Université de Strasbourg **L2 MPC/PSI**

Calcul Formel et Simulation Numérique

2010-2011

Table des matières

Chapitre 1

Introduction à Maple

1.1 Introduction

Maple est un logiciel (payant) de calcul formel. Il permet de faire des calculs (comme une calculatrice), de manière exacte ou en utilisant une précision arbitraire. Les algorithmes présentés par la suite seront implémentés en Maple.

1.2 Instruction(s), exécution(s) et résultat(s)

Après un certain temps, vous voyez apparaître une interface graphique avec un fichier nommé *Untitled(1)* et un curseur qui attend que vous tapiez votre première instruction :

```
2+2;
```

N'oubliez pas le `;`, car il signifie la fin de l'instruction et que le résultat sera affiché. Ne rajouter par l'invite de commande `>` : il est déjà écrit et correspond à un bloc d'instructions qui est exécuté dès que l'on appuie sur la touche *Entrée*.
A la place du `;`, on peut aussi utiliser le `:`. Dans ce cas l'instruction sera exécutée, mais non affichée. Cela peut être pratique si l'on ne veut pas que l'ordinateur affiche plein de résultats intermédiaires qui ne sont pas utiles.
Lorsque l'on appuie sur la touche *Shift+Entrée*, on va à la ligne sans exécuter. Cela permet d'écrire un bloc d'instructions de manière plus claire et évite de devoir appuyer de nombreuses fois sur la touche *Entrée* pour arriver au résultat final.

1.3 Opérations de bases

Voici quelques exemples d'utilisation de Maple comme calculatrice :

```
sqrt(2+3*7/5);
abs(1-2**5);
cos(Pi/2);exp(1);ln(1);
```

De manière générale pour avoir l'aide sur une commande, on peut écrire `?` suivi du nom de la commande

```
?int
?plot
?solve
```

1.4 Les variables

Contrairement à d'autres langages de programmation, comme le C ou le Fortran, les variables ne sont pas déclarées à l'avance et l'utilisateur n'a pas besoin de préciser le type (*réel,entier,caractère, pointeur...*).
Il existe des variables prédéfinies :

```
Pi,Digits,I;
```

`I` est le nombre complexe i (on a $i^2 = -1$). Certaines de ces variables peuvent être modifiées (comme `Digits`) et d'autres non (comme `I` et `Pi`).
Les autres variables sont définies à partir du moment où elles sont affectées. L'affectation d'une variable se fait par l'instruction `:=`.

```
a:=1:a;
```

La valeur 1 est mise dans la variable `a`. Il peut parfois être intéressant de libérer une variable, c'est-à-dire d'enlever la valeur qu'elle contenait. Cela se fait à l'aide de la fonction `unassign`.

```
a:=1:a;unassign('a'):a;
```

Ainsi, dans cet exemple, `a` redevient une variable formelle.
Pour réinitialiser toutes les variables et tout ce qui a été fait auparavant, on utilise la commande

```
restart;
```

1.5 Boucles et instructions conditionnelles

1.5.1 Boucles

```
a:=0:for i from 0 to 10 do a:=a+1:od:a;
```

Par défaut c'est `from 1`.

1.5.2 Tests if/while

```
if (..) then .. else .. fi:
while (..) do .. od:
```

1.5.3 Opérateurs logiques

```
and,or,<>,=,not
```

1.6 Les fonctions

1.6.1 Fonctions et expressions

Supposons que l'on veuille définir la fonction $f(x) = x^2 + 1$, afin de calculer par exemple son intégrale. Pour cela, il y a deux manières de faire : soit on écrit

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

et dans ce cas, pour évaluer la fonction en un point, par exemple pour $x = 0.1$, on écrit

```
f(0.1):
```

soit, on considère l'expression

```
f:=x^2+1:
```

et dans ce cas, pour évaluer l'expression en $x = 0.1$, on écrit

```
subs(x=0.1,f);
```

Remarquons que l'on peut passer d'une forme à l'autre :

```
f:=x^2+1:f:=unapply(f,x);
f:=x->x^2+1:f:=f(x);
```

Notons que dans le cas d'une fonction la variable est *muette* ; on peut la changer sans changer la définition (on aurait pu écrire par exemple `f :=y->y^ 2 ;`). Ceci n'est pas vrai pour une expression où il faut faire attention que la variable utilisée soit libre (on peut toujours libérer une variable avec `unassign`, comme nous l'avons déjà vu). En général, on préfèrera les expressions qui sont des objets plus simples.

1.6.2 Procédures

Lorsque l'on veut faire plusieurs calculs avec des paramètres différents, il peut être intéressant d'englober le calcul dans une *procédure*. Cet objet se présente de la manière suivante :

```
toto:=proc(f,N)
 local i,tmp;
 tmp:=0.;
 for i to N do tmp:=tmp+evalf(subs(x=i/N,f)):od:
 tmp/N;
 end proc:
f:=x^2:N:=10:toto(f,N);N:=20:toto(f,N);f:=exp(-x^2):toto(f,N);N:=40:toto(f,N);
```

Pour le premier `>`, on définit la fonction ; `toto` est le nom, `f,N` sont des variables paramètres qui sont utilisées dans la procédure, `i` et `tmp` sont des variables locales c'est-à-dire qu'elles n'existent qu' à l'intérieur de la procédure. Le dernier résultat avant la fin de la procédure (matérialisé par `end proc :`) est ce qui sera renvoyé par la procédure lors de l'appel.
Pour le deuxième `>`, on appelle la fonction avec différents paramètres.

1.7 Les tableaux

Considérons maintenant la suite définie de la manière suivante :

$$c_1 = \frac{1}{7}, \quad \frac{5k+9}{6} c_k = \sum_{j=1}^{k-1} j c_j c_{k-j} + \frac{1}{3} - \sum_{j=1}^{k-1} c_j, \quad k \geq 2.$$

Nous voyons que pour calculer le $k^{\text{ième}}$ terme c_k, nous avons besoin de tous les termes précédents $c_{k-1}, \ldots, c_1$. Nous avons donc besoin de stocker ces valeurs une fois qu'elles sont calculées.

1.7.1 La commande `array`

Pour cela, nous pouvons utiliser un tableau par la commande

```
N:=100:c:=array(1..N):
```

Cela veut dire que l'on a réservé des variables $c_1, \ldots, c_N$ que l'on pourra affecter par la suite. On peut écrire ainsi :

```
c[6]:=1:
```

Notons que les variables peuvent être de n'importe quel type ; en particulier, l'espace occupé par les variables non encore affectées n'est pas alloué, mais sera alloué lors de l'affectation.

Remarque 1. *Lorsque l'on connait la taille des objects, on peut parfois réserver à l'avance la place mémoire. Cela peut augmenter les performances pour le temps d'accès au tableau. Néanmoins, on ne connait pas toujours la place occupée par les variables (par exemple, certains entiers peuvent être très grands et nécessiteront donc beaucoup de stockage). Le lecteur intéressé pourra consulter l'aide sur la commande* `rtable`.

Un intérêt de la commande `array` est que l'on peut faire commencer et terminer les indices où l'on souhaite. Ainsi, on peut écrire

```
deb:=-10:fin:=15:c:=array(deb..fin):
```

On peut également indicer les tableaux en plusieurs dimensions

```
N:=10:M:=12:L:=array(1..N,1..M):
```

L'accès se fait alors par `L[i,j]`. Pour calculer la suite précédente, on peut donc écrire.

```
N:=30:L:=array(1..N):L[1]:=1/7:
for k from 2 to N do
S:=add(L[j],j=1..k-1):T:=add(j*L[j]*L[k-j],j=1..k-1):
L[k]:=(6/(5*k+9))*(T-S+1/3);
od:evalf(L[N]);
```

On a utilisé ici la commande `add` qui fait la somme des éléments de la liste (on aurait aussi pu utiliser la commande `for`) et on a utilisé `from`, pour faire commencer k par 2 et non la valeur 1 par défaut.
Pour afficher tous les éléments du tableau, on peut écrire

```
eval(L);
```

1.7.2 Les listes

On peut aussi définir des tableaux par des listes de la manière suivante :

```
L:=[1,4,7];
```

Un intérêt est la déclaration qui est plus simple. On peut aussi accéder au nombre d'éléments à l'aide de la commande `nops`

```
nops(L);
```

La fonction `nops` compte le nombre d'*opérandes* de l' objet L et les opérandes sont données par la fonction `op`. L'affichage de toute la liste se fait tout simplement en écrivant

```
L;
```

Attention, dans une liste, les indices commencent par 1 :

```
L:=[3,6,7]:L[1];op(1,L);
```

La valeur 3 est ici affichée, puisque c'est le premier élément de la liste, qui est aussi la première opérande de L.
Notons aussi qu'une liste est une *séquence* entre crochets et pour générer une séquence, il est commode d'utiliser la commande `seq`

```
L:=[seq(i*i,i=1..5)];
```

De manière générale, on préferera utiliser `array` pour des grands tableaux et des listes pour des petits tableaux (qui peuvent être une liste de paramètres à tester, que l'on mettra de préférence au début du programme).
Remarquons aussi la commande `map` qui applique une fonction à une liste (ou un `array`) :

```
N:=10:L:=[seq(k/N,k=0..N)]:f:=x->exp(x):fL:=map(f,L):
```

1.8 Graphiques

1.8.1 Des fonctions

On peut visualiser des fonctions par :

```
f:=sin(4*x):plot(f,x=0..2*Pi);
f:=x->sin(4*x):plot(f,0..2*Pi);
```

Dans le premier cas, on utilise la forme *expression* tandis que dans le deuxième cas, il s'agit de la forme *fonction*. On peut aussi visualiser plusieurs graphiques en même temps ; cela peut être extrêmement utile lorsque l'on veut faire une comparaison. Pour cela, on utilise

```
plot([sin,cos],0..Pi,color=[red,blue],legend=["sin","cos"]);
```

Remarquez les options utilisées pour bien distinguer les courbes, grâce aux couleurs et à la légende.

Exercice 1. *A l'aide de la fonction* `plot`*, localiser le* $5^{ième}$ *zéro strictement positif de la fonction* $x - tan(x)$*, à* 10^{-2} *près ; vérifier le résultat avec* `fsolve`*.*

Un autre moyen de dessiner deux courbes en même temps est d'utilise la fonction `display` qui fait partie de la libraire `plots`. Pour cela, on charge d'abord la librairie en écrivant

```
with(plots):
```

Puis, on définit les deux graphiques de chaque courbe avant de créer la graphique qui contiendra les deux courbes.

```
G1:=plot(sin,0..Pi,color=red,legend="sin"):G2:=plot(cos,0..Pi,color=blue,legend="cos"):
display(G1,G2);
```

1.8.2 Un ensemble de points

On peut aussi dessiner un ensemble de points, ce qui peut être utilisé si on connait la fonction cherchée seulement en un nombre fini de points. Pour cela, la commande est la suivante

```
plot([[0,0.5],[1,0.2]]);
```

Ainsi les points $(0, 0.5)$ et $(1, 0.2)$ sont reliés par un segment (on peut ne tracer que les points en rajoutant l'option `style=point`). On peut également tracer des courbes paramétrées de la forme $(x(t), y(t))$; par exemple, pour dessiner le cercle de centre $(0, 0)$ et de rayon 1, on écrit

```
plot([cos(t),sin(t),t=0..2*Pi],scaling=constrained);
```

L'option est mise pour que les axes soient les mêmes (sinon, on aurait une ellipse !).

1.8.3 Faire une animation

Il peut être intéressant de vouloir dessiner l'évolution d'un graphique ; cela s'obtient facilement avec la fonction `display`, en rajoutant l'option `insequence=true` : on définit tout d'abord la liste d'images et on appelle ensuite la fonction `display` que l'on applique à cette suite. En cliquant sur l'image, on voit ensuite apparaître des boutons *play, pause,...* qui permettent de contrôler l'animation.

1.9 Résolution d'équations différentielles

On souhaite simuler le déplacement $x(t)$ d'une masse accrochée à un ressort. Par application du théorème de Newton, on obtient l'équation différentielle :

$$m\frac{d^2}{dt^2}x(t) + \alpha\frac{d}{dt}x(t) + kx(t) = 0$$

Maple possède différents outils permettant la résolution d'équations différentielles, comme la commande `dsolve` ou encore la fonction `odeplot` pour la visualisation des solutions (voir l'aide sur ces deux fonctionnalités).
Dans un premier temps, on définit l'équation différentielle. Il existe plusieurs manières de définir une dérivée. On utilisera de préférence les commandes `D` ou `diff`. La dérivée d'une fonction f(x) est alors : `diff(f,x)` ou `D(f)`. La dérivée de f en x=0 est `D(f)(0)`. La dérivée seconde s'écrit par exemple `diff(f,x,x)` ou encore `diff(f,x$2)` (d'autres formulations sont données dans l'aide).
L'équation différentielle est donnée ainsi :

```
ressort:={m*diff(x(t),t,t)+alpha*diff(x(t),t)+k*x(t)=0};
```

Puis les différents paramètres sont fixés : la masse m, le coefficient de frottement α et la raideur du ressort k :

```
m:=2:alpha:=0.5:k:=5:
```

On définit également les conditions initiales. A l'instant initial, le ressort a une longueur l (soit $x(0) = l$) et une vitesse nulle (soit $\frac{dx(0)}{dt} = 0$). De manière à changer plus facilement les conditions initiales, on peut définir une fonction CI telle que :

```
CI:=(a,b)->{x(0)=a, D(x)(0)=b};
```

On constatera que l'équation différentielle et ses conditions initiales sont définies comme des ensembles (entourés de {}), la commande `dsolve` nécessitant cette syntaxe.
Enfin on peut résoudre l'équation différentielle :

```
sol:=dsolve(ressort union CI(0.5,0), x(t), numeric);
```

L'option `numeric` indique que la solution de l'équation est approchée par une méthode numérique
Grâce à la commande `odeplot`, on peut représenter le mouvement du ressort dans le temps :

```
odeplot(sol, [t, x(t)], 0..10, numpoints=400);
```

et aussi modéliser l'état du système dans l'espace des phases :

```
odeplot(sol, [x(t), diff(x(t),t)], 0..10, numpoints=400);
```

Exercice 2 (Le pendule simple rigide)**.** *On souhaite simuler le mouvement d'un pendule simple. On rappelle que l'équation différentielle régissant le mouvement est :*

$$\frac{d^2}{dt^2}\theta(t) + 2f\frac{d}{dt}\theta(t) + w_0^2 \sin(\theta(t)) = 0$$

où w_0 est la pulsation propre du pendule et f le coefficient de frottement (on donnera $w_0 = 1$ et $f = 0.1$).
De plus on suppose qu'à l'instant initial, le pendule est forme un angle de $1.5rad$ avec l'axe vertical et que la vitesse angulaire vaut 2.
Comme précédemment, résoudre l'équation différentielle et donner une représentation graphique de la trajectoire et de l'état du système dans l'espace des phases.

Exercice 3 (Attracteurs de Lorenz)**.** *On considère un fluide chauffé (par exemple, de l'air au dessus d'un radiateur). L'air chaud étant moins dense que l'air froid, il monte. En montant, il se refroidit et redescend. Le mouvement du fluide s'organise autour de rouleaux dits de convection. D'après le modèle de Lorenz, ces rouleaux sont parallèles et tournent en sens inverse l'un de l'autre.*
La position $(x(t), y(t), z(t))$ d'un élément du fluide est décrite par le système différentiel suivant :

$$\begin{aligned}
\tfrac{d}{dt}x(t) &= s(y(t) - x(t)) \\
\tfrac{d}{dt}y(t) &= rx(t) - y(t) - x(t)z(t) \\
\tfrac{d}{dt}z(t) &= x(t)y(t) - bz(t)
\end{aligned}$$

où s, r et b dépendent du volume chauffé et du mode de chauffage (on choisit $s = 10$, $r = 28$, $b = \frac{8}{3}$).
On souhaite visualiser la trajectoire d'un élément. Pour ce faire :

1. *Utiliser le package* `DEtools`*, qui offre plus de fonctionnalités quant à la manipulation d'équations différentielles*

2. *Définir le système d'équations différentielles (comme une suite, et non plus comme un ensemble, i. e., on ne met pas les accolades) et stocker cette suite dans la variable* `Lorenz`

3. *Définir les conditions initiales (également comme une suite) en posant $x(0) = y(0) = z(0) = 1$ et stocker cette suite dans la variable* `CI`

4. *Utiliser la fonction* `DEplot3d` *pour représenter la trajectoire :*

```
DEplot3d({Lorenz},[x(t), y(t), z(t)], t=0..100, stepsize=0.01, [[CI]],
 orientation=[-35,75], linecolor=t, thickness=1);
```

1.10 Pour s'évaluer

1.10.1 Enoncé

1. Comment aller à la ligne sans exécuter ?
2. π avec 20 chiffres significatifs.
3. Commandes Maple pour affecter 2 à une variable a, puis libérer la variable a ; commande pour réinitialiser toutes les variables.
4. Définir la fonction $f(x) = \sin(2x)$ sous forme d'une fonction et sous forme d'une expression. Evaluer dans les 2 cas f en $x = 0.3$ et $x = 0.5$.
5. Définir une procédure somme qui additionne 2 nombres. Appeler cette somme pour calculer $3 + 5$.
6. Procédure pour calculer $\sum_{k=1}^{N} \frac{1}{k}$ et l'appeler pour $N = 10$.
7. Définir la liste des nombres $a + k(b - a)/N$, $k = 0, \ldots, N$ avec $a = 15$, $b = 20$ et $N = 50$.
8. Appliquer la fonction $f : x \to cos(\pi x)$ à la liste précédente.
9. On définit la suite $c_0 = 1$, $c_{n+1} = \sum_{k=0}^{n}(k + 1)c_k$, $n = 0, \ldots$ Calculer c_{100}.
10. Ecrire une fonction qui prend une liste en paramètre et renvoie le nombre d'éléments strictement positifs de la suite.

Chapitre 2

Interpolation

2.1 Introduction

Soit $f : [0,1] \to \mathbb{R}$ une fonction. On souhaite représenter cette fonction qu'avec un nombre fini de valeurs. Soit $N \in \mathbb{N}^*$, souvent une puissance de 2. On divise l'intervalle $[0,1]$ en N sous intervalles de même longueur

$$[x_k, x_{k+1}],\ k = 0, \ldots, N-1,\ x_k = k/N,\ k = 0, \ldots, N.$$

On dit que $x_k, k = 0, \ldots, N$ est la **grille uniforme** de l'intervalle $[0,1]$ de **pas de discrétisation** $h = 1/N$. On dit aussi que x_k est un **point de la grille**. On suppose connues $N+1$ valeurs

$$f_0, \ldots, f_{N-1}, f_N,$$

qui sont censées être des approximations de la fonction f en les points x_k :

$$f_k \simeq f(x_k).$$

Soit maintenant $x \in [0,1]$, on cherche à trouver une valeur approchée de $f(x)$ à partir de $f_0, f_1, \ldots, f_N$ et x. Cela se traduit d'un point de vue informatique par choisir une procédure

$$\textbf{interpol}([f_0, \ldots, f_N], x),$$

qui prend la liste de réels $f_0, \ldots, f_N$ et le réel x en paramètres. Cette procédure est censée renvoyer un réel qui approche $f(x)$. On va définir ici plusieurs possibilités pour cette procédure `interpol`, qui doit satisfaire les **conditions d'interpolation**

$$\textbf{interpol}([f_0, \ldots, f_N], k/N) = f_k,\ k = 0, \ldots, N.$$

2.2 Interpolation linéaire

2.2.1 Définition

L'interpolation la plus simple est de relier les points (x_k, f_k) par un segment de droite. La formule est donnée par

$$\textbf{interpolin}([f_0, \ldots, f_N], x) = f_k + \frac{f_{k+1} - f_k}{h}(x - x_k),\ x_k \leq x < x_{k+1}.$$

2.2.2 Algorithme

D'un point de vue algorithmique, on doit d'abord chercher l'indice k, comme x est dans un unique intervalle $[x_k, x_{k+1}]$. Pour cela, on prend la partie entière de $x * N$ et la position à l'intérieur de la maille est donnée par $\frac{x-x_k}{h}$ qui se simplifie $x * N - k$.

Voici une implémentation en Maple de cette procédure.

```
interpolin:=proc(fX,x)
  N:=nops(fX)-1:
  if(x=1)then return evalf(fX[N+1]);fi:
  k:=floor(x*N):
  alpha:=x*N-k;
  evalf(fX[k+1]+alpha*(fX[k+2]-fX[k+1]));
end proc:
```

Supposons que l'on ait les données suivantes :

x	0.	0.25	0.5	0.75	1
$f(x)$	1	2	2.5	2	1

et on cherche une approximation de $f(0.3)$. On peut alors écrire en Maple

```
interpolin([1,2,2.5,2,1],0.3);
```

qui fournit une approximation donnée par l'interpolation linéaire.

2.2.3 Exercices

Exercice 4. *On relève les valeurs suivantes de la viscosité cinématique ν en fonction de la température. Les résultats sont donnés dans la Table 2.1. On souhaite maintenant avoir une estimation de la viscosité pour $T = 23$. Donner un résultat possible en utilisant l'interpolation linéaire.*

T	10	15	20	25	30	35	40
ν	1.308	1.141	1.005	0.896	0.804	0.727	0.661

TABLE 2.1 – Evolution de la viscosité en fonction de la température

Exercice 5. *Modifier la procédure* `interpolin(fX,x)`*, en une procédure* `interpolin2(fX,x,a,b)` *pour que l'on puisse traiter le cas d'un intervalle $[a, b]$ au lieu de l'intervalle $[0, 1]$.*

Exercice 6. *Quelle est la valeur de k et α pour l'exemple donné dans la sous-section algorithme ?*

Exercice 7. *Donner l'instruction Maple adéquate pour retrouver la valeur de la viscosité en $T = 23$ en utilisant* `interpolin2`*.*

Exercice 8. *On se donne maintenant la table suivante (Table 2.2). Comment peut-on trouver la vitesse en $t = 16$ en faisant un calcul à la main ? Est-ce que*

```
interpolin2([0,227.04,362.78,512.35,602.97,901.67],16,0,30)}
```

t en s	0	10	15	20	22.5	30
$v(t)$ en m/s	0	227.04	362.78	512.35	602.97	901.67

TABLE 2.2 – Evolution de la vitesse d'un objet en fonction du temps

donne la même valeur ? Modifier les paramètres d'`interpolin2` *pour obtenir la bonne valeur.*

Exercice 9. *Ecrire une procédure de localisation* `localisation(X,x)`*, qui prend en paramètre une liste de points rangés dans l'ordre croissant :*

$$X = [x_0, x_1, \ldots, x_N],\ x_0 < x_1 \cdots < x_N$$

et qui renvoie l'unique indice i tel que $x_i \leq x < x_{i+1}$. On supposera que x est dans l'intervalle $[x_0, x_N]$ (Si $x = x_N$, la procédure doit renvoyer N).

Exercice 10. *Utiliser l'exercice précédent pour effectuer une interpolation dans le cas où les intervalles $[x_k, x_{k+1}]$ ne sont plus forcément de même taille. On écrira une procédure* `interpolin3(X,fX,x)`*. Vérifier le résultat de l'exercice 8 en utilisant* `interpolin3`*.*

2.3 Interpolation cubique

2.3.1 Interpolation sur un intervalle élémentaire

Supposons que l'on ait des données

$$f(-1),\ f(0),\ f(1),\ f(2).$$

On souhaite avoir une approximation de $f(\alpha)$ pour $0 \leq \alpha \leq 1$. On a vu que l'interpolation linéaire peut fournir un résultat. Néanmoins, cette approximation est parfois trop grossière ; en particulier, on n'utilise pas l'information que l'on connait $f(-1)$ et $f(2)$.
Notons que si f est un polynôme de degré ≤ 1 (i.e $f(x) = ax + b$) alors l'interpolation linéaire fournit la valeur exacte de f. On peut généraliser cette technique en cherchant un polynôme de degré ≤ 3 (i. e. $P(x) = a + bx + cx^2 + dx^3$) satisfaisant $P(-1) = f(-1),\ P(0) = f(0),\ P(1) = f(1)$ et $P(2) = f(2)$.
Comme on connait f en les points $-1, 0, 1, 2$ on a donc 4 équations pour 4 inconnues :

$$\begin{aligned} a - b + c - d &= f(-1) \\ a &= f(0) \\ a + b + c + d &= f(1) \\ a + 2b + 4c + 8d &= f(2). \end{aligned}$$

On peut alors vérifier que P s'écrit sous la forme

$$P(\alpha) = f(-1)w_{-1}(\alpha) + f(0)w_0(\alpha) + f(1)w_1(\alpha) + f(2)w_2(\alpha),$$

où les w_i sont des polynômes de degré ≤ 3, vérifiant :

$$w_{-1}(-1) = 1,\ w_{-1}(0) = 0,\ w_{-1}(1) = 0,\ w_{-1}(2) = 0 \tag{2.1}$$
$$w_0(-1) = 0,\ w_0(0) = 1,\ w_0(1) = 0,\ w_0(2) = 0 \tag{2.2}$$
$$w_1(-1) = 0,\ w_1(0) = 0,\ w_1(1) = 1,\ w_1(2) = 0 \tag{2.3}$$
$$w_2(-1) = 0,\ w_2(0) = 0,\ w_2(1) = 0,\ w_2(2) = 1. \tag{2.4}$$

Les w_i ont en fait une forme explicite que l'on peut retrouver facilement : w_{-1} est un polynôme de degré ≤ 3 qui admet $0, 1$ et 2 comme racines et qui vaut 1 en -1. On peut donc écrire :

$$\tilde{w}_{-1}(x) = x(x-1)(x-2),\ w_{-1}(x) = \tilde{w}_{-1}(x)/\tilde{w}_{-1}(-1).$$

En résumé, on a

$$f(\alpha) \simeq f(-1)w_{-1}(\alpha) + f(0)w_0(\alpha) + f(1)w_1(\alpha) + f(2)w_2(\alpha),$$

avec

$$w_{-1}(\alpha) = -\frac{\alpha(1-\alpha)(2-\alpha)}{6} \tag{2.5}$$

$$w_0(\alpha) = \frac{(\alpha+1)(1-\alpha)(2-\alpha)}{2} \tag{2.6}$$

$$w_1(\alpha) = \frac{(\alpha+1)\alpha(2-\alpha)}{2} \tag{2.7}$$

$$w_2(\alpha) = -\frac{(\alpha+1)\alpha(1-\alpha)}{6}. \tag{2.8}$$

2.3.2 Interpolation sur un maillage uniforme

On suppose cette fois-ci que l'on a un maillage uniforme de l'intervalle $[0, 1]$, comme dans l'introduction. On dispose donc de valeurs

$$f_k \simeq f(x_k),\ k = 0, \ldots, N,\ x_k = kh,\ h = 1/N.$$

On suppose en outre connaitre f_{-1} et et f_{N+1}. L'approximation cubique sur l'intervalle $[x_k, x_{k+1}[$ (pour $k = 0, \ldots, N$) est alors donnée par

$$f_k^{[-1,2]}(\alpha) = f_{k-1}w_{-1}(\alpha) + f_k w_0(\alpha) + f_{k+1}w_1(\alpha) + f_{k+2}w_2(\alpha),$$

avec $x = x_k + \alpha h \in [x_k, x_{k+1}[$ (c'est-à-dire $\alpha \in [0, 1[$).

2.3.3 Algorithme

L'algorithme est similaire au cas de l'interpolation linéaire : on cherche d'abord dans quel intervalle $[x_k, x_{k+1}]$ le point x se situe ; on calcule $\alpha = (x - x_k)/h$ puis une utilise une (autre) formule qui fait intervenir cett fois-ci $f_{k-1}, f_k, f_{k+1}, f_{k+2}$, au lieu de seulement f_k et f_{k+1}
Voici une implémentation Maple de la procédure d'interpolation cubique. On suppose la connaissance de $[f_{-1}, f_0, \ldots, f_N, f_{N+1}]$ qui est stockée dans un tableau `fX`.

```
interpolcub:=proc(fX,x)
  N:=nops(fX)-3:
  if(x=1)then return evalf(fX[N+3]);fi:
  k:=floor(x*N):
  a:=x*N-k;
  s:=(a+1)*(2-a)/2*((1-a)*fX[k+2]+a*fX[k+3]);
  s:=s-a*(1-a)/6*((2-a)*fX[k+1]+(a+1)*fX[k+4]);
  evalf(s);
end proc:
```

2.3.4 Cas d'un intervalle quelconque

Soit g une fonction définie sur un intervalle $[a, b]$. On peut alors définir f sur l'intervalle $[0, 1]$ par

$$f(t) = g(a + t(b - a)).$$

Pour connaître g en un point $x \in [a, b]$, on calcule f en un point $t = \frac{x-a}{b-a} \in [0, 1]$.

2.3.5 Le bord du domaine

L'interpolation cubique nécessite de donner une valeur à f_{-1}.

Condition limite périodique. Si la fonction est périodique de période 1, on prend $f_{-1} = f_{N-1}$.

Condition limite nulle. Si la fonction est supposée nulle en dehors de l'intervalle $[0, 1]$, on prend $f_{-1} = 0$.

Reconstruction décentrée. Si on n'a pas d'information supplémentaire de la fonction en dehors de l'intervalle $[0, 1]$, on peut considérer une reconstruction décentrée qui ne fait pas intervenir le point f_{-1}. Pour cela, pour $x \in [x_0, x_1]$, on utilise $f_0^{[0,3]}(\alpha)$, au lieu de $f_0^{[-1,2]}(\alpha)$.

2.3.6 Exercices

Exercice 11. *Vérifier les polynômes donnés explicitement par (2.5)-(2.8) satisfont bien (2.1)-(2.4).*

Exercice 12. *Expliciter le terme* $f_0^{[0,3]}(\alpha)$.

Exercice 13. *Modifier la procédure* `interpolcub(fX,x)`*, en une procédure* `interpolcub2(fX,x,a,b)` *pour que l'on puisse traiter le cas d'un intervalle* $[a, b]$ *au lieu de l'intervalle* $[0, 1]$.

Exercice 14. *Reprendre les exercices 8 et 10 dans le cas de l'interpolation cubique.*

2.4 Résultats numériques

On considère comme cas-test, une fonction sinus :

$$f(x) = \sin(2\pi x),$$

une fonction gaussienne

$$f(x) = \exp(-100(x - 1/2)^2)$$

et une fonction créneau

$$f(x) = 1, \text{ si } 0.2 < x < 0.7, \ f(x) = 0, \text{ sinon}$$

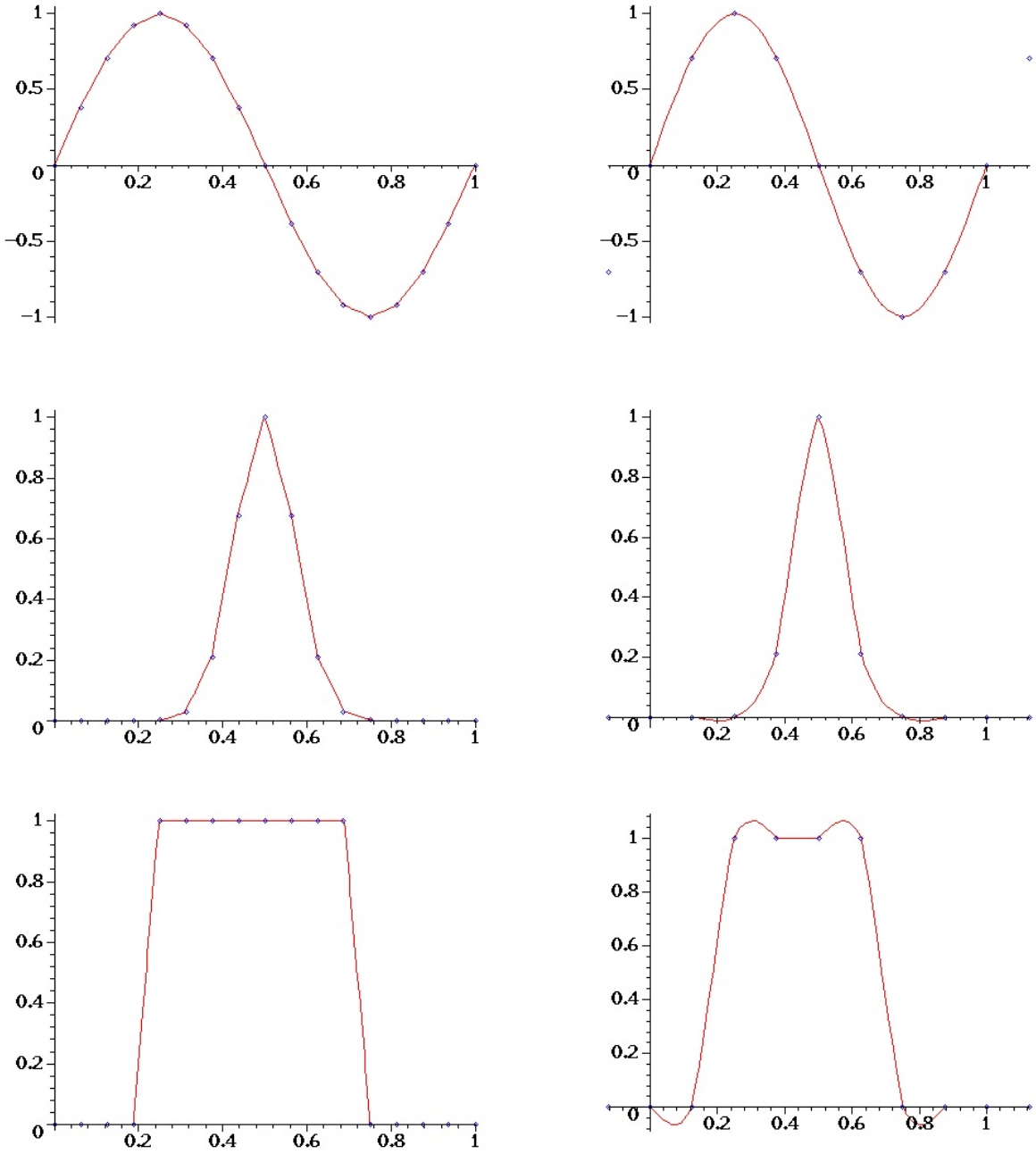

FIGURE 2.1 – Interpolation linéaire (à gauche, $N = 16$) et cubique (à droite, $N = 8$)

Chapitre 3

Intégration numérique

3.1 Introduction

Soit $f : [0,1] \to \mathbb{R}$ une fonction. On cherche une approximation de $\int_0^1 f(x)dx$. Soit $N \in \mathbb{N}^*$. On suppose connues des valeurs

$$f_k \simeq f(x_k),\ x_k = kh, h = 1/N, k = 0, \ldots, N,$$

et on cherche une approximation de $\int_{x_k}^{x_{k+1}} f(x)dx,\ k = 0, \ldots, N-1$ sous la forme

$$\sum_j \omega_j f_{k+j}.$$

Notons que l'on a alors l'approximation de l'intégrale sur l'intervalle $[0,1]$ par la formule

$$\int_0^1 f(x)dx = \sum_{k=0}^{N-1} \int_{x_k}^{x_{k+1}} f(x)dx.$$

3.2 Obtention des formules

Formule des rectangles Pour la formule des rectangles à gauche, on prend

$$\int_{x_k}^{x_{k+1}} f(x)dx \simeq hf_k,$$

tandis que pour la formule des rectangles à droite, on prend

$$\int_{x_k}^{x_{k+1}} f(x)dx \simeq hf_{k+1},$$

ce qui donne respectivement les formules

$$\int_0^1 f(x)dx \simeq h\sum_{k=0}^{N-1} f_k,$$

et

$$\int_0^1 f(x)dx \simeq h\sum_{k=1}^{N} f_k.$$

Remarquons que la formule des rectangles est exacte si f est une constante.

Formule des trapèzes On prend cette fois-ci

$$\int_{x_k}^{x_{k+1}} f(x)dx \simeq h\frac{f_k + f_{k+1}}{2},$$

ce qui donne

$$\int_0^1 f(x)dx \simeq h\frac{f_0 + f_N}{2} + h\sum_{k=1}^{N-1} f_k.$$

La formule des trapèzes est exacte pour f polynôme de degré ≤ 1.

Formule de Simpson On suppose que N est pair. On écrit alors

$$\int_0^1 f(x)dx = \sum_{k=0}^{N/2-1} \int_{x_{2k}}^{x_{2k+2}} f(x)dx,$$

et on cherche à approcher

$$\int_{x_{2k}}^{x_{2k+2}} f(x)dx \simeq \omega_0 f_{2k} + \omega_1 f_{2k+1} + \omega_2 f_{2k+2},$$

en choisissant $\omega_0, \omega_1, \omega_2$ de telle sorte que l'approximation soit exacte pour f polynôme de degré le plus grand possible.
En prenant $f = 1$, on obtient

$$\omega_0 + \omega_1 + \omega_2 = 2h$$

En prenant $f(x) = (x - x_{2k+1})$, on obtient

$$\omega_0 = \omega_2.$$

Enfin, en prenant $f(x) = (x - x_{2k})^2$, on obtient

$$8h^3/3 = (\omega_1 + 4\omega_2)h^2.$$

On a alors $2\omega_2 = \frac{2h}{3}$, puis $\omega_0 = \omega_2 = \frac{h}{3}$ et $\omega_1 = \frac{4h}{3}$, et la formule reste exacte pour tout polynôme de degré ≤ 3.

3.3 Changement d'intervalle

Si on veut calculer $\int_a^b g(x)dx$, on définit $f(t) = g(a + t(b - a))$. On a alors

$$\int_a^b g(x)dx = (b - a)\int_0^1 g(a + t(b - a))dt.$$

3.4 Exercices

Exercice 15. *Ecrire une procédure* `intrap(f,a,b,N)` *qui renvoie une approximation de l'intégrale de f (donnée sous forme d'expression de la variable x) sur l'intervalle $[a, b]$, en utilisant la méthode des trapèzes.*

Exercice 16. *Même exercice, mais en utilisant la méthode de Simpson*

Exercice 17. *Même exercice, mais en utilisant la méthode des rectangles à droite.*

Exercice 18. *On considère un tonneau de vin (cf Figure 3.4) d'hauteur* 90 *cm, de petit diamètre* $55cm$ *et de grand diamètre* 70 *cm. Approcher l'aire du tonneau en utilisant la formule de Simpson.*

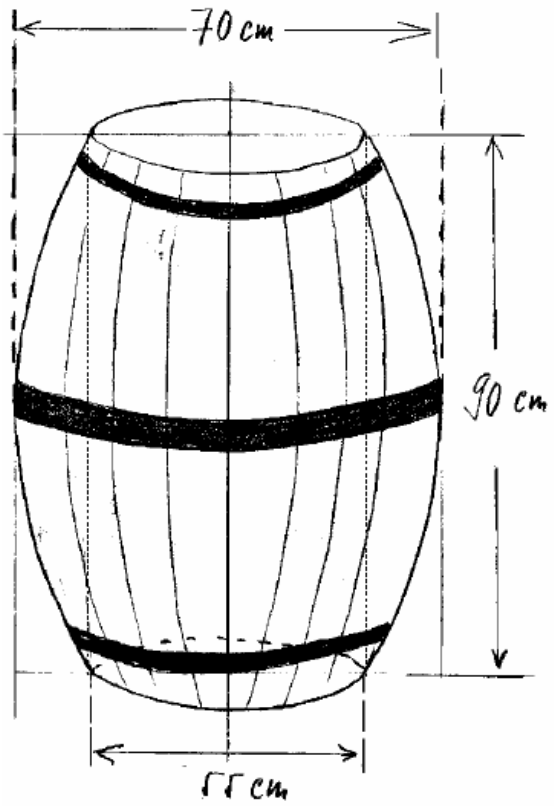

FIGURE 3.1 – Un tonneau de vin

Exercice 19. *Utiliser la formule de Simpson pour calculer une approximation de* $\int_0^{\pi} \frac{1}{1+\sin(x)} dx$, *en utilisant* 6 *intervalles ($N = 6$). On utilisera la 3.1.*

Exercice 20. *Le voltage d'un composant (exprimé en mV) en des intervalles réguliers de* $0.01s$ *sur un demi-cycle est donné par* $0, 19.5, 35, 45, 40.5, 25, 20.5, 29, 27, 12.5$ *et* 0. *Calculer*

x	0	$\frac{\pi}{6}$	$\frac{\pi}{3}$	$\frac{\pi}{2}$	$\frac{2\pi}{3}$	$\frac{5\pi}{6}$	π
$\frac{1}{1+\sin(x)}$	1	0.6667	0.5359	0.5	0.5359	0.6667	1

TABLE 3.1 – la fonction $\frac{1}{1+\sin(x)}$

la valeur rms du voltage sur un demi-cycle donné par la formule

$$V_{rms} = \sqrt{\frac{1}{b-a}\int_a^b V^2 dt}.$$

Exercice 21. *Utiliser la formule de Simpson pour évaluer*

$$\int_{0.1}^{1.3} 5x\exp(-2x)dx.$$

Comparer avec la valeur exacte et calculer l'erreur relative : $\frac{I_{\text{calc}}-I_{\text{exact}}}{I_{\text{exact}}}$.

Exercice 22. *Même exercice, mais en utilisant la formule des trapèzes.*

Exercice 23. *Calculer une approximation de* $\int_{0.05}^{1} f(x)dx$*, en utilisant la Table 3.2*

x	0.05	0.1	0.15	0.2	0.3	0.4
$f(x)$	0.0785	0.1564	0.2334	0.3090	0.4540	0.5878
x	0.5	0.6	0.7	0.8	0.9	1
$f(x)$	0.7071	0.8090	0.8910	0.9511	0.8977	1.0000

TABLE 3.2 – Une fonction tabulée

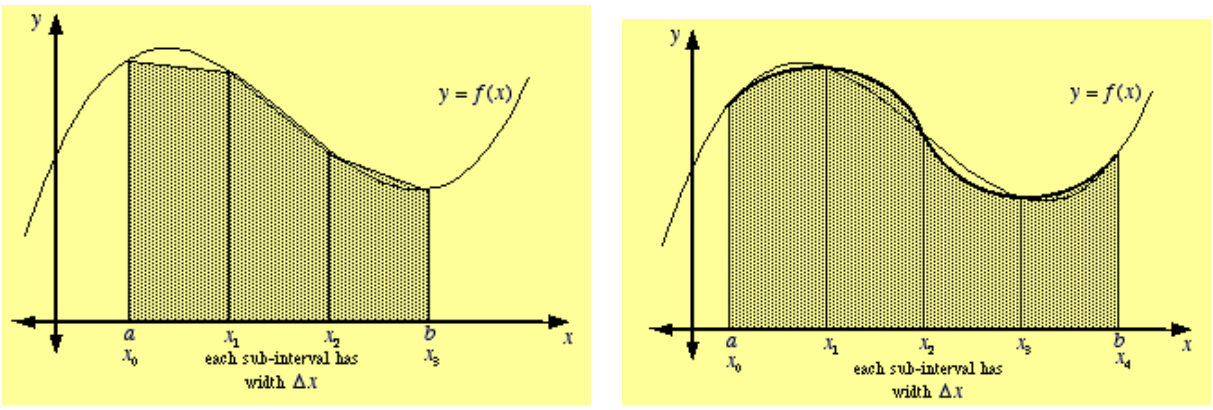

FIGURE 3.2 – Méthode des trapèzes (à gauche) et de Simpson (à droite)

Chapitre 4

Recherche de zéros

Soit $f : \mathbb{R} \to \mathbb{R}$ une fonction que l'on suppose continue. On cherche à trouver x tel que

$$f(x) = 0.$$

On dit que x est un zéro de f.

4.1 Une première localisation

On considère un intervalle $[a, b]$ et on cherche le ou les zéros de f sur l'intervalle $[a, b]$. Pour cela on fixe $N \in \mathbb{N}^*$ et on considère une discrétisation de l'intervalle $[a, b]$ en N sous intervalles $[a + k\frac{b-a}{N}, a + (k + 1)\frac{b-a}{N}], k = 0, \ldots, N - 1$. Il y aura alors au moins un zéro sur chaque sous intervalle où f change de signe, ce qui se traduit par le test

$$f(a + k\frac{b-a}{N})f(a + (k + 1)\frac{b-a}{N}) < 0.$$

Si N est suffisamment grand, souvent f n'admet alors qu'un zéro dans cet intervalle. Cela n'est pas vrai si f s'annule sur tout un intervalle ou si on a une accumulation de zéros autour d'un point. Dans ce cas, on aura fait une première localisation des zéros de f où f change de signe autour du zéro en question.
Notons que si f est continûment dérivable et si $f'(x_0) \neq 0$ pour x_0 zéro de f, alors f change de signe dans un voisinage suffisamment petit de x_0. On se placera donc dans ce cas pour la suite.
On se fixe donc un sous intervalle que l'on note de nouveau $[a, b]$ et on suppose que $f(a)f(b) < 0$. Comme f est continue, on sait qu'il existe $x^* \in]a, b[$ tel que $f(x) = 0$. On suppose que ce x^* est unique.

4.2 La méthode de la bisection

On l'appelle aussi dichotomie ; elle part du principe que l'on coupe chaque fois un intervalle en 2 et on regarde dans lequel des intervalles le zéro se trouve.

Construction de la méthode On a un intervalle de départ $[a_0, b_0]$ tel que $f(a_0)f(b_0) < 0$. On initialise à l'étape 0 a à a_0, b à b_0. On calcule alors $c = \frac{a+b}{2}$. A l'étape $n \geq 1$, Il y a alors 3 possibilités

- Si $f(a)f(c) < 0$, le zéro se trouve sur l'intervalle $[a, c]$ et on remplace b par c.
- Si $f(c)f(b) < 0$, le zéro se trouve sur l'intervalle $[c, b]$ et on remplace a par c.
- Si $f(c) = 0$, on a trouvé le zéro.

On redéfinit c à $\frac{a+b}{2}$ et on passe à l'étape $n + 1$.

Implémentation Voici un programme Maple qui donne une implémentation de la méthode de la bisection.

```
dicho:=proc(f,aa,bb,N)
a:=evalf(aa):b:=evalf(bb):
Xa:=array(1..N):
Xb:=array(1..N):
Xa[1]:=a:
Xb[1]:=b:
c:=(a+b)/2:
Xc[1]:=c:
fa:=evalf(f(a)):fb:=evalf(f(b)):
for i from 2 to N do
fc:=evalf(f(c)):
if evalf(fa*fc)<0. then b:=c:fb:=evalf(f(b)):
elif evalf(fc*fb)<0. then a:=c:fa:=evalf(f(a)):
end:
Xa[i]:=a:Xb[i]:=b:
c:=(a+b)/2:
Xc[i]:=c:
od:
[[seq(Xa[i],i=1..N)],[seq(Xb[i],i=1..N)],[seq(Xc[i],i=1..N)]]:
end proc:
```

4.3 Corde, Sécante et Newton

On cherche des méthodes plus rapides que la méthode de la dichotomie. Pour cela, on va utiliser des informations données par f, voire même par f'. On part du DL :

$$f(x^*) = 0 = f(x) + (x^* - x)f'(\xi),$$

de sorte que x^* est donné par

$$x^* = x - \frac{f(x)}{f'(\xi)}.$$

On va donc construire une suite $x^{(k)}$ censée tendre vers x^*, définie par

$$x^{(k+1)} = x^{(k)} - \frac{f(x^{(k)})}{q_k},$$

où q_k est une pente censée approcher $f'(\xi)$. On obtient donc

$$x^{(k+1)} = x^{(k)} + \frac{(x^* - x^{(k)})f'(\xi)}{q_k},$$

ce qui donne l'estimation

$$x^{(k+1)} - x^* = (1 - \frac{f'(\xi)}{q_k})(x^{(k)} - x^*).$$

On considère un intervalle initial $[a, b]$.

4.3.1 La méthode de la corde

Construction La méthode de la corde consiste à se donner une valeur de départ x_0 et à prendre

$$q_k = \frac{f(b) - f(a)}{b - a},$$

ce qui donne

$$x^{(k+1)} = x^{(k)} - \frac{(b-a)f(x^{(k)})}{f(b) - f(a)}.$$

Implémentation Voici un programme Maple qui donne une implémentation de la méthode de la corde. Pour l'appel, on prendra $x_0 = \frac{a+b}{2}$.

```
>chord:=proc(f,aa,bb,x0,N)
a:=evalf(aa):b:=evalf(bb):
X:=array(1..N):
X[1]:=x0:
fa:=evalf(f(a)):fb:=evalf(f(b)):
q:=(b-a)/(fb-fa):
for i from 2 to N do
X[i]:=X[i-1]-q*evalf(f(X[i-1])):
od:
[seq(X[i],i=1..N)]:
end proc:
>L:=chord(f,xmin,xmax,0.5*(xmin+xmax),N):
```

4.3.2 La méthode de la sécante

Construction La méthode de la sécante consiste à se donner deux valeurs de départs x_{-1} et x_0 et à prendre

$$q_k = \frac{f(x^{(k)}) - f(x^{(k-1)})}{x^{(k)} - x^{(k-1)}},$$

ce qui donne

$$x^{(k+1)} = x^{(k)} - \frac{(x^{(k)} - x^{(k-1)})f(x^{(k)})}{f(x^{(k)}) - f(x^{(k-1)})}.$$

4.3.3 Implémentation

Voici un programme Maple qui donne une implémentation de la méthode de la sécante. Pour l'appel, on prendra $x_{-1} = a, x_0 = b$.

```
>secant:=proc(f,aa,bb,N)
a:=evalf(aa):b:=evalf(bb):
X:=array(1..N):
X[1]:=a:X[2]:=b:f1:=evalf(f(a)):f2:=evalf(f(b)):
for i from 3 to N do
X[i]:=X[i-1]-(X[i-1]-X[i-2])/(f2-f1)*f2;
f1:=f2:f2:=evalf(f(X[i])):
od:
[seq(X[i],i=1..N)]:
end proc:
>L:=secant(f,xmin,xmax,N):
```

4.3.4 La méthode de Newton

Construction On prend cette fois-ci $q_k = f'(x^{(k)})$, ce qui donne

$$x^{(k+1)} = x^{(k)} - \frac{f(x^{(k)})}{f'(x^{(k)})}.$$

Implémentation Voici un programme Maple qui donne une implémentation de la méthode de Newton. Pour l'appel, on prendra $x_0 = \frac{a+b}{2}$.

```
>newton:=proc(f,df,x0,N)
x:=evalf(x0):X[1]:=x:
for i from 2 to N do
fx:=evalf(f(x)):
dfx:=evalf(df(x)):
x:=x-fx/dfx:
X[i]:=x:
od:
[seq(X[i],i=1..N)]:
end proc:
>L:=newton(f,df,0.5*(xmax+xmin),N):
```

4.3.5 Autres méthodes

Il est aussi possible de combiner une méthode convergente mais lente (comme la méthode de la bisection), avec une méthode rapide, mais qui ne converge pas toujours. Ainsi on peut par exemple choisir un intervalle de départ $[a, b]$ puis appliquer une itération de Newton. Si la valeur sort de l'intervalle $[a, b]$, on fait une itération de la méthode de bisection ce qui définit un nouvel intervalle et on recommence.

4.4 Exercices

Exercice 24. *Utiliser la méthode de Newton pour calculer* $\sqrt{3}$*, en remarquant que* $\sqrt{3}$ *est la racine positive de* $f(x) = x^2 - 3$.

Exercice 25. *Trouver les* 3 *points où le graphe de* $y = 3 - x^2$ *intersecte le graphe de* $y = 1/x$.

Exercice 26. *La longueur d'un arc circulaire entre deux points P et Q est de $2.1cm$, tandis que la distance en ligne droite entre P et Q vaut $2cm$. Trouver le rayon de l'arc de cercle.*

Exercice 27. *Trouver la valeur minimale du polynôme $f(x) = x^6 - x^2 - 40x$.*

Exercice 28. *On considère une planète en orbite autour du soleil (voir Figure 4.1). Soit ω la fréquence de cette orbite, t le temps écoulé depuis que la planète était la plus proche du soleil (perifocus) et $e = \sqrt{1 - \frac{b^2}{a^2}}$ l'excentricité de l'orbite elliptique de la planète. Alors d'après les lois de Kepler du mouvement planétaire, la position de la planète à l'instant t est donnée par*

$$x(t) = a(\cos(E(t)) - e),\ y(t) = a\sqrt{1 - e^2}\sin(E(t)), \tag{4.1}$$

où $E(t)$ est appelé l'anomalie excentrique et est donnée par

$$E(t) = \omega t + e\sin(E(t)).$$

Prenons le cas de la Terre : $e \simeq 0.0167$, $\omega \simeq \frac{2\pi}{365.25635\ jours}$, $a \simeq 149.6 \cdot 10^6 km$. Calculer $E(t)$, $x(t)$ et $y(t)$ dans le cas où $t = 91$ jours, $t = 182$ jours et $t = 273$ jours.

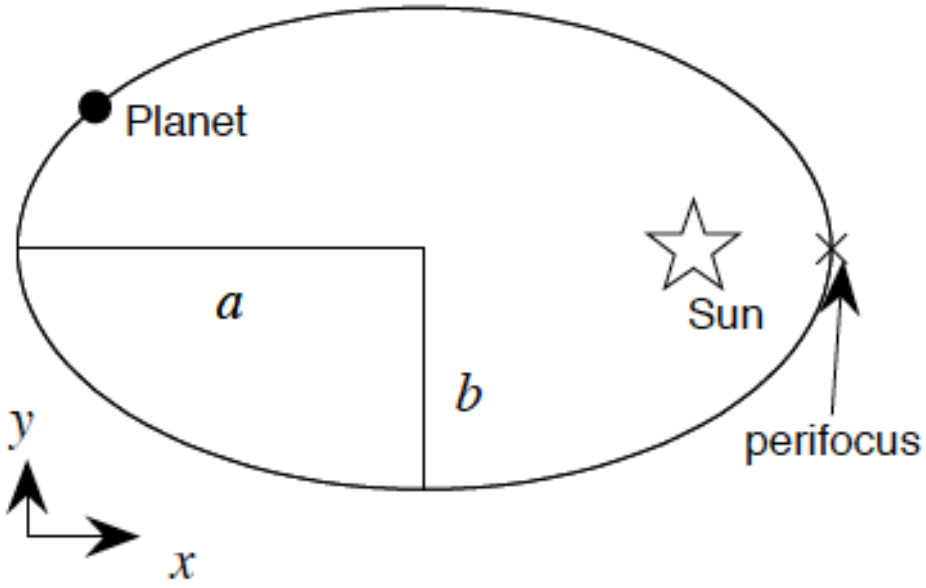

FIGURE 4.1 – Orbite d'une planète

Exercice 29. *On considère une courbe fermée $M(t) = (x(t), y(t))$ et un point $M(t_0)$ de cette courbe. On suppose que deux personnes partent de ce point et se déplacent en sens opposé et à vitesse constante. En quel point se retrouvent-elles ? On aura besoin de calculer la distance parcourue entre deux points $M(t_1)$ et $M(t_2)$ de la courbe, qui est donnée par*

$$\int_{t_1}^{t_2} \sqrt{|x'(\sigma)|^2 + |y'(\sigma)|^2} d\sigma.$$

On prendra les courbes suivantes comme exemple :

1. *Le coeur (en coordonnées polaires)* $\rho = |\tan(\theta)|^{1/|\tan(\theta)|}$ *(Figure 4.2)*
2. *L'oeuf* $x(t) = a/(1+t^2)^2,\ y(t) = tx$ *(Figure 4.3)*
3. *La goutte d'eau* $x(t) = a\cos^2(t),\ y(t) = a^2\cos^3(t)\sin(t)/b$ *(Figure 4.4)*

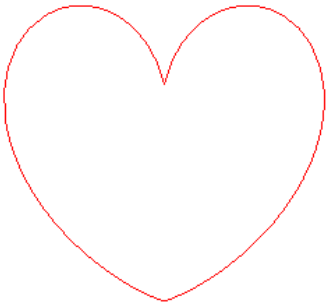

FIGURE 4.2 – Le coeur

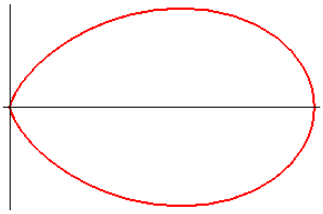

FIGURE 4.3 – L'oeuf

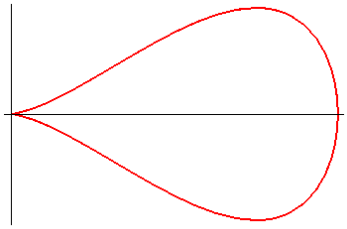

FIGURE 4.4 – La goutte d'eau

Chapitre 5

Equations différentielles

On cherche à trouver la solution $y(t)$ de l'équation

$$y'(t) = f(t, y(t)),\ y(t_0) = y_0 \tag{5.1}$$

Pour cela, on se fixe une discrétisation de l'intervalle $[t_0, t_0 + T]$:

$$t_n = t_0 + nh,\ n = 0, \ldots, N,\ N \in \mathbb{N}^*,\ h = (T - t_0)/N,$$

et on cherche alors des valeurs $y_n \simeq y(t_n),\ n = 1, \ldots, N$.

5.1 La méthode d'Euler

On approche $y'(t_n) \simeq \frac{y_{n+1} - y_n}{h}$. La suite y_n est alors définie par

$$y_{n+1} = y_n + hf(t_n, y_n),\ n = 0, \ldots, N - 1.$$

Il s'agit de la méthode d'Euler explicite.
Remarquons qu'en intégrant (5.1) entre t_n et t_{n+1}, on obtient

$$y(t_{n+1}) - y(t_n) = \int_{t_n}^{t_{n+1}} f(t, y(t))dt.$$

La méthode d'Euler explicite revient alors à approcher l'intégrale de droite par la formule des rectangles à gauche.
La méthode d'Euler implicite correspond à prendre la formule des rectangles à droite. :

$$y_{n+1} = y_n + hf(t_{n+1}, y_{n+1}),\ n = 0, \ldots, N - 1.$$

Attention, dans ce cas, y_{n+1} est défini de manière implicite en fonction de y_n. On pourra alors utiliser une méthode du chapitre "Recherche de zéros", pour trouver y_{n+1}.

5.2 Le schéma de Runge explicite

On considère cette fois-ci la formule du point milieu :

$$\int_{t_n}^{t_{n+1}} f(t, y(t))dt \simeq hf(t_n + h/2, y(t_n + h/2)),$$

et on approche $y(t_n + h/2)$ par Euler explicite ce qui donne au final

$$k_1 = f(t_n, y_n),\ k_2 = f(t_n + h/2, y_n + h/2k_1),\ y_{n+1} = y_n + hk_2.$$

5.3 La méthode de Runge-Kutta d'ordre 4 (RK4)

Cette méthode est d'usage très fréquent et est donnée par

$$y_{n+1} = y_n + h\frac{k_1 + 2k_2 + 2k_3 + k_4}{6},$$

avec

$k_1 = f(t_n, y_n),\ k_2 = f(t_n+h/2, y_n+h/2k_1), k_3 = f(t_n+h/2, y_n+h/2k_2),\ k_4 = f(t_{n+1}, y_n+hk_3)$

5.4 Implémentation

La méthode d'Euler explicite peut être implémentée de la manière suivante

```
euler2:=proc(f,y0,t0,T,N)
  y:=array(0..N):
  y[0]:=y0:
  h:=evalf(T-t0)/N:
  for n from 0 to N-1 do
    y[n+1]:=y[n]+h*f(t0+n*h,y[n]):
  od:
  y;
end proc:
```

Pour la méthode de Runge, on peut écrire

```
runge:=proc(f,y0,t0,T,N)
  y:=array(0..N):
  y[0]:=y0:
  h:=evalf(T-t0)/N:
  for n from 0 to N-1 do
    k1:=f(t0+n*h,y[n]):
    k2:=f(t0+(n+1/2)*h,y[n]+h/2*k1):
    y[n+1]:=y[n]+h*k2:
  od:
  y;
end proc:
```

Enfin la méthode de Runge Kutta d'ordre 4 peut s'écrire

```
rk4:=proc(f,y0,t0,T,N)
  y:=array(0..N):
  y[0]:=y0:
  h:=evalf(T-t0)/N:
  for n from 0 to N-1 do
    k1:=f(t0+n*h,y[n]):
    k2:=f(t0+(n+1/2)*h,y[n]+h/2*k1):
    k3:=f(t0+(n+1/2)*h,y[n]+h/2*k2):
    k4:=f(t0+(n+1)*h,y[n]+h*k3):
```

```
    y[n+1]:=y[n]+h*(k1+2*k2+2*k3+k4)/6:
  od:
  y;
end proc:
```

5.5 Le cas de systèmes d'équations différentielles

Prenons par exemple le système de Lotka-Volterra :

$$x'(t) = x(t)(\alpha - \beta y(t)),\ y'(t) = -y(t)(\gamma - \delta x(t)). \tag{5.2}$$

$x(t)$ représente l'effectif des proies, $y(t)$ l'effectif des prédateurs, et α, β, γ et δ sont des paramètres qui désignent respectivement : le taux de reproduction des proies en l'absence de prédateurs, le taux de mortalité des proies due aux prédateurs, le taux de reproduction des prédateurs en fonction des proies mangées, et le taux de mortalité des prédateurs en l'absence de proies.
En introduisant $z(t) = (x(t), y(t))$, on a

$$z'(t) = F(t, z(t)),\ F(t, x, y) = (x(\alpha - \beta y), -y(\gamma - \delta x)),$$

et on peut donc appliquer les algorithmes précédents. Pour Euler explicite par exemple, on peut écrire

$$x_{n+1} = x_n + hx_n(\alpha - \beta y_n),\ y_{n+1} = y_n - hy_n(\gamma - \delta x_n),\ n = 0, \ldots, N-1.$$

5.6 Le cas de systèmes d'ordre 2 (ou plus)

Si l'on a maintenant un système d'ordre 2 :

$$w"(t) = f(t, w(t), w'(t)),$$

on pose $x(t) = w(t),\ y(t) = w'(t)$, on obtient un système de deux équations d'ordre 1 et on peut donc réécrire le système précédent sous la forme

$$z'(t) = F(t, z(t)),\ F(t, x, y) = (y, f(t, x, y)),$$

avec $z(t) = (x(t), y(t))$.

5.7 Exercices

Dans chacun des exercices proposés, on cherchera à résoudre numériquement les équations différentielles dont il est question.

Exercice 30 (Le problème du pendule)**.** *Le mouvement d'un pendule de masse m, suspendu à un point O par un fil non pesant de longueur ℓ, en rotation d'angle $\theta(t)$ autour de O est gouverné par l'équation*

$$\theta"(t) = -g \sin(\theta(t))/\ell.$$

L'angle $\theta(t)$ est mesuré par rapport à une verticale passant par O. On s'intéresse au mouvement entre l'instant 0 et l'instant $T > 0$.
On se donne des conditions initiales :

$$\theta(0) = \pi/3, \quad \theta'(0) = 0.$$

Exercice 31 (Thermodynamique)**.** *Considérons un corps de température interne T dans un milieu ambiant de température constante T_e. Alors le transfert de chaleur entre le corps et le milieu ambiant peut être décrit par la loi de Stefan-Boltzmann*

$$v(t) = \epsilon\gamma S(T^4(t) - T_e^4),$$

où t est la variable temporelle, ϵ est la constante de Boltzmann (égal à $5.6 \cdot 10^{-8} J/m^2K^4s$, où J est pour Joule, K pour Kelvin et bien sûr m pour mètre et s pour seconde), γ est la constante d'émissivité du corps, S la superficie du corps et v le taux de transfert de chaleur. Le taux de variation de l'énergie $E(t) = mCT(t)$ (où C est la chaleur spécifique au matériau constituant le corps) est égal en valeur absolue à ce taux. Donc, en écrivant $T(0) = T_0$, le calcul de $T(t)$ nécessite la solution de l'équation différentielle ordinaire

$$T'(t) = -\frac{v(t)}{mC}.$$

Exercice 32. *Considérons une population de bactéries dans un environnement confiné dans lequel il ne peut y avoir plus que B éléments. Supposons qu'à l'instant initial, le nombre de bactéries est égal à $y_0 << B$ et que le taux d'accroissement est une constante $C > 0$. Dans ce cas, le taux de changement de la population est proportionnel au nombre de bactéries existantes, sous la restriction que ce nombre ne peut excéder B. Cela s'exprime par l'équation différentielle*

$$y'(t) = Cy(1 - \frac{y}{B}), \tag{5.3}$$

où la solution $y = y(t)$ désigne le nombre de bactéries à l'instant t.

Exercice 33. *Résoudre le problème de Lotka Volterra (5.2). On prendra $\alpha = \beta = \delta = 1$ et $\gamma = 2$.*

Exercice 34 (Le problème à deux corps)**.** *Pour calculer le mouvement de deux corps qui s'attirent l'un avec l'autre, on choisit un des deux corps comme centre de notre système de coordonnées ; le mouvement va alors rester dans un plan et on peut utiliser des coordonnées bidimensionnelles $q = (q_1, q_2)$ pour la position du deuxième corps. Les lois de Newton donnent alors (après une renormalisation convenable)*

$$q_1'' = -\frac{q_1}{(q_1^2 + q_2^2)^{3/2}}, \ q_2'' = -\frac{q_2}{(q_1^2 + q_2^2)^{3/2}}.$$