

# Partie Introduction au calcul scientifique

MAP101 - DLST - UGA - 2024-2025

Licence 1 - Parcours IMA, MIN Int. et S&D

## Table des matières

### **Intro. à Scilab**

1 Séance machine TP

### **Zéro(s) de fonction**

27 Séance préparatoire CTD

33 Séance machine TP

### **Interpolation**

37 Séance préparatoire CTD

43 Séance machine TP

### **Intégration numérique**

45 Séance préparatoire CTD

51 Séance machine TP

### **Résolution numérique d'équation différentielle**

53 Séance préparatoire CTD

57 Séance machine TP

Ce document correspond à la partie Introduction au calcul scientifique du module MAP101. La première partie consiste à une introduction au logiciel Scilab qui sera utilisée par la suite, suivie de quatre thèmes de base en mathématiques appliquées.

Pour ces quatre thèmes, une séance préparatoire en CTD aura lieu : l'étudiant.e devra au préalable avoir lu le document correspondant.

Ce document est disponible sur le site Chamilo-UGA (<http://chamilo.univ-grenoble-alpes.fr/courses/GBX1MP11/>) .

Notes :

# Introduction à Scilab


## séance pratique

Ce premier thème consiste à prendre en main le logiciel Scilab avec les principales fonctionnalités qui seront utilisées tout au long du semestre.

Ce premier thème est prévu sur 2 séances pratiques de 3h, et à faire en binôme.

Il est conseillé de faire les parties 1, 2, 3 et 4 lors de la première séance de 3h, puis les parties 5 et 6 lors de la seconde séance de 3h.

Suivez le déroulement de cette fiche, testez les différents exemples présentés.

**Différents exercices sont proposés au cours de cette séance. Certains exercices ou questions d'exercice indiqués par l'icône  feront l'objet d'un compte-rendu. Ce compte-rendu, à faire en binôme, consiste à créer un document de type traitement de texte, à y ajouter les réponses aux différents exercices, puis à la fin de la deuxième séance, exporter le compte-rendu au format PDF et l'envoyer par e-mail à votre enseignant.**

Pour ce premier thème, tous les exercices (1 à 6) font l'objet d'un compte-rendu.

Scilab est un logiciel libre, disponible gratuitement sur le site [www.scilab.org](http://www.scilab.org) que vous pouvez télécharger si vous souhaitez l'installer sur votre ordinateur personnel.

## 1 - Début avec Scilab

Au début de chaque séance de TP, il vous faut créer un répertoire où vous stockerez les différents fichiers utilisés lors de la séance.

Pour cette première séance, ouvrez votre répertoire personnel (menu *Démarrer*, *Ordinateur* puis *Emplacement réseau - home* ) qui correspond à votre répertoire personnel sur lequel vous pouvez mettre vos propres documents et que vous retrouverez tel quel lors de chaque séance sur machine.

Dans ce répertoire personnel, créez un nouveau répertoire nommé MAP101 puis dans celui-ci un nouveau répertoire nommé INTRO\_SCILAB.

Au début de chaque thème suivant, il suffira de créer un nouveau répertoire à côté du répertoire INTRO\_SCILAB.

### 1.1 - Démarrage de Scilab

Démarrez SCILAB : double-cliquez sur l'icône correspondant ou utilisez le menu **Programmes** de Windows.

La première fois, la fenêtre Scilab s'ouvre par défaut avec une *disposition intégrée* où différents outils de Scilab sont présents sous forme d'une seule fenêtre.

Cette fenêtre principale s'ouvre avec différents menus et icônes correspondants, avec la partie centrale (appelée *console*) où on peut entrer au clavier différentes commandes et où l'affichage de certains résultats est fait.

### 1.2 - Répertoire de travail

La première chose importante à faire est de vous placer dans le répertoire que vous avez créé précédemment afin de stocker les différents fichiers de la séance : dans le menu *Fichier*, sélectionnez l'item *Changer le répertoire courant ...* puis dans le dialogue, placez-vous dans le

répertoire voulu et cliquez sur le bouton **Ok**.

On peut à tout moment changer de répertoire courant.

On peut aussi retrouver les répertoires de travail déjà utilisés via l'item *Favoris* du menu *Fichier*, notamment lors d'un nouveau démarrage de Scilab.

### 1.3 - Mode interactif

On peut utiliser SCILAB comme une calculatrice interactive, en entrant des instructions dans la partie console. Normalement, il est affiché `-->` (le *prompt*), indiquant que la console est prête à exécuter des instructions entrées par l'utilisateur.

➡ si dans la console, autre chose que le prompt est affiché ou bien si rien n'est affiché, il est nécessaire de la remettre dans son état normal :

- soit en tapant simultanément **Ctrl C**,
- soit dans le menu *Contrôle*, en choisissant les items *Interrompre* et/ou *Abandonner*
- en dernier lieu, fermer et redémarrer l'application Scilab.

Exemple : Dans la console, tapez les instructions suivantes :

```
2.5*4-7/2
exp(2)
%e**2
3^2 , cos(%pi/3)
```

⇒ les fonctions mathématiques usuelles sont définies

⇒ la virgule (,) permet de séparer différentes instructions sur une même ligne

⇒ différentes constantes sont définies, on utilisera notamment la constante  $\pi \simeq 3,1416$  notée `%pi` et la constante  $e = \exp(1) \simeq 2,71828$  notée `%e`

Il est possible d'utiliser des variables afin de stocker le résultat d'un calcul afin de pouvoir s'en servir ensuite.


Exemple : Dans la console, tapez les instructions suivantes :

```
H = 2, R = 1.5, V = %pi * R**2 * H
H, R, V
```

⇒ on peut éventuellement réafficher, modifier et réexécuter les différentes instructions entrées précédemment en utilisant les touches *flèches haut* ↑ et *bas* ↓ du clavier.

### 1.4 - Utilisation de fichiers d'instructions

L'utilisation de la console peut devenir fastidieuse dès que le nombre d'instructions à écrire augmente. Il est alors préférable d'écrire les instructions à l'aide d'un éditeur de texte, pour ensuite les exécuter, les modifier et éventuellement les sauvegarder dans des fichiers qui pourront être réutilisés plus tard.

Pour utiliser l'éditeur de texte intégré, cliquez sur l'icône  ou bien sélectionnez l'item *SciNotes* du menu *Application* : l'éditeur de texte intégré de Scilab s'ouvre.


➡ il est recommandé dans un premier temps de sélectionner le répertoire où seront sauvegardés vos fichiers : pour cela dans le menu *Fichier* de l'éditeur SciNotes, sélectionner l'item *Répertoires de travail* puis sélectionner le répertoire qui vous servira par la suite.

⇒ par la suite les fichiers créés avec Scilab, seront nommés *fichiers-scripts* ou simplement *scripts*.

Dans la fenêtre de l'éditeur, tapez les instructions suivantes :

```
a = 7
b = 2
c = a/b
```

et sauvegardez-le (menu *Fichier*, item *Sauvegarder sous ...*) en le nommant **prog1.sce**  
⇒ par convention, les scripts Scilab contenant une suite d'instructions ont un nom avec le suffixe **.sce**


Une fois le script enregistré, pour exécuter les instructions qu'il contient, le plus simple est de cliquer sur l'icône .

⇒ remarquez que dans le cas de l'exécution de l'ensemble du fichier, rien n'est écrit à l'écran. Dans ce cas, il faut alors explicitement utiliser l'instruction **disp** afin d'afficher une valeur.  
**Exemple :** Complétez le script **prog1.sce** en rajoutant l'instruction

```
disp(c)
```

à la fin du fichier, puis refaites une exécution complète : la valeur de **c** est écrite dans la console.

Il est aussi possible d'exécuter la totalité d'un script ou seulement une partie de différentes manières :

- exécuter l'ensemble du fichier en tapant simultanément les touches **Ctrl**, **Majuscule** et **E**,
- en les sélectionnant une suite d'instructions à la souris dans l'éditeur puis en tapant simultanément les touches **Ctrl** et **E**,
- sauvegarder et exécuter l'ensemble du fichier en cliquant sur l'icône  dans la barre du haut de l'éditeur.

**Exemple :** Testez ces différentes possibilités avec le fichier **prog1.sce**

Une fois créé et sauvegardé, un fichier-script Scilab peut être réutilisé par la suite.


**Exemple :** Fermez l'éditeur de texte, puis dans la console, tapez l'instruction

```
exec("prog1.sce", -1)
```

⇒ le deuxième argument **-1** évite l'affichage du script dans la console.

⇒ on peut aussi utiliser l'item *Exécuter ...* du menu *Fichier* de la fenêtre *console*.

## 1.5 - Aide en ligne

Pour avoir de l'aide sur les différentes fonctionnalités et instructions de Scilab, utilisez l'instruction **help** ou cliquez sur l'icône .

Pour avoir l'aide sur une instruction particulière, utilisez l'onglet *loupe* du dialogue d'aide ou tapez l'instruction **help instruction**.

**Exemple :** Pour avoir l'aide sur l'instruction **clear**, tapez l'instruction **help clear**

## 2 - Valeurs et variables

Scilab permet la manipulation de valeurs (numériques, booléennes, texte), et de les conserver dans des variables.

Une variable est identifiée par un nom qui est une suite de caractères (lettres, chiffres, caractère *souligné*), le premier caractère devant être une lettre.

Exemples de noms de variables : `a`, `b`, `v1`, `t_35`, `Diametre_du_cercle`.

### 2.1 - Nombres, booléens et chaînes de caractères

Scilab permet de manipuler notamment des valeurs numériques, des booléens et des chaînes de caractères.

- **Valeur numérique** : les valeurs numériques permettent de manipuler des réels (et donc aussi des entiers). On peut utiliser avec des valeurs numériques les opérateurs classiques (+ *addition*, - *soustraction* ou *opposé*, \* *multiplication*, / *division*, \*\* ou ^ *puissance*).

Par exemple, testez les instructions suivantes

```
a = 3
a+4, 7-a, -a, a*5, 1/a, a**2, a^3
```

- **Booléen** : une valeur booléenne est une valeur qui peut être soit *vrai*, soit *faux*. Les valeurs booléennes servent entre autres dans des programmes à faire des tests ou des boucles, et exécuter ou non une suite d'instructions suivant la valeur booléenne d'une expression conditionnelle en comparant par exemple deux valeurs numériques.

Testez les instructions suivantes et observez les résultats obtenus :

```
a = 3, b = 3
a==b // a EGAL A b
a~=b // a DIFFERENT DE b
a<b // a INFÉRIEUR STRICTEMENT A b
a<=b // a INFÉRIEUR OU EGAL A b
a>b // a SUPÉRIEUR STRICTEMENT A b
a>=b // a SUPÉRIEUR OU EGAL A b
```

⇒ le caractère `~` s'obtient avec **AltGr 2** et le caractère `|` s'obtient avec **AltGr 6**.

⇒ le résultat de chaque comparaison entre `a` et `b` est soit *VRAI* (affiché **T**), soit *FAUX* (affiché **F**).

⇒ sur une ligne d'instruction, ce qui suit les deux symboles `//` est un commentaire, qui n'est pas pris en compte lors de l'exécution.

On peut ensuite construire des expressions booléennes plus complexes en utilisant les trois opérateurs booléens classiques (& *ET*, | *OU*, ~ *NON*).

Testez les instructions suivantes et observez les résultats obtenus :

```
a = 3, b = 3, c = 4
~(a==b)
(a<b) | (b<c)
(a<c) & (b+1<c)
```

⇒ on peut aussi utiliser `&&` pour l'opérateur *ET*, ainsi que `||` pour l'opérateur *OU*.

- **Chaîne de caractères** : Scilab permet aussi de manipuler des chaînes de caractères. Une chaîne de caractères est délimitée par des *double-quotes* (") . On se servira des chaînes de caractères notamment pour la définition de fonctions, ainsi que pour certaines options graphiques. Testez les instructions suivantes :

```
t = "Ceci est un texte"
disp(t)
```

## 2.2 - Définition, création de vecteurs

L'un des intérêts du logiciel Scilab est de pouvoir effectuer des calculs sur des tableaux de valeurs. Scilab permet de manipuler des variables de type *tableau* puis d'effectuer des opérations sur celles-ci.

Dans l'UE MAP101, nous utiliserons uniquement des *vecteurs-lignes*.

- Un *vecteur-ligne* est un tableau formé d'une seule ligne. Par la suite, on nommera *vecteur-ligne* par *vecteur*.

Exemple : Avec l'éditeur de texte, créez un nouveau fichier nommé **vecteurs.sce** avec les instructions suivantes :

```
u = [2 3 1 -7 9.5] // création d'un vecteur : séparer les
v = [1,3,5]         // valeurs par des espaces ou des virgules
w = [0,3,5,6]
disp(u), disp(v), disp(w) // afficher les vecteurs
```

puis exécutez-les.

## 2.3 - Accès aux éléments d'un vecteur

Pour accéder à un élément d'un vecteur, il suffit d'indiquer entre parenthèses l'indice correspondant (chaque indice est un entier supérieur ou égal à 1) :

Exemple : Observez les valeurs des instructions suivantes :

```
u(1), u(2) , v(3) , w(1) , w(4)
v(0) , v(6)
```

➡ en mathématiques, l'élément d'indice  $i$  d'un tableau  $t$  est noté  $t_i$ , et en Scilab, il est noté  $t(i)$  .

⇒ si l'indice utilisé est strictement inférieur à 1, ou strictement supérieur au nombre d'éléments du vecteur, un message d'erreur est indiqué.

Il est possible de *concaténer* des vecteurs afin de créer des vecteurs plus grands

Exemple : Testez les instructions suivantes :

```
v1 = [2 1]
v2 = [0 6 4]
v3 = [v1 v2]
v4 = [v2,v2,v2,v1]
v1 = [v1 8] // ajout de la valeur 8 à la fin du vecteur v1
v1 = [7 v1] // ajout de la valeur 7 au début du vecteur v1
```

## 2.4 - Création de vecteurs particuliers

Scilab fournit différentes fonctions afin de créer des tableaux particuliers.

- L'instruction `zeros(m,n)` crée un tableau avec  $m$  lignes et  $n$  colonnes et dont toutes les valeurs sont égales à 0.

Pour créer un vecteur-ligne, la première dimension ( $m$ ) est égale à 1.

Exemple : Testez les instructions suivantes :

```
zeros(1,3) , zeros(1,5) , zeros(1,7)
```

➡ cette instruction sert en particulier à créer un vecteur de taille donnée avec toutes ses valeurs nulles.

- L'instruction `ones(m,n)` crée un tableau avec  $m$  lignes et  $n$  colonnes et dont toutes les valeurs sont égales à 1.

Exemple : Testez les instructions suivantes :

```
ones(1,3) , ones(1,6)
```

- L'instruction `linspace(a,b,n)` crée un vecteur-ligne formé des  $n$  valeurs équiréparties entre  $a$  et  $b$ .

Exemple : Testez les instructions suivantes :

```
linspace(1,5,5) , linspace(0,4,9) , linspace(10,1,4)
```

➡ cette instruction crée un tableau de valeurs équiréparties dans un intervalle  $[a, b]$ , et sera notamment utilisé pour l'évaluation et le tracé de fonctions.

⇒ on remarque que le *pas* (la différence entre deux valeurs consécutives) est donnée par la formule  $\text{pas} = (b - a) / (n - 1)$ .

- Si on souhaite construire un vecteur-ligne en spécifiant le *pas*, on utilisera alors la notation `a:pas:b`

Exemple : Testez les instructions suivantes :

```
1:1:5 , 0:0.5:4 , 10:-1:1 , 10:-3:1 , 2:3:10 , 1:5 , 0:9
```

➡ si le *pas* n'est pas spécifié (syntaxe `a:b`), il est égal à 1. On utilisera cette notation en particulier dans les boucles `for`.

## 2.5 - Dimensions d'un vecteur

A tout moment, il est possible de connaître la dimension (nombre d'éléments) d'un vecteur avec la fonction `length`.

Exemple : Testez les instructions suivantes en observant le résultat après chacune d'elles :

```
v = 1:8 , w = [3 6 1 4] , t_vide = []
length(v) // donne le nombre de valeurs du vecteur v
length(w) // donne le nombre de valeurs du vecteur w
length(t_vide) // donne le nombre de valeurs du vecteur t_vide
```

⇒ on peut définir un vecteur vide `[]` contenant aucune valeur.



## 2.6 - Opérations arithmétiques sur les tableaux

Une fois un tableau défini, on peut réaliser certaines opérations arithmétiques sur l'ensemble des éléments d'un tableau ou entre tableaux de même dimensions.

Exemple : Testez les instructions suivantes :

```
v = [1 2 3 4 5 6]
v + 2      // ajoute 2 à tous les éléments de v
v - 3      // retranche 3 à tous les éléments de v
v .* (-4)  // multiplie tous les éléments de v par -4
v ./ 10    // divise tous les éléments de v par 10
v .^ 2     // élève tous les éléments de v au carré
2 .** v    // calcule des puissances de 2
```

⊛ : pour les opérations arithmétiques *multiplication*, *division* et puissance, où l'un des opérandes est un tableau, il faut utiliser les opérateurs `.*` `./` `.^` `.**` au lieu des opérateurs `*` `/` `^` `**`

On peut aussi effectuer des opérations arithmétiques (sauf l'opération `**`) terme à terme entre deux tableaux de même dimensions.

Exemple : Testez les instructions suivantes :

```
u = [1 2 3 4 5 6] , v = [4 7 1 0 2 8]
u + v , v + u , u - v , v - u
u .* v , v .* u , v ./ u
1 ./ u    // les inverses des éléments de u
```

⇒ il est conseillé de mettre un espace AVANT et APRÈS chaque opérateur.

Pour effectuer une opération arithmétique entre deux tableaux, ils doivent nécessairement être de même dimensions.

Exemple : Testez les instructions suivantes :

```
u = [1 2 3 4 5 6] , v = [1 1 1 1]
u + v
u .* v
```

### Exercice 1 :

En utilisant les instructions vues précédemment, écrire de la manière la plus simple possible les instructions pour créer les vecteurs suivants :

$$v1 = ( 5 \ 6 \ 7 \ 8 \ 9 \ 10 \ 11 \ 12 \ 13 )$$

$$v2 = ( 5 \ 6 \ 7 \ 8 \ 9 \ 10 \ 11 \ 12 \ 13 \ 18 \ 15 \ 12 \ 9 \ 6 \ 3 \ 0 )$$

$$v3 = ( \underbrace{0 \ 0 \ \dots \ 0 \ 0}_{\text{indices 1 à 100}} \ \underbrace{1 \ 1 \ \dots \ 1 \ 1}_{\text{indices 101 à 200}} \ \underbrace{2 \ 2 \ \dots \ 2 \ 2}_{\text{indices 201 à 300}} )$$

$v3$  vecteur de taille 300 avec les 100 premières valeurs nulles, les 100 valeurs suivantes égales à 1, et les 100 dernières valeurs égales à 2.

$$v4 = ( 0 \ 4 \ 8 \ \dots \ 388 \ 392 \ 396 )$$

$v4$  vecteur de taille 100 avec les 100 premiers nombres naturels multiples de 4.

**CR** Dans le compte-rendu, copiez les instructions Scilab que vous avez utilisé pour créer les vecteurs  $v1$ ,  $v2$ ,  $v3$  et  $v4$ .

### 3 - Programmation

Scilab dispose d'un langage avec instructions structurées afin d'écrire des programmes complexes. Le langage est de type *impératif* (comme par exemple les langages C, Java, Python).

#### 3.1 - Variables et affectation

On peut stocker différentes valeurs (ou tableaux de valeurs) dans des variables, une variable étant identifiée par un nom ; le nom d'une variable est formée de un ou plusieurs caractères (lettres, chiffres, caractère souligné), le premier caractère doit être une lettre ou le caractère souligné, et il doit y avoir au moins une lettre dans le nom d'une variable.

En Scilab, une variable se définit *à la volée* en lui affectant le résultat d'une expression.

#### Affectation

Notation algorithmique	Notation Scilab
$variable \leftarrow expression$	$variable = expression$

#### 3.2 - Entrée-sortie

L'instruction `disp` permet l'affichage d'une variable ou d'une expression.

L'instruction `input` permet à l'utilisateur d'entrer une valeur ou un tableau.

Exemple : créez le fichier suivant en le nommant `ex_entree_sortie.sce`

```
disp("Valeur de pi : ") , disp(%pi)

n = input("Entrer un entier : ")
disp("n*n = ") , disp(n*n)

t = input("Entrer un tableau de valeurs (entre crochets) : ")
disp("Valeurs de t au carré = ") , disp(t**2)
```

puis exécutez le script `ex_entree_sortie.sce`.

L'instruction `disp` permet l'affichage simplifié de valeurs, notamment, dans le cas de valeurs non entières, le nombre de décimales après la virgule est fixé.

La fonction Scilab `mprintf` (équivalent de la fonction `printf` du langage C) permet un affichage personnalisé pour des valeurs.

Exemple : créez le fichier suivant en le nommant `ex_mprintf.sce`


```
//----- affichage d'une valeur entière -----
// affichage standard
mprintf("5 + 7 = %d\n", 5+7)
// affichage sur 4 caractères
mprintf("8 * 5 = %4d\n", 8*5)

//----- affichage d'une valeur au format décimal -----
// affichage standard
mprintf("1 / 7 = %f\n", 1/7)
// affichage de 7 chiffres après la virgule
mprintf("1 / 7 = %.7f\n", 1/7)
// affichage de 10 chiffres après la virgule
// et 15 caractères au total
mprintf("1 / 7 = %15.10f\n", 1/7)
```

```
// ----- affichage au format scientifique -----
// ----- affichage avec une puissance de 10 -----
// ----- correspondant à son ordre de grandeur -----
// affichage standard
mprintf("2**100 = %e\n", 2**100)
// affichage de 3 chiffres apres la virgule
mprintf("2**100 = %.3e\n", 2**100)
// affichage de 12 chiffres apres la virgule
// et 20 caracteres au total
mprintf("2**100 = %20.12e\n", 2**100)

// ----- affichage de plusieurs valeurs -----
v = 8
mprintf("l'inverse de %f est %f\n", v, 1/v)
```

puis exécutez le script `ex_mprintf.sce`.

➡ un script contenant une instruction `input` doit être exécuté *sans écho* (Ctrl-Majuscule-E ou icône ) sinon la console risque d'être bloquée.

### Entrée-sortie

Notation algorithmique	Notation Scilab
<b>lire</b> ( <i>variable</i> )	<code>variable = input</code> <code>variable = input(texte)</code>
<b>ecrire</b> ( <i>valeur</i> )	<code>disp(valeur)</code> <code>mprintf("%d\n", valeur_entiere)</code> <code>mprintf("%f\n", valeur)</code> <code>mprintf("%e\n", valeur)</code>

### 3.3 - Commentaire

Un commentaire correspond à la partie d'une ligne se trouvant après deux caractères `//` et n'est pas pris en compte lors de l'exécution.

### 3.4 - Test conditionnel

L'instruction `if` permet d'exécuter une suite d'instructions si et seulement si une expression est vraie.

#### Test simple

Notation algorithmique	Notation Scilab
<b>si</b> <i>expression_booleenne</i> <b>alors</b> <i>instructions</i> <b>fin_si</b>	<b>if</b> <i>expression_booleenne</i> <b>then</b> <i>instructions</i> <b>end</b>
<b>si</b> <i>expression_booleenne</i> <b>alors</b> <i>instructions_1</i> <b>sinon</b> <i>instructions_2</i> <b>fin_si</b>	<b>if</b> <i>expression_booleenne</i> <b>then</b> <i>instructions_1</i> <b>else</b> <i>instructions_2</i> <b>end</b>

On peut aussi enchaîner plusieurs tests à la suite :

#### Test multiple

Notation algorithmique	Notation Scilab
<b>si</b> <i>expression_booleenne</i> <b>alors</b> <i>instructions_1</i> <b>sinon_si</b> <i>expression_booleenne</i> <b>alors</b> <i>instructions_2</i> : <b>sinon</b> <i>instructions_autres</i> <b>fin_si</b>	<b>if</b> <i>expression_booleenne</i> <b>then</b> <i>instructions_1</i> <b>elseif</b> <i>expression_booleenne</i> <b>then</b> <i>instructions_2</i> : <b>else</b> <i>instructions_autres</i> <b>end</b>

Exemple : créez le fichier suivant en le nommant `ex_test.sce`

```
n = input("Entrer un entier n : ")

if n>2 then
    disp("n est supérieur à 2")
end

if n==0 then
    disp("n est nul")
elseif n>0 then
    disp("n est strictement positif")
else
    disp("n est strictement négatif")
end
```

puis exécutez le script `ex_test.sce` plusieurs fois en entrant différentes valeurs pour `n`.

• opérateurs de comparaison :

<	>	==
inférieur strictement à	supérieur strictement à	égal à
<=	>=	<>
inférieur ou égal à	supérieur ou égal à	différent de

• opérateurs booléens :

&		~
&&		
ET	OU	NON

### 3.5 - Boucle conditionnelle

L'instruction **while** permet de répéter une suite d'instructions tant qu'une expression booléenne est vraie.

En général, on utilise une boucle **while** quand on ne connaît pas *a priori* le nombre d'itérations qu'on va faire dans la boucle.

#### Boucle conditionnelle

Notation algorithmique	Notation Scilab
<b>tant_que</b> <i>expression_booléenne</i> <b>faire</b> <i>instructions</i> <b>fin_tant_que</b>	<b>while</b> <i>expression_booléenne</i> <i>instructions</i> <b>end</b>

Exemple : créez le fichier suivant en le nommant `ex_boucle1.sce`

```
// création d'un vecteur v de valeurs positives ou nulles,
// l'utilisateur entre différentes valeurs, tant qu'elles sont
// positives ou nulles, elles sont stockées dans un vecteur v,
// la boucle s'arrête dès que l'utilisateur entre une valeur négative

v = []
boucle = %T
while boucle
    val = input("Entrer une valeur (négative = arrêt) : ")
    if val >= 0 then
        // ajout de val dans le vecteur v
        v = [v val]
    else
        // fin de la boucle
        boucle = %F
    end
end
mprintf("%d valeur(s) entrée(s) : ", length(v))
disp(v)
```

puis exécutez le script `ex_boucle1.sce`.

### 3.6 - Boucle inconditionnelle

L'instruction **for** permet de répéter une suite d'instructions pour un ensemble de valeurs.

En général, on utilise une boucle **for** quand on connaît le nombre d'itérations qu'on va faire dans la boucle.

#### Boucle inconditionnelle

Notation algorithmique	Notation Scilab
<b>pour</b> <i>variable</i> <b>appartenant à</b> <i>tableau_valeurs</i> <b>faire</b> <i>instructions</i> <b>fin_pour</b>	<b>for</b> <i>variable</i> = <i>tableau_valeurs</i> <i>instructions</i> <b>end</b>

Souvent, on l'utilise avec la syntaxe suivante (avec *v\_min* et *v\_max* des nombres entiers) :

Notation algorithmique	Notation Scilab
<b>pour</b> <i>variable</i> <b>de</b> <i>v_min</i> <b>à</b> <i>v_max</i> <b>faire</b> <i>instructions</i> <b>fin_pour</b>	<b>for</b> <i>variable</i> = <i>v_min</i> : <i>v_max</i> <i>instructions</i> <b>end</b>

et les instructions sont exécutées pour  $variable = valeur\_min$ ,  $variable = valeur\_min+1, \dots$ , jusqu'à  $variable = valeur\_max$  (incluse).

Exemple : créez le fichier suivant en le nommant `ex_boucle2.sce`

```
t = [2.4 7.4 8 3.1 9.5 0.1]
n = length(t) // nombre d'éléments du tableau t

// affichage des éléments de t
disp("Elements de t")
for i = 1:n
    disp(t(i))
end

// calcul de la somme des éléments de t
somme_t = 0
for i = 1:n
    somme_t = somme_t+t(i)
end
mprintf("La somme des elements de t est %f\n", somme_t)
```

puis exécutez le script `ex_boucle2.sce`.

➡ retenez la manière de calculer la somme des éléments d'un tableau en utilisant une boucle `for`.

### Exercice 2 :

Ecrire un script Scilab nommé `ex_suite.sce` qui effectue les opérations suivantes :

- demande à l'utilisateur d'entrer une valeur réelle positive  $a$ ,
- crée un vecteur  $u$  de taille 10 avec toutes les valeurs nulles :

$$u = (u_1, u_2, \dots, u_{10}) = (0, 0, 0, 0, 0, 0, 0, 0, 0, 0)$$

- modifie les valeurs du vecteur  $u$  en utilisant la récurrence suivante :

$$u_1 = \frac{a+1}{2} \quad \text{et} \quad \text{pour } n \text{ variant de } 2 \text{ à } 10, \quad u_n = \frac{a}{2 u_{n-1}} + \frac{u_{n-1}}{2}$$

- affiche les valeurs du vecteur  $u$  ainsi que les valeurs au carré du vecteur  $u$ .

**CR** Dans le compte-rendu, copiez le script `ex_suite.sce`.

## 4 - Fonctions

### 4.1 - Fonctions prédéfinies

Scilab fournit un certain nombre de fonctions notamment les principales fonctions mathématiques (voir la partie *Elementary functions* de l'aide en ligne).

La plupart des fonctions peuvent s'appliquer aussi bien à une seule valeur qu'à un tableau de valeurs.

Exemple : Testez les instructions suivantes :

```
sqrt(4) // racine carrée de 4
k = 0:12
sqrt(k) // les valeurs [sqrt(0), sqrt(1), sqrt(2), ..., sqrt(12)]
t = k * %pi / 6
cos(t) // les valeurs [cos(0), cos(pi/6), cos(2*pi/6), ..., cos(2*pi)]
```

➡ Dans Scilab, les fonctions usuelles sont définies, notamment :

- abs (valeur absolue),
- exp (fonction exponentielle), log (logarithme népérien ln),
- cos, sin, tan (fonctions trigonométriques),
- cosh, sinh, tanh (fonctions hyperboliques).

On peut à partir des opérateurs et fonctions prédéfinies de Scilab, créer ses propres fonctions.

Exemple : Pour calculer les valeurs de la fonction  $f(x) = \frac{1}{1+x^2}$  pour les valeurs  $x = 0, x = 0,5, x = 1, \dots, x = 9,5$  et  $x = 10$ , tapez les instructions suivantes :

```
x = 0:0.5:10
y = 1 ./ (1 + x .** 2) // espace obligatoire avant ./
```

On aimerait pouvoir définir la fonction  $f(x)$  puis l'utiliser à l'aide de l'instruction  $y=f(x)$ . Scilab fournit deux manières de définir ses propres fonctions.

### 4.2 - Définition de fonctions simples

Pour des fonctions définies par une expression unique, il est conseillé d'utiliser l'instruction `deff`.

Exemple : Avec l'éditeur de texte, créez le fichier suivant en le nommant `ex_fct1.sce`

```
// definition de la fonction f avec y = f(x) = 1/(1+x^2)
deff("y = f(x)" , "y = 1 ./ (1 + x .** 2)");

//----- utilisation de la fonction f -----

x = 0:0.5:10 // calcul de y = f(x) pour x=0 , x=0,5 ,
y = f(x) // x=1 ... x=9,5 et x=10
disp(y)

t = linspace(-1,2,1000) // calcul de f(t) pour 1000 valeurs de t
z = f(t) // équiréparties entre -1 et 2
disp(z)
```

puis exécutez le script `ex_fct1.sce`.

➡ pour la multiplication, la division et la puissance, il est obligatoire d'utiliser les opérations terme à terme (`.* ./ .^ .**`) dans la définition d'une fonction pour pouvoir ensuite utiliser la fonction avec un tableau de valeurs.

⇒ une fois la fonction définie, on peut l'utiliser avec n'importe quelle variable (pas nécessairement avec des variables ayant les mêmes noms que dans la définition de la fonction).

⇒ dans le cas de l'affichage d'un tableau avec un grand nombre de valeurs, Scilab demande à l'utilisateur de continuer ou non l'affichage du tableau dans la console.

Une ou plusieurs fonctions peuvent être définies dans un fichier-script puis être utilisées dans un autre fichier-script Scilab.

Exemple : Avec l'éditeur de texte, créez le fichier suivant en le nommant `mes_fonctions.sci`

```
// definition de la fonction f1 avec y = f1(x) = 1/(1+x^2)
deff("y = f1(x)" , "y = (1) ./ (1 + x .** 2)");

// definition de la fonction f2 avec y = f2(t) = (t+1)^2
deff("y = f2(t)" , "y = (t+1) .** 2");
```

puis créez le fichier suivant en le nommant `ex_fct2.sce`

```
// chargement du contenu du fichier nommé mes_fonctions.sci
exec("mes_fonctions.sci", -1);

// calcul de f1(x) pour x=0 x=0,5 x=1 ... x=9,5 et x=10
x = 0:0.5:10
y = f1(x)
disp(y)

// calcul de f2(u) pour 100 valeurs de u entre -10 et 10
u = linspace(-10,10,100)
z = f2(u)
disp(z)
```

puis exécutez le script `ex_fct2.sce`.

⇒ par convention, les fichiers contenant uniquement des définitions de fonctions ont un nom avec le suffixe `.sci` alors que les scripts Scilab ont un nom avec le suffixe `.sce`

### Exercice 3 :

ajouter au fichier nommé `mes_fonctions.sci` la définition des trois fonctions :

$$f_3(x) = \ln \left( x + \sqrt{x^2 - 1} \right), \quad f_4(x) = \ln \left( x + \sqrt{x^2 + 1} \right), \quad f_5(x) = \frac{1}{2} \ln \left( \frac{1+x}{1-x} \right)$$

puis écrire un script Scilab nommé `ex_fct3.sce` qui permet :

- de créer le vecteur `t` des valeurs entre 0 et 5 avec un pas de  $0,2 = 1/5$ ,
- de calculer les trois vecteurs `u = f3(cosh(t))`, `v = f4(sinh(t))` et `w = f5(tanh(t))`,
- et d'afficher les tableaux `t`, `u`, `v` et `w`.

Les fonctions  $f_3$ ,  $f_4$  et  $f_5$  étant les fonctions réciproques respectivement de  $\cosh$ ,  $\sinh$  et  $\tanh$ , les 4 tableaux doivent être identiques.

**CR** Dans le compte-rendu, ajoutez les instructions (copiez le contenu) de vos scripts `mes_fonctions.sci` et `ex_fct3.sce`.



### 4.3 - Fonctions définies par morceaux

Pour des fonctions définies par plusieurs expressions (fonctions définies *par morceaux*), la définition puis l'évaluation des ces fonctions doit se faire différemment :

- la fonction est définie à l'aide du bloc `function ...endfunction`,
- l'évaluation de la fonction doit se faire valeur réelle par valeur réelle, et non pas directement avec un tableau de valeurs

Exemple : on va définir la fonction suivante

$$g(x) = \begin{cases} \exp(x) & \text{si } x < 0 \\ -x^2/2 + x + 1 & \text{si } x \geq 0 \end{cases}$$

puis l'évaluer pour différentes valeurs entre -1 et 1.

La fonction  $g$  est définie *par morceaux*, l'expression  $y = g(x)$  dépend de l'intervalle auquel appartient  $x$ . Pour une telle fonction, on ne peut pas l'évaluer directement avec un tableau de valeurs, mais valeur par valeur.

Avec l'éditeur de texte, créez le fichier suivant nommé `ex_fonction_par_morceaux.sce` puis exécutez-le.

```
// définition de la fonction g(x)
function y=g(x)
    if x<0 then
        y = exp(x)
    else
        y = -x.**2./2+x+1
    end
endfunction

// calcul et affichage de la fonction y = g(x)
// pour 21 valeurs équiréparties entre -1 et 1
xt = linspace(-1,1,21)
n = length(xt) // nb de valeurs dans le tableau xt
for i=1:n
    x = xt(i)
    y = g(x)
    mprintf("x = %15.10f , y = g(x) = %15.10f\n", x, y)
end
```

➡ pour l'expression correspondant à  $-x^2/2 + x + 1$ , on aurait pu écrire `-x**2/2+x+1`, mais pour une expression définissant une fonction, prenez l'habitude d'utiliser `.* ./ .^ .**`

## 5 - Graphique et tracé de fonction

Scilab dispose de fonctionnalités afin de tracer des graphiques à partir de données sous forme de tableaux de valeurs. En MAP101, on utilisera essentiellement l'instruction `plot` sous l'une des formes suivantes

```
plot(x,y)
plot(x,y,options_graphiques)
```

### 5.1 - Représentation de points du plan

Les deux premiers arguments  $x$  et  $y$  de l'instruction `plot` sont deux vecteurs de même taille définissant les coordonnées de points dans le plan, le vecteur  $x$  correspond aux abscisses des points et le vecteur  $y$  correspond aux ordonnées des points.

- On peut alors représenter un ensemble de points  $(x_i, y_i)$ ,  $1 \leq i \leq n$  du plan, en définissant un vecteur  $\mathbf{x} = [x_1 \ x_2 \ \dots \ x_n]$  et un vecteur  $\mathbf{y} = [y_1 \ y_2 \ \dots \ y_n]$ , chaque vecteur contenant  $n$  valeurs.

**Exemple :** Pour représenter les 4 points  $(2, 1)$ ,  $(0, 0)$ ,  $(3, -1)$  et  $(4, 2)$ , d'abord définir les deux vecteurs pour les abscisses et les ordonnées :

```
x1 = [ 2  0  3  4]
y1 = [ 1  0 -1  2]
```

puis effectuer le tracé :

```
scf() // creer une nouvelle fenetre graphique
plot(x1,y1,".") // option "." : tracé de points
```

⇒ on remarque que les limites du repère correspondent aux limites des données soit l'intervalle  $[0, 4]$  en abscisse et l'intervalle  $[-1, 2]$  en ordonnée, et certains points sont peu visibles.

- Pour modifier les limites du repère, il suffit d'utiliser l'instruction `replot([xmin,ymin,xmax,ymax])`

**Exemple :** Testez les instructions suivantes en observant le résultat après chacune d'elles :

```
replot([-1 -2 5 3]), replot([-10 -10 10 10]), replot([0 0 4 4])
```

- On peut modifier le mode de tracé en modifiant le troisième paramètre de la procédure `plot` qui est une option de tracé sous forme d'une chaîne de caractères combinant couleur et style.

**Exemple :** Testez les instructions suivantes en observant le résultat après chacune d'elles et observez notamment le résultat visuel des différentes options :

```
scf(), plot(x1,y1,"g-"), replot([-3 -2 5 3])
scf(), plot(x1,y1,"k--"), plot(x1,y1,'ro'), replot([-3 -2 5 3])
scf(), plot(x1,y1), plot(x1,y1,'c.'), replot([-3 -2 5 3])
```

⇒ on peut effectuer différents tracés dans une même fenêtre graphique en effectuant plusieurs instructions `plot` à la suite.

- Par défaut, le tracé ne respecte pas automatiquement la même échelle pour l'axe des  $x$  et l'axe des  $y$  (repère non orthonormé).

**Exemple :** Exécutez les instructions suivantes :

```
scf()
x2 = [-3  0  3  0 -3]
y2 = [ 0  3  0 -3  0]
plot(x2,y2,'b:'), plot(x2,y2,'ro')
replot([-4 -4 4 4])
```

Les points des vecteurs  $\mathbf{x2}$  et  $\mathbf{y2}$  correspondent aux sommets d'un carré, alors que la représentation graphique ne donne pas nécessairement un carré (on voit plutôt un losange). Pour avoir une représentation graphique plus juste, il faut faire en sorte que le repère soit normalisé (orthonormé : même échelle en abscisse et en ordonnée).

Exécutez l'instruction suivante :

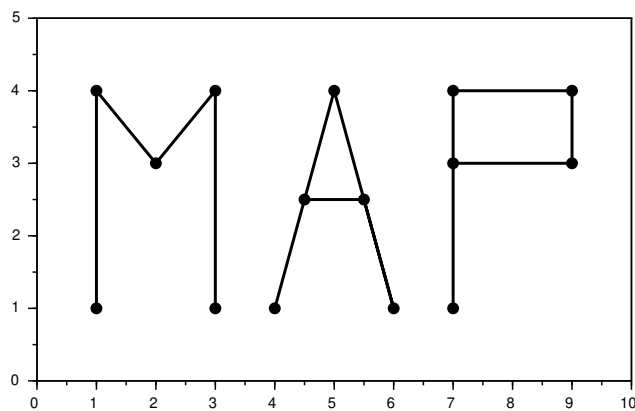
```
set(gca(),"isoview","on")
```

⇒ pour plus d'information sur l'instruction `plot`, tapez l'instruction `help plot` .

⇒ pour plus d'information sur les options de tracé, tapez l'instruction `help LineSpec` .

#### Exercice 4 :

Ecrire un script Scilab permettant d'obtenir la figure suivante :



**CR**

Dans le compte-rendu, copiez les instructions de votre script.

## 5.2 - Tracé de courbes représentatives de fonctions

Dans ce paragraphe, nous allons voir comment tracer le graphe  $y = f(x)$  d'une fonction  $f$  dont on connaît l'expression en fonction de  $x$ .

### • Tracé d'une fonction continue sur un intervalle $[a, b]$

Exemple : tracer le graphe de  $f(x) = x^2$ .

Le domaine de définition de  $f$  étant  $\mathbb{R}$ , on ne peut pas représenter le graphe pour toutes les valeurs  $x$  de  $\mathbb{R}$  mais seulement pour un intervalle  $I = [a, b]$ .

De plus, on ne peut pas calculer la fonction  $f$  sur toutes les valeurs de l'intervalle  $I$  (car il y en a une infinité), mais seulement avec un nombre fini de valeurs entre  $a$  et  $b$ .

Créez un fichier script Scilab nommé `trace_fonction1.sce` avec les instructions suivantes :

```
deff("y = f(x)","y = x .^ 2") // définir la fonction f
a = -4; b = 4 // les bornes de l'intervalle I
xt = linspace(a,b,10) // créer un tableau de 10 valeurs entre a et b
yt = f(xt) // calculer les valeurs yt correspondant à xt
scf() // créer une fenetre graphique
plot(xt,yt,'.') // tracer les points (xt(i),yt(i))
```

et exécutez ce script.

Seuls les 10 points du graphe de  $f$  correspondant aux 10 valeurs du tableau `x` sont affichés.

Pour avoir un tracé visuellement continu, il faut relier les plans entre eux.

Modifiez le script précédent en remplaçant l'instruction `plot(xt,yt,'.')` par `plot(xt,yt,'-')` et ré-exécutez-le.

Le graphique présente une courbe continue mais avec des points anguleux (la courbe a un rendu visuel *non lisse*).

Pour avoir une courbe avec un rendu visuel lisse, il faut augmenter le nombre de valeurs dans le tableau `x` : modifiez le script précédent en remplaçant l'instruction `xt = linspace(a,b,10)` par `xt = linspace(a,b,1000)` et ré-exécutez-le.

Il est possible ensuite de fixer les limites du repère avec l'instruction `replot`.

Par exemple, si on souhaite que le repère du graphique se limite à  $x$  entre  $-2$  et  $2$ , et  $y$  entre  $-2$  et  $6$ , rajoutez dans le script précédent, l'instruction `replot([-2 -2 2 6])`.

➡ pour tracer une fonction  $y = f(x)$  définie et continue sur un intervalle  $[a; b]$ , il faut :

1. définir la fonction  $y = f(x)$  avec l'instruction `deff("y = f(x)","y = ...")`
2. définir un tableau `xt` de valeurs de  $x$  entre  $a$  et  $b$  avec l'instruction `xt = linspace(a,b,1000)`
3. calculer les valeurs  $y = f(x)$  avec l'instruction `yt = f(xt)`
4. éventuellement créer une nouvelle fenêtré graphique avec l'instruction `scf()`
5. effectuer le tracé graphique avec l'instruction `plot(xt,yt,...)`
6. fixer les limites du repère avec l'instruction `replot`

⇒ on peut éventuellement enrichir et personnaliser la représentation.

Rajoutez à la fin du script précédent, les instructions suivantes :

```
xtitle("f(x) = x^2") // ajouter un titre au graphique
axes = gca() // le repère-axes graphique
axes.x_location = "origin" // repère-axes passant par l'origine
axes.y_location = "origin"
axes.box = "off" // supprimer la boite englobant le repère-axes
axes.isoview = "on" // normaliser le repère
```

### Tracé d'une fonction non définie pour certaines valeurs d'un intervalle

Exemple : tracer la fonction  $f(x) = \frac{1}{x^2}$  pour  $x$  compris entre -6 et 6, et en faisant en sorte que le repère soit limité à  $x$  entre -6 et 6, et  $y$  entre -2 et 8.

La fonction  $f(x)$  n'étant pas définie en  $x = 0$ ,  
il faut la tracer d'abord sur l'intervalle  $I_1 = [-6; 0[$  puis sur l'intervalle  $I_2 = ]0; 6]$ .

Comme pour l'exemple précédent, les deux intervalles ne contenant pas 0, on choisit un intervalle un peu plus petit au voisinage de 0 : on remplace l'intervalle  $[-6; 0[$  par  $[-6; 0 - \varepsilon]$ , et l'intervalle  $]0; 6]$  par  $[0 + \varepsilon; 6]$ .

Créez un fichier script Scilab nommé `trace_fonction2.sce` avec les instructions suivantes puis exécutez-le :

```
// définition de f
deff("y = f(x)" , "y = 1 ./ x.**2;") // mettre un espace entre 1 et ./

eps = 10^(-8)

scf()

// tracé sur [-6,0[ : tracé sur [-6,0-eps]
xt = linspace(-6,0-eps,1000)
yt = f(xt)
plot(xt,yt,"b-")

// tracé sur ]0,6] : tracé sur [0+eps,6]
xt = linspace(0+eps,6,1000)
yt = f(xt)
plot(xt,yt,"b-")
replot([-6 -2 6 8]) // modifier les bornes du repere

axes = gca() // l'objet 'repere-axes'
axes.x_location = "origin" // axe des x passant par l'origine
axes.y_location = "origin" // axe des y passant par l'origine
axes.box = "off" // supprimer la boite englobant le repere-axes
```

⇒ **IMPORTANT** : si une fonction  $f$  n'est pas définie en un point  $c$  d'un intervalle  $[a; b]$ , pour faire sa représentation graphique, il faut la tracer en deux parties, une partie correspondant à l'intervalle  $[a; c - \varepsilon]$ , et une partie correspondant à l'intervalle  $[c + \varepsilon; b]$  avec  $\varepsilon$  une valeur strictement positive proche de 0.

### Tracé d'une fonction continue définie par morceaux

Exemple : tracer le graphe de  $g(x) = \begin{cases} \exp(x) & \text{si } x < 0 \\ -x^2/2 + x + 1 & \text{si } x \geq 0 \end{cases}$  pour  $x \in [-1; 1]$ .

On peut remarquer que la fonction est continue en  $x = 0$  car

$$\begin{cases} \lim_{x \rightarrow 0^-} g(x) = \lim_{x \rightarrow 0^-} \exp(x) = \exp(0) = 1 \\ \lim_{x \rightarrow 0^+} g(x) = g(0) = -0^2/2 + 0 + 1 = 1 \end{cases}$$

Il y a deux possibilités pour effectuer le tracé d'une telle fonction.

(1) Créez un fichier script Scilab nommé `trace_fonction3.sce` avec les instructions suivantes puis exécutez-le :

```
// définition de la fonction g(x)
function y=g(x)
    if x<0 then
        y = exp(x)
    else
        y = -x.**2./2+x+1
    end
endfunction

// calcul de la fonction g pour des valeurs x entre -1 et 1
n = 1000 // nombre de valeurs pour x
xt = linspace(-1,1,n)
yt = zeros(1,n) // créer un vecteur de meme taille que xt
for k=1:n
    yt(k) = g(xt(k))
end

// tracé de g
scf()
plot(xt,yt)
```

(2) Créez un fichier script Scilab nommé `trace_fonction4.sce` avec les instructions suivantes puis exécutez-le :

```
scf()

// définition et tracé de g pour x <= 0
deff("y = g(x)" , "y = exp(x)")
xt = linspace(-1,0,1000)
yt = g(xt)
plot(xt,yt,"b-")

// définition et tracé de g pour x >= 0
deff("y = g(x)" , "y = -x.**2./2+x+1")
xt = linspace(0,1,1000)
yt = g(xt)
plot(xt,yt,"r-")
```

⇒ notez les avantages et inconvénients des deux méthodes.

**Exercice 5 :**

Le but de cet exercice consiste en la création d'un formulaire avec les graphiques des fonctions usuelles vues en cours-TD d'analyse.

Pour cela, inspirez-vous des exemples précédents afin de tracer dans des fenêtres séparées les courbes représentatives des différentes fonctions suivantes :

1.  $f(x) = x^2$  pour  $x$  entre  $-2$  et  $2$  et  $y$  entre  $0$  et  $4$ ,
2.  $f(x) = x^3$  pour  $x$  entre  $-2$  et  $2$  et  $y$  entre  $-4$  et  $4$ ,
3. sur un même graphique  $f_1(x) = x$ ,  $f_2(x) = x^2$ ,  $f_3(x) = x^3$ ,  $f_4(x) = x^4$ ,  $f_5(x) = x^5$  pour  $x$  entre  $0$  et  $1$  et  $y$  entre  $0$  et  $1$ ,
4.  $f(x) = 1/x$  pour  $x$  entre  $-10$  et  $10$  et  $y$  entre  $-10$  et  $10$ ,
5.  $f(x) = |x|$  pour  $x$  entre  $-10$  et  $10$  et  $y$  entre  $0$  et  $10$ ,
6. sur un même graphique les deux fonctions  $f_1(x) = \exp(x)$ ,  $f_2(x) = \ln(x)$  avec les bornes du repère graphique tel que  $x$  entre  $-6$  et  $6$  et  $y$  entre  $-6$  et  $6$ ,
7.  $f(x) = \sqrt{x}$  pour  $x$  entre  $0$  et  $10$ ,
8. sur un même graphique les 9 fonctions suivantes  $f(x) = a^x$  correspondant aux valeurs de  $a$  suivantes :

$a = 1/3$	$a = 2/5$	$a = 1/2$	$a = 2/3$	$a = 1$	$a = 3/2$	$a = 2$	$a = 5/2$	$a = 3$
-----------	-----------	-----------	-----------	---------	-----------	---------	-----------	---------

avec les bornes du repère graphique tels que  $x$  entre  $-3$  et  $3$  et  $y$  entre  $0$  et  $10$ ,

9. sur un même graphique les trois fonctions  $\cos(x)$ ,  $\sin(x)$ ,  $\tan(x)$ , avec les bornes du repère graphique tels que  $x$  entre  $-3\pi/2$  et  $3\pi/2$  et  $y$  entre  $-4$  et  $4$ ,
10. sur un même graphique les trois fonctions  $\cosh(x)$ ,  $\sinh(x)$ ,  $\tanh(x)$ , avec les bornes du repère graphique tels que  $x$  entre  $-4$  et  $4$  et  $y$  entre  $-4$  et  $4$ .

**Pour les fonctions discontinues ou non définies en certains points ou certains intervalles, pensez à les tracer sur différents intervalles où elles sont continues.**

**CR**

1. faites un copier-coller des différentes figures dans ce document : pour chaque fenêtre graphique faite avec Scilab, copiez le contenu dans le presse-papier (menu Fichier, item Copier dans le presse-papier) puis allez dans votre document texte LibreOffice et collez la figure à l'endroit voulu. Vous pouvez ensuite réduire la taille de la figure si vous le souhaitez.
2. copiez dans votre compte-rendu les instructions correspondant au tracé n° 9 des trois fonctions trigonométriques  $\cos$ ,  $\sin$  et  $\tan$ .

## 6 - Nombres en informatique

Dans cette partie, nous allons voir que les valeurs numériques (entiers ou réels) dans un logiciel informatique, présentent des limitations, et cela a des conséquences lorsqu'on fait des *calculs numériques* (c'est à dire des *calculs sur ordinateur*).

On verra le principe du codage des nombres entiers et des nombres *réels flottants*, et ce que cela implique en terme de précision notamment, et la manière de faire correctement (ou non) certains calculs.

### 6.1 - Principe de codage des valeurs numériques

En mathématiques, on utilise souvent des ensembles de nombres qui ont une infinité d'éléments. Tous les ensembles classiques  $\mathbb{N} \subset \mathbb{Z} \subset \mathbb{Q} \subset \mathbb{R}$  possèdent une infinité d'éléments (puisque'il y a une infinité de nombres naturels).

Or dès qu'on travaille avec un ordinateur, on a une limite finie (notamment à cause de la taille mémoire de l'ordinateur), donc on ne peut pas représenter tous les nombres, et cela a une conséquence sur les résultats de certains calculs.

Toute information manipulée sur un ordinateur est codée en *binnaire* (utilisation de 0 et de 1), notamment pour représenter et manipuler des valeurs numériques.

Le logiciel Scilab utilise des valeurs numériques codées en nombre flottant suivant une norme standard en informatique appelé *codage IEEE 754 - 64 bits*.

Pour simplifier, ce codage permet d'obtenir des valeurs réelles  $v$  à partir de trois entiers naturels  $S$  (*signe*),  $M$  (*mantisse*) et  $E$  (*exposant*) sous la forme suivante :

$$v = (-1)^S \times M \times 2^E$$

avec  $S$  égal à 0 ou 1,  $M$  entier naturel entre 0 et  $M_{max}$ , et  $E$  entier relatif entre  $E_{min} < 0$  et  $E_{max} > 0$ .

⇒ Ce codage permet aussi de représenter et manipuler les valeurs  $-\infty$  et  $+\infty$ , et permet par exemple de faire les calculs suivants :  $v + \infty$ ,  $w \times \infty$ ,  $v/\infty$ ,  $w/0$ ,  $\infty + \infty$ , avec  $v$  réel et  $w$  réel non nul.

Pour s'en convaincre, testez les instructions suivantes :

```
// %inf : constante Scilab correspondant à +infini
2 * %inf
1 / %inf
3 - %inf
%inf + %inf
```

Si un calcul est impossible à faire, Scilab indique **Nan** (*Not a number*).

Testez les instructions suivantes (correspondant à des formes indéterminées) :

```
%inf - %inf
%inf / %inf
0 / 0
0 * %inf
```

⇒ Scilab permet aussi de manipuler des nombres complexes, mais nous ne les utiliserons pas dans l'UE MAP101.



## 6.2 - Limitations des valeurs

- Le codage  $v = (-1)^S \times M \times 2^E$  utilisé par Scilab implique des limitations sur les valeurs numériques que l'on peut manipuler.

La valeur de  $S$  donne le signe de  $v$  :  $S = 0 \iff v \geq 0$  et  $S = 1 \iff v \leq 0$ .

La valeur  $M = 0$  est utilisée uniquement pour obtenir la valeur  $v = 0$ .

On va s'intéresser ensuite à des valeurs strictement positives (correspondant à  $S = 0$  et  $M > 0$ ), les limitations observées se transposent aux valeurs strictement négatives (correspondant à  $S = 1$  et  $M > 0$ ).

- Les limitations sont de deux types, et proviennent des limites sur les valeurs de  $M$  et  $E$ .

(a) *limites sur les ordres de grandeurs :*

Les valeurs de  $E_{min}$  et  $E_{max}$  donnent les ordres de grandeur minimal et maximal qu'on peut atteindre pour  $v$  :

- La plus petite valeur strictement positive qu'on peut obtenir avec Scilab correspond à  $M = 1$  et  $E = E_{min} = -1074$  :

$$v = 1 \times 2^{-1074} = 2^{-1074}$$

Testez les instructions suivantes :

```
2**(-1073)
2**(-1074)
2**(-1075)
```

- La plus grande valeur qu'on peut obtenir avec ce codage correspond à  $M = M_{max} = 2^{53} - 1$  et  $E = E_{max} = 971$  :

$$v = (2^{53} - 1) \times 2^{971}$$

Testez les instructions suivantes :

```
(2**53-1) * (2**971)
(2**53) * (2**971)
(2**53-1) * (2**972)
```

On voit qu'en ordre de grandeur exprimé en puissance de 10, ce codage permet de manipuler des valeurs strictement positives comprises entre (environ)  $10^{-323}$  et  $10^{+308}$ .

(b) *limites sur la précision :*

La valeur de  $M_{max}$  donne la précision qu'on peut utiliser. La précision correspond à l'écart entre deux valeurs strictement positives  $v_1$  et  $v_2$  consécutives qu'on peut représenter.

En général, on s'intéresse plutôt à la *précision relative* correspond au rapport entre la précision et la valeur  $v_1$  :

$$\text{précision relative} = \frac{|v_2 - v_1|}{v_1}$$

La précision relative dépend essentiellement du nombre total de possibilités pour les valeurs de  $M$ , et dans le cas de Scilab, la précision relative est égal à  $2^{-52} \simeq 2 \times 10^{-16}$ , soit environ 16 chiffres décimaux significatifs dans l'écriture (en base 10) d'une valeur.

Testez les instructions suivantes :

```
mprintf("%.20e\n", 1)
mprintf("%.20e\n", 1+2**(-54))
mprintf("%.20e\n", 1+2**(-53))
mprintf("%.20e\n", 1+2**(-52))
mprintf("%.20e\n", 1+10**(-16))
mprintf("%.20e\n", 1+10**(-15))
mprintf("%20.0f\n", 10**16)
mprintf("%20.0f\n", 10**16+1)
mprintf("%20.0f\n", 10**19)
mprintf("%20.0f\n", 10**19+23)
```

Cet exemple montre qu'en général, il est inutile de manipuler et d'afficher une valeur numérique avec plus de 16 chiffres significatifs.

Par exemple pour Scilab  $1 + 10^{-16}$  est égal à 1, ceci signifie que  $10^{-16}$  est numériquement *négligeable* par rapport à 1.

C'est aussi le cas pour 1 par rapport à  $10^{16}$ , ou bien 23 par rapport à  $10^{19}$ .

- Le codage utilisé ne permet pas de représenter de manière exacte certaines valeurs décimales.

**Exemple :** dans ce premier exemple, on va voir que les valeurs 0,3, 0,6 et 0,9 ne peuvent pas être représentées de manière exacte, et que cela induit des (petites) erreurs numériques sur des calculs simples.

Testez les instructions suivantes :

```
mprintf("%.60f\n", 0.3)
mprintf("%.60f\n", 0.6)
mprintf("%.60f\n", 0.9)
```

Les valeurs indiquées par l'instruction `mprintf` avec affichage de 60 décimales permet pour chaque valeur d'obtenir la valeur la plus proche qui a été codée par l'ordinateur.

Cela a alors une conséquence sur les calculs que l'on fait avec certaines valeurs.

Testez les instructions suivantes :

```
mprintf("%.5e\n", 0.9-0.6-0.3)
mprintf("%.5e\n", 0.9-0.3-0.6)
```

On pourrait s'attendre à obtenir à chaque fois la valeur nulle, ce qui n'est pas le cas, et de plus, les deux résultats sont différents.

**Exemple :** dans ce deuxième exemple, on va effectuer le processus suivant : on part avec la valeur  $s = 0$ , puis on va lui ajouter cent mille fois la valeur 0,3, puis lui retrancher dix mille fois la valeur 3.

Normalement on doit retrouver  $s = 0$  car  $0 + 100000 \times 0,3 - 10000 \times 3 = 0$ .

Avec l'éditeur de texte Scilab, créez le fichier suivant en le nommant `ex_numerique1.sce`, exécutez-le et observez le résultat :

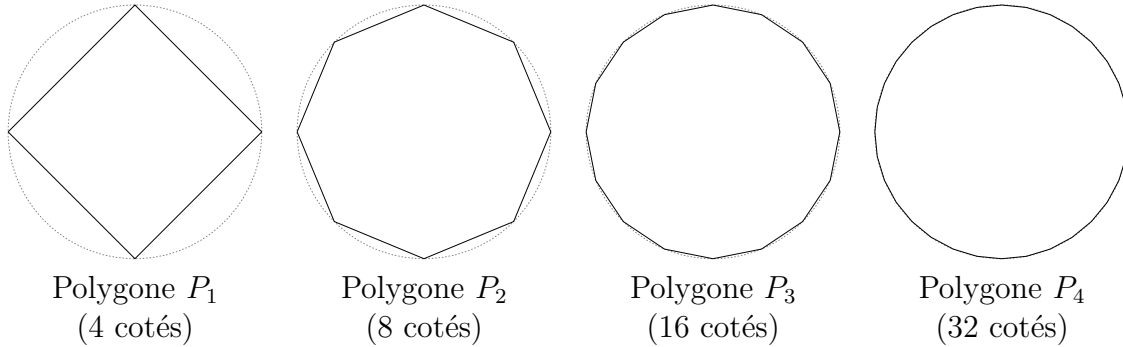
```
s = 0
for k=1:100000
    s = s+0.3
end
for k=1:10000
    s = s-3
end
mprintf("%.5e\n", s)
```

la valeur de  $s$  n'est pas nulle.

Cet exemple simple montre le problème d'accumulation des *erreurs numériques*.

**Exemple :** dans ce dernier exemple, on va calculer une approximation du nombre  $\pi$  à l'aide d'une suite de valeurs censée converger vers  $\pi$ .

La valeur  $\pi$  est le périmètre d'un cercle de diamètre 1, et pour avoir une approximation de  $\pi$ , on peut considérer les périmètres de polygones réguliers inscrits dans un cercle de diamètre 1.



On considère la suite de polygones réguliers  $P_n$ ,  $P_n$  polygone régulier avec  $2^{n+1}$  cotés, en commençant par  $P_1$  (carré inscrit dans le cercle), et en passant de  $P_n$  à  $P_{n+1}$  en doublant le nombre de cotés. On note  $v_n$  le périmètre de  $P_n$ .

La suite de polygones  $P_n$  tend vers le cercle unité et donc la suite des valeurs des périmètres  $v_n$  tend vers la valeur  $\pi$ .

On peut montrer que :  $v_1 = 2\sqrt{2}$  et  $\forall n \in \mathbb{N}, v_{n+1} = 2^{n+1} \times \sqrt{2 - \sqrt{4 - \left(\frac{v_n}{2^n}\right)^2}}$ .

En effet, pour tout  $n$ ,  $v_n = 2^{n+1} \sin\left(\frac{\pi}{2^{n+1}}\right)$ .

Posons  $a_n = \frac{\pi}{2^{n+1}}$  et  $u_n = \sin(a_n)$ , on a alors  $u_n = \frac{v_n}{2^{n+1}}$  et  $a_n = 2a_{n+1}$ .

On a aussi  $\sin^2(a_n) = 1 - \cos^2(a_n)$  et  $\cos(a_n) = \cos(2a_{n+1}) = 1 - 2\sin^2(a_{n+1})$

$$\Rightarrow \sqrt{1 - \sin^2(a_n)} = \cos(a_n) = 1 - 2\sin^2(a_{n+1})$$

$$\Rightarrow \sin(a_{n+1}) = \sqrt{\frac{1 - \sqrt{1 - \sin^2(a_n)}}{2}} = \sqrt{\frac{2 - 2\sqrt{1 - \sin^2(a_n)}}{4}} = \frac{1}{2}\sqrt{2 - \sqrt{4 - 4\sin^2(a_n)}}$$

Comme  $\sin(a_{n+1}) = u_{n+1} = \frac{v_{n+1}}{2^{n+2}}$  et  $4\sin^2(a_n) = (2\sin(a_n))^2 = (2u_n)^2 = \left(\frac{v_n}{2^n}\right)^2$ , on obtient :

$$\frac{v_{n+1}}{2^{n+2}} = u_{n+1} = \frac{1}{2}\sqrt{2 - \sqrt{4 - 4u_n^2}} = \frac{1}{2}\sqrt{2 - \sqrt{4 - \left(\frac{v_n}{2^n}\right)^2}}$$

$$\Leftrightarrow v_{n+1} = 2^{n+1} \times \sqrt{2 - \sqrt{4 - \left(\frac{v_n}{2^n}\right)^2}}$$

Avec l'éditeur de texte Scilab, créez le fichier suivant en le nommant `ex_suite_pi.sce`, exécutez-le et observez le résultat :

```
vn = 2*sqrt(2);
for n=1:30
    mprintf("n = %2d , v(%2d) = %20.15f\n", n, n, vn);
    vn = 2**(n+1)*sqrt(2-sqrt(4-(vn/2**n)**2));
end
```

Cette boucle calcule et écrit les 30 premiers termes de la suite  $v_n$ .

Les valeurs  $v_n$  s'approchent de  $\pi$  puis continuent à croître jusqu'à 4 et deviennent ensuite nulles.

L'explication est la suivante : le terme  $\left(\frac{v_n}{2^n}\right)^2$  va tendre vers 0 donc numériquement va devenir négligeable par rapport à 4, et à partir d'une certaine valeur de  $n$

$$2^{n+1} \times \sqrt{2 - \sqrt{4 - \left(\frac{v_n}{2^n}\right)^2}} \simeq 2^{n+1} \times \sqrt{2 - \sqrt{4 - 0}} = 2^{n+1} \times \sqrt{2 - 2} = 2^{n+1} \times 0 = 0$$

Pour éviter ce problème, il faut modifier la formule de récurrence entre  $v_n$  et  $v_{n+1}$  ainsi :

$$v_{n+1} = 2^{n+1} \times \sqrt{2 - \sqrt{4 - \left(\frac{v_n}{2^n}\right)^2}} = \frac{2^{n+1} \times \sqrt{2 - \sqrt{4 - \left(\frac{v_n}{2^n}\right)^2}} \times \sqrt{2 + \sqrt{4 - \left(\frac{v_n}{2^n}\right)^2}}}{\sqrt{2 + \sqrt{4 - \left(\frac{v_n}{2^n}\right)^2}}} \quad (1)$$

### Exercice 6 :

Simplifiez le **plus possible** le **numérateur** de la formule (1) précédente, afin de trouver une nouvelle expression pour  $v_{n+1}$  en fonction de  $v_n$  puis modifiez le script `ex_suite_pi.sce` en conséquence, et exécutez-le.

**CR** Dans votre compte-rendu, indiquez :

- le numérateur simplifié de la formule (1),
- la modification faite dans le script `ex_suite_pi.sce`,
- la valeur de  $v_{30}$  obtenue (écrire cette valeur avec 15 chiffres après la virgule).

• • •

**CR** Une fois votre compte-rendu terminé, indiquez au début du document les noms et prénoms de votre binôme, ainsi que votre groupe, sauvegardez-le puis exportez-le au format PDF (menu *Fichier*, item *Exporter au format PDF*) puis envoyez ce fichier PDF par e-mail à votre enseignant de TP en indiquant comme sujet du message :

[MAP101] - Compte-rendu TP Scilab - noms du binôme - groupe

# Zéro(s) de fonction

## séance préparatoire

### 1 - Présentation

Dans ce thème, on s'intéresse à un problème de base en calcul numérique qu'est la recherche de zéro(s) d'une fonction  $f$ , i.e. connaissant  $f$  (par son expression analytique), trouver un réel  $x$  telle que  $f(x) = 0$ , la fonction  $f$  présentant certaines "bonnes" propriétés, notamment la continuité.

En général avec un ordinateur, tous les réels ne pouvant être représentés de manière exacte, on cherche plutôt une valeur  $\bar{x}$  tel que  $f(\bar{x})$  est proche de 0.

Dans ce thème, on va voir différentes méthodes algorithmiques, permettant de trouver une telle valeur de  $\bar{x}$ .

Bien évidemment, pour certaines fonctions, il n'existe pas de solution au problème  $f(x) = 0$ . Par exemple pour la fonction  $f(x) = x^2 + 1$ , il n'existe aucun réel  $x$  tel que  $f(x) = 0$  (car pour tout réel  $x$ ,  $f(x) = x^2 + 1 \geq 1 > 0$ ).

Pour d'autres fonctions, il peut y avoir plusieurs solutions voire une infinité de solutions. Par exemple la fonction  $f$  définie par  $f(x) = \cos(x) - 1/2$  possède une infinité de solutions, car  $x = \pi/3$  est une solution de  $f(x) = 0$  alors  $x = \pi/3 + 2k\pi$  (avec  $k$  un entier) est aussi une solution de ce problème. Pour cette fonction  $f$ , on pourrait alors se limiter à chercher  $x$  dans l'intervalle  $[0, \pi]$ .

On va donc faire certaines hypothèses sur la fonction  $f$  ainsi que sur la valeur de  $x$ .

### 2 - Hypothèse générale

On suppose que la fonction  $f$  est continue sur un intervalle  $I = [a, b]$  ( $a$  et  $b$  sont aussi connus), et  $f$  change de signe sur  $I$  c'est à dire

$$\left\{ \left\{ f(a) < 0 \text{ ET } f(b) > 0 \right\} \text{ OU } \left\{ f(a) > 0 \text{ ET } f(b) < 0 \right\} \right\} \iff f(a)f(b) < 0$$

Si on a ces hypothèses, alors le théorème des valeurs intermédiaires nous dit qu'il existe  $x \in ]a, b[$  tel que  $f(x) = 0$ .

De plus si  $f$  est strictement monotone sur  $I$  alors cette valeur  $x \in ]a, b[$  est unique.

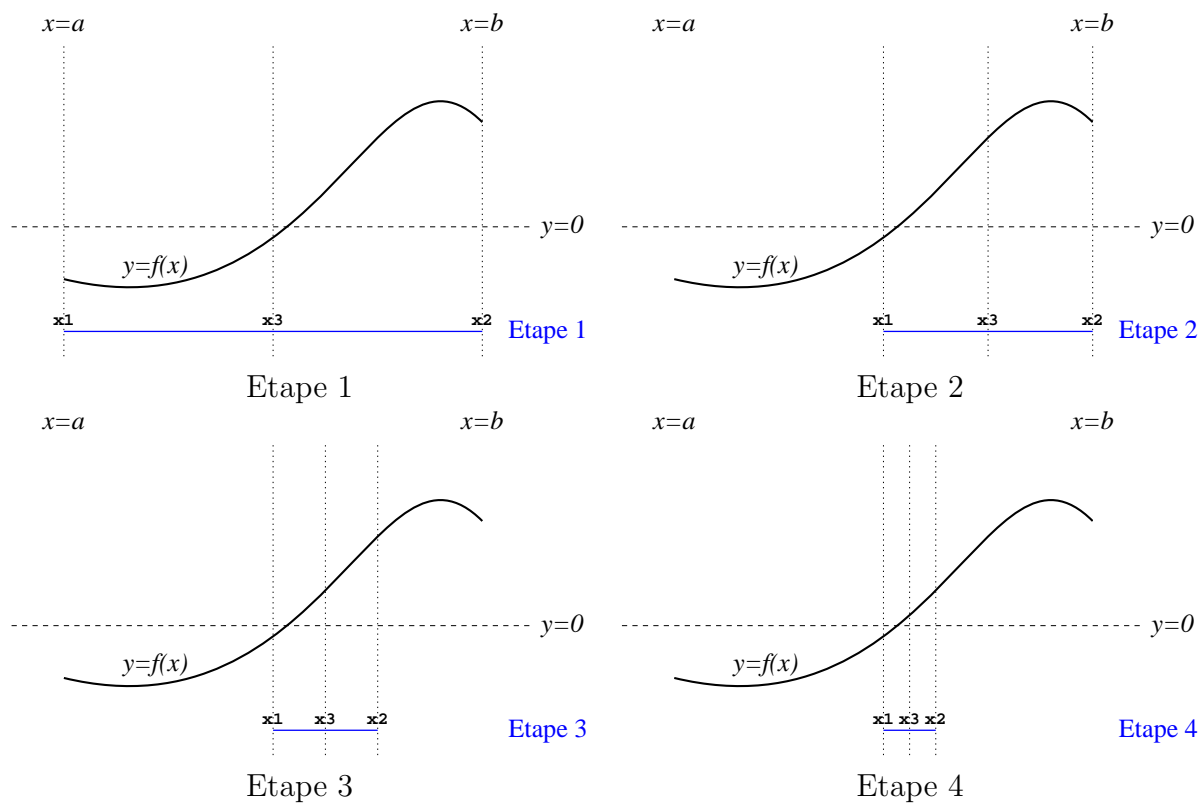
Pour les deux premières méthodes présentées, il n'y a pas de condition particulière sur la monotonie, par contre la troisième méthode nécessite des conditions supplémentaires sur la fonction.

### 3 - Méthode par dichotomie

Cette première méthode correspond à la méthode utilisée dans la démonstration du théorème des valeurs intermédiaires.

Le principe est de partir de l'intervalle  $[x_1, x_2] = [a, b]$  puis par itérations successives, réduire de moitié l'intervalle  $[x_1, x_2]$  sur lequel la fonction change de signe jusqu'à ce que la valeur  $|f(x_1)|$  ou la valeur  $|f(x_2)|$  soit proche de 0 avec une précision souhaitée  $\varepsilon$ .

Les figures ci-dessous montrent le principe de cette méthode sur un exemple.



Connaissant :

- une fonction  $f$ ,
- deux valeurs  $a$  et  $b$  (tel que  $f(x)$  change de signe sur l'intervalle  $[a, b]$ ),
- un réel  $\varepsilon > 0$  définissant la précision voulue

l'algorithme s'écrit ainsi :

```

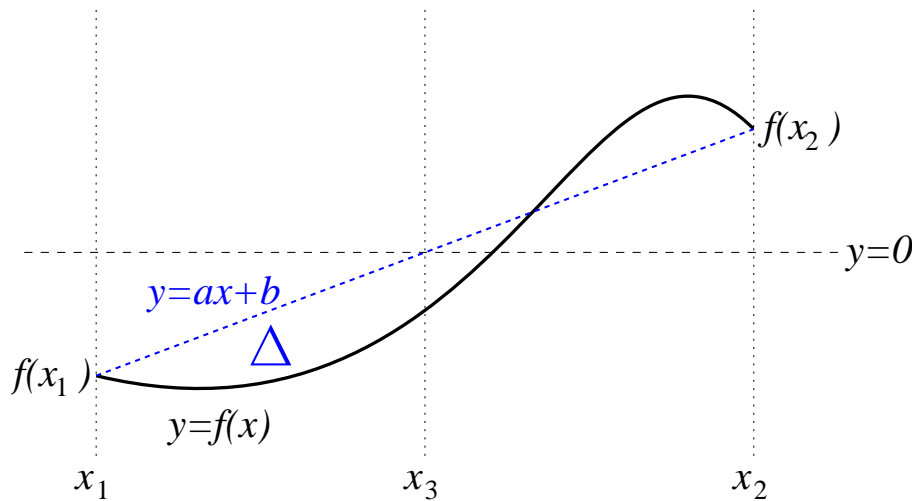
// initialisation
x1 ← a
x2 ← b
fx1 ← f(x1) // valeur de f en x1
fx2 ← f(x2) // valeur de f en x2
n ← 0 // numéro de l'étape
// boucle réduisant l'intervalle I = [x1, x2] en conservant le changement de signe sur I
tant_que |fx1| ≥ ε et |fx2| ≥ ε faire
|   x3 ← (x1 + x2)/2 // x3 milieu de l'intervalle I
|   fx3 ← f(x3) // valeur de f en x3
|   si fx1 × fx3 < 0 alors
|   |   // réduire l'intervalle à [x1, x3]
|   |   x2 ← x3 // x2 prend la valeur de x3
|   |   fx2 ← fx3 // fx2 prend la valeur de fx3
|   sinon
|   |   // réduire l'intervalle à [x3, x2]
|   |   x1 ← x3 // x1 prend la valeur de x3
|   |   fx1 ← fx3 // fx1 prend la valeur de fx3
|   fin_si
|   n ← n + 1 // passer à l'étape suivante
fin_tant_que
si |fx1| < ε alors
|   xbar ← x1
sinon
|   xbar ← x2
fin_si

```

La valeur finale  $\bar{x} = \text{xbar}$  vérifie  $|f(\bar{x})| < \varepsilon$ .

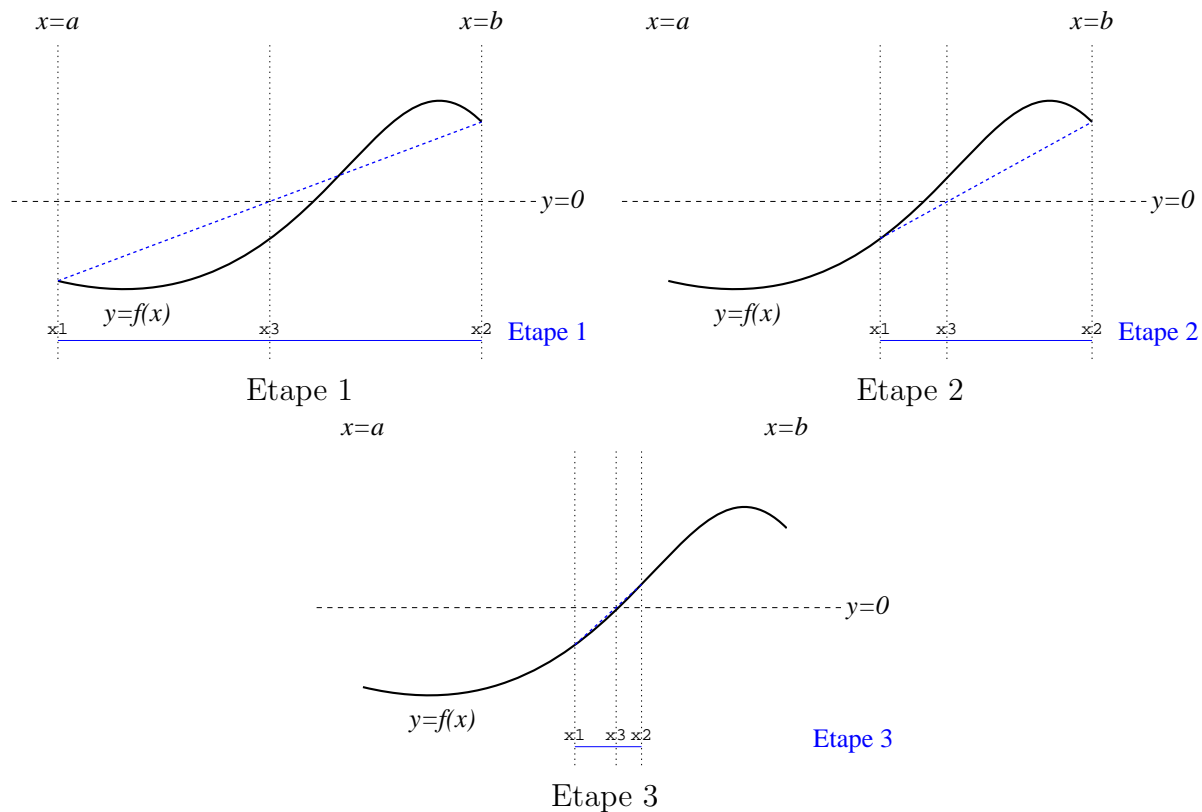
## 4 - Méthode de la fausse position

Cette méthode est une variante de la méthode par dichotomie : à chaque étape, pour déterminer l'abscisse  $x_3$ , au lieu de choisir  $(x_1 + x_2)/2$ , on considère la droite  $\Delta$  passant par les points  $(x_1, f(x_1))$  et  $(x_2, f(x_2))$  puis on détermine le point  $(x_3, 0)$  intersection de la droite  $\Delta$  avec l'axe des abscisses  $\{y = 0\}$ .



Une étape de la méthode de la *fausse position*

Les figures ci-dessous montrent le principe de cette méthode sur un exemple.





**Exercice 1 :**

L'algorithme de la fausse position s'écrit de la même manière que la méthode par dichotomie, la seule différence est la formule donnant l'abscisse  $x_3$ .

Pour cela il faut déterminer l'équation  $y = ax + b$  de la droite passant par les deux points  $(x_1, f(x_1))$  et  $(x_2, f(x_2))$  puis déterminer l'intersection de cette droite avec l'axe horizontal  $y = 0$ .

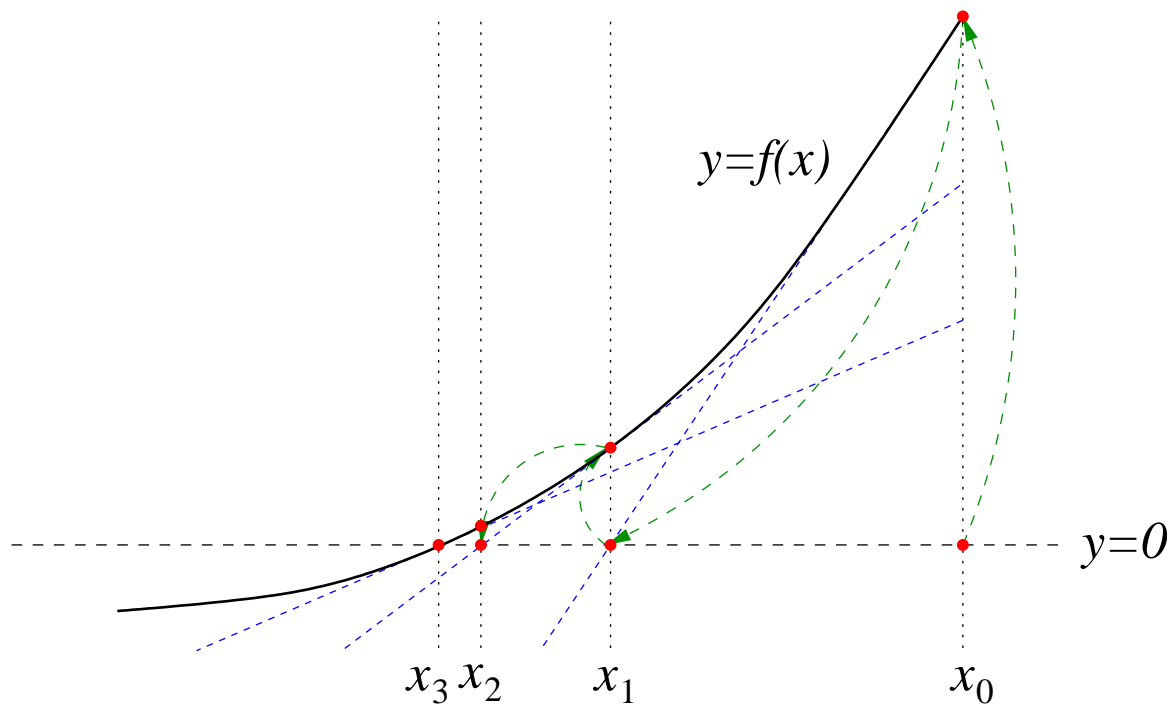
1. Déterminer les expressions des deux coefficients  $a$  et  $b$  à partir de  $x_1$ ,  $x_2$ ,  $f(x_1)$  et  $f(x_2)$ .
2. En déduire la valeur de  $x_3$  tel que  $ax_3 + b = 0$  en fonction de  $x_1$ ,  $x_2$ ,  $f(x_1)$  et  $f(x_2)$ .

**5 - Méthode de Newton**

Cette méthode fonctionne différemment des deux précédentes, le principe est le suivant pour une fonction  $f$  :

- on choisit une abscisse initiale  $x_0$ ,
- à l'étape  $n$ , on calcule la nouvelle abscisse  $x_{n+1}$  à partir de  $x_n$  en considérant la droite tangente à la courbe représentative de  $f$  au point  $(x_n, f(x_n))$ , et on cherche le point  $(x_{n+1}, 0)$  intersection de la droite tangente avec l'axe des abscisses  $\{y = 0\}$  ;
- on remplace  $x_n$  par  $x_{n+1}$  ;
- on s'arrête lorsque  $|f(x_n)| < \varepsilon$  ;
- la valeur  $\bar{x}$  est alors égale à  $x_n$ .

La figure ci-dessous montre le principe de la méthode.



Trois étapes de la méthode de Newton

Comme pour les méthodes précédentes, il faut supposer que la fonction  $f$  admet un zéro (il existe  $x$  tel que  $f(x) = 0$ ), mais il faut des hypothèses supplémentaires sur la fonction  $f$  car la dérivée  $f'$  de la fonction ne doit jamais s'annuler afin que toute droite tangente puisse intersecter l'axe des abscisses.

De plus, pour assurer que la suite  $x_n$  converge, il faut rajouter une hypothèse supplémentaire sur  $f''$  dérivée seconde de  $f$ .

Donc pour cette méthode, on suppose pour la fonction  $f$ , qu'on connaît un intervalle  $[a, b]$  tel que  $f(a) < 0 < f(b)$  et  $\forall x \in [a, b]$ ,  $f'(x) > 0$  et  $f''(x) \geq 0$  (la fonction  $f$  est strictement croissante et *convexe* sur l'intervalle  $[a, b]$ ).

Une étape consiste à calculer  $x_{n+1}$  à partir de  $x_n$  ainsi :

- la droite tangente à la courbe représentative de  $f$  au point  $(x_n, f(x_n))$  a pour équation  $y = f(x_n) + f'(x_n)(x - x_n)$
- cette droite intersecte l'axe des abscisses  $y = 0$  pour  $x_{n+1}$  vérifiant

$$f(x_n) + f'(x_n)(x_{n+1} - x_n) = 0 \iff x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

Connaissant :

- une fonction  $f$ ,
- deux valeurs  $a$  et  $b$  (tel que  $f(x)$  change de signe sur l'intervalle  $[a, b]$ ),
- un réel  $\varepsilon > 0$  définissant la précision voulue

l'algorithme s'écrit ainsi :

```
// initialisation
xn ← b
n ← 0 // numéro de l'étape
// boucle des différentes étapes
tant_que  $|f(xn)| \geq \varepsilon$  faire
|   xn ← xn -  $\frac{f(xn)}{f'(xn)}$ 
|   n ← n + 1
fin_tant_que
xbar ← xn
```

Pour le TP sur les zéros de fonctions, préparer l'exercice 1 (page 33) qui devra être inclus dans le compte-rendu du TP.

# Zéro(s) de fonction

## séance pratique

Séance pratique (3h) à faire en binôme.

**Pour ce thème, le compte rendu (en binôme) devra être fait sous forme manuscrite (ou éventuellement imprimé) et rendu impérativement à la fin de la séance de 3h.**

Dans ce TP, on utilisera deux fonctions :

$$f_1(x) = x^6 + x^4 + x^2 + 13x - 8 \text{ sur l'intervalle } I_1 = [-1, 1],$$

$$f_2(x) = 3e^x + 6x^4 + x - 4 \text{ sur l'intervalle } I_2 = [0, 1].$$

### Exercice 1 :

**CR** *Le but de cet exercice est de montrer que pour chacune des fonctions avec son intervalle, les trois méthodes peuvent être appliquées.*

1. Pour la fonction  $f_1(x) = x^6 + x^4 + x^2 + 13x - 8$  et l'intervalle  $[a, b] = I_1 = [-1, 1]$ , montrer que  $f(a) < 0 < f(b)$ , calculer  $f'(x)$  et  $f''(x)$  et montrer que  $f'(x) > 0$  et  $f''(x) \geq 0$  pour tout  $x \in [a, b]$ .
2. Même exercice avec la fonction  $f_2(x) = 3e^x + 6x^4 + x - 4$  et l'intervalle  $[a, b] = I_2 = [0, 1]$ .

### Exercice 2 :

1. Avec Scilab, tracez la fonction  $f_1$  sur l'intervalle  $I_1$  afin de vérifier qu'elle possède bien un zéro sur cet intervalle.
2. Même exercice avec la fonction  $f_2$  sur l'intervalle  $I_2$ .

## 1 - Méthode par dichotomie

### Exercice 3 :

1. Ecrivez un fichier-script nommé `tp_zf_dichotomie.sce` qui effectue les opérations suivantes :
  - (a) définit la fonction  $f(x) = f_1(x) = x^6 + x^4 + x^2 + 13x - 8$ ,
  - (b) définit l'intervalle  $I_1 = [a, b] = [-1, 1]$  (tel que  $f(a)f(b) < 0$ ),
  - (c) demande à l'utilisateur d'entrer la valeur  $\varepsilon > 0$ ,
  - (d) effectue l'algorithme de la méthode par dichotomie,
  - (e) puis affiche à la fin le nombre d'étapes  $n$  ainsi que les valeurs de  $\bar{x}$  et  $|f(\bar{x})|$ .  
Affichez  $n$  comme un entier, affichez  $\bar{x}$  avec 12 chiffres après la virgule, et affichez  $|f(\bar{x})|$  au format scientifique :

```
mprintf("n = %4d \n", n)
mprintf("xbar = %15.12f \n", xbar)
mprintf("|f(xbar)| = %10.3e \n", abs(f(xbar)))
```

2. Testez ensuite le script en notant dans le tableau ci-dessous les valeurs de  $n$ ,  $\bar{x}$  et  $|f(\bar{x})|$  obtenues pour la précision  $\varepsilon$ .

fonction  $f(x) = x^6 + x^4 + x^2 + 13x - 8$  avec  $a = -1$  et  $b = 1$

$\varepsilon$	$10^{-2}$	$10^{-4}$	$10^{-6}$	$10^{-8}$	$10^{-10}$
$n$					
$\bar{x}$					
$ f(\bar{x}) $					

3. Modifiez le script afin de tester la fonction  $f(x) = f_2(x)$  avec  $a = 0$  et  $b = 1$ , et complétez le tableau suivant

fonction  $f(x) = f_2(x) = 3e^x + 6x^4 + x - 4$  avec  $a = 0$  et  $b = 1$

$\varepsilon$	$10^{-2}$	$10^{-4}$	$10^{-6}$	$10^{-8}$	$10^{-10}$
$n$					
$\bar{x}$					
$ f(\bar{x}) $					

4. Pour chaque fonction, qu'observe-t-on pour  $n$  et  $|f(\bar{x})|$  lorsque la valeur  $\varepsilon$  diminue ?

## 2 - Méthode de la fausse position

### Exercice 4 :

1. A partir du script `tp_zf_dichotomie.sce` que vous avez écrit précédemment, créez un nouveau script nommé `tp_zf_fausse_pos.sce` afin d'implémenter la méthode de la *fausse position*.
2. Testez ensuite le script en notant dans le tableau ci-dessous les valeurs de  $n$  et  $\bar{x}$  obtenues pour la précision  $\varepsilon$ .

fonction  $f(x) = f_1(x) = x^6 + x^4 + x^2 + 13x - 8$  avec  $a = -1$  et  $b = 1$

$\varepsilon$	$10^{-2}$	$10^{-4}$	$10^{-6}$	$10^{-8}$	$10^{-10}$
$n$					
$\bar{x}$					
$ f(\bar{x}) $					

3. Modifiez le script afin de tester la fonction  $f(x) = f_2(x)$  avec  $a = 0$  et  $b = 1$ , et complétez le tableau suivant

fonction  $f(x) = f_2(x) = 3e^x + 6x^4 + x - 4$  avec  $a = 0$  et  $b = 1$

$\varepsilon$	$10^{-2}$	$10^{-4}$	$10^{-6}$	$10^{-8}$	$10^{-10}$
$n$					
$\bar{x}$					
$ f(\bar{x}) $					

### 3 - Méthode de Newton

#### Exercice 5 :

1. En s'inspirant des scripts précédents `tp_zf_dichotonie.sce` et `tp_zf_fausse_pos.sce`, créer un nouveau script nommé `tp_zf_newton.sce` afin d'implémenter la méthode de *Newton*.  
Dans le script de cette méthode, en plus de la définition de la fonction  $f$ , il faudra définir une deuxième fonction correspondant à la dérivée de  $f$ .
2. Testez ensuite le script en notant dans le tableau ci-dessous les valeurs de  $n$  et  $\bar{x}$  obtenues pour la précision  $\varepsilon$ .

fonction  $f(x) = x^6 + x^4 + x^2 + 13x - 8$  avec  $x_0 = 1$

$\varepsilon$	$10^{-2}$	$10^{-4}$	$10^{-6}$	$10^{-8}$	$10^{-10}$
$n$					
$\bar{x}$					
$ f(\bar{x}) $					

3. Modifiez le script afin de tester la fonction  $f_2$  avec  $a = 0$  et  $b = 1$ , et complétez le tableau suivant

fonction  $f(x) = 3e^x + 6x^4 + x - 4$  avec  $x_0 = 1$

$\varepsilon$	$10^{-2}$	$10^{-4}$	$10^{-6}$	$10^{-8}$	$10^{-10}$
$n$					
$\bar{x}$					
$ f(\bar{x}) $					

## 4 - Comparatif des 3 méthodes

### Exercice 6 :

Le but de cet exercice est de noter le nombre  $n$  d'itérations nécessaires pour une méthode donnée et une valeur d'erreur  $\varepsilon$  donnée. Puis, pour une méthode donnée, observez comment le nombre d'itérations  $n$  évolue avec la valeur de  $\varepsilon$ , et pour conclure, comparez les trois méthodes.

Pour les trois méthodes (1) *par dichotomie*, (2) *fausse position*, (3) *Newton*, remplissez les tableaux suivant les valeurs obtenues pour le nombre d'itérations  $n$  pour les deux fonctions-tests et différentes valeurs de  $\varepsilon$  :

$$\text{fonction } f_1(x) = x^6 + x^4 + x^2 + 13x - 8$$

$\varepsilon$	$10^{-2}$	$10^{-4}$	$10^{-6}$	$10^{-8}$	$10^{-10}$
$n$ (1)					
$n$ (2)					
$n$ (3)					

$$\text{fonction } f_2(x) = 3e^x + 6x^4 + x - 4$$

$\varepsilon$	$10^{-2}$	$10^{-4}$	$10^{-6}$	$10^{-8}$	$10^{-10}$
$n$ (1)					
$n$ (2)					
$n$ (3)					

**CR** Dans votre compte-rendu, recopier les deux tableaux ci-dessus complétés, puis au vu des résultats ci-dessus, ajoutez un commentaire pour comparer les méthodes suivant les valeurs de  $n$  obtenues.

# Interpolation

## séance préparatoire

### 1 - Présentation

Dans ce thème, on s'intéresse à un thème important en calcul numérique qu'est l'interpolation : on a des points de données  $(\mathbf{tt}_k, \mathbf{yy}_k)$  dans le plan avec les abscisses  $\mathbf{tt}_k$  distinctes entre elles, et on cherche une fonction  $y = f(t)$  passant par tous les points de données.

Par exemple, on a des mesures effectuées au cours du temps, c'est à dire à chaque pas de temps  $\mathbf{tt}_k$  on a une mesure associée  $\mathbf{yy}_k$  qui est un réel,  $\mathbf{yy}_k$  pouvant représenter une grandeur physique, par exemple une température, une vitesse, une longueur, ...

Le principe de l'interpolation est de pouvoir estimer cette grandeur physique pour tout instant  $t$  dans un certain intervalle de temps  $I$  à partir de  $N$  mesures  $(\mathbf{tt}_k, \mathbf{yy}_k)$ ,  $k$  variant entre 1 et  $N$ . Les valeurs  $\mathbf{tt}_k$  forment une suite strictement croissante  $(\mathbf{tt}_1 < \mathbf{tt}_2 < \dots < \mathbf{tt}_k < \mathbf{tt}_{k+1} < \dots < \mathbf{tt}_{N-1} < \mathbf{tt}_N)$  et on note  $I$ , l'intervalle  $I = [\mathbf{tt}_1, \mathbf{tt}_N]$ .

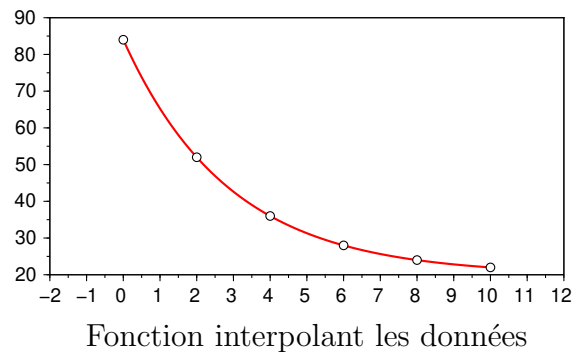
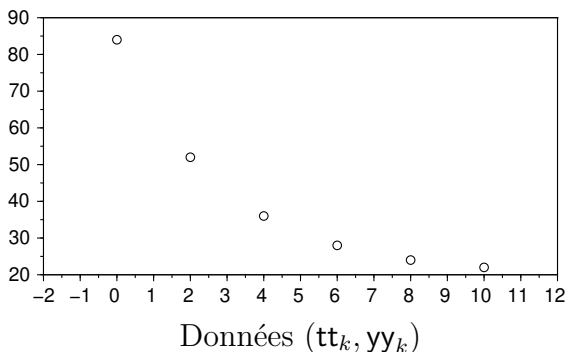
Pour cela on cherche une fonction  $f$  définie sur l'intervalle  $I$ , fonction au moins continue sur  $I$ , et telle que pour chaque mesure  $(\mathbf{tt}_k, \mathbf{yy}_k)$  on ait  $f(\mathbf{tt}_k) = \mathbf{yy}_k$  (*conditions d'interpolation*), pour pouvoir ensuite estimer pour une valeur de temps  $t$  quelconque, la grandeur correspondante  $y = f(t)$ . Par la suite, on déterminera donc toujours une fonction  $f$  continue sur  $I$ .

**Exemple :** on a mesuré la température d'une pièce de métal à différents instants :

indice $k$ de la mesure	1	2	3	4	5	6
temps $\mathbf{tt}_k$ (en minutes)	0	2	4	6	8	10
température $\mathbf{yy}_k$ (en degrés Celsius)	84	52	36	28	24	22

Pour ces données, on peut par exemple choisir la fonction  $f(t) = 2^{6-t/2} + 20$  qui *interpole* les données.

La figure de gauche ci-dessous montre une représentation graphique des données  $(\mathbf{tt}_k, \mathbf{yy}_k)$  et la figure de droite les données avec le graphe de la fonction  $f$ .



On peut ensuite estimer la grandeur  $y$  pour n'importe quelle valeur du temps  $t$  entre 0 et 10 minutes, par exemple pour  $t = 5$  minutes, on peut estimer que la température de la pièce est  $f(5) = 2^{6-5/2} + 20 \simeq 31,3$ .

Dans ce thème, nous allons voir quelques méthodes simples d'interpolation que vous aurez à programmer en Scilab, puis tester et comparer.

L'exemple ci-dessus est un cas d'école car la fonction  $f(t)$  qui interpole les données a une expression unique. En général, ce n'est pas le cas, et il est nécessaire de déterminer une expression particulière pour  $f$  pour chaque intervalle  $[\mathbf{tt}_k, \mathbf{tt}_{k+1}]$  : on dit que la fonction  $f$  est définie *par intervalle* ou *par morceaux*.

## 2 - Méthodes d'interpolation

### 2.1 - Interpolation linéaire par intervalle (linéaire par morceaux)

C'est la méthode la plus simple pour obtenir une fonction interpolante  $f$  continue sur l'intervalle  $I = [\mathbf{tt}_1, \mathbf{tt}_N]$ .

Le principe est de considérer les données par paires consécutives  $\{ (\mathbf{tt}_k, \mathbf{yy}_k) ; (\mathbf{tt}_{k+1}, \mathbf{yy}_{k+1}) \}$ . Sur l'intervalle  $[\mathbf{tt}_k, \mathbf{tt}_{k+1}]$ , on cherche la fonction  $f$  tel que le graphe de la fonction corresponde au segment (de droite) passant par les deux points  $(\mathbf{tt}_k, \mathbf{yy}_k)$  et  $(\mathbf{tt}_{k+1}, \mathbf{yy}_{k+1})$  : la fonction  $f$  est unique et s'écrit  $y = f(t) = a t + b$ .

Cette méthode est aussi appelée *interpolation affine par morceaux*.

#### Exercice 1 :

sur l'intervalle  $[\mathbf{tt}_k, \mathbf{tt}_{k+1}]$  avec  $1 \leq k \leq N-1$ , déterminer (à partir de  $\mathbf{tt}_k, \mathbf{tt}_{k+1}, \mathbf{yy}_k, \mathbf{yy}_{k+1}$ ) les expressions des coefficients  $a$  et  $b$  tel que  $y = f(t) = a t + b$  vérifie  $\begin{cases} f(\mathbf{tt}_k) = \mathbf{yy}_k \\ f(\mathbf{tt}_{k+1}) = \mathbf{yy}_{k+1} \end{cases}$

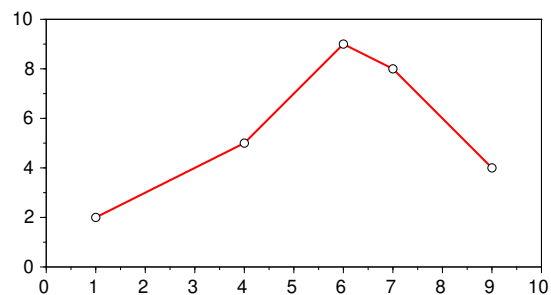
Bien évidemment l'expression de la fonction est valable uniquement pour  $t \in [\mathbf{tt}_k, \mathbf{tt}_{k+1}]$ . Pour connaître la totalité de la fonction sur l'intervalle  $I = [\mathbf{tt}_1, \mathbf{tt}_N]$ , il faut déterminer les expressions de la fonction sur chacun des  $N-1$  intervalles  $[\mathbf{tt}_k, \mathbf{tt}_{k+1}]$  avec  $k$  entre 1 et  $N-1$ .

#### Exercice 2 :

Pour les 5 données suivantes

$k$	1	2	3	4	5
$\mathbf{tt}_k$	1	4	6	7	9
$\mathbf{yy}_k$	2	5	9	8	4

- 1) déterminer les expressions de  $f(t)$  sur les différents intervalles.
- 2) quelle est la valeur de  $f(3)$  ?
- 3) quelle est la valeur de  $f(8)$  ?



Fonction  $f$  interpolant les données



## 2.2 - Interpolation parabolique

Cette deuxième méthode consiste à interpoler 3 données  $(\mathbf{tt}_1, \mathbf{yy}_1)$ ,  $(\mathbf{tt}_2, \mathbf{yy}_2)$  et  $(\mathbf{tt}_3, \mathbf{yy}_3)$  avec  $\mathbf{tt}_1 < \mathbf{tt}_2 < \mathbf{tt}_3$  par une unique fonction  $f(t) = at^2 + bt + c$  avec  $t \in I = [\mathbf{tt}_1, \mathbf{tt}_3]$ .

En écrivant les trois *contraintes d'interpolation*, on obtient les valeurs des trois coefficients  $a$ ,  $b$  et  $c$  :

$$\left\{ \begin{array}{l} f(\mathbf{tt}_1) = \mathbf{yy}_1 \\ f(\mathbf{tt}_2) = \mathbf{yy}_2 \\ f(\mathbf{tt}_3) = \mathbf{yy}_3 \end{array} \right\} \Longleftrightarrow \left\{ \begin{array}{l} a \mathbf{tt}_1^2 + b \mathbf{tt}_1 + c = \mathbf{yy}_1 \quad (1) \\ a \mathbf{tt}_2^2 + b \mathbf{tt}_2 + c = \mathbf{yy}_2 \quad (2) \\ a \mathbf{tt}_3^2 + b \mathbf{tt}_3 + c = \mathbf{yy}_3 \quad (3) \end{array} \right\}$$

$$\left\{ \begin{array}{l} (2) - (1) : a(\mathbf{tt}_2^2 - \mathbf{tt}_1^2) + b(\mathbf{tt}_2 - \mathbf{tt}_1) = \mathbf{yy}_2 - \mathbf{yy}_1 \quad (4) \\ (3) - (2) : a(\mathbf{tt}_3^2 - \mathbf{tt}_2^2) + b(\mathbf{tt}_3 - \mathbf{tt}_2) = \mathbf{yy}_3 - \mathbf{yy}_2 \quad (5) \end{array} \right\}$$

$$\left\{ \begin{array}{l} (4) \times \frac{1}{\mathbf{tt}_2 - \mathbf{tt}_1} : a(\mathbf{tt}_2 + \mathbf{tt}_1) + b = \frac{\mathbf{yy}_2 - \mathbf{yy}_1}{\mathbf{tt}_2 - \mathbf{tt}_1} \quad (6) \\ (5) \times \frac{1}{\mathbf{tt}_3 - \mathbf{tt}_2} : a(\mathbf{tt}_3 + \mathbf{tt}_2) + b = \frac{\mathbf{yy}_3 - \mathbf{yy}_2}{\mathbf{tt}_3 - \mathbf{tt}_2} \quad (7) \end{array} \right\}$$

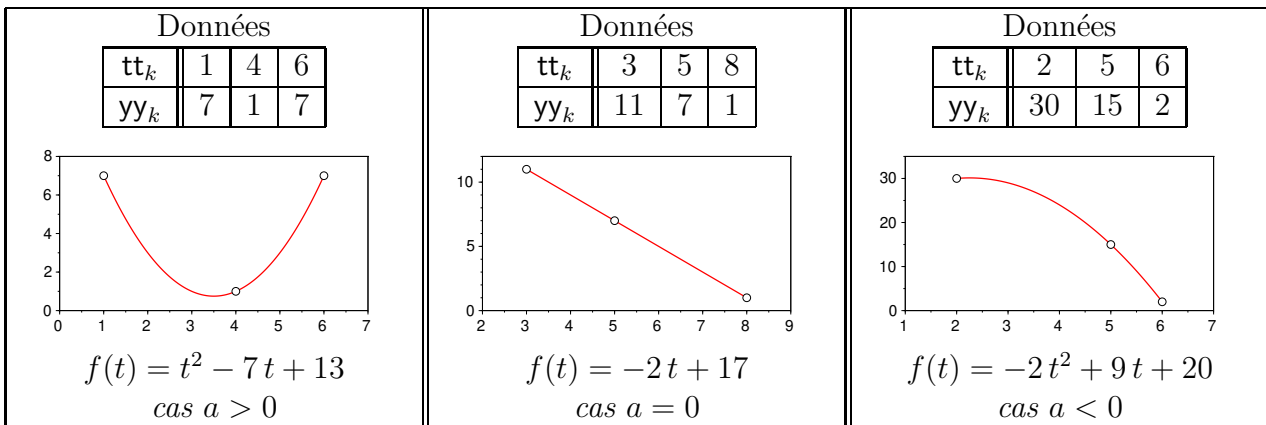
$$(7) - (6) : a(\mathbf{tt}_3 - \mathbf{tt}_1) = \frac{\mathbf{yy}_3 - \mathbf{yy}_2}{\mathbf{tt}_3 - \mathbf{tt}_2} - \frac{\mathbf{yy}_2 - \mathbf{yy}_1}{\mathbf{tt}_2 - \mathbf{tt}_1} \Longleftrightarrow a = \frac{\frac{\mathbf{yy}_3 - \mathbf{yy}_2}{\mathbf{tt}_3 - \mathbf{tt}_2} - \frac{\mathbf{yy}_2 - \mathbf{yy}_1}{\mathbf{tt}_2 - \mathbf{tt}_1}}{\mathbf{tt}_3 - \mathbf{tt}_1}$$

En notant  $d_1 = \frac{\mathbf{yy}_2 - \mathbf{yy}_1}{\mathbf{tt}_2 - \mathbf{tt}_1}$  et  $d_2 = \frac{\mathbf{yy}_3 - \mathbf{yy}_2}{\mathbf{tt}_3 - \mathbf{tt}_2}$ , on a

$$a = \frac{d_2 - d_1}{\mathbf{tt}_3 - \mathbf{tt}_1} \xRightarrow{(6)} b = d_1 - a(\mathbf{tt}_1 + \mathbf{tt}_2) \xRightarrow{(1)} c = \mathbf{yy}_1 - a \mathbf{tt}_1^2 - b \mathbf{tt}_1$$

On obtient ainsi une fonction dont le graphe est (en général) un arc de parabole sur l'intervalle  $[\mathbf{tt}_1, \mathbf{tt}_3]$ .

Les trois figures ci-dessous montrent trois exemples d'interpolation parabolique.



### Exercice 3 :

Déterminer l'expression de la fonction  $f(t)$  interpolant les données

$\mathbf{tt}_k$	0	2	3
$\mathbf{yy}_k$	3	3	6

### 2.3 - Interpolation cubique par intervalle (cubique par morceaux)

La méthode d'interpolation *linéaire par intervalle* vue précédemment permet d'obtenir une fonction continue sur  $I = [\mathbf{tt}_1, \mathbf{tt}_N]$  mais en général non dérivable sur  $I$ , la fonction pouvant présenter des points *anguleux* pour certaines valeurs  $\mathbf{tt}_k$ , ce qui est préjudiciable dans certains cas.

La méthode présentée ci-après permet d'obtenir une fonction dérivable sur l'intervalle  $I$  et définit comme un polynôme cubique (de degré 3) sur chaque intervalle  $[\mathbf{tt}_k, \mathbf{tt}_{k+1}]$ . Deux étapes sont nécessaires pour cette méthode :

**Etape (1)** On applique la méthode d'interpolation *parabolique* en considérant les données par série de trois consécutives :

- avec les trois données  $(\mathbf{tt}_1, \mathbf{yy}_1)$ ,  $(\mathbf{tt}_2, \mathbf{yy}_2)$  et  $(\mathbf{tt}_3, \mathbf{yy}_3)$ , on calcule la fonction interpolante  $f_2(t)$  qui est un polynôme de degré 2,
- avec les trois données  $(\mathbf{tt}_2, \mathbf{yy}_2)$ ,  $(\mathbf{tt}_3, \mathbf{yy}_3)$  et  $(\mathbf{tt}_4, \mathbf{yy}_4)$ , on calcule la fonction interpolante  $f_3(t)$  qui est un polynôme de degré 2,
- ...
- avec les trois données  $(\mathbf{tt}_{N-2}, \mathbf{yy}_{N-2})$ ,  $(\mathbf{tt}_{N-1}, \mathbf{yy}_{N-1})$  et  $(\mathbf{tt}_N, \mathbf{yy}_N)$ , on calcule la fonction interpolante  $f_{N-1}(t)$  qui est un polynôme de degré 2,

La fonction  $f_k(t) = a_k t^2 + b_k t + c_k$  qui interpole les trois données  $(\mathbf{tt}_{k-1}, \mathbf{yy}_{k-1})$ ,  $(\mathbf{tt}_k, \mathbf{yy}_k)$  et  $(\mathbf{tt}_{k+1}, \mathbf{yy}_{k+1})$ , est définie par ces trois coefficients  $a_k$ ,  $b_k$  et  $c_k$  :

$$\begin{aligned} \text{on calcule en premier } a_k &= \frac{\frac{\mathbf{yy}_{k+1} - \mathbf{yy}_k}{\mathbf{tt}_{k+1} - \mathbf{tt}_k} - \frac{\mathbf{yy}_k - \mathbf{yy}_{k-1}}{\mathbf{tt}_k - \mathbf{tt}_{k-1}}}{\mathbf{tt}_{k+1} - \mathbf{tt}_{k-1}} \\ \text{puis } b_k &= \frac{\mathbf{yy}_k - \mathbf{yy}_{k-1}}{\mathbf{tt}_k - \mathbf{tt}_{k-1}} - a_k (\mathbf{tt}_{k-1} + \mathbf{tt}_k) \\ \text{et enfin } c_k &= \mathbf{yy}_{k-1} - a_k \mathbf{tt}_{k-1}^2 - b_k \mathbf{tt}_{k-1} \end{aligned}$$

A ce niveau, on a déterminé  $N - 2$  fonctions  $f_k$  avec  $2 \leq k \leq N - 1$ .

On ajoute les deux fonctions supplémentaires  $f_1 = f_2$  et  $f_N = f_{N-1}$  afin d'avoir  $N$  fonctions  $f_k$  avec  $1 \leq k \leq N$  :  $\underbrace{a_1 = a_2, b_1 = b_2, c_1 = c_2}_{f_1=f_2}$  et  $\underbrace{a_N = a_{N-1}, b_N = b_{N-1}, c_N = c_{N-1}}_{f_N=f_{N-1}}$

On a alors  $\forall k, 1 \leq k \leq n, f_k(\mathbf{tt}_k) = \mathbf{yy}_k$

**Etape (2)** Une fois déterminées les  $N$  fonctions  $f_1, f_2, \dots, f_N$ , la fonction finale  $f$  qui interpole les  $N$  données  $(\mathbf{tt}_k, \mathbf{yy}_k)$  est définie ainsi : sur chaque intervalle  $[\mathbf{tt}_k, \mathbf{tt}_{k+1}]$  avec  $1 \leq k \leq N - 1$  :

$$f(t) = \frac{\mathbf{tt}_{k+1} - t}{\mathbf{tt}_{k+1} - \mathbf{tt}_k} f_k(t) + \frac{t - \mathbf{tt}_k}{\mathbf{tt}_{k+1} - \mathbf{tt}_k} f_{k+1}(t) = \frac{(\mathbf{tt}_{k+1} - t) f_k(t) + (t - \mathbf{tt}_k) f_{k+1}(t)}{\mathbf{tt}_{k+1} - \mathbf{tt}_k}$$

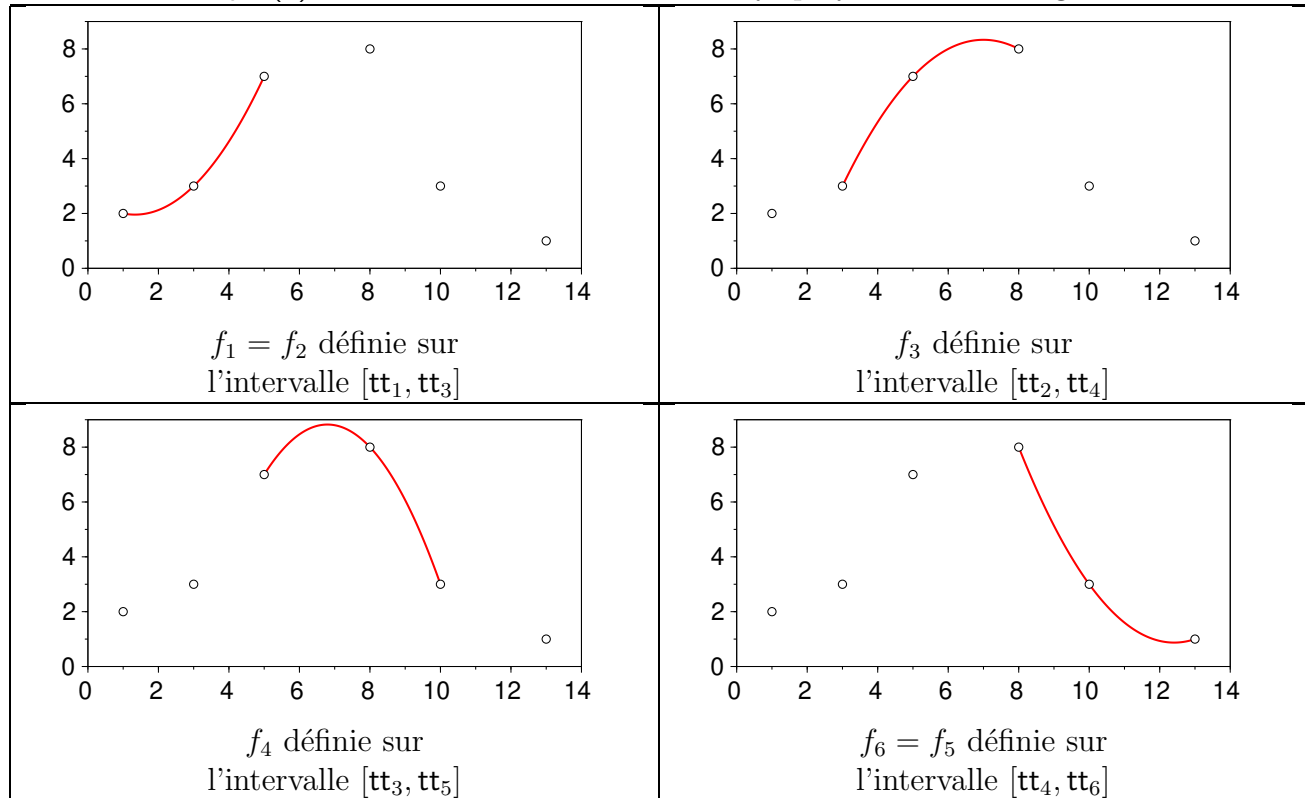
1. sur chaque intervalle  $[\mathbf{tt}_k, \mathbf{tt}_{k+1}]$ , (l'expression de) la fonction est un polynôme de degré 3 (somme de deux termes, chaque terme est le produit d'un polynôme de degré 1 par un polynôme de degré 2).
2. la fonction  $f$  est continue et dérivable sur  $I = [\mathbf{tt}_1, \mathbf{tt}_n]$ , et la fonction dérivée  $f'$  est aussi continue sur  $I = [\mathbf{tt}_1, \mathbf{tt}_n]$  (la démonstration de ce point correspond à l'exercice 11 du chapitre sur les dérivées de la partie "Analyse élémentaire").

**Exemple :** on considère les données suivantes

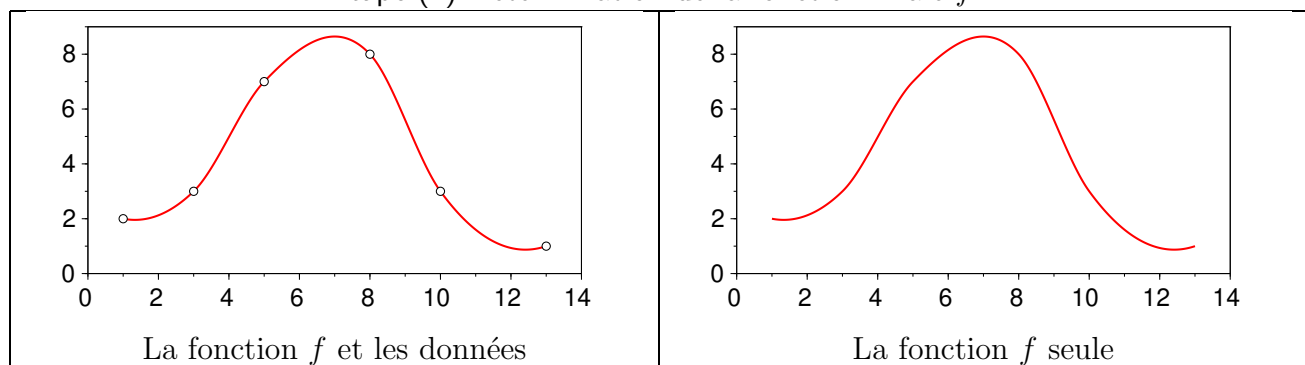
$tt_k$	1	3	5	8	10	13
$yy_k$	2	3	7	8	3	1

Les figures ci-dessous montrent les différentes fonctions  $f_k$  ( $1 \leq k \leq 6$ ) qui interpolent les données par séries de trois consécutives, et la fonction  $f$  calculée à partir des fonctions  $f_k$ .

**Etape (1) Détermination des  $N$  fonctions  $f_k$  polynomiales de degré 2**



**Etape (2) Détermination de la fonction finale  $f$**



## Algorithme de calcul et tracé de la fonction $f$

Connaissant le nombre  $N$  de données, et les valeurs  $(\mathbf{tt}_k, \mathbf{yy}_k)$  pour  $k$  de 1 à  $N$ , l'algorithme s'écrit ainsi :

```
// Etape (1) : calcul des  $N$  fonctions  $f_k$ 
// les coefficients  $a_k, b_k, c_k$  sont stockés dans trois vecteurs  $\mathbf{a}, \mathbf{b}, \mathbf{c}$  de taille  $N$ 
créer 3 vecteurs  $\mathbf{a}, \mathbf{b}, \mathbf{c}$  de taille  $N$  (avec des valeurs nulles)
// calcul des  $N - 2$  fonctions  $f_k$  pour  $k$  de 2 à  $N - 1$ 
pour  $k$  de 2 à  $N - 1$  faire
     $\mathbf{t1} \leftarrow \mathbf{tt}(k - 1)$  ,  $\mathbf{t2} \leftarrow \mathbf{tt}(k)$  ,  $\mathbf{t3} \leftarrow \mathbf{tt}(k + 1)$ 
     $\mathbf{y1} \leftarrow \mathbf{yy}(k - 1)$  ,  $\mathbf{y2} \leftarrow \mathbf{yy}(k)$  ,  $\mathbf{y3} \leftarrow \mathbf{yy}(k + 1)$ 
     $\mathbf{d1} \leftarrow (\mathbf{y2} - \mathbf{y1}) / (\mathbf{t2} - \mathbf{t1})$ 
     $\mathbf{d2} \leftarrow (\mathbf{y3} - \mathbf{y2}) / (\mathbf{t3} - \mathbf{t2})$ 
     $\mathbf{a}(k) \leftarrow (\mathbf{d2} - \mathbf{d1}) / (\mathbf{t3} - \mathbf{t1})$ 
     $\mathbf{b}(k) \leftarrow \mathbf{d1} - \mathbf{a}(k) \times (\mathbf{t1} + \mathbf{t2})$ 
     $\mathbf{c}(k) \leftarrow \mathbf{y1} - \mathbf{a}(k) \times \mathbf{t1} \times \mathbf{t1} - \mathbf{b}(k) \times \mathbf{t1}$ 
fin_pour
// compléter avec  $f_1 = f_2$  et  $f_N = f_{N-1}$ 
 $\mathbf{a}(1) \leftarrow \mathbf{a}(2)$  ,  $\mathbf{b}(1) \leftarrow \mathbf{b}(2)$  ,  $\mathbf{c}(1) \leftarrow \mathbf{c}(2)$ 
 $\mathbf{a}(N) \leftarrow \mathbf{a}(N - 1)$  ,  $\mathbf{b}(N) \leftarrow \mathbf{b}(N - 1)$  ,  $\mathbf{c}(N) \leftarrow \mathbf{c}(N - 1)$ 

// Etape (2) : calcul et tracé de la fonction finale  $f$ 
// le tracé se fait pour chaque intervalle  $[\mathbf{tt}_k; \mathbf{tt}_{k+1}]$  avec  $k$  variant de 1 à  $N - 1$ 
créer une fenêtre graphique
pour  $k$  de 1 à  $N - 1$  faire
     $\mathbf{t1} \leftarrow \mathbf{tt}(k)$  ,  $\mathbf{t2} \leftarrow \mathbf{tt}(k + 1)$  //  $[\mathbf{t1}; \mathbf{t2}] = [\mathbf{tt}(k); \mathbf{tt}(k + 1)]$ 
    créer un vecteur  $\mathbf{t}$  de 1000 valeurs dans l'intervalle  $[\mathbf{t1}; \mathbf{t2}]$ 
    // calcul des valeurs  $f_k(\mathbf{t}) \rightarrow$  vecteur  $\mathbf{f1}$ 
     $\mathbf{f1} \leftarrow \mathbf{a}(k) \times \mathbf{t} \times \mathbf{t} + \mathbf{b}(k) \times \mathbf{t} + \mathbf{c}(k)$ 
    // calcul des valeurs  $f_{k+1}(\mathbf{t}) \rightarrow$  vecteur  $\mathbf{f2}$ 
     $\mathbf{f2} \leftarrow \mathbf{a}(k + 1) \times \mathbf{t} \times \mathbf{t} + \mathbf{b}(k + 1) \times \mathbf{t} + \mathbf{c}(k + 1)$ 
    // calcul des valeurs  $f(\mathbf{t}) \rightarrow$  vecteur  $\mathbf{y}$ 
     $\mathbf{y} \leftarrow ((\mathbf{t2} - \mathbf{t}) \times \mathbf{f1} + (\mathbf{t} - \mathbf{t1}) \times \mathbf{f2}) / (\mathbf{t2} - \mathbf{t1})$ 
    tracer les points  $(\mathbf{t}, \mathbf{y})$ 
fin_pour
```

# Interpolation

## séance pratique

Séance pratique (3h) à faire en binôme.

Créez un répertoire nommé TP\_INTERPOLATION, démarrez Scilab et placez-vous dans ce répertoire.

Récupérez sur le WEB les fichiers-scripts `interpolation_lineaire.sce`, `interpolation_parabolique.sce` et `interpolation_cubique.sce` :

allez à l'adresse <http://chamilo.univ-grenoble-alpes.fr/courses/GBX1MP11/>  
entrez vos login et mot de passe  
cliquez sur les liens *Documents* puis *TP* puis *INTERPOLATION*  
cliquez sur chaque fichier et enregistrez-le dans votre répertoire de travail

Pour ce thème, le compte-rendu par binôme est à faire au fur et à mesure du TP, sous forme d'un document de type *traitement de texte*.

### Exercice 1 : tracé de données et fonction interpolante

Ecrivez un script SCILAB afin de pouvoir tracer dans une même fenêtre graphique les points de données  $(tt_k, yy_k)$  ( $1 \leq k \leq 6$ ) avec

indice $k$ de la mesure	1	2	3	4	5	6
temps $tt_k$ (en minutes)	0	2	4	6	8	10
température $yy_k$ (en degrés Celsius)	84	52	36	28	24	22

et la fonction  $f(t) = 2^{6-t/2} + 20$  afin de vérifier qu'elle interpole bien les données.

## 1 - Interpolation linéaire par morceaux

### Exercice 2 : interpolation linéaire par morceaux

Ouvrez le script `interpolation_lineaire.sce` avec l'éditeur de texte *SciNotes* de Scilab.

- complétez le script (`// LES DIFFERENTES PARTIES A COMPLETER //`), et vérifiez qu'il fonctionne correctement avec les données A-1 fournies,
- utilisez le script (en modifiant la partie 1 `les données`) avec les données A-2, puis les données A-3 suivantes :

$k$	1	2	3	4	5
$tt_k$	1	4	6	7	9
$yy_k$	2	5	9	8	4

Données A-2

$$yy_k = \cos(tt_k) \quad \text{avec } tt_k = \frac{(k-1)\pi}{8} \text{ et } 1 \leq k \leq 9$$

$k$	1	2	3	...	8	9
$tt_k$	0	$\pi/8$	$2\pi/8$	...	$7\pi/8$	$\pi$
$yy_k$	1	$\cos(\pi/8)$	$\cos(2\pi/8)$	...	$\cos(7\pi/8)$	-1

Données A-3

**CR** Dans votre compte-rendu, mettez le script `interpolation_lineaire.sce` que vous avez complété ainsi que la figure obtenue pour les données A-3.

## 2 - Interpolation cubique par morceaux

### Exercice 3 : interpolation parabolique

Ouvrez le script `interpolation_parabolique.sce` avec l'éditeur de texte *SciNotes* de Sci-lab.

- complétez le script, et vérifiez qu'il fonctionne correctement avec les données B-1 fournies,
- modifiez le script pour utiliser les données B-2, puis les données B-3 suivantes :

$k$	1	2	3
$tt_k$	1	7	9
$yy_k$	20	8	4

Données B-2

$k$	1	2	3
$tt_k$	3	6	11
$yy_k$	27	48	3

Données B-3

- Pour chaque série de donnée, observez la forme de la courbe représentative de la fonction  $f$  et la valeur du coefficient  $a$  de cette fonction. Conclusion ?

**CR** Dans votre compte-rendu, mettez le script `interpolation_parabolique.sce` que vous avez complété, ainsi que les figures obtenues pour les données B-1, B-2 et B-3.

### Exercice 4 : interpolation cubique par morceaux

Ouvrez le script `interpolation_cubique.sce` avec l'éditeur de texte

- complétez le script principal, et vérifiez qu'il fonctionne correctement avec les données C-1 fournies,
- modifiez le script pour tester les données C-2 suivantes :

$k$	1	2	3	4	5	6	7
$tt_k$	1	3	4	6	8	9	10
$yy_k$	37	20	15	12	10	10	10

Données C-2

- modifiez le script pour tester les données C-3 ci-dessous, et tracer en plus dans la même figure, la fonction  $g(t) = \sin(t)$  pour  $t \in [0, \pi]$ .

$$tt_k = (k - 1)\pi/6 \quad \text{avec } 1 \leq k \leq 7 \quad \text{et} \quad yy_k = \sin(tt_k)$$

$k$	1	2	3	4	5	6	7
$tt_k$	0	$\pi/6$	$\pi/3$	$\pi/2$	$2\pi/3$	$5\pi/6$	$\pi$
$yy_k$	0	$1/2$	$\sqrt{3}/2$	1	$\sqrt{3}/2$	$1/2$	0

Données C-3

**CR** Dans votre compte-rendu, mettez le script `interpolation_cubique.sce` que vous avez complété, et correspondant à la série de données C-3, ainsi que les figures pour les données C-1, C-2 et C-3.

Exportez votre document au format PDF, et envoyez-le par e-mail à votre enseignant de TP en indiquant comme sujet du message :

[MAP101] - Compte-rendu TP interpolation - nom(s) du binôme - groupe

# Intégration numérique

## séance préparatoire

### Présentation

Dans ce thème, on va présenter différentes méthodes pour calculer l'intégrale d'une fonction  $f(t)$  définie sur un intervalle  $[a; b]$

$$A = \int_a^b f(t) dt$$

En effet, pour certaines fonctions  $f$ , il n'existe pas d'expression analytique pour leurs primitives  $F$  et donc on ne peut pas calculer  $I$  sous la forme  $A = F(b) - F(a)$ .

Les méthodes présentées calculent, non pas  $A$  de manière exacte, mais une valeur  $\bar{A}$  proche de  $A$ .

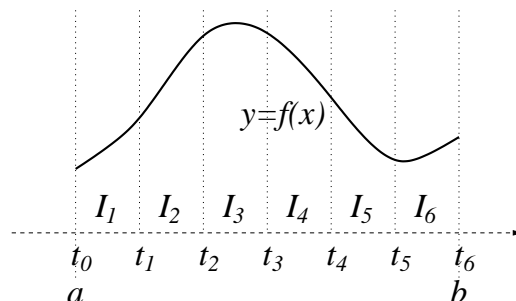
Par la suite, on se donne l'expression de la fonction  $f(t)$ , les deux réels  $a$  et  $b$  avec  $a < b$ , et on suppose que la fonction  $f(t)$  est continue sur l'intervalle  $I = [a; b]$ .

Pour les méthodes présentées, le principe général consiste à découper l'intervalle en  $n$  sous-intervalles  $I_1, I_2, \dots, I_n$  de même longueur, et sur chaque sous-intervalle  $I_k$ , on remplace  $f$  par une fonction  $g_k$ , proche de  $f$ , et pour laquelle on connaît une primitive  $G_k$ , c'est à dire on sait calculer l'intégrale (l'*aire signée* entre le graphe de la fonction  $g_k$  et l'axe horizontal).

On choisit  $n$  un entier strictement positif et pair, et on note

$$h = \frac{b-a}{n}, \quad t_k = a + k h \text{ pour } 0 \leq k \leq n, \quad I_k = [t_{k-1}, t_k] \text{ pour } 1 \leq k \leq n$$

l'intervalle  $I = [a; b]$  est découpé en  $n$  intervalles  $I_k = [t_{k-1}, t_k]$  de longueur  $h$ , avec  $a = t_0 < t_1 < t_2 < \dots < t_{n-1} < t_n = b$ .



Exemple de découpage de  $I = [a; b]$  en  $n = 6$  sous-intervalles

$$A = \sum_{k=1}^n A_k \simeq \bar{A} = \sum_{k=1}^n \bar{A}_k \quad \text{avec} \quad A_k = \int_{t_{k-1}}^{t_k} f(t) dt \quad \text{et} \quad \bar{A}_k = \int_{t_{k-1}}^{t_k} g_k(t) dt = G_k(t_k) - G_k(t_{k-1})$$

## 1 - Méthode des rectangles

Pour cette méthode, sur chaque sous-intervalle  $I_k = [t_{k-1}, t_k]$ , on remplace  $y = f(t)$  par  $y = g_k(t) = C_k = f(s_k)$  avec  $s_k \in I_k$ .

L'intégrale de  $f$  sur  $I_k$  est alors approchée par l'aire signée du rectangle de largeur  $h$  et hauteur  $C_k$  :

$$\begin{aligned} A_k &= \int_{t_{k-1}}^{t_k} f(t) dt \simeq \int_{t_{k-1}}^{t_k} C_k dt = [C_k t]_{t_{k-1}}^{t_k} = C_k t_k - C_k t_{k-1} = C_k h = \bar{A}_k \\ \Rightarrow A &= \int_a^b f(t) dt = A_1 + A_2 + \dots + A_n = \sum_{k=1}^n A_k \\ &\simeq \bar{A} = \sum_{k=1}^n \bar{A}_k = \sum_{k=1}^n C_k h = h \sum_{k=1}^n C_k = h (C_1 + C_2 + \dots + C_n) \end{aligned}$$

On peut choisir :

1.  $C_k$  valeur de la fonction à gauche de l'intervalle  $I_k$  :  $s_k = t_{k-1}$

$$C_k = f(t_{k-1}) = f(a + (k-1)h), \quad 1 \leq k \leq n$$

$$\begin{aligned} A \simeq \bar{A} &= h \left( f(a) + f(a+h) + f(a+2h) + \dots + f(a+(n-2)h) + f(a+(n-1)h) \right) \\ &= h \sum_{k=1}^n f(a + (k-1)h) \end{aligned}$$

2.  $C_k$  valeur de la fonction à droite de l'intervalle  $I_k$  :  $s_k = t_k$

$$C_k = f(t_k) = f(a + kh), \quad 1 \leq k \leq n$$

$$\begin{aligned} A \simeq \bar{A} &= h \left( f(a+h) + f(a+2h) + f(a+3h) + \dots + f(a+(n-1)h) + f(a+nh) \right) \\ &= h \sum_{k=1}^n f(a + kh) \end{aligned}$$

3.  $C_k$  valeur de la fonction au milieu de l'intervalle  $I_k$  :  $s_k = (t_{k-1} + t_k)/2$

$$C_k = f\left(\frac{t_{k-1} + t_k}{2}\right) = f\left(a + \left(k - \frac{1}{2}\right)h\right), \quad 1 \leq k \leq n$$

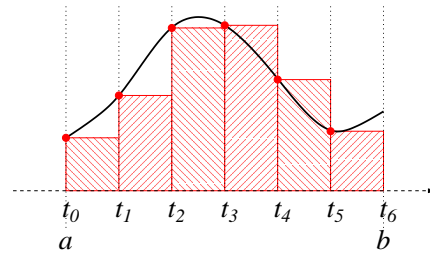
$$\begin{aligned} A \simeq \bar{A} &= h \left[ f\left(a + \frac{1}{2}h\right) + f\left(a + \frac{3}{2}h\right) + f\left(a + \frac{5}{2}h\right) + \dots \right. \\ &\quad \left. \dots + f\left(a + \frac{(2n-3)}{2}h\right) + f\left(a + \frac{(2n-1)}{2}h\right) \right] \\ &= h \sum_{k=1}^n f\left(a + \frac{2k-1}{2}h\right) \end{aligned}$$



On obtient trois variantes pour la méthode des rectangles :

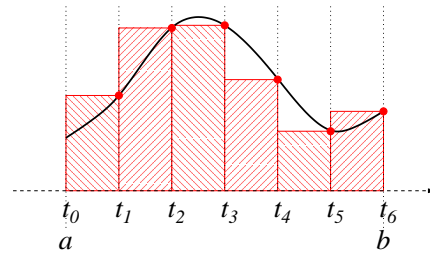
1. méthode des rectangles *valeur à gauche* :

$$\bar{A} = \bar{A}_{\text{RG}} = h \sum_{k=1}^n f(a + (k-1)h)$$



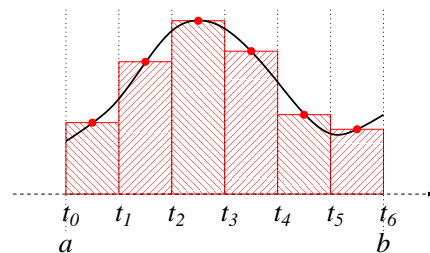
2. méthode des rectangles *valeur à droite* :

$$\bar{A} = \bar{A}_{\text{RD}} = h \sum_{k=1}^n f(a + kh)$$



3. méthode des rectangles *valeur au milieu* :

$$\bar{A} = \bar{A}_{\text{RM}} = h \sum_{k=1}^n f\left(a + \frac{2k-1}{2}h\right)$$



### Exercice 1

Calculer  $A$  pour  $f(t) = t^2$  et  $I = [a; b] = [-4; 8]$ , puis pour les trois variantes de la méthode des rectangles, calculer  $\bar{A}_{\text{RG}}$ ,  $\bar{A}_{\text{RD}}$  et  $\bar{A}_{\text{RM}}$  avec  $n = 6$ .

## 2 - Méthode des trapèzes

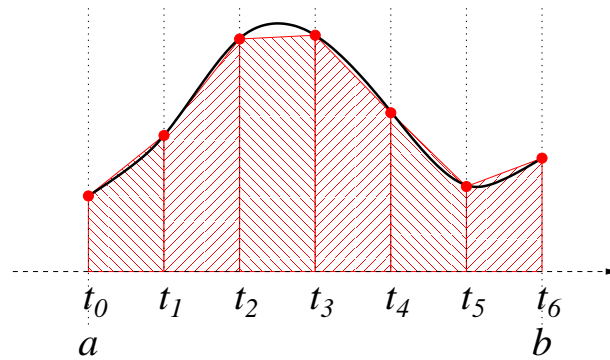
Pour cette méthode, sur chaque sous-intervalle  $I_k = [t_{k-1}, t_k]$ , on remplace  $y = f(t)$  par la fonction affine  $g_k(t) = \alpha_k t + \beta_k$  tel que

$$\begin{cases} g_k(t_k) &= f(t_k) \\ g_k(t_{k+1}) &= f(t_{k+1}) \end{cases}$$

Sur chaque intervalle  $I_k = [t_{k-1}, t_k]$ , l'aire (signée)  $\bar{A}_k$  correspondant à la fonction  $g_k$  est l'aire du trapèze dont la base est de largeur  $t_k - t_{k-1} = h$ , et les deux hauteurs sont  $f(t_{k-1})$  et  $f(t_k)$  :

$$\bar{A}_k = \frac{h}{2} (f(t_{k-1}) + f(t_k))$$

$$\Rightarrow \bar{A} = \sum_{k=1}^n \bar{A}_k = \sum_{k=1}^n \frac{h}{2} (f(t_{k-1}) + f(t_k)) = \boxed{\bar{A}_T = \frac{h}{2} \sum_{k=1}^n (f(a + (k-1)h) + f(a + kh))}$$



Exemple avec  $n = 6$  sous-intervalles

### Exercice 2 :

Calculer  $\bar{A}_T$  pour  $f(t) = t^2$ ,  $I = [a; b] = [-4; 8]$  et  $n = 6$ .

## 3 - Méthode de Simpson (méthode des paraboles)

Pour cette méthode, la valeur de  $n$  doit être nécessairement paire. Par la suite  $n = 2m$ , avec  $m$  entier strictement positif.

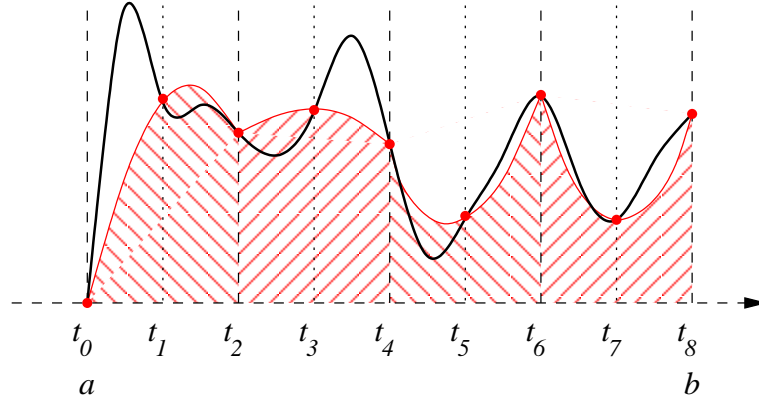
L'intervalle  $I$  est donc découpé en un nombre pair de sous-intervalles  $I_k$ , puis on regroupe les sous-intervalles par paquets de deux sous-intervalles consécutifs.

Pour deux sous-intervalles consécutifs  $I_{2k-1} = [t_{2k-2}, t_{2k-1}]$  et  $I_{2k} = [t_{2k-1}, t_{2k}]$ , on remplace  $y = f(t)$  par une même fonction  $g_k(t) = a_k t^2 + b_k t + c_k$ , fonction polynomiale de degré 2 telle que :

$$\begin{cases} g_k(t_{2k-2}) &= f(t_{2k-2}) \\ g_k(t_{2k-1}) &= f(t_{2k-1}) \\ g_k(t_{2k}) &= f(t_{2k}) \end{cases}$$

L'exemple ci-dessous correspond à  $n = 8$ , on forme donc  $m = 4$  paquets de 2 sous-intervalles :

- sur les deux intervalles  $[t_0, t_1]$  et  $[t_1, t_2]$ , on remplace  $f$  par un polynôme  $g_1$  de degré 2,
- sur les deux intervalles  $[t_2, t_3]$  et  $[t_3, t_4]$ , on remplace  $f$  par un polynôme  $g_2$  de degré 2,
- sur les deux intervalles  $[t_4, t_5]$  et  $[t_5, t_6]$ , on remplace  $f$  par un polynôme  $g_3$  de degré 2,
- sur les deux intervalles  $[t_6, t_7]$  et  $[t_7, t_8]$ , on remplace  $f$  par un polynôme  $g_4$  de degré 2.



On peut alors montrer que la fonction  $g_k(t) = a_k t^2 + b_k t + c_k$  vérifie :

$$\bar{A}_{2k-1} + \bar{A}_{2k} = \int_{t_{2k-2}}^{t_{2k}} g_k(t) dt = h \left( \frac{f(t_{2k-2}) + 4f(t_{2k-1}) + f(t_{2k})}{3} \right)$$

On en déduit une approximation de  $A$  :

$$A = \int_a^b f(t) dt = \sum_{k=1}^{n/2} (\bar{A}_{2k-1} + \bar{A}_{2k}) \simeq \bar{A} = \sum_{k=1}^{n/2} (\bar{A}_{2k-1} + \bar{A}_{2k})$$

$$\Rightarrow \bar{A} = \boxed{\bar{A}_S = \frac{h}{3} \sum_{k=1}^{n/2} \left( f(a + (2k-2)h) + 4f(a + (2k-1)h) + f(a + 2kh) \right)}$$

### Exercice 3 :

Calculer  $\bar{A}_S$  pour  $f(t) = t^2$ ,  $I = [a; b] = [-4; 8]$  et  $n = 6$ .

• • •

### Exercice 4 :

Comparer les différentes valeurs de  $\bar{A}$  obtenues dans les exercices précédents avec la valeur exacte  $A$ .

#### 4 - Preuve de la formule de la méthode de Simpson

Cette preuve n'est pas à apprendre, elle est donnée en complément comme justification de la méthode de Simpson.

$$\text{Soit } \overline{B}_k = \overline{A}_{2k-1} + \overline{A}_{2k} = \int_{t_{2k-2}}^{t_{2k}} g_k(t) dt.$$

On fait le changement de variable :  $t = \varphi(u) = h u + t_{2k-1} \iff u = \frac{t - t_{2k-1}}{h} \Rightarrow dt = h du$

$$\begin{cases} t = t_{2k-2} &= t_{2k-1} - h &\iff u = -1 \\ t = t_{2k-1} &&\iff u = 0 \\ t = t_{2k} &= t_{2k-1} + h &\iff u = 1 \end{cases}$$

$$\Rightarrow \overline{B}_k = \int_{-1}^1 g_k(h u + t_{2k-1}) h du$$

La fonction  $g : t \mapsto g_k(t)$  étant un polynôme de degré 2 en la variable  $t$ , alors la fonction  $p : u \mapsto p(u) = g_k(h u + t_{2k-1})$  est aussi un polynôme de degré 2 en la variable  $u$  :

$$\begin{aligned} p(u) &= g_k(h u + t_{2k-1}) = a_k (h u + t_{2k-1})^2 + b_k (h u + t_{2k-1}) + c_k \\ &= a_k h^2 u^2 + (2 a_k + b_k) h u + a_k t_{2k-1}^2 + b_k t_{2k-1} + c_k \end{aligned}$$

On peut donc écrire  $p(u) = \alpha u^2 + \beta u + \gamma$  :

$$\begin{aligned} \overline{B}_k &= \int_{-1}^1 g_k(h u + t_{2k-1}) h du = h \int_{-1}^1 p(u) du = h \int_{-1}^1 (\alpha u^2 + \beta u + \gamma) du \\ &= h \left[ \alpha \frac{u^3}{3} + \beta \frac{u^2}{2} + \gamma u \right]_{-1}^1 = h \left( \frac{\alpha}{3} + \frac{\beta}{2} + \gamma \right) - h \left( -\frac{\alpha}{3} + \frac{\beta}{2} - \gamma \right) = h \left( \frac{2\alpha}{3} + 2\gamma \right) \end{aligned}$$

Il reste à déterminer les valeurs de  $\alpha$  et  $\gamma$ .

Par hypothèse :

$$\begin{aligned} &\begin{cases} g(t_{2k-2}) &= f(t_{2k-2}) \\ g(t_{2k-1}) &= f(t_{2k-1}) \\ g(t_{2k}) &= f(t_{2k}) \end{cases} \\ \Rightarrow &\begin{cases} g(t_{2k-2}) &= p(-1) &= \alpha - \beta + \gamma &= f(t_{2k-2}) \\ g(t_{2k-1}) &= p(0) &= \gamma &= f(t_{2k-1}) \\ g(t_{2k}) &= p(1) &= \alpha + \beta + \gamma &= f(t_{2k}) \end{cases} \end{aligned}$$

On a donc  $\gamma = f(t_{2k-1})$ .

$$\begin{cases} \alpha - \beta + \gamma &= f(t_{2k-2}) \\ \alpha + \beta + \gamma &= f(t_{2k}) \end{cases} \Rightarrow \begin{cases} \alpha - \beta &= f(t_{2k-2}) - f(t_{2k-1}) &(1) \\ \alpha + \beta &= f(t_{2k}) - f(t_{2k-1}) &(2) \end{cases}$$

$$\Rightarrow (1) + (2) : 2\alpha = f(t_{2k-2}) - 2f(t_{2k-1}) + f(t_{2k}) \iff \alpha = \frac{f(t_{2k-2}) - 2f(t_{2k-1}) + f(t_{2k})}{2}$$

On obtient la formule finale pour  $\overline{B}_k$  :

$$\begin{aligned} \overline{B}_k &= h \left( \frac{2\alpha}{3} + 2\gamma \right) = h \left( \frac{f(t_{2k-2}) - 2f(t_{2k-1}) + f(t_{2k})}{3} + 2f(t_{2k-1}) \right) \\ &= h \frac{f(t_{2k-2}) + 4f(t_{2k-1}) + f(t_{2k})}{3} \end{aligned}$$

# Intégration numérique

## séance pratique

Séance pratique (3h) à faire en binôme.

Dans ce TP, vous allez programmer en Scilab les méthodes vues en Cours-TD, puis utilisez ces scripts pour comparer les méthodes entre elles.

Pour cela, on va choisir des fonctions  $y = f(t)$  pour lesquelles on sait calculer  $A = \int_a^b f(t) dt$ , puis pour évaluer une méthode, on calculera l'intégrale approchée  $\bar{A}$  puis l'erreur  $e = |A - \bar{A}|$ , plus l'erreur  $e$  sera proche de 0, meilleure sera la méthode.

Pour ce thème, le compte-rendu (à faire par binôme) consiste à compléter au fur et à mesure du TP le document imprimé (qui vous sera distribué), et à le rendre à la fin de la séance.

**IMPORTANT** - vous aurez à remplir différents tableaux :

- pour chaque valeur de  $A$ , écrire la valeur numérique exacte,
- pour afficher puis écrire les différentes valeurs numériques des intégrales approchées  $\bar{A}_k$ , vous utiliserez l'instruction `mprintf("%18.12f\n", A)` (12 chiffres après la virgule).
- pour afficher puis écrire la valeur d'une erreur  $e = |A - \bar{A}|$ , vous utiliserez l'instruction `mprintf("%10.3e\n", e)` (format scientifique avec 3 chiffres significatifs).

### 1 - Calcul d'une somme

Dans les différentes méthodes, le calcul de l'aire (approchée)  $\bar{A}$  se fait à l'aide d'une somme. Il faut donc dans un premier temps savoir comment calculer avec Scilab une somme du type

$$\bar{A} = \sum_{k=1}^n \bar{A}_k$$

Ecrivez un fichier-script Scilab nommé `calcul_somme.sce` qui :

- demande à l'utilisateur d'entrer un entier  $n$  entre 1 et 100,
- calcule et écrit la somme  $\sum_{k=1}^n (4k^3 - 6k^2 + 4k - 1)$ ,
- et calcule et écrit la valeur  $n^4$ .

Testez votre script avec différentes valeurs de  $n$  entre 1 et 200 pour vérifier que

$$\sum_{k=1}^n (4k^3 - 6k^2 + 4k - 1) = n^4$$

## 2 - Méthode des rectangles

1. Calculez  $A = \int_0^{\pi/2} \sin(t) dt$
2. Ecrivez un fichier-script Scilab nommé `methode_rectangles.sce` afin de :
  - définir la fonction  $f(t) = \sin(t)$ , l'intervalle  $[a, b] = [0, \pi/2]$  et la valeur  $A$ ,
  - demander à l'utilisateur d'entrer au clavier un entier  $n$  pair et strictement positif,
  - calculer  $\overline{A}_{RG}$  la valeur approchée de  $A$  par la méthode des rectangles (valeur à gauche), puis afficher à l'écran les valeurs de  $\overline{A}_{RG}$  et  $e_1 = |A - \overline{A}_{RG}|$ ,
  - calculer  $\overline{A}_{RD}$  la valeur approchée de  $A$  par la méthode des rectangles (valeur à droite), puis afficher à l'écran les valeurs de  $\overline{A}_{RD}$  et  $e_2 = |A - \overline{A}_{RD}|$ ,
  - calculer  $\overline{A}_{RM}$  la valeur approchée de  $A$  par la méthode des rectangles (valeur au milieu), puis afficher à l'écran les valeurs de  $\overline{A}_{RM}$  et  $e_3 = |A - \overline{A}_{RM}|$ .
3. Utilisez votre script afin de commencer à remplir le tableau 1 du compte-rendu.
4. Modifiez votre script afin de calculer l'intégrale de  $f(t) = e^t$  sur l'intervalle  $[\ln(1/2), \ln(2)]$  et remplissez la première partie du tableau 2.

## 3 - Méthode des trapèzes

1. En s'inspirant du fichier-script de la méthode précédente, écrivez un fichier-script Scilab nommé `methode_trapezes.sce` afin de :
  - définir la fonction  $f(t) = \sin(t)$ , l'intervalle  $[a, b] = [0, \pi/2]$  et la valeur  $A$ ,
  - demander à l'utilisateur d'entrer au clavier un entier  $n$  pair et strictement positif,
  - calculer  $\overline{A}_T$  la valeur approchée de  $A$  par la méthode des trapèzes, puis afficher à l'écran les valeurs de  $\overline{A}_T$  et  $e_4 = |A - \overline{A}_T|$ ,
2. Utilisez votre script afin de continuer à remplir le tableau 1 du compte-rendu.
3. Modifiez votre script afin de calculer l'intégrale de  $f(t) = e^t$  sur l'intervalle  $[\ln(1/2), \ln(2)]$  et remplissez la deuxième partie du tableau 2.

## 4 - Méthode de Simpson

1. En s'inspirant des fichiers-scripts des méthodes précédentes, écrivez un fichier-script Scilab nommé `methode_simpson.sce` afin de :
  - définir la fonction  $f(t) = \sin(t)$ , l'intervalle  $[a, b] = [0, \pi/2]$  et la valeur  $A$ ,
  - demander à l'utilisateur d'entrer au clavier un entier  $n$  pair et strictement positif,
  - calculer  $\overline{A}_S$  la valeur approchée de  $A$  par la méthode de Simpson, puis afficher à l'écran les valeurs de  $\overline{A}_S$  et  $e_5 = |A - \overline{A}_S|$ ,
2. Utilisez votre script afin de terminer à remplir le tableau 1 du compte-rendu.
3. Modifiez votre script afin de calculer l'intégrale de  $f(t) = e^t$  sur l'intervalle  $[\ln(1/2), \ln(2)]$  et remplissez la troisième partie du tableau 2.

• • •

**CR** Remplissez la fiche de compte-rendu distribuée lors de la séance.

Page 1 : complétez les cadres et tableaux, pour les valeurs exactes  $A$  des intégrales, donnez les valeurs numériques correspondantes.

Page 2 : remplissez les cadres (A) et (B) du compte-rendu en répondant aux questions, et rendez le compte-rendu à la fin de la séance.

# Résolution numérique d'équation différentielle

## séance préparatoire

**But :** présenter des méthodes numériques (calcul par ordinateur) pour calculer (de manière approchée) la solution d'une équation différentielle.

En séance pratique, écrire des scripts Scilab en implémentant les méthodes vues en CTD, et tester ces méthodes sur des exemples particuliers pour lesquels on sait calculer la solution exacte.

### Présentation

Dans ce thème, on va présenter des méthodes pour résoudre une équation différentielle (E.D.)

1. soit du premier ordre de la forme

$$\begin{cases} y'(t) + a(t)y(t) = f(t) & \text{avec } t \in [t_{min}, t_{max}] \\ y(t_{min}) = y_0 \end{cases}$$

L'intervalle  $I = [t_{min}, t_{max}]$ , la valeur réelle  $y_0$ , les fonctions  $a(t)$  et  $f(t)$  (supposées continues sur  $I$ ) sont connus, et on veut déterminer la fonction  $y(t)$  pour  $t \in I$ .

2. soit du deuxième ordre de la forme

$$\begin{cases} y''(t) + a y'(t) + b y(t) = f(t) & \text{avec } t \in [t_{min}, t_{max}] \\ y(t_{min}) = y_0 \\ y'(t_{min}) = z_0 \end{cases}$$

L'intervalle  $I = [t_{min}, t_{max}]$ , les valeurs réelles  $y_0$  et  $z_0$ , les constantes  $a$  et  $b$  sont connus, et on veut déterminer la fonction  $y(t)$  pour  $t \in I$ .

• • •

Parfois, il n'est pas possible de déterminer la fonction  $y(t)$  sous forme d'une expression analytique.

Le principe des méthodes présentées consiste à diviser l'intervalle  $I$  en  $n$  intervalles  $[\mathbf{tt}_k, \mathbf{tt}_{k+1}]$  de même longueur, puis rechercher une fonction  $\bar{y}$  (proche de la fonction  $y$ ) en calculant uniquement la fonction  $\bar{y}$  pour les valeurs  $\mathbf{tt}_k$ .

Une méthode sera d'autant plus efficace que les valeurs approchées  $\bar{y}(\mathbf{tt}_k)$  seront proches des valeurs exactes  $y(\mathbf{tt}_k)$ .

• • •

Par la suite, on se donne un entier strictement positif  $n$ , et on note

$$h = \frac{t_{max} - t_{min}}{n}, \quad \mathbf{tt}_k = t_{min} + (k-1)h \quad \text{pour } 1 \leq k \leq n+1$$

et on va calculer les valeurs  $\mathbf{yy}_k = \bar{y}(\mathbf{tt}_k) \simeq y(\mathbf{tt}_k)$ .

## 1 - E.D. du premier ordre

Soit l'équation différentielle :

$$\begin{cases} y'(t) + a(t)y(t) = f(t) & \text{avec } t \in [t_{\min}, t_{\max}] \\ y(t_{\min}) = y_0 \end{cases}$$

On connaît les fonctions  $a$  et  $f$ , les réels  $t_{\min}$ ,  $t_{\max}$ ,  $y_0$ , et on se donne un entier  $n$  strictement positif.

On calcule le pas  $h = \frac{t_{\max} - t_{\min}}{n}$ .

Le principe de la méthode, est de calculer les valeurs de  $yy_k$  sous forme d'une récurrence.

(1) Initialisation Connaissant la valeur  $y_0$  pour  $t = t_0$ , on a donc  $tt_1 = t_0$  et  $yy_1 = y_0$ .

(2) Récurrence Connaissant la valeur approchée  $yy_k$  de  $y(tt_k)$ , on détermine alors la valeur approchée  $yy_{k+1}$  de  $y(tt_{k+1}) = y(tt_k + h)$  de la manière suivante.

On a :

$$\begin{aligned} y'(t) + a(t)y(t) &= f(t), \forall t \in [t_{\min}, t_{\max}] \\ \Rightarrow y'(tt_k) + a(tt_k)y(tt_k) &= f(tt_k) \iff y'(tt_k) = f(tt_k) - a(tt_k)y(tt_k) \end{aligned}$$

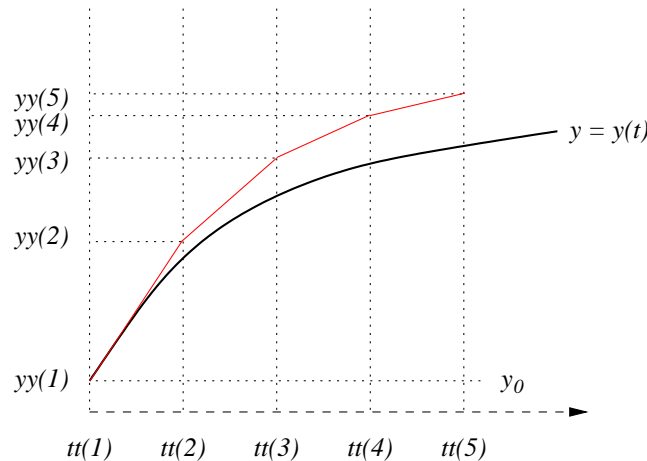
On approche alors la dérivée  $y'(tt_k)$  par le taux d'accroissement

$$\begin{aligned} \frac{y(tt_k + h) - y(tt_k)}{h} &\simeq y'(tt_k) \\ \iff \frac{y(tt_k + h) - y(tt_k)}{h} &\simeq f(tt_k) - a(tt_k)y(tt_k) \end{aligned}$$

Comme  $yy_k$  est une valeur approchée de  $y(tt_k)$ , on calcule alors  $yy_{k+1}$  valeur approchée de  $y(tt_{k+1}) = y(tt_k + h)$  de manière à ce que :

$$\frac{yy_{k+1} - yy_k}{h} = f(tt_k) - a(tt_k)yy_k \Rightarrow yy_{k+1} = yy_k + h(f(tt_k) - a(tt_k)yy_k)$$

et on a ainsi l'expression de  $yy_{k+1}$  en fonction de  $yy_k$  et des données du problème.





L'algorithme de la méthode d'Euler s'écrit ainsi :

on se donne  $t_{min}$ ,  $t_{max}$ , les expressions des fonctions  $f(t)$  et  $a(t)$ , la valeur  $y_0$  et un entier  $n$  strictement positif, et on calcule les deux suites (tableaux) **tt** et **yy**.

```

 $h \leftarrow (t_{max} - t_{min})/n$  // calcul du pas  $h$ 
// initialisation
créer deux tableaux tt et yy de  $n + 1$  valeurs (nulles)
 $tt(1) \leftarrow t_{min}$ 
 $yy(1) \leftarrow y_0$ 
pour  $k$  de 1 à  $n$  faire
     $tt(k + 1) \leftarrow tt(k) + h$ 
     $yy(k + 1) \leftarrow yy(k) + h \left( f(tt(k)) - a(tt(k)) yy(k) \right)$ 
fin_pour

```

### Exercice 1

Soit l'E.D. suivante :

$$\begin{cases} y'(t) - y(t) = t & \text{avec } t \in [t_{min}; t_{max}] = [0; 5] \\ y(t_{min}) = 0 \end{cases}$$

- Quelles sont les données du problème ?
- Pour  $n = 5$ , que devient la récurrence pour la suite  $(yy(k))$  ?  
Calculer les valeurs des suites  $(tt(k))$  et  $(yy(k))$  pour  $k$  entre 1 et  $n + 1 = 6$ .

## 2 - E.D. du second ordre

Soit l'équation différentielle :

$$\begin{cases} y''(t) + a y'(t) + b y(t) = f(t) & \text{avec } t \in [t_{min}, t_{max}] \\ y(t_{min}) = y_0 \\ y'(t_{min}) = z_0 \end{cases}$$

On connaît la fonction  $f$ , les réels  $a$ ,  $b$ ,  $t_{min}$ ,  $t_{max}$ ,  $y_0$ ,  $z_0$ , et on se donne un entier  $n$  strictement positif.

Pour résoudre l'E.D. du second ordre précédente, on se ramène à deux E.D. du premier ordre en posant la fonction  $z(t) = y'(t) \Rightarrow z'(t) = y''(t)$ , et l'E.D. s'écrit alors

$$\left\{ \begin{array}{l} y'(t) \\ z'(t) + a z(t) + b y(t) \end{array} = \begin{array}{l} z(t) \\ f(t) \end{array} \right\} \text{ avec } t \in [t_{min}, t_{max}] \text{ et } \left\{ \begin{array}{l} y(t_{min}) = y_0 \\ z(t_{min}) = z_0 \end{array} \right.$$

En approchant  $y'(t)$  par  $\frac{y(t+h) - y(t)}{h}$  et  $z'(t)$  par  $\frac{z(t+h) - z(t)}{h}$ , on obtient

$$\left\{ \begin{array}{l} y'(t) \\ z'(t) + a z(t) + b y(t) \end{array} = \begin{array}{l} z(t) \\ f(t) \end{array} \right\} \Rightarrow \left\{ \begin{array}{l} \frac{y(t+h) - y(t)}{h} \simeq z(t) \\ \frac{z(t+h) - z(t)}{h} \simeq f(t) - a z(t) - b y(t) \end{array} \right.$$

$$\Leftrightarrow \left\{ \begin{array}{l} y(t+h) \simeq y(t) + h z(t) \\ z(t+h) \simeq z(t) + h \left( f(t) - a z(t) - b y(t) \right) \end{array} \right.$$

La méthode va consister à calculer deux suites

$$yy(k) \simeq y(t_{min} + (k-1)h) \quad \text{et} \quad zz(k) \simeq z(t_{min} + (k-1)h) \quad \text{avec} \quad 1 \leq k \leq n+1$$

ainsi :

$$\begin{cases} yy(1) &= y(t_{min}) &= y_0 \\ zz(1) &= z(t_{min}) &= z_0 \end{cases}$$

et pour  $k$  variant de 1 à  $n$  :

$$\begin{cases} yy(k+1) &= yy(k) + h \, zz(k) \\ zz(k+1) &= zz(k) + h \left( f(tt(k)) - a \, zz(k) - b \, yy(k) \right) \end{cases}$$

L'algorithme de la méthode d'Euler s'écrit ainsi :

on se donne  $t_{min}$ ,  $t_{max}$ , l'expression de la fonction  $f(t)$ , les valeurs de  $a$ ,  $b$ ,  $y_0$ ,  $z_0$  et un entier  $n$  strictement positif, et on calcule les trois suites (tableaux) **tt**, **yy** et **zz**.

```

h ← (tmax - tmin)/n  // calcul du pas h
// initialisation
créer trois tableaux tt, yy et zz de n + 1 valeurs (nulles)
tt(1) ← tmin
yy(1) ← y0
zz(1) ← z0
pour k de 1 à n faire
    tt(k+1) ← tt(k) + h
    yy(k+1) ← yy(k) + h zz(k)
    zz(k+1) ← zz(k) + h ( f(tt(k)) - a zz(k) - b yy(k) )
fin_pour

```

## Exercice 2

(à préparer pour le TP)

Pour chacune des équations différentielles suivantes, déterminer la (fonction) solution  $y$  :

1.  $\begin{cases} y'(t) = \cos(t) & \text{avec } t \in [0; 10\pi] \\ y(0) = 0 \end{cases}$
2.  $\begin{cases} y'(t) + y(t) = 3e^{-t} & \text{avec } t \in [0; 5] \\ y(0) = 1 \end{cases}$   
( chercher une solution particulière de l'équa. diff. sous la forme  $y_0(t) = \alpha t e^{-t}$  )
3.  $\begin{cases} y''(t) + 2y'(t) + 26y(t) = 0 & \text{avec } t \in [0; 2\pi] \\ y(0) = 5 \\ y'(0) = 0 \end{cases}$
4.  $\begin{cases} y''(t) + y(t) = t/2 & \text{avec } t \in [0; 10\pi] \\ y(0) = 1 \\ y'(0) = -1/2 \end{cases}$   
( montrer que  $y_0(t) = t/2$  est une solution particulière de l'équa. diff. )
5.  $\begin{cases} y''(t) - y(t) = 0 & \text{avec } t \in [-1; 1] \\ y(-1) = e + 1/e = 2 \cosh(-1) \\ y'(-1) = -e + 1/e = 2 \sinh(-1) \end{cases}$

# Résolution numérique d'équation différentielle

## séance pratique

Séance pratique (3h) à faire en binôme.

Dans ce TP, vous allez programmer en Scilab les méthodes numériques (algorithmes) vues en Cours-TD, puis les utiliser avec des équations différentielles.

Pour cela, on va choisir des équations différentielles pour lesquelles on sait calculer l'expression analytique de la solution  $y(t)$  puis on va comparer graphiquement la solution exacte  $y(t)$  avec la solution approchée  $\bar{y}(t)$  obtenue par la méthode numérique.

Pour ce thème, le compte-rendu par binôme est à faire au fur et à mesure du TP, sous forme d'un document de type *traitement de texte*.

### 1 - E.D. du premier ordre

1) Ecrivez un script Scilab effectuant les opérations suivantes :

- définir les bornes  $t_{min}$  et  $t_{max}$ , les fonctions  $a(t)$  et  $f(t)$ , la valeur  $y_0$ , l'entier strictement positif  $n$ ,
- implémenter l'algorithme pour calculer les tableaux **tt** et **yy**
- définir la solution exacte  $y(t)$ ,
- tracer dans une même fenêtre la solution approchée  $\bar{y}$  (points (**tt**, **yy**)) (en rouge), et la courbe de la solution exacte  $y(t)$  pour  $t \in [t_{min}; t_{max}]$  (en noir).

2) Utilisez votre script avec l'équation différentielle :

$$\begin{cases} y'(t) = \cos(t) & \text{avec } t \in [0; 10\pi] \\ y(0) = 0 \end{cases} \quad (E_1)$$

et tracer les graphiques correspondants à  $n = 5$ ,  $n = 10$ ,  $n = 100$  et  $n = 1000$ .

**CR** Dans votre compte-rendu :

- mettez le script *SCILAB* que vous avez écrit,
- mettez les graphiques obtenus pour les différentes valeurs de  $n$ ,
- pour le cas  $n = 5$ , refaites le même exercice que l'exercice 1 de la séance préparatoire, et montrer que l'allure du graphe de la solution approchée est bien une droite.

3) Utilisez votre script avec l'équation différentielle :

$$\begin{cases} y'(t) + y(t) = 3e^{-t} & \text{avec } t \in [0; 5] \\ y(0) = 1 \end{cases} \quad (E_2)$$

et tracer les graphiques correspondants à  $n = 10$ ,  $n = 100$  et  $n = 1000$ .

**CR** Dans votre compte-rendu :

- mettez les graphiques obtenus pour les différentes valeurs de  $n$ ,

## 2 - E.D. du deuxième ordre

1) Ecrivez un script Scilab effectuant les opérations suivantes :

- définir les bornes  $t_{min}$  et  $t_{max}$ , les constantes  $a$  et  $b$ , la fonction  $f(t)$ , les valeurs  $y_0$  et  $z_0$ , l'entier strictement positif  $n$ ,
- implémenter l'algorithme pour calculer les tableaux **tt** et **yy**
- définir la solution exacte  $y(t)$ ,
- tracer dans un même graphique la solution approchée  $\overline{y}$  (points (**tt**, **yy**)) et la courbe de la solution exacte  $y(t)$  pour  $t \in [t_{min}; t_{max}]$ .

2) Utilisez votre script avec l'équation différentielle :

$$\begin{cases} y''(t) + 2 y'(t) + 26 y(t) = 0 & \text{avec } t \in [0; 2\pi] \\ y(0) = 5 \\ y'(0) = 0 \end{cases} \quad (E_3)$$

et tracer les graphiques correspondants à  $n = 50$ ,  $n = 100$ ,  $n = 1000$  et  $n = 10000$ .

**CR** Dans votre compte-rendu :

- mettez le script SCILAB que vous avez écrit,
- mettez les graphiques obtenus pour les différentes valeurs de  $n$ .

3) Utilisez votre script avec l'équation différentielle :

$$\begin{cases} y''(t) + y(t) = t/2 & \text{avec } t \in [0; 10\pi] \\ y(0) = 1 \\ y'(0) = -1/2 \end{cases} \quad (E_4)$$

et tracer les graphiques correspondants à  $n = 100$ ,  $n = 1000$  et  $n = 10000$ .

**CR** Dans votre compte-rendu :

- mettez les graphiques obtenus pour les différentes valeurs de  $n$ .

4) Utilisez votre script avec l'équation différentielle :

$$\begin{cases} y''(t) - y(t) = 0 & \text{avec } t \in [-1; 1] \\ y(-1) = e + 1/e \\ y'(-1) = -e + 1/e \end{cases} \quad (E_5)$$

et tracer les graphiques correspondants à  $n = 10$ ,  $n = 100$  et  $n = 1000$ .

**CR** Dans votre compte-rendu :

- mettez les graphiques obtenus pour les différentes valeurs de  $n$ .

• • •

**CR** Terminez votre compte-rendu avec un commentaire sur les résultats obtenus, notamment pour une même équation différentielle, qu'observe-t-on lorsque la valeur de  $n$  augmente ? Exportez votre compte-rendu au format PDF, puis envoyez-le par e-mail à votre enseignant en indiquant comme objet/sujet du message

[MAP101] - Compte-rendu TP Resol ED - noms du binôme - groupe



# Mémento Scilab

(numéro à droite entre crochets : page d'un exemple dans le thème *Intro. à Scilab*)

## Constantes prédéfinies

%e (exp(1))	%pi ( $\pi$ )	%T (VRAI)	%F (FAUX)
-------------	---------------	-----------	-----------

## Fonctions usuelles prédéfinies

abs (valeur absolue)	sqrt (racine carrée)	exp	log (logarithme népérien ln)
cos	sin	tan	cosh sinh tanh

## Scalaire / vecteur

$r = \text{expression}$	(création du scalaire $r$ à partir d'une expression)	[4]
$v = [v_1 \ v_2 \ \dots v_n]$	(création du vecteur $v$ à partir de valeurs $v_k$ )	[5]
$v(k)$	(accès à l'élément d'indice $k$ du vecteur $v$ )	[5]
zeros(1, $n$ )	(vecteur de $n$ valeurs nulles)	[6]
ones(1, $n$ )	(vecteur de $n$ valeurs égales à 1)	[6]
linspace( $a, b, n$ )	(vecteur de $n$ valeurs entre $a$ et $b$ )	[6]
$n_1:n_2$	(vecteur des entiers de $n_1$ à $n_2$ )	[6]

## Opérateurs arithmétiques

+	(addition)	-	(soustraction)	*	(multiplication)	/	(division)	**	^	(puissance)
.*	./	. **	. ^	(multiplication/division/puissance avec vecteur)					[7]	

## Opérateurs booléens

&	(ET)	&&	(ET)		(OU)		(OU)	~	(NON)	[4]
---	------	----	------	--	------	--	------	---	-------	-----

## Opérateurs de comparaison

< ( <i>inférieur strict</i> )	<= ( <i>inférieur ou égal</i> )	== ( <i>égal</i> )	
> ( <i>supérieur strict</i> )	>= ( <i>supérieur ou égal</i> )	~= ( <i>différent</i> )	[4]

## Lecture/écriture

input (lecture)	disp (écriture)	mprintf (écriture formatée)	[8][9]
-----------------	-----------------	-----------------------------	--------

## Test conditionnel

if expression_booléenne then instructions end	[10]
if ... then ... else ... end	
if ... then ... elseif ... then ... else ... end	[10]

## Boucle conditionnelle

while expression_booléenne instructions end	[11]
---	------

## Boucle inconditionnelle

for compteur = vecteur instructions end	[12]
---	------

## Définition de fonction

deff("y=nom_fonction(x)" , "y= expression")	[13]
function y=nom_fonction(x) instructions endfunction	[15]

## Représentation graphique

scf()	(création d'un fenêtre graphique)	[18]
plot( $x, y, options$ )	(tracé de points)	[19]
replot([ $xmin, ymin, xmax, ymax$ ])	(bornes du repère)	[20]

## Divers

// ...	(commentaire)	
exec("nom_fichier", -1)	(chargement d'un fichier)	[14]