```
Function[x, expr dépendant de x]
```

Qui à *x* associe *expr dépendant de x.*
On peut alors lui donner un nom et l'utiliser comme on le ferait en Mathématiques :

```
f = Function[x, Log[x]]

f[t]
f[2]
f[1]
f[d / e]

Clear[f]
```

■ La définition de fonction

La manière la plus pratique pour définir une fonction *f* reste celle qui utilise le symbole d'affectation retardée *:=*.

```
f[x_] := Cos[x]
```

```
f[t]
f[Pi / 4]

Clear[f]
```

le signe _ signifie que ce qui précède est la variable. Le signe := signifie que l'affectation ne doit pas avoir lieu dès la validation de la ligne, mais doit être effectuée lors des prochaines évaluations.

■ Exercice:

Avec les lignes suivantes, expliquez pourquoi chacunes des particularités (x_ et :=) doivent être respectées.

```
h[x_] := Tan[x];
hh[x] := Tan[x];
{h[0.], hh[0.], h[x], hh[x], h[truc], hh[truc]}

x = Pi / 4;
h[x_] := Tan[x];
hh[x_] = Tan[x];
{h[0.], hh[0.], h[x], hh[x], h[truc], hh[truc]}

Clear[h, hh, x]
```

■ Exercice :

Créez une fonction *unDeux* qui à une liste associe la liste d'entrée à laquelle on a ôté un élément sur deux.
Créez une fonction *unSurDeux* qui à deux listes associe la liste composée, un élément sur deux des éléments de la première liste et des éléments de la seconde pour ceux restants.

■ Fonctions de plusieurs variables

Nous avons déja rencontré dans *Mathematica* des fonctions admettant plusieurs arguments.

```
Plus[aa, 3, bb]
```

La méthode de construction de vos propres fonctions à plusieurs arguments reste la même:

```
carres[x_, y_, z_] := x^2 + y^2 + z^2

carres[1, 3, -2]
```

■ Exercice :

Construire une fonction *moyenneArith* qui à 3 nombres *a*, *b* et *c* associe leur moyenne $\frac{a+b+c}{3}$.
Construire une fonction *moyenneGeo* qui à 3 nombres *a*, *b* et *c* associe leur moyenne géométrique $(a*b*c)^{\frac{1}{3}}$.

■ Suites ou familles de fonctions

On peut aussi définir une liste de fonctions.

```
jj[x_, n_] := x^n

jj[t, 0]
jj[t, 3]
```

Ou bien encore ce qui est équivalent

```
jj[x_][n_] := x^n

jj[t, 0]
jj[t, 3]
```

■ Fonctions définies par morceaux

En utilisant des test booléens sur la variable, on peut donner une valeur dépendant des caractéristiques de celle-ci.
La commande permettant ceci est */;* qui signifie "dans le cas où".

```
positif[x_ /; x > 0] := 1;
positif[x_ /; x <= 0] := -1;

?positif

Plot[positif[x], {x, -3, 3}];
```

On peut aussi utiliser directement une formulation de type *Si[test ,testvrai ,testfaux, nesaispas]* le troisième argument étant facultatif *:*

```
x = 1;
If[x > 0, "nombre positif", "nombre négatif", "je ne sais pas"]

x = -1;
If[x > 0, "nombre positif", "nombre négatif", "je ne sais pas"]

x = I;
If[x > 0, "nombre positif", "nombre négatif", "je ne sais pas"]
```

■ Fonctions avec des booléens

En combinant les test avec des *And* et des *Or*, on peut affiner les définitions de fonctions.

```
ex[x_] := If[Or[x < 0, x ≥ 5], -2, 1]

Plot[ex[t], {t, -3, 10}];
```

■ Exercice :

Créez une fonction qui vaut *exp[-1/Abs[x^2-1]]* sur *[-1,1]* et *0* ailleurs.

■ Exercice :

Construire une fonction *cercle* qui pour un couple *{x,y}*renvoie *True* si *x ^2 + y^2≤1* et *False* sinon.
Construire une fonction *losange* qui pour un couple *{x,y}*renvoie *True* si *Abs[x]+Abs[y]<=1* et *False* sinon.
Construire une fonction *carre* qui pour un couple *{x,y}*renvoie *True* si *Max[x,y]<=1* et *False* sinon.

■ Exercice :

En utilisant la commande *Random* créez une liste de 5000 points {x,y}, où x et y sont compris entre -1 et 1.
Selectionnez dans cette liste les points ceux qui vérifient *cercle.*
Affichez ces points avec *ListPlot.*
Faites de même avec les fonctions *losange* et *carre.*

■ Propriétés

Vous pouvez spécifier certaines caractéristiques des fonctions définies, grâce à la commande *Attributes* (celle ci sert aussi pour consulter les attributs d'une fonction) .

La première propriété, que nous avons déja rencontrées, est définie par l'attribut *Listable* : si une fonction listable est appliquée à une liste, elle rentre dans la liste et s'applique à chacun de ses éléments.

```
lisNombre = Table[Random[Integer, {-2, 2}], {i, 1, 10}]

ex[lisNombre]

Attributes[ex] = {Listable}

ex[lisNombre]
```

Quelques exemples :

```
Attributes[Length]
```

Protected signifie que la définition de la fonction ne peut être modfifiée par l'utilisateur.

```
Attributes[Plus]
```

OneIdentity signifie que la fonction admet un élément neutre (ici le 0), *Orderless* qu'elle est commutative (*ie Plus[a,b]==Plus[b,a]*) et *Flat* qu'elle est associative (*ie Plus[Plus[a,b],c]==Plus[a,b,c]==Plus[a,Plus[b,c]]*).

■ Exercice :

Rendre la fonction *positif* listable.
Rendre la fonction *carre* listable. Que se passe-t-il ?

Représentations graphiques

Voici quelques options et précisions quant à l'utilisation de *Plot* pour la représentation de fonctions.

■ Tracer des familles de fonctions

Il faut demander une évaluation avant le tracé, en utilisant la commande *Evaluate* :

```
Clear[g]

g[n_][x_] := Sin[x]^n

Plot[Table[g[j][t], {j, 1, 3}], {t, 0, Pi / 2}]

Plot[Evaluate[Table[g[j][t], {j, 1, 10}]], {t, 0, Pi / 2}]
```

On peut ajouter de la couleur à tout cela :

```
clr = Table[Hue[i], {i, 0, 1, .1}];

Plot[Evaluate[Table[g[j][t], {j, 1, 10}]], {t, 0, Pi / 2}, PlotStyle → clr];

clr = Table[Hue[i], {i, 0, 1, .1}];

gg[n_][t_] := Cos[t] * n

Plot[Evaluate[Table[gg[j][t], {j, 0, 1, .1}]], {t, -5 Pi, 5 Pi}, PlotStyle → clr];

clr = Table[Hue[i], {i, 0, 1, .01}];
Plot[Evaluate[Table[gg[j][t], {j, 0, 1, .01}]],
  {t, -2 Pi - Pi / 2, 2 Pi + Pi / 2}, PlotStyle → clr, Axes → False];
```

Outils pour l'étude de fonctions de ℝ dans ℝ

Bien que cette partie simplifie grandement l'étude de fonctions réelles, il faut toujours garder un oeil critique quant aux réponses fournies par *Mathématica*. La source principale d'erreur provient de ce que vous ne posez pas forcément la question que vous pensez, ou qu'elle est tout simplement mal formulée...la réponse fournie est alors juste, mais pas pour la question que vous aviez en tête.

■ Limites

La commande qui vous permet de faire travailler *Mathematica* à votre place :

```
Limit[(a * x^5 + b * x^3 - c) / (d * x^5 + e * x - f), x → Infinity]
Limit[Sin[x] / x, x → 0]
Limit[Log[x] / x, x → 0]
```

■ Exemple 1:

```
f0[x_] := x^2 + 1 / (x^2 - 5)

Plot[f0[x], {x, -5, 5}];

Limit[f0[t], t → -Infinity]

Limit[f0[t], t → Sqrt[5], Direction → 1]
Limit[f0[t], t → Sqrt[5], Direction → -1]
```

■ Exemple 2 :

```
f1[x_] := (x + 1) / (x^2 - 3)
```

```
Plot[f1[t], {t, -5, 5}];

Limit[f1[t], t → -Infinity]

Limit[f1[t], t → Sqrt[3], Direction → 1]
Limit[f1[t], t → Sqrt[3], Direction → -1]
```

■ **Exercice :**

Tracez, étudiez les limites et trouvez une asymptote oblique de :

```
f2[x_] := x + 1 / (x^2 - 3)
```

■ Dérivées

Pour dériver, on utilise le ' ou la commande *Derivative*

```
f0'[x]
```

$$2\,x - \frac{2\,x}{(-5+x^2)^2}$$

```
Derivative[3][f0][x]
```

$$-\frac{48\,x^3}{(-5+x^2)^4} + \frac{24\,x}{(-5+x^2)^3}$$

Attention toute fois, le résultat retourné n'est que la valeur de la fonction dérivée au point *x*. Si l'on veut obtenir la fonction dérivée, il nous faut créer une fonction en utilisant les règles de remplacement.

```
f0Deriv[t_] := f0'[x] /. {x → t}

f0Deriv[3]

f0Deriv[z]
```

Voici une fonction qui permet d'obtenir la fonction tangente en un point *x0* d'une courbe:

```
tangente[unefonction_][x0_][x_] := unefonction'[x0] * (x - x0) + unefonction[x0]

g0[x_] := x^3

tangente[g0][1][t]
```

■ **Exercice :**

Tracez la famille des courbes tangentes à *g0* en certains points