

Programmer avec Maple

1. Introduction

L'utilisation que l'on fait généralement de *Maple* est une utilisation essentiellement *directe* : nous utilisons une commande, ou une série de commandes, qui affichent les résultats dans l'ordre où ces commandes étaient écrites.

Dans les cours de mathématiques, on insiste pour que ces commandes respectent un ordre cohérent particulièrement pour les *procédures* :

- 1) Initialisation avec la commande **restart**;
- 2) Entrée des données (paramètres);
- 3) Définitions des variables nécessaires;
- 4) Utilisation des commandes nécessaires pour résoudre le problème posé.

Une fois écrites, ces procédures permettent de résoudre des classes déterminées de problèmes où il suffit d'entrer les valeurs des *paramètres* propres à un problème particulier pour en obtenir la solution.

La programmation permet de généraliser à un niveau beaucoup plus puissant ce type de démarche propre à l'ensemble des langages de programmation. Par rapport à la démarche utilisée jusqu'à maintenant, notamment dans les procédures, la programmation se caractérise par les éléments suivants :

- Les étapes à suivre sont représentées par un schéma, un organigramme, qui représente la structure de la solution (algorithme), et où sont précisées les actions à poser;
- La série de commandes, traduisant ces actions à poser, est « encapsulée » dans une commande de *Maple*, **proc()**, qui transforme cette série de commandes en une commande unique, le *programme*;
- La séquence des commandes figurant dans ce programme correspond à la séquence que nous avons suivie : initialisation, définition des paramètres et entrée des données, utilisation des commandes nécessaires et affichage des résultats souhaités.

Cette démarche nécessaire pour établir un programme est une démarche de type *algorithmique*. Elle permet de résoudre une classe beaucoup plus large de problèmes que ne le permettait la démarche des *procédures* et aussi d'utiliser plus efficacement les capacités des ordinateurs.

Elle permet d'atteindre les objectifs d'apprentissage suivants :

- Initier l'élève aux rudiments d'une démarche algorithmique;
- Nommer et reconnaître les structures algorithmiques de base;
- Transposer des algorithmes dans le langage de programmation de *Maple*;
- Intégrer l'algorithmique et la programmation à une démarche de modélisation.

Ce qui suit n'est qu'une simple initiation à ce que permet *Maple* en terme de programmation. Sur cette base vous pourrez par la suite développer progressivement votre maîtrise des outils et de la puissance que vous offre *Maple*. Vous pourrez aussi utiliser ces bases pour vous initier à d'autres langages de programmation comme Pascal, C, etc.

2. Structure algorithmique

Un algorithme est une suite ordonnée d'opérations éventuellement transposées en instructions d'un langage de programmation pour constituer un programme à exécuter sur un ordinateur. Un algorithme est basé sur une écriture systématique selon les règles de programmation structurée de la solution d'un problème en vue de sa solution à l'aide d'un ordinateur.

Cette suite doit se traduire par un schéma simple, qui est celui de la solution du problème. Ce schéma est généralement appelé un *organigramme*.

L'organigramme permet d'illustrer la séquence des commandes et les structures de contrôle propres à tout langage de programmation : les structures de *séquences*, *alternatives* (choix) et *répétitives* (boucles).

2.1 Structure en séquence

Comme le nom l'indique, une structure en séquence correspond à une suite d'instructions que *Maple* effectue dans l'ordre où elles sont entrées. Une structure en séquence se traduit par le schéma ci-dessous, dans la colonne de gauche. Dans la colonne du centre, la structure est décrite en pseudo-code, c'est-à-dire en langage indépendant du langage de programmation utilisé. La colonne de droite indique l'écriture en code *Maple*. Comme exemple, nous avons pris le cas de la détermination de l'équation d'une tangente à une fonction.

Structure	Pseudo-code	Code Maple
	<p><i>Initialisation</i></p> <p><i>Paramètres et données du problème</i></p> x_0 $y = f(x)$ <p><i>Étapes (commandes)</i></p> <p>calcul de la pente m</p> <p>calcul de l'ordonnée à l'origine b</p> <p>détermination de l'équation de la tangente</p> <p><i>Affichage des résultats</i></p> <p>représentation de la courbe $y = f(x)$ et de sa tangente au point d'abscisse x_0</p>	<pre> restart; "Données"; f:=x->x^2-4; x0:=3; "Commandes"; m:=D(f)(x0); b:=f(x0)-m*x0; y:=x->m*x+b; "Affichage"; plot({f(x),y(x)}, x=-5..5); </pre>

2.2 Structure alternative

Une structure alternative intervient lorsqu'il y a un choix à effectuer. Ce choix doit être formulé de telle façon que *Maple* ait à évaluer si cette formulation est vraie ou fausse.

Il faut prévoir deux suites d'instructions : celle que *Maple* doit effectuer au cas où la formulation est vraie et celle que *Maple* doit suivre au cas où elle est fausse.

La commande alternative à condition simple s'écrit de la façon suivante :

```

if condition then
    séquence_1
else
    séquence_2
fi;

```

Cela signifie que si la *condition* est vérifiée, *Maple* exécutera la *séquence_1*, si elle n'est pas vérifiée, elle exécutera la *séquence_2*. Il faut remarquer que la commande **if** doit toujours se terminer par **fi**.

Remarques

L'alternative **else** n'est pas obligatoire. Dans ce cas, si la *condition* n'est pas satisfaite, *Maple* passe à l'instruction suivante.

Il n'y a pas de point-virgule après les éléments **if**, **then** et **else** de la commande alternative. L'élément **fi** est suivi d'un point-virgule. Ces éléments doivent cependant être suivis d'un espace.

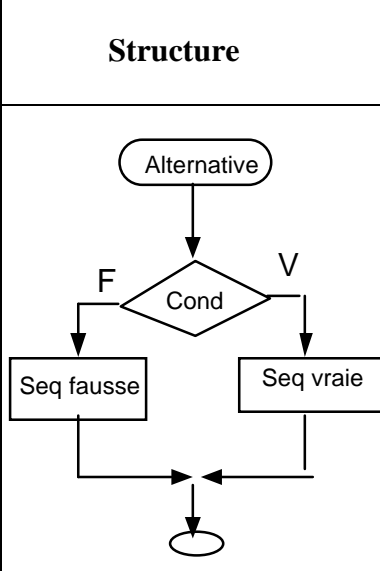
Dans un exemple, plus loin, nous verrons que l'on peut ajouter l'élément **elif** lorsqu'il y a plusieurs alternatives.

Nous avons utilisé un décalage (indentation) des instructions pour montrer l'imbrication de la séquence d'instructions dans l'alternative.

Exemple

Comme exemple, nous chercherons à obtenir le plus grand de deux nombres *a* et *b*.

Cette structure alternative se traduit par le schéma ci-dessous :

Structure	Pseudo-code	Code Maple
	<p><i>Initialisation</i></p> <p><i>Paramètres et données du problème</i> Soit deux nombres <i>a</i> et <i>b</i></p> <p><i>Condition</i> Si $a < b$</p> <p><i>Séquence vraie</i> On veut <i>b</i></p> <p><i>Séquence fausse</i> On veut <i>a</i></p>	<pre>restart; a:=2;b:=5; if a<b then b; else a; fi;</pre>

2.3 Structure répétitive

Une structure répétitive intervient lorsqu'il y a un calcul répétitif à effectuer tant qu'une condition reste vraie. La condition doit être formulée de telle façon que *Maple* ait à évaluer si elle est vraie ou fausse.

Il faut prévoir une suite d'instructions que *Maple* doit effectuer tant que la condition est vraie. Dès qu'elle devient fausse, *Maple* sort de la structure répétitive.

La commande répétitive conditionnelle s'écrit généralement de la façon suivante :

```
while condition do
  Séquence_1;
  Séquence_2;
  .....
od;
```

Dans cette commande, *Maple* exécute *Séquence* tant que la *condition* est vraie. Quand la *condition* est fausse, la boucle se termine avec **od**.

On remarque que les éléments **while** et **do** de la commande d'itérations conditionnelles sont suivies d'un espace et non d'un point-virgule.

Exemple

Comme exemple, calculons les termes d'une série géométrique de raison 2, de premier terme 5, $5, 5, 5 \times 2, 5 \times 2^2, \dots, 5 \times 2^n, \dots$, jusqu'à 100.

Cette structure répétitive se traduit par le schéma ci-dessous :

Structure	Pseudo-code	Code Maple
	<p><i>Initialisation</i> Le premier terme est 5</p> <p><i>Condition</i> Tant que le terme obtenu, $2u$ sera inférieur à 100</p> <p><i>Instruction</i> Calculons $2u$</p>	<pre>restart; u:=5; while 2*u<100 do u:=2*u od;</pre>

2.4 Structure itérative

Une structure itérative est une structure répétitive dans laquelle on prévoit le nombre de répétitions d'une séquence d'instructions. À cet effet, on devra gérer un compteur, ou

itérateur, dont la valeur sera initialisée avant l'entrée dans la boucle, variée (augmentée ou diminuée) dans la répétitive et testée dans la condition.

La commande itérative s'écrit de la façon suivante :

```
for compteur from debut to fin by pas do  
    Séquence_1;  
    Séquence_2;  
    .....  
od;
```

Dans cette commande, *compteur* est l'itérateur qui varie de *debut* à *fin* selon le *pas*. Pour chaque valeur de l'itérateur, *Maple* exécute les instructions *Instruction_1*, *Instruction_2*, etc. Quand l'itérateur atteint la valeur *fin*, la boucle se termine avec **od**.

Si le *pas* n'est pas mentionné, *Maple* prend un pas de 1 par défaut.

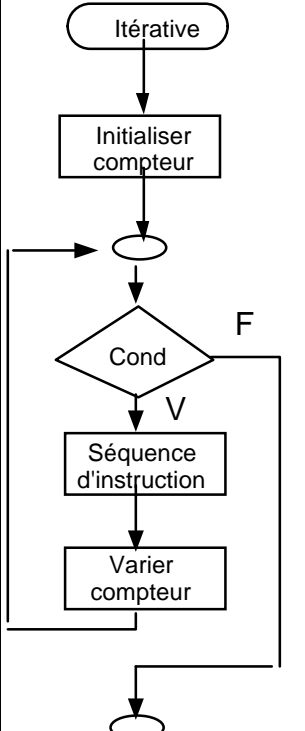
On remarque que les éléments **for**, **from**, **to**, **by**, **do** de la commande d'itérations répétitives sont suivis d'un espace et non d'un point-virgule.

Cette structure peut ne pas inclure les éléments **from** et (ou) **by**. La valeur par défaut est alors 1.

Exemple

Comme exemple, calculons les carrés des 5 premiers nombres entiers

Cette structure répétitive se traduit par le schéma ci-dessous.

Structure	Pseudo-code	Code Maple
 <pre> graph TD Start([Itérative]) --> Init[Initialiser compteur] Init --> Conn1(()) Conn1 --> Cond{Cond} Cond -- V --> Seq[Séquence d'instruction] Seq --> Var[Varier compteur] Var --> Conn1 Var --> End([]) </pre>	<p><i>Initialisation</i> Ici le calcul porte sur l'itérateur i qui commence à 1</p> <p><i>Condition</i> Tant que i est compris entre 1 et 5</p> <p><i>Instruction</i> Calculer i^2</p>	<pre> restart; for i from 1 to 5 do i^2; od; </pre>

Remarques

La structure itérative est assez proche de la commande `seq()`.

Le logiciel *Maple* utilise parfois la version ci-dessous :

```

for compteur from début by pas while condition do
  Séquence_1;
  Séquence_2;
  .....
od;
  
```

Dans cette version on utilise un *compteur*, ou *itérateur* qui prend initialement la valeur *début* et qui à chaque boucle augmente de *pas*. À chaque boucle la variable d'itération est vérifiée pour s'assurer qu'elle répond à la *condition*.

3. Les programmes avec Maple

Un programme est une combinaison des différentes structures algorithmiques que nous venons de voir : des structures en séquence, alternatives, répétitives ou alternative. Pour définir un programme, *Maple* possède une commande particulière, la commande **proc()**.

3.1 La commande proc()

La commande **proc()** est une commande de *Maple* qui « encapsule » une série d'instructions écrites selon une séquence semblable à celle que nous avons utilisée particulièrement dans les exercices de types *Procédures*. Cette séquence débute généralement par l'introduction des variables données, puis par la définition des variables propres au problème et l'utilisation des commandes nécessaires et se termine par l'affichage des résultats. Une fois bien définie, on peut lui attribuer un nom et *Maple* la considère comme un programme. En fait, 80 % des commandes régulières de *Maple* sont des programmes définis à l'aide de la commande **proc()**.

La commande **proc()** systématise cette séquence et se présente sous la forme simplifiée suivante :

```
nom_du_prog :=proc(para1,para2,para3,...)  
local v11, v12, v13;  
global vg1, vg2 ,vg3 ;  
Instruction 1;  
Instruction 2;  
.....  
print(graphique 1, graphique 2, ...);  
RETURN(résultat 1, résultat 2, ...);  
end;
```

Reprenons, ligne par ligne, la forme simplifiée ci-dessus de la commande **proc()**.

```
nom_du_prog :=proc(para1,para2,para3,...)
```

Les arguments de la commande **proc()**, **para1**, **para2**, **para3...** sont les *paramètres* en fonction desquels on souhaite obtenir une réponse. Ici la notion de *paramètre* correspond à la notion de *variable indépendante* en mathématique; **nom_du_prog** est le nom que l'on donne à cette procédure et nous permettra de l'appeler. On remarque que cette première ligne se termine sans point-virgule.

```
local v11 v12 v13  
global vg1 vg2 vg3
```

Les variables utilisées dans le cadre de la commande **proc()** doivent être identifiées comme étant des variables locales (**local**) ou globales (**global**). Les valeurs attribuées aux variables locales ne sont reconnues qu'à l'intérieur de la commande **proc()**. Elles peuvent prendre d'autres valeurs ailleurs dans la feuille de calcul. Au contraire, une variable qui est déclarée

comme **global** conserve la même valeur ailleurs dans la feuille de calcul. Il est donc conseillé de déclarer les variables que l'on utilise comme **local**.

Instruction 1

Instruction 2

Il s'agit des commandes nécessaires pour résoudre le problème. Elles peuvent se regrouper en structures de types séquentielles, alternatives, répétitives et itératives.

print()

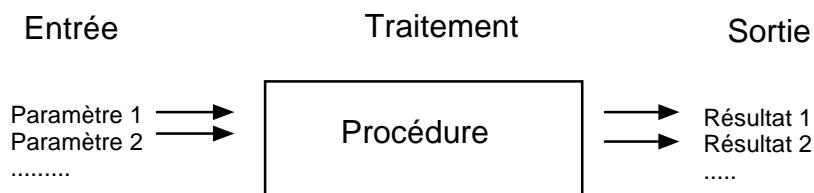
RETURN()

La commande **print()** permet d'afficher des graphiques, tandis que la commande **RETURN()** permet d'afficher des résultats. Ces commandes ne sont pas indispensables et *Maple* afficherait alors le résultat de la dernière instruction.

end

Cette dernière commande, indispensable, indique à *Maple* que la procédure est complète.

On peut considérer une procédure comme une « boîte noire » qui reçoit des données en entrée, effectue un traitement sur ces données et produit un résultat de sortie. Les données entrées sont généralement qualifiées de « paramètres ». Cela se schématise de la façon suivante :



En fait, il est particulièrement important de définir deux choses :

1. Les paramètres dont doit dépendre le programme et qui doivent figurer comme arguments de la commande **proc()**;
2. Les variables représentant les réponses que l'on souhaite afficher et qui doivent figurer comme arguments de la commande **RETURN()** ou **display()**.

3.2 Les étapes de la programmation, ou une application du cycle de modélisation

Un programme est donc une suite ordonnée d'instructions permettant de résoudre un problème. Pour établir un programme, il faut au préalable bien comprendre le problème et établir le schéma de la solution. Il est donc conseillé de suivre les étapes ci-dessous :

1. Analyse :

Il faut au départ bien cerner la problématique, c'est-à-dire identifier la nature du problème, les outils et résultats mathématiques correspondant à ce type de problème.

Il faut ensuite énumérer les paramètres figurant dans les données (commande **proc()**) et les variables qui apparaissent dans les réponses (commandes **print()** et **RETURN()**).

Il faut ensuite schématiser le problème de façon à identifier les structures qui apparaissent et leur séquence. Une fois identifiées, ces structures, séquentielles, alternatives, répétitives et itératives, doivent être étudiées successivement.

Dans le cas des structures itératives ou répétitives, il faut identifier l'itérateur.

Il faut se faire une première idée des étapes qu'il faudra suivre et identifier les différentes structures algorithmiques qui apparaissent.

Cette étape correspond à la première étape du cycle de modélisation.

2. Schéma (organigramme) :

Les étapes doivent être précisées sous forme d'un organigramme. Cet organigramme reprend les schémas propres aux différentes structures identifiées. Ces étapes peuvent s'écrire en langage quasi-courant (pseudo-code), ce qui clarifie la nature des choix ou des calculs répétitifs à effectuer. Cela facilite aussi le passage de l'organigramme à l'écriture en code *Maple*. Cette étape correspond à la seconde étape du cycle de modélisation.

3. Traduction en langage *Maple* et validation :

Chacune de ces étapes est traduite par une instruction *Maple*.

Une fois les instructions écrites, le programme est évalué au niveau de la syntaxe : *Maple* signale souvent les erreurs d'écriture ou de cohérence des instructions, mais il faut parfois utiliser les commandes de « débogage ». Cette étape correspond à la troisième étape du cycle de modélisation.

4. Évaluation :

Il faut s'assurer que le programme donne de bonnes solutions en le testant sur des cas particuliers. Il faut, si nécessaire, l'ajuster. Il est également possible d'évaluer le temps de

calcul avec la commande `time()` et de chercher des simplifications. Cette étape correspond à la quatrième étape du cycle de modélisation.

Exemple

Déterminez un programme permettant de représenter une fonction f et sa tangente au point d'abscisse x_1 .

Solution

Suivons les étapes indiquées.

1. Analyse

La tangente à une fonction $y = f(x)$ est une droite de la forme $y = mx + b$. La pente m de la tangente à une fonction est la valeur de la dérivée en ce point. L'ordonnée à l'origine b s'obtient en écrivant que l'équation de la tangente doit vérifier les coordonnées $(x_1, f(x_1))$ du point de tangence. Il faudra donc calculer m et b .

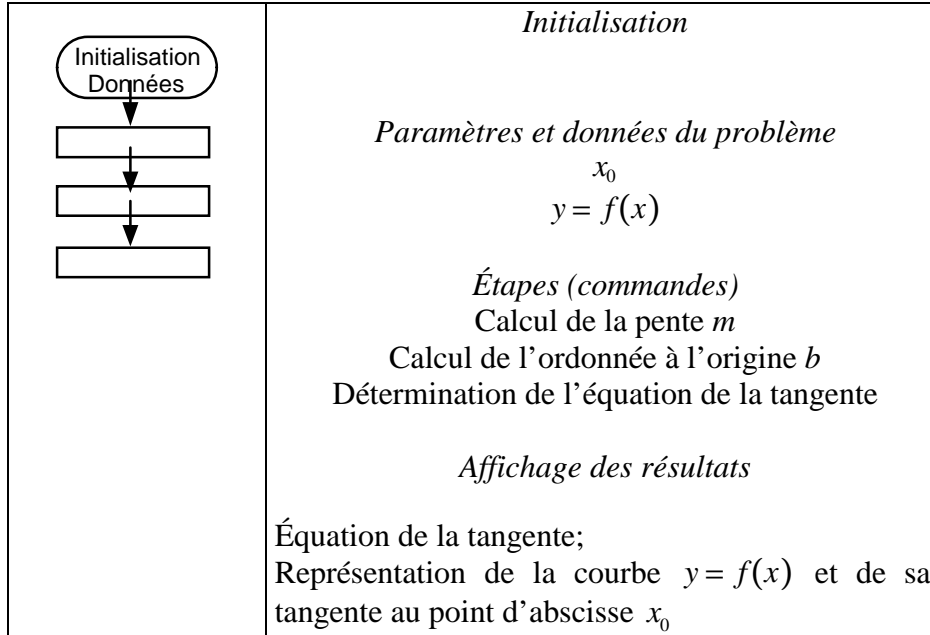
Les paramètres dont nous avons besoin sont la fonction $y = f(x)$ et le point de tangence x_1 . Ce seront les arguments de la commande `proc()`.

Nous voulons obtenir l'équation de la tangente et nous voulons aussi obtenir les graphiques de la fonction et de sa tangente. Ce seront les arguments des commandes `RETURN()` et `display()`.

En terme de structure, il semble n'y avoir ni alternative à envisager, ni calcul répétitif ou itératif. Nous aurons donc une structure en séquence.

2. Schéma (organigramme)

Pour obtenir l'équation de la tangente, il faut donc obtenir la pente m et l'ordonnée à l'origine b . Ces résultats s'obtiennent par une simple séquence représentée par la figure ci-dessous .



3. Traduction en langage Maple et validation

La séquence des instructions résumées sur le schéma se traduit par la séquence d'instructions suivantes en langage *Maple* :

```
droite:=proc(f, x1, xi, xf, yi, yf)
local m, b, dr, g1, g2;
with(plots):
m:=D(f)(x1);
b:=f(x1)-m*x1;
dr:=m*x+b;
g1:=plot(dr, x=xi..xf, yi..yf);
g2:=plot(f(x), x=xi..xf, yi..yf);
print(display(g1, g2));
RETURN(dr);
end;
```

On obtient le programme ci-dessous défini par la commande **proc()** :

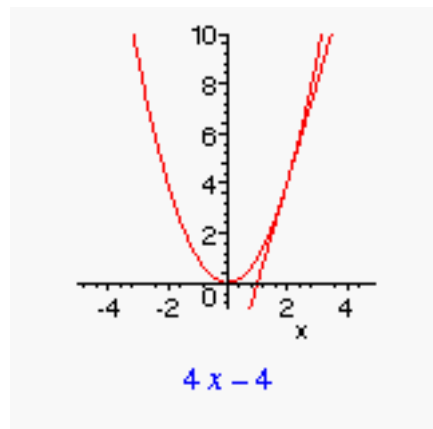
```

droite := proc(f, x1, xi, xf, yi, yf)
local m, b, dr, g1, g2;
  with(plots);
  m := D(f)(x1);
  b := f(x1) - m*x1;
  dr := m*x + b;
  g1 := plot(dr, x = xi .. xf, yi .. yf);
  g2 := plot(f(x), x = xi .. xf, yi .. yf);
  print(display(g1, g2));
  RETURN(dr)
end

```

Validons le programme obtenu pour la fonction $y = f(x) = x^2$ au point d'abscisse $x_1 = 2$.
 Nous obtenons

```
droite(x->x^2, 2, -5, 5, -5, 5);
```



4. Évaluation

Le programme pourrait être complété en évaluant les extremums, les éventuelles asymptotes verticales, l'étude de la fonction dérivée, etc.

Bibliographie

- [1] CORNIL, Jack-Michel, Philippe TESTUD. *Maple, Introduction raisonnée*, Paris, Éditions Springer Verlag, 1995, 463 p.
- [2] DUCHESNE, Pierre. *Initiation au langage symbolique Maple*, Québec, Université Laval, 1999, 51 p.
- [3] ETCHECOPAR, Philippe. *Manuel d'initiation à Maple V*, Rimouski, Presses pédagogiques de l'Est, 1999, 109 p.

- [4] LALANCETTE, Paul-Edmond, *Laboratoires Maple V (4 cahiers)*, Québec, Éditions Le Griffon d'argile, 1999.
- [5] LANTAGNE, Pierre. *Calcul 1 Laboratoires Mathématiques avec Maple*, Québec, Éditions Gaétan Morin, 2000, 261 p.
- [6] LEROUX, Alain, Roland POMES. *Toutes les applications de Maple*, Paris, Éditions Vuibert, 1995, 282 p.
- [7] MONAGAN, M.B., K.O.GEDDES, K.M. HEAL, G. LABAHN et S.M. VORKOETTER. *Maple V Programming Guide*, New-York, Éditions Springer Verlag, 1998, 379 p.
- [8] RAMBACH, Philippe. *MapleV en classes prépas*, Paris, Éditions ellipses, 1998, 142 p.
- [9] REDFERN, Darren. *The Maple Handbook*, New-York, Éditions Springer Verlag, 1996, 495 p.

Site Internet

<http://www.maplesoft.com>