METHODES NUMERIQUES APPLIQUEES

cours, exercices corrigés et mise en œuvre en JAVA

M.A. Aziz Alaoui et C. Bertelle
Faculté des Sciences et Techniques
25 rue Philippe Lebon - BP 540
76058 Le Havre Cedex - France

13 septembre 2002

Résumé

On présente brièvement quelques méthodes numériques usuelles et élémentaires à l'usage des étudiants en fin de premiers cycles scientifiques universitaires ou de ceux débutant un second cycle.

..... A REVOIR .....

Les mathématiciens et autres scientifiques se sont toujours intéressés à la résolution numérique des problèmes qu'ils rencontrent, l'analyse mathématique classique ne pouvant résoudre tous les problèmes qui se posent (par exemple en intégration, équations différentielles, interpolation, résolution d'équations non linéaires ...). L'intérêt des scientifiques pour les méthodes et l'analyse numérique en général, n'a donc cessé d'augmenter. Deux notions s'avèrent alors importantes :

-l'erreur numérique

-la notion de stabilité.

Abordées de manière rapide dans cet ouvrage, nous renvoyons le lecteur à des ouvrages spécialisés [**?**, **?**] pour approfondir ces notions utiles pour la validation des méthodes numériques.

Les théorèmes et propositions énoncés ne seront pas démontrés, la plupart sont, ou peuvent être facilement, traités en exercices, n'exigeant pas d'arguments mathématiques profonds.

Nous complétons un certain nombre de chapitres par des petits programmes d'applications en Java où l'on a privilégié une lecture facile pour des débutants dans ce langage et la programmation objet plutôt que des développements exhaustifs et génériques nécessitant souvent des constructions élaborées qui dépassent le cadre des objectifs définis pour ce manuel. Le choix de Java pour implémenter des méthodes numériques est discutable en raison des faibles performances de calcul des plate-formes de développement actuelles en Java mais qui s'améliorent régulièrement de manière importante. Notre souci premier est avant tout pédagogique et Java est un langage orienté objet propre et qui se veut simple. Sa portabilité intégrant des possibilités graphiques intéressantes répond à nos préoccupations pédagogiques et de rentabilisation de l'investissement dans un nouveau langage dans une discipline informatique en évolution permanente.

Table des matières

Chapitre 1

Introduction à Java

L'objectif de ce chapitre n'est pas de faire une présentation exhaustive de Java, mais d'en donner les notions essentielles au lecteur qui n'y est pas coutumier, pour qu'il en saisisse l'esprit et puisse comprendre les développements d'applications de méthodes numériques qui seront faites dans les chapitres suivants. Le lecteur désirant des renseignements plus complets sur ce langage pourra consulter, par exemple [**?**, **?**].

Nous supposons, par ailleurs, que le lecteur a une connaissance préalable d'un langage de programmation évolué - C par exemple - mais pas nécessairement en programmation objet. On peut considérer que c'est le cas de la majorité des étudiants en premier cycle scientifique.

1.1 Présentation de Java

1.1.1 Objectifs

Le langage Java a été développé afin de pouvoir générer des applications qui soient indépendantes des machines et de leur système d'exploitation. Une des autres caractéristiques du langage est de pouvoir écrire des applications structurellement distribuées sur des réseaux. Il appartient, par ailleurs, à la famille des langages objets purs où rien ne peut exister en dehors des classes.

1.1.2 Historique

Au début des années 1990, une équipe de développeurs de la société SUN Microsystems travaille sur l'implémentation du langage OAK pour l'intégrer en domotique, notamment pour le développement de la télévision interactive. Il fallait que les codes des applications dans ce langage soit peu volumineux, efficaces

et indépendants de l'architecture. A cette époque, la télévision interactive n'a pas connue l'essor escompté alors que le développement d'Internet et du Web faisait apparaître des besoins urgents de même nature. En 1995, OAK devient Java et se trouve popularisé rapidement pour ses possibilités de développement liés au Web. Les années qui suivent font connaître à Java une popularité en tant que langage généraliste qui dépasse les prévisions de SUN. Les grands industriels du développement l'adopte rapidement en l'espace de quelques années. Pour les années à venir, les industriels projettent la création de puces électroniques dédiées à Java ... un juste retour vers les préoccupations initiales des concepteurs originaux de OAK/Java.

Certes Java est un langage propriétaire, mais de licence ouverte, ce qui lui a permis de se faire adopter par tous les professionnels du développement avec un engouement sans précédent par rapport aux autres langages plus anciens.

1.1.3 Une machine virtuelle

Le langage Java est un langage compilé et interprété. Cette définition contradictoire s'explique par le fait que le code source est transformé dans un byte-code universel exécutable par une machine virtuelle. Cette machine virtuelle peut être installée à partir d'une distribution du langage comme le Java Development Kit (JDK) de SUN. Elle peut être également intégrée sous une forme compactée dans un navigateur afin qu'il puisse exécuter des applets Java.

Ces caractéristiques confèrent au langage des propriétés de portabilité sans précédents, mais ont aussi un coût correspondant au fonctionnement de la machine virtuelle qui réduit les performances d'exécution du programme. Afin de résoudre ce problème, un effort important est consenti pour développer des compilateurs à la volée en code machine qui fonctionnent lors de l'exécution du programme (compilateurs identifiés sous la dénomination de JIT - Just In Time).

1.1.4 Caractéristiques

Le langage Java possède une syntaxe inspirée du C++. Il se veut plus propre en terme de développement objet, ne permettant pas de construction en dehors des classes. Il se veut aussi plus simple en affranchissant le programmeur de toute la gestion dynamique des objets construits, grâce au fonctionnement d'un ramasse-miettes (Garbage Collector) dont la fonction est d'identifier et d'éliminer tous les objets qui ne sont plus référencés.

Java propose des exécutions parallèles de programmes grâce à une utilisation qui se veut plus facile des *threads* (ou processus légers) par rapport au langage C. Il possède également des aspects liés à la distribution grâce à ses possibilités d'intégration dans des documents Web distribués par les applets, mais également

avec la bibliothèque RMI (Remote Methods Invocation) qui propose de la programmation d'objets répartis. L'évolution de cette bibliothèque la fait converger progressivement vers CORBA, le standard dans le domaine des objets répartis. Les bibliothèques CORBA sont aujourd'hui intégrées dans les distributions du JDK 1.2 ou supérieures de SUN.

Dans cet ouvrage où Java sert de langage élémentaire d'implémentation de méthodes numériques, les *threads* et la programmation en objets distribués ne seront pas utilisés. Le parallélisme et la programmation distribuée sont indéniablement des apports récents et majeurs en calcul scientifique et feront certainement l'objet d'ouvrages spécifiques complémentaires à celui que nous présentons ici.

1.2 Types primitifs et structures de contrôle

1.2.1 Types primitifs

Comme tout langage de programmation évolué, Java possède un certain nombre de types de données de base :

- le type `boolean` qui prend une des 2 valeurs `true` ou `false` sur 1 octet ;
- le type `char` qui correspond à un caractère sur 2 octets ;
- les types `byte, short, int` et `long` qui sont 4 types d'entiers stockés respectivement sur 1, 2, 4 et 8 octets ;
- les types `float` et `double` qui sont 2 types de flottants stockés respectivement sur 4 et 8 octets.

Les déclarations de telles variables peuvent se faire n'importe où dans le code mais avant leur utilisation, comme en C++.

On utilise les opérateurs usuels sur les variables de ces types avec la syntaxe du C.

1.2.2 Structures de contrôle

Ce sont essentiellement les mêmes constructions qu'en C, à savoir

- L'affectation qui se fait par l'opérateur = et qui consiste à recopier la valeur de la variable primaire à droite du symbole dans la variable située à gauche.
- Les structures conditionnelles :
  - `if (cond) instruction1; [else instruction2;]`
  - `switch (selecteur) {`

```
case c1 : instructions1;
...
case cn : instructionsN;
```

```
default : instructionsNP;
}
```

- Les structures répétitives ou boucles :
  - `for (initialisation; condition; instructionDeSuite)`,
  - `while (condition) instruction;`
  - `do instruction; while (condition)`.

1.3 Classes et objets en Java

1.3.1 Définition d'une classe

Une classe permet de définir un type d'objets associant des données et des opérations sur celles-ci appelées *méthodes*. Les données et les *méthodes* sont appelés *composants* de la classe. L'exemple de base décrit dans un paragraphe suivant permettra de définir une classe `Vecteur` représentant des vecteurs au sens mathématique. A chaque vecteur, seront rattachés :

- une structure de données - un tableau - pour le stockage de ses coefficients ;
- des traitements comme les suivants,
  - son addition avec un autre vecteur,
  - son produit scalaire avec un autre vecteur,
  - ...
  - et aussi son procédé de création.

1.3.2 Déclaration, création et destruction d'objets

Pour manipuler un objet (par exemple, un vecteur particulier de la classe `Vecteur`), on doit tout d'abord déclarer une *référence* sur la classe correspondant au type d'objet. Cette opération permet de réserver en mémoire une adresse qui référencera l'objet. Par exemple, pour déclarer un objet x de type `Vecteur` on écrira :

```
Vecteur x;
```

Pour que cet objet soit réellement construit, c'est à dire que la référence désigne un emplacement à partir duquel on pourra accéder aux caractéristiques de l'objet, on devra appeler l'opération `new` qui alloue cet emplacement en mémoire et le renvoie à la référence de l'objet. On dispose alors d'une nouvelle *instance* de l'objet. Par exemple, pour la construction effective de notre vecteur x, on écrira :

```
x = new Vecteur();
```

Cette instruction fait appel à un procédé de construction d'objets, appelé *constructeur*, qui est défini par défaut, mais qui peut aussi être redéfini comme opération dans la classe `Vecteur`. Si cette instruction de construction n'est pas effectuée, la référence associée à `x` est initialisée par défaut à `NULL`, qui est une adresse fictive ne pointant vers aucun emplacement mémoire réel.

Par ailleurs, si l'on a besoin de référencer l'objet courant dans la définition de la classe, cette référence se fait par le mot réservé `this` qui vaut donc l'adresse de l'objet courant dans une *instance* de la classe.

La destruction des objets est pris en charge par le *Garbage Collector*, appelé encore *ramasse-miettes* dans sa dénomination francisée. Cet outil logiciel fonctionne en même temps que le programme, il recherche, identifie et supprime de la mémoire les objets qui ne sont plus référençables.

On peut toutefois ajouter à chaque classe un service `finalize()`, qui sera appelé au moment de la destruction de l'objet, s'il est utile d'effectuer des opérations spécifiques à cet instant.

Comme nous l'avons indiqué précédemment, le nom d'un objet permet de définir une *référence*, c'est à dire une adresse. Il est alors possible de faire une affectation entre deux objets de même nature. Une telle opération va donc recopier l'adresse de l'objet affecté. Ainsi une affectation est notablement différente lorsqu'elle se fait entre des variables de type simple (int, char, double, ...) ou entre des objets.

Le passage de paramètres dans les fonctions se fait par valeur, comme en C. Ainsi, soit le paramètre est de type simple alors on recopie sa valeur dans celui de la fonction appelée, soit c'est un objet, on recopie alors l'adresse référencée par l'objet (qui est bien sa valeur) dans celui de la fonction appelée.

1.3.3 Tableaux

Un tableau va permettre de stocker un certain nombre d'éléments de même type dans une structure de données qui dispose d'un index pour y accéder. Un tableau en Java est un objet à part entière.

Par exemple, un tableau monodimensionnel de flottants de type `double` sera déclaré de la manière suivante :

```
double monTableau[];
```

ou encore

```
double[] monTableau;
```

Comme nous l'avons expliqué précédemment, cette déclaration a permis d'affecter une référence au nom du tableau une référence (c'est à dire une adresse,

initialisée par défaut à `NULL`). Pour construire effectivement le tableau, c'est à dire disposer de plusieurs emplacements mémoire, par exemple 10, qui lui sont propres, on invoque l'opération de construction d'objets suivante :

```
monTableau = new double[10];
```

Ces deux instructions de constructions peuvent s'écrire en une seule fois :

```
double[] monTableau = new double[10];
```

Il est également possible d'initialiser la construction en affectant un tableau constant de la manière suivante :

```
double[] montableau = {0.0, 1.1, 3.5};
```

Les tableaux en tant qu'objets proposent un certain nombre d'opérations, notamment ils possèdent une donnée propre `length` qui renvoie la taille du tableau.

Par exemple, les instructions suivantes permettent d'afficher le contenu du tableau précédent, en utilisant la méthode[1] d'affichage standard sur l'écran `System.out.print`[2] :

```
for (int i=0; i<montableau.length; i++)
    System.out.print(montableau[i]+" ");
```

L'exemple précédent nous apprend par ailleurs que, comme en C, le premier indice d'un tableau est `0`.

On dispose aussi de mécanismes de vérification de dépassement de bornes des tableaux que l'on peut interroger par l'intermédiaire des techniques d'*exceptions*[3] que l'on décrira après. L'exception concernée se nomme `ArrayIndexOutOfBoundsException`.

1.3.4 Construction de la classe vecteur

Nous allons contruire maintenant une classe `Vecteur` permettant de manipuler des vecteurs au sens mathématique. C'est une classe élémentaire allégée de certains concepts de programmation objet qui seront présentés et introduits progressivement dans la suite de cet ouvrage.

Dans cette classe, on utilise une structure de données, appelée `composant`, correspondant à un tableau où seront stockés les coefficients du vecteur.

On définit deux *constructeurs* qui sont des méthodes portant le nom de la classe et qui ne renvoie pas de résultat :

[1] voir 1.3.1

[2] voir 1.7 pour une description générale des méthodes d'entrées/sorties

[3] voir 1.6

– Le premier constructeur construit un vecteur dont le nombre de composants est donné en paramètre.
– Le second constructeur construit un vecteur en recopiant un tableau passé en paramètre.

Par l'intermédiaire de ces deux *constructeurs*, il est ainsi possible de définir des méthodes de même nom, à condition qu'elles diffèrent au niveau du type ou du nombre de paramètres ou de leur résultat renvoyé. On parle alors de *surcharge* de méthodes.

On définit différentes méthodes :
– `elt` renvoyant la valeur de sa composante dont l'indice est donné en paramètre ;
– `toElt` permettant de modifier la composante dont l'indice et la nouvelle valeur sont passés en paramètres ;
– `dim` renvoyant la taille du vecteur ;
– `afficher` affichant les valeurs de ses composantes ;
– `add` renvoyant un vecteur qui est la somme du vecteur courant avec le vecteur passé en paramètre ;
– `prodScalaire` renvoyant le produit scalaire du vecteur courant avec le vecteur passé en paramètre.

Cet exemple permet d'illustrer la façon dont on accède aux composants de l'objet courant, en invoquant simplement leur nom, mais aussi la façon dont on accède aux composants d'un objet extérieur, en invoquant le nom de la composante precédée d'un point et du nom de l'objet en question.

L'écriture de la classe `Vecteur` est la suivante,

```
class Vecteur {
    double[] composant;

    // constructeurs
    Vecteur(int dim) { composant = new double[dim]; }

    Vecteur(double tableau[]) { composant = tableau; }

    // acces a la composante i
    double elt(int i) { return composant[i]; }

    // modification de la composante i
    void toElt(int i, double x) { composant[i] = x; }

    // renvoie sa tailla
    int dim() { return composant.length; }
```

```
    // affiche ses composantes
    void afficher() {
      for (int i=0; i<dim(); i++)
          System.out.print(elt(i)+" ");
      System.out.println("");
    }

    // renvoie sa somme avec le vecteur en parametre
    Vecteur add(Vecteur x) {
      Vecteur w = new Vecteur(dim());
      for (int i=0; i<dim(); i++)
          w.toElt(i, elt(i) + x.elt(i));
      return w;
    }

    // renvoie son produit scalaire avec le vecteur en parametre
    double prodScalaire(Vecteur x) {
      double p = 0;
      for (int i=0; i<dim(); i++)
          p += elt(i)*x.elt(i);
      return p;
    }
}
```

Nous donnons dans le listing suivant un exemple de classe qui va contenir un programme principal, c'est-à-dire une méthode de type `public static void main(String args[])` . Le paramètre `args` correspond à d'éventuels arguments d'appel.

```
class TestVecteur {
    public static void main(String args[]) {
        double []t1 = {1.0, 2.0, 3.0};
        double []t2 = {5.5, 7.5, 9.5};

        Vecteur x1 = new Vecteur(3);
        for (int i=0; i<x1.dim(); i++)
            x1.toElt(i, t1[i]);
        System.out.println("premier vecteur :");
        x1.afficher();

        Vecteur x2 = new Vecteur(t2);
        System.out.println("deuxieme vecteur :");
        x2.afficher();
```

```
        Vecteur x3 = x1.add(x2);
        System.out.println("leur somme vaut :");
        x3.afficher();

        double produit=x1.prodScalaire(x2);
        System.out.println("leur produit scalaire vaut : "+ produit);
    }
}
```

Les commandes à utiliser pour compiler et exécuter le programme seront décrites au paragraphe 1.4.2. Le programme génère alors l'affichage suivant à l'exécution :

```
premier vecteur :
1.0 2.0 3.0
deuxieme vecteur :
5.5 7.5 9.5
leur somme vaut :
6.5 9.5 12.5
leur produit scalaire vaut : 49.0
```

1.3.5 Composants de type `static`

Il est possible de définir des *composants*[4], données ou méthodes, qui ne sont pas rattachés de manière propre à chaque objet instancié, c'est-à-dire à chaque instance de la classe, mais qui sont communs à toutes. Pour cela il suffit de déclarer le composant avec le qualificatif `static`.

Par exemple, on peut ajouter à la classe vecteur une donnée entière qui va compter le nombre d'objets de la classe qui ont été instanciés. On peut également remplacer la méthode `add` par une méthode qui est `static` et qui prend 2 objets `Vecteur` en paramètres : on redonne à l'écriture de cette fonction une apparence de symétrie sur ses arguments correspondant à la propriété de communtativité de l'addition. Ci-dessous nous avons partiellement réécrit la classe `Vecteur` en prenant en compte ces modifications.

```
class Vecteur {
    double[] composant;
    static int nb =0;

    // constructeurs
```

[4] voir 1.3.1

```
    Vecteur(int dim) {
        composant = new double[dim];
        nb++; System.out.println("creation de l'objet "+nb);
    }

    Vecteur(double tableau[]) {
        composant = tableau;
        nb++; System.out.println("creation de l'objet "+nb);
    }

    ...

    // renvoie sa somme avec le vecteur en parametre
    static Vecteur add(Vecteur x, Vecteur y) {
      vecteur w = new vecteur(x.dim());
      for (int i=0; i<x.dim(); i++)
          w.toElt(i, x.elt(i) + y.elt(i));
      return w;
    }

    ...

}
```

Pour accéder à la nouvelle méthode `add`, on procédera de la manière suivante :

```
double[] t1 = {1.0, 3.2, 5.3};
double[] t2 = {3.0, 4.1, 6.3};
Vecteur x1 = new Vecteur(t1);
Vecteur x2 = new Vecteur(t2);
Vecteur x3 = Vecteur.add(x1, x2);
```

1.3.6 Composants de type `public` et de type `private`

Une notion fondamentale en programmation objet consiste à séparer, dans la description ou l'implémentation des objets, les parties visibles de l'extérieur et que l'on appelle *interface* de celles qui n'ont pas besoin d'être connues à l'extérieur de l'objet.

Les composants de la première partie porteront alors le qualificatif `public` et ceux de la seconde le qualificatif `private`. Dans notre exemple de classe vecteur, nous avons défini les opérations d'accès en lecture (fonction `elt`) et en écriture (fonction `toElt`) dans un objet, si bien qu'il n'est jamais utile d'accéder

au tableau `composant` interne à la classe. Ainsi nous déclarerons `public` les deux fonctions `elt` et `toElt` mais `private` le tableau `composant`.

L'intérêt de séparer ainsi les parties publiques des parties privées est de garantir une évolutivité possible des classes, sans avoir à modifier les programmes qui les utilisent, à partir du moment où l'on conserve leur interface publique. Ainsi la classe `vecteur` pourra utiliser des types de structures de données autres que des tableaux pour stocker ses composantes. On pourra par exemple, utiliser un stockage directe sur fichier (pour des vecteurs de dimension importante) ou encore un stockage spécifique pour des structures creuses (en ne stockant que les coefficients non nuls et leur position). Il suffira alors de redéfinir correctement les deux fonctions d'accès en lecture (`elt`) et en écriture (`toElt`) en respectant leur mode d'appel. Tous les programmes utilisant des classes vecteurs n'auront alors pas lieu de subir la moindre modification pour pouvoir utiliser ces nouveaux types de vecteurs.

La classe vecteur pourra alors être partiellement réécrite avec des parties publiques et privées, comme ci-dessous :

```
class Vecteur {
    private double[] composant;
    static int nb =0;

    // acces a la composante i
    public double elt(int i) { return composant[i]; }

    // modification de la composante i
    public void toElt(int i, double x) { composant[i] = x; }

    ...

}
```

L'utilisation de ces fonctions n'est pas affectée par ces déclarations supplémentaires : on introduit simplement des limitations aux composants comme décrit précédemment.

Il est à noter qu'en n'indiquant ni `public` ni `private`, les composants sont considérés comme étant définis `public` par défaut, sauf si l'on se trouve dans un autre *package* que celui de la classe considérée - nous reviendrons sur cette nuance dans le paragraphe 1.4.3.

1.3.7 Chaînes de caractères

Les chaînes de caractères en Java sont des objets, *instances* de la classe prédéfinie `String`, et elles référencent des chaînes constantes. On pourra les

déclarer comme dans l'exemple suivant :

```
String ch1 = new String("bonjour");
```

ou encore, sous une forme condensée qui est spécifique au type `String` :

```
String ch1 = "bonjour";
```

La chaîne `"bonjour"` est ici constante mais `ch1` peut être réaffectée pour référencer une autre chaîne constante, comme dans l'exemple suivant :

```
String ch2 = "au revoir";
ch1 = ch2;
```

L'ensemble des méthodes de la classe String peut être obtenu en consultant la documentation de l'API (Application Programming Interface) associée au JDK utilisé. Cette documentation, au format HTML, se révèle insdispensable dans la pratique, pour pouvoir accéder aux descriptions des interfaces des nombreuses classes proposées dans Java. Toutefois nous allons examiner quelques unes des méthodes les plus utiles relatives à la classe String :

- La méthode `static String valueOf(int i)` renvoie une chaîne contenant la valeur de `i`. Cette fonction existe aussi pour des paramètres de tous les types primaires. Notons que c'est une méthode statique qui s'appelle, par exemple, de la manière suivante : `String.valueOf(12)` et qui retourne ici la chaîne "12".
- La méthode `boolean equals(String s)` compare le contenu de la chaîne courante avec la chaîne s.
- La méthode `String concat(String s)` renvoie la concaténation de la chaîne courante (celle qui va préfixée la fonction `concat`) et de `s`. Il faut noter toutefois que les concaténations de chaînes peuvent se faire simplement avec l'opérateur +, comme on peut le remarquer dans les appels de la fonction d'affichage dans certains des exemples qui précèdent.
- La méthode `int length()` renvoie la longueur de la chaîne courante.
- La méthode `int indexOf(int c)` renvoie la position de la première occurence du charactère de code ASCII `c`. Elle renvoie `-1` si ce caractère n'apparaît pas.
- La méthode `char charAt(int i)` renvoie le caractère à la position `i`.

Nous avons vu que les objets `String` référencent des chaînes constantes. On peut, en fait, travailler avec des chaînes modifiables, en utilisant des objets de la classe prédéfinie `StringBuffer` dont on va décrire, sommairement, les méthodes essentielles. Là encore, pour plus d'informations, on consultera la documentation en ligne de l'API.

Les différents constructeurs de la classe `StringBuffer` sont :

– `StringBuffer()` permettant de créer une chaîne vide ;
– `StringBuffer(int dim)` permettant de créer une chaîne de longueur `dim` ;
– `StringBuffer(String s)` permettant de créer une chaîne contenant `s`.

Les principales méthodes de la classe `StringBuffer` sont :
– `int length()` renvoyant la longueur de la chaîne ;
– `StringBuffer append (String s)` ajoutant `s` à la fin de la chaîne courante ;
– `String toString()` renvoyant dans une chaîne constante la chaîne modifiable courante.

1.4 Organisation des fichiers sources d'un programme Java

1.4.1 Structure des fichiers sources

Un programme Java correspond à une collection de classes dans un ou plusieurs fichiers sources dont l'extension est `"java"`. L'un de ces fichiers doit contenir une classe qui implémente la méthode `public static void main(String args[])`, comme cela est fait dans l'exemple précédent de construction de la classe vecteur et de son programme de test.

1.4.2 Commandes de compilation et de lancement d'un programme

Pour compiler et exécuter les programmes java on utilise les outils fournis avec le JDK. On commence par lancer la compilation des fichiers avec la commande `javac`. Par exemple, pour les deux fichiers relatifs à notre classe vecteur et à son programme de test, on écrira :

```
javac Vecteur.java
javac TestVecteur.java
```

Deux fichiers : `Vecteur.class` et `TestVecteur.class` ont été générés et correspondent aux noms de toutes les classes définies dans les fichiers sources. Ce sont des fichiers en byte-code portables sur toute machine devant être traités par la machine virtuelle java lancée par la commande `java`. On exécute donc le programme principal, qui est dans la classe `TestVecteur`, en tapant la commande :

```
java TestVecteur
```

1.4.3 Packages

Un package permet de regrouper un ensemble de classes. L'instruction `package nomPack` en début de fichier indique que les classes qui y sont définies appartiennent au package nommé `nomPack`. Pour accéder aux classes de ce package, lorsque l'on est dans une classe qui n'y appartient pas, on utilise la dénomination `nomPack.className`, où `className` désigne le nom de la classe.

Les désignations de package suivent un schéma de construction arborescent du type : `name.subname.subsubname`. Il existe un lien entre les noms de package et les répertoires où se trouvent les classes y appartenant . Par exemple, une classe `watch` appartenant au package `time.clock` doit se trouver dans le fichier `time/clock/watch.class`.

Les répertoires où Java effectue sa recherche de packages sont définis dans la variable d'environnement : `CLASSPATH`.

L'instruction `import packageName` permet d'utiliser des classes du package défini, sans avoir besoin de les préfixer par leur nom de package. On peut importer toutes les classes d'un package en utilisant un import du type `import packageName.*;`, mais, **ATTENTION**, l'import d'un niveau de package ne permet pas d'importer les packages qui sont en-dessous dans l'arborescence des répertoires.

Voici un exemple d'illustration qui montre une organisation de classes java, dans différents répertoires et leur utilisation.

La variable d'environnement `CLASSPATH` doit être dans le fichier `.profile` ou dans `.bashrc`, par exemple, sous Unix, de la manière suivante :

```
CLASSPATH = ~/myJavaClass
```

Voici maintenant des extraits de différents fichiers rangés dans les répertoires indiqués :

– le fichier /myJavaClass/bibMat/calVecteur/Vecteur.java correspond à

```
package bibMat.calVecteur;
public class Vecteur { ... }
```

– le fichier /myJavaClass/bibMat/calMatrice/Matrice.java correspond à

```
package bibMat.calMatrice;
public class Matrice { ... }
```

– le fichier /myJavaClass/calUtil/TestBibMat.java correspond à

```
package calUtil;
import bibMat.calMatrice.*;
public class TestBibMat {
  public static void main (String args[]) {
    bibMat.calVecteur.Vecteur x =
        new bibMat.calVecteur.Vecteur(3);
    Matrice M = new Matrice(3, 3);
    ...
  }
}
```

1.4.4 Visibilité des composants dans les packages

On a vu précédemment que l'accessibilité des composants d'une classe se fait grâce aux qualificatifs `private` ou `public`. En fait, elle est également liée aux localisations dans les packages. Ainsi, lorsqu'un composant ne précise pas sa nature (`private` ou `public`) alors il est, par défaut, `public` dans les classes du package (ou du répertoire) auquel il appartient et `private` en dehors.

Nous illustrons ces propos avec l'exemple suivant :

– Package P1 :

```
class C1 {
  public int xa;
  int xc;
  private int xd;
}
class C2 { ... }
```

– Package P2 :

```
class C3 { ... }
```

Dans cet exemple, la classe `C2` peut accéder à `xa` et `xc`. Par contre `C3` ne peut accéder qu'à `xa` uniquement.

1.4.5 packages prédéfinis en Java

Le langage Java possède un grand nombre de packages prédéfinis regroupés par thèmes. Ces packages et les classes qu'ils contiennent sont décrits de manière exhaustive dans une documentation fournie par Sun et qui reprend l'ensemble des interfaces. Cette documentation est appelée API (Application Programming Interface) et est distribuée sous un format HTML, permettant ainsi une navigation hy-

pertexte particulièrement adaptée aux recherches d'informations nécessaires pour le développement de programmes. On y trouve principalement :

- `java.lang` qui correspond aux classes de base (chaînes, math, ...),
- `java.util` qui correspond aux structures de données (vector, piles, files, ...),
- `java.io` qui correspond aux entrées/sorties,
- `java.awt` qui correspond au graphisme et fenêtrage,
- `java.net` qui correspond aux communications Internet,
- `java.applet` qui correspond aux insertions dans des documents HTML.

1.5 Héritage

1.5.1 Construire une classe dérivée

La notion d'héritage est importante en programmation objet. Elle permet de définir une classe dérivée à partir d'une autre classe dont on dit qu'elle *hérite*. La classe dérivée possède alors, par défaut, l'ensemble des composants de la classe dont elle hérite - on l'appelle classe mère - sauf si des restrictions ont été posés sur ces composants. L'héritage est donc un concept essentiel renforçant les propriétes de réutilisabilité des programmes objets.

L'utilisation répétée de l'héritage sur des classes successives conduit à la construction d'une hiérarchie entre elles, que l'on peut schématiser par un arbre d'héritage. La figure 1.1 présente un exemple d'arbre d'héritage construit sur des classes permettant de représenter des figures géométriques.

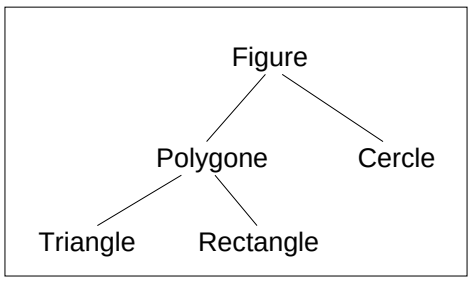

FIG. 1.1: Arbre d'héritage de classes d'objets géométriques

La construction effective des différentes classes de la figure 1.1 est faite dans l'exemple complet du paragraphe 1.5.5.

Pour définir une classe ClB qui dérive de la classe ClA, on fera une déclaration du type :

```
class ClB extends ClA { ... }
```

Un objet de la classe ClB est alors aussi un objet de la classe ClA, il peut être utilisé partout où un objet de la classe ClA est attendu.

On peut tester l'appartenance d'un objet à une classe grâce à l'opérateur `instanceof`, comme dans l'exemple qui suit :

```
ClB x;
if ( x instanceof ClA)
    System.out.println(''voici un objet ClA !'');
```

L'exécution des lignes précédentes provoque l'affichage de "`voici un objet ClA !`".

1.5.2 Constructeur d'une classe dérivée

Si l'on définit un constructeur d'une classe dérivée, celui-ci doit explicitement faire appel à un constructeur de la classe mère, auquel on accède avec la méthode prédéfinie `super( ... )`. Si cet appel explicite n'est pas fait, l'exécution du programme provoquera un appel implicite du constructeur par défaut, c'est à dire sans paramètre, de la classe mère ; il faut donc impérativement que celui-ci existe sinon une erreur de compilation sera diagnostiquée. Ainsi, si l'on a défini un ou des constructeurs dans la classe mère, une de ses versions devra être sans paramètre.

L'appel explicite du constructeur de la classe mère se fait par la méthode prédéfinie `super( ... )`, cet appel est obligatoirement la première instruction du constructeur. On ne peut donc transmettre que des valeurs de paramètres du constructeur courant lors de l'appel de `super( ... )`.

On retrouve de telles constructions dans l'exemple du paragraphe 1.5.5 où la classe `Figure` possède un composant de type `Point` correspondant à son origine et pouvant être initialisé par le paramètre `x` du constructeur `Figure(Point p)`. On définit la classe `Cercle` dérivant de `Figure`, dont le constructeur est défini par :

```
Cercle (Point centre, double r)
{ super(centre); ... }
```

1.5.3 Accessibilité : public, protected et private

On a vu précédemment que les composants d'une classe pouvaient être éventuellement qualifiés des termes `public` ou `private`. Il existe, en fait, un

troisième qualificatif `protected` qui indique que ce composant est accessible uniquement dans les classes dérivées et les classes du même package.

Par exemple, soit la classe ClA définie de la manière suivante :

```
package aa;
public class ClA {
    protected int JJ;
    ...
}
```

et la classe ClB définie ainsi :

```
package bb;
import aa.*;
class ClB extends ClA {
    void PP() {
        JJ++;  // autorisé
        ClB b;
        b.JJ++ // autorisé
        ClA a;
        a.JJ++ // interdit
    }
}
```

1.5.4 Méthodes virtuelles et classes abstraites

Une classe peut annoncer une méthode sans la définir, on dit alors que la classe est abstraite. Elle doit être introduite avec le mot clé `abstract`.

Par exemple, on peut définir la classe abstraite `ClA` suivante et une classe `ClB` qui en dérive.

```
abstract class ClA {
    abstract void fctP() ;
    void fctQ() { ... };
    ...
}
class ClB extends ClA {
    void fctP() { ... };
    ...
}
```

En raison de sa définition incomplète, il est impossible d'*instancier* une classe abstraite qui ne sert qu'à la construction de classes dérivées. Ces dernières devront redéfinir toutes les méthodes abstraites, pour ne pas l'être elles-mêmes et pouvoir ainsi être instanciées.

1.5.5 Un exemple : quelques objets géométriques

Nous donnons un exemple d'implémentation des différentes classes décrites sur la figure 1.1. Nous commençons par définir une classe `Point` qui servira dans les autres classes :

```
class Point {
  double abscisse;
  double ordonnee;
  Point(double x, double y)
    {abscisse=x; ordonnee=y;}
  Point(Point p)
    {abscisse=p.abscisse; ordonnee=p.ordonnee;}
  static double distance(Point p, Point q) {
    double dx=p.abscisse-q.abscisse;
    double dy=p.ordonnee-q.ordonnee;
    return Math.sqrt(dx*dx+dy*dy);
  }
}
```

Nous définissons ensuite une classe abstraite pour définir le type `Figure` constitué de deux méthodes abstraites d'affichage et de calcul de périmètre :

```
abstract class Figure {
  private static final Point zero=new Point(0,0);
        Point origine;
  Figure(){origine=zero;}
  Figure(Point p){origine=new Point(p);}
  abstract double perimetre();
  abstract void affiche();
}
```

La classe `Cercle` qui suit dérive de la classe `Figure` en implémentant ses deux méthodes abstraites `perimetre` et `affiche` :

```
class Cercle extends Figure {
  private static final double pi=3.141592;
  double rayon;
  Cercle(Point centre, double r)
    {super(centre); rayon=r;}
  double perimetre()
    {return 2*pi*rayon;}
  void affiche() {
    System.out.println("Cercle");
```

```
        System.out.println("rayon : " + rayon +" et centre : " +
                           "(" + origine.abscisse +
                           "," + origine.ordonnee + ")  ");
        }
    }
```

La classe `Polygone` dérive de la classe `Figure`. Elle se caractérise par un tableau de `Point` :

```
class Polygone extends Figure {
  Point sommet[]= new Point[100];
  int nbs;
  Polygone(){nbs=0;}
  Polygone(Point[] m, int n) {
    super(m[0]);
    nbs=n;
    for (int i=0; i<n; i++)
      sommet[i]=m[i];
  }
  double lcote(int i) {
    if (i<nbs)
      return Point.distance(sommet[i-1], sommet[i]);
    else
      return Point.distance(sommet[i-1], sommet[0]);
  }
  double perimetre() {
    double somme=0;
    for (int i=1; i<=nbs; i++)
      somme += lcote(i);
    return somme;
  }
  void affiche() {
    System.out.println("Polygone");
    for (int i=0; i<nbs; i++)
      System.out.print("(" + sommet[i].abscisse +
                       "," + sommet[i].ordonnee + ")  ");
    System.out.println();
  }
}
```

La classe `Triangle` est une classe élémentaire dérivée de la classe `polygone` :

```
class Triangle extends Polygone {
```

```
  Triangle(Point[] m) { super(m,3); }
}
```

La classe `Rectangle` dérive de la classe `Polygone`. Elle est définie avec ses caractéristiques mathématiques classiques, c'est à dire la longueur de ses côtés et un de ses sommets, le sommet inférieur gauche. On a alors une bonne illustration de l'utilisation des constructeurs successifs des classes dérivées. Pour appeler le constructeur de `Polygone` qui possède le tableau de ses sommets en paramètres, il faudrait tout d'abord faire la construction de ce tableau à partir des caractéristiques de `Rectangle`, ce qui n'est pas possible, car l'appel de `super` doit être la première opération. Il faut donc utiliser le constructeur par défaut dans la classe `Polygone` qui sera appelé au début de l'exécution du constructeur de `Rectangle`. La composante `sommet` de la classe sera alors construite plus loin :

```
class Rectangle extends Polygone {
  double largeur;
  double longueur;
  Rectangle(Point m, double lo, double la) {
     // appel implicite du constructeur de
     // Polygone sans parametre
     // l'appel du constructeur de Polygone avec parametres
     // ne peut se faire car il faut d'abord construire
     // le tableau a lui transmettre qui ne peut se faire
     // qu'apres l'appel explicite ou implicite de super
     Point P1= new Point(m.abscisse+ lo, m.ordonnee);
     Point P2= new Point(m.abscisse, m.ordonnee+ la);
     Point P3= new Point(m.abscisse+ lo, m.ordonnee+ la);
     Point mr[]={m, P1, P3, P2};
     sommet=mr;
     nbs=4;
     largeur= la;
     longueur= lo;
  }
}
```

Voici un programme principal de test :

```
class Geometrie {
  public static void main(String args[]) {
    Point P1= new Point(3,4);
    Point P2= new Point(4,4);
    Point P3= new Point(0,0);
    Point P4= new Point(1,0);
```

```
        Point P5= new Point(0,1);
        Point [] TabP={P3, P4, P5};
        Cercle c= new Cercle(P1,2);
        Rectangle r= new Rectangle(P2,5,2);
        Triangle t= new Triangle(TabP);
        Figure f; //autorise, mais pas new Figure !
        System.out.println("perimetre cercle");

        f=c; f.affiche(); // appel de affiche de Figure
                          // puis de sa forme derivee de cercle
        System.out.println("perimetre : " + f.perimetre());

        f=r; f.affiche();
        System.out.println("perimetre : " + f.perimetre());

        f=t; f.affiche();
        System.out.println("perimetre : " + f.perimetre());
      }
    }
```

L'exécution du programme affiche :

```
java Geometrie
perimetre cercle
Cercle
rayon : 2.0 et centre : (3.0,4.0)
perimetre : 12.566368
Polygone
(4.0,4.0)  (9.0,4.0)  (9.0,6.0)  (4.0,6.0)
perimetre : 14.0
Polygone
(0.0,0.0)  (1.0,0.0)  (0.0,1.0)
perimetre : 3.414213562373095
```

1.5.6 Interfaces

Une *interface*, en Java, permet de décrire un modèle de construction de classe dans lequel on n'indique uniquement que les en-têtes des méthodes. Cela équivaut, d'une certaine manière, à une classe où toutes les méthodes sont abstraites.

On dira qu'une classe *implémente* une *interface*, si elle redéfinit toutes les méthodes décrites dans cette *interface*.

Par exemple, on définit un modèle de problème par une interface qui comporte deux méthodes `poserProbleme` et `resoudreProbleme` :

```
interface AResoudre {
    void poserProbleme();
    void resoudreProbleme();
}
```

On peut alors construire une classe `equationPremierDegre` qui implémente l'interface `aResoudre` :

```
class EquationPremierDegre implements AResoudre {
    double coefx, coef1, solution;
    void poserProbleme()
    {  // on lira coefx et coef1 }
    void resoudreProbleme()
    {  // on affecte une valeur a solution }
    ...
}
```

Un autre intérêt des interfaces provient de la limitation de Java en terme d'héritage multiple. En effet, Java ne permet pas qu'une classe dérivée puisse avoir plusieurs classes mères et donc de bénéficier des composants définis dans ces différentes classes. Pour pallier cette limitation, on devra utiliser conjointement l'héritage et l'implémentation d'interfaces. Il est, par ailleurs, possible d'implémenter plusieurs interfaces en Java, contrairement à l'héritage.

1.5.7 Passage d'une fonction en paramètre d'une méthode

Nous nous intéressons au problème du passage d'une fonction en tant que paramètre dans une méthode. Par exemple, on souhaite décrire dans une classe un procédé qui approche le calcul d'une dérivée d'une fonction réelle d'une variable réelle par un taux d'accroissement :

$$f'(x) \simeq \frac{f(x+h/2) - f(x-h/2)}{h}$$

Nous allons montrer comme ce procédé peut être décrit d'une manière générique, c'est à dire en utilisant une fonction abstraite f, paramètre du procédé. On applique alors ce procédé de dérivation numérique à une fonction particulière en la transmettant par l'intermédiaire de ce paramètre.

Dans beaucoup de langages, comme le C ou le C++, le passage d'une fonction en paramètre s'effectue en gérant un pointeur qui contient l'adresse effective du code de la fonction. Le langage Java ne proposant pas de gestion explicite de pointeur, on doit alors créer une *enveloppe* de type objet contenant une méthode

correspondante à l'évaluation de la fonction. C'est cette classe *enveloppe* qui correspond au paramètre à gérer.

Il faut donc commencer par définir une classe abstraite, ou une interface, qui décrit les fonctionnalités minimales de la classe correspondant au paramètre fonctionnel.

Nous donnons un exemple qui s'appuie sur l'utilisation d'une interface minimale caractérisant une fonction réelle d'une variable réelle. Les nombres réels sont implémentés par le type `double` :

```
interface FoncD2D { public double calcul(double x); }
```

Nous pouvons alors utiliser cette interface pour décrire un procédé générique de calcul de dérivation numérique, comme décrit ci-dessus :

```
class DiffFinies {
    public double derivOrdre1 (FoncD2D f, double x, double h) {
        return (f.calcul(x+h/2) - f.calcul(x+h/2))/h ;
    }
}
```

Pour utiliser ce procédé, il suffit maintenant de définir une fonction particulière dans une classe qui implémente l'interface `FoncD2D` :

```
class FoncCarre implements FoncD2D {
    public double calcul (double x) { return x*x ; }
}
```

Le programme principal suivant va alors construire un objet de la classe `FoncCarre` qui permet d'utiliser la fonction particulière qui y est définie. Il construit aussi un objet de la classe `DiffFinies` qui permet d'utiliser le procédé de dérivation numérique qui est invoqué sur l'objet de type `FoncCarre`, reconnu comme une implémentation de `FoncD2D` :

```
class TestDiffFinies {
    public static void main (String args[]) {
        FoncCarre f = new FoncCarre();
        DiffFinies df = new DiffFinies();
        System.out.println ("Différences finies d'ordre un "+
                            "en 1 de pas 0.01 : "+
                            df.derivOrdre1(f, 1, 0.01));
    }
}
```

Le résultat de l'exécution est :

```
java TestDiffFinies
Différences finies d'ordre un en 1
de pas 0.01 : 1.9999999999999685
```

1.6 Exceptions

1.6.1 Notions générales

L'introduction de la notion d'*exception* dans un langage de programmation a pour but de simplifier le traitement de certaines situations. Ces dernières sont considérées comme exceptionnelles, au sens où leur détection nécessite de les gérer, en les sortant du contexte dans lequelle elles sont détectées.

Dans les langages ne gérant pas spécifiquement ces situations d'exception, il est nécessaire d'utiliser de nombreuses instructions conditionnelles successives, afin de réorienter le déroulement du programme de manière adéquate. De tels processus peuvent conduire à une complexité du programme dont la lecture finit par s'alourdir.

Certains langages proposent de manière facultative l'utilisation des exceptions (c'est le cas du C++). Avec Java, on a l'obligation de les gérer lorsque certains appels de méthodes sont succeptibles, de par leur conception, de déclancher des traitements d'exception.

Une gestion d'exception est caractérisée par une séquence

```
try - Catch - finally
```

qui correspond typiquement au déroulement suivant :

```
try {
    //séquence succeptible de déclancher une exception
    ...
}
catch (classException e1) {
    //traitement à effectuer si e1 a été déclanchée
    ...
}
catch ( ....) { ... } //autre déclanchement éventuel
finally {
    //traitement effectué avec ou sans déclanchement d'exception
    ...
}
```

Nous donnons ci-dessous un exemple de programme qui récupère les arguments fournis au lancement du programme. Il calcule et affiche la moyenne de ces

arguments lorsque ce sont des entiers. Il déclanche un traitement d'exception si l'un des arguments ne correspond pas à un entier.

```
class exceptionCatch {
    static int moyenne (String[] liste) {
        int somme=0, entier, nbNotes=0, i;
        for (i=0; i<liste.length; i++)
            try {
                entier=Integer.parseInt(liste[i]);
                // conversion chaîne en valeur entière
                somme += entier; nbNotes++;
            }
            catch (NumberFormatException e) {
                System.out.println("note: "+(i+1)+" invalide");
            }
        return somme/nbNotes;
    }

    public static void main (String[]argv) {
        System.out.println("moyenne "+moyenne(argv));
    }
}
```

Une exécution possible du programme est la suivante :

```
java exceptionCatch ha 15 12 13.5
note: 1 invalide
note: 4 invalide
moyenne 13
```

1.6.2 Définir sa propre exception

Pour définir sa propre exception, il faut définir une classe qui hérite de la classe prédéfinie `Exception`. On pourra surcharger, en particulier, la méthode prédéfinie `toString()` dont la chaîne renvoyée correspond au message affiché, lorsque l'on demande d'afficher l'exception elle-même.

Une méthode succeptible de déclancher une exception devra avoir une clause `throws`, suivie de l'exception déclanchable dans son en-tête. Dans le corps de la méthode, on précisera dans quelle condition l'exception est déclanchée, en invoquant l'opérateur `throw`.

On donne ci-après un complément du programme précédent déclanchant une exception, lors du calcul de la moyenne d'un tableau, s'il ne contient pas d'éléments.

```
class ExceptionRien extends Exception {
    public String toString() {
        return ("aucune note !");
    }
}
class ExceptionThrow {
    static int moyenne (String[] liste) throws ExceptionRien {
        int somme=0, entier, nbNotes=0, i;
        for (i=0; i<liste.length; i++)
            try {
                entier=Integer.parseInt(liste[i]);
                // conversion chaîne en valeur entière
                somme += entier; nbNotes++;
            }
            catch (NumberFormatException e) {
                System.out.println("note: "+(i+1)+" invalide");
            }
        if (nbNotes == 0) throw new ExceptionRien();
        return somme/nbNotes;
    }
    public static void main (String[]argv) {
        try {
            System.out.println("moyenne "+moyenne(argv));
        }
        catch (Exception e) {
            System.out.println(e);
        }
    }
}
```

Voici deux exécutions successives du programme précédent :

```
java exceptionThrow q d 1 3 5.2
note: 1 invalide
note: 2 invalide
note: 5 invalide
moyenne 2

java exceptionThrow q d 3.1 a 5.2
note: 1 invalide
note: 2 invalide
note: 3 invalide
note: 4 invalide
note: 5 invalide
```

```
aucune note !
```

1.7 Entrées/Sorties

1.7.1 Classes de gestion de flux

Dans le langage Java, deux types d'entrées/sorties sont utilisables :
- Les entrées/sorties traditionnelles, c'est-à-dire utilisant les flux de communications "par défaut", à savoir le clavier ou l'écran, ou encore les fichiers ;
- Les entrées/sorties basées sur des interactions avec un système de fenêtrage. Celles-ci seront développées dans le chapitre sur le graphisme.

Nous présentons, dans la suite, des notions basées sur l'API 1.1 et qui ont été enrichies significativement par rapport aux versions précédentes. Les raisons de ces enrichissements sont principalement dues à des questions d'efficacité et à la possibilité d'utiliser des codes internationaux (UNICODE) qui enrichissent le code ASCII avec des caractères accentués, entre autres.

Les principales classes de gestion de flux sont organisées suivant la hiérarchie d'héritage décrite dans la figure 1.2.

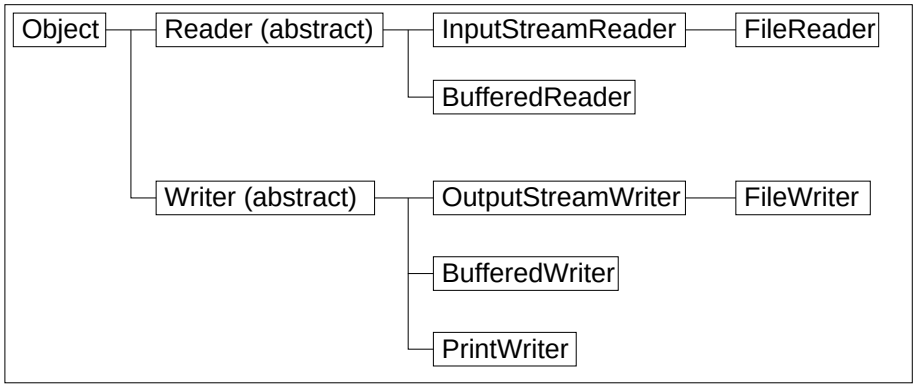

FIG. 1.2: Hiérarchie des classes de gestion de flux

Dans cette hiérarchie, les classes suivantes apparaissent :
- La classe `Object` qui est la classe de base en Java, dont toutes les autres héritenté ;
- Les classes abstraites `Reader` et `Writer` qui concernent respectivement les flux de caractères pour les lectures et les écritures ;
- Les classes `InputStreamReader` et `OutputStreamWriter` qui permettent de faire la traduction des données brutes en caractères UNICODE et inversement ;

– Les classes `BufferedReader` et `BufferedWriter` qui permettent l'utilisation d'une mémoire tampon pour les entrées-sorties. Cette mémoire est indispensable pour l'utilisation des périphériques standards (écran et clavier) ;
– Les classes `FileReader` et `FileWriter` qui permettent l'utilisation de fichiers ;
– La classe `PrintWriter` qui permet l'écriture de données formattées semblables aux affichages à l'écran.

Cette liste n'est pas exhaustive mais correspond aux principales classes que nous serons amenés à utiliser dans la suite.

1.7.2 Saisies au clavier

Pour effectuer une saisie au clavier, on construit successivement :
– Un flux `InputStreamReader` avec le flux de l'entrée standard, à savoir `System.in` ;
– Un flux de lecture bufferisé de type `BufferedReader` à partir du flux précédent.

On présente, ci-après, un exemple typique de lecture au clavier. Dans cet exemple, on lit dans le flux bufferisé avec la méthode `readLine()` permettant la lecture d'un chaîne de caractères jusqu'à ce que l'on rencontre un saut de ligne. La chaîne de caractère est alors convertie en entier, avec la méthode `() parseInt`, ou en flottant, avec un construction plus complexe qui utilise la méthode `floatValue()`, sur l'objet de la classe `Float` obtenu en appelant `valueof()`. Cette dernière méthode est statique et a pour paramètre la chaîne lue. Il faut noter qu'à partir du JDK 1.2, on pourra plus simplement appeler la méthode `parseFloat()`, similaire à `parseInt()`.

On remarquera que, dans cet exemple, il est nécessaire de gérer le déclanchement éventuel d'exceptions peuvant se produire,
– soit par un problème de lecture de flux qui provoque le déclanchement de l'exception `IOException` ;
– soit par un problème de conversion de chaîne de caractères en valeur numérique qui provoque l'exception `NumberFormatException`.

```
import java.io.*;
class Testio {
  public static void main(String[] args) {
    InputStreamReader fluxlu = new InputStreamReader(System.in);
    BufferedReader lecbuf = new BufferedReader(fluxlu);

    try {
```

```
      System.out.print("taper 1 ligne de caracteres : ");
      String line = lecbuf.readLine();
      System.out.println("ligne lue : " + line);

      System.out.print("taper 1 nombre entier : ");
      line = lecbuf.readLine();
      int i = Integer.parseInt(line);
      System.out.println("entier lu : " + i);

      System.out.print("taper 1 nombre reel : ");
      line = lecbuf.readLine();
      float f = Float.valueOf(line).floatValue();
      System.out.println("réel lu : " + f);

      System.out.println("somme des deux nombres : " + (i+f) );
    }
    catch(IOException e) {
      System.out.println("erreur de lecture"); }

    catch(NumberFormatException e) {
      System.out.println("erreur conversion chaine-entier"); }
  }
}
```

L'affichage obtenu à la suite de l'exécution du programme est le suivant :

```
java Testio
taper 1 ligne de caracteres : java sait lire
ligne lue : java sait lire
taper 1 nombre entier : 3
entier lu : 3
taper 1 nombre reel : 12.5
réel lu : 12.5
somme des deux nombres : 15.5
```

1.7.3 Lecture d'un fichier

Une lecture dans un fichier est effectué dans le programme suivant. Il est similaire au précédent avec, toutefois, quelques différences :

- `FileReader` est la classe instanciée pour construire le flux d'entrée à partir du nom du fichier. Cette instanciation pourra déclancher l'exception `FileNotFoundException`, si le fichier n'est pas trouvé.
- La méthode `close()` ferme le fichier.

– On utilise une boucle qui détecte la fin de fichier, suite au résultat d'une lecture qui renvoie la constante `null`.

```
import java.io.*;
class Testfile {
  public static void main(String[] args) {
    FileReader fichier=null;
    BufferedReader lecbuf;

    String line;
    float f, somme=0;
    int nbnombre=0;

    try {
      fichier = new FileReader("donnee.dat");
      lecbuf = new BufferedReader(fichier);

      while ( (line = lecbuf.readLine()) != null ) {
        f = Float.valueOf(line).floatValue();
        somme += f; nbnombre++;
        System.out.println("nombre lu : " + f);
      }
      if ( nbnombre > 0 )
        System.out.println("Moyenne : " + (somme/nbnombre));
    }
    catch(FileNotFoundException e) {
      System.out.println("fichier donnee.dat inexistant !"); }

    catch(IOException e) {
      System.out.println("erreur de lecture"); }

    catch(NumberFormatException e) {
      System.out.println("erreur conversion chaine-entier"); }

    finally {
      if (fichier!=null)
        try { fichier.close(); }
        catch(IOException e) {}
    }
  }
}
```

On exécute le programme précédent avec le fichier `"donnee.dat"` suivant :

```
2.3
1.2
3.4
2.1
5.2
```

L'affichage, produit par le programme, est alors le suivant :

```
java Testfile
nombre lu : 2.3
nombre lu : 1.2
nombre lu : 3.4
nombre lu : 2.1
nombre lu : 5.2
Moyenne : 2.84
```

1.7.4 Ecriture dans un fichier

Le programme suivant va utiliser un fichier dans lequel on écrit. On construit un flux de type `FileWriter`, utilisé dans un tampon d'écriture (de type `BufferedWriter`). Un flux, de type `PrinterWriter`, permet alors d'effectuer des écritures formattées avec la méthode `println`, comme on le fait couramment sur la sortie standard, c'est-à-dire l'écran.

```
import java.io.*;
class Testfileout {
  public static void main(String[] args) throws IOException {
    FileWriter fichier = new FileWriter("out.txt");
    BufferedWriter ecrbuf = new BufferedWriter(fichier);
    PrintWriter out = new PrintWriter(ecrbuf);
    out.println("coucou");
    out.println(5.6);
    System.out.println("fin d'ecriture dans le fichier out.txt");
    out.close();
  }
}
```

1.7.5 Compléments

Il est possible de lire et d'écrire des données brutes (donc non formattées) avec les classes `DataInputStream` et `DataOutputStream`. Les fichiers, ainsi construits, ne sont pas lisibles directement sous un éditeur de texte, par exemple, mais leur taille est plus réduite.

La classe `StringTokenizer` est une classe qui permet d'instancier un petit analyseur de texte qui peut, par exemple, découper des lignes en sous-chaînes de caractères, en reconnaissant un certain nombre de séparateurs (blancs, virgules, ...).

La classe `java.io.File` permet de faire des manipulations de fichiers, similaires aux commandes d'un système d'exploitation. Elle permet par exemple, de lister un répertoire, de renommer ou supprimer un fichier, etc.

1.8 Conclusion provisoire

Nous terminons ici cette introduction à Java. Des compléments seront donnés au fur et à mesure de leur besoin. On abordera la partie graphisme dans le chapitre 4.

Chapitre 2

Résolution des équations non linéraires dans $\mathbb{R}$

Dans ce chapitre, on présente un résumé de cours sur quelques méthodes classiques de résolution numérique des équations non linéaires dans $\mathbb{R}$,

$$F(x) = 0, \quad x \in \mathbb{R}. \tag{2.1}$$

La généralisation aux systèmes d'équations non linéaires à plusieurs variables ($x \in \mathbb{R}^n$) n'est pas traité dans ce volume, (mais une introsduction en est donnée sous forme d'exercice, présenté dans le chapitre 3, voir exercice 25) on pourra consulter les références bibliographiques en fin de volume, par exemple, [**?**], ...

Soit $F : \mathbb{R} \longrightarrow \mathbb{R}$ une application continue, on se propose de trouver la (ou les) solution(s) de l'équation (2.1). Excepté quelques cas très simples, par exemple pour les polynôme de degré inférieur ou égal à trois, les méthodes analytiques de résolution d'équations ne permettent pas de résoudre de tels problèmes. Le recours aux méthodes numériques, fournissant des solutions approchées, devient alors incontournable.

2.1 Localisation (ou séparation) des racines

La plupart des méthodes numériques supposent que l'on connaisse un intervalle contenant la racine cherchée et aucune autre. On dit alors qu'elle est localisée ou séparée, des autres éventuelles racines.
Les deux méthodes les plus classiques pour localiser ou séparer les racines sont :

1. L'étude des variations de F, puis l'utilisation du théorème de la valeur intermédiaire.

2. La réécriture de l'équation $F(x) = 0$ sous la forme $F_1(x) = F_2(x)$, puis la recherche de(s) point(s) d'intersection des courbes représentatives de F_1 et de F_2. Le problème sera plus simple à traiter si l'une des deux fonctions et l'identité. La solution cherchée correspond alors à un point fixe de l'autre fonction. Voir ci-dessous.

Remarque 1 *: On supposera dans la suite que F est continue et que la racine $\overline{x}$ est localisée dans un intervalle borné [a,b].*

2.2 Méthode des approximations successives

Dans cette méthode, on construit à partir de l'équation à résoudre, la suite

$$\left\{ \begin{array}{l} x_{n+1} = g(x_n), \\ x_0 \text{ choisi dans } [a, b], \end{array} \right. \tag{2.2}$$

dans l'espoir qu'elle tende vers la solution $\overline{x}$ du problème. La limite de cette suite vérifie $\overline{x} = g(\overline{x})$.

Définition 1 . *Soit $g : \mathbb{R} \longrightarrow \mathbb{R}$, un point fixe de g est un réel $\overline{x}$ solution de l'équation $x = g(x)$.*

L'équation $F(x) = 0$ peut s'écrire, sous la forme $x = g(x)$. On peut choisir par exemple

$$g(x) = F(x) + x. \tag{2.3}$$

Ainsi, résoudre $F(x) = 0$ revient à trouver les points fixes de $g(x)$.

Exemple 1 . *Soit l'équation*

$$F(x) = x^2 - x - 3\ln(x) = 0,$$

elle peut sécrire, sous la forme $x = g(x)$ avec

- $g(x) = x^2 - 3\ln(x) = x$,
- $g(x) = \sqrt{x + 3\ln(x)} = x$,
- $g(x) = e^{(x^2-x)/3} = x$.

où il faut choisir l'écriture permettant la convergence du processus itératif $x_{n+1} = g(x_n)$ vers la solution cherchée, lorsque cette convergence est possible.

Exemple 2 . *Soit l'équation*

$$F(x) = 2\sin(x) - x\cos(x) = 0,$$

les valeurs $x = \pi/2 + k\pi$ n'étant pas racines de $F(x) = 0$, on peut écrire cette équation, entre autres, sous la forme : $g(x) = 2\tan(x) = x$, sur un intervalle $I_k =]\pi/2 + k\pi, \pi/2 + (k+1)\pi[$ qui isole une racine et sur lequel $g(x)$ est définie, les racines de $F(x) = 0$ sont les points fixes de $g(x)$.

Définition 2 . *Soit $g : \mathbb{R} \longrightarrow \mathbb{R}$, s'il existe $k \in [0, 1[$ tel que, pour tous y et z éléments de $\mathbb{R}$, on ait $|g(y) - g(z)| \leq k|y - z|$, alors on dit que g est contractante (ou est une contraction) sur $\mathbb{R}$. Si l'inégalité est stricte, on dira que g est strictement contractante. k est appelé rapport de contraction.*

Proposition 1 . *Soit g une application dérivable sur l'intervalle $[a, b]$, si la dérivée g' vérifie, $max_{x\in[a,b]}|g'(x)| = k < 1$, alors g est strictement contractante dans $[a, b]$.*

Le théorème suivant fournit une condition suffisante de convergence du processus des approximations successives.

Théorème 1 (du point fixe) . *Soit $g : \mathbb{R} \longrightarrow \mathbb{R}$ vérifiant les conditions suivantes :*

1. *$x \in [a, b] \Longrightarrow g(x) \in [a, b]$, (c'est à dire $g[a, b] \subset [a, b]$)*
2. *g est une application strictement contractante dans $[a, b]$, de rapport de contraction k,*

alors pour tout $x_0 \in [a, b]$ la suite récurrente définie par,

$$\begin{cases} x_0 \in [a, b] \\ x_{n+1} = g(x_n), \quad n \in \mathbb{N} \end{cases}$$

converge vers l'unique solution $\overline{x}$ de $x = g(x)$, avec $\overline{x} \in [a, b]$. De plus, la formule suivante donne une majoration de l'erreur commise à l'itération n, lors du calcul de $\overline{x}$,

$$|x_n - \overline{x}| \leq \frac{k^n}{1-k}|x_1 - x_0|.$$

Il est souvent délicat de déterminer un intervalle $[a, b]$ dans lequel les hypothèses du théorème du point fixe sont vérifiées, cependant on a le résultat suivant.

Proposition 2 *: Si g est dérivable au voisinage d'un point fixe $\overline{x}$, et si $|g'(\overline{x})| < 1$, alors il existe un voisinage $\mathcal{V}_{\overline{x}}$ de $\overline{x}$ tel que pour tout x_0 dans $\mathcal{V}_{\overline{x}}$, les itérations, $x_{n+1} = g(x_n)$, convergent vers $\overline{x}$.*

Proposition 3 *: Soit $\overline{x}$ une solution de l'équation $x = g(x)$, si g' est continue au voisinage de $\overline{x}$ et si $|g'(\overline{x})| > 1$, alors pour toute condition initiale x_0, $x_0 \neq \overline{x}$, la suite définie par x_0 et $x_{n+1} = g(x_n)$ ne converge pas vers $\overline{x}$.*

———————————-
———————————-
———————————-

ICIIIIIIIIIIIIIIIIIII fin corrections faites le dim 11 nov 01 IIIIIIIIIIIIIIIIIIci

Mettre ici la remarque ecrite dans l'exo 5 vieille p :57

Remarque : Attention ...

———————————-
———————————-
———————————-

Résumé et Interprétation graphique :

On estime $g'(\overline{x})$,

1. si $|g'(\overline{x})| > 1$, (figure 1), la suite ne converge pas vers $\overline{x}$, c'est un point *répulsif*, les itérés successifs de la condition initiale x_0 (ou ceux de x_0') s'éloignet de $\overline{x}$, on élimine alors la méthode.

2. si $|g'(\overline{x})| < 1$, (figures 2 et 3), la méthode converge, $\overline{x}$ est un point *attractif*, les itérés successifs de la condition initiale x_0 (ou ceux de x_0') se rapprochent de plus en plus de $\overline{x}$, (il faut alors déterminer un intervalle $[a, b]$ contenant $\overline{x}$, dans lequel $max_{x\in[a,b]}|g'(x)| < 1$ et $g([a, b]) \subset [a, b]$), deux cas se présentent :
   (a) $0 \leq g'(\overline{x}) < 1$, (figure 2),
   si $x_0 \leq \overline{x}$ la suite (x_n) est croissante, elle est décroissante sinon (cas où la condition initiale est x_0'). Dans les deux cas, elle tend vers $\overline{x}$.
   (b) $-1 < g'(\overline{x}) \leq 0$, (figure 3),
   la suite (x_n) est alternée, deux itérés successifs de x_0 donnent un encadrement de $\overline{x}$ et la suite converge vers $\overline{x}$.

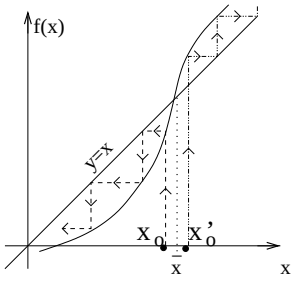

Figure 1.

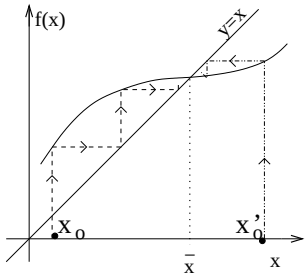

Figure 2.

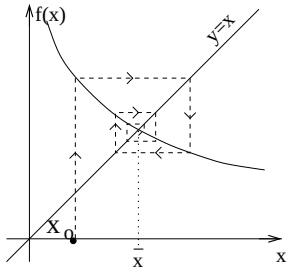

Figure 3.

(c) si $|g'(\overline{x})| = 1$, c'est un cas plus délicat car la méthode peut converger (figure 4) ou non (figure 5) :

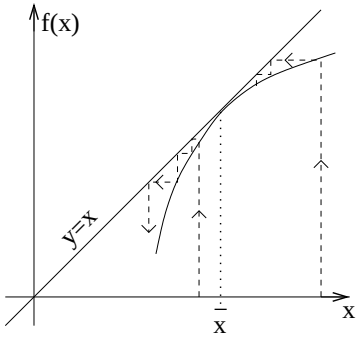

Figure 4.

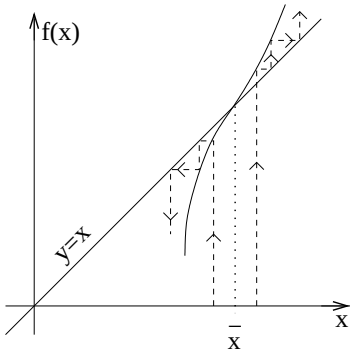

Figure 5.

Dans la figure 4, les itérés successifs de x_0 convergent vers $\overline{x}$ si $x_0 > \overline{x}$, et divergent dans le cas contraire. Dans le cas de la figure 5, ils diveegent quelque soit la position de x_0.

2.3 Ordre d'une méthode

Une méthode itérative pouvant converger plus ou moins rapidement (voir les figures 6 et 7), il existe des notions qui mesurent cette rapidité de convergence, et mesurent donc la diminution de l'erreur $e_n = x_n - \overline{x}$ d'une itération à la suivante.

Définition 3 *: La méthode définie par x_0 et $x_{n+1} = g(x_n)$ est dite d'ordre p si $\frac{|e_{n+1}|}{|e_n|^p}$ a une limite réelle strictement positive lorsque n tend vers $+\infty$. On dit alors que la méthode à un taux de convergence égal à p.*

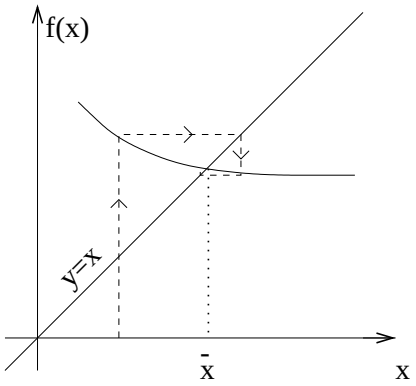

Figure 6.

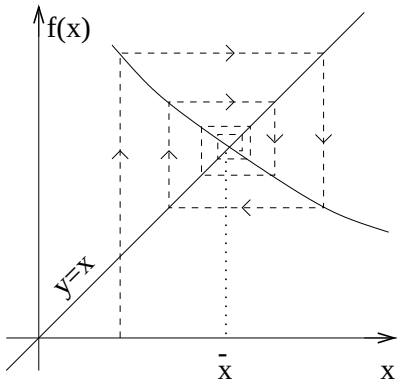

Figure 7.

Théorème 2 *: Si la suite (x_n) converge vers $\overline{x}$ et si g est suffisamment dérivable au voisinage de $\overline{x}$ alors l'ordre de la suite (x_n) est le plus petit entier p tel que :*
$g'(\overline{x}) = \cdots = g^{(p-1)}(\overline{x}) = 0$ *et* $g^{(p)}(\overline{x}) \neq 0$*. De plus on a*

$$\lim_{n \to +\infty} \frac{e_{n+1}}{{e_n}^p} = \frac{1}{p!} g^{(p)}(\overline{x})$$

où $e_n = x_n - \overline{x}$*.*

Ce théorème donne une estimation de l'erreur e_n à la n^{eme} itération.

Remarque 2 *: Une méthode est dite linéaire lorsqu'elle est d'ordre 1, on a alors* $g'(\overline{x}) \neq 0$*. Elle est dite quadratique si elle est d'ordre 2.*

Remarque 3 *: Pour une méthode d'ordre p, lorsqu'on passe d'un itéré à l'autre, le nombre de chiffres décimaux exacts est environ multiplié par p.*

Tests d'arrêt des itérations.
Si le processus itératif converge, la solution $\overline{x}$, étant la limite de la suite, ne peut être atteinte qu'au bout d'un nombre infini d'itérations. Par conséquent l'algorithme sera arrêté en utilisant l'un des tests suivants, où ε désigne la précision à laquelle on souhaite obtenir la solution,

(i) $|x_n - x_{n-1}| \leq \varepsilon$
(ii) $\frac{|x_n - x_{n-1}|}{|x_n|} \leq \varepsilon$
(iii) $|g(x_n) \leq \varepsilon|$, c'est à dire $g(x_n)$ est presque nulle.

Le processus sera considéré comme non convergent, vers la solution cherchée, si la précision ε souhaitée n'est pas atteinte au dela d'un nombre 'raisonable' d'itérations, N_{max}, fixé à l'avance.

Remarque 4 *En pratique, lorsque on cherche la racine $\overline{x}$ à 10^{-m} près, les itérations seront arrêtées lorsque deux itérés successifs x_p et x_{p+1} présentent les mêmes décimales, de la première à la m^{eme} .*

2.4 Exemples de méthodes itératives

2.4.1 Méthode de Lagrange (ou de la corde)

Soit à résoudre l'équation $F(x) = 0$, où l'application F est continue, strictement monotone sur $[a, b]$ telle que $F(a)F(b) < 0$, et soit $\overline{x}$ la racine cherchée, située dans $[a, b]$. Dans cette méthode on remplace le graphe de F restreint à $[a, b]$ par le segment de droite joignant les points $A(a, F(a))$ et $B(b, F(b))$, (c'est a dire que l'on interpole F par un polynome de degré 1) ; ce segment coupe l'axe ox en un point d'abscisse x_1. Soit M_1 le point de coordonnées $(x_1, F(x_1))$, on reitère alors le même procédé qu'auparavant en joignant M_1 et l'extremité fixe, A ou B. L'extremité fixe est celle telle que $F(x)F''(x) > 0$. Voir la figure 8 ci-dessous.

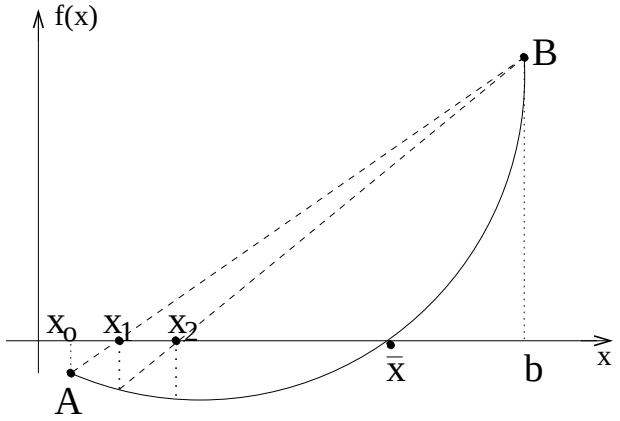

Figure 8.

En écrivant l'équation de la corde (AB) on obtient la méthode de Lagrange ci-dessous.

1. si l'extremité A est fixe, cette méthode s'écrit $x_{n+1} = g(x_n)$ et x_0 donné, où :
$$\begin{cases} x_0 = b \\ x_{n+1} = x_n - F(x_n)\frac{x_n - a}{F(x_n) - F(a)} = g(x_n). \end{cases}$$
C'est une suite décroissante et bornée, donc convergente, ($a < \overline{x} < ... < x_n < ... < x_1 < x_0$).
2. si l'extremité B est fixe, elle s'écrit :
$$\begin{cases} x_0 = a \\ x_{n+1} = x_n - F(x_n)\frac{b - x_n}{F(b) - F(x_n)} = g(x_n). \end{cases}$$
C'est une suite croissante et bornée, donc convergente ($x_0 < x_1 ... < x_n < .... < \overline{x} < b$).

FIGUREsq 9 a, b , c ,d

Exemple 3 $F(x) = x^3 + x - 1 = 0$ *sur* $[\frac{1}{2}, 1]$, *on a* $F(\frac{1}{2}) = -0.375$ *et* $F(1) = 1$ *donc* $\overline{x} \in [\frac{1}{2}, 1]$. *D'autre part* $F''(x) > 0$ *sur* $[\frac{1}{2}, 1]$, *ainsi* $F(1)F''(1) > 0$, *c'est donc l'extremité* $B(1, F(1))$ *qui sera fixe, figure 9(c). Par conséquent, en commencant le processus par* $x_0 = 1$, *on trouve aisément :* $x_1 = 0.64..., x_2 = 0.67$, *... etc ..., ainsi,* $\overline{x} = 0.67$ *à* 10^{-2} *près.*

2.4.2 Méthode de Newton

L'idée consiste cette fois ci à remplacer l'arc de courbe (AB) par la droite tangente à la courbe aux points A ou B, l'intersection de cette dernière avec l'axe ox détermine le second élément x_1 de la suite cherchée. On reitère en passant la tangente à la courbe au point d'abscisse x_1. etc...

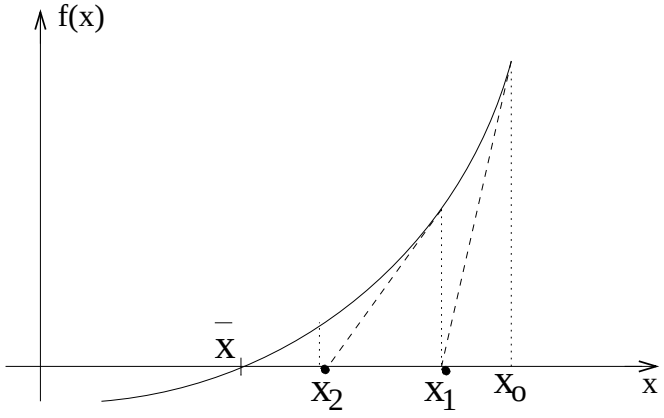

Figure 10.

D'une manière générale les itérations de Newton pour résoudre $F(x) = 0$ sont :

$$\begin{cases} x_{n+1} = x_n - \frac{F(x_n)}{F'(x_n)}, & \text{n=0,1,2 ...} \\ x_0 \text{ donnée dans } [a, b] \end{cases}$$

Cette méthode est bien définie pour x_n 'voisin' de $\overline{x}$ à condition que $F'(x_n) \neq 0$.
Déterminer un intervalle $[a, b]$ sur lequel il y'a convergence de Newton n'est pas toujours facile, mais le théorème suivant donne une condition suffisante.

Théorème 3 (convergence globale de la méthode de Newton) *Si* $F \in C^2[a,b]$ *vérifie :*

1. $F(a)F(b) < 0$,
2. $\forall x \in [a,b]\ F'(x) \neq 0$ *(stricte monotonnie de* F*),*
3. $\forall x \in [a,b]\ F''(x) \neq 0$ *(concavité de* F *dans le même sens),*

Alors en choisissant $x_0 \in [a,b]$ *tel que* $F(x_0)F''(x_0) > 0$*, les itérations de Newton convergent vers l'unique solution* $\overline{x}$ *de* $F(x) = 0$ *dans [a,b].*

Théorème 4 *: Si* $\overline{x}$ *est un zéro simple, la méthode de Newton est au moins quadratique.*

(D'une itération à la suivante, le nombre de chiffres décimaux exacts de la solution cherchée est multiplié environ par deux, donc x_{n+1} présentera deux fois plus de décimales exactes que x_n).

Exemple 4 $F(x) = x^3 + x - 1 = 0$ *sur* $[\frac{1}{2}, 1]$*, on a* $F(\frac{1}{2}) = -0.375$ *et* $F(1) = 1$ *donc* $\overline{x} \in [\frac{1}{2}, 1]$.

exple ... A FINNNNNNNNNNNIIIIIIIIR

Par conséquent, en commencant le processus par $x_0 = 1$*, on trouve aisément :* $x_1 = 0.64...$, $x_2 = 0.67$*, ... etc ..., ainsi,* $\overline{x} = 0.67$ *à* 10^{-2} *près.*

2.5 Accéleration de la convergence

Une méthode itérative peut être accélerée en transformant la suite (x_n) en une suite (y_n) convergeant plus rapidement, vers la même limite, ou en transformant F de façon à obtenir une méthode d'ordre plus élevé. Voici quelques méthodes simples d'accéleration de la convergence.

2.5.1 Méthode d'Aitken, ou Procédé Δ^2 d'Aitken

Soit (x_n) une suite qui tend vers $\overline{x}$, le procédé d'Aitken consiste à transformer (x_n) en (y_n) où,

$$y_n = \frac{x_n x_{n+2} - {x^2}_{n+1}}{x_{n+2} - 2x_{n+1} + x_n} \quad n = 0, 1, 2 \cdots$$

On démontre que la suite (y_n) converge plus rapidement que la suite (x_n) vers la même limite $\overline{x}$. Mais il est conseillé d'eviter cette formulation de (y_n) qui est numériquement instable, et de l'écrire sous la forme équivalente suivante, mais plus stable :

$$y_n = x_{n+1} + \frac{1}{\frac{1}{x_{n+2} - x_{n+1}} - \frac{1}{x_{n+1} - x_n}} \quad n = 0, 1, 2 \cdots$$

2.5.2 Méthode de Steffensen, exemple de composition de méthodes

Lorsqu'on dispose de deux méthodes pour résoudre la même équation, on peut les composer pour obtenir une méthode plus rapide.
La méthode de Steffensen compose une méthode itérative $x_{n+1} = g(x_n)$ avec le procédé Δ^2 d'Aitken : d'où les itérations suivantes :

$$x_{n+1} = \frac{x_n g(g(x_n)) - g^2(x_n)}{g(g(x_n)) - 2g(x_n) + x_n} \quad n = 0, 1, 2 \cdots$$

De même, cette écriture n'est pas numériquement stable, d'où la formulation suivante, plus stable :

$$u_0 = x_n, \;\; u_1 = g(u_0), \;\; u_2 = g(u_1), x_{n+1} = u_1 + \frac{1}{\frac{1}{u_2 - u_1} - \frac{1}{u_1 - u_0}} \quad n = 0, 1, 2 \cdots$$

2.5.3 Méthode de Regula-Falsi

Dans la méthode de Newton, le calcul de $F'(x_n)$ peut s'averer très coûteux en temps de calcul, très complexe ou même impossible, on remplacera alors cette derivée par son approximation

$$F'(x_n) = \frac{F(x_n) - F(x_{n-1})}{x_n - x_{n-1}}$$

d'où la méthode de Regula-Falsi :

$$\begin{cases} x_0 \text{ et } x_1 \text{ donnés,} \\ x_{n+1} = \dfrac{x_n F(x_{n-1}) - x_{n-1} F(x_n)}{F(x_{n-1} - F(x_n)} \qquad n = 1, 2 \cdots \end{cases}$$

On démontre que cette méthode est d'ordre $\frac{1+\sqrt{5}}{2} = 1.618...$, donc inférieur à celui de la méthode de Newton ; cependant Regula-Falsi peut s'avérer plus rapide que Newton puisque n'exigeant qu'une seule évaluation de fonction (celle de $F(x_n)$), $F(x_{n-1})$ ayant été calculé lors de l'itération précédente, alors que Newton en exige deux, $F(x_n)$ et $F'(x_n)$.
Ainsi, avec deux évaluations de fonctions, on effectue une itération de Newton et on multiplie environ par deux le nombre de décimales exactes (puisque cette méthode est quadratique, théorème 4) ; désignons par τ le temps nécessaire pour cette itération. Dans la méthode de Regula-Falsi, durant le même temps τ, on effectue approximativement deux itérations (puisqu'une seule évaluation de fonctions est faite par itération), et on multiplie ainsi à chaque itération, environ par $\left(\frac{1+\sqrt{5}}{2}\right)^2 \approx 2.62$, le nombre de décimales exactes, ce qui est meilleur et explique la rapidité de Regula-Falsi par rapport à Newton.

2.6 Enoncés des exercices corrigés

Exercice 1 :
Parmi les fonctions suivantes lesquelles sont contractantes et sur quel intervalle si celui-ci n'est pas indiqué :

(a) $g(x) = 5 - \frac{1}{4}\cos 3x$, $0 \leq x \leq \frac{2\pi}{3}$;
(b) $g(x) = 2 + \frac{1}{2}|x|$, $-1 \leq x \leq 1$;
(c) $g(x) = 3 - \frac{1}{2}\sin 3x$
(d) $g(x) = \sqrt{x+2}$.

Exercice 2 :
Voir si chacune des fonctions suivantes admet zero, un ou plusieurs points fixes, puis donner pour chacun un intervalle de separation :
$g(x) = \frac{1}{\sqrt{x}}$, $g(x) = e^{-x}$, $g(x) = x + (x-2)^3$, $g(x) = (x-2)^2 + x - \frac{e^x}{\pi}$

Exercice 3 :
Montrer que l'équation $x = \omega - \epsilon\sin(x)$ admet une unique racine dans l'intérvalle $[\omega - \pi, \omega + \pi]$; $\omega \in \mathbb{R}$, et $|\epsilon| \leq 1$.

Exercice 4 :
Déterminer à l'aide de la méthode du point fixe les deux racines réelles de $x^2 - 100x + 1 = 0$ avec une erreur $e \leq 10^{-3}$. Utiliser pour l'une de ces racines la méthode itérative $x = f(x) = \frac{x^2+1}{100}$, et pour l'autre $x = g(x) = 100 - \frac{1}{x}$.

Exercice 5 :
Soit la fonction $F(x) = 2x^3 - x - 2$, on se propose de trouver les racines réelles de F par la méthode des approximations successives.
Montrer que F possède une seule racine réelle $\bar{x} \in [1; 2]$.
Etudier la convergence des trois méthodes itératives suivantes : $x_0 \in [1; 2]$ donné et

(α) $x_{n+1} = 2x_n^3 - 2$;
(β) $x_{n+1} = \frac{2}{2x_n^2 - 1}$;
(γ) $x_{n+1} = \sqrt[3]{1 + \frac{x_n}{2}}$.

Si l'une de ces méthodes converge l'utiliser pour déterminer $\bar{x}$ à 10^{-3} près.

Exercice 6 :
Soit l'équation $x = \ln(1+x) + 0.2$ dans $\mathbb{R}^+$.
Montrer que la méthode itérative définie par $g(x) = \ln(1+x) + 0.2$ est convergente (vérifier les hypothèses du théorème du point fixe). Choisir x_0, condition initiale de l'itération, dans l'intervalle de convergence puis trouver $\bar{x}$ limite de la suite. Donner l'ordre de la méthode.

Exercice 7 :
On veut résoudre dans $\mathbb{R}^+$ l'équation $x = g(x)$ où, $g(x) = -\ln x$,
a) 1) Montrer qu'elle admet une seule racine $\bar{x}$, montrer que $\bar{x} \in I = [0; 1]$.
2) Montrer que la méthode itérative : $x_{n+1} = g(x_n)$ diverge.
3) on considère alors $g^{-1}(x) = g^{-1}og(x) = x$, (remarquer que g^{-1} existe), montrer que la méthode itérative : $x_{n+1} = g^{-1}(x)$ converge.
En posant $e_n = x_n - \bar{x}$ montrer que e_{n+1} est de signe opposé à e_n, qu'en conclut-on ?
Donner le bon test d'arrêt des itérations pour avoir $\bar{x}$ à $\epsilon = 10^{-4}$ près, puis donner cette racine approchée.
b) Retrouver $\bar{x}$ à l'aide de la méthode de Newton.
Remarquer que Newton est d'ordre 2.

Exercice 8 :
On applique la méthode de Newton au calcul de la racine carrée d'un réel positif α.
Montrer que les itérations $x_{n+1} = \frac{1}{2}(x_n + \frac{\alpha}{x_n}) \quad n = 0, 1, 2 \cdots$ permettent d'aboutir.
On pose $e_n = \frac{x_n - \sqrt{\alpha}}{\sqrt{\alpha}}$, $n = 0, 1, \ldots$. Montrer que $e_{n+1} = \frac{e_n^2}{2(1+e_n)}$, $n = 0, 1, \ldots$.
On applique la méthode de Regula-Falsi au même problème, expliciter les itérations obtenues.
Donner l'expression de e_{n+1} en fonction de e_n et e_{n-1}. En deduire que l'ordre p de Regula-Falsi verifie $1 < p < 2$.

Exercice 9 :

1. Démontrer que si f est une fonction continue dérivable d'un intervalle I de $\mathbb{R}$ dans lui même de dérivée négative et bornée sur I, alors on peut mettre l'équation $x = f(x)$ sous la forme $x = g(x)$, où g est une application contractante de I.
2. Utiliser la question précédente pour trouver la racine $\bar{x}$ de $x^2 - 2 = 0$ dans $[1, 2]$ avec une erreur $\leq 10^{-5}$; prendre $f(x) = \frac{2}{x}$.

2.7 Enoncés des exercices non corrigés

Exercice 10 :
Détérminer en utilisant la méthode de Lagrange la racine de $\overline{x}$ de l'équation $x - \tan x = 0$ sachant que $\overline{x} \in [\pi, 3\pi/2]$

Exercice 11 :
Détérminer en utilisant simultanément les méthode de Newton et de Lagrange la racine $\overline{x}$ située dans [0,1], avec une erreur $\leq 10^{-5}$ de $x - e^{-2x} = 0$, $2x - e^{-3x} = 0$, puis de $x^3 - 4x + 2 = 0$.
Retrouver ces racines grâce à la méthode des approximations successives.

Exercice 12 :
On cherche les racines réelles de l'équation $x^2 = \ln(1 + x)$.

1. Montrer que la fonction $F(x) = x^2 - \ln(1 + x)$ admet deux racines, l'une évidente qu'on donnera, et l'autre que l'on note $\overline{x}$ (que l'on veut approcher dans la suite).
2. Localiser $\overline{x}$ dans un intervalle de longueur 1/4.
3. Ecrire la méthode de Newton relative a F. Donner un choix de la condition initiale des itérations de Newton x_0 qui assure la convergence du processus.
4. Soient les méthodes d'approximations succesives suivantes :
   (a) $x_{n+1} = \sqrt{\ln(1 + x_n)}$ et (b) $x_{n+1} = e^{x_n^2} - 1$
   Préciser si elles convergent ou divergent. En cas de convergence indiquer un choix de la condition initiale x_0.

Exercice 13 :
Soit $F(x) = e^x + 2x - 2$, localiser graphiquement la racine $\bar{x}$ de $F(x) = 0$.
En donner une approximation avec une erreur inférieure à 10^{-5} en utilisant simultanément la méthode de Newton et de la corde.

Exercice 14 :
On cherche les racines réelles de l'équation $e^{-x} = x^2$. On pose

$$F(x) = e^{-x} - x^2$$

1. Montrer que l'équation $F(x) = 0$ admet une unique racine sur $\mathbb{R}$, que l'on notera $\tilde{x}$. Localiser $\tilde{x}$ entre deux entiers consécutifs ; soit I l'intervalle obtenu.
2. Soit la méthode d'approximation successive suivante :

$$x_{n+1} = x_n - e^{-x_n} + x_n^2, \quad x_0 \in I.$$

   Cette suite converge t-elle vers $\tilde{x}$, pourquoi ?

3. Soit la méthode d'approximation successive suivante :

$$x_{n+1} = e^{-x_n/2}, \quad x_0 \in I.$$

   Cette suite converge t-elle vers $\tilde{x}$, pourquoi ?.

4. Ecrire la méthode de Newton pour F. Donner une valeur explicite de x_0 qui assure la convergence de la méthode de Newton (si cette converegence est possible), justifier.
5. Si l'une des trois méthodes proposées ci-dessus converge, l'utiliser pour donner une approximation de $\tilde{x}$ à 10^{-2} près.

Exercice 15 :
On veut résoudre l'équation non linéaire dans $\mathbb{R}$, (1) : $F(x) = e^x(x-1) - x = 0$

1. Par la méthode du point fixe :
   (a) Montrer par une étude analytique, puis par une étude graphique, que l'équation (1) admet deux racines $\bar{x}_1$ et $\bar{x}_2$ que l'on localisera chacune dans un intervalle de longueur 1.
   (b) Proposer une méthode itétrative utilisant le point fixe : $x = g(x)$, la plus simple possible pour l'équation (1) (éviter la fonction log).
   (c) En utilsant le théorème du point fixe, et la méthode itérative $x_{n+1} = g(x_n)$ ci-dessus avec x_0 donné, montrer que l'une des deux racines (notée $\bar{x}_1$) est attractive, l'autre répulsive.
   (d) Montrer que deux itérés successifs de la suite donnent un encadrement de la racine $\bar{x}_1$.
   (e) donner en fonction de $k = max|g'(x)|$, et de e_0 (où$e_n = x_n - \bar{x}_1$), le nombre d'itérations N assurant la précision : $|x_N - \bar{x}_1| < 10^{-p},\ p \in \mathbb{N}$.
   (f) montrer que l'ordre de cette méthode est plus petit ou égal à 1. Le vérifier numériquementà la question ci dessous.
   (g) Donner alors cette racine à 10^{-2} près.
2. Par la méthode de Newton.
   (a) Ecrire la méthode de Newton relativeà l'équation (1).
   (b) Vérifier que cette méthode converge vers la racine $\bar{x}_1$ et donner une valeur explicite de la condition initiale x_0 qui assure cette convergence.
   (c) Donner alors cette racineà 10^{-2} prés.

Exercice 16 :
Soit F la fonction définie par $F(x) = \cos x - xe^x = 0, \quad 0 \leq x \leq \pi/2$.

1. Montrer que l'équation $F(x) = 0$ admet une unique racine $\bar{x}$ dans l'intervalle $[0, \pi/2]$.
2. Soit la méthode itérative suivante :

$$\begin{cases} x_{n+1} = \cos(x_n)e^{-x_n} = g(x_n) \\ x_0 \text{ choisi .} \end{cases}$$

   (a) Montrer qu'en choisissant un intervalle $[a, b] \subset [0, \pi/2]$, cette méthode peut converger vers $\bar{x}$.

   (b) Montrer que $|x_n - \bar{x}| \leq k^n |x_0 - \bar{x}|$, on explicitera k.

   (c) Montrer que $\bar{x} \in [0.45; 6]$. Déterminer k dans ce cas.

   (d) Donner le nombre d'itérations N assurant l'inégalité : $|x_N - \bar{x}| < 10^{-4}$.
3. Ecrire la méthode de Newton pour F. Donner une valeur explicite de x_0 qui assure la convergence de la méthode de Newton vers $\bar{x}$.

Exercice 17 :
Soit la fonction $F(x) = x^4 - 2x^3 + 1$, on se propose de trouver les racines réelles de F par la méthode des approximations successives (méthode du point fixe que l'on appliquera correctement), puis par la méthode de Newton.

1. (a) Montrer que F possède deux racines réelles, une évidente que l'on donnera, l'autre $\bar{x}$ que l'on localisera dans un intervalle I de longueur $1/2$.

   (b) Etudier la convergence, vers $\bar{x}$, des trois méthodes itératives suivantes : $x_0 \in I$ donné et :

   (i) $x_{n+1} = x_n^4 - 2x_n^3 + x_n + 1$,

   (ii) $x_{n+1} = \frac{-1}{(x_n - 2)^{1/3}}$,

   (iii) $x_{n+1} = 2 - \frac{1}{x_n^3}$,

   (c) Si l'une de ces méthodes converge l'utiliser pour déterminer $\bar{x}$ à 10^{-2} près.
2. (a) Ecrire la méthode de Newton pour F.

   (b) Vérifier le théorème de convergence globale de cette méthode sur un intervalle que vous donnerez.

   (c) Donner une valeur explicite de x_0 qui assure la convergence de la méthode de Newton vers $\bar{x}$.

(d) Déterminer alors $\bar{x}$ à 10^{-2} près.

Exercice 18 :
On veut résoudre l'équation non linéaire dans $\mathbb{R}$, : $F(x) = 0$, où F est une application deux fois continûment dérivable de $\mathbb{R}$ dans $\mathbb{R}$, on propose les itérations données par $x_{n+1} = g(x_n)$ et x_0 donné dans $\mathbb{R}$ ($n = 0, 1 \cdots$), où

$$(*) \qquad g(x) = x - \frac{F(x)F'(x)}{(F'(x))^2 - \frac{1}{2}F(x)F''(x)}$$

1. Montrer que l'ordre de convergence de cette méthode est au moins égal à 2.
2. Quel est le schéma itératif résultant de $(*)$ pour calculer $\sqrt[3]{\alpha}$ à partir de $F(x) = x^3 - \alpha$, ($\alpha \in \mathbb{R}$).
3. Partant de $x_0 = 1$, donner alors $\sqrt[3]{2}$ à 10^{-3} près .

Remarque : On rappelle qu'une méthode $x_{n+1} = g(x_n)$ est d'ordre p si $g(\bar{x}) = g'(\bar{x}) = \cdots = g^{(p-1)}(\bar{x}) = 0$ et $g^{(p)}(\bar{x}) \neq 0$ ($\bar{x}$ étant la racine cherchée).

2.8 Corrigés des exercices

Exercice 1

(a) $g(x) = 1 - \frac{1}{5}\sin(4x), x \in \mathbb{R}$.
Montrons que g est contractante sur $\mathbb{R}$, on a :

$$g'(x) = -\frac{4}{5}\cos(4x) \quad \text{et } |g'(x)| \leq \frac{4}{5},$$

donc, d'après la proposition 1, g est contractante de rapport de contraction inférieur ou égal à$\frac{4}{5}$.

(b) $g(x) = 2 + \frac{1}{2}|x|, x \in [-1, 1]$.
Soient $x, y \in [-1, 1]$, montrons que $|g(x) - g(y)| \leq \frac{1}{2}|x - y|$. On a

$$\begin{aligned} |g(x) - g(y)| &= \left|2 + \frac{1}{2}|x| - 2 - \frac{1}{2}|y|\right| \\ &= \frac{1}{2}\big||x| - |y|\big| \\ &\leq \frac{1}{2}|x - y|. \end{aligned}$$

En effet, d'une manière générale, on peut montrer que $\big||x| - |y|\big| \leq |x - y|$:
Supposons que $|x| \geq |y|$ alors

$$\begin{aligned} \big||x| - |y|\big| &= |x - y + y| - |y| \\ &\leq |x - y| + |y| - |y| \text{d'après l'inégalité triangulaire} \\ &\leq |x - y|. \end{aligned}$$

On fait de même si $|x| \leq |y|$, d'où le résultat $\big||x| - |y|\big| \leq |x - y|$.
Ainsi, $|g(x) - g(y)| \leq \frac{1}{2}|x - y|$ et le rapport de contraction est $k = \frac{1}{2}$.

(c) $g(x) = \frac{1}{x}, x \in [2, 3]$.
On a :

$$g'(x) = -\frac{1}{x^2} \quad \text{et} \quad 4 < x^2 < 9 \Leftrightarrow \frac{1}{9} \leq \left|-\frac{1}{x^2}\right| \leq \frac{1}{4}$$

donc, $\forall x \in [2, 3], \quad |g'(x)| \leq \frac{1}{4}$.
Ainsi, g est contractante de rapport $k \leq \frac{1}{4}$.

(d) $g(x) = \sqrt{x + 2}$.
g est définie sur $[-2, +\infty[$ mais n'y est pas lipschitzienne.

En effet, g est lipschitzienne sur I s'il existe une constante réelle $L > 0$, telle que $\forall (x, y) \in I^2$, $|g(x) - g(y)| \leq L|x - y|$, c'est à dire que le rapport $\left|\dfrac{g(x) - g(y)}{x - y}\right|$, pour $x \neq y$, est borné.
Posons $y = -2$. Ce rapport vaut

$$\left|\frac{g(x) - g(-2)}{x + 2}\right| = \frac{\sqrt{x+2}}{x+2} = \frac{1}{\sqrt{x+2}} \quad \xrightarrow[x \to -2]{} \quad +\infty$$

donc non bornable sur tout intervalle contenant -2 ; ainsi g ne peut être lipschitzienne sur $[-2, +\infty[$.
En fait, on montre que g est lipschitzienne sur tout intervalle $[a, +\infty[$ avec $a > -2$. Sur cet intervalle, $g'(x) = \dfrac{1}{2\sqrt{x+2}} \leq \dfrac{1}{2\sqrt{a+2}}$. Ainsi, grâce à la proposition 1, g est lipschitzienne de constante $L \leq \dfrac{1}{2\sqrt{a+2}}$.
En outre, g est contractante si $L < 1$, donc si $\dfrac{1}{2\sqrt{a+2}} < 1$, c'est à dire pour $a > \dfrac{-7}{4}$.
En conclusion, g est contractante sur $]\dfrac{-7}{4}, +\infty[$.

Exercice 2

(a) Points fixes de $g(x) = \dfrac{1}{\sqrt{x}}$. Rappelons qu'un point fixe de g est un point d'abscisse $\bar{x}$ vérifiant $g(\bar{x}) = \bar{x}$. Par abus de langage, et dans tous les exercices qui suivent, on dira que x est le point fixe de g (au lieu de l'abscisse du point fixe de g).
Ici g est définie sur R^{+*} et on a

$$g(x) = x \quad \Rightarrow \quad x\sqrt{x} = 1 \quad \Rightarrow x = 1.$$

$x = 1$ est clairement la seule solution sur $\mathbb{R}^{+*}$ de cette équation et est par conséquent le seul point fixe de g.
Démontrons le autrement :

$$g(x) = x \quad \Rightarrow \quad \frac{1}{\sqrt{x}} - x = 0 \quad \Rightarrow \quad x\sqrt{x} - 1 = 0 \text{ et } x > 0.$$

Posons $F(x) = x\sqrt{x} - 1$; F est continue sur $\mathbb{R}^+$ et dérivable sur $\mathbb{R}_*^+$ et $F'(x) = \frac{3}{2}\sqrt{x} > 0$, donc F est strictement croissante sur $\mathbb{R}_*^+$. D'autre part, $F(0.1) < 0$ et $F(2) \geq 0$, donc $F(0.1)F(2) < 0$. Ainsi, d'après le théorème

de la valeur intermédiaire, il existe un et un seul réel $c \in [0.1, 2]$ tel que $F(c) = 0$; celui-ci est donc le seul point fixe de g sur $[0.1, 2]$. Le lecteur pourra aisément démontrer que cela reste vrai sur tout $\mathbb{R}^{+*}$.

(b) Points fixes de $g(x) = e^{-x}$.

Posons $F(x) = e^{-x} - x$. F est continue et dérivable sur $\mathbb{R}$, et $F'(x) = -e^{-x} - 1 < 0$, donc F est strictement décroissante. D'autre part, $F(0) = 1$ et $F(1) = \frac{1}{e} - 1 < 0$. D'après le théorème de la valeur intermédiaire, il existe un et un seul réel $c \in [0, 1]$ tel que $F(c) = 0$. Ce réel est donc l'unique point fixe de g sur $[0, 1]$. De même, on peut aisément démontrer que cela reste vrai sur tout $\mathbb{R}$.

(c) Points fixes de $g(x) = x + (x-2)^3$.

$$g(x) = x \quad \Rightarrow \quad (x-2)^3 = 0 \quad \Rightarrow \quad x = 2.$$

Donc 2 est l'unique point fixe de g sur $\mathbb{R}$; ce point fixe est dit triple à cause de la puissance 3 du terme $(x-2)$.

(d) Points fixes de $g(x) = (x-2)^2 + x - \dfrac{e^x}{\pi}$.

$$g(x) = x \quad \Rightarrow \quad (x-2)^2 = \frac{e^x}{\pi}.$$

Appliquons le théorème de la valeur intermédiaire à

$$F(x) = (x-2)^2 - \frac{e^x}{\pi}$$

. F est continue et dérivable sur R.

$$F'(x) = 2(x-2) - \frac{e^x}{\pi}.$$

Montrons que $\forall x \in \mathbb{R}, F'(x) < 0$.

Pour cela, on étudie le signe de F'', on a :

$$F''(x) = 2 - \frac{e^x}{\pi} > 0 \quad \Rightarrow \quad e^x < 2\pi \quad \Rightarrow \quad x < \ln(2\pi),$$

En conséquence, F' est strictement croissante sur $]-\infty, \ln(2\pi)[$, strictement croissante sur $]\ln(2\pi, \infty)[$ et $F'(\ln(2\pi)) < 0$. Ainsi, $\forall x \in \mathbb{R}, F'(x) < 0$, donc F est strictement décroissante.

D'après le théorème de la valeur intermédiaire, il existe un et un seul réel $c \in [A, \ln(2\pi)[$ tel que $F(c) = 0$. Ce dernier est l'unique point fixe de g.

Exercice 3

$x = \omega - \varepsilon \sin(x)$, $\omega \in \mathbb{R}, |\varepsilon| \leq 1$.

Montrons qu'il existe un unique réel $\bar{x} \in [\omega - \pi, \omega + \pi]$ solution de l'équation

$$F(x) = x - \omega + \varepsilon \sin x = 0.$$

On a

$$F'(x) = 1 - \varepsilon \cos x \geq 0, \text{ car} |\varepsilon| \leq 1,$$

ainsi F est monotone. Or,

$$\begin{aligned} F(\omega - \pi) &= -\pi + \varepsilon \sin(\omega - \pi) < 0 \\ F(\omega + \pi) &= \pi + \varepsilon \sin(\omega - \pi) > 0 \end{aligned}$$

donc, $F(\omega - \pi)F(\omega + \pi) < 0$, et d'après le théorème de la valeur intermédiaire, il existe un unique réel $\bar{x} \in [\omega - \pi, \omega + \pi]$ tel que $F(\bar{x} = 0)$.

exercice 4

Soit l'équation $F(x) = x^2 - 100x + 1 = 0$.

a) Posons $g_1(x) = \dfrac{x^2 + 1}{100}$

Étudions donc la méthode itérative $x_{n+1} = \dfrac{x_n^2 + 1}{100}$.

Remarquons tout d'abord que si cette méthode converge, elle converge bien vers une des racines de $F(x) = 0$, si $\bar{x}$ est la limite de la suite (x_n), alors $\bar{x} = \dfrac{\bar{x}^2 + 1}{100}$ donc $\bar{x}^2 - 100\bar{x} + 1 = 0$, c'est à dire que $F(\bar{x}) = 0$.

Localisons la racine $\bar{x}$ de cette équation. On a $F(0) = 1$ et $F(1) = -98$ donc $F(0)F(1) < 0$, et comme F est dérivable sur $\mathbb{R}$ et $F'(x) = 2x - 100 < 0$ sur $[0, 1]$ alors, grâce au théorème de la valeur intermédiaire, il existe un unique réel $\bar{x} \in [0, 1]$ solution de l'équation $F(x) = 0$.

On a $g_1(0) = \frac{1}{100} > 0$ et $g_1(1) = \frac{1}{50} < 1$. Comme g_1 est monotone sur $\mathbb{R}^+$ (puisque $g_1'(x) = x/50 > 0$ sur R^+), on a donc $g_1([0, 1]) \subset [0, 1]$.

Démontrons que $g_1(x) = \dfrac{x^2 + 1}{100}$ est contractante sur $[0, 1]$:

$$\begin{aligned} |g_1(x) - g_1(y)| &= \left| \frac{x^2 + 1}{100} - \frac{y^2 + 1}{100} \right| \\ &= \frac{1}{100} \left| x^2 - y^2 \right| \\ &= \frac{|x + y|}{100} |x - y| \leq \frac{1}{50} |x - y|. \end{aligned}$$

On aurait pu aussi la proposition 1, puisque $g_1'(x) = \dfrac{x}{50}$ et $max_{[0,1]}|g_1'(x)| = 1/50$ donc g_1 contractante de rapport $1/50$.

Ainsi, $\forall x_0 \in [0,1]$, $x_{n+1} = g_1(x_1) = \frac{x_n^2+1}{100}$ converge vers $\bar{x}$, unique solution de $x^2 - 100x = 0 + 1$ dans $[0,1]$.

Calculons cette racine partant de $x_0 = 0$, on a

$$\begin{aligned} x_0 &= 0 \\ x_1 &= g_1(x_0) = g_1(0) = \frac{1}{100} = 0.010001 \\ x_2 &= g_1(x_1) = g_1(0.010001) = 0.01000100020001 \\ x_3 &= g_1(x_2) = g_1(0.01000100020001) = ... \\ x_4 &= g_1(x_3) = ... \end{aligned}$$

Si on cherche $\bar{x}$ à ε près, on arrêtera les calculs à l'itération p telle que $|x_{p+1} - x_p| \leq \varepsilon$. Ainsi la solution $\bar{x}$ ici vaut 0.010001 à 10^{-6} près.

b) L'autre solution est obtenue grâce à la méthode itérative $x_{n+1} = g_2(x_n) = 100 - \frac{1}{x_n}$. Cette question est laissée en exercice.

exercice 5

Soit l'équation $F(x) = 2x^3 - x - 2 = 0$. Il est clair que F est continue et dérivable sur $\mathbb{R}$.

On a $F(1) = -1$, $F(2) = 12$, donc $F(1)F(2) < 0$. D'autre part, $F'(x) = 6x^2 \geq 0$ sur $[1,2]$. Donc, d'après le théorème de la valeur intermédiaire, il existe une seule solution $\bar{x} \in [1,2]$ telle que $F(\bar{x}) = 0$.

(a) Etudions la convergence de la suite $x_{n+1} = g_1(x_n) = 2x_n^3 - 2$. Tout d'abord, cette suite, si elle converge, conduit bien à une racine de $F(x) = 0$ car si $\bar{x}$ est la limite de la suite (x_n), alors

$$\bar{x} = 2\bar{x}^3 - 2 \quad \text{donc} \quad F(\bar{x}) = 2\bar{x}^3 - \bar{x} - 2 = 0.$$

Par ailleurs, $g_1'(x) = 6x^2 \geq 6$ sur $[1,2]$. Par conséquent, grâce au théorème des accroissements finis, il existe ξ_n compris entre x_n et x_{n+1} tel que

$$|g_1(x_{n+1}) - g_1(x_n)| = g_1'(\xi_n)|x_{n+1} - x_n|.$$

Donc

$$\begin{aligned} |g_1(x_{n+1}) - g_1(x_n)| &\geq 6|x_{n+1} - x_n| \\ &\geq 6^2|x_n - x_{n-1}| \\ &\vdots \\ &\geq 6^n|x_1 - x_0|. \end{aligned}$$

Ainsi, cette suite diverge et la méthode est à rejeter.

(b) Étudions la convergence de $x_{n+1} = g_2(x_n) = \dfrac{2}{2x_n^2 - 1}$. Cette méthode, si elle converge conduit vers la racine $\bar{x}$ de $F(x)$ dans $[1, 2]$, car si $\bar{x}$ est la limite de la suite (x_n), alors

$$\bar{x} = \frac{2}{2\bar{x}^2 - 1} \quad \text{donc} \quad F(\bar{x}) = 2\bar{x}^3 - 2\bar{x} - 1 = 0.$$

$$\begin{aligned} g_2'(x) &= \frac{-8x}{(2x^2-1)^2} \\ g_2''(x) &= \frac{8(6x^2+1)}{(2x^2-1)^3} \end{aligned}$$

	1		2
g_2''		+	
g_2'	-8	$\nearrow$	$\frac{16}{49}$

En conséquence, on ne peut conclure sur la monotonie de g_2. Cependant on a

$$g_2'(\bar{x}) = \frac{-8\bar{x}}{(2\bar{x}^2 - 1)^2} = -2\bar{x}\left(\frac{2}{2\bar{x} - 1}\right)^2,$$

or $\bar{x}$ le point fixe de F vérifie

$$\frac{2}{2\bar{x} - 1} = \bar{x}.$$

Donc $g_2'(\bar{x}) = -2\bar{x}^3$, et comme g_2' est continue, il existe un voisinage V de $\bar{x}$ tel que $V \subset [1, 2]$, et $\forall x \in V, |g_2'(x)| > 2$. Donc cette méthode ne peut pas converger d'après la proposition 3. En effet, grâce au théorème des accroissements finis, on a $|x_{n+1} - x_n| \geq 2^n|x_1 - x_0|$.

(c) Étudions la convergence de $x_{n+1} = g_3(x_n) = \sqrt[3]{1 + \frac{x_n}{2}}$. Si elle converge, cette méthode conduit à la racine de $F(x) = 0$ dans $[1, 2]$ car si $\bar{x}$ est la limite de la suite (x_n), alors

$$\bar{x} = \sqrt[3]{1 + \frac{\bar{x}}{2}} \quad \text{donc} \quad \bar{x}^3 = 1 + \frac{\bar{x}}{2} \quad \text{et} \quad F(\bar{x}) = 2\bar{x}^3 - 2\bar{x} - 1 = 0.$$

On a

$$0 < g_3'(x) \quad = \quad \frac{1}{6\sqrt[3]{(1 + \frac{x}{2})^2}} < 1,$$

donc g_3 est strictement contractante d'après la proposition 1. D'autre part, $g_3(1) = \sqrt[3]{\frac{3}{2}} > 1$, $g_3(2) = \sqrt[3]{2} < 2$, or g_3 est monotone, donc $g_3([1, 2]) \subset [1, 2]$. Donc d'après le théorème du point fixe, la suite

$$\begin{cases} x_0 \in [1, 2] \\ x_{n+1} = g_3(x_n) \end{cases}$$

converge vers l'unique racine $\bar{x} \in [1, 2]$ de l'équation $x = g_3(x)$.

Calcul numérique de cette racine à 10^{-3} près, à partir de $x_0 = 1$:

n	0	1	2	3	4
x_n	1	1.144	1.**16**2	1.**165**	0.**165**

Donc $\bar{x} = 1.165$ est solution de l'équation à 10^{-3} près.

exercice 6

Soit l'équation $x = \ln(1 + x) + 0.2$ dans R^+.

Considérons la méthode itérative définie par :

$$x_{n+1} = g(x_n) = \ln(1 + x_n) + 0.2$$

Montrons d'abord l'existence d'une solution pour cette équation.

Soit $F(x) = \ln(1 + x) + 0,2 - x = 0$, on a $F'(x) = \frac{-x}{1+x} < 0$ sur $\mathbb{R}^+$, donc l'équation $F(x) = 0$ admet au plus une racine. D'autre part on a $F(0) = 0.2$ et $F(1) = \ln 2 - 0.8 < 0$, donc $F(0)F(1) < 0$; ainsi, d'après le théorème de la valeur intermédiaire, il existe une unique racine $\bar{x} \in [0, 1]$ solution de l'équation $F(x) = 0$.

Appliquons la Méthode du point fixe pour $g(x) = \ln(1 + x) + 0,2$.

g est contractante sur $I = [a, b] \subset]0, 1]$ car

$$\forall x \in I, 0 < g'(x) = \frac{1}{1+x} < 1.$$

Donc, si $g([a, b]) \subset [a, b]$, d'après le théorème du point fixe, il existe une unique racine $\bar{x} \in [a, b]$ solution de l'équation $F(x) = 0$.

Par exemple, on vérifie que $g([0.7, 0.8]) \subset [0.7, 0.8]$. En effet,$g(0.7) = 0.73... > 0.7$ et $g(0.8) = 0.78... < 0.8$.

Calcul numérique de cette racine à 10^{-2} et 10^{-3} près :

n	0	1	2	3	4	5	6	7	8	9
x_n	0.7	0.730	0.748	0.758	0.**76**4	0.**76**7	0.769	0.770	0.**771**	0.**771**

Ainsi la racine cherchée est $\bar{x} = 0.76$ à 10^{-2} près, et $\bar{x} = 0.771$ à 10^{-3} près.

exercice 7

Soit l'équation $x = g(x)$ où $g(x) = -\ln x$.

1. 1) Posons $F(x) = x - g(x) = x + \ln(x)$, $\forall x \in \mathbb{R}^{*+}$. Appliquons le théorème de la valeur intermédiaire à F sur $[a, 1]$ où $0 < a \leq 1$.
   F est continue sur $[a, 1], \forall a \in]0, 1]$. $\forall x \in [a, 1]$, $F'(x) = 1 + \frac{1}{x}$ et $F'(x) > 0$, donc F est strictement monotone sur $[a, 1]$.
   D'autre part on a $F(1) = 1 > 0$, et comme $\lim\limits_{x \longrightarrow 0^+} \ln(x) = -\infty$, alors il existe $a \in]0, 1]$ tel que $\ln(a) < -a < 0$; par conséquent, $F(1)F(a) < 0$, et d'après le théorème de la valeur intermédiaire, il existe un unique $\bar{x} \in [a, 1]$ (d'ou $\bar{x} \in]0, 1]$) tel que $F(\bar{x}) = 0$, et donc tel que $\bar{x} = g(\bar{x}) = \ln \bar{x}$.
   2) Si $x_{n+1} = g(x_n)$ converge, elle conduit bien à la racine de l'équation car cette dernière vérifie $\bar{x} = g(\bar{x})$ donc

   $$\bar{x} = -\ln(\bar{x}) \implies \bar{x} + \ln(\bar{x}) = 0$$

   Mais, $\forall x \in [a, 1[, g'(x) = -\frac{1}{x}$, et $|g'(x)| > 1$ donc la méthode $x_{n+1} = -\ln(x_n)$ diverge pour tout $x_0 \in [a, 1[$.
   3) g^{-1} existe car g est continue et strictement croissante donc bijective. Montrons que $x_{n+1} = g^{-1}(x_n)$ est convergente.
   Attention à la notation utilisée : g^{-1} désigne la réciproque de g et non $\frac{1}{g(x)}$.
   g^{-1} est dérivable et on a $(g^{-1})' = \frac{1}{g' \circ g^{-1}}$.
   En effet, d'une manière générale $(u \circ v)' = (u' \circ v)v'$, par conséquent $(g \circ g^{-1})'(x) = (g' \circ g^{-1})(x)(g^{-1})'(x)$.

Or, $g \circ g^{-1} = Id$, donc $(g \circ g^{-1})' = 1$. On a bien alors $(g^{-1})'(x) = \frac{1}{g' \circ g^{-1}}$.
Or, on a montré que $\forall x \in [a, 1[$, $|g'(x)| > 1$. Mais $g^{-1} = e^{-x}$, donc $y = g^{-1}(x) \in]e^{-1}, e^{-a}]$, car e^{-x} est décroissante. Puisque $a > 0$, alors $e^{-a} < e^0 = 1$, donc $0 < y < 1$ et $|g'(y)| = \left|-\frac{1}{y}\right| > 1$
Par conséquent $|(g^{-1})'(x)| < 1$.
Ainsi la méthode $x_{n+1} = g^{-1}(x_n)$ converge au voisinage de $\bar{x}$, d'après la proposition 2.

4) Posons $e_n = x_n - \bar{x}$ et $h(x) = e^{-x}$. La méthode $x_{n+1} = e^{-x_n}$ converge (d'ailleurs, on a $h'(x) = -e^{-x}$ et $|h'(x)| = e^{-x} < 1$, $\forall x \in \mathbb{R}_+^*$). D'autre part, grâce au théorème des accroissements finis on sait qu'au voisinage de $\bar{x}$, il existe ξ_n compris entre x_n et $\bar{x}$ tel que :

$$e_{n+1} = x_{n+1} - \bar{x} = h(x_n) - h(\bar{x}) = h'(\xi_n)(x_n - \bar{x}).$$

Ainsi, $e_{n+1} = h'(\xi_n)e_n$. Or, $h'(x) < 0$ $\forall x \in \mathbb{R}^{*+}$. Donc e_{n+1} et e_n sont de signes opposés, par conséquent deux itérations successives donnent un encadrement de $\bar{x}$.

5) Un test d'arrêt des itérations est : $|e_{n+1} - e_n| = |x_{n+1} - x_n| \leq \varepsilon = 10^{-4}$.
Prenons $I = [a, 1]$ avec $a = 0.1$. On a bien $|h'(x)| < 1$ sur I et $h(I) \subset I$ car :

$$\begin{aligned} h(0.1) = e^{-0,1} &> 0.1 \\ h(1) = \frac{1}{e} &< 1 \end{aligned}$$

et $h(x) = e^{-x}$ est monotone sur $[0.1, 1]$
Calcul numérique de la racine à 10^{-3} près. Soit donc la méthode $x_{n+1} = e^{-x_n}$ et $x_0 = 1$

n	0	1	2	3	4	5	6	7	8
x_n	1	0.367	0.692	0.500	0.606	0.**5**45	0.**5**79	0.**5**60	0.**5**71

n	9	10	11	12	13	14	15
x_n	0.**56**4	0.**56**8	0.**56**6	0.**56**7	0.**56**6	0.**567**	0.**567**

Ainsi la racine cherchée est $\bar{x} = 0.567$ à 10^{-3} près.

2. Méthode de Newton.
Soit $F(x) = x - e^{-x} = 0$, F est clairement indéfiniment dérivable. La

méthode de Newton s'écrit,

$$x_{n+1} = x_n - \frac{F(x_n)}{F'(x_n)} = x_n - \frac{x_n - e^{-x_n}}{1 + e^{-x_n}}.$$

D'autre part on a $F(0) = -1 < 0$ et $F(1) = 1 - \frac{1}{e} > 0$, donc $F(1)F(0) < 0$ et la racine est située dans $[0, 1]$, elle est unique puisque F est strictemnet monotone, car $F'(x) = 1 + e^{-x} > 0$ pour tout $x \in \mathbb{R}$. On a aussi $F''(x) = -e^{-x} < 0$ pour tout $x \in \mathbb{R}$. Ainsi d'après le théorème de convergence globale de cette méthode (voir théorème 3), pour tout $x_0 \in [0, 1]$ tel que $F(x_0)F''(x_0) > 0$ l'itération de Newton converge. Prenons alors, par exemple, $x_0 = 0$, alors $F(0)F''(0) = 1/e > 0$, donc la méthode

$$x_{n+1} = x_n - \frac{x_n - e^{-x_n}}{1 + e^{-x_n}} x_0$$

convegera vers l'unique racine $\bar{x}$ de l'équation.

Calcul numérique

==========

2.9 Mise en œuvre en Java

On présente, dans la suite, la mise en œuvre de la méthode de Lagrange (cf. paragraphe 2.4) ainsi que la méthode de Steffenson (cf. paragraphe 2.5). Cette dernière est écrite sous une forme générique, permettant la composition d'une méthode itérative quelconque avec le procédé Δ^2 d'Aitken. Nous illustrons ces programmes, par des exemples de calcul qui permettent de comparer ces deux méthodes.

2.9.1 Une classe abstraite de description de processus itératifs

Nous allons utiliser dans la suite deux processus itératifs, à savoir les méthodes de Lagrange et de Steffensen. Ces deux processus sont basés sur des calculs similaires : calcul d'une formule de récurrence et processus de répétition jusqu'à une éventuelle convergence, liée à un critère donné. Beaucoup d'autres schémas numériques sont susceptibles de suivre de tels processus. Pour cette raison, nous avons commencé par définir une classe *abstraite* de description d'un processus itératif, appelée `IterGene`. Cette classe est *abstraite* car elle contient une méthode *abstraite*, non implémentée, correspondant à la formule de récurrence. Celle-ci va dépendre de la formule spécifique d'une classe particulière qui en dérivera. Par contre, un schéma de construction de suite récurrente, susceptible de converger, peut être implémenté dans cette classe *abstraite* en s'appuyant sur sa formule de récurrence, elle-même *abstraite*.

Il nous faut gérer la convergence d'un processus itératif grâce à une vérification de pseudo-stabilité sur deux itérés successifs. Dans le cas où il n'y a pas convergence, un nombre maximum d'itérations est prédéfini et permet de stopper le processus. Dans ce cas, il reste alors à renseigner les programmes appelant de la non-convergence. Pour cela, nous utilisons un mécanisme d'exception qui est spécifiquement adapté à un tel traitement ; il sera alors capable de faire remonter, en cascade, l'information de non-convergence, aux différents programmes appelants. On définit alors une classe exception élémentaire capable d'être évaluée sous la forme d'une chaîne de caractères et implémentant, pour cela, la méthode `toString`.

```
class NonConvergenceException extends Exception {
    public String toString() { return("Defaut de convergence"); }
}
```

Nous pouvons alors définir la classe `IterGene` qui est composée :

– des données `xn, epsilon, maxIter` qui correspondent respectivement, au terme courant du processus itératif, à la précisison de conver-

gence et au nombre maximun d'itérations autorisées. On implémente des méthodes permettant de fixer leurs valeurs ;
- d'une méthode *abstraite* d'itération `iter`, sans paramètre, qui actualisera le terme courant du processus itératif. Nous décrivons aussi, une autre version de cette méthode, `iter`, avec un paramètre qui correspond à une mise à jour du terme courant, avant l'application du processus itératif ;
- d'un calcul de suite recurrente `calculSuite` s'appuyant sur la méthode d'itération et écrite sous deux versions : sans paramètre ou avec un paramètre booléen qui permet d'afficher une trace des termes successifs de la suite calculée.

```
import java.lang.*;
import java.io.*;

abstract class IterGene {
    double xn;         // terme courant de la suite
    double epsilon;    // precision de convergence
    int maxIter;       // nombre maxi d'iterations

    // fixe la valeur du terme initial de la suite :
    public void set_xn(double x0) { xn = x0; }

    // fixe la valeur de la precision :
    public void set_epsilon(double precision) { epsilon = precision; }

    // fixe le nombre maxi d'iterations :
    public void set_maxIter(int m) { maxIter = m; }

    // methode abstraite du calcul d'une iteration
    // a partir de xn courant :
    abstract public double iter() ;

    // methode de calcul d'une iteration
    // a partir d'une valeur x donne :
    public double iter(double x) {
        set_xn(x);
        return iter();
    }

    // calcul d'une suite recurrente
    //a partir de iter et jusqu'a convergence :
    public double calculSuite(boolean trace)
      throws NonConvergenceException {
```

```
        double xOld;
        int nbIter=0;
        if (trace) {
            System.out.println("valeurs successives calculees :");
            System.out.println(xn);
        }
        do {
            xOld=xn;
            iter();
            nbIter++;
            if (trace)
                { System.out.println("iter. "+nbIter+" : "+xn);}
        } while (Math.abs(xn-xOld) > epsilon && nbIter <= maxIter);
        if (nbIter > maxIter) throw new NonConvergenceException();
        return xn;
    }

    public double calculSuite ()
      throws NonConvergenceException {
        return calculSuite(false);
    }
}
```

2.9.2 La méthode de Lagrange

Nous décrivons, maintenant, la classe mettant en œuvre la méthode de Lagrange, vue dans le paragraphe 2.4 et qui, à partir d'une application F continue, strictement monotone sur $[a, b]$ et telle que $F(a)F(b) < 0$, construit le processus itératif suivant :

$$\begin{cases} x_0 = a \text{ ou } b, \text{ de sorte que } F(x_0)F''(x_0) > 0 \\ x_{n+1} = x_n - F(x_n)\frac{x_n - a}{F(x_n) - F(x_0)} \end{cases}$$

La classe `IterLagrange` est telle que son constructeur possède les paramètres suivants : la valeur initiale x_0 et une fonction quelconque réelle d'une variable réelle (les réels étant représentés par le type `double`).

Le passage d'une fonction, comme paramètre, se fait grâce à une interface qui sera implémentée effectivement, par la fonction particulière utilisée (comme décrit en 1.5.7). Cette interface propose une méthode d'évaluation `calcul` de la fonction pour un paramètre donné. Elle s'écrit :

```
interface FoncD2D { double calcul(double x); }
```

La classe `IterLagrange`, qui dérive de la classe `IterGene`, s'écrit alors :

```
import IterGene;
import FoncD2D;

class IterLagrange extends IterGene {
    FoncD2D f;         // fonction dont on recherche le zero
    double x0; // bornes de l'intervalle de calcul

    // constructeurs
    IterLagrange(FoncD2D fonc, double xini) {
        f=fonc; x0=xini;
        epsilon=0.001;
        maxIter=100;
    }

    // calcul d'une iteration :
    public double iter() {
        xn = xn - f.calcul(xn)*(xn-x0)/
                 (f.calcul(xn)-f.calcul(x0)) ;
        return xn;
    }
}
```

Nous présentons ci-dessous, un programme de test de la méthode de Lagrange, en commençant par définir la fonction F, dans une classe implémentant l'interface `FoncD2D` correspondant à :

$$F(x) = X^3 - 8 \text{ sur l'intervalle } [1, 5]$$

La valeur initiale de la suite récurrente est la borne 5, qui satisfait à la condition exprimée précédemment.

Le programme correspondant s'écrit :

```
import java.io.*;
import FoncD2D;
import IterLagrange;

class MaFonction implements FoncD2D {
    public double calcul(double x) {
        return x*x*x-8;
    }
}
```

```
class TestLagrange {
    public static void main(String args[]) {
        MaFonction f = new MaFonction();
        IterLagrange rechercheZero = new IterLagrange(f,5);
        rechercheZero.set_epsilon(1E-6);
        try {
            double resultat = rechercheZero.calculSuite(true);
            System.out.println("le zero trouve vaut : "+resultat);
        }
        catch(NonConvergenceException e) {
            System.out.println(e);
        }
    }
}
```

Nous donnons ci-après une trace de l'exécution montrant la convergence de la recherche en 45 itérations :

```
java TestLagrange

valeurs successives calculees :
5.0
iter. 1 : 1.225806451612903       iter. 2 : 2.8774769746022884
iter. 3 : 1.575783340029592       iter. 4 : 2.3837064484468686
iter. 5 : 1.7721357917725993      iter. 6 : 2.183912111336465
iter. 7 : 1.8801284843229529      iter. 8 : 2.091190544058032
iter. 9 : 1.9378012139273417      iter. 10 : 2.0458883891378012
iter. 11 : 1.9679808942878763     iter. 12 : 2.0232527320893645
iter. 13 : 1.9835875282989195     iter. 14 : 2.011822859470857
iter. 15 : 1.9916061843487987     iter. 16 : 2.006021559594976
iter. 17 : 1.995712170773491      iter. 18 : 2.0030695016733913
iter. 19 : 1.9978109552571226     iter. 20 : 2.00156536536722
iter. 21 : 1.998882781401325      iter. 22 : 2.000798472022616
iter. 23 : 1.9994298969642061     iter. 24 : 2.00040733587803
iter. 25 : 1.9997091067417003     iter. 26 : 2.0002078119872224
iter. 27 : 1.9998515787290103     iter. 28 : 2.0001060232863277
iter. 29 : 1.9999242732103275     iter. 30 : 2.00005409267069
iter. 31 : 1.9999613634529168     iter. 32 : 2.0000275980820095
iter. 33 : 1.999980287364067      iter. 34 : 2.0000140805969857
iter. 35 : 1.9999899425035552     iter. 36 : 2.0000071839631905
iter. 37 : 1.9999948686166795     iter. 38 : 2.000003665283473
iter. 39 : 1.9999973819453114     iter. 40 : 2.000001870041581
iter. 41 : 1.999998664257298      iter. 42 : 2.0000009541025854
```

```
iter. 43 : 1.9999993184984877     iter. 44 : 2.000000486786965
iter. 45 : 1.999999652295112
le zero trouve vaut : 1.999999652295112
```

2.9.3 La méthode de Steffensen

Nous effectuons, maintenant, une implémentation de la méthode d'accélération de Steffesen qui compose le procédé Δ^2 d'Aitken avec une méthode itérative, notée g. Nous utilisons la formulation stable décrite dans le paragraphe 2.5 :

$$u_0 = x_n,\ \ u_1 = g(u_0),\ \ u_2 = g(u_1), x_{n+1} = u_1 + \frac{1}{\frac{1}{u_2-u_1} - \frac{1}{u_1-u_0}} \quad n = 0, 1, 2 \cdots$$

La classe suivante, `IterSteffensen`, permet cette composition avec une méthode itérative quelconque, qui dérive de la classe générale `IterGene`. Par ailleurs, elle dérive elle-même de la classe `IterGene`, en tant que processus itératif. Elle s'écrit alors :

```
import IterGene;

class IterSteffensen extends IterGene {
    IterGene g;  // methode iterative a composer avec d2 d'aitken

    // constructeur :
    IterSteffensen(IterGene fonc, double xini) {
        g=fonc; xn=xini; epsilon=0.001; maxIter=100;
    }

    // calcul d'une iteration :
    public double iter() {
        double u1 = g.iter(xn);
        double u2 = g.iter(u1);
        xn = u1 + 1./( (1./(u2-u1)) -(1./(u1-xn)) );
        return xn;
    }
}
```

Nous donnons ci-après un programme permettant de tester cette méthode et utilisant la méthode de Lagrange, comme processus itératif à coupler avec le procédé Δ^2 d'Aitken. La fonction F, dont on cherche le zéro, est la même que dans le paragraphe précédent. On part de la même valeur initiale x_0 et on utilise la même précision de calcul.

```
import java.io.*;
import FoncD2D;
import IterLagrange;
import IterSteffensen;

class MaFonction implements FoncD2D {
    public double calcul(double x) {
        return x*x*x-8;
    }
}

class TestSteffensen {
    public static void main(String args[]) {
        MaFonction f = new MaFonction();
        IterLagrange methodeLagrange = new IterLagrange(f,1,5,5);
        IterSteffensen methodeSteffensen =
            new IterSteffensen(methodeLagrange, 5);
        methodeSteffensen.set_epsilon(1E-6);
        try {
            double resultat = methodeSteffensen.calculSuite(true);
            System.out.println("le zero trouve vaut : "+resultat);
        }
        catch(NonConvergenceException e) {
            System.out.println(e);
        }
    }
}
```

Nous donnons ci-après, une trace de l'exécution montrant de manière spectaculaire l'accélération de la convergence, par rapport à la méthode de Lagrange, pour les mêmes conditions. En effet, on passe de 45 à 5 itérations pour obtenir un résultat final qui est plus précis.

```
java TestSteffensen

valeurs successives calculees :
5.0
iter. 1 : 2.374697052247919      iter. 2 : 2.0174585514794128
iter. 3 : 2.000046185893817      iter. 4 : 2.0000000003264917
iter. 5 : 2.0
le zero trouve vaut : 2.0
```

Chapitre 3

Résolution des systèmes linéaires $AX = B$

On note $\mathcal{M}_n(\mathbb{R})$, l'espace vectoriel des matrices carrées d'ordre n et à coefficients réels. Soit $A \in \mathcal{M}_n(\mathbb{R})$ et $B \in \mathbb{R}^n$, on cherche le vecteur $X \in \mathbb{R}^n$, solution du système linéaire $AX = B$. Ce système admet une solution unique lorsque le déterminant de A est non nul, ce que nous supposerons dans la suite.
Remarquons que la résolution de ce système à l'aide des formules de Cramer est impraticable lorsque n est 'grand', car ces formules nécessitent approximativement $n!n^2$ opérations arithmétiques élémentaires ; par exemple si n=15, cela représente de l'ordre de 4 mois de calcul pour un ordinateur moyen effectuant 10^6 opérations à la seconde. Ce temps de calcul évolue de manière exponentielle avec n.
Nous rappelons dans la suite les principales méthodes, beaucoup plus rapides, car 'polynomiales en temps' :

. Quelques méthodes directes : Gauss et sa forme factorisée LU

. Quelques méthodes itératives : Jacobi et Gauss-Seidel.

Les méthodes directes calculent une solution théorique exacte, en un nombre fini d'opérations, mais avec une accumulation d'erreurs d'arrondis qui d'autant plus grande que la dimension l'est.

Par conséquent, pour des matrices de dimension importante, il est préférable d'utiliser des méthodes itératives basées sur la construction d'une suite convergente vers la solution du système. Dans ce cas, le contrôle de la convergence de la suite par une condition d'arrêt renvoie une solution plus précise.

3.1 Méthode d'élimination de Gauss

La méthode de Gauss engendre un algorithme fini exact dont l'idée est de transformer le système initial en un système triangulaire (inférieur ou supérieur).

3.1.1 Résolution d'un système triangulaire

On donne ici l'algorithme de résolution d'un système triangulaire inférieur. Il se fait en "remontant" les équations de la dernière ligne à la première.
Soit donc le système $AX = B$ où $A = (a_{ij})_{i,j=1,n}$ est triangulaire inférieure et inversible. Dans ce cas, $\forall i = 1 \cdots n$, $a_{ii} \neq 0$ puisque $Det(A) = \prod_{i=1}^{n} a_{ii} \neq 0$.
L'algorithme de résolution est :

$$\begin{cases} x_n = \frac{b_n}{a_{nn}} \\ \text{et} \\ x_i = \frac{1}{a_{ii}}(b_i - \sum_{j=i+1}^{n} a_{ij}x_j) \text{ pour } i = n-1, ..., 1 \end{cases}$$

3.1.2 Méthode de Gauss

Pour la clarté des explications, on applique la méthode de Gauss dans le cas où n=3. On suppose que $a_{11} \neq 0$.

$$(1) \ : \begin{cases} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 = b_1 & : \ L_1^{(1)} \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 = b_2 & : \ L_2^{(1)} \\ a_{31}x_1 + a_{32}x_2 + a_{33}x_3 = b_3 & : \ L_3^{(1)} \end{cases}$$

On élimine le terme $a_{21}x_1$ dans la ligne $L_2^{(1)}$ grâce à la combinaison $L_2^{(1)} - \frac{a_{21}}{a_{11}}L_1^{(1)}$ et le terme $a_{31}x_1$ dans $L_3^{(1)}$ grâce à la combinaison $L_3^{(1)} - \frac{a_{31}}{a_{11}}L_1^{(1)}$ ce qui donne le système $A^{(2)}x = B^{(2)}$, équivalent au système initial :

$$(2) \ : \begin{cases} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 = b_1 & : \ L_1^{(1)} \\ a_{22}^{(2)}x_2 + a_{23}^{(2)}x_3 = b_2^{(2)} & : \ L_2^{(2)} \\ a_{32}^{(2)}x_2 + a_{33}^{(2)}x_3 = b_3^{(2)} & : \ L_3^{(2)} \end{cases}$$

où $a_{ij}^{(2)} = a_{ij} - \frac{a_{i1}}{a_{11}}a_{1j} \quad i = 2, 3 \quad \text{et} \quad j = 2, 3$ et $\quad b_j^{(2)} = b_j - \frac{a_{i1}}{a_{11}}b_1$.

Ensuite, en supposant que $a_{22} \neq 0$, on élimine le terme $a_{32}^{(2)}x_2$ dans $L_3^{(2)}$ grâce à la combinaison $L_3^{(2)} - \frac{a_{32}^{(2)}}{a_{22}^{(2)}}L_2^{(2)}$, ce qui donne le système $A^{(3)}x = B^{(3)}$ équivalent au système initial :

$$(3) \ : \ \left\{ \begin{array}{rcl} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 = b_1 & : & L_1^{(1)} \\ a_{22}^{(2)}x_2 + a_{23}^{(2)}x_3 = b_2^{(2)} & : & L_2^{(2)} \\ a_{33}^{(3)}x_3 = b_3^{(3)} & : & L_3^{(3)} \end{array} \right.$$

où $a_{33}^{(3)} = a_{33}^{(2)} - \frac{a_{32}^{(2)}}{a_{22}^{(2)}} a_{23}^{(2)}$ et $b_3^{(3)} = b_3^{(2)} - \frac{a_{32}^{(2)}}{a_{22}^{(2)}} b_2^{(2)}$.

Le système obtenu est donc triangulaire. On le résout grâce à l'algorithme du paragraphe précédent.

3.1.3 Factorisation LU

L'étape de triangularisation de la méthode précédente revient à factoriser la matrice A en un produit de deux matrices triangulaires l'une supérieure l'autre inférieure. En effet, les différentes étapes de la triangularisation peuvent s'écrire sous forme matricielle. Si on pose $A = A^{(1)}$, la matrice $A^{(2)}$ du second système est obtenue en multipliant à gauche $A^{(1)}$ par

$$M^{(1)} = \begin{pmatrix} 1 & 0 & 0 \\ -\frac{a_{21}}{a_{11}} & 1 & 0 \\ -\frac{a_{31}}{a_{11}} & 0 & 1 \end{pmatrix}$$

on a donc

$$A^{(2)} = M^{(1)}A^{(1)} = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ 0 & a_{22}^{(2)} & a_{23}^{(2)} \\ 0 & a_{32}^{(2)} & a_{33}^{(2)} \end{pmatrix}.$$

De même la matrice $A^{(2)}$ du troisième système vérifie $A^{(3)} = M^{(2)}A^{(2)}$ où

$$M^{(2)} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -\frac{a_{32}^{(2)}}{a_{22}^{(2)}} & 1 \end{pmatrix}$$

on a donc

$$A^{(3)} = M^{(2)}M^{(1)}A^{(1)} = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ 0 & a_{22}^{(2)} & a_{23}^{(2)} \\ 0 & 0 & a_{33}^{(3)} \end{pmatrix}.$$

Ainsi $A^{(1)} = (M^{(2)}M^{(1)})^{-1}A^{(3)} = (M^{(1)})^{-1}(M^{(2)})^{-1}A^{(3)}$. Les matrices $M^{(1)}$ et $M^{(2)}$ sont inversibles car leurs déterminants sont tous égaux à 1 et leurs inverses sont faciles à calculer. En effet, on peut vérifier qu'on les obtient en transformant

en leur opposé les termes figurant sous la diagonale, les autres termes restant inchangés. On trouve alors

$$L = (M^{(1)})^{-1}(M^{(2)})^{-1} = \begin{pmatrix} 1 & 0 & 0 \\ \frac{a_{21}}{a_{11}} & 1 & 0 \\ \frac{a_{31}}{a_{11}} & \frac{a_{32}^{(2)}}{a_{22}^{(2)}} & 1 \end{pmatrix}.$$

Si on pose $U = A^{(3)}$, qui est triangulaire supérieure, A peut alors s'écrire comme le produit des deux matrices L et U :

$$A = LU$$

Finalement, on obtient la solution X du système initial $AX = B$ en résolvant successivement les deux systèmes suivants :

$$LY = B$$
$$UX = Y$$

Ceci se généralise à une matrice d'ordre n. Le lecteur pourra le traiter en exercice. Colmplément écrit sur brouillon

Remarque 5 *Lorsque n est "grand", le nombre d'opérations de l'algorithme de résolution du système triangulaire est équivalent à n^2, car il requiert pour chaque x_i : (n-i) multiplications, (n-i) soustractions et une division, soit au total $\sum_{i=1}^{n}(2(n-i)+1)$ opérations. D'autre part, le nombre d'opérations de l'algorithme de triangularisation est de l'ordre de n^3. Donc, la résolution complète nécessite au total un nombre d'opérations équivalent à n^3. Pour $n = 15$, sur la même configuration de machine que celle citée en début du chapitre, il faudra de l'ordre d'une seconde pour la résolution totale du système !*

Faire un paragraphe complet sur la méthode des pivots lignes ... plus remarque sur le pivotage colonnes.

3.1.4 Remarque sur le Pivot

A chaque étape de calcul le pivot $a_{kk}^{(k-1)}$ doit être non nul, sinon une permutation de lignes ou de colonnes s'impose. Sur le plan numérique si le pivot est "trop petit" (par rapport aux autres coefficients de la matrice ou par rapport à la précisison de la machine), le cumul des erreurs numériques peut être important. Ainsi pour des raisons de stabilité numérique, il faut effectuer des permutations de lignes ou de colonnes de façon à aboutir à un pivot "plus grand". On dira que l'on fait un pivotage total lorsqu'on permute des lignes et des colonnes,

sinon le pivotage est dit partiel. Si on n'effectue qu'un pivotage partiel sur les lignes, cela ne revient qu'à changer l'ordre d'énumération des lignes du système qui n'est donc pas modifié. Lorsque l'on effectue un échange de colonnes, cela revient à permuter les éléments du vecteur inconnu X qui sera donc différent de la solution du système initial. D'une manière pratique, dans ce dernier cas, il est donc nécessaire de mémoriser toutes les permutations de colonnes afin de pouvoir reconstruire le vecteur solution en respectant l'ordre de ses éléments correspondant au système initial.

(la notion de petitesse dépend de la précision de l'ordinateur utilisé), l'erreur d'arrondi ou de troncature peut être 'grande', donc, pour des raisons de stabilité numérique, il faut aussi amener par permutation de ligne ou de colonne de façon à avoir un pivot assez 'grand' en module ; on fait alors un pivotage. Remarqouns que si le pivot est nul après tout pivotage alors la matrice A est singulière.

Exemple 5 *A l'aide d'un ordinateur fictif à 3 chiffres déçimaux significatifs la résolution du système ci-dessous, qui est traitée dans l'exercice 22 :*

$$\begin{cases} 10^{-4}x + y &= 1 \\ x + y &= 2 \end{cases}$$

donne

- *sans permutation de lignes, la solution* $(0, 1)$ *;*
- *avec permutation de lignes, la solution* $(1, 1)$.

La solution exacte est $(1.0001, 0.9999)$*, ce qui montre que la première solution est fausse alors que la deuxième est la meilleure que l'on puisse obtenir, compte-tenu de la précision de la machine.*

———————-

COMPLETER ce cours

———————

3.2 Enoncés des exercices corrigés

Exercice 19 :
Soit le système linéaire suivant :

$$(1)\ \begin{cases} 3x_1 - 2x_2 + x_3 &= 2 \\ 2x_1 + x_2 + x_3 &= 7 \\ 4x_1 - 3x_2 + 2x_3 &= 4 \end{cases}$$

a) Résoudre ce système par la méthode de Gauss.
b) Factoriser la matrice A du système en produit LU où L est une matrice triangulaire inférieure (avec des 1 sur la diagonale principale) et U triangulaire supérieure, puis résoudre ce système.

Exercice 20 :
Soit le système linéaire $AX = B$ où : $A = \begin{pmatrix} 1 & 0 & -3 \\ 0 & 1 & 2 \\ 4 & -3 & 0 \end{pmatrix}$, $X = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}$, et $B = \begin{pmatrix} 2 \\ 5 \\ -1 \end{pmatrix}$. Factoriser la matrice A en produit LU puis résoudre le système.

Exercice 21 :
Soit le système linéaire $AX = B$ où : $A = \begin{pmatrix} 2 & 3 & -1 \\ 4 & 4 & -3 \\ -2 & 3 & -1 \end{pmatrix}$, $X = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}$, et $B = \begin{pmatrix} 2 \\ 0 \\ -1 \end{pmatrix}$.

1. Factoriser la matrice A en produit LU puis résoudre le système.
2. Trouver A^{-1} (on rappelle que si M et N sont deux matrices (n,n) inversibles alors $(MN)^{-1} = N^{-1}M^{-1}$).

Exercice 22 :
Soit le système linéaire :

$$(2)\ \begin{cases} \epsilon x + y &= 1 \\ x + y &= 2 \end{cases}$$

Appliquer la méthode de Gauss à ce système :
a) directement,
b) en permutant les deux lignes du système.
Application avec $\epsilon = 10^{-4}$ et un ordinateur fictif à 3 chiffres décimaux significatifs. Qu'obtient-on dans les deux cas ? expliquer ces résultats .

Exercice 23 :
Résoudre le système suivant :
a) En représentant les nombres en virgule flottante avec 8 chiffres significatifs,
b) puis exactement,

$$(3)\ \begin{cases} x+y & = 2 \\ (1+10^{-8})x+y & = 2+10^{-8} \end{cases}$$

Exercice 24 :
On applique la méthode de Gauss au système tridiagonal suivant :

$$\begin{pmatrix} b_1 & c_1 & 0 & 0 & 0 & \cdots & & 0 \\ a_2 & b_2 & c_2 & 0 & & & & 0 \\ 0 & a_3 & b_3 & c_3 & & & & 0 \\ . & & & & & & & . \\ . & & & & & & & . \\ . & & & & & & & . \\ 0 & & & & & & & 0 \\ 0 & & & & 0 & a_{n-1} & b_{n-1} & c_{n-1} \\ 0 & & & & 0 & 0 & a_n & b_n \end{pmatrix} . \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ . \\ . \\ . \\ . \\ x_{n-1} \\ x_n \end{pmatrix} = \begin{pmatrix} k_1 \\ k_2 \\ k_3 \\ . \\ . \\ . \\ . \\ k_{n-1} \\ k_n \end{pmatrix}$$

Montrer que la triangularisation revient à poser :

$$\begin{cases} c_n = 0 \\ w_1 = \frac{c_1}{b_1}, \quad y_1 = \frac{k_1}{b_1} \\ w_i = \dfrac{c_i}{b_i - a_i w_{i-1}} & \text{pour i=2, ... ,n} \\ y_i = \dfrac{k_i - a_i y_{i-1}}{b_i - a_i w_{i-1}} & \text{pour i=2, ... ,n .} \end{cases}$$

Evaluer le nombre d 'opérations nécessaires, y compris la résolution du système triangulaire.

Exercice 25 :

Résolution d'un système non linéaire dans $\mathbb{R}^2$

$$\begin{cases} x - x^2 - y^2 & = & 0 & (1) \\ y - x^2 + y^2 & = & 0 & (2) \end{cases}$$

3.3 Enoncés des exercices non corrigés

Exercice 26 :
Résoudre par Gauss direct, puis par factorisation de la matrice le système :

$$\begin{cases} x + \frac{y}{2} + \frac{z}{3} & = 1 \\ \frac{x}{2} + \frac{y}{3} + \frac{z}{4} & = 0 \\ \frac{x}{3} + \frac{y}{4} + \frac{z}{5} & = 0 \end{cases}$$

Exercice 27 :
Soit le système linéaire en dimension 3 , $AX = B$, où α et β sont deux réels et

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 0 & \alpha & 1 \\ 3 & 3 & 6 \end{pmatrix} \quad , \quad X = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \quad \text{et} \quad B = \begin{pmatrix} \beta \\ 2 \\ 0 \end{pmatrix}.$$

1. Donner la décomposition LU de A où L est une matrice triangulaire à diagonale unité et U une matrice triangulaire supérieure. Pour quelles valeurs de α cette décomposition existe.
2. Pour quelles valeurs de α et β ce système admet une solution unique ; donner cette solution.
3. En utilisant la décomposition LU trouver A^{-1} lorsqu'elle existe.

Exercice 28 :

Soit le système linéaire en dimension 4, $CX = d$, où :

$$C = \begin{pmatrix} 3 & 0 & -1 & 0 \\ 2 & 1 & \frac{1}{3} & 0 \\ 1 & 1 & 0 & -1 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad , \quad X = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} \quad \text{et} \quad d = \begin{pmatrix} 1 \\ \frac{1}{3} \\ \frac{4}{3} \\ 5 \end{pmatrix}.$$

1. La résolution de ce système peut être ramenée à celle d'un système linéaire de dimension trois, l'inconnue x_4 étant facile à déterminer. Donner cette inconnue et le système 3×3 restant, que l'on notera $Ax = b$.
2. Factoriser la matrice A en produit LU (où L est triangulaire inféieure à diagonale unité et U triangulaire supérieure) puis résoudre le système.
3. De $A = LU$ déduire A^{-1} ;

Exercice 29 :
Soit le système linéaire en dimension 4, $CX = d$, où :

$$C = \begin{pmatrix} 7 & 0 & 0 & 0 \\ 0 & 1 & 1 & -1 \\ 0 & 2 & 0 & -3 \\ 0 & -2 & 1 & 2 \end{pmatrix} \quad , \quad X = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} \quad \text{et} \quad d = \begin{pmatrix} -5 \\ 1 \\ 3 \\ -\frac{3}{2} \end{pmatrix}.$$

1. La résolution de ce système peut être ramenée à celle d'un système linéaire de dimension trois, l'inconnue x_1 étant facile à déterminer. Donner cette inconnue et le système 3X3 restant, que l'on notera $Ax = b$, ($x = (x_2, x_3, x_4)^T$).
2. Factoriser la matrice A en produit LU (où L est triangulaire inférieure à diagonale unité et U triangulaire supérieure), puis résoudre le système.

Exercice 30 :
Soit le système linéaire en dimension 4, $CX = d$, où :

$$C = \begin{pmatrix} 1 & 2 & 0 & 3 \\ 2 & 2 & 0 & 3 \\ 0 & 0 & 4 & 0 \\ 3 & 3 & 0 & 3 \end{pmatrix} \quad , \quad X = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} \quad \text{et} \quad d = \begin{pmatrix} 0 \\ 1 \\ 11 \\ 0 \end{pmatrix}.$$

1. La résolution de ce système peut être ramenée à celle d'un système linéaire de dimension trois. Quelle est l'inconnue x_j facile à déterminer. Donner cette inconnue et le système 3X3 restant, que l'on notera $Ax = b$.
2. Factoriser la matrice A en produit LU (où L est triangulaire inférieure à diagonale unité et U triangulaire supérieure) puis résoudre le système.
3. Trouver A^{-1}.

3.4 Corrigés des exercices

exercice 19

$$\left\{\begin{array}{lcl} 3x_1 - 2x_2 + x_3 & = & 2 \\ 2x_1 + x_2 + x_3 & = & 7 \\ 4x_1 - 3x_2 + 2x_3 & = & 4 \end{array}\right.$$

Ce système s'écrit sous la forme $AX = B$, où

$$A = \begin{pmatrix} 3 & -2 & 1 \\ 2 & 1 & 1 \\ 4 & -3 & 2 \end{pmatrix} \text{ et } B = \begin{pmatrix} 2 \\ 7 \\ 4 \end{pmatrix}$$

Posons $A^{(1)} = A$, on calcule $A^{(2)} = M^{(1)}A^{(1)}$, où

$$M^{(1)} = \begin{pmatrix} 1 & 0 & 0 \\ -\frac{2}{3} & 1 & 0 \\ -\frac{4}{3} & 0 & 1 \end{pmatrix},$$

d'où :

$$A^{(2)} = \begin{pmatrix} 3 & -2 & 1 \\ 0 & \frac{7}{3} & \frac{1}{3} \\ 0 & -\frac{1}{3} & \frac{2}{3} \end{pmatrix}$$

on calcule $A^{(3)} = M^{(2)}A^{(2)}$, où

$$M^{(2)} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & \frac{1}{7} & 1 \end{pmatrix}.$$

Donc,

$$A^{(3)} = \begin{pmatrix} 3 & -2 & 1 \\ 0 & \frac{7}{3} & \frac{1}{3} \\ 0 & 0 & \frac{5}{7} \end{pmatrix}.$$

La matrice $A^{(3)}$ est ainsi triangulaire supérieure, c'est la matrice U recherchée.

D'autre part, on a $A^{(3)} = M^{(2)}A^{(2)} = M^{(2)}M^{(1)}A^{(1)}$, on en déduit donc que

$$A^{(1)} = \underbrace{(M^{(1)})^{-1}(M^{(2)})^{-1}}_{L}\underbrace{A^{(3)}}_{U}.$$

Ainsi, $A = A^{(1)} = LU$, avec

$$L = \begin{pmatrix} 1 & 0 & 0 \\ \frac{2}{3} & 1 & 0 \\ \frac{4}{3} & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -\frac{1}{7} & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ \frac{2}{3} & 1 & 0 \\ \frac{4}{3} & -\frac{1}{7} & 1 \end{pmatrix}.$$

On a ainsi factorisé A sous la forme :

$$A = \begin{pmatrix} 1 & 0 & 0 \\ \frac{2}{3} & 1 & 0 \\ \frac{4}{3} & -\frac{1}{7} & 1 \end{pmatrix} \begin{pmatrix} 3 & -2 & 1 \\ 0 & \frac{7}{3} & \frac{1}{3} \\ 0 & 0 & \frac{3}{7} \end{pmatrix}.$$

Présantation de la méthode d'identification

Résoudre $AX = B$ revient à résoudre $LUX = B$. On pose alors $Y = UX$, la résolution du système initial revient à résoudre successivement les deux systèmes triangulaires :

$$\begin{cases} LY &= B \\ UX &= Y \end{cases}$$

$$LY = B \iff \begin{pmatrix} 1 & 0 & 0 \\ \frac{2}{3} & 1 & 0 \\ \frac{4}{3} & -\frac{1}{7} & 1 \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} 2 \\ 7 \\ 4 \end{pmatrix} \Rightarrow Y = \begin{pmatrix} 2 \\ \frac{17}{3} \\ \frac{15}{7} \end{pmatrix}.$$

Finalement, on résout :

$$UX = Y \iff \begin{pmatrix} 3 & -2 & 1 \\ 0 & \frac{7}{3} & \frac{1}{3} \\ 0 & 0 & \frac{3}{7} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 2 \\ \frac{17}{3} \\ \frac{15}{7} \end{pmatrix} \Rightarrow X = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}.$$

Exercice 20

Soit le système linéaire $AX = B$ où :

$$A = \begin{pmatrix} 1 & 0 & -3 \\ 0 & 1 & 2 \\ 4 & -3 & 0 \end{pmatrix} \text{ et } B = \begin{pmatrix} 2 \\ 5 \\ -1 \end{pmatrix}.$$

Factorisons la matrice A en produit LU.

Posons $A^{(1)} = A$, on calcule $A^{(2)} = M^{(1)}A^{(1)}$, où

$$M^{(1)} = \begin{pmatrix} 1 & 0 & 0 \\ -0 & 1 & 0 \\ -4 & 0 & 1 \end{pmatrix},$$

d'où :

$$A^{(2)} = \begin{pmatrix} 1 & 0 & -3 \\ 0 & 1 & 2 \\ 0 & -3 & 12 \end{pmatrix}$$

on calcule $A^{(3)} = M^{(2)}A^{(2)}$, où

$$M^{(2)} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 3 & 1 \end{pmatrix}.$$

Donc,

$$A^{(3)} = \begin{pmatrix} 1 & 0 & -3 \\ 0 & 1 & 2 \\ 0 & 0 & 18 \end{pmatrix}.$$

La matrice $A^{(3)}$ est ainsi triangulaire supérieure, c'est la matrice U recherchée. D'autre part, on a $A^{(3)} = M^{(2)}A^{(2)} = M^{(2)}M^{(1)}A^{(1)}$. On en déduit donc

$$A^{(1)} = \underbrace{(M^{(1)})^{-1}(M^{(2)})^{-1}}_{L}\underbrace{A^{(3)}}_{U}.$$

Ainsi, $A = A^{(1)} = LU$, avec

$$L = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 4 & 0 & 1 \end{pmatrix}\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -3 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 4 & -3 & 1 \end{pmatrix}.$$

A se factorise donc sous la forme :

$$A = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 4 & -3 & 1 \end{pmatrix}\begin{pmatrix} 1 & 0 & -3 \\ 0 & 1 & 2 \\ 0 & 0 & 18 \end{pmatrix}.$$

Résolvons le système $AX = B$. Cela revient à résoudre $LUX = B$, c'est à dire à résoudre successivement les systèmes $LY = B$ puis $UX = Y$.

$$LY = B \iff \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 4 & -3 & 1 \end{pmatrix}\begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} 2 \\ 5 \\ -1 \end{pmatrix} \Rightarrow Y = \begin{pmatrix} 2 \\ 5 \\ 6 \end{pmatrix}.$$

Finalement, on résout :

$$UX = Y \iff \begin{pmatrix} 1 & 0 & -3 \\ 0 & 1 & 2 \\ 0 & 0 & 18 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 2 \\ 5 \\ 6 \end{pmatrix} \Rightarrow X = \begin{pmatrix} 3 \\ \frac{13}{3} \\ \frac{1}{3} \end{pmatrix}.$$

Exercice 18

C'est un exercie sur le pivot de Gauss. Soit le système linéaire :

$$(1)\ \begin{cases} \epsilon x + y & = 1 \quad (L_1) \\ x + y & = 2 \quad (L_2) \end{cases}$$

a)Appliquons la méthode de Gauss classique à ce système :
On élimine alors le premier terme de la seconde ligne (L_2) du système (1) grâce à la combinaison linéaire $(L_2) - \dfrac{a_{21}}{a_{11}}(L_1)$ d'où le système

$$(2)\ \begin{cases} \epsilon x + y & = 1 \quad (L_1) \\ 0 + (1 - \frac{1}{\epsilon} \times 1)y & = 2 - \frac{1}{\epsilon} \times 1 \quad (L_2') \end{cases}$$

Si on pose $\epsilon = 10^{-4}$ alors le système (2) s'écrit

$$(2')\ \begin{cases} 10^{-4}x + y & = 1 \quad (L_1) \\ 0 + (1 - \frac{1}{10^{-4}})y & = 2 - \frac{1}{10^{-4}} \times 1 \quad (L_2') \end{cases}$$

SI on traite ce calcule avec un ordinateur fictif approchant les nombres flottants (les 'réels' sur ordinateur) avec 3 chiffres décimaux significatifs, alors dans (2') on aura

$$-10^4 y \approx -10^4,$$

et on obtiendrait ainsi

$$y \approx 1$$

d'où en reportant dans la première ligne du système (2') on trouve

$$x = 0.$$

La solution du système donnée par cet ordinateur fictif serait alors $(0, 1)$.

b) Permutant maintenant les deux lignes du système initial (1). on résout donc

$$(3)\ \begin{cases} x + y & = 2 \quad (L_2) \\ \epsilon x + y & = 1 \quad (L_1) \end{cases}$$

La méthode de Gauss donne :

$$(3')\ \begin{cases} x + y & = 2 \quad (L_2) \\ 0 + (1 - \frac{\epsilon}{1} \times 1)y & = 1 - \frac{\epsilon}{1} \times 2 \quad (L_1') \end{cases}$$

Soit en posant $\epsilon = 10^{-4}$,

$$(2')\ \begin{cases} x + y & = 2 \quad (L_2) \\ 0 + (1 - \frac{10^{-4}}{1} \times 1)y & = 1 - \frac{10^{-4}}{1} \times 2 \quad (L_1') \end{cases}$$

D'où, avec le même ordinateur fictif,

$$y \approx 1,$$

et en reportant dans la première ligne du système (2') on trouve

$$x \approx 1.$$

La solution du système donnée par cet ordinateur fictif serait alors $(1, 1)$, très proche de la solution exacte qui est bien sur $(1.0001, 0.9999)$.

La solution obtenue grâce au système (1) sans permutation de lignes est erronée car, lors de l'élimination de Gauss, on divise par le pivot $\epsilon = 10^{-4}$, très voisin de la précision de la machine utilisée, d'où l'erreur numérique. En permutant les deux lignes du système on évite ce problème puisque le nouveau pivot est alors égal à 1, valeur très grande par rapport à la précision de la machine.

Exercice 19

Résoudre le système suivant :
a) En représentant les nombres en virgule flottante avec 8 chiffres significatifs,
b) puis exactement,

$$(3) \begin{cases} x + y & = 2 \\ (1 + 10^{-8})x + y & = 2 + 10^{-8} \end{cases}$$

SOLUTION a TAPER ...

Exercice 20

$$\begin{pmatrix} b_1 & c_1 & & & \dots & 0 & 0 & 0 \\ a_2 & b_2 & c_2 & & & \vdots & \vdots & \vdots \\ & a_3 & b_3 & c_3 & & & & \\ & & \ddots & \ddots & \ddots & & & \\ & & & \ddots & \ddots & \ddots & & \\ & & & & \ddots & \ddots & \ddots & \\ \vdots & \vdots & \vdots & & & a_{n-1} & b_{n-1} & c_{n-1} \\ 0 & 0 & 0 & \dots & & & a_n & b_n \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ \\ \\ \vdots \\ x_{n-1} \\ x_n \end{pmatrix} = \begin{pmatrix} k_1 \\ k_2 \\ k_3 \\ \vdots \\ \\ \\ \vdots \\ k_{n-1} \\ k_n \end{pmatrix}$$

Triangularisation : On annule a_2 en faisant $L_2 \leftarrow L_2 - L_1\frac{a_2}{b_1}$, on a alors :

$$\begin{pmatrix} b_1 & c_1 & & & \dots & 0 & 0 & 0 \\ 0 & b_2 - \frac{a_2}{b_1} & c_2 & & & \vdots & \vdots & \vdots \\ & a_3 & b_3 & c_3 & & & & \\ & & \ddots & \ddots & \ddots & & & \\ & & & \ddots & \ddots & \ddots & & \\ & & & & \ddots & \ddots & \ddots & \\ \vdots & & \vdots & \vdots & & a_{n-1} & b_{n-1} & c_{n-1} \\ 0 & & 0 & 0 & \dots & & a_n & b_n \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ \\ \\ \vdots \\ x_{n-1} \\ x_n \end{pmatrix} = \begin{pmatrix} k_1 \\ k_2 - \frac{a_2}{b_1}k_1 \\ k_3 \\ \vdots \\ \\ \\ \vdots \\ k_{n-1} \\ k_n \end{pmatrix}$$

Si on pose

$$\begin{cases} w_1 = \frac{c_1}{b_1} & y_1 = \frac{k_1}{b1} & (\text{et } c_n = 0) \\ w_i = \frac{c_i}{b_i - a w_{i-1}} & y_i = \frac{k_i - a_i y_{i-1}}{b_i - a_i w_{i-1}} & \text{pour } i = 2, ..., n \end{cases}$$

Cette étape s'écrit alors

$$\begin{pmatrix} b_1 & c_1 & & & \dots & 0 & 0 & 0 \\ 0 & b_2 - a_2 w_1 & c_2 & & & \vdots & \vdots & \vdots \\ & a_3 & b_3 & c_3 & & & & \\ & & \ddots & \ddots & \ddots & & & \\ & & & \ddots & \ddots & \ddots & & \\ & & & & \ddots & \ddots & \ddots & \\ \vdots & & \vdots & \vdots & & a_{n-1} & b_{n-1} & c_{n-1} \\ 0 & & 0 & 0 & \dots & & a_n & b_n \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ \\ \\ \vdots \\ x_{n-1} \\ x_n \end{pmatrix} = \begin{pmatrix} k_1 \\ k_2 - a_2 y_1 \\ k_3 \\ \vdots \\ \\ \\ \vdots \\ k_{n-1} \\ k_n \end{pmatrix}$$

——— ? ? ? ? ? ? ? ? ? ? ? ? ? ——————— COMPleter ici ——————-

Et ainsi de suite. . . On retrouve bien les formules proposées.

Nombre d'opérations :

$$\begin{aligned} w_1 &\rightarrow 1 \text{ opération} \\ y_1 &\rightarrow 1 \text{ opération} \\ w_i &\rightarrow 1 \text{ division} + 1 \text{ soustraction} + 1 \text{ multiplication, } (n-1) \text{ fois} \\ y_i &\rightarrow 1 \text{ division} + 2 \text{ soustraction} + 2 \text{ multiplication, } (n-1) \text{ fois} \\ 2^{nd} membre &\rightarrow (n-1)(1 \text{ soustraction} + 1 \text{ mutliplication}) \end{aligned}$$

Exercice 21

Résolution d'un système dans $\mathbb{R}^2$

$$\begin{cases} x - x^2 - y^2 &= 0 \quad (1) \\ y - x^2 + y^2 &= 0 \quad (2) \end{cases}$$

$$\begin{aligned} (1) &\Leftrightarrow (x - \frac{1}{2})^2 - y^2 = \frac{1}{4} \\ (2) &\Leftrightarrow (y + \frac{1}{2})^2 - x^2 = \frac{1}{4} \end{aligned}$$

1. **Méthode du point fixe.**

$$\begin{cases} x_{n+1} &= x_n^2 + y_n^2 &= f(x_n, y_n) \\ y_{n+1} &= x_n^2 - y_n^2 &= g(x_n, y_n) \end{cases}$$

NB : pour que le processus du point fixe converge, il suffit que

$$L = \max_{x \in D} \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2 + \left(\frac{\partial g}{\partial x}\right)^2 + \left(\frac{\partial g}{\partial y}\right)^2} < 1$$

ici, $F(x, y) = x^2 + y^2$ et $G(x, y) = x^2 - y^2$, donc,

$$L = \max_{X \neq 0} \sqrt{(2x)^2 + (2y)^2 + (2x)^2 + -(2y)^2} = \max \sqrt{8(x^2 + y^2)} < 1$$

c.à.d. que $\max_{X \neq 0} \sqrt{x^2 + y^2} < \frac{1}{2\sqrt{2}}$.
Pour $x_0 = 0,8$ et $y_0 = 0,4$, on a :

n	x_n	y_n
1	$0,8$	$0,48$
2	$0,8704$	$0,4096$
3	$0,9205$	$0,5898$
4	$1,2040$	$0,5081$

Ce procédé diverge-t-il ? Il faut montrer qu'en $(\bar{x}, \bar{y})$, on a $\sqrt{x^2 + y^2} > \frac{1}{2\sqrt{2}}$. Or, $x_{n+1} = x_n^2 + y_n^2$ donc $\bar{x} = \bar{x}^2 + \bar{y}^2$. Ainsi, $\max \sqrt{8(\bar{x}^2 + \bar{y}^2)} = \max \sqrt{8\bar{x}}$, et $\sqrt{8\bar{x}} < 1$ implique que $\sqrt{\bar{x}} < \frac{1}{2\sqrt{2}}$, ce qui est impossible puisque $\bar{x} \simeq 0,8$. Donc, $\sqrt{\bar{x}} \simeq \sqrt{0,8} \simeq 0,89 > 0,35\ldots$ La méthode diverge.

2. **Méthode de Newton.**
   $X_{n+1} = X_n + \Delta X_n$, avec

$$\begin{aligned} X_n &= \begin{pmatrix} x_n \\ y_n \end{pmatrix}, \text{ et} \\ x_n &= \frac{1}{D_n}\left(-F(x_n, y_n) \cdot \frac{\partial G}{\partial y}(x_n, y_n) + \frac{\partial F}{\partial y}(x_n, y_n)\right) \\ y_n &= \frac{1}{D_n}\left(-\frac{\partial F}{\partial x}(x_n, y_n) \cdot G(x_n, y_n) + \frac{\partial G}{\partial x}(x_n, y_n)\right) \\ D_n &= \left(\frac{\partial F}{\partial x} \cdot \frac{\partial G}{\partial y} - \frac{\partial F}{\partial y} \cdot \frac{\partial G}{\partial x}\right) \end{aligned}$$

   Ici,

$$\begin{aligned} F(x,y) &= x - x^2 - y^2 \quad \Rightarrow \quad \frac{\partial F}{\partial x} = 1 - 2x \\ \frac{\partial F}{\partial y} &= -2y \\ G(x,y) &= y - x^2 + y^2 \quad \Rightarrow \quad \frac{\partial G}{\partial x} = -2x \\ \frac{\partial G}{\partial y} &= 1 + 2y \end{aligned}$$

   Par conséquent, Newton s'écrit :

$$\begin{aligned} x_{n+1} &= x_n + \frac{(x_n^2 + y_n^2 - x_n)(1 + 2y_n) + (-2y_n)(y_n - x_n^2 - y_n^2)}{(1 - 2x_n)(1 + 2y_n) - (-2x_n)(-2y_n)} \\ y_{n+1} &= y_n + \frac{(2x_n - 1)(y_n - x_n^2 + y_n^2) + (x_n - x_n^2 - y_n^2)(-2x_n)}{(1 - 2x_n)(1 + 2y_n) - (2x_n)(2y_n)} \end{aligned}$$

   Pour $x_0 = 0,8$ et $y_0 = 0,4$, on a :

n	x_n	y_n
0	$0,8$	$0,4$
1	$0,8$	$0,48$
3	....	
4	$0,77 = \bar{x}$	$0,42 = \bar{y}$

3.5 Mise en œuvre en Java

3.5.1 Les tableaux multidimensionnels en Java

Les tableaux multidimensionnels en Java correspondent à des tableaux de tableaux. Ce sont donc des objets et les déclarer sans les construire avec `new` consiste à réserver une adresse pour l'objet. Il est alors possible de construire séparemment les sous-tableaux qui ne sont pas nécessairement de taille identique.

On peut donc, soit créer des tableaux de taille homogènes, comme dans la construction qui suit :

```
int [] [] t = new int[5][10] ;
```

soit construire des sous-tableaux de tailles distinctes, comme ci-dessous :

```
int m [] [];
m = new int [3] []; // tableau pouvant contenir les 3 adresses
                    // des sous-tableaux
m[0] = new int[2];
m[1] = new int[4];
m[2] = new int[1];
```

Notons aussi que le calcul de la longueur d'un tableau multidimensionnel avec `length` correspond au domaine de variation de premier indice du tableau : ainsi, dans l'exemple précédent, `m.length` vaut `3`.

3.5.2 Une classe matrice

On construit ci-dessous une classe matrice qui permet de définir les opérateurs matriciels classiques, d'une manière non exhaustive. A partir de cette construction, le lecteur pourra facilement ajouter les opérateurs dont il a besoin en s'inspirant de ceux déjà définis.

Le stockage des coefficients de la matrice se fait grâce à un tableau d'objets du type vecteur que l'on a définit dans le premier chapitre, chacun des vecteurs ayant la même taille. Ce tableau est privé et on définit les fonctions `coef( , )` et `toCoef( , , )` qui permettent respectivement d'aller rechercher un coefficient et d'en changer la valeur.

```
public class matrice
{
    private vecteur[] composant;

    matrice (int dim1, int dim2)
```

```
    /** constructeur creant une matrice de coefficients nuls
      * de dim1 lignes
      * et dim2 colonnes
      */
    {
composant = new vecteur [dim1];
for (int i=0; i<dim1; i++)
    composant[i] = new vecteur(dim2);
    }

    matrice (double tableau [][])
    /** constructeur creant une matrice a partir du
      * parametre tableau
      */
    {
composant = new vecteur [tableau.length];
for (int i=0; i<tableau.length; i++)
    composant[i] = new vecteur (tableau[i]);
    }

    public int nbLignes()
    /** renvoie le nombres de lignes de la matrice
      */
    { return composant.length; }

    public int nbColonnes()
    /** renvoie le nombre de colonnes de la matrice
      */
    { return composant[0].dim(); }

    public double coef(int nl, int nc)
    /** renvoie le coefficient a la position (nl, nc)
      */
    { return composant[nl].elt(nc); }

    public void toCoef(int i, int j, double x)
    /** affecte la valeur de x au coefficient a la
      * position (nl, nc)
      */
    { composant[i].toElt(j, x); }
```

```
    void afficher()
    /** affiche les coefficients de la matrice
      */
    {
for (int i=0; i<nbLignes(); i++)
{
    for (int j=0; j<nbColonnes(); j++)
System.out.print(coef(i,j)+" ");
    System.out.println("");
}
System.out.println("");
    }

    public static vecteur produit(matrice m, vecteur x)
    /** renvoie le vecteur obtenu en multipliant
      * la matrive m par le vecteur x
      */
    {
vecteur y = new vecteur(m.nbLignes());
for (int i= 0; i<m.nbLignes(); i++)
{
    double somme = 0.0;
    for (int j= 0; j<m.nbColonnes(); j++)
somme += m.coef(i,j) * x.elt(j);
    y.toElt(i, somme);
}
return y;
    }

    public static matrice produit(matrice m1, matrice m2)
    /** renvoie la matrice obtenue en multipliant les deux
      * matrices m1 et m2
      */
    {
matrice result = new matrice(m1.nbLignes(), m2.nbColonnes());
for (int i=0; i<m1.nbLignes(); i++)
    for (int j=0; j<m2.nbColonnes(); j++)
    {
double somme = 0.0;
for (int k=0; k<m1.nbColonnes(); k++)
    somme += m1.coef(i,k) * m2.coef(k,j);
```

```
result.toCoef(i, j, somme);
    }
return result;
    }

    public static matrice addition(matrice m1, matrice m2)
    /** renvoie la matrice obtenue en additionnant les deux
      * matrices m1 et m2
      */
    {
matrice result = new matrice(m1.nbLignes(), m2.nbColonnes());
for (int i=0; i<m1.nbLignes(); i++)
    for (int j=0; j<m2.nbColonnes(); j++)
result.toCoef(i, j, m1.coef(i,j) + m2.coef(i,j));
return result;
    }

    public static matrice soustraction(matrice m1, matrice m2)
    /** renvoie la matrice obtenue en soustrayant la matrice m2
      * a la matrice m1
      */
    {
matrice result = new matrice(m1.nbLignes(), m2.nbColonnes());
for (int i=0; i<m1.nbLignes(); i++)
    for (int j=0; j<m2.nbColonnes(); j++)
result.toCoef(i, j, m1.coef(i,j) - m2.coef(i,j));
return result;
    }

}
```

Programme de test :

```
class testMatrice
{
    public static void main(String args[])
    {
        double [][] t1 = { {1., 2., 3.}, {4.,  5., 6.} };
        double [][] t2 = { {2., 4.}, {4.,  2.}, {1., 1.} };
        double [][] t3 = { {2., 3., 2.}, {5., 6., 5.} };
        double [] t = {2., 1., 0.};
```

```
        matrice m1 = new matrice(2, 3);
        for (int i=0; i<m1.nbLignes(); i++)
            for (int j=0; j<m1.nbColonnes(); j++)
                m1.toCoef(i, j, t1[i][j]);
        System.out.println("matrice m1 :"); m1.afficher();

        matrice m2 = new matrice(t2);
        System.out.println("matrice m2 :"); m2.afficher();

        matrice m3 = new matrice(t3);
        System.out.println("matrice m3 :"); m3.afficher();

        vecteur x = new vecteur(t);
        System.out.println("vecteur x :"); x.afficher();
        vecteur y = matrice.produit(m1, x);
        System.out.println("m1*x :"); y.afficher();

        matrice m4 = matrice.produit(m1, m2);
        System.out.println("m1*m2 : "); m4.afficher();

        m4 = matrice.addition(m1, m3);
        System.out.println("m1+m3 : ");m4.afficher();

        m4 = matrice.soustraction(m1, m3);
        System.out.println("m1-m3 : ");m4.afficher();

    }
}
```

L'exécution du programme précédent génère l'affichage suivant :

```
java testMatrice
matrice m1 :
1.0 2.0 3.0
4.0 5.0 6.0

matrice m2 :
2.0 4.0
4.0 2.0
1.0 1.0
```

```
matrice m3 :
2.0 3.0 2.0
5.0 6.0 5.0

vecteur x :
2.0 1.0 0.0
m1*x :
4.0 13.0
m1*m2 :
13.0 11.0
34.0 32.0

m1+m3 :
3.0 5.0 5.0
9.0 11.0 11.0

m1-m3 :
-1.0 -1.0 1.0
-1.0 -1.0 1.0
```

3.5.3 Une classe abstraite de système linéaire

Nous allons être amené à traiter et en particulier à résoudre des systèmes linéaires de différentes natures, à savoir triangulaire supérieur ou inférieur, à diagonal unité ou non, ou encore des systèmes généraux. Les grandes bibliothèques connues de type BLAS (Basic Linear Algebra Subroutine) construisent autant de fonctions de résolution distinctes qu'il y a de types différents de systèmes. S'appuyant sur le principe de réutilisabilité de la programmation objet, nous allons définir une classe abstraite générale qui sera le dénominateur commun de tous les sytèmes linéaires et dont héritera les classes spécialisées.

On considerera qu'un système linéaire ne peut être construit qu'après la construction effective de la matrice et du second membre du système et le constructeur d'un système ne peut donc se faire qu'avec ses deux éléments. Par ailleurs, la seule méthode qui nous intéresse, pour l'instant, est celle qui calcule la solution du système. Cette résolution peut éventuellement conduire à un échec si la matrice est singulière et non allons pour cela faire la gestion d'une exception `SysLinEception` succeptible d'être levée en cours de calcul.

La classe abstraite générale et la classe d'exception associée sont alors les suivantes :

```
class SysLinException extends Exception{
    public String toString(){
return( "Systeme singulier");
    }
}

abstract class sysLin{
    int ordre;
    protected matrice matriceSysteme;
    protected vecteur secondMembre;
    sysLin (matrice m, vecteur x){
matriceSysteme = m;
secondMembre = x;
ordre = x.dim();
    }
    abstract public vecteur resolution() throws SysLinException;
}
```

3.5.4 Les classes systèmes linéaires triangulaires

Nous pouvons écrire maintenant une classe de systèmes linéaires triangulaires supérieurs qui va implémenter la méthode `resolution` et gérer l'exception `SysLinException` lorsqu'un élément de la diagonale de la matrice est nul :

```
class sysTriangSup extends sysLin {

    sysTriangSup(matrice m, vecteur x) { super(m,x); }

    public vecteur resolution() throws SysLinException {
vecteur x = new vecteur(ordre);
double somme;
for (int i=ordre-1; i>=0; i--){
    somme=secondMembre.elt(i);
    for (int j=i+1; j<ordre; j++)
somme -= matriceSysteme.coef(i,j) * x.elt(j);
    double termDiagonal = matriceSysteme.coef(i,i);
    if (termDiagonal == 0) throw new SysLinException();
    x.toElt(i, somme / matriceSysteme.coef(i,i));
}
return x;
    }
```

```
}
```

Pour tester notre classe, il nous faut saisir la matrice et le second membre du système. Pour cela nous allons passer par l'intermédiaire d'un fichier en reprenant les constructions qui ont été présentées au premier chapitre et en y ajoutant l'utilisation d'un analyseur de type `StringTokenizer` qui va pouvoir décomposer une ligne constituée de plusieurs valeurs numériques et les extraire. Une lecture attentive du listing qui suit permet facilement d'en comprendre le fonctionnement : tout est basé sur l'utilisation de la fonction `nextToken()` qui décompose la chaîne jusqu'au prochain séparateur.

```
import java.io.*;
import java.util.*;
class bibMatFileIn{
    FileReader fichier=null;
    BufferedReader lecbuf;

    public bibMatFileIn(String nomfichier) {
try{
    fichier = new FileReader (nomfichier);
    lecbuf = new BufferedReader (fichier);
}
catch (FileNotFoundException e){
    System.out.println("fichier inexistant ! ");
}
    }

    vecteur lectureVecteur(){
String line, word;
vecteur vlu=null;
try{
    // lecture taille du vecteur (sur 1 ligne) :
    line = lecbuf.readLine();
    int nb = Integer.parseInt(line);
    vlu = new vecteur(nb);
    // lecture coef. du vecteur (sur 1 ligne) :
    int n=0;
    line = lecbuf.readLine();
    StringTokenizer st = new StringTokenizer(line);
    //     while ((word=st.nextToken()) != null)
    for(int i=0; i<nb; i++){
word = st.nextToken();
```

```
vlu.toElt(n++, Double.valueOf(word).doubleValue());
    }
}
catch(IOException e){
    System.out.println("erreur de lecture");
}
catch(NumberFormatException e){
    System.out.println("erreur conversion chaine");
}
return vlu;
    }

    matrice lectureMatrice(){
String line, word;
matrice mlu=null;
StringTokenizer st;
try{
    // lecture ordre matrice (sur 1 ligne) :
    line = lecbuf.readLine();
    st = new StringTokenizer(line);
    word = st.nextToken();
    int nbLignes = Integer.parseInt(word);
    word = st.nextToken();
    int nbColonnes = Integer.parseInt(word);
    mlu = new matrice (nbLignes, nbColonnes);
    // lecture coef. matrice, ligne par ligne :
    for (int i=0; i<nbLignes; i++){
line = lecbuf.readLine();
st = new StringTokenizer(line);
for (int j=0; j<nbColonnes; j++){
    word = st.nextToken();
    mlu.toCoef(i,j, Double.valueOf(word).doubleValue());
}
    }
}
catch(IOException e){
    System.out.println("erreur de lecture");
}
catch(NumberFormatException e){
    System.out.println("erreur conversion chaine");
}
```

```
return mlu;
    }

    public void fermer(){
try{ fichier.close(); }
catch (IOException e) {
    System.out.println("Le fichier ne peut pas être fermer");
}
    }
}
```

Nous écrivons ensuite une classe de test qui va mettre en œuvre une programme principal :

```
class testSysTriangSup{
    public static void main(String args[]){
bibMatFileIn f = new bibMatFileIn("donnee.dat");
// on lit dans le fichier "donnee.dat" :
// l'ordre de la matrice : nblignes et nbcolonnes sur 1 ligne
// les coef. de la matrice ligne par ligne
// la taille du second membre sur 1 ligne
// les coef. du second membre sur 1 ligne
// Attention : les dimensions matrices et vecteurs doivent
//             etre corrects ... pas de verification !
matrice m = f.lectureMatrice();
vecteur b = f.lectureVecteur();
f.fermer();
System.out.println("        SYSTEME LU");
System.out.println("La matrice lue est : "); m.afficher();
System.out.println("Le sd membre lu est : "); b.afficher();
sysTriangSup sts = new sysTriangSup(m, b);
try{
    vecteur x = sts.resolution();
    System.out.println("La solution trouvee est : "); x.afficher();
    vecteur v = matrice.produit(m, x);
    System.out.println("Verification - le produit de la matrice "+
       "par la solution vaut :");
    v.afficher();
}
catch (SysLinException e){
    System.out.println(e);
}
```

```
    }
}
```

On présente maintenant deux tests d'exécution.
Le fichier de données du premier test contient
– sur la première ligne, les dimensions de la matrice (5,5) ;
– sur les 5 lignes qui suivent, chacune des lignes de la matrice ;
– sur la ligne suivante, la taille du vecteur second membre ;
– sur la dernière ligne, les coefficients du second membre.
Le fichier est donc le suivant :

```
5 5
1 2 3 4 5
0 7 8 9 8
0 0 6 7 3
0 0 0 5 6
0 0 0 0 1
5
5 4 3 2 1
```

L'affichage généré par le programme, à partir de ce fichier, est le suivant :

```
java testSysTriangSup
        SYSTEME LU
La matrice lue est :
1.0 2.0 3.0 4.0 5.0
0.0 7.0 8.0 9.0 8.0
0.0 0.0 6.0 7.0 3.0
0.0 0.0 0.0 5.0 6.0
0.0 0.0 0.0 0.0 1.0

Le sd membre lu est :
5.0 4.0 3.0 2.0 1.0
La solution trouvee est :
1.6190476190476195 -0.6095238095238098 0.9333333333333336 -0.8 1.0
Verification - le produit de la matrice par la solution vaut :
5.0 4.0 3.0000000000000001 2.0 1.0
```

Le deuxième test sert à vérifier le bon fonctionnement de l'exception qui détecte un système singulier. Le fichier de données est le suivant :

```
5 5
1 2 3 4 5
```

```
0 7 8 9 8
0 0 6 7 3
0 0 0 5 6
0 0 0 0 0
5
5 4 3 2 1
```

L'affichage généré par le programme, à partir de ce fichier, est le suivant :

```
java testSysTriangSup
        SYSTEME LU
La matrice lue est :
1.0 2.0 3.0 4.0 5.0
0.0 7.0 8.0 9.0 8.0
0.0 0.0 6.0 7.0 3.0
0.0 0.0 0.0 5.0 6.0
0.0 0.0 0.0 0.0 0.0

Le sd membre lu est :
5.0 4.0 3.0 2.0 1.0
Systeme singulier
```

Nous construisons ensuite une classe système triangulaire inférieure sur le même modèle, mais en ne s'intéressant qu'au cas où la matrice est, de plus, à diagonale unitée. En effet, c'est ce cas qui sera rencontré lorsque l'on mettra en œuvre la méthode de factorisation LU.

```
class sysTriangInfUnite extends sysLin{

    sysTriangInfUnite (matrice m, vecteur x) { super(m, x); }

    public vecteur resolution() {
vecteur x = new vecteur(ordre);
double somme;
for (int i=0; i<ordre; i++){
    somme = secondMembre.elt(i);
    for (int j=0; j<i; j++)
somme -= matriceSysteme.coef(i,j) * x.elt(j);
    x.toElt(i, somme);
}
return x;
    }
}
```

3.5.5 La classe systèmes linéaires générals implémentant la factorisation LU

Rappels sur l'algorithme de calcul des coefficients de L et de U

Nous présentons ici un algorithme de calcul effectif des coefficients des matrices L et U qui factorise une matrice A d'ordre n. L est une matrice triangulaire inférieure à diagonale unité et U est une matrice triangulaire supérieure.

Nous procédons par identification à partir du produit

$$A = LU$$

avec $L_{ij} = 0$ si $j > i$, $L_{ij} = 1$ si $j = i$ et $U_{ij} = 0$ si $j < i$.

La formule du produit matriciel s'écrit :

$$A_{ij} = \sum_{k=1}^{n} L_{ik} U_{kj}$$

On considère alors les deux cas suivants :

– $j < i$

– $j \geq i$

... à finir ...

Compléments de la classe matrice avec la méthode de factorisation LU

L'algorithme de factorisation LU d'une matrice ne concerne que la matrice et doit donc trouver sa place dans la classe `matrice`. C'est pour cette raison que nous complètons celle-ci avec la méthode suivante, qui reprends l'algorithme précédent :

```
    public void factorLU() throws SysLinException
    /** calcul de la factorisation LU de la matrice courante
      * les coefficients de L et de U sont stockes en lieu et place
      * des coefficients de memes rangs de la matrice d'origine
      */
    {
int i, j, k;
double somme, coefDiagonal;
for (i=0; i<nbLignes(); i++) {
    for (j=0; j<i; j++) {
somme=coef(i,j);
for (k=0; k<j; k++)
    somme -= coef(i,k)*coef(k,j);
```

```
coefDiagonal = coef(j,j);
if (coefDiagonal == 0) throw new SysLinException();
toCoef(i, j, somme/coefDiagonal);
    }
    for (j=i; j<nbColonnes(); j++) {
somme=coef(i,j);
for (k=0; k<i; k++)
    somme -= coef(i,k)*coef(k,j);
toCoef(i, j, somme);
    }
}
    }
```

La classe système général résolu par factorisation LU

Nous pouvons alors définir une classe système général résolu par une méthode de factorisation LU. C'est la classe qui suit dans laquelle on notera que l'on a ajouté une méthode `resolutionPartielle()` qui permet de résoudre le système lorsque la matrice est déjà sous la forme factorisée. Il faut, en effet, se rappeler que l'un des intérêts de cette méthode de factorisation est qu'elle ne nécessite pas de refactoriser la matrice (opération la plus coûteuse) lorsque l'on doit résoudre plusieurs systèmes avec la même matrice. Par exemple, le calcul de l'inverse d'une matrice d'ordre n se fait en résolvant n systèmes ayant tous la même matrice : c'est la matrice à inverser. Les seconds membres sont, quant à eux, les n vecteurs de la base canonique.

La classe système général avec méthode de factorisation LU est donc la suivante :

```
class sysGeneLU extends sysLin {

    sysGeneLU (matrice m, vecteur x) { super(m,x); }

    public vecteur resolution() throws SysLinException {
vecteur x = null;
try{
    matriceSysteme.factorLU();
    sysTriangInfUnite sysTIU = new sysTriangInfUnite( matriceSysteme,
      secondMembre);
    x = sysTIU.resolution();
    sysTriangSup sysTS = new sysTriangSup( matriceSysteme,
   x);
```

```
    x = sysTS.resolution();
}
catch (SysLinException e) { System.out.println(e); }
return x;
    }
    public vecteur resolutionPartielle() throws SysLinException {
sysTriangInfUnite sysTIU = new sysTriangInfUnite( matriceSysteme,
  secondMembre);
vecteur x = sysTIU.resolution();
sysTriangSup sysTS = new sysTriangSup( matriceSysteme,
       x);
try{ x = sysTS.resolution(); }
catch (SysLinException e) { System.out.println(e); }
return x;
    }
}
```

Exemple d'exécution avec fichier de données et traitement d'exception

Nous avons écrit un programme de test tout à fait similaire à celui qui a servit à tester la résolution d'un système triangulaire supérieur :

```
class testSysGeneLU{
    public static void main(String args[]){
bibMatFileIn f = new bibMatFileIn("donneeSGLU.dat");
matrice m = f.lectureMatrice();
vecteur b = f.lectureVecteur();
f.fermer();
System.out.println("        SYSTEME LU");
System.out.println("La matrice lue est : "); m.afficher();
System.out.println("Le sd membre lu est : "); b.afficher();
matrice mcopy = new matrice(m.nbLignes(), m.nbColonnes());
mcopy.recopier(m);
sysGeneLU sglu = new sysGeneLU(m, b);
try{
    vecteur x = sglu.resolution();
    System.out.println("La solution trouvee est : "); x.afficher();
    vecteur v = matrice.produit(mcopy, x);
    System.out.println("Verification - le produit de la matrice "+
       "par la solution vaut :");
    v.afficher();
```

```
}
catch (SysLinException e){
    System.out.println(e);
}
    }
}
```

Pour faire la vérification, on peut remarquer qu'il a été nécessaire de recopier la matrice d'origine dans une autre matrice, puisque la matrice d'origine se verra transformer lors de l'appel de sa factorisation sous la forme LU.

Il faut donc compléter la classe matrice de la manière suivante pour y ajouter la méthode `recopier` :

```
    public void recopier(matrice m)
    /** recopie la matrice m dans la matrice courante
      */
    {
for (int i=0; i<nbLignes(); i++)
    for (int j=0; j<nbColonnes(); j++)
toCoef(i, j, m.coef(i,j));
    }
```

Voici le fichier test qui a été utilisé :

```
5 5
1 2 3 4 5
2 5 1 6 7
2 6 1 8 6
5 6 4 7 7
2 5 5 2 2
5
5 4 3 2 1
```

Voici le résultat de l'affichage du programme :

```
java testSysGeneLU
        SYSTEME LU
La matrice lue est :
1.0 2.0 3.0 4.0 5.0
2.0 5.0 1.0 6.0 7.0
2.0 6.0 1.0 8.0 6.0
5.0 6.0 4.0 7.0 7.0
2.0 5.0 5.0 2.0 2.0
```

```
Le sd membre lu est :
5.0 4.0 3.0 2.0 1.0
La solution trouvee est :
-1.1396574440052718 -0.29776021080368853 0.5335968379446643
0.1159420289855072 0.9341238471673253
Verification - le produit de la matrice par la solution vaut :
5.0 3.9999999999999982 2.999999999999999 1.9999999999999947 1.0
```

Chapitre 4

Graphisme scientifique avec Java

4.1 Applets

Une applet est un programme qui est inclus dans une page HTML et qui va donc être exécuter par le navigateur lisant cette page, à condition qu'il possède les fonctionnalités pour le faire. Il doit donc contenir une machine virtuelle Java compressée (c'est le cas des navigateurs usuels qui contiennent aujourd'hui un noyau Java correspondant à un JDK 1.1).

La classe `java.applet.Applet` doit être la classe mère de toute applet incluse dans un document.

4.1.1 Un premier exemple

Le programme **HelloWorld.java** suivant définit une applet `HelloWorld` qui utilise un objet `Graphics` qui constitue un élément de base de la bibliothèque `awt` qui est repris avec plus de précision dans le paragraphe 4.2. Cette applet utilise la méthode `drawString` sur cet objet, ce qui lui permet d'afficher une chaîne de caractères à une position donnée sur la fenêtre courante du navigateur.

```
import java.applet.Applet ;
import java.awt.Graphics ;

public class HelloWorld extends Applet {
  public void paint(Graphics g) {
    g.drawString("Hello world", 50, 25);
  }
}
```

Il faut noter dès à présent que cette applet ne possède pas de point d'entrée `main()`, celui-ci se trouve gérer implicitement par le navigateur.

Voici maintenant un exemple de page HTML qui appelle l'applet :

```
<html>
  <head>
    <title>  Un exemple d'applet </title>
  </head>
  <body>
    <applet code="HelloWorld.class" width=150  height=25>
    </applet>
  </body>
</html>
```

En appelant cette page HTML dans un navigateur intégrant Java, on verra apparaître dans le navigateur le message "Hello world".

4.1.2 Passage de paramètres

La méthode suivante permet de récupérer des paramètres passés à une applet dans une page HTML :

```
public String getParameter(String name)
```

Cette méthode ne peut être appelée que dans les méthodes `init()` ou `start()` d'une applet (ces méthodes sont décrites plus loin).

Voici donc une nouvelle version `HelloWorld2.java`

```
import java.applet.Applet;
import java.awt.Graphics;

public class HelloWorld2 extends Applet {
  String e;
  public void init() { e=getParameter("message"); }
  public void paint(Graphics g) { g.drawString(e, 50, 25); }
}
```

Voici un exemple de page HTML qui appelle l'applet :

```
<html>
  <head>
    <title>  Un exemple d'applet avec paramètres</title>
  </head>
  <body>
    <applet code="HelloWord2.class" width=150  height=25>
    <param name="message" value="Bonjour">
    </applet>
  </body>
</html>
```

4.1.3 Cycle de vie

Le navigateur utilisé pour exécuter l'applet contrôle la vie et l'activation de l'applet grâce aux quatres méthodes suivantes :

- `public void init()` est appelée après le chargement de l'applet dans la page html ;
- `public void stop()` est appelée chaque fois que le navigateur arrêt l'exécution de l'applet, soit parce que l'utilisateur change de page web, soit parce qu'il iconifie le navigateur ;
- `public void start()` est appelée chaque fois que l'applet doit démarrer, après `init()` ou apres un `stop()` lorsque l'utilisateur revient sur la page web contenant l'applet ou lorsqu'il désiconifie le navigateur ;
- `public void destroy()` est appelée à la fermeture du navigateur. Elle détruit les ressources allouées pour l'applet.

4.1.4 Compléments

Nous donnons brièvement et de manière non exhaustive quelques points d'entrée pour les lecteurs désirant utiliser les aspects *multimédia* de Java mais que nous n'utiliserons pas dans le cadre des applications présentées dans cet ouvrage.

Des méthodes sont disponibles pour récupérer des images et des sons :

- `public Image getImage(URL url);`
- `public Image getImage(URL url, String name);`
- `public AudioClip getAudioClip(URL url);`
- `public AudioClip getAudioClip(URL url, String name);`

Des méthodes de la classe `java.applet.AudioClip` permettent de manipuler les sons ainsi récupérés :

- `public abstract void play();`
- `public abstract void loop();`
- `public abstract void stop()`.

Des méthodes sont également disponibles pour jouer directement les sons :

- `public void play(URL url);`
- `public void play(URL url, String name)`.

4.2 Gestion de fenêtres avec AWT

4.2.1 Tracés dans des applets

Nous décrivons comment effectuer des tracés géométriques dans la zone graphique d'une applet. Nous utilisons pour cela des méthodes de la classe

`java.awt.Graphics`. Elles sont résumées ci-dessous. (une description plus précise sera trouvé dans la documentation des API).

- `public void draw3DRect(int x, int y, int width, int height, boolean raised)` trace un rectangle en relief ;
- `public abstract void drawArc(int x, int y, int width, int height, int startAngle, int arcAngle)` trace un arc de cercle ;
- `public abstract void drawLine(int x1, int y1, int x2, int y2)` trace un segment de droite ;
- `public abstract void drawOval(int x, int y, int width, int heignt)` trace une ellipse ;
- `public abstract void drawPolygon(int xPoints[], int yPoints[], int nPoints)` trace un polygone ;
- `public void drawRect(int x, int y, int width, int height)` trace un rectangle vide ;
- `public abstract void drawRoundRect(int x, int y, int width, int height, int arcWidth, int arcHeight)` trace un rectangle vide à bords arrondis ;
- `public void fill3Drect(int x, int y, int width, int height, boolean raised)` trace un rectangle plein en relief ;
- `public abstract void fillArc(int x, int y, int width, int height, int startAngle, int arcAngle)` trace un arc de cercle plein ;
- `public abstract void fillOval(int x, int y, int width, int heignt)` trace une ellipse pleine ;
- `public abstract void fillPolygon(int xPoints[], int yPoints[], int nPoints)` trace un polygone plein ;
- `public abstract void fillRect(int x, int y, int width, int height)` trace un rectangle plein ;
- `public abstract void fillRoundRect(int x, int y, int width, int height, int arcWidth, int arcHeight)` trace un rectangle plein à bords arrondis.

Voici un exemple d'applet contenant des graphiques :

```
import java.awt.*;

public class Test extends java.applet.Applet {
  public void paint(Graphics g) {
    int i;
    g.setColor(Color.yellow);
    for (i= 0; i<14; i++)
      g.drawLine(10, 10+16*i, 10+16*i, 218);
```

```
    for (i= 0; i<14; i++)
      g.drawLine(10+ 16*i, 10, 218,  10+16*i);
    for (i= 0; i<14; i++)
      g.drawLine(10, 218-16*i, 10+16*i, 10);
    for (i= 0; i<14; i++)
      g.drawLine(10+16*i, 218, 218, 218-16*i);
  }
}
```

Pour terminer, nous signalons que, dans la suite de cet ouvrage, nous présenterons principalement des applications graphiques autonomes et non sous forme d'applets, car elles sont, de fait, plus générales. Les indications données précédemment sont suffisantes pour permettre une transformation assez facile de ces applications autonomes en applets.

4.2.2 Construire des interfaces fenêtrées

Une première fenêtre

Dans l'exemple qui suit, on construit une simple fenêtre d'une dimension initiale, avec un titre. La fenêtre construite dérive de la classe `Frame` qui permet de définir des fenêtres avec barre de titre, menu, bords, etc ...

On utilise `WindowListener` pour pouvoir gérer la fermeture de la fenêtre qui s'effectue lorsque l'utilisateur clique sur l'icône supérieure droite du bandeau de fenêtre : il faut alors décrire qui gère des évènements de ce type. Dans Java 1.1 et supérieur, cette gestion se fait par des "écouteurs" d'évènements (`listener`). Ici, pour un objet héritant de `WindowListener`, on se mettra "à l'écoute des évènements" grâce à la méthode `addWindowListener`.

`WindowListener` est une interface (c'est une classe abstraite particulière, sans variable d'instance et où toutes les méthodes sont abstraites : on représente ainsi un ensemble de comportements qui doivent être implémentés dans les classes qui implémentent cette interface). Elle contient un certain nombre de méthodes abstraites qu'il faut redéfinir. Ici, on redéfinit non trivialement la seule méthode `windowClosing` qui permettra de gérer la fermeture de la fenêtre.

```
import java.awt.*;
import java.awt.event.*;

public class Fenetre extends Frame implements WindowListener {
  public Fenetre() { setSize(400, 300); }
  public void windowClosing(WindowEvent event) {
    System.exit(0);
```

```
  }
  public void windowClosed(WindowEvent event) {}
  public void windowDeiconified(WindowEvent event) {}
  public void windowIconified(WindowEvent event) {}
  public void windowActivated(WindowEvent event) {}
  public void windowDeactivated(WindowEvent event) {}
  public void windowOpened(WindowEvent event) {}

  public static void main(String arg[]) {
    Fenetre Test = new Fenetre();
    Test.setTitle("ma fenetre JAVA");
    Test.show();
    Test.addWindowListener(Test);
  }
}
```

Dans la suite de ce chapitre(4.3.2), nous donnerons une alternative plus allégée à cette gestion d'évènements attachés à des fenêtres, en s'appuyant sur un exemple.

Les différentes parties d'une fenêtre

Une fenêtre gérée par l'AWT peut comorter plusieurs éléments caractéristiques :

- Une page de fond ou canevas, dans laquelle on pourra tracer des figures géométriques comme celles qui ont été décrites au paragraphe 4.2.1 ;
- Une étiquette (`Label`) qui permet d'afficher un texte ;
- Une zone de texte (`TextArea`) pour un affichage pouvant être modifié ;
- Une liste déroulante permettant de faire un choix (`List`) ;
- Une zone de saisie de texte (`TextField`) ;
- Un bouton (`Button`) ;
- Une case d'option (`Checkbox`) ;
- Un menu déroulant (`Menu` et `MenuBar`).

Un exemple : un compteur

Dans l'exemple commenté suivant, on met en œuvre une petite interface graphique incluant différents boutons et gérant des évènements.

```
import java.awt.* ;
import java.awt.event.* ;

class FenetreCompteur extends Frame {
```

```
int compteur ;

// définition de boutons
// en paramètre : le texte des boutons
Button boutonIncr = new Button("+");
Button boutonDecr = new Button("-");
Button boutonQuit = new Button("quit");

// un champ qui permettra d'afficher la valeur du compteur
// en paramètre : la taille des caractères
TextField affichageCompteur = new TextField(7);

//gestion des évènements provoqués
//par les clics sur les boutons
class ActionIncr implements ActionListener {
  public synchronized void actionPerformed(ActionEvent e)
  {  compteur ++; afficherCompteur(); }
};

class ActionDecr implements ActionListener {
  public synchronized void actionPerformed(ActionEvent e)
  {  compteur --; afficherCompteur(); }
};

class ActionQuit implements ActionListener {
  public synchronized void actionPerformed(ActionEvent e)
  {  System.exit(0); }
};

void afficherCompteur()
{  affichageCompteur.setText(String.valueOf(compteur)); }

// constructeur
public FenetreCompteur(String nom) {
  super(nom);
  compteur = 0;
  setSize(240, 80);
  setLayout(new FlowLayout());
  add(boutonIncr);
  add(boutonDecr);
  add(boutonQuit);
  add(affichageCompteur);
  boutonIncr.addActionListener( new ActionIncr());
```

```
    boutonDecr.addActionListener( new ActionDecr());
    boutonQuit.addActionListener( new ActionQuit());
  }
}

public class TestAWT {
  static public void main(String argv[]) {
    FenetreCompteur x = new FenetreCompteur("compteur");
    x.show();
 }
}
```

FIG. 4.1: Fenêtre générée par le programme TestAWT

Gestion des évènements

Sur l'exemple précédent, on a vu comment utiliser des évènements de type `ActionEvent` à partir des composants `Button` et `TextField`.

D'autres types d'évènements existent :

- `MouseEvent` pour les mouvements et cliquage de souris ;
- `FocusEvent` pour savoir si la souris est au-dessus de la zone considérée ;
- `KeyEvent` pour l'enfoncement d'une touche ;
- `TextEvent` pour la modification de texte pour un composant intégrant une zone de texte.

Il faut alors créer, relativement à un événement de type `xxxEvent`, une classe qui implémente `xxxListener` où l'on définira le traitement à faire lorsque l'événement a lieu (méthode `actionPerformed`). Dans la classe fenêtre, on lancera l'écoute des évènements avec `addListener`.

Placement des composants

Les différents composants d'une fenêtre peuvent être placés de plusieurs manières avec un gestionnaire de placements (`layout manager`) :

- `FlowLayout` range les composants ligne par ligne et de gauche à droite ;
- `BorderLayout` place les composants dans 5 zones : le centre et les 4 côtés ;
- `GridLayout` place les composants sur une grille 2D ;
- `GridBagLayout` place les composants sur une grille 2D, avec des coordonnées. Les zones n'ont pas toutes nécessairement la même dimension.

4.3 Construction de courbes de tracés scientifiques

Nous allons maintenant présenter quelques classes qui permettent d'effectuer des tracés simples de courbes scientifiques bidimensionnelles. Nous proposons une organisation en plusieurs classes par soucis de généricité et de réutilisabilité. Pour ne pas alourdir les programmes qui se veulent avant tout pédagogiques, nous présentons des constructions qui peuvent être grandement améliorées sur leurs aspects esthétiques.

4.3.1 Les domaines bidimensionnels de l'utilisateur et du dispositif d'affichage

Le graphisme scientifique nécessite de représenter des courbes dont les domaines de définition des coordonnées peuvent être très variables d'une utilisation à l'autre. On souhaite malgré cela afficher à l'écran ou sur un autre dispositif d'affichage des graphiques qui vont tous occupés une place a peu près similaire. Il faut donc définir :

- un domaine correspondant à la variation des coordonnées des différents points de la figure à représenter ;
- un domaine correspondant à l'espace utilisé sur le dispositif d'affichage.

Pour cela, nous définissons deux classes élémentaires qui vont respectivement représenter ces deux domaines :

```
class DomaineDouble {
    public double xmin, ymin, xmax, ymax;
    DomaineDouble(double x0, double y0, double x1, double y1) {
        xmin=x0; ymin=y0; xmax=x1; ymax=y1;
    }
}
class DomaineInt {
    public int xmin, ymin, xmax, ymax;
```

```
    DomaineInt(int x0, int y0, int x1, int y1) {
        xmin=x0; ymin=y0; xmax=x1; ymax=y1;
    }
}
```

Il faut aussi définir un processus de passage d'un domaine à l'autre, ce qui est décrit dans les paragraphes suivants.

4.3.2 Un gestionnaire de fenêtres de base

Nous décrivons d'abord une classe dérivée de la classe `Frame` qui va gérer un environnement fenêtré rudimentaire qui est constitué de deux composants situés l'un en-dessous de l'autre (c'est-à-dire dans les parties `North` et `Center` du gestionnaire de mise en page `BorderLayout`) :

- un `Canvas` où seront tracées la ou les courbes ;
- un `Label` qui donne un titre à la fonction tracée. cette information sera souvent récupérée d'une méthode appelée `libelle` que nous avons ajoutée à l'interface `FoncD2D`, déjà utilisé précédemment et qui s'écrit maintenant :

```
interface FoncD2D {
    public double calcul(double x);
    public String libelle();
}
```

Par ailleurs, nous ne faisons qu'une gestion minimale de la fenêtre que lancera notre programme final, en ne s'occupant que de sa fermeture, grâce à l'icône supérieure droite de la barre de titre. Nous avons choisit une méthode alternative à ce que nous avons présenté précédemment (4.2.2) : On utilise une classe dérivée de la classe prédéfinie `WindowAdapter`. Ceci nous permet de ne redéfinir explicitement que la méthode `windowClosing`, comme on le voit dans le listing qui suit :

```
import java.awt.*;
import java.awt.event.*;
import DomaineInt;

class MiniWinAdapter extends WindowAdapter {
    public void windowClosing(WindowEvent e) {System.exit(0);}
}

class FenetreGraphe extends Frame {
    FenetreGraphe(DomaineInt de, Canvas c, String texte) {
        setTitle(new String("graphe d'une fonction"));
        setSize((int)(de.xmax - de.xmin), (int)(de.ymax - de.ymin)+50);
        setLayout(new BorderLayout());
```

```
        addWindowListener(new MiniWinAdapter());
        add("North",c);
        add("Center", new Label(texte, Label.CENTER));
    }
}
```

4.3.3 Une classe d'utilitaires pour le tracé de graphismes scientifiques

Dans la classe qui suit, nous avons réunit un ensemble de procédures utiles pour faire des tracés de courbes :

- `mapX` et `mapY` s'occupent de faire le changement de coordonnées à partir du domaine que l'on doit représenter, vers le domaine du dispositif d'affichage et ceci, respectivement, pour chacune des coordonnées.
- `Line` et `Carre` effectuent les tracés géométriques élémentaires correspondant à leur dénomination.
- `coloreFond` colore le fond du `Canvas` d'affichage.
- `tracePoints` et `tracePolyline` affichent un ensemble de points dont les coordonnées sont données en paramètres. Pour la deuxième méthode, les points sont reliés par des segments.
- `traceAxes` affichent des axes de coordonnées passant par l'origine en affichant les valeurs des coordonnées des extrémités des axces. Ces axes ne sont effectivement affichés que s'ils sont situés dans la vue représentée qui est définie par le domaine à représenter. Pour cette raison, on définit l'alternative suivante.
- `traceAxesCentres` effectuent le même travail à la différence que les axes sont centrés par rapport au milieu du domaine d'affichage et ne passent donc pas, en général, par l'origine.
- `traceFonction` effectue le tracé graphique d'une fonction de type `FoncD2D`, passée en paramètre.

```
import java.awt.*;
import FoncD2D;
import DomaineDouble;
import DomaineInt;

class ManipGraphe {

    DomaineInt ce;     // coordonnees ecran
    DomaineDouble crv;  // coordonnees reelles visualisees
    DomaineDouble cr;
    double dx;         // pas sur x
```

```
ManipGraphe(DomaineDouble domReel, DomaineInt domEcran) {
    ce=domEcran;
    cr=domReel;
    // calcul d'une marge autour du domaine utilisateur
    double deltax = (domReel.xmax - domReel.xmin)/10;
    double deltay = (domReel.ymax - domReel.ymin)/10;
    crv = new DomaineDouble (domReel.xmin - deltax,
          domReel.ymin - deltay, domReel.xmax + deltax,
                                 domReel.ymax + deltay);
    dx = (cr.xmax - cr.xmin)/30;
}

public int mapX (double x) {
    return ce.xmin + (int) ((x-crv.xmin)/(crv.xmax-crv.xmin)*
                                        (ce.xmax-ce.xmin));
}

public int mapY (double y) {
    return ce.ymin + (int) ((crv.ymax-y)/(crv.ymax-crv.ymin)*
                                        (ce.ymax-ce.ymin));
}

public void Line(Graphics g, double x1, double y1,
                             double x2, double y2) {
    g.drawLine(mapX(x1), mapY(y1), mapX(x2), mapY(y2));
}

public void Carre(Graphics g, double x, double y) {
    g.drawRect(mapX(x)-2, mapY(y)-2,4,4);
}

public void coloreFond(Graphics g, Color c) {
    g.setColor(c);
    g.fillRect(ce.xmin, ce.ymin, ce.xmax-ce.xmin, ce.ymax-ce.ymin);
}

public void tracePoints(Graphics g, Color c, double[] x,
                                             double[] y) {
    g.setColor(c);
    for (int i=0; i<x.length; i++) Carre(g, x[i], y[i]);
}
```

```
    public void tracePolyLine(Graphics g, Color c, double[] x,
                                                   double[] y) {
        g.setColor(c);
        for (int i=1; i<x.length; i++)
            Line(g, x[i-1], y[i-1], x[i], y[i]);
    }

    public void traceAxes(Graphics g, Color c) {
        g.setColor(c);
        Line(g, 0, cr.ymin, 0, cr.ymax);
        Line(g, cr.xmin, 0, cr.xmax, 0);
        g.drawString(new String(String.valueOf(cr.ymin)),
                     mapX(dx/2), mapY(cr.ymin));
        g.drawString(new String(String.valueOf(cr.ymax)),
                     mapX(dx/2), mapY(cr.ymax)+10);
        g.drawString(new String(String.valueOf(cr.xmin)),
                     mapX(cr.xmin), mapY(dx/2));
        g.drawString(new String(String.valueOf(cr.xmax)),
                     mapX(cr.xmax)-30, mapY(dx/2));
    }

    public void traceAxesCentres(Graphics g, Color c) {
        g.setColor(c);
        double crymid = (cr.ymax + cr.ymin)/2;
        double crxmid = (cr.xmax + cr.xmin)/2;
        Line(g, crxmid, cr.ymin, crxmid, cr.ymax);
        Line(g, cr.xmin, crymid, cr.xmax, crymid);
        g.drawString(new String(String.valueOf(cr.ymin)),
                     mapX(crxmid+dx/2), mapY(cr.ymin));
        g.drawString(new String(String.valueOf(cr.ymax)),
                     mapX(crxmid+dx/2), mapY(cr.ymax)+10);
        g.drawString(new String(String.valueOf(cr.xmin)),
                     mapX(cr.xmin), mapY(crymid+dx/2));
        g.drawString(new String(String.valueOf(cr.xmax)),
                     mapX(cr.xmax)-30, mapY(crymid+dx/2));
    }

    public void traceFonction(Graphics g, Color c, FoncD2D f,
                              double mini, double maxi) {
        double x1, y1, x2, y2;
        g.setColor(c);
        x2=mini; y2 = f.calcul(x2);
        for (x1=mini; x1<= maxi; x1 += dx) {
```

```
            y1= f.calcul(x1);
            Line(g, x1, y1, x2, y2);
            x2=x1; y2=y1;
        }
    }
}
```

4.3.4 Un exemple d'utilisation

L'utilisateur final de cette bibliothèque graphique rudimentaire aura à écrire un programme similaire à celui qui est donné dans la suite. Il contient une classe implémentant l'interface `FoncD2D` car il effectue le tracé d'une fonction qui est transmise sous ce type à la bibliothèque graphique. Il contient également deux autres classes qui devront toujours apparaître lorsque l'on sera amené à utiliser la bibliothèque :

- une classe qui dérive de la classe prédéfinie `Canvas` qui représente la partie de fenêtre où doit être tracé le graphique. Cette classe possède un constructeur qui a pour paramètres les domaines - de l'utilisateur et du dispositif graphique - concernés et les composantes graphiques qui doivent être affichées, ici une fonction. Cette classe contient une méthode essentielle qui est la méthode `paint` : c'est ici que l'on doit décrire précisément tout ce qui devra être tracé au final, à l'exécution du programme.
- une classe contenant le programme principal et qui construit les domaines, le `Canvas` et la fenêtre d'affichage.

```
import java.awt.*;
import FoncD2D;
import ManipGraphe;
import FenetreGraphe;
import DomaineDouble;
import DomaineInt;

class CanvasGraphe extends Canvas {

    ManipGraphe mp = null;
    FoncD2D f;
    DomaineDouble cr;
    DomaineInt ce;

    CanvasGraphe(FoncD2D fonc, DomaineDouble domReel,
                               DomaineInt domEcran) {
        cr=domReel; ce=domEcran;f = fonc;
```

```
        setSize(new Dimension(ce.xmax-ce.xmin, ce.ymax-ce.ymin));
        mp = new ManipGraphe(domReel, domEcran);
    }

    public void paint(Graphics g) {
        mp.coloreFond(g,Color.white);
        mp.traceAxes(g,Color.black);
        mp.traceFonction(g, Color.blue, f, cr.xmin, cr.xmax);
        mp.Carre(g,0,0);
    }
}

class MaFonction implements FoncD2D {
    public double calcul(double x) {return x*x;}
    public String libelle() {return "f(x)=x*x";}
}

class GraphFonc {

    static public void main(String argv[]) {
        MaFonction fct = new MaFonction();
        DomaineDouble dr = new DomaineDouble(-2, -0.5, 2, 5);
        DomaineInt de = new DomaineInt(0, 0, 600, 450);
        CanvasGraphe cg = new CanvasGraphe(fct, dr, de);
        FenetreGraphe x = new FenetreGraphe(de, cg, fct.libelle());
        x.show();
    }
}
```

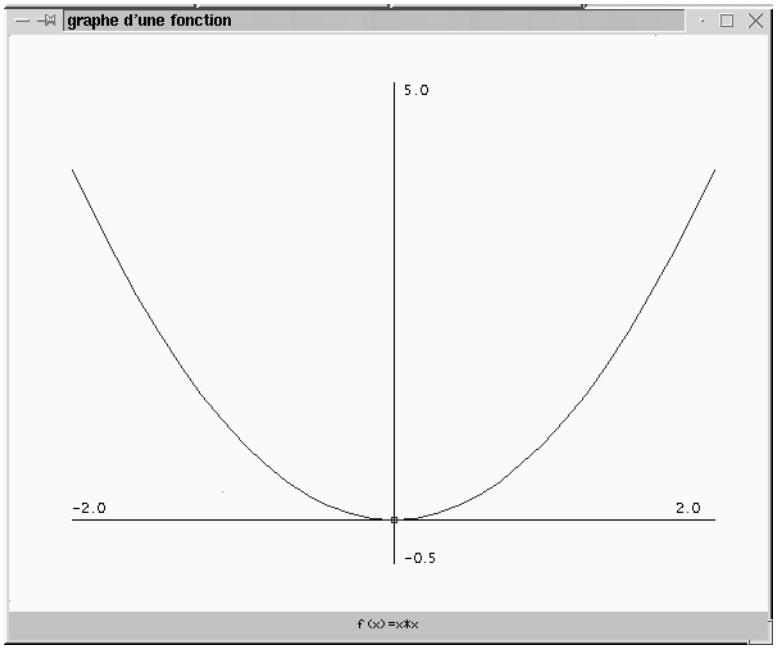

FIG. 4.2: Fenêtre générée par le programme de tracé de courbe donné en exemple

Chapitre 5

Interpolation Polynômiale

5.1 Introduction. Existence et unicité du polynôme d'interpolation

Soit f une application de $\mathbb{R}$ dans $\mathbb{R}$, dont on connait n+1 points $(x_i, f(x_i))$, pour $i = 0, \cdots, n$. Le but du problème d'interpolation est de déterminer une fonction g simple à calculer, telle que :

$$g(x_i) = f(x_i), \quad i = 0, \cdots, n$$

Les points $(x_i, f(x_i))$ sont appelés points d'interpolation ou d'appui.
Les fonctions g les plus couramment utilisées sont des polynômes des fractions rationnelles, des sommes d'exponentielles etc $\cdots$.
Dans ce volume on ne traitera que le cas où g est un polynôme.

Théorème 5 *Une condition nécessaire et suffisante pour qu'il existe un polynôme P_n unique interpolant f est que les points d'interpolation x_i soient tous distincts.*

5.2 Interpolation de Lagrange

Interpolation linéaire Connaissant deux points d'appui $(x_1, f(x_2))$ et $(x_2, f(x_2))$, on approche la valeur de $f(x)$, pour tout x dans $[x_1, x_2]$ par l'ordonnée du point de même abscisse x se trouvant sur la droite joignant les deux points d'appui.
Dans ce cas le polynôme d'interpolation est du premier degré et s'écrit :

$$P_1(x) = f(x_1)\frac{x - x_2}{x_1 - x_2} + f(x_2)\frac{x - x_1}{x_2 - x_1}$$

Interpolation quadratique Connaissant trois points d'appui $(x_1, f(x_2))$, $(x_2, f(x_2))$ et $(x_3, f(x_3))$, on approche la valeur de $f(x)$, pour tout x dans $[x_1, x_2]$ par l'ordonnée du point de même abscisse x se trouvant sur la parabole passant par les trois points d'appui.
Dans ce cas le polynôme d'interpolation est du second degré et s'écrit :

$$P_1(x) = f(x_1)\frac{(x-x_2)(x-x_3)}{(x_1-x_2)(x_1-x_3)} + f(x_2)\frac{(x-x_1)(x-x_3)}{(x_2-x_1)(x_2-x_3)} + f(x_3)\frac{(x-x_1)(x-x_2)}{(x_3-x_1)(x_3-x_2)}$$

Interpolation de Lagrange, cas général On appellera polynôme de Lagrange de degré n basés sur les points d'appui $(x_i, f(x_i))$, pour $i = 0, \cdots, n$ les n+1 polynômes de degré n :

$$L_k(x) = \prod_{\substack{i=0 \\ i \neq k}}^{n} \frac{x - x_i}{x_k - x_i} \qquad i = 0, \cdots, n$$

tels que $L_k(x_k) = 1$ et $L_k(x_i) = 0$ pour $i \neq k$. Le polynôme d'interpolation de Lagrange est donné par

$$P_n(x) = \sum_{k=0}^{n} L_k(x) f(x_k)$$

Remarque 6 *La formule ci-dessus nécéssite un grand nombre d'opérations pour sa mise en oeuvre informatique (au total $4n^2-3n$). En regroupant astucieusement les termes de $L_k(x)$, on peut économiser le nombre d'opérations, d'où, si on pose : $\gamma_{n+1}(x) = \prod_{j=0}^{n}(x - x_j)$ et $D_k(x) = (\prod_{j=0,k\neq j}^{n}(x_k - x_j))(x - x_k)$;*
alors $L_k(x) = \frac{\gamma_{n+1}(x)}{D_k(x)}$ et

$$P_n(x) = \gamma_{n+1}(x) \sum_{k=0}^{n} \frac{f(x_k)}{D_k(x)}$$

Théorème 6 (erreur de l'interpolation de Lagrange) *Soit $f \in C^{n+1}[a, b]$, et soit $P_n(x)$ le polynôme d'interpolation de f sur les points $(x_i, f(x_i))$, pour $i = 0, \cdots, n$. Pour tout $x \in [a, b]$, il existe $\xi_x \in]min(x, x_i), max(x, x_i)[$ tel que l'erreur $f(x) - P_n(x)$ soit*

$$E(x) = \frac{\gamma_{n+1}(x)}{(n+1)!} f^{(n+1)}(\xi_x)$$

Si on pose $M_{n+1} = max_{a \leq x \leq b} |f(n+1)(x)|$, on alors $E(x) \leq \frac{|\gamma_{n+1}(x)|}{(n+1)!} M_{n+1}$.

5.3 Interpolation d'Hermite

En plus de la valeur de f aux n+1 abscisses x_i on fixe la valeur de sa derivee f' en ces mêmes abscisses, on cherche donc un polynome P tel que :

$$(*) \quad \begin{cases} P(x_i) = f(x_i) \\ P'(x_i) = f'(x_i), \quad i = 0, \cdots, n \end{cases}$$

Théorème 7 *Si les points d'appui x_i sont tous distincts, alors il existe un polynôme P et un seul de degré au plus 2n+1 tel que les égalites (*) soient verifiées. Ce polynôme P s'écrit*

$$P(x) = \sum_{i=0}^{n} H_i(x) f(x_i) + \sum_{i=0}^{n} K_i(x) f'(x_i)$$

où

$$\begin{cases} H_i(x) = [1 - 2(x - x_i)L_i'(x_i)]L_i^2(x) \\ K_i(x) = (x - x_i)L_i^2(x) \\ \textit{avec} \quad L_i(x) = \prod_{\substack{j=0, \\ j \neq i}}^{n} \frac{x - x_j}{x_i - x_j} \end{cases}$$

Théorème 8 (erreur de l'interpolation d'Hermite) *Soit $f \in C^{2n+2}[a, b]$, et soit $P_n(x)$ le polynôme d'interpolation de Hermite de f sur les points $(x_i, f(x_i))$, pour $i = 0, \cdots, n$. Pour tout $x \in [a, b]$, il existe $\xi_x \in]min(x, x_i), max(x, x_i)[$ tel que l'erreur $f(x) - P_n(x)$ est :*

$$E(x) = \frac{(\gamma_{n+1}(x))^2}{(2n+2)!} f^{(2n+2)}(\xi_x)$$

où $\gamma_{n+1}(x) = \prod_{j=0}^{n} (x - x_j)$.

5.4 Enoncés des Exercices Corrigés

Exercice 31 :
Déterminer le polynôme d'interpolation de Lagrange satisfaisant au tableau ci-dessous

x	0	2	3	5
$f(x)$	-1	2	9	87

Exercice 32 :
Soit $f(x) = \frac{1}{1+x^2}$.
Déterminer le polynôme d'interpolation de Lagrange pour les points d'appui d'abscisses : -2, -1, 0, 1, 2 .
Discuter l'erreur d'interpolation.

Exercice 33 :
Avec quelle précision peut-on calculer $\sqrt{115}$ à l'aide de l'interpolation de Lagrange, si on prend les points : $x_0 = 100, x_1 = 121, x_2 = 144$.

Exercice 34 :
Soit $f(x) = \ln x$; estimer la valeur de $\ln(0.60)$ avec :

x	0.40	0.50	0.70	0.80
$f(x)$	-0.916291	-0.6993147	-0.356675	-0.223144

Comparer avec la valeur 'exacte' obtenue grâce à votre calculatrice.

Exercice 35 :
a) Reécrire la formule d'interpolation de Lagrange dans le cas où les points d'appui sont équidistants.
b) Utiliser le même tableau qu'à l'exercice précédent complété par la 'vraie' valeur de $\ln 0.60$ et estimer la valeur de $\ln 0.54$.

Exercice 36 :
a) Utiliser la formule d'interpolation de Lagrange pour trouver la cubique passant par 0.4, 0.5, 0.7, 0.8 pour $f(x) = \sin x$.
b) Même question pour $f(x) = \frac{1}{\tan x}$.

Exercice 37 :
Soit $f(x) = \sqrt{2+x}$.

1. Determiner le polynôme $P(x)$ de Lagrange basé sur les points d'abscisses 0, 1 et 2.
2. Calculer $P(0.1)$ et $P(0.9)$, et comparer aux valeurs 'exactes'. Evaluer l'erreur d'interpolation en ces deux points. Commentaires ?

Exercice 38 :
Soit $f(x) = \frac{x^2-1}{x^2+1}$.

1. Determiner le polynôme $P(x)$ de Lagrange basé sur les points d'abscisses -2, 0 et 2.
2. Calculer $P(1)$, et comparer à la valeur exacte. Evaluer l'erreur d'interpolation en ce point. Commentaires ?

Exercice 39 :
Calculer le polynôme d'Hermite Q tel que :
Q(0)=f(0), Q'(0) =f'(0), Q(5)=f(5), Q'(5)=f'(5), pour $f(x) = \frac{1}{1+x^2}$.
En déduire la valeur de Q(4), comparer f(4) à Q(4).

5.5 Enoncés des exercices non corrigés

Exercice 40 :
Démontrer le théorème 5, qui donne condition nécessaire et suffisante pour qu'il existe un polynôme P_n unique interpolant une fonction f, à savoir que les points d'interpolation x_i soient tous distincts.

Exercice 41 :
Soit $f(x) = \sqrt{2+x}$.

1. Determiner le polynôme $P(x)$ de Lagrange basé sur les points d'abscisses 0, 1 et 2.
2. Calculer $P(0.1)$ et $P(0.9)$, et comparer aux valeurs 'exactes'. Evaluer l'erreur d'interpolation en ces deux points. Commentaires ?

Exercice 42 :
On se donne les trois points d'interpolation suivants : $x_0 = 9$, $x_1 = 16$, $x_2 = 25$. Avec quelle précision peut-on calculer $\sqrt{17}$ à l'aide de l'interpolation de Lagrange appliquée à une fonction f que vous donnerez ?

Exercice 43 :
Soit $f(x) = \frac{1}{1+x}$.

1. Determiner le polynôme $P(x)$ de Lagrange basé sur les points d'abscisses 0, 1 et 2.
2. Calculer $P(\frac{1}{2})$, et comparer à la valeur exacte. Evaluer l'erreur d'interpolation en ce point. Commentaires ?

Exercice 44 :
Approcher $\ln(529.62)$ par interpolation de Lagrange, sachant que $\ln(529) = 6.270988$ et $\ln(530) = 6.272877$. Donner une majoration de l'erreur théorique de cette interpolation et la comparer à l'erreur effectivement commise (utiliser une dalculatrice pour avoir la valeur 'exacte' de $\ln(529.62)$.

5.6 Corrigés des exercices

exercice 22

Rappelons que le polynôme de Lagrange basé sur les points d'appui d'abscisses $x_0, x_1, ..., x_N$ est d'ordre N et s'écrit :

$$P_n(x) = \sum_{k=0}^{N} f(x_k) L_k(x),$$

avec

$$L_k(x) = \prod_{j=0, j\neq k}^{N} \frac{x - x_j}{x_k - xj}.$$

ici les points d'appui donnés par :

$$\begin{array}{rclrcl} x_0 & = & 0 & f(x_0) & = & -1 \\ x_1 & = & 2 & f(x_1) & = & 2 \\ x_2 & = & 3 & f(x_2) & = & 9 \\ x_3 & = & 5 & f(x_3) & = & 87, \end{array}$$

détermineront donc un polynôme de Lagrange d'ordre 3, celui-ci s'écrit :

$$P_3(x) = \sum_{k=0}^{3} f(x_k) L_k(x),$$

avec

$$\begin{aligned} L_0(x) &= \frac{(x - x_1)(x - x_2)(x - x_3)}{(x_0 - x_1)(x_0 - x_2)(x_0 - x_3)} \\ &= \frac{(x - 2)(x - 3)(x - 5)}{(0 - 2)(0 - 3)(0 - 5)} \\ &= -\frac{1}{30}(x - 2)(x - 3)(x - 5) \end{aligned}$$

$$\begin{aligned} L_1(x) &= \frac{(x - x_0)(x - x_2)(x - x_3)}{(x_1 - x_0)(x_1 - x_2)(x_1 - x_3)} \\ &= \frac{(x - 0)(x - 3)(x - 5)}{(2 - 0)(2 - 3)(2 - 5)} \\ &= \frac{1}{6}x(x - 3)(x - 5) \end{aligned}$$

$$\begin{aligned} L_2(x) &= \frac{(x-x_0)(x-x_1)(x-x_3)}{(x_2-x_0)(x_2-x_1)(x_2-x_3)} \\ &= \frac{(x-0)(x-2)(x-5)}{(3-0)(3-2)(3-5)} \\ &= -\frac{1}{6}x(x-2)(x-5) \end{aligned}$$

$$\begin{aligned} L_3(x) &= \frac{(x-x_0)(x-x_1)(x-x_2)}{(x_3-x_0)(x_3-x_1)(x_3-x_2)} \\ &= \frac{1}{30}x(x-2)(x-3) \end{aligned}$$

Finalement,

$$\begin{aligned} P_3(x) &= f(x_0)L_0(x) + f(x_1)L_1(x) + f(x_2)L_2(x) + f(x_3)L_3(x) \\ &= \frac{53}{30}x^3 - 7x^2 + \frac{253}{30}x - 1. \end{aligned}$$

exercice 23

Soit $f(x) = \dfrac{1}{1+x^2}$. Les points d'appui sont :

$$\begin{array}{rclrcl} x_0 &=& -2 & f(x_0) &=& \frac{1}{5} \\ x_1 &=& -1 & f(x_1) &=& \frac{1}{2} \\ x_2 &=& 0 & f(x_2) &=& 1 \\ x_3 &=& 1 & f(x_3) &=& \frac{1}{2} \\ x_4 &=& 2 & f(x_4) &=& \frac{1}{5}. \end{array}$$

Le polynôme de Lagrange est donc d'ordre 4. Il s'écrit

$$P_4(x) = \sum_{k=0}^{4} f(x_k)L_k(x)$$

avec

$$\begin{aligned}
L_0(x) &= \frac{1}{24}x(x+1)(x-1)(x-2)\\
L_1(x) &= -\frac{1}{8}x(x+2)(x-1)(x-2)\\
L_2(x) &= \frac{1}{4}(x+2)(x+1)(x-1)(x-2)\\
L_3(x) &= -\frac{1}{6}x(x+2)(x+1)(x+2)\\
L_4(x) &= \frac{1}{24}x(x+2)(x+1)(x-1).
\end{aligned}$$

Finalement,

$$\begin{aligned}
P_4(x) &= f(x_0)L_0(x) + f(x_1)L_1(x) + f(x_2)L_2(x) + f(x_3)L_3(x) + f(x_4)L_4(x)\\
&= \frac{1}{120}x(x+1)(x-1)(x-2) - \frac{1}{12}x(x+2)(x-1)(x-2)\\
&+ \frac{1}{4}(x+2)(x+1)(x-1)(x-2) - \frac{1}{12}x(x+2)(x+1)(x+2)\\
&+ \frac{1}{120}x(x+2)(x+1)(x-1)\\
&= \frac{1}{10}x^4 - \frac{3}{5}x^2 + 1.
\end{aligned}$$

Calculons l'erreur théorique sur cette interpolation. celle-ci est donnée ou point x par :

$$E(x) = f(x) - P_n(x) = \gamma_{n+1}(x) - \frac{1}{(n+1)!}f^{(x+1)}(\xi_x),$$

où, $\xi \in (\min x_i, \max x_i) = I$. Elle vérife,

$$|E(x)| \leq |\gamma_{n+1}(x)|\frac{1}{(n+1)!}M_{n+1}$$

où

$$\begin{aligned}
\gamma_{n+1}(x) &= \prod_{k=0}^{n}(x - x_k)\\
M_{n+1} &= \max_{t\in I}|f^{n+1}(t)|
\end{aligned}$$

Comme ici on a 5 points d'appui, cette erreur est majorée par :

$$|E(x)| \leq |\gamma_5(x)|\frac{1}{5!}M_5.$$

On a clairement $\gamma_5(x) = \prod_{k=0}^{5}(x-x_k) = x(x^2-1)(x^2-4)$. Il reste à calculer $M_5 = \max_{t\in I}|f^5(t)|$. Un calcul assez long donne : $f^{(5)}(x) = \dfrac{-240x(3-10x^2+3x^4)}{(1+x^2)^6}$
(Ceci peut être par exemple obtenu en décomposant f de la facon suivante, $f(x) = \frac{1}{1+x^2} = \frac{1}{2}.\frac{1}{1+ix} + \frac{1}{2}.\frac{1}{1-ix}$. On calcule alors la dérivée d'ordre p $(\frac{1}{1+\alpha x})^{(p)}$, ce qui est plus simple,

$$\begin{aligned}\left(\frac{1}{1+\alpha x}\right)' &= \frac{-\alpha}{(1+\alpha x)^2}\left(\frac{1}{1+\alpha x}\right)'' \\ &= \frac{(-1)^2 2\alpha^2}{(1+\alpha x)^3}\end{aligned}$$

$$\vdots$$

$$\left(\frac{1}{1+\alpha x}\right)^{(p)} = \frac{(-1)^P \alpha^p p!}{(1+\alpha x)^{p+1}}$$

d'où

$$\begin{aligned}f^{(5)}(x) &= \frac{1}{2}\left(\frac{1}{1+ix}\right)^{(5)} + \frac{1}{2}\left(\frac{1}{1-ix}\right)^{(5)} \\ &= \frac{1}{2}.\frac{-120i}{(1+ix)^6} + \frac{1}{2}.\frac{120i}{(1-ix)^6} \\ &= \frac{-240x(3-10x^2+3x^4)}{(1+x^2)^6}\end{aligned}$$

de même, on trouve $f^{(6)}(x) = \frac{-240}{(1+x^2)^7}\left[-21x^6+105x^3-63x^2+3\right]$.)

Ainsi l'étude de $f^{(5)}$ donne $M_5 = 100$, (pour trouver les extremas de $f^{(5)}$, c'est à dire les racines de l'équation $frac-240(1+x^2)^7\left[-21x^6+105x^3-63x^2+3\right] = 0$, on a recours au chapitre 2 sur la résolution des équations non linéaire dans $\mathbb{R}$).

Finalement,

$$|E(x)| \leq |\gamma_5(x)|\frac{1}{(5)!}M_5 = |x(x^2-1)(x^2-4)|\frac{100}{5!},$$

$$|E(x)| \leq |\gamma_5(x)|\frac{1}{(5)!}M_5 = |x(x^2-1)(x^2-4)|\frac{5}{6}.$$

Application à $x = \frac{1}{2}$. On a $f(\frac{1}{2}) = 0.8$, $P(\frac{1}{2}) = -0.86$, donc l'erreur effective est $E_e(\frac{1}{2}) = |f(\frac{1}{2}) - P(\frac{1}{2})| = 0.06$. Or

$$|E(\frac{1}{2})| \leq |\gamma_5(\frac{1}{2})|\frac{1}{(5)!}M_5 = 0.29.$$

Par consquent $|E_e(\frac{1}{2})| < |E(\frac{1}{2})|$, l'interpolation de Lagrange, dans ce cas, donne une bonne approximation de $f(\frac{1}{2})$.

exercice 25

Soit $f(x) = \ln x$ Estimons la valeur de $\ln(0.6)$ grâce à une interpolation de Lagrange basée sur les points d'appui donés dans l'exercice. Le polynôme de Lagrange s'écrit

$$P_3(x) = \sum_{k=0}^{3} f(x) L'_k x.$$

(Il est de dégré 3 car on a 4 points d'appui), avec

$$L_k(x) = \prod_{j=0, j\neq k}^{3} \frac{x - x_j}{x_k - x_j}.$$

On cherche donc

$$P_3(0,6) = \sum_{k=0}^{3} f(x_k) L_k(0,6).$$

Calculons tout d'abord les nombres $L_k(0,6)$, on a

$$\begin{aligned} L_0(0,6) &= \frac{(x-x_1)(x-x_2)(x-x_3)}{(x_0-x_1)(x_0-x_2)(x_0-x_3)} \\ &= \frac{(0,1)(-0,1)(-0,2)}{(-0,1)(-0,3)(-0,4)} \\ &= -\frac{1}{6} \end{aligned}$$

De même, on trouve :

$$\begin{aligned} L_1(0,6) &= \frac{2}{3}, \\ L_2(0,6) &= \frac{2}{3}, \\ L_3(0,6) &= -\frac{1}{6}. \end{aligned}$$

Ainsi, $P_3(0,6) = f(0,4)L_0(0,6) + f(0,5)L_1(0,6) + f(0,7)L_2(0,6) + f(0,8)L_3(0,6)$ soit $P_3(0,6) = -0,509975$.

Estimation de l'erreur théorique dans ce cas. Elle est majorée par

$$|E(x)| \leq \frac{\gamma_{n+1}(x)}{(n+1)!} M_{n+1}$$

où $M_{n+1} = \max |f^{(n+1)}(x)|$, donc

$$|E(x)| \leq |(x-x_0)(x-x_1)(x-x_2)(x-x_3)| \frac{1}{4!} M_4.$$

Cherchons M_4. On a $f(x) = \ln(x)$, $f'(x) = \frac{1}{x}$, $f''(x) = -\frac{1}{x^2}$. Donc $f^{(3)}(x) = \frac{2}{x^3}$ et $f^{(4)}(x) = -\frac{6}{x^4}$. Ainsi, $M_4 = |f^{(4)}(0,4)| = \frac{6}{(0,4)^4} = \frac{6}{256} \times 10^4$.

Donc,

$$E(x) \leq |(x-0.4)(x-0.5)(x-0.7)(x-0.8)| \frac{1}{24} \frac{6}{256} 10^4.$$

Ainsi, $E(0,6) \leq \frac{1}{256}$. En fait, on a $-\frac{1}{256} < E(0.6) < -\frac{1}{4096}$. Or, la valeur exacte de $\ln(0.6) = -0.510826$. L'erreur effectivement commise lors de l'approximation de $\ln(0.6)$ par $P_3(0.6)$ est donc $|P_3(0.6) - (-0.510826)| = 8.51 \times 10^{-4} < \frac{1}{256}$. Donc, dans ce cas, l'interpolation de Lagrange donne un bon résultat.

exercice 26

a) On cherche la formule de Lagrange dans le cas où les $N+1$ points sont équidistants.
Dans ce cas on a $\forall i \in \{0, 1, 2, ..., N\}$ $x_{i+1} - x_i = h$, (où $h = \frac{x_N - x_0}{N}$), d'où $x_i = x_0 + ih$. D'autre part, pour tout $x \in x_0, x_N$, il existe un réel s tel que

$x = x_0 + sh$. Par conséquent,

$$\begin{aligned}
L_k(x) = L_k(x_0 + sh) = \ell(s) &= \prod_{j=0, j\neq k}^{n} \frac{(x - x_j)}{x_k - x_j} \\
&= \prod_{j=0, j\neq k}^{n} \frac{(x_0 + sh) - (x_0 + jh)}{(x_0 + kh) - (x_0 + jh)} \\
&= \prod_{j=0, j\neq k}^{n} \frac{sh - jh}{kh - jh} \\
&= \prod_{j=0, j\neq k}^{n} \frac{s - j}{k - j} \\
&= \frac{s}{k}.\frac{s-1}{k-1}.\frac{s-2}{k-2}\cdot\cdot\cdot\cdot\cdot\cdot\frac{s-n}{k-n} \quad \text{avec } j \neq k \\
&= \frac{s(s-1)...(s-k+1)(s-k-1)...(s-n)}{k!(-1)^{n-k}(n-k)!}
\end{aligned}$$

b) On a $x_0 = 0.4$, $x_1 = 0.5$, $x_2 = 0.6$, $x_3 = 0.7$, $x_4 = 0.8$, donc cinq points équidistants. Comme $x = 0.54 = x_0 + sh = 0.5 + s \times 0.1$, alors, $s = 1.4$. Calculons les $L_k(x)$ pour $k = 0, 1, 2, 3, 4$.

$$\begin{aligned}
\ell_0(.54) &= \frac{(-1)^{n-0}}{0!(n-0)!}(s-1)(s-2)(s-3)(s-4) \\
&= \frac{(-1)}{4!}(s-1)(s-2)(s-3)(s-4) \\
&= \frac{1}{24}(0.4)(-0.6)(-1.6)(-2.6) \\
&= -0.0416 \\
\ell_1(0.54) &= \frac{(-1)^{n-1}}{1!(n-1)!}s(s-2)(s-3)(s-4) \\
&= \frac{-1}{3!}(1.4)(1.4-2)(1.4-3)(1.4-4) \\
&= 0.5824.
\end{aligned}$$

On trouve de même

$$\begin{aligned}
\ell_2(0.54) &= 0.5824 \\
\ell_3(0.54) &= -0.1456 \\
\ell_4(0.54) &= 0.0224.
\end{aligned}$$

Finalement, $P(0.54) = \sum_{k=0}^{4} f(x_k)L_k(x) = \sum_{k=0}^{4} f(x_k)\ell_k(s) = -0.616142$. Alors que la valeur exacte est -0.616186.

exercice 29

Soit $f(x) = \frac{1}{1+x^2}$. Le polynôme d'Hermite Q_n s'écrit,

$$Q_n(x) = \sum_{k=0}^{n}(x)f(x_k) + \sum_{k=0}^{n} K_k(x)f'(x_k)$$

avec

$$\begin{aligned} H_k(x) &= \big(1 - 2(x - x_k)L'_k(x_k)\big)L_k^2(x) \\ K_k(x) &= (x - x_k)L_k^2(x). \end{aligned}$$

Calculons les polynômes L_k, H_k et K_k, sachant que les abscisses des points d'appui sont $x_0 = 0$ et $x_1 = 5$.

$$\begin{aligned} L_0(x) &= \frac{x - x_1}{x_0 - x_1} = 1 - \frac{x}{5} \\ L_1(x) &= \frac{x - x_0}{x_1 - x_0} = \frac{x}{5} \\ et\ \ L'_0(x) &= -\frac{1}{5} \\ L'_1(x) &= \frac{1}{5} \end{aligned}$$

$$\begin{aligned} H_0(x) &= \big(1 - 2(x - x_0)L'_0(x)\big)L_0^2(x) \\ &= \left(1 - 2(x - 0)\left(-\frac{1}{5}\right)\right)\left(1 - \frac{x}{5}\right)^2 \\ &= \frac{2}{125}x^3 + \frac{3}{25}x^2 + 1 \end{aligned}$$

$$\begin{aligned} et\ \ H_1(x) &= \big(1 - 2(x - x_1)L'_1(x)\big)L_1^2(x) \\ &= \left(1 - 2(x - 5)\left(\frac{1}{5}\right)\right)\left(\frac{x}{5}\right)^2 \\ &= -\frac{2}{125}x^3 + \frac{3}{25}x^2 \end{aligned}$$

D'autre part,

$$K_0(x) = (x - x_0)L_0^2(x) = (x - 0)\left(1 - \frac{x}{5}\right)^2 = \frac{x^3}{25} - \frac{2}{5}x^2 + x$$

$$K_1(x) = (x - x_1)L_1^2(x) = (x - 5)\left(\frac{x}{5}\right)^2 = \frac{x^3}{25} - \frac{x^2}{5}.$$

Ainsi,

$$\begin{aligned} Q_3(x) &= \sum_{k=0} H_k(x)f(x_k) + \sum_{k=0} K_k(x)f'(x_k) \\ &= H_0(x)f(x_0) + H_1 f(x_1) + K_0(x)f'(x_0) + K_1(x)f'(x_1) \\ &= (\frac{2}{125}x^3 - \frac{3}{25}x^2 + 1) + (-\frac{2}{125}x^3 + \frac{3}{25}x^2)(\frac{1}{26}) + (\frac{1}{25}x^3 - \frac{1}{5}x^2)(-\frac{10}{26^2}) \\ &= \frac{10}{262}x^3 - \frac{76}{262}x^2 + 1 \end{aligned}$$

Ainsi on a $Q_3(4) \approx 0.147$ et pour comparaison $f(4) \approx 0.0588$.

L'écart peut sembler important, mais un calcul de l'erreur théorique sur l'interpolation d'Hermite montre que l'erreur effectivement commise est plus petite que cette erreur théorique, (laissé en exercice). En outre, la valeur $x = 4$ est relativement proche du bord de l'intervalle d'interpolation qui est $[0, 5]$, et en général, pour l'interpolation de Lagrange ainsi que celle d'Hermite, l'erreur s'amplifie au bord de l'intervalle d'interpolation (ce phénomène est dit de Runge), voir la partie travux pratiques.

5.7 Mise en œuvre en Java

On présente dans la suite une mise en œuvre d'un calcul d'interpolation polynômiale, basé sur la formule de Lagrange décrite dans le paragraphe 5.2, sous sa forme énoncée dans la remarque 5 de ce paragraphe.

Soient (x_i, y_i) pour $0 \leq i \leq n - 1$, les n points d'appui du calcul d'interpolation. Le calcul de l'interpolé en x s'écrit :

$$P_{n-1}(x) = \gamma_n(x) \sum_{k=0}^{n-1} \left(\frac{y_k}{D'_k}\right) \frac{1}{x - x_k}$$

avec

$$\begin{aligned} D'_k &= \prod_{j=0, j\neq k}^{n-1} (x_k - x_j) \\ \gamma_n(x) &= \prod_{j=0}^{n} (x - x_j) \end{aligned}$$

Pour effectuer ce calcul, on utilise un fichier pour récupérer les coordonnées des points de support. On écrit pour cela une classe `TableauFileIn`, similaire à celle écrite dans le paragraphe 3.4.4 et qui effectue ce travail. Les deux tableaux des abscisses et des ordonnées sont lus l'un après l'autre. Cette classe correspond à :

```
import java.io.*;
import java.util.*;

class TableauFileIn {
    FileReader fichier=null;
    BufferedReader lecbuf;
    String line;

    public TableauFileIn(String nomfichier) {
        try{
            fichier = new FileReader (nomfichier);
            lecbuf = new BufferedReader (fichier);
        }
        catch (FileNotFoundException e) {
            System.out.println("fichier inexistant !");
        }
    }

    public double[] lectureTableau() {
        String line, word;
        double [] tlu = null;
        StringTokenizer st;
        try {
            // lecture de la taille du tableau sur la ligne 1 :
            line = lecbuf.readLine();
            int nb = Integer.parseInt(line);
            tlu = new double[nb];
            // lecture des coef. du tableau sur 1 seule ligne :
            line = lecbuf.readLine();
```

```
            st = new StringTokenizer(line);
            for (int j=0; j<nb; j++) {
                word = st.nextToken();
                tlu[j] = Double.valueOf(word).doubleValue();
            }
        }
        catch (IOException e) {
            System.out.println("erreur de lecture");
        }
        catch (NumberFormatException e) {
            System.out.println("erreur de conversion chaine vers double");
        }
        return tlu;
    }

    public void fermer() {
        try { fichier.close(); }
        catch (IOException e) {
            System.out.println("Le fichier ne peut pas être fermer");
        }
    }
}
```

On présente ensuite la classe `InterPoly` qui permet de représenter une interpolation polynômiale dont le constructeur se chargera de mémoriser les tableaux des coordonnées des points d'appui qui lui sont transmis en paramètres. Ce constructeur commence également le calcul des coefficients qui ne dépendent pas de l'abscisse à laquelle se fera la calcul d'interpolation : il s'agit des coefficients $\left(\frac{y_k}{D'_k}\right)$. Une méthode privée se charge du calcul de γ_n et finalement la méthode publique `interP` effectue le calcul effectif de l'interpolation.

```
import java.lang.*;
import java.io.*;
import java.util.*;
import TableauFileIn;

class InterPoly {
    int n;         // nombre de points d'interpolation
    double [] xk;  // abscisses des points d'interpolation
    double [] yk;  // ordonnes des points d'interpolation
    double [] ydp; // coefficients de la formule d'interpolation
```

```
    static double epsilon = 1e-10; // epsilon-machine

    InterPoly(double [] xdata, double [] ydata) {
        n = xdata.length;
        xk = xdata; yk = ydata;
        ydp = new double[n];
        calCoef();
    }

    // calcul de gamma_n(x)
    private double gamma(double x) {
        double prod = x-xk[0];
        for (int i=1; i<n; i++) { prod *= x-xk[i]; }
        return prod;
    }

    private void calCoef() {
        double prod;
        int k,j;
        for (k=0; k<n; k++) {
            prod=1;
            for (j=0;   j<k; j++) { prod *= xk[k]-xk[j]; }
            for (j=k+1; j<n; j++) { prod *= xk[k]-xk[j]; }
            ydp[k] = yk[k]/prod;
        }
    }

    public double interP(double x) {
        double diff;
        diff = x- xk[0];
        if (Math.abs(diff) < epsilon) return yk[0];
        double som = ydp[0]/diff;
        for (int k=1; k<n; k++) {
            diff = x-xk[k];
            if (Math.abs(diff) < epsilon) return yk[k];
            som += ydp[k]/diff;
        }
        return som*gamma(x);
    }
}
```

Nous présentons ensuite une classe qui contient un programme de test qui lit un fichier contenant les coordonnées des points d'appui et qui affiche dans une fenêtre un graphique qui contient les points de supports (sous la forme de

petits carrés) et une courbe de la fonction d'interpolation construite à partir de ces points. Ces tracés graphiques utilisent la bibliothèque décrite dans la chapitre précédent et sont décrits dans la méthode `paint` de la classe `CanvasGraphe` dont le constructeur initialise les composants graphiques qui seront utilisés et qui sont transmis en paramètres.

Le domaine de représentation de ce graphique est obtenu en recherchant les coordonnées extrèmes des points de support. Ceci se fait grâce à la classe `MaxminTabDouble` construite à cet effet et décrite dans le listing qui suit :

```
import java.lang.*;
import java.io.*;
import java.util.*;
import java.awt.*;
import TableauFileIn;
import InterPoly;
import DomaineDouble;
import DomaineInt;
import ManipGraphe;
import FenetreGraphe;

class CanvasGraphe extends Canvas {
    ManipGraphe mp = null;
    DomaineDouble cr;
    DomaineInt ce;
    FoncD2D f;
    double[]xdata;
    double[]ydata;

    CanvasGraphe(DomaineDouble domReel, DomaineInt domEcran,
                 double[] x, double[] y, FoncD2D fonc) {
        cr = domReel; ce = domEcran;
        xdata = x; ydata = y; f = fonc;
        setSize(new Dimension(ce.xmax-ce.xmin, ce.ymax-ce.ymin));
        mp = new ManipGraphe (domReel, domEcran);
    }

    public void paint(Graphics g) {
        mp.coloreFond(g, Color.white);
        mp.traceAxesCentres(g, Color.black);
        mp.tracePoints(g, Color.red, xdata, ydata);
        mp.traceFonction(g, Color.blue, f, cr.xmin, cr.xmax);
```

```
    }
}

class MaxminTabDouble {
    double maxi, mini;

    MaxminTabDouble(double[] t) {
        maxi=t[0]; mini=t[0];
        for (int i=1; i<t.length; i++) {
            if (t[i] < mini) mini=t[i];
            if (t[i] > maxi) maxi=t[i];
        }
    }
    public double getmaxi() {return maxi;}
    public double getmini() {return mini;}
}


class PrgInterPolyGra {
    public static void main(String args[]) {
        // lecture du fichier de données
        TableauFileIn f = new TableauFileIn("donnees.dat");
        double[] xlu = f.lectureTableau();
        double[] ylu = f.lectureTableau();
        f.fermer();

        // calcul des extrema des tableaux :
        MaxminTabDouble mxlu = new MaxminTabDouble(xlu);
        double maxxlu = mxlu.getmaxi();
        double minxlu = mxlu.getmini();
        MaxminTabDouble mylu = new MaxminTabDouble(ylu);
        double maxylu = mylu.getmaxi();
        double minylu = mylu.getmini();

        // construction de l'interpolation polynomiale :
        InterPoly pp = new InterPoly(xlu, ylu);

        // representation graphique du calcul :
        DomaineDouble dr = new
               DomaineDouble(minxlu, minylu, maxxlu, maxylu);
        DomaineInt de = new DomaineInt(0, 0, 600, 450);
        CanvasGraphe cg = new CanvasGraphe(dr, de, xlu, ylu, pp);
        FenetreGraphe x = new FenetreGraphe(de, cg, pp.libelle());
```

```
        x.show();
    }
}
```

Nous montrons les résultats obtenus pour deux exemples :

Premier exemple

Le fichier d'entrée qui a été utilisé pour les coordonnées des points d'appui est composé de 4 points et permet d'obtenir un résultat prévisible et satisfaisant. Il correspond aux données suivantes :

```
4
0.4 0.5 0.7 0.8
4
-0.91 -0.69 -0.85 -0.22
```

La fenêtre graphique obtenue comme résultat de ce programme, à partir de ces données, correspond à la figure 5.1.

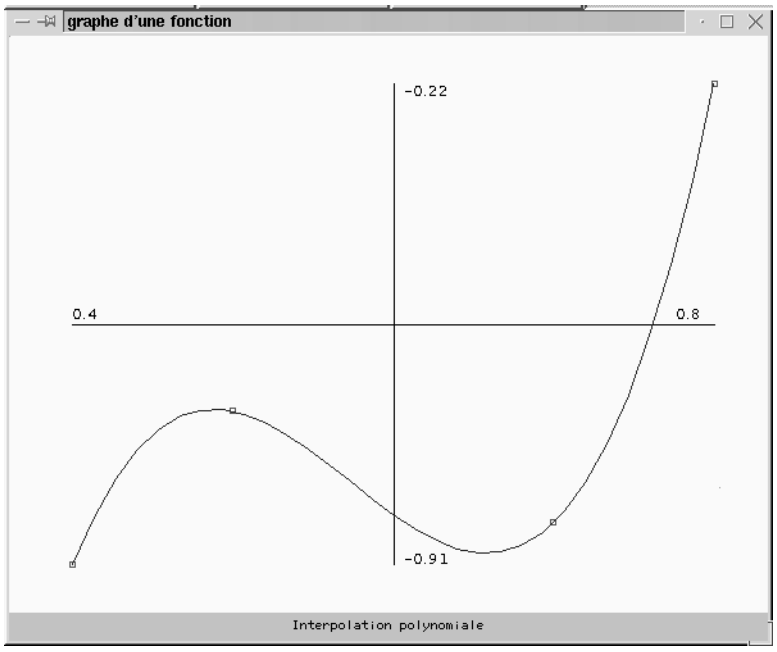

FIG. 5.1: Fenêtre générée par 1er exemple de calcul d'interpolation polynômiale

Deuxième exemple

Nous prenons maintenant l'exemple du à Runge et qui consiste à calculer une interpolation polynômiale à partir de points d'appui extraits de la courbe suivante :

$$f(x) = \frac{1}{1+x^2}$$

Runge a montré que l'erreur commise entre cette fonction et son interpolation polynômiale tendait vers l'infini lorsque le degré d'interpolation tendait lui-aussi vers l'infini.

Le fichier de données utilisé est alors le suivant :

```
11
-5 -4 -3 -2 -1 0 1 2 3 4 5
11
0.038 0.059 0.1 0.2 0.5 1 0.5 0.2 0.1 0.059 0.038
```

Comme le laisse prévoir la propriété énoncée précédemment, la courbe d'interpolation va présenter des oscillations, ces dernières se situent pour les valeurs d'abscisses ayant les plus grandes valeurs absolues. C'est effectivement ce que donne la représentation graphique obtenue par le programme et qui est représentée sur la figure 5.2.

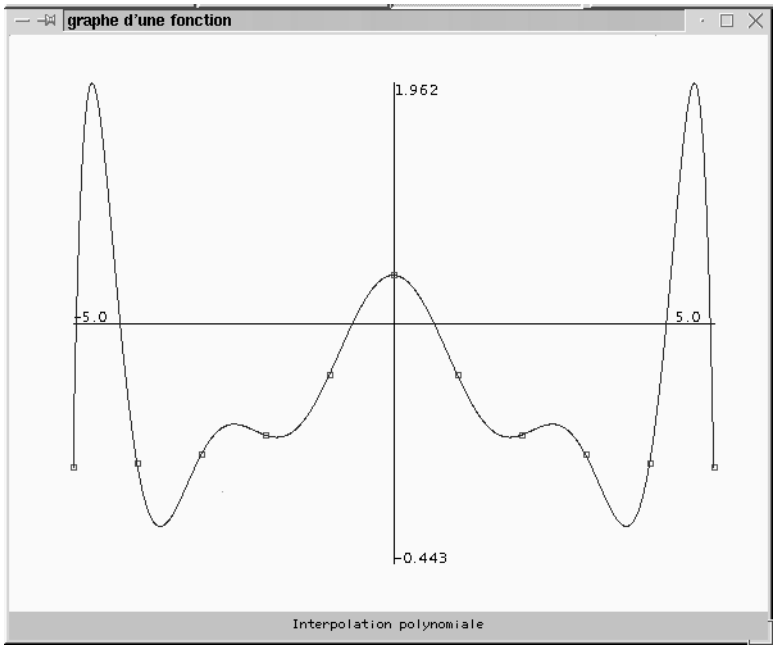

FIG. 5.2: Exemple de calcul d'interpolation polynômiale correspondant au phénomène de Runge

Chapitre 6

Approximation par moindres carrés

6.1 Principe d'approximation et critère de moindres carrés

On considère les n+1 points (x_i, c_i), pour $i = 0, ..., n$, de $\mathbb{R}^2$. Dans le chapitre précédent, on s'est intéressé à la recherche et au calcul de la fonction d'interpolation qui passe exactement par ces points. Dans ce chapitre, on cherche à construire une courbe qui, dans un sens à préciser, s'approche des points (x_i, c_i), ces points pouvant provenir de mesures, par exemple, et donc contenir des erreurs expérimentales.

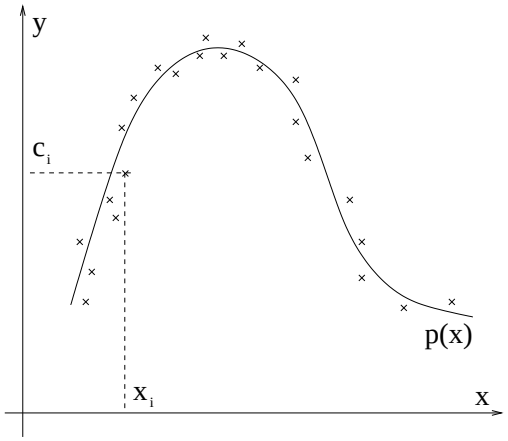

FIG. 6.1: Approximation de données par une fonction p(x)

On recherche donc une fonction d'approximation, encore appelée *modèle* et notée $p(x)$ qui va répondre à cette question. Ce modèle dépend de paramètres :

par exemple, s'il correspond à une fonction polynôme d'un degré donné, les paramètres sont les coefficients du polynôme.

Le problème que l'on se pose consiste donc à rechercher le meilleur jeux de paramètres de façon à ce que la courbe représentative du modèle passe *"au plus prêt"* des points de données.

La notion de meilleure proximité retenue ici est le **critère des moindres carrés**. Il consiste à rechercher le minimum de la fonction :

$$\sum_{i=0}^{n} (p(x_i) - c_i)^2$$

Le minimum est réalisé pour des valeurs particulières des paramètres qui correspondent donc à l'ajustement du modèle par rapport aux points de données.

6.2 Régression linéaire

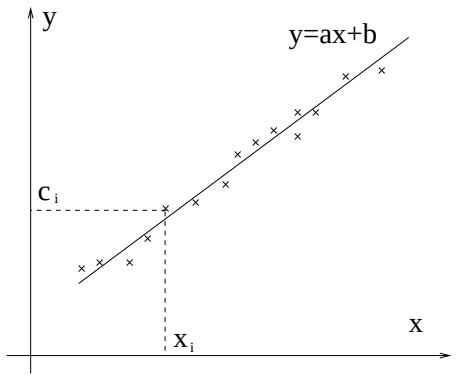

FIG. 6.2: Régression linéaire

On recherche la droite d'équation $y = ax + b$ la plus proche de l'ensemble des points (x_i, c_i), $i = 0, ..., n$, au sens des moindres carrés. Si on note $e_i = c_i - (ax_i + b)$ l'écart entre la droite de régression et le point considéré, on cherche le minimum de

$$Q(a, b) = \sum_{i=0}^{n} e_i^2 = \sum_{i=0}^{n} (c_i - ax_i - b)^2$$

afin d'obtenir les meilleures valeurs des paramètres a et b.

Le minimum est obtenue si

$$\frac{\partial Q}{\partial a} = 0 \quad et \quad \frac{\partial Q}{\partial b} = 0$$

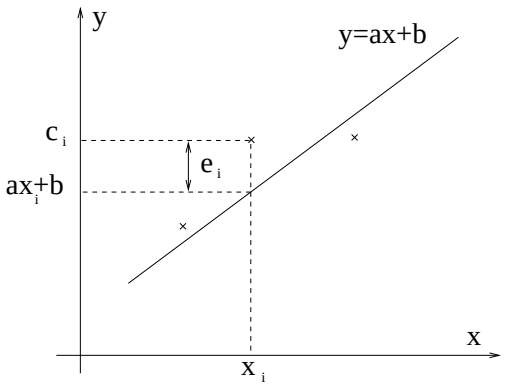

FIG. 6.3: Ecart entre un point et une droite de régression

Il faut donc résoudre le système linéaire suivant d'ordre 2 :

$$\begin{cases} \dfrac{\partial Q}{\partial a} = -2\displaystyle\sum_{i=0}^{n} x_i(c_i - ax_i - b) = 0 \\ \dfrac{\partial Q}{\partial b} = -2\displaystyle\sum_{i=0}^{n} (c_i - ax_i - b) = 0 \end{cases}$$

C'est à dire

$$\begin{cases} a\displaystyle\sum_{i=0}^{n} x_i^2 + b\sum_{i=0}^{n} x_i = \sum_{i=0}^{n} x_i c_i \\ a\displaystyle\sum_{i=0}^{n} x_i + (n+1)b = \sum_{i=0}^{n} c_i \end{cases}$$

de la dernière équation, on en déduit que

$$b = \bar{c} - a\bar{x}$$

où l'on note $\bar{x}$, la moyenne des x_i, à savoir $\dfrac{1}{n+1}\displaystyle\sum_{i=0}^{n} x_i$.

En substituant dans la première équation, on obtient

$$a\sum_{i=0}^{n} x_i^2 + (\bar{c} - a\bar{x})\sum_{i=0}^{n} x_i = \sum_{i=0}^{n} x_i c_i$$

En utilisant la variance

$$V_x = \frac{1}{n+1}\left(\sum_{i=0}^{n}(x_i - \bar{x})^2\right) = \frac{1}{n+1}\sum_{i=0}^{n} x_i^2 - \bar{x}^2$$

on obtient :

$$a(n+1)V_x = \sum_{i=0}^{n} x_i c_i - (n+1)\bar{x}\bar{c}$$

Et donc la valeur de a vaut :

$$a = \frac{\sum_{i=0}^{n} x_i c_i - (n+1)\bar{x}\bar{c}}{(n+1)V_x}$$

Il reste alors à déterminer la pertinence du calcul de régression linéaire par rapport aux données. Pour cela, on définit le *coefficient de corrélation linéaire* :

$$R = \frac{\sum_{i=0}^{n} x_i c_i - (n+1)\bar{x}\bar{c}}{(n+1)\sqrt{V_x V_c}}$$

- On montre que $-1 \leq R \leq 1$
- Si $|R|$ est voisin de 0 alors il n'y a pas de liaison linéaire entre les 2 coordonnées des points de données et le calcul de régression linéaire n'est pas représentatif de ces données ;
- Si $|R|$ est voisin de 1 alors les points de données sont proches d'un alignement et le calcul de régression se justifie.

6.2.1 Mise en œuvre en Java

Nous présentons maintenant une mise en œuvre du calcul de régression linéaire en utilisant les formules précédentes. Nous calculons les coefficients de la droite de régression et le coefficient de corrélation à partir du calcul des différentes sommes intervenant dans les formules.

La classe `Reglin` effectue ces calculs dans la méthode `calculCoef`. Elle possède un constructeur qui a deux paramètres correspondants aux deux tableaux contenant les coordonnées des points d'appui. Ces deux tableaux sont mémorisés dans des données membres. Le calcul des coefficients est alors lancé en fin de construction. On accède aux résultats, stockés dans des données membres, grâce à des opérateurs d'accès du type `getxxx`.

```
import java.lang.*;
import FoncD2D;

class Reglin implements FoncD2D{
    int n;        // nombre de points de support
    double[] x;   // abscisses des points de support
    double[] y;   // ordonnees des points de support
    double a, b;  // resultats de la regression : ax+b
    double r;     // coef. de correlation

    Reglin(double[] xdata, double[] ydata) {
        x = xdata; y = ydata; n = x.length;
        calculCoef();
    }

    private void calculCoef() {
        double sx=0, sx2=0, sy=0, sy2=0, sxy=0;
        for (int i=0; i<n; i++) {
            sx += x[i]; sx2 += x[i]*x[i];
            sy += y[i]; sy2 += y[i]*y[i];
            sxy += x[i]*y[i];
        }
        double num = sxy  - sx * sy /n;
        double denx = sx2 - sx * sx /n;
        double deny = sy2 - sy * sy /n;
        a = num/denx;
        r = num/Math.sqrt(denx*deny);
        b = (sy - a*sx) / n;
    }

    public double geta() {return a;}
    public double getb() {return b;}
    public double getr() {return r;}

    public double calcul(double z) {return a*z+b;}
    public String libelle() {return "regression lineaire";}
}
```

Nous présentons maintenant un programme utilisant la classe `Reglin` pour calculer la droite de régression à partir de données stockées dans un fichier. Ce programme est, en fait, très similaire au programme d'exemple du chapitre précédent sur l'interpolation polynômiale. On utilise la même classe `CanvasGraphe` qui n'est pas réécrite ci-dessous.

```
import java.lang.*;
import java.io.*;
import java.util.*;
import java.awt.*;
import TableauFileIn;
import Reglin;
import DomaineDouble;
import DomaineInt;
import ManipGraphe;
import FenetreGraphe;
import MaxminTabDouble;

class PrgReglin {
    public static void main(String args[]) {
        // lecture du fichier de données
        TableauFileIn f = new TableauFileIn("donnees.dat");
        double[] xlu = f.lectureTableau();
        double[] ylu = f.lectureTableau();
        f.fermer();

        // calcul des extrema des tableaux :
        MaxminTabDouble mxlu = new MaxminTabDouble(xlu);
        double maxxlu = mxlu.getmaxi();
        double minxlu = mxlu.getmini();
        MaxminTabDouble mylu = new MaxminTabDouble(ylu);
        double maxylu = mylu.getmaxi();
        double minylu = mylu.getmini();

        // construction de l'interpolation polynomiale :
        Reglin rl = new Reglin(xlu, ylu);
        System.out.println("Resultats du calcul de regression lineaire :");
        System.out.println("Fonction de regression : "
                           +rl.geta()+ " x + " +rl.getb());
        System.out.println("coefficient de correlation : "+rl.getr());

        // representation graphique du calcul :
        DomaineDouble dr = new DomaineDouble(minxlu, minylu, maxxlu, maxylu
        DomaineInt de = new DomaineInt(0, 0, 600, 450);
        CanvasGraphe cg = new CanvasGraphe(dr, de, xlu, ylu, rl);
        FenetreGraphe x = new FenetreGraphe(de, cg, rl.libelle());
        x.show();
    }
```

```
}
```

Ce programme est testé à partir des données du fichier “`donnees.dat` ” suivant :

```
13
0.8 4.9 5.1 6.7 27 34 38 65 72 85 95 97 98
13
0.8 1.6 3.1 6.0 15 26 41 66 78 86 93 96 96
```

Le programme renvoie des résultats sous forme texte qui sont les suivants :

```
java PrgReglin

Resultats du calcul de regression lineaire :
Fonction de regression : 1.0312158159925882 x + -3.047626180872437
coefficient de correlation : 0.9939141140402419
```

Le programme lance également l’ouverture de la fenêtre graphique suivante représentée par la figure 6.4.

6.3 Généralisation aux modèles linéaires

Nous allons étendre le calcul de régression linéaire à un modèle plus général du type :

$$p(x) = \sum_{j=0}^{m} \alpha_j \phi_j(x)$$

à partir des données $(x_i, c_i), 0 \leq i \leq n$

Les fonction ϕ_j sont connues alors que les coefficients α_j sont les paramètres à ajuster par rapport aux données.

Un tel modèle est appelé *linéaire* en raison de son expression qui est effectivement linéaire par rapport aux paramètres α_i.

On retrouve le modèle de régression linéaire en posant $m = 1$, $\phi_0(x) = 1$ et $\phi_1(x) = x$. L’expression du modèle est donc $p(x) = c_0 + c_1 x$.

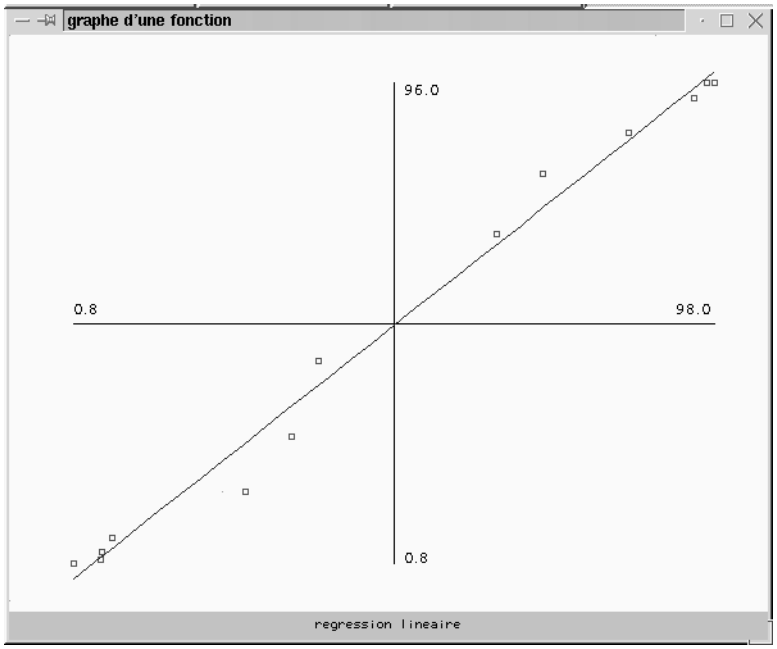

FIG. 6.4: Fenêtre générée par le programme de test du calcul de régression linéaire

Comme pour le calcul de régresxion linéaire, on applique le critère des moindres carrés qui consiste à rechercher le minimum de la fonction

$$\begin{aligned} S(\alpha_0, \alpha_1, \ldots, \alpha_m) &= \sum_{i=0}^{n} (c_i - p(x_i))^2 \\ &= \sum_{i=0}^{n} \left(c_i - \sum_{j=0}^{m} \alpha_j \phi_j(x_i) \right)^2 \end{aligned}$$

Ce minimum est atteind lorsque les dérivées partielles de S par rapport à chaque coefficient α_k, $k = 0 \ldots m$, sont nulles :

$$\begin{aligned}
\frac{\partial S(\alpha_0,\ldots,\alpha_m)}{\partial \alpha_k} &= 0 \\
2\sum_{i=0}^{n}\left(c_i - \sum_{j=0}^{m}\alpha_j\phi_j(x_i)\right)(-\phi_k(x_i)) &= 0 \\
\sum_{i=0}^{n}\sum_{j=0}^{m}\alpha_j\phi_j(x_i)\phi_k(x_i) &= \sum_{i=0}^{n} c_i\phi_k(x_i) \\
\sum_{j=0}^{m}\alpha_j\sum_{i=0}^{n}\phi_k(x_i)\phi_j(x_i) &= \sum_{i=0}^{n}\phi_k(x_i)c_i
\end{aligned}$$

C'est une équation linéaire à $m+1$ inconnues : α_j, $j = 0,\ldots,m$ qui se décline pour chaque valeur de k, $k = 0,\ldots,m$. On doit donc résoudre un système d'ordre $m+1$.

6.3.1 Ecriture matricielle du problème

On pose F matrice d'ordre $(n+1, m+1)$ dont les coefficients valent :

$$F_{ij} = \phi_j(x_i)$$

et on pose Y le vecteur de composantes $(c_0,\ldots,c_n)$.
On peut montrer que le système précédent s'écrit :

$$\left(F^tF\right)X = \left(F^tY\right)$$

La solution X correspond aux $(m+1)$ coefficients α_i cherchés. En effet

$$\begin{aligned}
\left(F^tY\right)_k &= \sum_{i=0}^{n}\left(F^t\right)_{ki}Y_i = \sum_{i=0}^{n}\phi_k(x_i)c_i \\
\left(\left(F^tF\right)X\right)_k &= \sum_{j=0}^{m}\left(F^tF\right)_{kj}X_j = \sum_{j=0}^{m}\left(\sum_{i=0}^{n}\phi_k(x_i)\phi_j(x_i)\right)\alpha_j
\end{aligned}$$

On a donc bien, pour $0 \le k \le m$:

$$\left(\left(F^tF\right)X\right)_k = (F^tY)_k$$

6.3.2 Mise en œuvre en Java

Nous présentons maintenant une mise en œuvre de ce calcul matriciel permettant une identification d'un modèle linéaire, à partir d'un ensemble de points d'appui. Nous utilisons les différentes classes construites dans les chapitres précédents, à savoir :

- `vecteur` : la classe de vecteurs définit en 1.3.4 ;
- `matrice` : la classe de matrices définit en 3.4.2 ;
- `sysGeneLU` : la classe de systèmes linéaires généraux définit en 3.4.5. Nous utilisons aussi une classe d'exception `SysLinException` définie dans le même chapitre et appelée lorsque la résolution d'un système linéaire échoue (matrice numériquement singulière, par exemple).

Le constructeur de la classe `Modlin` qui suit se charge uniquement de récupérer le tableau de fonctions passés en paramètres et l'affecte à une donnée propre du même type. La méthode `identifie` se charge de faire le calcul d'identification sous forme matriciel comme décrit précédemment : les matrices et vecteurs nécessaires sont construits et on appelle la méthode de résolution de système de la classe `sysGeneLU`. Une méthode `getcoef` permet de renvoyer un des coefficients identifiés. On définit aussi les deux méthodes de l'interface `FoncD2D` que notre classe implémente. Ces méthodes seront utilisées, par exemple, pour permettre une représentation graphique grâce à la bibliothèque définie au chapitre 4.

```
import vecteur;
import matrice;
import sysGeneLU;
import SysLinException;
import FoncD2D;

class Modlin implements FoncD2D{
    int m;          // nombre de fonctions de base du modele
    vecteur coef;   // coef. du modele a identifier
    FoncD2D [] phi; // fonctions de base

    Modlin(FoncD2D [] foncbase) {
        m = foncbase.length;
        phi = foncbase;
    }

    public void identifie(double[] xdata, double[] ydata)
             throws SysLinException {
        // 1. construction des matrices et vecteurs :
```

```
        vecteur y = new vecteur(ydata);
        int n = ydata.length;
        matrice f = new matrice(n,m);
        matrice ft = new matrice(m,n);
        for (int i=0; i<n; i++)
            for (int j=0; j<m; j++) {
                double coefij = phi[j].calcul(xdata[i]);
                f.toCoef(i, j, coefij);
                ft.toCoef(j, i, coefij);
            }
        // 2. Construction et résolution du systeme lineaire :
        matrice a = matrice.produit(ft,f);
        vecteur b = matrice.produit(ft,y);
        sysGeneLU syst = new sysGeneLU(a,b);
        coef = syst.resolution();
    }

    public double calcul(double x) {
        double result=0;
        for(int i=0; i<m; i++)
            result += coef.elt(i) * phi[i].calcul(x);
        return result;
    }

    public String libelle() {return "modele lineaire de moindres carres";}

    public double getcoef(int i) {return coef.elt(i);}
}
```

Le programe suivant donne un exemple d'utilisation de cette classe. Il a pour but de trouver une approximation par un polynôme de degré 3 d'un ensemble de points lus dans un fichier. Il est en grande partie similaire à celui du paragraphe précédent, sauf pour la construction d'une instance de la classe `Modlin`. L'appel à la méthode d'identification qui suit se fait en gérant l'exception éventuellement déclanchée si la résolution du système échoue. On remarquera le processus de construction des différentes fonctions de base du modèle et du tableau qui les contient.

```
import java.lang.*;
import java.io.*;
import java.util.*;
import java.awt.*;
```

```
import TableauFileIn;
import Modlin;
import DomaineDouble;
import DomaineInt;
import ManipGraphe;
import FenetreGraphe;
import MaxminTabDouble;

class Fbase1 implements FoncD2D{
    public double calcul(double x) {return 1;}
    public String libelle() {return "f(x)=1";}
}

class Fbase2 implements FoncD2D{
    public double calcul(double x) {return x;}
    public String libelle() {return "f(x)=x";}
}

class Fbase3 implements FoncD2D{
    public double calcul(double x) {return x*x;}
    public String libelle() {return "f(x)=xˆ2";}
}

class Fbase4 implements FoncD2D{
    public double calcul(double x) {return x*x*x;}
    public String libelle() {return "f(x)=xˆ3";}
}

class PrgModlinPoly {
    public static void main(String args[]) {
        // lecture du fichier de données
        TableauFileIn f = new TableauFileIn("donnees.dat");
        double[] xlu = f.lectureTableau();
        double[] ylu = f.lectureTableau();
        f.fermer();

        // calcul des extrema des tableaux :
        MaxminTabDouble mxlu = new MaxminTabDouble(xlu);
        double maxxlu = mxlu.getmaxi();
        double minxlu = mxlu.getmini();
        MaxminTabDouble mylu = new MaxminTabDouble(ylu);
        double maxylu = mylu.getmaxi();
        double minylu = mylu.getmini();
```

```
        // construction de l'interpolation polynomiale :
        Fbase1 f1 = new Fbase1();
        Fbase2 f2 = new Fbase2();
        Fbase3 f3 = new Fbase3();
        Fbase4 f4 = new Fbase4();
        FoncD2D [] fbase = {f1, f2, f3, f4};
        Modlin ml = new Modlin(fbase);
        try {
            ml.identifie(xlu, ylu);
            System.out.println("Resultats de l'identifiaction des coef. :")
            for (int i=0; i<fbase.length; i++)
                System.out.println("coef. "+i+" : "+ml.getcoef(i));

            // representation graphique du calcul :
            DomaineDouble dr = new DomaineDouble(minxlu, minylu,
                                                 maxxlu, maxylu);
            DomaineInt de = new DomaineInt(0, 0, 600, 450);
            CanvasGraphe cg = new CanvasGraphe(dr, de, xlu, ylu, ml);
            FenetreGraphe x = new FenetreGraphe(de, cg, ml.libelle());
            x.show();
        }
        catch(SysLinException e) {
            System.out.println("erreur au cours de l'identification");
        }
    }
}
```

Le programme précédent est testé sur le jeu de données qui suit :

```
13
-6 -5 -4 -3 -2 -1 0 1 2 3 4 5 6
13
-12 -13 -9 -7 -9 0 5 13 11 6 9 5 0
```

Le programme renvoie les résultats sous forme texte qui suivent :

```
java PrgModlinPoly

Resultats de l'identifiaction des coef. :
coef. 0 : 4.160839160839161
coef. 1 : 3.772560772560772
```

```
coef. 2 : -0.3026973026973027
coef. 3 : -0.07925407925407925
```

Il provoque également l'ouverture de la fenêtre qui est décrite dans la figure 6.5.

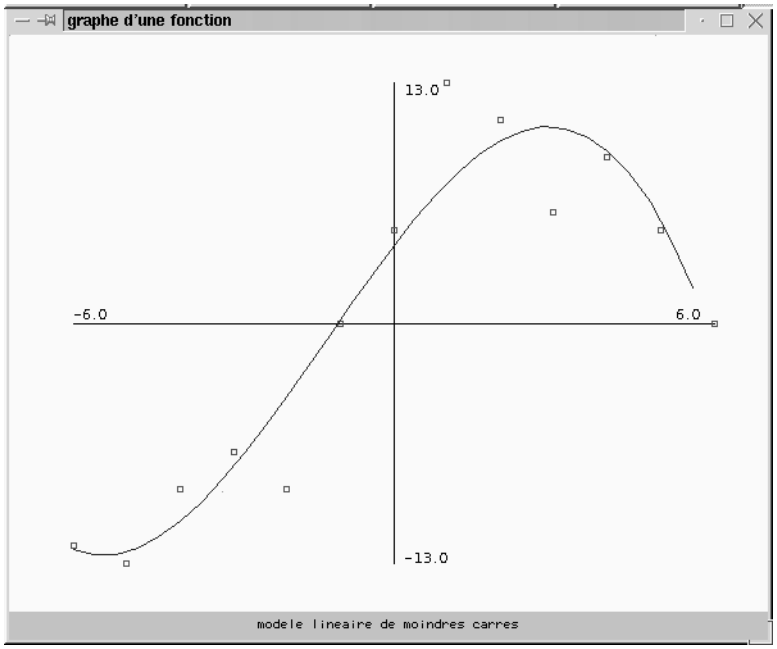

FIG. 6.5: Fenêtre générée par le programme de calcul d'un modèle linéaire en utilisant un polynôme de degré 3

Le résultat obtenue reste assez éloigné des points de support car le degré du polynôme est relativement peu élevé par rapport aux variations des points de données. On montre ci-dessous comment il est très simple de construire un nouveau modèle, ici un polynôme de degré 9, et de l'intégrer au programme de test. Toutes les classes utilisées sont suffisamment génériques pour ne pas avoir besoin d'être modifiée. Le programme de test devra définir toutes les classes qui vont permettre de construire les fonctions de base nécessaires, appelées ici `Fbasexxx`. Il ne restera plus qu'à les instancier effectivement et utiliser ces instances pour construire le tableaux des fonctions de base. Tout le reste reste inchangé. Voici ci-dessous la partie de programme à modifier :

```
...
```

```
class Fbase1 implements FoncD2D{
    public double calcul(double x) {return 1;}
    public String libelle() {return "f(x)=1";}
}

class Fbase2 implements FoncD2D{
    public double calcul(double x) {return x;}
    public String libelle() {return "f(x)=x";}
}

class Fbase3 implements FoncD2D{
    public double calcul(double x) {return x*x;}
    public String libelle() {return "f(x)=x^2";}
}

...

class Fbase9 implements FoncD2D{
    public double calcul(double x) {return x*x*x*x*x*x*x*x*x;}
    public String libelle() {return "f(x)=x^8";}
}

class Fbase10 implements FoncD2D{
    public double calcul(double x) {return x*x*x*x*x*x*x*x*x*x;}
    public String libelle() {return "f(x)=x^9";}
}

class PrgModlinPoly {
    public static void main(String args[]) {
        // lecture du fichier de données
        TableauFileIn f = new TableauFileIn("donnees.dat");
        double[] xlu = f.lectureTableau();
        double[] ylu = f.lectureTableau();
        f.fermer();

        // calcul des extrema des tableaux :
        MaxminTabDouble mxlu = new MaxminTabDouble(xlu);
        double maxxlu = mxlu.getmaxi();
        double minxlu = mxlu.getmini();
        MaxminTabDouble mylu = new MaxminTabDouble(ylu);
        double maxylu = mylu.getmaxi();
        double minylu = mylu.getmini();
```

```
        // construction de l'interpolation polynomiale :
        Fbase1 f1 = new Fbase1();
        Fbase2 f2 = new Fbase2();
        Fbase3 f3 = new Fbase3();
        Fbase4 f4 = new Fbase4();
        Fbase5 f5 = new Fbase5();
        Fbase6 f6 = new Fbase6();
        Fbase7 f7 = new Fbase7();
        Fbase8 f8 = new Fbase8();
        Fbase9 f9 = new Fbase9();
        Fbase10 f10 = new Fbase10();
        FoncD2D [] fbase = {f1, f2, f3, f4, f5, f6, f7, f8, f9, f10};
        Modlin ml = new Modlin(fbase);

...
```

Les résultats obtenus sont donnés ci-après, ainsi que la fenêtre graphique générée dans la figure 6.6.

```
java PrgModlinPoly

Resultats de l'identifiaction des coef. :
coef. 0 : 7.005477494642518
coef. 1 : 8.749201812979114
coef. 2 : -2.1606780667741687
coef. 3 : -1.4914188653043547
coef. 4 : 0.22366539817694758
coef. 5 : 0.1196340396538589
coef. 6 : -0.00928900068800065
coef. 7 : -0.003989574631021212
coef. 8 : 1.2400793650802364E-4
coef. 9 : 4.5863252708478754E-5
```

Nous montrons maintenant un nouveau calcul, à partir des mêmes données, et qui utilise un modèle constitué d'une somme de 4 fonctions trigonométriques :

$$p(x) = a_1 \cos(0.5x) + a_2 \sin(0.5x) + a_3 \cos(1.5x) + a_4 \sin(1.5x)$$

La partie de programme à modifier est la suivante :

```
...
```

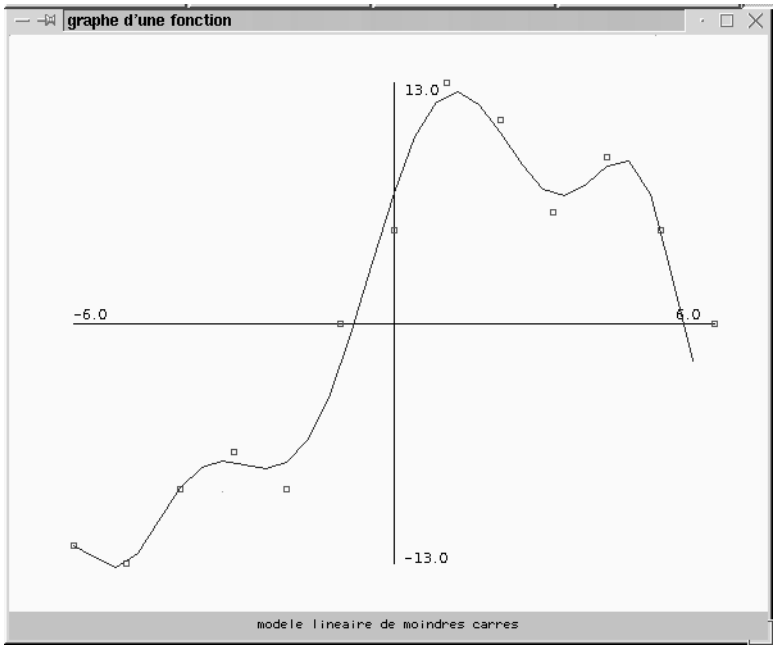

FIG. 6.6: Fenêtre générée par me programme de calcul d'un modèle linéaire en utilisant un polynôme de degré 9

```
class Fbase1 implements FoncD2D{
    public double calcul(double x) {return Math.cos(x/2);}
    public String libelle() {return "f(x)=cox(x/2)";}
}

class Fbase2 implements FoncD2D{
    public double calcul(double x) {return Math.sin(x/2);}
    public String libelle() {return "f(x)=sin(x/2)";}
}

class Fbase3 implements FoncD2D{
    public double calcul(double x) {return Math.cos(1.5*x);}
    public String libelle() {return "f(x)=cos(1.5*x)";}
}

class Fbase4 implements FoncD2D{
    public double calcul(double x) {return Math.sin(1.5*x);}
    public String libelle() {return "f(x)=sin(1.5*x)";}
}
```

```
class PrgModlinTrigo {
    public static void main(String args[]) {
        // lecture du fichier de données
        TableauFileIn f = new TableauFileIn("donnees.dat");
        double[] xlu = f.lectureTableau();
        double[] ylu = f.lectureTableau();
        f.fermer();

        // calcul des extrema des tableaux :
        MaxminTabDouble mxlu = new MaxminTabDouble(xlu);
        double maxxlu = mxlu.getmaxi();
        double minxlu = mxlu.getmini();
        MaxminTabDouble mylu = new MaxminTabDouble(ylu);
        double maxylu = mylu.getmaxi();
        double minylu = mylu.getmini();

        // construction de l'interpolation polynomiale :
        Fbase1 f1 = new Fbase1();
        Fbase2 f2 = new Fbase2();
        Fbase3 f3 = new Fbase3();
        Fbase4 f4 = new Fbase4();
        FoncD2D [] fbase = {f1, f2, f3, f4};
        Modlin ml = new Modlin(fbase);

...
```

Les résultats numériques et graphiques (cf. figure 6.7) sont donnés ci-après.

```
java PrgModlin

Resultats de l'identifiaction des coef. :
coef. 0 : 5.208201909446939
coef. 1 : 10.348725340528471
coef. 2 : 1.4466187029503472
coef. 3 : 3.250853663738151
```

Nous remarquons une assez bonne solution grâce à ce polynôme trigonométrique alors que, seuls, quatre coefficients sont à calculer, comparativement aux 10 coefficients de l'approximation polynômiale précédente. Toutefois, il est nécessaire que les fréquences des fonctions trigonométriques entrant en jeu soient correctement ajustées aux données d'entrée. En effet, ces fréquences ne sont pas

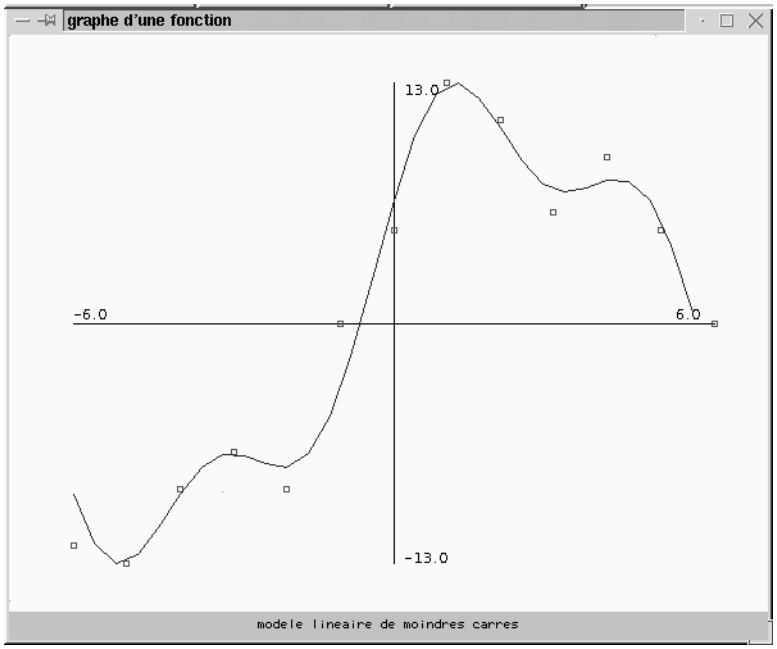

FIG. 6.7: Fenêtre générée par me programme de calcul d'un modèle linéaire en utilisant un polynôme trigonométrique à 4 termes

calculées par la méthode d'ajustement. Pour qu'elles soient déterminées par un processus de moindres carrés, il serait nécessaire d'introduire des coefficients à identifier à l'intérieur des fonctions trigonométriques : le modèle n'est alors plus linéaire et la méthode décrite, ici, n'est plus suffisante. Il faut élaborer une méthode d'ajustement de modèles non linéaires. Pour cela, un procédé classique serait de plonger la méthode précédente dans un processus itératif où l'on remplace les fonctions non linéaires par leur développement limité au premier ordre. La convergence du processus itératif permet alors d'obtenir la solution de notre problème non linéaire. Nous n'irons pas plus loin, dans le cadre de cet ouvrage, sur cette méthode suggérée ici, même si les classes que nous avons déjà présentées, notamment la classe `IterGene` du chapitre 2, nous fournissent des outils intéressants de démarrage pour une mise en œuvre.

6.4 Enoncés des exercices corrigés

Exercice 45 :

.....

Exercice 46 :

.....

Exercice 47 :

.....

Exercice 48 :

.....

Exercice 49 :

.....

6.5 Enoncés des exercices non corrigés

6.6 Corrigés des exercices

.....
A FAIRE
.....

exercice 22

exercice 23

exercice 25

exercice 26

exercice 29

Chapitre 7

Intégration Numérique

Les méthodes d'intégration numérique, interviennent essentiellement lorsque une primitive de f est d'expression assez compliquée ou inconnue, ou lorsque f n'est connue que par points, par exemple si elle résulte de mesures physiques, on peut l'approcher alors par interpolation (chap. précédent), puis on intègre numériquement l'interpolée. On traitera seulement les intégrales du type :

$$I = \int_a^b f(x)dx$$

où $[a, b]$ est un intervalle fini de $\mathbb{R}$ et f est continue sur $[a, b]$.

7.1 Méthodes des Rectangles

7.1.1 Méthode des Rectangles supérieurs et inférieurs

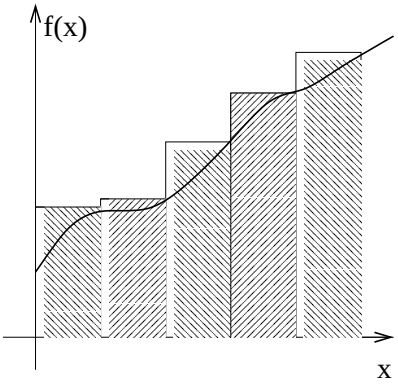

Figure 8

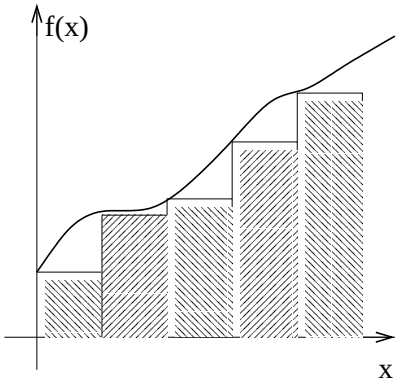

Figure 9

On considère une subdivision de $[a, b]$ en sous intervalles égaux $[x_i, x_{i+1}]$ où $x_i = a + ih$, $(i = 0, \cdots, n)$, $a = x_0, b = x_n$ et $h = \frac{b-a}{n}$. On suppose les valeurs

$f(x_i)$ connues pour $i = 0, \cdots, n$. Puis sur chaque sous-intervalle $[x_i, x_{i+1}]$, ($i = 0, \cdots, n-1$) on remplace la fonction $f(x)$ par $f(x_i)$ dans le cas de la méthode des rectangles supérieurs (fig 8) et par $f(x_{i+1})$ dans le cas de la méthode des rectangles inférieurs (fig 9).

7.1.2 Méthode des réctangles points-milieux

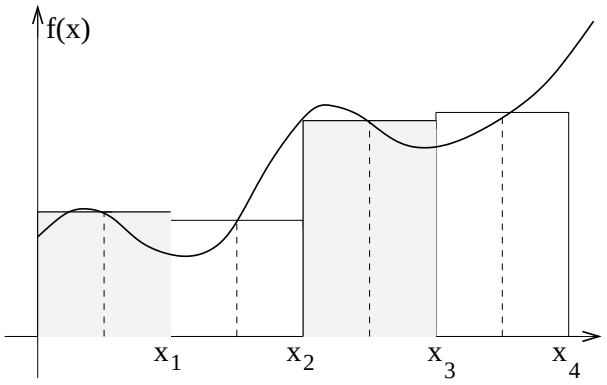

Figure 10

Cette fois-ci on prend pour chaque sous-intervalle $[x_i, x_{i+1}]$, $f(\frac{x_i+x_{i+1}}{2})$ comme hauteur du rectangle, on a alors

$$\int_{x_i}^{x_{i+1}} f(x)dx \approx hf(\frac{x_i + x_{i+1}}{2})$$

d'où

$$\int_a^b f(x)dx \approx \sum_{i=0}^{n-1} hf(\frac{x_i + x_{i+1}}{2}) = \sum_{i=0}^{n-1} hf(x_0 + \frac{2i+1}{2}h)$$

On démontre que l'erreur de cette méthode est :

$$\frac{(b-a)^2}{2N} \max_{x\in[a,b]} |f'(x)|$$

7.2 Méthodes des trapèzes

On considère une subdivision de [a,b] en sous intervalles égaux $[x_i, x_{i+1}]$ de bornes $x_i = a + ih$, ($i = 0, \cdots, n$), $a = x_0, b = x_n$ et $h = \frac{b-a}{n}$. On suppose connues les valeurs $f(x_i)$ pour $i = 0, \cdots, n$. Sur chaque intervalle $[x_i, x_{i+1}]$ on remplace la fonction $f(x)$ par la droite : $y = f(x_i) + (x - x_i)\frac{f(x_{i+1}) - f(x_i)}{x_{i+1} - x_i}$.

L'aire exacte est donc remplacée par l'aire T_i du trapèze. Par conséquent en sommant les aires des n trapèzes de base $[x_i, x_{i+1}]$, $(i = 0, \cdots, n)$, on obtient une approximation $T_h$de l'intégrale I ; comme $T_i = \frac{h}{2}(f(x_{i+1}) + f(x_i))$, alors

$$I = \int_a^b f(x)dx \approx \frac{h}{2}[f(x_0) + f(x_n) + 2\sum_{i=1}^{n-1} f(x_i)] = T_h$$

Pour justifier ce procédé il faut montrer que T_h peut être rendu aussi proche que l'on veut de I pourvu que le pas h soit 'assez petit' et que f soit 'assez régulière' (voir les références données en fin de ce polycope). On démontre que l'erreur $E_h = I - T_h$ commise par application de cette méthode est donnée par :

$$E_h = -\frac{(b-a)h^2}{12} f"(\xi) \ , \quad \xi \in [a, b]$$

Donc si $M_2 = \max_{[a,b]} |f''(t)|$, alors $|E_h| \leq \frac{(b-a)^3}{12N^2} M_2$

7.3 Méthode de Simpson

Dans cette méthode on suppose que n est pair (soit n=2s), puis on subdivise $[a, b]$ en s sous intervalles égaux $[x_{i-1}, x_{i+1}]$ de longueur h, ($i = 1 \cdots s - 1$), puis on remplace $f(x)$ sur chaque intervalle $[x_{i-1}, x_{i+1}]$ non pas par une droite comme dans les trapèzes, mais par une parabole ayant aux abscisses $x_{i-1}, x_i \ et \ x_{i+1}$ les mêmes valeurs que f ; c'est à dire par une interpolation quadratique sur ces trois points. Le polynôme P_1 d'interpolation en $x_{i-1}, x_i \ et \ x_{i+1}$ est (voir chapitre 4) :

$$\begin{aligned} P_1(x) &= f(x_{i-1})\frac{(x - x_{i+1})(x - x_i)}{2h^2} + \\ &= f(x_i)\frac{(x - x_{i-1})(x - x_{i+1})}{-h^2} + \\ &= f(x_{i+1})\frac{(x - x_i)(x - x_{i-1})}{2h^2} \end{aligned}$$

D'autre part la surface S_i délimitée par cette parabole, les droites $x = x_{i-1}$, $x = x_{i+1}$ et l'axe des abscisses s'obtient en calculant : $\int_{x_{i-1}}^{x_{i+1}} P_1(x)dx$ d'où ;

$$S_i = \frac{h}{3}(f(x_{i-1}) + 4f(x_i) + f(x_{i+1}))$$

Ceci donne l'approximation de Simpson S_h de I, en sommant les aires S_i pour $i = 1$ à $n - 1$. Finalement :

$$\begin{aligned} S_h &= \frac{h}{3}[f(x_0) + f(x_{2s}) + 4[f(x_1) + f(x_3) + \cdots \\ &\quad + f(x_{2s-1})] + 2[f(x_2) + f(x_4) + \cdots + f(x_{2s-2})]] \end{aligned}$$

c'est à dire :

$$S_h = \frac{h}{3}[f(x_0) + f(x_{2s}) + 4\sum_{k=1}^{s} f(x_{2k-1}) + 2\sum_{k=1}^{s-1} f(x_{2k})]$$

On démontre que l'erreur $E_h = I - S_h$ commise par application de Simpson est donnée par :

$$E_h = -\frac{(b-a)h^4}{180} f^{(4)}(\xi) \ , \quad \xi \in [a, b]$$

Donc si $M_4 = \max_{[a,b]} |f^{(4)}(t)|$, alors

$$|E_h| \leq \frac{(b-a)^5}{180N^4} M_4$$

Remarque 7 *En général la méthode de Simpson donne une meilleure approximation que celle des trapèzes, (sous certaines conditions de régularité de f, voir références ...), car l'erreur commise lorsqu'on applique les trapèzes est proportionnelle à h^2, alors que pour Simpson elle est proportionnelle à h^4, et comme par une transformation de la variable d'intégration on peut toujours se ramener de $[a, b]$ à $[0, 1]$ et qu'alors $h \in [0, 1]$, on a donc $h^4 < h^2$.*

Remarque 8 *Comparaison des erreurs :*

Rectangles	$(b-a)M_1 h$
Trapèzes	$\frac{b-a}{12} M_2 h^2$
Simpson	$\frac{b-a}{180} M_4 h^4$

7.4 Méthode de Romberg

C'est un procédé d'accélération de la convergence d'autres méthodes comme les trapèzes. En général pour calculer $I = I(0) = \int_a^b f(x)dx$, on fixe un pas $h = \frac{b-a}{n}$ et on approxime $I(0)$ par $I(h) + E(h)$, où $E(h)$ est l'erreur commise, avec l'hypothèse que cette erreur tend vers 0 avec h pour f suffisamment régulière.

L'idée de l'algorithme de Romberg est de prendre une combinaison linéaire de $I(h)$ et de $I(\frac{h}{2})$ qui donnera une meilleure approximation de $I(0)$, puis de recommencer avec $I(\frac{h}{4})$, etc $\cdots$.

Prenons par exemple la méthode des trapèzes on a $I(0) = T_n + E(h)$ où $T_n = T_h = I(h)$ désigne l'approximation pour n subdivisions de $[a, b]$ de longueur h.

Dans le cas où l'on peut faire un développement limité en h on a :

$$T_n = I(0) + a_1 h^2 + a_2 h^4 + \cdots + a_p h^{2p} + h^{2p}\epsilon(h)$$

donc l'erreur est

$$E(h) = -(a_1 h^2 + a_2 h^4 + \cdots + a_p h^{2p} + h^{2p}\epsilon(h))$$

qui est une série convergente (et $\approx h^2$). D'autre part :

$$T_{2n} = I(0) + a_1 (\frac{h}{2})^2 + a_2 (\frac{h}{2})^4 + \cdots + a_p (\frac{h}{2})^{2p} + (\frac{h}{2})^{2p}\epsilon(h)$$

Si on fait $\frac{4I(\frac{h}{2})-I(h)}{4-1}$, (c'est à dire $\frac{4T_{2n}-T_n)}{4-1}$), les termes en h^2 sont éliminés, donc l'erreur commise est équivalente à h^4, ce qui est bien sûr meilleur.
En continuant ce raisonnement avec $\frac{h}{4}, \frac{h}{8}, \cdots$ on obtient l'algorithme de Romberg dans lequel, pour clarifier les notations, les valeurs obtenues directement par les trapèzes seront notées $T_{1,n}, T_{1,2n}, T_{1,4n}, \cdots$, au lieu de $T_n, T_{2n}, T_{4n}, \cdots$ De même on notera $T_{k,n}, T_{k,2n} \cdots$ le résultat de la k^{eme} combinaison de Romberg, voir le tableau ci-dessous qui résume cet algorithme :

$$\begin{array}{llllll}
T_{1,n} & & T_{1,2n} & & T_{1,4n} & \cdots \\
\downarrow \alpha & \beta \swarrow & \downarrow \alpha & \beta \swarrow & \downarrow \alpha & \\
T_{2,n} & & T_{2,2n} & & \cdots & \\
\downarrow \alpha & \beta \swarrow & \downarrow \alpha & & & \\
T_{3,n} & & \cdots & & & \\
\downarrow \alpha & \beta \swarrow & & & & \\
T_{4,n} & & \cdots & & & \\
\vdots & & & & &
\end{array}$$

On construit la première ligne qui correspond aux trapèzes avec n subdivisions, $2n$ subdivisions, etc... et on désigne par les flêches α et β les différentes combinaisons à l'origine de la flêche , donc la deuxième ligne est obtenue grâce à : $\frac{4\beta-\alpha}{3} = \frac{2^2\beta-\alpha}{2^2-1}$ la troisième ligne est obtenue grâce : $\frac{4^2\beta-\alpha}{15} = \frac{2^{2X2}\beta-\alpha}{2^{2X2}-1}$, etc... d'où la règle de récurrence qui donne le passage de la k^{eme} ligne à la $(k+1)^{eme}$ ligne :

$$\begin{cases} \text{Une fois calculés } T_{k,n}, T_{k,2n}, T_{k,4n} \quad \cdots, \text{ on a} \\ T_{k+1,n} = \frac{2^{2k}T_{k,2n} - T_{k,n}}{2^{2k}-1} \end{cases}$$

la k^{eme} ligne du tableau de Romberg donne des approximations de l'intégrale pour lesquels l'erreur est en h^{2k} ; donc $T_{k,n}$ est plus précis que $T_{j,n}$ pour $j \leq k$, et d'autre part $T_{k,n}$ est moins précis que $T_{k,2n}$ qui l'est moins que $T_{k,4n}$, $\cdots$.
La précision augmente de haut en bas et de gauche à droite du tableau, mais attention, pas nécéssairement sur les diagonales.

Remarque 9 . *On remarque que dans la deuxième ligne du tableau de Romberg, $T_{2,n}$ est exactement le résultat de la méthode de Simpson.*

7.5 Enoncés des exercices corrigés

Exercice 50 :
a) Donner les formules des rectangles approchant $I = \int_a^b f(x)dx$.
b) Trouver l'erreur commise en appliquant la méthode des rectangles, ainsi que le nombre minimum n de subdivisions de $[0, 1]$ pour avoir $\int_0^1 e^{x^2}dx$ à 1/100 près
c) Donner la méthode des rectangles à point milieu.
d) Trouver l'erreur commise en appliquant la méthode des rectangles à point milieu. Même application qu'à la question b).

Exercice 51 :
Déterminer par le méthode des trapèzes puis par celle de Simpson $\int_0^{\frac{\pi}{2}} f(x)dx$ sur la base du tableau suivant :

x	0	$\pi/8$	$\pi/4$	$3\pi/8$	$\pi/2$
$f(x)$	0	0.382683	0.707107	0.923880	1

Ces points d'appui sont ceux donnant $\sin x$, comparer alors les résultats obtenus avec la valeur exacte.

Exercice 52 :
On lance une fusée verticalement du sol et l'on mesure pendant les premières 80 secondes l'accéleration γ :

$t(ens)$	0	10	20	30	40	50	60	70	80
$\gamma(enm/s^2)$	30	31.63	33.44	35.47	37.75	40.33	43.29	46.70	50.67

Calculer la vitesse V de la fusée à l'instant t=80 s, par les Trapèzes puis par Simpson.

Exercice 53 :
Calculer à l'aide de la méthode des Trapèzes l'intégrale $I = \int_0^{\pi} \sin(x^2)dx$ (avec N=5 puis N=10 points d'appui).

Exercice 54 :
Trouver le nombre N de subdivisions nécessaires de l'intervalle d'intégration $[-\pi, \pi]$, pour evaluer à 0.510^{-3} près, grâce à Simpson, l'intégrale : $I = \int_{-\pi}^{\pi} \cos x dx$.

Exercice 55 :
Evaluer a l'aide de la méthode des Trapèzes, l'intégrale : $\int_0^{\pi} \frac{\sin x}{x}dx$, avec une erreur inférieure à 10^{-4}.

Exercice 56 :
Evaluer a l'aide de la méthode des trapèzes puis celle de Romberg basée sur les trapèzes, l'intégrale : $\int_0^1 \frac{1}{1+x} dx$, avec une erreur inférieure à 10^{-3}.
Comparer les vitesses de convergence. Comparer à la valeur 'exacte' .

Exercice 57 :
Soit $F(x) = \int_0^x te^{-t} dt$. Combien faut-il de subdivisions de [0,1] pour evaluer $F(1)$ a 10^{-8} près en utilisant

1. la méthode des trapèzes
2. la méthode de Simpson

7.6 Enoncés des exercices non corrigés

Exercice 58 :
Soit $F(x) = \int_0^x t^2 e^{-t} dt$. Combien faut-il de subdivisions de [0,1] pour evaluer $F(1)$ a 10^{-3} pres en utilisant

1. la méthode des trapèzes
2. la méthode de Simpson

Exercice 59 :
Utiliser la méthode de Simpson pour approcher $2\int_0^1 \sqrt{\cosh(2x)} dx$. en subdivisant l'intervalle $[0, 1]$ en $N = 10$ parties égales.

Exercice 60 :
Pour évaluer $F(1)$, où la fonction F est donnée par $F(x) = \int_0^x \sin(t) dt$, on intégre le développement de Taylor de $\sin(t)$ (qui commence comme suit : $t - \frac{t^3}{3!} + \frac{t^5}{5!}$...).

1. Donner l'erreur commise sur le résultat si on se contente du développement à l'ordre 6 de $F(x)$.
2. Estimer alors $F(1)$ à cette erreur près, et comparer à la valeur 'exacte' obtenue par votre calculette.

Exercice 61 :
Soit $F(x) = \int_0^x e^{-t^2} dt$.

1. En utilisant la méthode des trapèzes, combien faut-il de subdivisions de $[0, 1]$ pour evaluer $F(1)$ à 10^{-2} près ?
2. Déterminer alors $F(1)$ par les trapèzes à 10^{-2} près .

Exercice 62 :
Soit $G(x) = \int_0^x \cos^2(t) dt$.

1. En utilisant la méthode de Simpson, combien faut-il de subdivisions de $[0, 1]$ pour évaluer $G(1)$ à 10^{-3} près ?
2. Déterminer alors $G(1)$ par la méthode de Simpson à 10^{-3} près .

Exercice 63 :
Soit $F(x) = \int_0^x \sin(2t) dt$.

1. En utilisant la méthode de Simpson, combien faut-il de subdivisions de $[0, \pi/2]$ pour évaluer $F(\pi/2)$ à 10^{-2} près ?
2. Déterminer alors $F(\pi/2)$.

Exercice 64 :
Soit $F(x) = \int_0^x \sin(t)\cos(t)dt$.

1. En utilisant la méthode de Simpson, combien faut-il de subdivisions de $[0, 1]$ pour évaluer $F(1)$ à 10^{-3} près ?
2. Déterminer alors $F(1)$ par la méthode de Simpson à 10^{-3} près .

Exercice 65 :
Soit $F(x) = \int_0^x \tan(t)dt$.

1. En utilisant la méthode de Simpson, combien faut-il de subdivisions de $[0, \frac{\pi}{4}]$ pour évaluer $F(\frac{\pi}{4})$ à 10^{-3} près ?
2. Déterminer alors $F(\frac{\pi}{4})$. Comparer à la solution exacte.

Exercice 66 :
Soit $F(x) = \int_1^x \ln(t^2)dt$.

1. En utilisant la méthode de Simpson, combien faut-il de subdivisions de $[1, e]$ pour évaluer $F(e)$ à 10^{-2} près ?
2. Déterminer alors $F(e)$.

Exercice 67 :
Soit $F(x) = \int_0^x \frac{1}{(1+t)^2}dt$.

1. En utilisant la méthode des trapèzezs, combien faut-il de subdivisions de $[0, 1]$ pour évaluer $F(1)$ à 10^{-3} près ?
2. Déterminer alors $F(1)$ par la méthode des trapèzezs à 10^{-3} près .

Exercice 68 :
Soit $I = \displaystyle\int_a^b f(t)dt$. On considère une subdivision de $[a, b]$ en n sous intervalles égaux de bornes $t_i = a + ih$, où $i = 0, \cdots, n$, $a = t_0$, $b = t_n$ et $h = \dfrac{b-a}{n}$.

1. Retrouver l'erreur $E_i(h)$ commise par la méthode des trapèzes sur un sous intervalle $[t_i, t_{i+1}]$. Montrer que la formule générale de l'erreur $E = I - T_h$ commise par application de cette méthode, sur tout $[a, b]$, est donnée par :

$$E_h = -\frac{(b-a)h^2}{12} f"(\xi) \ , \quad \xi \in [a, b]$$

   En déduire que si $M_2 = \max_{[a,b]} |f''(t)|$, alors $|E_h| \leq \frac{(b-a)^3}{12N^2} M_2$.
2. Soit $f(t) = \dfrac{e^{-t}}{t^2}$ et $I_\infty = \displaystyle\int_1^{+\infty} f(t)dt$.

(a) Montrer que $I_\infty = J_1$ où $J_1 = \int_0^1 g(u)du$ et où $g(u) = e^{-\frac{1}{u}}$, (penser à un bon changement de variable).

(b) En utilisant le fait que $\lim_{u \longrightarrow 0^+} e^{-\frac{1}{u}} = 0$, approximer J_1 par la méthode des trapèzes. Si $h = 0.5$ donner la valeur approchée de J_1 par cette méthode.

(c) Combien faut-il de subdivisions de [0,1] pour évaluer J_1 à 10^{-3} près en utilisant la méthode des trapèzes.

Exercice 69 :
Pour approcher numériquement l'intégrale $I = \int_0^{+\infty} \frac{dx}{1+x^3}$, on l'écrit sous la forme $I = K_\beta + K_\infty$ où $K_\beta = \int_0^{+\beta} \frac{dx}{1+x^3}$ et $K_\infty = \int_\beta^{+\infty} \frac{dx}{1+x^3}$ (β réel strictement positif).

1. Montrer que l'on peut déterminer un réel strictement positif β tel que $K_\infty \leq \frac{1}{2}10^{-4}$.
   (Indication : majorer $\frac{1}{1+x^3}$ par $\frac{1}{x^3}$).
2. Pour calculer K_β, on utilise la méthode des trapèzes :
   (a) Rappeler la formule des trapèzes ainsi que l'erreur commise par son application. Donner une démonstration de l'erreur.
   (b) Exprimer en fonction de $M = max_{[0,\beta]}|f"(x)|$ le nombre N de points d'intégration nécessaire pour calculer K_β avec une erreur inférieure à $\frac{1}{2}10^{-4}$.
   (c) Trouver M et en déduire N.

7.7 Corrigés des exercices

exercice 31

$$I = \int_0^{\frac{\pi}{2}} f(x)dx$$

a) Soit T l'approximation de I par la méthode des trapèzes, le pas h est donné par, $h = \frac{x_N - x_0}{N} = \frac{\pi}{8}$.

$$\begin{aligned} T &= \frac{h}{2}\left(f(x_0) + f(x_1) + 2\sum_{i=1}^{3} f(x_i)\right) \\ &= \frac{\pi}{16}\big(0 + 1 + 2(0,382683 + 0.707107 + 0.92388)\big) \\ &= 0.987116 \end{aligned}$$

b) Soit S l'approximation de I par la méthode de Simpson. Celle-ci s'écrit,

$$\begin{aligned} S &= \frac{h}{3}\left(y_0 + y_4 + 4(y_1 + y_3) + 2y_2\right) \\ &= \frac{\pi}{8}.\frac{1}{3}\big[(0 + 1 + 4(0.38\cdots + 0.92\cdots) + 2 \cdot 0.707)\big] \\ &= 1.000135 \end{aligned}$$

Les points d'appui donnés dans cet exercice correspondent à la fonction $\sin x$. Et $I = \int_0^{\frac{\pi}{2}} \sin x\, dx = 1$. On constate donc que l'approximation de I par Simpson est meilleure que celle par les trapèzes, puisque $|S - I| = 0.000135$ et $|T - I| = 0.012884$.

exercice 32

On sait que l'acceleration γ est la dérivée de la vitesse V, donc,

$$V(t) = V(0) + \int_0^t \gamma(s)\, ds$$

$$V(80) = 0 + \underbrace{\int_0^{80} \gamma(s)\, ds}_{I}$$

a) Calculons I par la méthode des trapèzes. Ici, d'après le tableau des valeurs, $h = 10$.

$$\begin{aligned} I &= \frac{h}{2}\left(\gamma(x_0) + \gamma(x_n) + 2\sum_{i=1}^{n-1}\gamma(x_i)\right) \\ &= \frac{1}{2}.10\big(30 + 50,67 + 2(31,63 + ... + 46,70)\big) \\ &= 3089\, m.s^{-1} \end{aligned}$$

b) Calculons I par la méthode de Simpson.

$$\begin{aligned} V(80) &= \frac{h}{3}\big(\gamma(x_0) + \gamma(x_n) + 4(\gamma(x_1) + \gamma(x_3) + \cdots) + 2(\gamma(x_2) + \gamma(x_4) + \cdots)\big) \\ &= \frac{10}{3}\big(30 + 50,67 + 4(31,63 + 35,47 + ...) + 2(33,44 + 37,75 + ...)\big) \\ &= 3087\, m.s^{-1} \end{aligned}$$

exercice 34

Soit

$$I = \int_0^{\pi} \sin x^2\, dx$$

a) $N = 5$ donc le pas d'intégration est $h = \frac{\pi}{5}$. Calculons I par la méthode des trapèzes.

$$\begin{aligned} I &= \frac{h}{2}\left(f(x_0) + f(x_n) + 2\sum_{i=1}^{n-1} f(x_i)\right) \\ &= \frac{\pi}{10}\big(\sin(\pi^2) + \sin(0) + 2(\sin(\tfrac{\pi}{5})^2 + \sin(\tfrac{2\pi}{5})^2 + \sin(3\tfrac{\pi}{5})^2 + \sin(\tfrac{4\pi}{5})^2)\big) \\ &= 0.504431 \end{aligned}$$

a) $N = 10$ donc le pas d'intégration est $h = \frac{\pi}{10}$.

$$\begin{aligned} I &= \frac{\pi}{20}\big(\sin(\pi^2) + \sin(0) + 2(\sin(\tfrac{\pi}{10})^2 + \sin(\tfrac{2\pi}{10})^2 + \sin(\tfrac{3\pi}{10})^2 + \cdots + \sin(\tfrac{9\pi}{10})^2)\big) \\ &= 0.72238 \end{aligned}$$

alors que la valeur 'exacte' est approximativement 0.772651. Avec ce pas plus petit l'approximation numérique est meilleure.

exercice 35

Soit

$$I = \int_{-\pi}^{\pi} \cos x \, dx$$

Le pas d'intégration est $h = \dfrac{b-a}{N} = \dfrac{2\pi}{N}$. D'autre part l'erreur théorique sur la méthode de Simpson est donnée par

$$\begin{aligned} E(h) &= \frac{-(b-a)}{180} h^4 f^{(4)}(\xi) \\ E(h) &= \frac{-2\pi}{180} . \left(\frac{2\pi}{N}\right)^4 \cos\xi \end{aligned}$$

où $\xi \in [a, b]$, par conséquent,

$$|E(h)| \leq \left| \frac{2\pi}{180} . \left(\frac{2\pi}{N}\right)^4 \right|$$

Ainsi pour que $|E(h)| \leq 0.5 \times 10^{-3}$ il suffit que N vérifie $|\frac{\pi}{90}\frac{16\pi^4}{N^4}| \leq 0.5 \times 10^{-3}$, donc, $N^4 \geq \dfrac{1}{0.5 \times 10^{-3}} \dfrac{\pi}{90} 16\pi^4$. Ainsi N vérifie $N \geq 18.6$. On prendra $N = 20$, car pour Simpson, le nombre de subdivisions de l'intervalle $[a, b]$ doit toujours être pair.

exercice 36

Soit

$$\int_0^{\pi} \frac{sin(x)}{x} dx$$

L'erreur de la méthode des trapèzes est :

$$E(h) = -\frac{b-a}{12} h^2 f''(\xi)$$

donc, $|E(h)| \leq \dfrac{b-a}{12}\dfrac{(b-a)^2}{n^2} M_2$ où $M_2 = \max\limits_{t\in[a,b]} |f''(t)|$. Ainsi, pour avoir $|E(h)| \leq \varepsilon$, où ε est l'erreur permise, il suffit de prendre N subdivisions de l'intervalle d'intégration telles que

$$\frac{(b-a)^3}{12}\frac{M_2}{N^2} \leq \varepsilon$$

Donc, il suffit que N vérifie,

$$N \geq \sqrt{\frac{(b-a)^3}{12\varepsilon} M_2} \qquad (1)$$

Cherchons M_2. On a $f(x) = \frac{\sin x}{x}$ et

$$f''(x) = -\frac{\sin x}{x} - \frac{2\cos x}{x^2} + \frac{2\sin x}{x^3}$$

Pour dterminer les extremas de f'', on cherche alors les racines de $f^{(3)}(x) = 0$. Un calcul simple donne

$$f^{(3)}(x) = \frac{1}{x^4}(-x^3\cos x + 6x\cos x + 3x^2\sin x - 6\sin x).$$

On cherche donc les racines de l'équation,

$$-x^3\cos x + 6x\cos x + 3x^2\sin x - 6\sin x = 0.$$

D'où

$$(-3x + 6x)\cos x = (6 - 3x^2)\sin x.$$

En supposant $\cos x \neq 0$ et $6 - 3x^2 \neq 0$, c'est à dire $x \neq (2k+1)\frac{\pi}{2}$, ($k \in \{0, 1, 2...\}$), et $x \neq \pm\sqrt{2}$, on obtient

$$\tan x = \frac{6x - x^2}{6 - 3x^2}.$$

Ain,si les racines cherchées sont $x = (2k+1)\frac{\pi}{2}$ ou $x = \pm\sqrt{2}$ ou celles de l'équation

$$\tan x = \frac{6x - x^3}{6 - 3x^2} \qquad (2).$$

Étudions alors cette équation (2) : Graphiquement, en tracant les courbes représentatives dans $[0, \pi]$ de $\tan x$ et de $\frac{6x-x^3}{6-3x^2}$, on voit rapidement (à faire) que le seul point d'intersection de $\tan x$ et de $\frac{6x-x^3}{6-3x^2}$ est d'abscisse $x = 0$.

On en déduit que f'' atteint ses extremas sur $[0, \pi]$ en $x = 0$ ou $x = \sqrt{2}$ ou $x = \frac{\pi}{2}$.

Cherchons $\lim\limits_{x \to 0^+} f''(x)$, le développement limité en $x = 0$ de $f''(x)$ donne :

$$\begin{aligned} f''(x) &= \frac{1}{x^3}\left(-x^2\left(x - \frac{x^3}{6} + ...\right) - 2x\left(-\frac{x^2}{2} + \frac{x^4}{4!} + ...\right) + 2\left(-\frac{x^3}{3!} + \frac{x^5}{5!}\right) + 0(x^5)\right) \\ &= \frac{1}{x^3}\left(-\frac{x^3}{3} + 0(x^5)\right). \end{aligned}$$

Donc, $\lim_{x\to 0^+} f''(x) = -\frac{1}{3}$. Par ailleurs, on vérifie facilement que, $\frac{1}{3} \geq |f''(\sqrt{2})|$ et $\frac{1}{3} \geq |f''(\frac{\pi}{2})|$. On a donc $M_2 = \frac{1}{3}$, et par conséquent de l'équation (1) on obtient,

$$\begin{aligned} N &\geq \sqrt{\frac{\pi^3}{12}10^2\frac{1}{3}} \\ &\geq 10. \end{aligned}$$

Application.

Prenons alors, $N = 10$, d'où $h = \frac{b-a}{N} = \frac{\pi}{10}$, et donnons une approximation de l'intégrale I par la méthode des trapèzes :

$$I = \int_0^{\pi} \frac{\sin x}{x} dx \simeq \frac{\pi}{20}\left(1 + 0 + 2\left(\frac{\sin\left(\frac{\pi}{10}\right)}{\frac{\pi}{10}} + \frac{\sin\left(\frac{2\pi}{10}\right)}{\frac{2\pi}{10}} + ... + \frac{\sin\left(\frac{9\pi}{10}\right)}{\frac{9\pi}{10}}\right)\right)$$

$$I \simeq 1,845$$

exercice 38

Soit

$$F(x) = \int_0^x te^{-t}dt.$$

On a $F(1) = \int_0^1 te^{-t}dt$,

a) Cherchons le nombre de subdivisions de $[0, 1]$ pour evaluer $F(1)$ a 10^{-8} près en utilisant la méthode des trapèzes.

L'erreur de la méthode des trapèzes est :

$$E(h) = -\frac{b-a}{12}h^2 f''(\xi)$$

donc, $|E(h)| \leq \frac{b-a}{12}\frac{(b-a)^2}{N^2}M_2$ où $M_2 = \max_{t\in[a,b]} |f''(t)|$. Ainsi, pour avoir $|E(h)| \leq \varepsilon$, où $\varepsilon = 10^{-8}$, il suffit de prendre N subdivisions de l'intervalle d'intégration telles que

$$\frac{(b-a)^3}{12}\frac{M_2}{N^2} \leq \varepsilon$$

Donc, il suffit que N vérifie,

$$N \geq \sqrt{\frac{(b-a)^3}{12\varepsilon}M_2} \qquad (1)$$

Cherchons M_2. On a $f(t) = te^{-t}$, $f''(t) = (t-2)e^{-t}$ et $f^{(3)}(t) = (3-t)e^{-t}$. Donc $M_2 = |f''(0)| = 2$.

Ainsi, l'inéquation (1) devient $N \geq \frac{10^4}{\sqrt{6}}$, soit $N \geq 4083$. Avec $N = 4083$ subdivisions, l'erreur commise lors de l'approximation de cette intégrale par la méthode des trapèzes serea plus petite que 10^{-8}.

b) Cherchons le nombre de subdivisions de $[0, 1]$ pour evaluer $F(1)$ a 10^{-8} près en utilisant la méthode de Simpson.

L'erreur théorique de cette méthode est :

$$E(h) = -\frac{b-a}{180}h^4 f^{(4)}(\xi)$$

donc, $|E(h)| \leq \frac{(b-a)^5}{180N^4}M_4$ où $M_4 = \max_{t\in[a,b]} |f^{(4)}(t)|$. Ainsi, pour avoir $|E(h)| \leq \varepsilon$, où $\varepsilon = 10^{-8}$, il suffit de prendre N subdivisions de l'intervalle d'intégration telles que

$$\frac{(b-a)^5}{180N^4}M_4 \leq \varepsilon.$$

Donc, il suffit que N vérifie,

$$N^4 \geq \frac{(b-a)^5}{180\varepsilon}M_4 \qquad (2)$$

Cherchons M_4. On a $f(t) = te^{-t}$, $f^{(4)}(t) = (t-4)e^{-t}$ et $f^{(5)}(t) = (5-t)e^{-t}$. Donc $M_4 = |f^{(4)}(0)| = 4$.

Ainsi, l'inéquation (2) devient $N^4 \geq \frac{4}{180\times10^{-8}}$, soit $N \geq 38,6$. On prendra alors $N = 40$ puisque, dans les cas de la méthode de Simpson, le nombre de subdivisions de l'intervalle d'integration doit toujours être pair.

7.8 Mise en œuvre en Java

Nous présentons des mises en œuvre des méthodes des Trapèzes, de Simpson et de Romberg.

7.8.1 Des classes abstraites d'intégration numérique

Les deux premières méthodes implémentées, celle des Trapèzes et de Romberg, nécessitent des interfaces analogues, tout en mettant en œuvre des calculs différents. Nous avons décrit des classes abstraites pouvant être dérivées avec l'une ou l'autre méthode. Ce procédé permettra d'utiliser ces classes abstraites pour la méthode de Romberg qui pourra alors être utilisée avec l'une ou l'autre méthode.

Deux approches sont mises en œuvre qui dépendent de la situation pratique dans laquelle se trouve l'utilisateur :

– L'utilisateur dispose d'une représentation analytique ou informatique de la fonction dont il cherche l'intégrale sur un intervalle $[a, b]$. La classe abstarite suivante va permettre de décrire cette situation. Elle utilise la classe `FoncD2D` décrite et utilisée de nombreuses fois dans les chapitres précédents :

```
abstract class IntNumFct {
    double a,b;
    FoncD2D f;
    IntNumFct(double b1, double b2, FoncD2D fonc) {
        a = b1; b = b2; f = fonc;
    }
    abstract public double calcul(int nbSub) throws NbSubException;
}
```

– L'utilisateur peut ne connaître une fonction qu'à partir d'un tableau de valeurs en des points régulièrement espacés. Dans ce cas, s'il ignore la forme analytique de la fonction, il ne pourra utiliser que ce tableau de valeurs qui fixe le pas h séparant chaque point d'évaluation de la fonction. La classe abstraite suivante va permettre de décrire cette situation. Elle utilise une description de la fonction dans un tableau de valeurs :

```
abstract class IntNumTab {
    double pas;
    double[] valf;
    IntNumTab(double longIntervalle, double[] t) {
        valf = t;
        pas = longIntervalle/(t.length - 1);
    }
```

```
        abstract public double calcul() throws NbSubException;
    }
```

Par ailleurs, il a été remarqué dans les paragraphes suivants que le nombre de subdivisions devait respecter des propriétés de parité pour pouvoir utiliser la méthode de Simpson. C'est pour cette raison, que nous allons utiliser un traitement d'exception pour gérer une situation ne vérifiant pas la condition adéquate. Nous créons donc la classe d'exception qui suit :

```
class NbSubException extends Exception {
    public String toString() {
        return ("Nombre de subdivisions impropre");
    }
}
```

7.8.2 Des classes d'implémentation de la méthode des trapèzes

Nous présentons ci-après deux classes implémentant de manière élémentaire la formule des trapèzes telle que décrite dans les paragraphes précédents. Chacune des classes dérive d'une des deux classes abstraites précédentes.

```
class ITrapFct extends IntNumFct {
    ITrapFct(double b1, double b2, FoncD2D fonc) {
        super(b1,b2,fonc);
    }
    public double calcul(int nbSub) throws NbSubException {
        double pas = (b-a)/nbSub;
        double x = a;
        double s = f.calcul(a) + f.calcul(b);
        for (int i=1; i<nbSub; i++) {
            x += pas;
            s += 2*f.calcul(x);
        }
        return s*pas/2;
    }
}

class ITrapTab extends IntNumTab {
    ITrapTab(double longIntervalle, double[] t) {
        super(longIntervalle,t);
    }
    public double calcul() throws NbSubException{
        int nbSub = valf.length-1;
```

```
        double s = valf[0] + valf[nbSub];
        for (int i=1; i<nbSub; i++)
            s += 2*valf[i];
        return s*pas/2;
    }
}
```

7.8.3 Des classes d'implémentation de la méthode de Simpson

Nous présentons ci-après deux classes implémentant de manière élémentaire la formule de Simpson telle que décrite dans les paragraphes précédents. Chacune des classes dérive d'une des deux classes abstraites précédentes. Ces classes sont succeptibles de lancer des exceptions si la parité du nombre de subdivisions n'est pas satisfaite.

```
class ISimpsonFct extends IntNumFct {
    ISimpsonFct(double b1, double b2, FoncD2D fonc) {
        super(b1,b2,fonc);
    }
    public double calcul(int nbSub) throws NbSubException {
        if ((nbSub % 2) != 0) throw new NbSubException();
        double pas = (b-a)/nbSub;
        double x = a;
        double s = f.calcul(a) + 4*f.calcul(a+pas) + f.calcul(b);
        for (int i=3; i<nbSub; i++,i++) {
            x += 2*pas;
            s += 2*f.calcul(x) + 4*f.calcul(x+pas);
        }
        return s*pas/3;
    }
}

class ISimpsonTab extends IntNumTab {
    ISimpsonTab(double longIntervalle, double[] t) {
        super(longIntervalle,t);
    }
    public double calcul() throws NbSubException {
        int nbSub = valf.length-1;
        if ((nbSub % 2) != 0) throw new NbSubException();
        double s = valf[0] + 4*valf[1] + valf[nbSub];
        for (int i=2; i<nbSub; i++,i++)
            s += 2*valf[i] + 4*valf[i+1];
```

```
        return s*pas/3;
    }
}
```

7.8.4 Un programme de test des méthodes des trapèzes et de Simpson

Nous présentons maintenant un programme de test des 4 classes précédentes et qui calcule l'intégrale de la fonction $\sin(x)$ sur l'intervalle $[0, \Pi/2]$. La valeur exacte étant égale à 1.

```
import java.lang.*;
import java.io.*;
import FoncD2D;
import TableauFileIn;
import NbSubException;
import ITrapFct;
import ISimpsonFct;
import ITrapTab;
import ISimpsonTab;

class SinF implements FoncD2D {
    public double calcul(double x) { return Math.sin(x); }
    public String libelle() { return "f(x)=sin(x)"; }
}

class TestIntNum {
    public static void main(String args[]) {
        SinF f = new SinF();
        double a=0, b=Math.PI/2;
        double longueur=b-a;
        int nbSub=4;

        // lecture du fichier de données
        TableauFileIn fic = new TableauFileIn("donnees.dat");
        double[] flu = fic.lectureTableau();
        fic.fermer();

        ITrapFct itf = new ITrapFct(a, b, f);
        ISimpsonFct isf = new ISimpsonFct(a, b, f);
        ITrapTab itt = new ITrapTab(longueur, flu);
        ISimpsonTab ist = new ISimpsonTab(longueur, flu);
```

```
        try {
            System.out.println("Integration numerique de sin(x)"+
                               "entre 0 et Pi/2, avec "+
                               nbSub+"subdivisions");
            System.out.println("Met. Trapezes version fct : "+
                               itf.calcul(nbSub));
            System.out.println("Met. Trapezes version tab : "+
                               itt.calcul());
            System.out.println("Met. Simpson version fct : "+
                               isf.calcul(nbSub));
            System.out.println("Met. Simpson version tab : "+
                               ist.calcul());
        }
        catch(NbSubException e) { System.out.println(e); }
    }
}
```

Nous donnons ci-après le fichier d'entrée utilisé par les méthodes travaillant sur des tableaux. Les valeurs suivantes correspondent à une tabulation de la fonction $\sin(x)$ sur $[0, \Pi/2]$, avec un pas de $\Pi/8$.

```
5
0 0.382683 0.707107 0.923880 1
```

Les résultats obtenus sont les suiavnts. Ils respectent parfaitement les calculs d'erreurs énoncés dans les paragraphes précédents.

```
java TestIntNum

Integration numerique de sin(x)entre 0 et Pi/2, avec 4subdivisions
Met. Trapezes version fct : 0.9871158009727754
Met. Trapezes version tab : 0.987115900693632
Met. Simpson version fct : 1.0001345849741938
Met. Simpson version tab : 1.000134660650108
```

7.8.5 Mise en œuvre de la méthode de Romberg

Nous présentons maintenant une classe implémentant de manière récursive la formule de calcul de la méthode de Romberg, telle qu'elle est décrite dans les paragraphes précédents :

```
class Romberg {
    IntNumFct methInt;
    Romberg(IntNumFct mi) {methInt = mi;}
    public double calcul(int ordre, int nbSub) throws NbSubException {
        int k=ordre-1;
        if (ordre==1) return methInt.calcul(nbSub);
        else {
            double ddk = Math.pow(2,2*k);
            return (ddk*calcul(k, 2*nbSub)-calcul(k, nbSub))/(ddk-1);
        }
    }
}
```

Le programme qui suit teste la méthode de Romberg en donnant les résultats de cette formulation pour différentes valeurs de l'ordre et du nombre de subdivisions. Par ligne d'affichage, l'ordre est constant : il commence à 1, puis à 2 et enfin à 3. Sue chaque ligne, on calcule la formule pour des nombres de subdivisions respectivement égaux à 2, puis 4 et enfin 8.

```
import java.io.*;
import FoncD2D;
import NbSubException;
import ITrapFct;
import Romberg;

class SinF implements FoncD2D {
    public double calcul(double x) { return Math.sin(x); }
    public String libelle() { return "f(x)=sin(x)"; }
}

class TestRomberg {
    public static void main(String args[]) {
        SinF f = new SinF();
        double a=0, b=Math.PI/2;

        ITrapFct itf = new ITrapFct(a, b, f);
        Romberg rmb = new Romberg(itf);

        System.out.println("tableau des valeurs approchees"+
                           "de l'integrale de sin(x) entre 0 et pi/2");
        System.out.println("sur la ligne k : Tk,2  Tk,4  Tk,8");
        try {
```

```
            for (int k=1; k<4; k++) {
                int n=1;
                for (int j=1; j<4; j++) {
                    n *= 2;
                    System.out.print(rmb.calcul(k,n)+"   ");
                }
                System.out.println();
            }
        }
        catch (NbSubException e) {System.out.println(e); }
    }
}
```

Nous présentons ci-après la trace de l'exécution du programme précédent :

```
java TestRomberg

tableau des valeurs approcheesde l'integrale de sin(x) entre 0 et pi/2
sur la ligne k : Tk,2  Tk,4  Tk,8
0.9480594489685199   0.9871158009727754   0.9967851718861696
1.0001345849741938   1.0000082955239675   1.0000005166847064
0.9999998762272857   0.9999999980954223   0.9999999999703542
```

Chapitre 8

Résolution Numérique des équations différentielles

8.1 Introduction, le problème mathématique

Soit $I = [a, b]$ un intervalle fermé de $\mathbb{R}$ et f une application donnée
$f : I \times \mathbb{R} \longrightarrow \mathbb{R}, \qquad (t, y) \longrightarrow f(t, y)$.
Et soit y une application différentiable de $\mathbb{R} \longrightarrow \mathbb{R}$.
On appelle *équation différentielle du premier ordre*, la relation

$$\frac{dy(t)}{dt} = f(t, y(t)). \tag{8.1}$$

On dit que y est la solution de cette équation différentielle sur $[a, b]$ si y vérifie la relation (1) pour tout $t \in [a, b]$.
On appelle *problème de Cauchy* ou *problème de condition initiale* l'équation différentielle à laquelle on adjoint la condition initiale $y(a) = y_0$ où y_0 est un nombre donnée :

$$\begin{cases} y'(t) = & f(t, y(t)) \\ y(a) = & y_0 \end{cases} \tag{8.2}$$

Remarque 10 *1. Si y est une fonction du temps, il est d'usage de noter la dérivée $\frac{dy}{dt}$ par $\dot{y}$ et de l'appeler vitesse. De même la dérivée seconde $\frac{d^2y}{dt^2}$ sera notée $\ddot{y}$ et appelée accéleration.*

2. Si on considère une équation différentielle d'ordre supérieure

$$y^{(p)} = f(t, y, y', \cdots, y^{(p-1)})$$

on peut ramener le problème à celui d'un système d'équations différentielles du premier ordre, en posant $z = (z_1, z_2, \cdots, z_p)$ et $y = z_1$ on obtient le

système :

$$(8.3) \qquad \begin{cases} z_1' & = z_2 \\ z_2' & = z_3 \\ \vdots & \vdots \\ z_{p-1}' & = z_p \\ z_p' & = f(t, z_1, z_2, \cdots, z_p) \end{cases}$$

Exemple 6 *Soit l'équation différentielle du second ordre :*

$$y''(t) = 3y'(t) - ty(t)$$

posons $y = z_1$ *et* $z_1' = z_2$ *alors cette équation se ramène au système :*

$$\begin{cases} z_1' & = z_2 \\ z_2' & = 3z_1' - tz_1 \end{cases}$$

Revenons au problème de Cauchy (2), un résultat fondamental est donné par le théorème ci-dessous, mais rappelons tout d'abord la définition d'une application lipschitzienne.

Définition 4 *Soit f une application définie sur* $[a, b]X\mathbb{R}$ *; s'il existe une constante* $L > 0$ *indépendante de* t, u *et* v *telle que* $|f(t, u) - f(t, v)| \leq L|u - v|$ $\forall u, v \in \mathbb{R}$*, et* $\forall t \in [a, b]$*, alors* f *est dite Lipschitzienne de rapport L sur* $[a, b]X\mathbb{R}$ *(ou simplement L-lipschitzienne).*

Théorème 9 *Si* f *est une application définie sur* $[a, b] \times \mathbb{R}$ *continue et L-lipschitzienne par rapport à* y*, alors le problème de cauchy (2) admet une solution unique sur [a,b] et ceci pour toute condition initiale* y_0*,* ($\forall y_0 \in \mathbb{R}$)*.*

Proposition 4 *Une condition suffisante pour que les hypothèses du théorème soient vérifiées est que* f *soit dérivable par rapport à* y *et que sa dérivée soit bornée. (dem en exercice ...)*

(Le théorème est en général faux sans la condition de Lipschitz.)

8.2 La Méthode d'Euler

Soit à intégrer numériquement le problème de Cauchy (2), et soit $y(t_i)$ la valeur exacte de la solution de ce problème à l'abscisse t_i. Une méthode d'analyse numérique pour intégrer cette équation différentielle consistera à fournir des approximations y_i de $y(t_i)$ pour $i = 1, \cdots, N$, $\quad N$ entier donné.

Les différentes méthodes d'intégration se distinguent par la manière d'obtenir ces y_i. La méthode d'Euler, est la plus simple et consiste à substituer la dérivée $y'(t) = \frac{dy}{dt}$ par l'expression

$$\frac{y(t+h) - y(t)}{h} \tag{8.4}$$

où $h = \frac{b-a}{N}$ est le pas d'intégration numérique.
Considérons alors une subdivision de $[a, b]$ en N sous intervalles $[t_i, t_{i+1}]$ de longueur h, $i = 0, \cdots, N$ et $t_i = a + ih$, avec $t_0 = a$ et $t_n = b$.
L'expression (4) entraine $y(t+h) = y(t) + hy'(t)$ d'où $y(t+h) = y(t) + hf(t, y)$.
Par conséquent, partant de la condition intiale (t_0, y_0), et prenant un pas régulier $h = t_{i+1} - t_i$, on obtient, en posant y_i approximation de $y(t_i)$:

$$y_{i+1} = y_i + hf(t_i, y_i) \tag{8.5}$$

L'algorithme d'Euler s'écrit alors :

$$\begin{cases} y_0 = y(a) & \text{donné} \\ y_{i+1} = y_i + hf(t_i, y_i) & i = 1, \cdots, N-1 \end{cases}$$

Interprétation géomètrique y_{i+1} approximation de $y(t_{i+1})$ est calculée à partir de (t_i, y_i) calculés précédemment grâce à la formule de réccurence (5) et à la condition initiale $y(t_0) = y_0$. On calcule une approximation de la solution y en tout point $t_0 + ih$ en remplacant dans $[t_i, t_{i+1}]$ la courbe intégrale $y(t)$ passant par y_i par sa tangente en t_i.

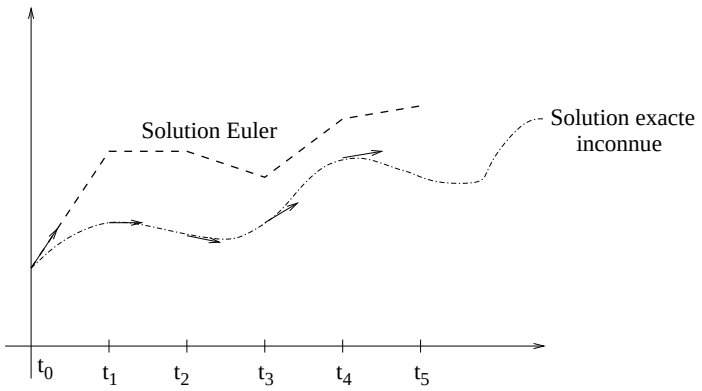

Figure 11

Remarque 11 *On dit que la méthode d'Euler est une méthode à pas séparés ou à un pas, parceque le calcul de y_{i+1} ne fait intervenir que y_i.*

8.2.1 Etude de l'erreur d'Euler

Définition 5 *Une méthode numérique approchant $y(t_j)$ par y_j telle que l'erreur $e_j = y(t_j) - y_j$ vérifie*

$$|e_j| \leq kh^p$$

est dite d'ordre p, ($k \in \mathbb{R}_*^+$).

Théorème 10 *Supposons que l'application $f(t, y)$ soit continue par rapport aux deux variables, et lipschitzienne par rapport à y uniformément par rapport à t, et que $y \in C^2[a, b]$. On pose $M_2 = max_{t\in[a,b]}|y''(t)|$, alors on a la majoration*

$$|e_i| \leq (e^{L(b-a)} - 1)\frac{M_2}{2L}h$$

où $e_i = y_i - y(t_i)$ est l'erreur commise au point (t_i, y_i) .

Remarque 12

1. *Ce résultat s'exprime sous la forme $|e_i| \leq kh$, c'est à dire que la méthode d'Euler est d'ordre 1.*
2. *Pour que $y \in C^2[a, b]$ il suffit que $f \in C^1([a, b]X\mathbb{R})$.*

On donne dans la suite des méthodes d'ordre plus élevé que celui de la méthode d'Euler, donc plus rapides et plus précises.

8.3 Méthodes de Taylor d'ordre deux

L'idée consiste à remplacer sur $[t_i, t_{i+1}]$ la courbe $y(t)$ solution de l'équation différentielle non plus par une droite mais par une parabole. En effet

$$y''(t) = \frac{\partial f(t, y)}{\partial t} + \frac{\partial f(t, y)}{\partial y} f(t, y)$$

et en négligeant les termes d'ordre supérieur, on obtient alors l'algorithme de *Taylor d'ordre deux* :

$$\begin{cases} h = \frac{b-a}{N},\ x_0 = a, \quad y_0 \text{ donné}, \quad t_{i+1} = t_i + h \\ y_{i+1} = y_i + hf(t_i, y_i) + \frac{h^2}{2}(\frac{\partial f}{\partial t} + \frac{\partial f}{\partial y} f)(t_i, y_i), \quad (i = 0, \cdots, N-1). \end{cases}$$

Cependant cette méthode présente un inconvenient certain qui réside dans le calcul de $\frac{\partial f}{\partial t} + \frac{\partial f}{\partial y} f$ qui peut être difficile ; pour éviter ce problème on utilise souvent la méthode de *Runge-Kutta-2* donnée plus bas.

Théorème 11 *Supposons que* $f(t,y) \in \mathcal{C}^2([a,b]X\!I\!R)$, fL_0*-lipschitzienne et* $\frac{\partial f}{\partial t} + \frac{\partial f}{\partial y} f$ L_1*-lipschitzienne par rapport à* y *uniformément par rapport à* t*. Posons* $M_3 = max_{t\in[a,b]}|y^{(3)}(t)|$*, alors on a la majoration*

$$|e_i| \leq (e^{(L_0 + \frac{b-a}{2}L_1)(b-a)} - 1)\frac{M_3}{6L_0}h^2$$

où $e_i = y_i - y(t_i)$ *est l'erreur commise au point* (t_i, y_i) *.*

8.4 Méthodes de Runge-Kutta

8.4.1 Runge-Kutta d'ordre 2 : RK2

On reprend l'algorithme de Taylor d'ordre 2 en remarquant qu'à des termes en h près on a grâce à un developpement de Taylor :

$$(f + h\frac{\partial f}{\partial t} + h\frac{\partial f}{\partial y}f)_{(t_i,y_i)} = f(t_i + h, y_i + hf(t_i, y_i))$$

d'où

$$y_{i+1} = y_i + \frac{h}{2}(f(t_i, y_i) + f(t_i + h, y_i + hf(t_i, y_i))$$

ainsi on obtient l'algorithme de RK2 :

$$\begin{cases} y_{i+1} = y_i + \frac{h}{2}(K_1 + K_2), & i = 0, \cdots, N-1 \quad \text{où} \\ K_1 = f(t_i, y_i), \quad \text{et} \quad K_2 = f(t_i + h, y_i + hK_1) \end{cases}$$

8.4.2 Runge-Kutta d'ordre 4 : RK4

Dans la pratique on utilise la méthode plus performante de Runge-Kutta d'ordre 4 (RK4), pour des fonctions suffisamment régulières on a alors $|e_i| = o(h^4)$:

$$\begin{cases} y_{i+1} = y_i + \frac{h}{6}(K_1 + 2K_2 + 2K_3 + K_4), & i = 0, \cdots, N-1, \quad \text{où} \\ K_1 = f(t_i, y_i), \quad K_2 = f(t_i + \frac{h}{2}, y_i + \frac{h}{2}K_1), \\ K_3 = f(t_i + \frac{h}{2}, y_i + \frac{h}{2}K_2), \quad \text{et} \quad K_4 = f(t_i + h, y_i + hK_2) \end{cases}$$

8.5 Généralités sur les méthodes à un pas

Les méthodes à un pas, aussi appelées méthodes à pas séparés, sont toutes de la forme :

$$y_{i+1} = y_i + h\Phi(t_i, y_i, h)$$

où Φ est une application continue de $[a,b]X\mathbb{R}X[0,h_0]$ dans $\mathbb{R}$; (h_0 est le rayon de stabilité de la méthode, souvent on prendra ici $h_0 = b - a$).
Les notions théoriques que doit vérifier la fonction Φ afin que y_i soit effectivement une approximation de $y(t_i)$ sont la consistance, la stabilité théorique et la convergence.

Définition 6 *Une méthode à un pas est consistante avec l'équation différentielle si pour toute solution continue y*

$$\lim_{h\to 0} max_{\,i} |\frac{1}{h}(y(t_{i+1}) - y(t_i) - \Phi(t_i, y(t_i), h)| = 0$$

Théorème 12 *Une condition nécéssaire et suffisante pour qu'une méthode à un pas soit consistante est que pour tout $t \in [a,b]$ et tout $v \in \mathbb{R}$ $\Phi(t,v,0) = f(t,v)$.*

Définition 7 *Stabilité théorique :*
Soient y_i et z_i ($i = 0, \cdots, N$) les solutions réspectives de

$$\begin{cases} y_{i+1} = y_i + h\Phi(t_i, y_i, h) \\ y_0 \in \mathbb{R} \quad \textit{donné} \end{cases}$$

et de

$$\begin{cases} z_{i+1} = z_i + h(\Phi(t_i, z_i, h) + \epsilon_i) \\ z_0 \in \mathbb{R} \quad \textit{donné} \end{cases}$$

On dit que la méthode à pas séparés est stable s'il existe deux constantes K_1 et K_2 indépendantes de h telles que :

$$\max_i |y_i - z_i| \leq K_1|y_0 - z_0| + K_2 max_i|\epsilon_i|$$

Théorème 13 *Si Φ vérifie une condition de Lipshitz par rapport à la seconde variable pour h suffisamment petit, alors la méthode à un pas est stable.*

Définition 8 *On dit qu'une méthode à un pas est convergente si :*

$$\forall y_0 \in \mathbb{R} \quad \lim_{h\to 0} \max_i |y_i - y(t_i)| = 0$$

Théorème 14 *Si une méthode à un pas est stable et consistante alors elle est convergente.*

8.6 Résolution de systèmes différentiels dans $\mathbb{R}^2$

Dans $\mathbb{R}^2$ soit le système différentiel :

$$\begin{cases} y'(t) = f(t, y, z) \\ z'(t) = g(t, y, z) \end{cases}$$

L'application de l'algorithme d'Euler à ce système se fait composante par composante d'où :

$$\begin{cases} y_{i+1} = y_i + hf(t_i, y_i, z_i) \\ z_{i+1} = z_i + hg(t_i, y_i, z_i) \end{cases}$$

Cet algorithme s'applique aux équations différentielles d'ordre 2 (ou plus ...) à condition de les réduire au préalable à des systèmes différentiels du premier ordre, comme vu précédemment.

L'application d'autres algorithmes par exemple RK2 se fait moins simplement, voir exercices corrigés.

8.7 Enoncés des exercices corrigés

Exercice 70 :
Soit l'équation différentielle à condition initiale $y'(t) = y(t) + t$ et $y(0) = 1$. Approcher la solution de cette équation en $t = 1$ à l'aide de la méthode d'Euler en subdivisant l'intervalle de travail en 10 parties égales. Comparer à la solution exacte.

Exercice 71 :

1. Soit une suite $(u_n)_{n\in I\!N}$ de nombres réels positifs telle que $u_{n+1} \leq au_n + b$, a et b sont des constantes positives ; montrer que $u_n \leq a^n u_0 + b\frac{a^n-1}{a-1}$, $n \in I\!N$.
2. Montrer que $\forall n \in I\!N$ et $\forall x \in I\!R^+$, $(1+x)^n \leq e^{nx}$.
3. Utiliser les deux questions précédentes pour démontrer le théorème 10 du cours.
4. De même pour le théorème 11.

Exercice 72 :
Approcher la solution de l'équation différentielle ci-dessous en $t_1 = 0.2$ en utilisant RK2, avec un pas $h = 0.2$

$$y'(t) = y(t) - \frac{2t}{y} \quad et \quad y(0) = 1$$

Comparer à la solution exacte.

Exercice 73 :
En donnant les solutions de l'équation différentielle ci-dessous avec la condition initiale $y(0) = 1$ puis $y(0) = 1 + \epsilon$, ϵ réel non nul, vérifier qu'elle conduit à des schémas instables.

$$y'(t) = 36y(t) - 37e^{-t}.$$

Exercice 74 :
Soit le problème de Cauchy suivant

$$\begin{cases} \text{y'(t)} & = \text{t + y(t)}, \ t \in [0,1] \\ \text{y(0)} & = 1 \end{cases}$$

1. Trouver la solution exacte de ce problème.
2. Appliquer la méthode d'Euler à ce problème, avec $h = 0.1$, puis evaluer la solution en $t = 0.3$. Comparer à la solution exacte.

Exercice 75 :
Soit le problème de Cauchy suivant

$$\begin{cases} \text{y'(t)} & = 2\text{t - y(t)}, \ t \in [0,1] \\ \text{y(0)} & = 1 \end{cases}$$

1. Trouver la solution exacte de ce problème.
2. Appliquer la méthode d'Euler à ce problème, avec $h = 0.1$, puis evaluer la solution en $t = 0.3$. Comparer à la solution exacte.

Exercice 76 :
Ecrire l'equation différentielle modélisant le mouvement du pendule simple. Appliquer la méthode d'Euler puis celle de Taylor d'ordre 2.

Exercice 77 :
Soit l'équation différentielle du second ordre à conditions initiales :

$$(1) \ : \begin{cases} y''(t) + 2y'(t) = 2y(t), & t \in [a,b] \\ y(a) = 1 \quad \text{et} \quad y'(a) = 2 \end{cases}$$

1. Ecrire cette équation différentielle sous la forme d'un système différentiel de deux équations différentielles d'ordre un.
2. Appliquer la méthode de RK2 à ce système .

Exercice 78 :
On considère le problème différentiel

$$(1) \begin{cases} y''(t) & = & f(t, y(t), y'(t)), & t \in [a,b] \\ y(a) & = & \alpha \quad \text{et} \quad y(b) = \beta \end{cases}$$

On divise $[a,b]$ en m intervalles de longueur h, dans (1) on remplace $y'(x_n)$ par $\dfrac{y_{n+1} - y_{n-1}}{2h}$ et $y''(t)$ par $\dfrac{y_{n+1} - 2y_n + y_{n-1}}{h^2}$

1. Montrer que le problème (1) se transforme en

$$(2) \begin{cases} y_{n+1} & = & \Phi(y_n, y_{n-1}) + h^2 f_n \\ y_0 & = & \alpha \quad \text{et} \quad y_m = \beta \end{cases}$$

   avec $n = 1 \cdots m - 1$ et $f_n = f(t_n, y_n, \frac{y_{n+1} - y_{n-1}}{2h})$. Donner Φ .
2. Ecrire (2) sous forme matricielle.
3. On suppose maintenant que $f(t, y, y') = -\lambda y$; quel système linéaire obtient-on pour $y_0, \cdots, y_m$.

Exercice 79 :
Pour résoudre l'équation différentielle : $y' = f(t,y)$, où f est continue de $[a,b]X\mathbb{R}$ dans $\mathbb{R}$, on propose la méthode à un pas suivante :

$$\begin{cases} y_{n+1} = y_n + h\Phi(t_n, y_n, h) \\ \Phi(t,y,h) = \alpha f(t,y) + \beta f(t+\frac{h}{2}, y+\frac{h}{2}f(t,y)) + \gamma f(t+h, y+hf(t,y)) \end{cases}$$

(α, β, et γ sont des réels de $[0,1]$).

1. Pour quelles valeurs du triplet (α,β,γ) retrouve t-on la méthode d'Euler ?. Même question pour la méthode RK2 ?.
2. On suppose dans la suite de l'exercice que $f(t,y) \in C^2([a,b]X\mathbb{R})$ et L-lipschitzienne en y :
   (a) Pour quelles valeurs α, β, γ la méthode proposée est stable ?
   (b) Quelle relation doit satisfaire (α,β,γ) pour que la méthode soit consistante ?
   (c) Qu'on conclure pour la convergence ?

Exercice 80 :
On considère l'équation différentielle

$$(1) : \begin{cases} y''(x) - q(x)y(x) = f(x), \quad x \in [a,b] \\ \text{y(a)} = \alpha \quad \text{et} \quad \text{y(b)}=\beta \end{cases}$$

où q et f sont continues sur $[a,b]$ et $q(x) \geq 0$ pour tout $x \in [a,b]$. (Ces hypothèses assurent l'existence et l'unicité de la solution.)

1. Montrer que si y est quatre fois continument dérivable dans $[a,b]$ alors

   $$y"(x) = \frac{1}{h^2}[y(x+h) - 2y(x) + y(x-h)] + \frac{h^2}{12}y^{(4)}(\xi)$$

   avec $\xi \in]x-h, x+h[$.
2. On subdivise $[a,b]$ en N+1 intervalles de longueur $h = \frac{b-a}{N+1}$ et on pose $x_i = a + ih$, $i = 0, \cdots, N+1$, on appelle y_i une approximation de $y(x_i)$ et on remplace $y"(x_i)$ par $\frac{1}{h^2}(y_{i+1} - 2y_i + y_{i-1})$ dans (1). Ecrire le système d'équations linéaires ainsi obtenu. On appelle A la matrice de ce système.
3. Montrer que $A = L + h^2Q$ où L est une matrice tridiagonale et Q une matrice diagonale, donner L et Q.

Exercice 81 :
Soit l'équation différentielle du second ordre à conditions initiales :

$$(1) : \begin{cases} y''(x) + 3sin(y'(x)) = 2y(x), \quad x \in [a,b] \\ y(a) = 1 \quad \text{et} \quad y'(a) = 2 \end{cases}$$

1. Ecrire cette équation différentielle sous la forme d'un système différentiel de deux équations différentielles d'ordre un.
2. Appliquer la méthodede RK2 à ce système .
3. Appliquer la méthodede de Taylor d'ordre 2 à ce système .

8.8 Enoncés des exercices non corrigés

Exercice 82 :
Soit f une application de $I \times \mathbb{R}$ dans $\mathbb{R}$ où $I \subset \mathbb{R}$, $I = [a, b]$, et soit $\eta \in \mathbb{R}$.

1. Donner le théorème assurant l'existence et l'unicité de la solution du problème de Cauchy :
$$\begin{cases} y'(t) = f(t, y(t)), & t \in I \\ y(a) = \eta, \text{ donné dans } \mathbb{R} \end{cases}$$
2. Donner une condition nécéssaire et suffisante sur f pour qu'elle soit lipschitzienne en y uniformément par rapport à t
3. Quels sont parmi les problèmes ci dessous, ceux qui admettent une solution unique :
$$(1) : \begin{cases} y'(t) = \frac{-y}{t \ln t} + \frac{1}{\ln t}, & \forall t \in [e, 5] \\ y(e) = e \end{cases} \qquad (2) : \begin{cases} y'(t) = \sqrt{y}, & \forall t \in [0, 1] \\ y(0) = 0 \end{cases}$$

Exercice 83 :
Soit le problème de Cauchy suivant
$$\begin{cases} y'(t) & = t^2 + y, \quad t \in [0, 1] \\ y(0) & = 1 \end{cases}$$

1. Trouver la solution exacte de ce problème.
2. Appliquer la méthode d'Euler à ce problème, avec $h = 0.1$, puis evaluer la solution en $t = 0, 3$. Comparer à la solution exacte.

Exercice 84 :
Soit le problèlme de Cauchy suivant :
$$\begin{cases} y'(t) = -y + 2t, & \forall t \in]0, 1] \\ y(0) = 1 \end{cases}$$

1. Donner la solution générale de ce problème.
2. Appliquer la méthode d'Euler à ce problème avec $h = 0.2$, puis donner la solution numérique en $t = 0.4$, à 10^{-3} près.
3. Donner l'erreur théorique de la méhode d'Euler dans ce cas et la comparer à l'erreur effectivement commise. Commentaires ?

Exercice 85 :
Soit le problème de Cauchy suivant :
$$\begin{cases} y'(t) = -y + e^{-t} + e^t, & \forall t \in]0, 1] \\ y(0) = \frac{1}{2} \end{cases}$$

1. Montrer que la solution générale de ce problème est $y(t) = \frac{1}{2}e^t + te^{-t}$.
2. Appliquer la méthode d'Euler à ce problème avec
   (a) $h = 0.2$, puis donner la solution numérique en $t = 0.4$, à 10^{-3} près.
   (b) $h = 0.1$, puis donner la solution numérique en $t = 0.4$, à 10^{-3} près.
3. Donner l'erreur théorique de la méhode d'euler dans chacun de ces cas et la comparer à l'erreur effectivement commise. Commentaires.

Exercice 86 :
Soit le problème de Cauchy suivant :

$$(1) : \begin{cases} y'(t) = e^{-t} - 2y(t) = f(t,y), & t \in [0,1] \\ y(0) = 1 \end{cases}$$

1. Montrer que $f(t,y)$ est lipshitzienne par rapport à y uniformément par rapport à t, et donner une constante de Lipshitz.
2. Montrer que ce problème admet une solution unique.
3. Donner la solution exacte de (1), ainsi que $y(0.2)$.
4. Appliquer la méthode d'Euler à ce problème, écrire l'algorithme correspondant et donner l'approximation y_2 de y(0.2) obtenue à l'aide d'un pas de discrétisation numérique $h = 0.1$.
5. Rappeler l'erreur théorique de la méthode d'Euler et la comparer à l'erreur commise sur le calcul de $y(0.2)$; commentaires ? .

Exercice 87 :
Pour résoudre l'équation différentielle : $y' = f(t,y)$, où f est continue de $[a,b]\times\mathbb{R}$ dans $\mathbb{R}$, on propose la méthode à un pas suivante :

$$\begin{cases} y_{n+1} = y_n + h\Phi(t_n, y_n, h) \\ \Phi(t,y,h) = \alpha f(t,y) + \beta f(t + \frac{h}{2}, y + \frac{h}{2}f(t,y)) + \gamma f(t+h, y + hf(t,y)) \end{cases}$$

(α, β, et γ sont des réels de $[0,1]$).

1. Pour quelles valeurs du triplet (α, β, γ) retrouve t-on la méthode d'Euler ?. Même question pour la méthode RK2 ?.
2. On suppose dans la suite de l'exercice que $f(t,y) \in C^2([a,b]X\mathbb{R})$ et L-lipschitzienne en y :
   (a) Pour quelles valeurs α, β, γ la méthode proposée est stable ?
   (b) Quelle relation doit satisfaire (α, β, γ) pour que la méthode soit consistante ?

(c) et pour la convergence ?

Exercice 88 :
Soit le problème de Cauchy suivant

$$(1)\begin{cases} \text{y'(t)} & = \text{f(t,y(t)), } t \in [0,a], \text{ a réel strictement positif,} \\ \text{y(0)} & = \eta \quad \text{donné dans } \mathbb{R}, \end{cases}$$

où f est deux fois continûment dérivable dans $[0,a]X\mathbb{R}$, on suppose que $\frac{\partial f}{\partial y}$ est bornée. On approche numériquement la solution de (1) par le schéma à un pas :

$$(2)\begin{cases} y_{n+1} & = y_n + h\Phi(t_n, y_n, h), \quad n = 0, 1, \ldots N. \\ y_0 & = \eta \quad \text{donné dans } \mathbb{R} \end{cases}$$

où Φ est une application de $[0,a] \times \mathbb{R} \times [0,h_0]$ dans $\mathbb{R}$ (h_0 est un réel positif $\leq a$), $t_0 = 0, t_n = t_0 + nh$ et $h = \frac{a}{N}$.
On donne le théorème suivant qui pourra être utilisé sans démonstration :
Théorème : si f est deux fois continument dérivable dans $[0,a] \times \mathbb{R}$ et si Φ, $\frac{\partial \Phi}{\partial h}$ existent et sont continues dans $[0,a] \times \mathbb{R} \times [0,h_0]$ alors le schéma (2) est d'ordre 2 si et seulement si

$$\Phi(t,y,0) = f(t,y) \quad \text{et} \quad \frac{\partial \Phi(t,y,0)}{\partial h} = \frac{1}{2}(\frac{\partial f}{\partial t} + \frac{\partial f}{\partial y}.f)_{(t,y)} \ \ \forall (t,y) \in [0,a] \times \mathbb{R}.$$

Soit donc à approcher les solutions de (1) par la méhode (2), avec $\Phi(t,y,h) = \alpha K_1 + \beta K_3$ où

$$K_1 = f(t,y), \ K_2 = f(t + \frac{h}{3}, y + \frac{h}{3}K_1), \ K_3 = f(t + 2\frac{h}{3}, y + 2\frac{h}{3}K_2)$$

1. Quelle relation doit lier α et β pour que le schéma (2) soit consistant. ?
2. Est - il stable, pour quels α et β . ?
3. Déterminer α et β pour que ce schéma soit d'ordre 2.
4. En conclure sur la convergence de (2).

Exercice 89 :
Soit l'équation différentielle du second ordre à conditions initiales :

$$(1) : \begin{cases} y''(t) - 2y'(t) = -y(t), & t \in [0,1] \\ y(0) = 1 \quad \text{et} \quad y'(0) = -1 \end{cases}$$

1. Ecrire cette équation différentielle sous la forme d'un système différentiel de deux équations d'ordre un.

2. On applique la méthode d'Euler à ce système, écrire l'algorithme correspondant. Même question pour la méthode de RK2. (on notera h le pas numérique).
3. Donner la solution exacte de (1), ainsi que $y(t)$ et $y'(t)$ pour $t = 0.2$. Comparer ces deux résultats à ceux numériquement obtenus grâce au schéma d'Euler de la question précédente en prenant $h = 0.1$.

Exercice 90 :
Soit l'équation différentielle du second ordre à conditions initiales :

$$(1) : \begin{cases} y''(t) + 3y'(t) = -2y(t), & t \in [0, 1] \\ y(0) = 1 \quad \text{et} \quad y'(0) = 2 \end{cases}$$

1. Ecrire cette équation différentielle sous la forme d'un système différentiel de deux équations différentielles d'ordre un.
2. On applique la méthodede d'Euler à ce système,édrire l'algorithme correspondant. Même question pour la méthode de RK2. (on notera h le pas numéique).
3. Donner la solution exacte de (1), ainsi que $y(t)$ et $y'(t)$ pour $t = 0.2$. Comparer ces deux résultats à ceux numériquement obtenus grâce au schéma d'Euler de la question précédente en prenant $h = 0.1$.

Exercice 91 :
Soit l'équation différentielle du second ordre à conditions initiales :

$$(1) : \begin{cases} y''(t) - 2y'(t) = -y(t), & t \in [0, 1] \\ y(0) = -1 \quad \text{et} \quad y'(0) = +3 \end{cases}$$

1. Ecrire cette équation différentielle sous la forme d'un système différentiel de deux équations d'ordre un.
2. On applique la méthode d'Euler à ce système, écrire l'algorithme correspondant. Même question pour la méthode de RK2, (on notera h le pas numérique).
3. Donner la solution exacte de (1), ainsi que $y(t)$ et $y'(t)$ pour $t = 0.2$. Comparer ces deux résultats à ceux numériquement obtenus grâce au schéma d'Euler de la question précédente en prenant $h = 0.1$.

Exercice 92 :
Soit l'équation différentielle du second ordre à conditions initiales :

$$(1) : \begin{cases} y''(t) + 4y'(t) = 2y(t), & t \in [0, 1] \\ y(0) = 1 \quad \text{et} \quad y'(0) = -2 \end{cases}$$

1. Ecrire cette équation différentielle sous la forme d'un système différentiel de deux équations d'ordre un.
2. On applique la méthode d'Euler à ce système, écrire l'algorithme correspondant. Même question pour la méthode de RK2, (on notera h le pas numérique).
3. *Question Facultative :* Donner la solution exacte de (1), ainsi que $y(t)$ et $y'(t)$ pour $t = 0.2$. Comparer ces deux résultats à ceux numériquement obtenus grâce au schéma d'Euler de la question précédente en prenant $h = 0.1$.

Exercice 93 :
Soit f une application de $I \times \mathbb{R}$ dans $\mathbb{R}$ où $I \subset \mathbb{R}$, $I = [a, b]$, et soit $\eta \in \mathbb{R}$.

1. Donner le théorème assurant l'existence et l'unicité de la solution du problème de Cauchy :

$$\begin{cases} y'(t) = f(t, y(t)), & t \in I \\ y(a) = \eta, \text{ donné dans } \mathbb{R} \end{cases}$$

2. Donner une condition nécéssaire et suffisante sur f pour qu'elle soit lipschitzienne en y uniformément par rapport à t
3. Quels sont parmi les problèmes ci dessous, ceux qui admettent une solution unique :

$$(1) : \begin{cases} y'(t) = \frac{-y}{t \ln t} + \frac{1}{\ln t}, & \forall t \in [e, 5] \\ y(e) = e \end{cases} \qquad (2) : \begin{cases} y'(t) = \sqrt{y}, & \forall t \in [0, 1] \\ y(0) = 0 \end{cases}$$

Exercice 94 :
Soit l'équation différentielle du second ordre à conditions initiales :

$$(1) : \begin{cases} \text{y"(x) +2y'(x) =2y(x)}, & x \in [0, 1] \\ \text{y(0) = 1} \quad \text{et } y'(0) = 1 \end{cases}$$

1. Ecrire cette équation différentielle sous la forme d'un système différentiel de deux équations différentielles d'ordre un.
2. Appliquer la méthodede RK2 à ce système .

Exercice 95 :
Soit l'équation différentielle du troisième ordre et à conditions initiales :

$$(1) : \begin{cases} y'''(t) = 3y(t), t \in [0, 1] \\ y(0) = 0 \quad y'(0) = -2 \quad \text{et } y''(0) = -1. \end{cases}$$

1. Ecrire cette équation différentielle sous la forme d'un système différentiel de trois équations différentielles chacune d'ordre un.
2. On applique la méthode d'Euler à ce système, écrire l'algorithme correspondant. (On notera h le pas d'intégration.)

Exercice 96 :
Soit le problème de Cauchy suivant :

$$(1) : \begin{cases} y'(t) = \sin(t) - y(t) = f(t,y), & t \in [0,1] \\ y(0) = 1 \end{cases}$$

1. Montrer que $f(t,y)$ est lipshitzienne par rapport à y uniformément par rapport à t, et donner une constante de Lipshitz.
2. Montrer que ce problème admet une solution unique.
3. Donner la solution exacte de (1), ainsi que $y(0.2)$.
4. Appliquer la méthode d'Euler à ce problème, écrire l'algorithme correspondant et donner l'approximation y_2 de y(0.2) obtenue à l'aide d'un pas de discrétisation numérique $h = 0.1$.
5. Rappeler l'erreur théorique de la méthode d'Euler et la comparer à l'erreur commise sur le calcul de $y(0.2)$. Commentaires ?

Exercice 97 :
On considère la méthode à pas séparés suivante (*) :

$$y_{n+1} = y_n + \frac{h}{2}[f(t_n, y_n) + f(t_{n+1}, y_n + hf(t_n, y_n))]$$

1. Etudier la consistance, la stabilité théorique, la convergence.
2. On veut calculer une valeur approchée de $I = \int_a^b f(t)dt$.
   Montrer que ce problème peut être remplacé par la résolution d'un problème de Cauchy dont on précisera la condition initiale, on posera $y(x) = \int_a^x f(t)dt$
   Comment peut-on obtenir une valeur approchée $\tilde{I}$ de I. ?
   Si on utilise la méthode (*) pour calculer $\tilde{I}$, donner l'expression de $\tilde{I}$, exprimer l'erreur $\tilde{I} - I$ en fonction de h.

8.9 Corrigés des exercices

exercice 39

$$\begin{cases} y'(t) &= y(t) + t = f(t,y) \\ y(0) &= 1 \end{cases} \quad (1)$$

L'intervalle d'intégration est $[0,1]$.

Remarquons tout d'abord que f étant continue et lipshitzienne par rapport à y le problème de Cauchy (1) admet une solution unique (théorème 9 de Cauchy-Lipshitz).

Méthode d'Euler Elle s'écrit :

$$\begin{aligned} y_{n+1} &= y_n + hf(t_n, y_n) \\ &= y_n + h(t_n + y_n) \\ &= y_n(1+h) + ht_n \end{aligned}$$

On a aussi $y(0) = y_0 = 1$, $h = \frac{1-0}{10} = 0.1$ $t_0 = 0$ et $t_n = t_0 + nh = \frac{n}{10}$.
D'où le tableau,

n	0	1	2	3	4	5	6	7	8	9	10
t_n	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
y_n	1	1.1	1.22	1.362							3.1874

C'est à dire que l'approximation en $t = 1$ de $y(t)$, est $y_{10} = 3.1874$.

Solution exacte de cette équation Appliquons la méthode de la variation de la constante.

1ère étape : équation sans second membre.

$$\frac{dy(t)}{dt} = y'(t) = y$$

$y = 0$ est une solution evidente. Les autres solutions sont données par $\frac{dy}{y} = dt$. D'où $y = ke^t$ avec $k \in \mathbb{R}$ (2).

2nde étape : une solution particulière On applique la méthode de la variation de la constante $y = ke^t$ d'où $y' = k'e^t + ke^t$ que l'on reporte dans (1) : $k'e^t + ke^t = ke^t + t$ ainsi, $k' = te^{-t}$.
$k = \int te^{-t}\,dt$ en intégrant par parties on trouve

$$\begin{aligned} k &= -te^{-t} + \int e^{-t}\,dt \\ &= -te^{-t} - e^{-t} + c \\ &= e^{-t}(-1-t) + c \qquad (3) \end{aligned}$$

avec $c \in \mathbb{R}$.

3ème étape : solution générale. On remplace k donné par (3) dans (2) :

$$y = \big(e^{-t}(-1-t) + c\big)e^{t}$$

donc,

$$y = -1 - t + ce^{t}$$

Finalement, grâce à la condition initiale $y(0) = 1$, on détermine c, d'où

$$y(0) = 1 = -1 - 0 + ce^{0} \Rightarrow c = 2$$

Ainsi, la solution exacte de (1) est $y = -1 - t + 2e^{t}$.

Estimation de l'erreur. La solution exacte ci-dessus donne $y(1) = -1-1+2e = 3.4366$. Ainsi, l'erreur effectivement commise lors de l'application de la méthode d'Euler est $|E_e| = |3.4366 - 3.1874| = 0.25$. Cherchons l'erreur théorique qui est donnée par :

$$E_t \leq (e^{L(b-a)} - 1)\frac{M_2 \times h}{2L}$$

Où $M_2 = max_{[a,b]}|y''(t)|$ et L est la constante de Lipschitz de f par rapport à y, qui se calcule aisément :

$$|f(t,y) - f(t,z)| = |y - z| \Rightarrow L = 1.$$

De même, on a

$$\begin{aligned} y''(t) &= 1 + t + y \\ &= 1 + t + (-1 - t + 2e^{t}) \\ &= 2e^{t} \end{aligned}$$

Ainsi $M_2 = 2e$. Donc,

$$\begin{aligned} |E_t| &\leq (e^{1(1-0)} - 1)\frac{2e10^{-1}}{2 \times 1} \\ &\leq (e-1)\frac{e}{10} = 0.4673 \end{aligned}$$

Clairement, $|E_e| \leq |E_t|$, donc la méthode d'Euler donne une bonne approximation de la solution de ce problème de Cauchy en $t = 1$.

exercice 42

$y'(t) = y(t) - \frac{2t}{y} = f(t,y)$ et $y(0) = 1$

Remarquons tout d'abord que f étant continue et lipshitzienne par rapport à y ce problème de Cauchy admet une solution unique (théorème 9 de Cauchy-Lipshitz).

L'intervalle d'intégration est $[0, 0.2]$ et le pas d'intégration est $h = 0.2$.

Méthode de RK2 Elle s'écrit :

$$y_{n+1} = y_n + \frac{h}{2}(M_1 + M_2),$$

soit

$$y_1 = y_0 + \frac{h}{2}(M_1 + M_2)$$

, avec $M_1 = f(0.1) = 1$ et $M_2 = f(0.2, 1 + 0.2 \cdot 1)$. Donc $f(0.2, 1.2) = 0.8666$.

Ainsi, l'approximation en $t = 0.2$ de $y(t)$, est

$$\begin{aligned} y_1 &= 1 + \frac{h}{2}(M_1 + M_2) = 1 + 0.1(1.8666) \\ y_1 &= 1.18666 \end{aligned}$$

Solution exacte de cette équation

$$(1) \quad y' = y - \frac{2t}{y}, \qquad y \neq 0$$

En multipliant (1) par y on a $y'y = y^2 - 2t$ d'où $\frac{1}{2}(y^2) = y^2 - 2t$. On pose donc $u = y^2$ d'où :

$$(2) \quad \frac{1}{2}u' = u - 2t$$

Ce qui est une équation différentielle linéaire du 1^{er} ordre. On l'intègre par la méthode de la variation de la constante comme à l'exercice précédent.

L'équation sans second membre est $u' = 2u$ de solution $u = 0$ ou $u = ke^{2t}$.

Une solution particulière par la variation de la constante. On a

$$(3) \quad u' = \lambda' e^{2t} + 2\lambda e^{2t}$$

avec $\lambda \in \mathbb{R}$. D'où, dans (2) $\frac{1}{2}\lambda' e^{2t} = -2t$ ce qui implique $\lambda' = -4te^{-2t}$.

Intégrons λ par parties :

$$\begin{aligned}\lambda &= -4\left[-\frac{1}{2}te^{-2t}+\frac{1}{2}\int e^{-2t}dt\right]\\ &= 2te^{-2t}+e^{-2t}+c\\ &= (2t+1)e^{-2t}+c\end{aligned}$$

avec $c \in \mathbb{R}$. La solution générale est donc $u = 2t+1+ce^{2t}$ comme $y(0) = 1$, $c = 0$. Finalement, $u = y^2 = 2t+1$. Ainsi, $y = \pm\sqrt{2t+1}$. Comme $y(0) = 1 > 0$ alors $y = \sqrt{2t+1}$.

Estimation de l'erreur. La solution exacte est $y(0.2) = \sqrt{2\cdot 0.2+1} = 1.18321$. Donc l'erreur commise est $|E_e| = |1.18321 - 1.18666| = 0.00345$. On peut comparer cette erreur effective à l'erreur théorique sur RK2, donnée par : ————-
(le théorème du cours)

exercice 43

Soit le problème de Cauchy suivant

$$\left\{\begin{array}{lcl} y'(t) &=& 2t - y(t) = f(t,y)\\ y(0) &=& 1\end{array}\right. \quad (1)$$

Remarquons tout d'abord que f étant continue et lipshitzienne par rapport à y le problème de Cauchy (1) admet une solution unique (théorème 9 de Cauchy-Lipshitz).

Méthode d'Euler. Elle s'écrit :

$$\begin{aligned} y_{n+1} &= y_n + hf(t_n, y_n)\\ &= y_n + h(2t_n - y_n)\\ &= y_n(1-h) + 2ht_n\end{aligned}$$

On a aussi $y(0) = y_0 = 1$, $h == 0.1$ $t_0 = 0$ et $t_n = nh$.
Donc $y_{n+1} = 0.9y_n + 0.02n$, d'où le tableau,

n	0	1	2	3
t_n	0	0.1	0.2	0.3
y_n	1	0.92	0.868	0.8412

C'est à dire que l'approximation en $t = 0.3$ de $y(t)$ avec le pas $h = 0.1$, est $y_{10} = 0.8412$.

Solution exacte de cette équation. En appliquant la méthode de la variation de la constante, comme au premier exercice, on trouve la solution générale, $y(t) = 2t - 2 + 3e^{-t}$.

Estimation de l'erreur. La solution exacte ci-dessus donne $y(0.3) = 0.822$. Ainsi, l'erreur effectivement commise lors de l'application de la méthode d'Euler est $|E_e| = |0.822 - 0.841| = 0.019$. Cherchons l'erreur théorique qui est donnée par :

$$E_t \leq (e^{L(b-a)} - 1)\frac{M_2 \times h}{2L}$$

Où $M_2 = max_{[a,b]}|y''(t)|$ et L est la constante de Lipschitz de f par rapport à y, qui est ici clairement égale à 1. On a

$$y''(t) = 3e^{-t}.$$

Ainsi $M_2 = 3$. Donc,

$$\begin{aligned} |E_t| &\leq (e^{1(0.3-0)} - 1)\frac{3 \times 10^{-1}}{2 \times 1} \\ &\leq 0.15(e^{0.3} - 1) \approx 0.05247. \end{aligned}$$

Clairement, $|E_e| \leq |E_t|$, donc la méthode d'Euler donne une bonne approximation de la solution de ce problème de Cauchy en $t = 0.3$.

exercice 44

$$(1) \begin{cases} y'(t) &= 36y(t) - 37e^{-t} \\ y(0) &= 1, \text{ puis } 1 + \varepsilon \end{cases}$$

En appliquant la méthode de la variation de la constante, comme au premier exercice, on trouve la solution générale, $y(t) = (e^{-37} + c)e^{36t} = e^{-t} + ce^{36t}$ où c est la constante d'intégration.

1. Si $y(0) = 1$, alors $c = 0$, donc la solution du problème est $y(t) = e^{-t}$.
2. Si $y(0) = 1 + \varepsilon$, alors $c = \varepsilon$, donc la solution du problème est $y_\varepsilon(t) = e^{-t} + \varepsilon e^{36t}$.

Conclusion : En comparant $y(t)$ et $y_\varepsilon(t)$, on voit que la différence $|y(t) - y_\varepsilon(t)| = \varepsilon e^{36t}$. Même si ε est très petit, cet écart tend vers $+\infty$, les deux solutions divergent l'une de l'autre. Ce problème est donc très sensible aux Conditions Initiales.

exercice 45

$$ay''(t) + g \sin y(t) = 0 \qquad (1)$$

Écrivons (1) sous la forme de deux équations différentielles d'ordre 1 chacune. Pour cela on pose $y' = z$, d'où $y'' = z' = -\frac{g}{a} \sin(y)$. Ainsi, on obtient le système :

$$\left\{ \begin{array}{lcll} y' & = & z & f(t,y,z) \\ z' & = & -\frac{g}{a} \sin(y) & g(t,y,z) \end{array} \right.$$

1. Appliquons Euler :

$$\left\{ \begin{array}{lcl} y_{n+1} & = & y_n + hf(t_n, y_n, z_n) \\ z_{n+1} & = & z_n + hg(t_n, y_n, z_n) \end{array} \right.$$

   Avec $f(t,y,z) = y$ et $g(t,y,z) = -\frac{g}{a} \sin(y(t))$, soit

$$\left\{ \begin{array}{lcl} y_{n+1} & = & y_n + hz_n \\ z_{n+1} & = & z_n - h\frac{g}{a} \sin y_n, \end{array} \right.$$

   avec y_0 et z_0 donnés.

2. Méthode de Taylor d'ordre 2, elle s'écrit :

$$U_{n+1} = U_n + hF(t_n, U_n) + \frac{h^2}{2} \left(\frac{\partial F}{\partial t}(t_n, U_n) + \frac{\partial F}{\partial U}(t_n, U_n).f(t_n, U_n) \right)$$

   On obtient donc,

$$\left\{ \begin{array}{lcl} y_{n+1} & = & y_n + h(z_n) + \frac{h^2}{2} \left(-\frac{g}{a} \sin(y_n) + 1z_n \right) \\ z_{n+1} & = & z_n + h \left(-\frac{g}{a} \sin(y_n) \right) + \frac{h^2}{2} \left(-\frac{g}{a} \cos(y_n) z_n + 0 \right) \end{array} \right.$$

exercice 48

$$(1) \left\{ \begin{array}{lcl} y''(t) + 2y'(t) & = & 2y(t) \\ y(a) & = & 1 \\ y'(a) & = & 2 \end{array} \right.$$

Avec $t \in [a, b]$.

1. On pose $y' = z$ d'où $y'' = z' = -2z + 2y$. On trouve le système

$$(2) \left\{ \begin{array}{lcl} y' & = & z \\ z' & = & 2y - 2z, \end{array} \right.$$

avec $t \in [a,b]$ et $y(a)=1$ et $z(a)=1$.
Matriciellement ce système s'écrit,

$$\begin{pmatrix} y \\ z \end{pmatrix}' = \begin{pmatrix} 0 & 1 \\ 2 & -2 \end{pmatrix} \begin{pmatrix} y \\ z \end{pmatrix}$$

ou $U' = F(t,U)$ avec $U = \binom{y}{z}$ et F l'endomorphisme associé à la matrice

$$A = \begin{pmatrix} 0 & 1 \\ 2 & -2 \end{pmatrix}, \qquad \begin{array}{rccl} F: & \mathbb{R}^2 & \to & \mathbb{R} \\ & (y,z) & \mapsto & F(t,y,z). \end{array}$$

Ce système peut donc sécrire,

$$\left| \begin{array}{lcl} U' & = & AU \\ U(a) & = & \binom{y(a)}{z(a)} = \binom{1}{2} \end{array} \right.$$

2. a) Appliquons la méthode d'Euler.
Elle s'écrit $U_{n+1} = U_n + hF(t_n, U_n)$. Appliquée à (2) elle s'écrit :

$$\begin{array}{rcl} U_{n+1} & = & U_n + hAU_n \\ & = & U_n + h \begin{pmatrix} 0 & 1 \\ 2 & -2 \end{pmatrix} U_n \\ & = & \binom{y_n}{z_n} + h \begin{pmatrix} 0 & 1 \\ 2 & -2 \end{pmatrix} \binom{y_n}{z_n} \\ & = & \binom{y_n}{z_n} + h \begin{pmatrix} 3n \\ 2y_n - 2z_n \end{pmatrix} \end{array}$$

$$\left\{ \begin{array}{lcl} y_{n+1} & = & y_n + hz_n \\ z_{n+1} & = & z_n + 2h(y_n - z_n) \end{array} \right.$$

avec $y_0 = 1$ et $z_0 = 2$.
b) Appliquons la méthode de RK2, elle s'écrit :

$$U_{n+1} = U_n + \frac{h}{2}(M_1 + M_2)$$

avec $M_1 = F(t_n, U_n)$ et $M_2 = F(t_n + h, U_n + hF(t_n, U_n))$. Calculons M_1 :

$$M_1 = F(t_n, U_n) = AU_n = \begin{pmatrix} 0 & 1 \\ 2 & -2 \end{pmatrix} \binom{y_n}{z_n} = \binom{z_n}{2y_n - 2z_n}$$

Calculons M_2 :

$$\begin{aligned} M_2 &= F(t_n + h, U_n + hM_1) \\ &= A(U_n + hM_1) \\ &= \begin{pmatrix} 0 & 1 \\ 2 & -2 \end{pmatrix} \left[\begin{pmatrix} y_n \\ z_n \end{pmatrix} + h \begin{pmatrix} z_n \\ 2y_n - 2z_n \end{pmatrix} \right] \\ &= \begin{pmatrix} 0 & 1 \\ 2 & -2 \end{pmatrix} \begin{pmatrix} y_n + hz_n \\ z_n + 2h(y_n - z_n) \end{pmatrix} \\ &= \begin{pmatrix} z_n + 2h(y_n - z_n) \\ 2(y_n + hz_n) + (6h - 2)z_n \end{pmatrix} \\ &= \begin{pmatrix} 2hy_n + z_n(1 - 2h \\ (2 - 4h)y_n + (6h - 2)z_n \end{pmatrix} \end{aligned}$$

Ainsi,

$$\begin{aligned} U_{n+1} &= \begin{pmatrix} y_{n+1} \\ z_{n+1} \end{pmatrix} = \begin{pmatrix} y_n \\ z_n \end{pmatrix} + \frac{h}{2} + (M_1 + M_2) \\ &= \begin{pmatrix} y_n \\ z_n \end{pmatrix} + \frac{h}{2} \begin{pmatrix} z_n + 2hy_n + z_n(1 - 2h) \\ 2y_n - 2z_n + (2 + 4h)y_n + (6h - 2)z_n \end{pmatrix}. \end{aligned}$$

Donc

$$\begin{aligned} y_{n+1} &= y_n(1 + h^2) + z_n(\frac{h}{2} + \frac{h}{2}(1 - 2h)) \\ z_{n+1} &= y_n(2 + z_h) + z_n(1 - h + (3h - 1)h) \end{aligned}$$

Par conséquent

$$\begin{aligned} y_{n+1} &= (1 + h^2)y_n + h(1 - h)z_n \\ z_{n+1} &= 2h(1 - h)y_n + (3h^2 - 2h + 1)z_n. \end{aligned}$$

8.10 Mise en œuvre en Java

On présente dans la suite une mise en œuvre des méthodes de résolution numérique des équations différentielles, puis des systèmes différentiels. On utilise dans les deux cas, les méthodes d'Euler et de Runge-Kutta d'ordre 2 et 4.

8.10.1 Résolution numérique des équations différentielles

Nous décomposons nos programmes en plusieurs classes :

- une classe abstraite de description de l'équation différentielle. Elle permet d'être utilisée d'une manière générique dans les méthodes et programmes de traitement ;
- une classe abstraite décrivant un processus d'itération de résolution qui pourra être utilisée dans des classes générique de traitement. Elle est dérivée en trois classes de processus, la méthode d'Euler et les méthodes de Runge-Kutta d'odre 2 et 4 ;
- une classe de traitement générique qui décrit un processus itératif de calcul de solution et une représentation graphique associée, ceci indépendamment de l'équation différentielle et du processus utilisés ;
- et finalement, une classe décrivant une équation différentielle particulière et la classe contenant le programme principal qui met en œuvre le traitement générique précédent auquel on transmettra une instance de la classe particulière construite et le processus choisis.

Une classe abstraite de description de l'équation

La classe abstraite suivante fait référence à une équation différentielle qui doit être décrite dans la méthode `calcul`.

```
abstract class FoncEDO{
    public double t0;
    public double y0;
    public double pas;
    public int nbpas;

    FoncEDO(double pt0, double py0, double ppas, int pnbpas) {
        t0=pt0; y0=py0; pas=ppas; nbpas=pnbpas;
    }

    abstract public double calcul (double t, double x);
    abstract public String libelle();
```

```
}
```

Des classes de description des méthodes

Nous décrivons ici une classe abstraite faisant référence à un processus de résolution numérique d'équation différentielle qui doit être décrite dans la méthode `iter`.

```
abstract class IterEDO {
    FoncEDO f;
    double h;
    double yCourant;
    double tCourant;

    IterEDO(FoncEDO fonc, double pas, double y0, double t0) {
        f=fonc; h=pas; yCourant=y0; tCourant=t0;
    }

    public void set_yCourant (double y0) { yCourant=y0; }
    public void set_tCourant (double t0) { tCourant=t0; }

    abstract public double iter();

    public double iter (double t, double y) {
        set_tCourant(t); set_yCourant(y); return iter();
    }
}
```

La classe suivante dérive de la précédente et décrit le processus de résolution basé sur la méthode d'Euler.

```
class IterEuler extends IterEDO {

    IterEuler (FoncEDO fonc, double pas, double y0, double t0) {
        super(fonc, pas, y0, t0);
    }

    public double iter() {
        yCourant = yCourant + h*f.calcul(tCourant, yCourant);
        tCourant += h;
        return yCourant;
```

```
    }
}
```

La classe suivante implémente la méthode de Runge-Kutta d'ordre 2.

```
class IterRK2 extends IterEDO {

    IterRK2 (FoncEDO fonc, double pas, double y0, double t0) {
        super(fonc, pas, y0, t0);
    }

    public double iter() {
        double k1 = f.calcul(tCourant, yCourant);
        tCourant += h;
        double k2 = f.calcul(tCourant, yCourant + h * k1);
        yCourant = yCourant + h * ( k1 + k2 )/2;
        return yCourant;
    }
}
```

La classe suivante implémente la méthode de Runge-Kutta d'ordre 4.

```
class IterRK4 extends IterEDO {

    IterRK4 (FoncEDO fonc, double pas, double y0, double t0) {
        super(fonc, pas, y0, t0);
    }

    public double iter() {
        double k1 = f.calcul(tCourant, yCourant);
        tCourant += 0.5*h;
        double k2 = f.calcul(tCourant, yCourant + 0.5*h * k1);
        double k3 = f.calcul(tCourant, yCourant + 0.5*h * k2);
        tCourant += 0.5*h;
        double k4 = f.calcul(tCourant, yCourant + h * k3);
        yCourant = yCourant + h * (k1 + 2*k2 + 2*k3 + k4)/6;
        return yCourant;
    }
}
```

Traitement générique

Nous décrivons maintenant un processus générique qui effectue un certain nombre d'itérations à partir d'un processus de résolution arbitraire - de type `IterEDO`. Une représentation graphique est réalisée. Cette dernière utilise la méthode `CanvasGraphe` définies dans les chapitres précédents.

```
import java.lang.*;
import java.io.*;
import java.util.*;
import java.awt.*;

class TraitFEDO {
    TraitFEDO(FoncEDO fedo, IterEDO methode) {
        double t0=fedo.t0, y0=fedo.y0, pas=fedo.pas;
        int nbpas=fedo.nbpas;
        double[] tc = new double[nbpas];
        double[] yc = new double[nbpas];
        tc[0]=t0; yc[0]=y0;
        for (int i=1; i<nbpas; i++) {
            tc[i] = tc[i-1]+pas;
            yc[i] = methode.iter();
        }

        // calcul des extrema du tableau yc :
        MaxminTabDouble myc = new MaxminTabDouble(yc);
        double maxyc = myc.getmaxi();
        double minyc = myc.getmini();

        // representation graphique du calcul :
        DomaineDouble dr = new DomaineDouble(tc[0], minyc, tc[nbpas-1], max
        DomaineInt de = new DomaineInt(0, 0, 600, 450);
        CanvasGraphe cg = new CanvasGraphe(dr, de, tc, yc);
        FenetreGraphe x = new FenetreGraphe(de, cg, fedo.libelle());
        x.show();
    }
}
```

Exemple d'utilisation

Nous décrivons maintenant une utilisation des classes précédentes. Nous définissons l'équation différentielle à résoudre dans la classe `FEDO` et nous créons

une instance du traitement générique précédent auquel nous transmettons des instances de la classe FEDO et d'un processus de résolution.

L'équation différentielle résolue ici est :

$$\begin{cases} \frac{dy}{dt} = \sin(2t - y) \\ y(0) = 0.5 \end{cases}$$

```
class FEDO extends FoncEDO {

    FEDO (double pt0, double py0, double ppas, int pnbpas) {
        super(pt0, py0, ppas, pnbpas);
    }

    public double calcul(double t, double y) {
        return Math.sin(2*t-y);
    }
    public String libelle() {return "dy/dt = sin(2t-y)"; }
}

class PrgEdo{

    public static void main(String args[]) {
        double t0=0, y0=0.5, pas=0.1;
        int nbpas=100;
        FEDO f = new FEDO(t0, y0, pas, nbpas);
        IterEDO methode = new IterRK4(f, pas, y0, t0);
        TraitFEDO work = new TraitFEDO(f, methode);
    }
}
```

La solution obtenue est tracée dans la figure 8.1.

8.10.2 Résolution numérique des systèmes différentiels

Nous suivons la même construction pour écrire un ensemble de classes de résolution numérique de systèmes différentiels.

Une classe abstraite de description du système

La classe abstraite suivante fait référence à un système qui sera décrit dans la méthode `calcul`.

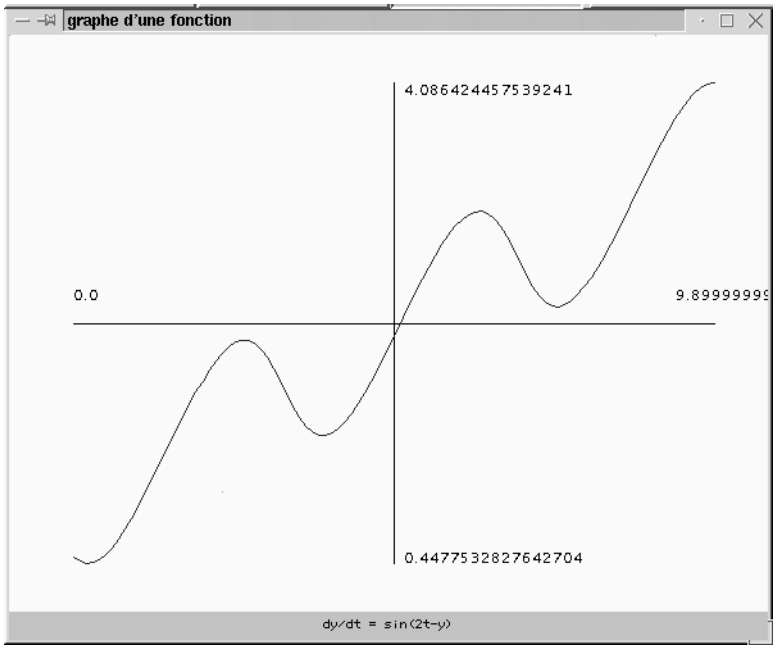

FIG. 8.1: Résolution numérique d'une équation différentielle

```
abstract class SysEDO {
    public int dim;
    public double t0;
    public double[] y0;
    public double pas;
    public int nbpas;
    public int ixGraph;
    public int iyGraph;

    SysEDO(int pdim, double pt0, double[] py0, double ppas, int pnbpas,
           int pixGraph, int piyGraph) {
        dim=pdim; t0=pt0; y0=py0; pas=ppas; nbpas=pnbpas;
        ixGraph=pixGraph; iyGraph=piyGraph;
    }

    abstract public double[] calcul (double t, double[] x);
    abstract public String libelle();
}
```

Des classes de description des méthodes

La classe suivante est une construction abstraite qui fait référence à un processus itératif de résolution de système décrit par la méthode `iter`.

```
abstract class IterSysEDO {
    SysEDO f;
    double h;
    public double[] yCourant;
    public double tCourant;
    int dim;

    IterSysEDO(SysEDO fonc, double pas, double[] y0, double t0) {
        f=fonc; h=pas; yCourant=y0; tCourant=t0; dim=y0.length;
    }

    public void set_yCourant (double[] y0) { yCourant=y0; }
    public void set_tCourant (double t0) { tCourant=t0; }

    abstract public double[] iter();

    public double[] iter (double t, double[] y) {
        set_tCourant(t); set_yCourant(y); return iter();
    }
}
```

On construit une classe qui en dérive et qui décrit la méthode d'Euler pour un système.

```
class IterSysEuler extends IterSysEDO {

    IterSysEuler (SysEDO fonc, double pas, double[] y0, double t0) {
        super(fonc, pas, y0, t0);
    }

    public double[] iter() {
        int i;
        double[] yNew = new double[dim];
        for (i=0; i<dim; i++)
            yNew[i] = yCourant[i] + h*(f.calcul(tCourant, yCourant))[i];
        for (i=0; i<dim; i++) {
          yCourant[i] = yNew[i];
```

```
          //System.out.println("yCourant["+i+"] = "+yCourant[i]);
        }
        //yCourant=yNew;
        tCourant += h;
        return yNew;
    }
}
```

La classe suivante implémente la méthode de Runge-Kutta d'ordre 2 pour un système.

```
class IterSysRK2 extends IterSysEDO {

    IterSysRK2 (SysEDO fonc, double pas, double[] y0, double t0) {
        super(fonc, pas, y0, t0);
    }

    public double[] iter() {
        int i;
        double[] k1 = new double[dim];
        double[] k2 = new double[dim];
        double[] yNew = new double[dim];
        for (i=0; i<dim; i++) {
            k1[i] = (f.calcul(tCourant, yCourant))[i];
            yNew[i] = yCourant[i] + h*k1[i];
        }
        for (i=0; i<dim; i++)
            k2[i] = (f.calcul(tCourant+h, yNew))[i];
        for (i=0; i<dim; i++)
            yNew[i] = yCourant[i]+0.5*h*(k1[i]+k2[i]);
        for (i=0; i<dim; i++)
            yCourant[i] = yNew[i];
        tCourant += h;
        return yNew;
    }
}
```

La classe suivante implémente la méthode de Runge-Kutta d'ordre 4 pour un système.

```
class IterSysRK4 extends IterSysEDO {
```

```
    IterSysRK4 (SysEDO fonc, double pas, double[] y0, double t0) {
        super(fonc, pas, y0, t0);
    }

    public double[] iter() {
        int i;
        double[] k1 = new double[dim];
        double[] k2 = new double[dim];
        double[] k3 = new double[dim];
        double[] k4 = new double[dim];
        double[] yNew = new double[dim];
        for (i=0; i<dim; i++) {
            k1[i] = (f.calcul(tCourant, yCourant))[i];
            yNew[i] = yCourant[i] + 0.5*h*k1[i];
        }
        for (i=0; i<dim; i++) {
            k2[i] = (f.calcul(tCourant+0.5*h, yNew))[i];
            yNew[i] = yCourant[i] + 0.5*h*k2[i];
        }
        for (i=0; i<dim; i++) {
            k3[i] = (f.calcul(tCourant+0.5*h, yNew))[i];
            yNew[i] = yCourant[i] + h*k3[i];
        }
        for (i=0; i<dim; i++)
            k4[i] = (f.calcul(tCourant+h, yNew))[i];
        for (i=0; i<dim; i++)
            yNew[i] = yCourant[i]+h/6*(k1[i]+2*k2[i]+2*k3[i]+k4[i]);
        for (i=0; i<dim; i++)
            yCourant[i] = yNew[i];
        tCourant += h;
        return yNew;
    }
}
```

Traitement générique

On adapte le traitement générique précédent au cas d'un système différentiel.

```
import java.lang.*;
import java.io.*;
import java.util.*;
import java.awt.*;
```

```
class TraitSEDO {
    TraitSEDO(SysEDO fedo, IterSysEDO methode) {
        int dim=fedo.dim;
        double t0=fedo.t0;
        double[] y0=fedo.y0;
        double pas=fedo.pas;
        int nbpas=fedo.nbpas;
        int ixGraph=fedo.ixGraph;
        int iyGraph=fedo.iyGraph;
        int i;
        double[] tc = new double[nbpas];
        double[][] yc = new double[nbpas][dim];
        tc[0]=t0;
        for (i=0; i<dim; i++)
            yc[0][i]=y0[i];
        for (i=1; i<nbpas; i++) {
            tc[i] = tc[i-1]+pas;
            yc[i] = methode.iter();
        }

        double[] xgraph = new double[nbpas];
        double[] ygraph = new double[nbpas];
        for (i=0; i<nbpas; i++) {
            if (ixGraph != -1)
                xgraph[i] = yc[i][ixGraph];
            else
                xgraph[i] = tc[i];
            ygraph[i] = yc[i][iyGraph];
        }

        // calcul des extrema des tableaux :
        MaxminTabDouble mxg = new MaxminTabDouble(xgraph);
        MaxminTabDouble myg = new MaxminTabDouble(ygraph);
        double maxx = mxg.getmaxi();
        double maxy = myg.getmaxi();
        double minx = mxg.getmini();
        double miny = myg.getmini();

        // representation graphique du calcul :
        DomaineDouble dr = new DomaineDouble(minx, miny, maxx, maxy);
        DomaineInt de = new DomaineInt(0, 0, 450, 450);
        CanvasGraphe cg = new CanvasGraphe(dr, de, xgraph, ygraph);
```

```
        FenetreGraphe x = new FenetreGraphe(de, cg, fedo.libelle());
        x.show();
    }
}
```

Exemple d'utilisation sur un mouvement circulaire

Nous testons les classes précédentes en effectuant une résolution numérique du système différentiel :

$$\begin{cases} \frac{dx}{dt} = -\sin(t) \\ \frac{dy}{dt} = \cos t \\ x(0) = 1; \quad y(0) = 0 \end{cases}$$

Nous traçons un portrait de phase, c'est à dire la trajectoire des points de coordonnées $(x(t), y(t))$ en faisant varier t. la solution théorique correspond au cercle trigonométrique.

```
class SEDOCirc extends SysEDO {

    SEDOCirc(int pdim, double pt0, double[] py0, double ppas, int pnbpas,
         int pixGraph, int piyGraph) {
        super(pdim, pt0, py0, ppas, pnbpas, pixGraph, piyGraph);
    }

    public double[] calcul(double t, double[] y) {
        double[] result = new double[2];
        result[0] = -Math.sin(t);
        result[1] = Math.cos(t);
        return result;
    }
    public String libelle() { return "Mouvement circulaire";}
}

class PrgCirc{
    public static void main(String args[]) {
        int dim=2;
        double t0=0;
        double[] y0={1,0};
        double pas=0.5;
        int nbpas=15;
        int ixGraph=0, iyGraph=1;
```

```
        SEDOCirc sf =
            new SEDOCirc(dim, t0, y0, pas, nbpas, ixGraph, iyGraph);
        IterSysEDO methode = new IterSysEuler(sf, pas, y0, t0);
        TraitSEDO work = new TraitSEDO(sf, methode);
    }
}
```

L'avant dernière ligne du programme contient le nom du processus effective de résolution utilisée. Il suffit simplement de modifier cette ligne pour utiliser un autre processus. C'est ce qui est fait dans la suite.

On utilise donc d'abord la méthode d'Euler qui donne de mauvais résultats car elle amplifie à chaque itération sa déviance par rapport au cercle-unité solution du problème.

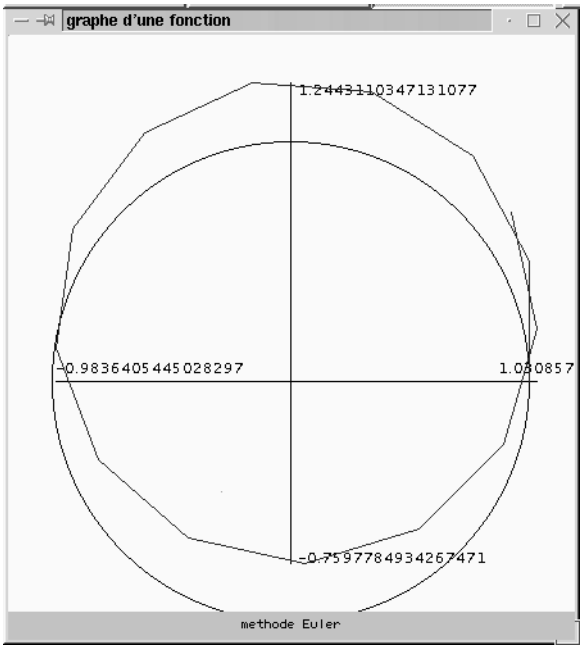

FIG. 8.2: Mouvement circulaire décrit par une méthode d'Euler

La trajectoire suivante est obtenue en utilisant la méthode de Runge-Kutta d'ordre 2. On voit que la solution numérique approchée est meilleure que pour la méthode d'Euler.

Finalement, nous utilisons la méthode de Runge-Kutta d'ordre 4 qui, à chaque itération, va fournir une solution située sur la trajectoire exacte du système (ici, le

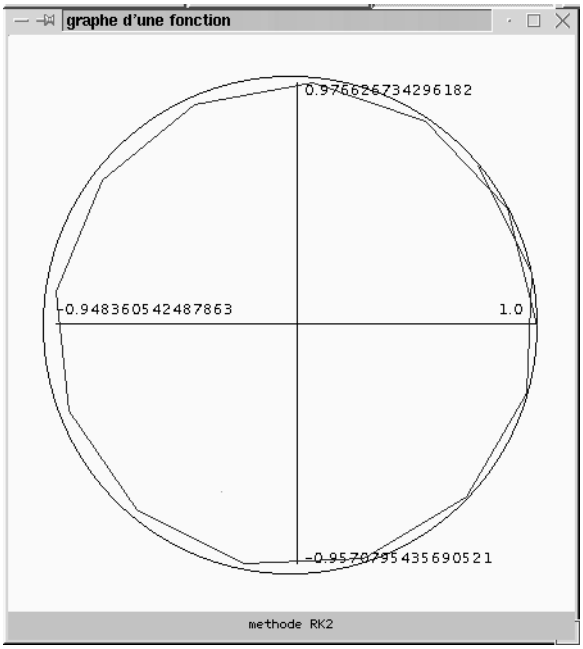

FIG. 8.3: Mouvement circulaire décrit par une méthode de Runge-Kutta d'ordre 2

cercle unité).

Exemple d'utilisation sur les équations de Lorentz

Nous appliquons maintenant notre programme pour résoudre les équations de Lorentz :

$$\begin{cases} \frac{dx}{dt} = a(y-x) \\ \frac{dy}{dt} = cx - xz - y \\ \frac{dz}{dt} = xy - bz \\ x(0) = 1; \quad y(0) = 1; \quad z(0) = 1 \end{cases}$$

a, b et c sont des paramètres qui seront égaux, ici, respectivement à 10, $8/3$ et 28.

```
class SEDOLorentz extends SysEDO {
    double a=10, b=8/3, c=28;

    SEDOLorentz(int pdim, double pt0, double[] py0, double ppas, int pnbpas
         int pixGraph, int piyGraph) {
        super(pdim, pt0, py0, ppas, pnbpas, pixGraph, piyGraph);
    }
```

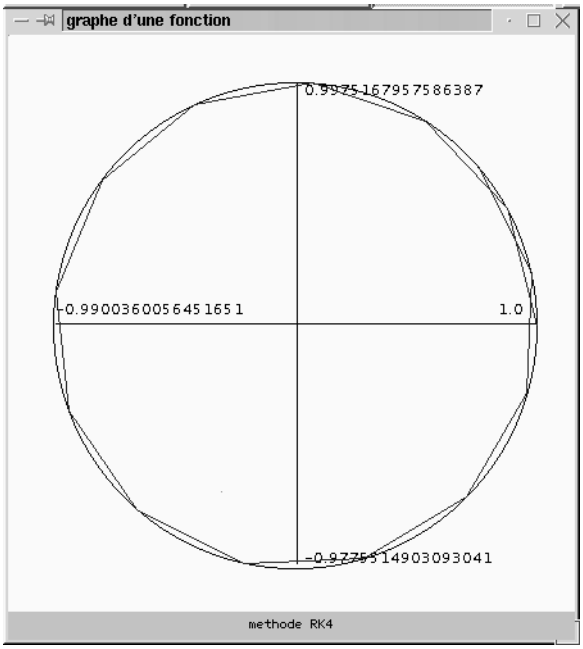

FIG. 8.4: Mouvement circulaire décrit par une méthode de Runge-Kutta d'ordre 4

```
    public double[] calcul(double t, double[] y) {
        double[] result = new double[3];
        result[0] = a*(y[1]-y[0]);
        result[1] = c*y[0]-y[0]*y[2]-y[1];
        result[2] = y[0]*y[1] -b*y[2];
        return result;
    }
    public String libelle() { return "Equations de Lorentz";}
}

class PrgLorentz{
    public static void main(String args[]) {
        int dim=3;
        double t0=0;
        double[] y0={1,1,1};
        double pas=0.01;
        int nbpas=4000;
        int ixGraph=0, iyGraph=2;
        SEDOLorentz sf =
            new SEDOLorentz(dim, t0, y0, pas, nbpas, ixGraph, iyGraph);
```

```
        IterSysEDO methode = new IterSysRK4(sf, pas, y0, t0);
        TraitSEDO work = new TraitSEDO(sf, methode);
    }
}
```

Nous montrons dans la figure 8.5 la solution trouvée en traçant $z(t)$ en fonction de $x(t)$.

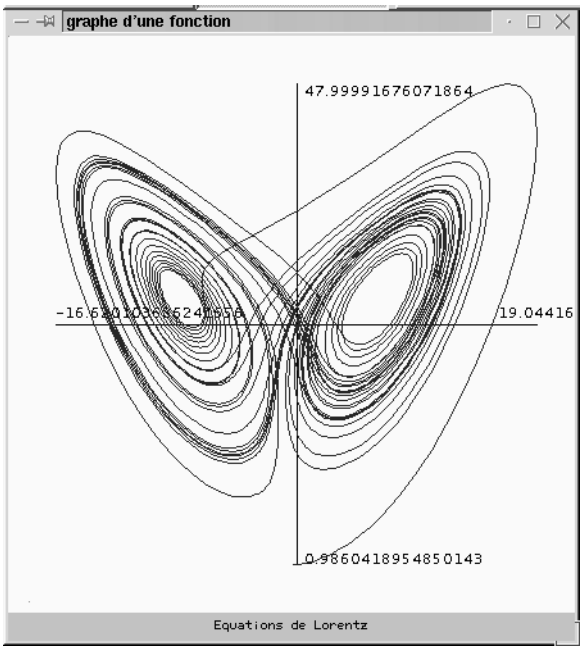

FIG. 8.5: Résolution numérique des équations de Lorentz - Portrait de phase en (x,z)