J. E. ROMBALDI

Algorithmique numérique

Avec Open Ada sur P. C.

TABLE DES MATIERES

Chapitre 3 — Analyse numérique linéaire

Chapitre 5 — Approximation et interpolation

Chapitre 7 — Résolution numérique des équations différentielles

Chapitre 8 — Méthode des différences finies

Avant-propos

Le but de cet ouvrage est de décrire, sans référence à un quelconque langage de programmation, des méthodes classiques d'analyse numérique, et de mettre à la disposition du programmeur scientifique (mathématicien ou utilisateur des mathématiques) une bibliothèque mathématique écrite en Ada et prête à l'emploi sur compatible PC.

Cet ouvrage pourra être très utile aux auditeurs du cours « programmation scientifique » du cycle B du Cnam ainsi qu'à tout étudiant en licence, maîtrise d'analyse numérique ou école d'ingénieurs. De manière plus générale il intéressera tout utilisateur de l'outil mathématique sur ordinateur.

L'informatique et les mathématiques sont ici des outils. C'est un ouvrage de *programmation mathématique* dans la lignée des « Numerical Recipes ».

Cet ouvrage n'a pas la prétention d'être un ouvrage d'informatique. Son but est de montrer que le langage Ada peut être très efficace pour résoudre des problèmes numériques (créneau non encore exploité). J'espère inciter le programmeur scientifique en Pascal ou autre à évoluer vers Ada. Il est vrai que la lecture sera plus aisée pour le lecteur connaissant Pascal (c'est théoriquement le cas pour tout étudiant en école d'ingénieur).

Dans le premier chapitre on fait un rapide tour d'horizon du langage Ada en mettant l'accent sur les grandes qualités de ce langage (essentiellement l'apprentissage d'une « bonne hygiène de programmation »). Le lecteur devra consulter la bibliographie pour plus de détails. Ce chapitre se termine par une brève description du compilateur Open Ada sur PC, distribué en France par Cerus Informatique.

Dans le deuxième chapitre on décrit une série de paquetages de base permettant de manipuler des fonctions d'une ou plusieurs variables réelles et d'exploiter les spécificités du PC pour travailler en mode graphique. Cette bibliothèque est inspirée du logiciel Modulog utilisé par les élèves de classes préparatoires. Elle est adaptée à l'utilisation du compilateur Open Ada sur PC. En général, les ouvrages sur le langage Ada ne se préoccupent pas d'une implémentation particulière d'un compilateur et pourtant il faudra bien travailler sur un type particulier de matériel.

A ma connaissance une telle bibliothèque livrée avec les sources Ada n'est pas disponible dans le domaine public et pourra être très utile au programmeur scientifique.

Dans l'ouvrage, seules les spécifications sont présentées. C'est tout ce que l'utilisateur à besoin de connaître. Le corps de ces paquetages est disponible sur la disquette livrée avec l'ouvrage.

Dans les chapitres 3 à 8 on décrit des méthodes numériques de base.

Pour chaque chapitre j'ai adopté la même philosophie de travail :

— Définir le problème à résoudre.
— Analyser le problème en donnant des résultats théoriques sur l'existence et l'unicité de solutions.
— Décrire les méthodes numériques classiques d'intérêt pédagogique et présenter des méthodes améliorées.

Par exemple, pour le calcul des intégrales on a les méthodes classiques des trapèzes et de Simpson puis celle plus performante de Romberg.

Ces méthodes sont décrites, du point de vue mathématique de façon rigoureuse, mais je n'ai pas jugé utile de réécrire certaines démonstrations que l'on peut trouver dans la littérature. J'utilise des références du type : voir Stoer et Burlisch p. 234.

— Ces méthodes étant destinées à être programmées, l'analyse précédente se termine par une programmation structurée en Français (en *italique* dans le texte) sans référence à un quelconque langage de programmation.

L'intérêt de cette façon de programmer est de ne pas restreindre le public aux seuls programmeurs Ada.

— Une série d'exercice est proposée pour chaque chapitre.
— Enfin le chapitre se termine par la programmation Ada correspondante sous forme de paquetages et d'exemples d'utilisation (en « `courier` » dans le texte). Seules les spécifications des paquetages et des programmes d'exemples figurent sur le manuscrit, l'intégralité se trouvant sur la disquette.

On donne également des exemples d'applications de ces méthodes à des problèmes issus de la mécanique des fluides. Ces exemples sont dus à Guy Aubry, professeur de mécanique des fluides à l'Ensam.

Les sujets traités sont :

Chapitre 3 — Résolution de systèmes d'équations linéaires et recherche de valeurs propres.

Chapitre 4 — Résolution de systèmes d'équations non linéaires.

Chapitre 5 — Mathématiques pour la CAO (Conception Assistée par Ordinateur) : approximations par des courbes de régression, de Bézier et B-Splines ; interpolations de Lagrange et Splines cubiques.

Chapitre 6 — Calculs d'intégrales et transformée de Fourier rapide (outils de la théorie du signal).

Chapitre 7 — Résolution d'équations et de systèmes différentiels.

Chapitre 8 — Résolution d'équations aux dérivées partielles par la méthode des différences finies.

Avec cet ouvrage je pense convaincre le lecteur que le langage Ada est vraiment bien adapté à la programmation mathématique (sécurité nettement supérieure à ce qu'on peut espérer avec les autres langages de programmation, réutilisabilité des composants logiciels, gestion des tableaux très efficace sans avoir à utiliser de pointeurs, ...).

Remerciements — Je tiens à remercier le Professeur André Warusfel qui a bien voulu étudier mon manuscrit. C'est pour moi un grand honneur de le publier dans la collection qu'il coordonne : « Logique Mathématiques et Informatique ».

Je remercie également Marcel Nicolas, Directeur de l'ENSAM de Châlons sur Marne qui a mis à ma disposition le matériel nécessaire à la réalisation de ce livre.

Enfin un grand merci à mon ami et collègue Jean Luc Bauchat pour avoir lu et critiqué une première version.

Contenu de la disquette fournie avec le livre — Cette disquette est découpée en répertoires comme décrit ci-dessous.

A:\DIVERS — Chapitre 2

PREMIERS.ADA	LIST.ADA	POLYGO.ADA	DEM_CART.ADA
DEM_NIV.ADA	DEM_CURV.ADA		

A:\BIBLIO — Chapitres 2 à 8

LIO.ADS	CO_CRT.ADS	CRT.ADS	CRT.ADB
CO_MATH0.ADS	MATH0.ADS	MATH0.ADB	CO_MATH1.ADS
MATH1.ADS	MATH1.ADB	LIRE_FCT.ADA	MATH2.ADS
MATH2.ADB	CO_MATH3.ADS	MATH3.ADS	MATH3.ADB
CO_GRAPH.ADS	GRAPH.ADS	GRAPH.ADB	CURVE.ADS
CURVE.ADB	CARTESIE.ADS	CARTESIE.ADB	CO_MATRI.ADS
MATRIX.ADS	MATRIX.ADB	NIVEAUX.ADS	NIVEAUX.ADB
ALGEBLIN.ADS	ALGEBLIN.ADB	SPECTRE.ADS	SPECTRE.ADB
DON_ALGL.ADS	DON_ALGL.ADB	CO_POLY.ADS	POLY.ADS
POLY.ADB	COMPLEXE.ADS	COMPLEXE.ADB	EQUATION.ADS
EQUATION.ADB	SYST_EQU.ADS	SYST_EQU.ADB	REGRESS.ADS
REGRESS.ADB	BEZIER.ADS	BEZIER.ADB	B_SPLINE.ADS
B_SPLINE.ADB	LAGRANGE.ADS	LAGRANGE.ADB	SPLINE.ADS
SPLINE.ADB	DON_PTS.ADS	DON_PTS.ADB	INTEGRAT.ADS
INTEGRAT.ADB	CO_FOURI.ADS	FOURIER.ADS	FOURIER.ADB
CO_QDIF.ADS	QDIF_11.ADS	QDIF_11.ADB	QDIF_P1.ADS
QDIF_P1.ADB	QDIF_1Q.ADS	QDIF_1Q.ADB	QDIF_PQ.ADS
QDIF_PQ.ADB	DIRICHLE.ADS	DIRICHLE.ADB	ELLIPTIC.ADS
ELLIPTIC.ADB	PARABOLI.ADS	PARABOLI.ADB	

A:\DONNEES — Données pour les programmes

REGR_AFF.DAT : Nuage de points pour la régression affine.

SPL_CART.DAT SPL_PARA.DAT : Nuages de points pour l'interpolation spline.

BILLE.DAT : Données pour BILLE1.ADA et BILLE.ADA.

B_SPLINE.DAT : Nuage de points pour l'interpolation B_Spline.

ALGEBLIN — Chapitre 3

HILBERT.ADA	DEM_PIV.ADA	DEM_LR.ADA	DEM_GJ.ADA
DEM_TRID.ADA	DEM_CHOL.ADA	DEM_JAC.ADA	DEM_REL.ADA
DEM_DEFL.ADA	DEM_VPJA.ADA	DEM_RUTI.ADA	DEM_SOU.ADA
DEM_VAND.ADA	DEM_QR.ADA	DEM_ALGL.ADA	

EQUATION — Chapitre 4

DEM_POL.ADA	DEM_EQU.ADA	DEM_SYST.ADA

INTERPOL — Chapitre 5

DEM_AFF.ADA DEM_POL.ADA DEM_CIRC.ADA DEM_APPR.ADA
BILLE1.ADA DEM_BEZI.ADA DEM_BSPL.ADA DEM_LAGR.ADA
DEM_SPL.ADA DEM_APP1.ADA DEM_APP2.ADA

INTEGRAT — Chapitre 6

DEM_INT.ADA DEM_FFT.ADA

EQUADIFF — Chapitre 7

BILLE.ADA DEM_RK11.ADA DEM_RKP1.ADA DEM_RK1Q.ADA
DEM_RKPQ.ADA DEM_QDIF.ADA

EDP — Chapitre 8

DEM_DIRI.ADA DEM_ELLI.ADA DEM_CHAL.ADA DEM_EDP.ADA

Notes — Les programmes proposés dans ce livre le sont à titre pédagogique. Je ne garantis en aucune manière qu'ils soient exempts d'erreurs. Le lecteur peut les utiliser à ses propres risques et périls. Ni l'auteur ni les éditions Masson ne peuvent être tenus pour responsables des dommages que l'utilisation de ces programmes pourrait occasionner.

En aucun cas ces programmes ne peuvent être utilisés à des fins commerciales ou industrielles.

Open Ada est une marque déposée de Meridian Software.

First Ada est une marque déposée de Alsys.

CHAPITRE 1

Le langage Ada avec OpenAda

1. Petit historique de Ada

C'est dans les années 1970 que l'on se rend compte des problèmes posés par le trop grand nombre de langages informatiques existants. Cette diversité de langages entraîne une maintenance de plus en plus difficile. En passant d'une machine à l'autre on ne dispose pas nécessairement des mêmes langages et il est parfois nécessaire de tout reprendre à zéro, la fiabilité n'étant pas très bien assurée.

Le langage Ada est donc né du besoin de définir un standard en matière de langage de programmation.

En 1975, le département de la défense Américain (le *« DOD » = Department Of Defense*), qui est considéré comme le plus gros consommateur d'informatique au monde, a lancé un concours international en vue de définir un nouveau langage de programmation prenant en compte toutes les améliorations actuelles. Ce langage doit permettre, entre autres, de traiter des problèmes numériques, de gestion, de calculs en temps réel, ainsi que la gestion d'activités parallèles et la programmation de systèmes embarqués (calculateurs sans systèmes d'exploitation, tel un missile guidé). Il doit également permettre la programmation modulaire et la définition de nouveaux types de variables, tout en étant lisible, facile à maintenir et assurer la portabilité maximale.

Les programmes étant de plus en plus ambitieux et nécessitant beaucoup de participants, le travail d'un membre de l'équipe doit pouvoir être lu sans documentation.

En 1979, c'est le projet du Français *Jean Ichbiach*, de chez *CII Honeywell-Bull*, qui a été retenu. Le langage obtenu a reçu le nom de « *ADA* » en hommage à la comtesse de Lovelace, *Adélaide Augusta Byron* (1815-1851), fille du poète Anglais Lord Byron, qui était l'assistante de *Babbage* et qui est considérée comme le premier programmeur.

Le standard définitif du langage est apparu en 1983 avec la norme *ANSI MIL-STD-1815A* et les premiers compilateurs sont disponibles à partir de 1985 sur différents types de matériels.

C'est la première fois qu'une norme est définie avant l'existence de tout compilateur. Pour les autres langages c'est l'inverse qui s'est produit d'où la diversité des compilateurs C, Fortran, Pascal ...

La validation des compilateurs est très sévère et elle doit être renouvelée tous les 18 mois. Ces restrictions donnent des compilateurs de très bonne qualité.

La norme est révisée tous les dix ans. Le projet actuel de révision est le projet *ADA 9X*. Avec cette nouvelle norme Ada sera véritablement orienté objet. Si Ada est à l'origine de la conception orientée objet, avec le livre de *Grady Booch :* Ingénierie du logiciel avec Ada, ce n'est pas à l'heure actuel un langage orienté objet au même titre que C++, Objective C ou Eiffel.

Actuellement, le département de la défense Américain utilise ce langage de façon quasi exclusive pour tout traitement informatique.

Le langage Ada est fortement inspiré du Pascal et de l'Algol60, mais on y retrouve certains des points positifs d'autres langages tels que Fortran, C, Cobol, Pl1 ou Algol. Il est considéré comme un langage d'usage général.

On ne peut pas vraiment considérer Ada comme un nouveau langage, mais plutôt comme la synthèse des connaissances actuelles sur les langages de programmation. Néanmoins, avec Ada, une nouvelle philosophie de programmation s'est développée .

Si l'apprentissage à un premier niveau est relativement aisé, spécialement pour le programmeur en Pascal, il est beaucoup plus difficile d'en exploiter toutes les subtilités.

Pour la lecture de cet ouvrage une connaissance du langage à un premier niveau est suffisante. Le livre de B. Leguy indiqué en référence est un très bon ouvrage d'initiation au langage Ada. On pourra également consulter le livre de Barnes qui est considéré comme un manuel de base. Dans la bibliographie on trouvera une liste d'ouvrages permettant une étude plus approfondie.

Comme Pascal, Ada est un langage très « typé », c'est-à-dire que toutes les variables utilisées doivent être déclarées au préalable et le programmeur a la possibilité de définir de nouveaux types de variables.

On dispose en Ada d'un grand éventail de types de variables dont les tableaux, les enregistrements et les pointeurs. Mais Ada est encore plus rigoureux que Pascal dans la manipulation des variables. Si, par exemple, en Pascal on peut combiner dans une expression travaillant sur des réels une variable de type INTEGER et une de type REAL, une telle opération n'est pas possible en Ada sans une conversion préalable de la variable de type INTEGER en type FLOAT (c'est le type réel de Ada).

Comme Pascal et C, Ada permet la récursivité.

Le langage Ada assure une meilleure sécurité que le Pascal tout en étant aussi puissant que le C (on a la possibilité de travailler au niveau de la machine).

La gestion des erreurs, grâce à la notion *d'exception*, est beaucoup plus efficace qu'en Pascal ou C.

L'écriture de gros programmes découpés en unités compilées séparément se fait de manière beaucoup plus sûre.

Toutes ces propriétés facilitent de beaucoup la maintenance des programmes qui constituent la partie « chère » des logiciels.

Parmi les nouveautés, on dispose de la notion de *« tâche »*. Un programme Ada peut consister en plusieurs tâches s'effectuant de manière asynchrone et en parallèle. La synchronisation des tâches est possible grâce à la notion de *« rendez-vous »*. Ce qui permet la programmation en *« temps réel »*.

La portabilité du langage, c'est-à-dire la qualité d'un logiciel à transférer son savoir faire sur un autre type de matériel à un coût très inférieur à sa réécriture, est également un des aspects intéressants du langage. Cette portabilité est assurée par les normes ANSI (en 1983) et ISO (en 1987).

2. Une présentation rapide du langage Ada

2.1 Le manuel de référence

Le langage est défini par son manuel de référence qui est contrôlé par une commission de validation. Tout compilateur doit être en parfait accord avec ce manuel. Ce qui dépend de l'implémentation de Ada sur un type de machine doit être décrit dans l'annexe F de la documentation fournie avec le compilateur.

Le manuel de référence étant une norme, il est identique pour tout type de compilateur. Les remarques particulières doivent être repérables.

Une mention à ce manuel se fera sous la forme [LRM 4.3.2 (11)], pour chapitre 4, section 3, paragraphe 2 et alinéa 11.

Souvent les messages d'erreur d'un compilateur feront une mention de ce type là.

Il existe une version Française de ce manuel, c'est : Manuel de Référence du langage de programmation Ada ALSYS : 29, Av. Lucien René Duschesne, 78170 La Celle Saint-Cloud.

2.2 Un survol du langage

Dans le survol qui suit, nous présentons les éléments du langage Ada qui peuvent être très facilement assimilés par le programmeur en Pascal ou C.

Pour une étude plus détaillée on pourra consulter les ouvrages de la bibliographie.

Un des points forts du langage est le *typage des données* qui assure une fiabilité optimale. Selon le cabinet américain de « consulting » Reifer, portant sur 41 projets et 15 millions de lignes de codes, on détecte en moyenne 4 à 13 erreurs pour 1000 lignes avec des langages différents de Ada. Ce taux est ramené à 3 à 5 erreurs pour Ada (01 Informatique, 10/01/92).

Le langage reprend le typage de Pascal, mais avec beaucoup plus de rigueur.

On dispose des *types énumératifs* identiques à ceux de Pascal, par exemple, on peut écrire :

```
type WEEK_END is (SAMEDI, DIMANCHE) ;
type COULEUR is (ORANGE, VERT, ROUGE) ;
type FRUIT is (ORANGE, POMME) ;
```

et il n'y aura pas de confusion entre une variable de type FRUIT prenant la valeur ORANGE et une variable de type COULEUR prenant la valeur ORANGE. Le compilateur vérifie avec beaucoup de soins la cohérence des affectations.

On dispose également de la possibilité de distinguer de façon logique des données ayant la même représentation. Ainsi, en écrivant :

```
type DOLLAR is range 0..100_000 ;
type FRANC is range 0..100_000 ;
```

on ne risque pas d'ajouter des dollars et des francs, une telle opération sera rejetée à la compilation.

On dispose également de la possibilité de préciser de façon très stricte le domaine de validité de certaines variables réelles, on peut par exemple écrire :

```
type DISTANCE is digits 3 range 0.0..3245.0 ;
type VOLT is delta 0.01 range 0.0..220.0 ;
```

pour des distances calculées avec trois décimales et des volts, avec un pas de 0.01. Tout dépassement des valeurs permises déclenchera une erreur.

On a également la possibilité de définir des tableaux homogènes ou hétérogènes, avec une dimension fixée, quelconque ou précisée à l'exécution.

Par exemple, on peut écrire :

```
type VECTEUR is array(POSITIVE range 1..20) of FLOAT ;
```

pour des vecteurs à 20 composantes, ou

```
type VECTEUR is array(POSITIVE range <>) of FLOAT ;
```

pour des tableaux dont la dimension sera précisée dans le programme appelant avec une déclaration du type :

```
v : VECTEUR(2..2) ;
```

Le type *access* permet de déclarer des variables de type *pointeur*. L'adresse d'un objet est contenue dans une variable et cette adresse mènera forcément vers un objet du bon type. On ne peut pas pointer sur n'importe quoi comme en C.

L'utilisation des types permet de masquer les détails d'implémentation, par exemple le concept abstrait de fichier est suffisant. C'est la notion de *types privés* qui permet de gérer cela au mieux. Cette manipulation abstraite des objets permet d'avoir une structure logique très stricte ce qui facilite le côté maintenance.

Un autre exemple significatif est celui des piles. On peut définir un type privé comme suit :

```
type PILE is private ;
```

et l'utilisateur n'a pas besoin de savoir comment est défini ce genre d'objet, s'il sait que c'est défini à l'aide de tableaux on a le risque de le voir trafiquer l'élément numéro 56, alors qu'on ne manipule que le sommet de la pile. Ce genre de comportement est catastrophique à la maintenance. Sachant qu'il dispose d'un type pile, l'utilisateur a seulement besoin de connaître les opérations disponibles sur ces objets, par exemple il peut avoir à sa disposition les procédures suivantes :

```
procedure EMPILE(TOTAL : in FLOAT ; P : in out PILE) ;
procedure DEPILE(TOTAL : out FLOAT ; P : in out PILE) ;
```

le changement d'implémentation de la pile et des procédures associées ne doit pas changer une ligne du programme de l'utilisateur.

Parmi les choses nouvelles, on dispose de la notion de *tâche* pour les applications en temps réel et les activités parallèles. Il n'est pas nécessaire de connaître cette notion pour lire cet ouvrage.

On dispose aussi de la notion de *type dérivé* qui permet de définir des sous-types d'un type donné. Les type « fils » héritant des propriétés des types « parents ».

Le langage est évidemment structuré. Les instructions structurées ayant une indication de fin plus pratique que les BEGIN...END ; de Pascal. Par exemple, on a :

```
loop            if...      case...
  ...              ...        ...
end loop;       end if;    end case;
```

Les sous-programmes peuvent être traités séparément ou regroupés dans des *paquetages*. Ils doivent être écrits de façon à être facilement réutilisables.

Pour ce qui est des paramètres formels des procédures, on dispose des modes *in*, *out* et *in out*. En mode *in* la variable doit avoir été initialisée et ne peut être modifiée dans la procédure. En mode *out* la variable n'est pas nécessairement initialisée et aura obtenu une valeur en fin de sous-programme et le mode *in out* permet de modifier une variable initialisée.

Pour les fonctions, seul le mode *in* est autorisé.

On dispose de la notion *d'agrégat* qui permet de placer les paramètres effectifs d'une procédure dans n'importe quel ordre sans risque d'erreur. Par exemple si on a la déclaration :

```
procedure P1(n in : POSITIVE ; x in out FLOAT ; y out FLOAT) ;
```

on pourra l'appeler de la façon suivante :

```
P1(y => 6.8, n => 2, x => 8.9) ;
```

cette façon de procéder présente l'avantage de ne pas être obligé de retourner à la documentation pour connaître l'ordre logique des paramètres.

On a également la possibilité de *surcharger les opérateurs existants*. On peut par exemple définir une opération d'addition notée + qui agit sur des objets de type matrice. On écrira :

```
function "+" (A, B : MATRICE) return MATRICE is
```

le type MATRICE ayant été préalablement défini comme un tableau à deux indices.

Quand on définit un paquetage, c'est-à-dire une bibliothèque de sous-programmes, on peut séparer la *spécification*, qui est la section de déclaration, de son corps principal où les procédures et fonctions sont effectivement traitées. Pour l'utilisateur l'interface seule compte, les détails d'implémentation peuvent ne pas l'intéresser. Cela présente aussi l'avantage de pouvoir travailler en équipe pour de gros projets. Si deux équipes différentes ont donné le même nom à des procédures différentes, on peut les utiliser sans accéder au code source, en fonction de la nature des paramètres le compilateur se débrouille pour savoir de quoi il s'agit.

On dispose aussi de la notion de sous-unité qui permet le développement hiérarchique de gros programmes en reportant à plus tard l'écriture de certaines procédures. Pour ce faire on utilise la clause *separate*. La sous-unité sera compilée à part sans avoir à recompiler l'unité mère.

La notion *d'unité générique* permet de définir des procédures agissant sur des objets dont on ne connaît pas la nature. Par exemple il est inutile d'écrire une procédure d'échange de deux variables pour chaque type de variable, il suffit d'écrire, dans la partie spécification de l'unité :

```
generic
     type VARIABLE is private ;
     procedure ECHANGER(x, y : in out VARIABLE) ;
```

puis dans le corps du paquetage :

```
     procedure ECHANGER(x, y : in out VARIABLE) is
     AUX : VARIABLE := x ;
     begin
       x := y ;
       y := AUX ;
     end ;
```

On précise alors sur quel type d'objet va agir ECHANGER en l'instanciant de la manière suivante :

```
procedure ECHANGER_REELS is new ECHANGER(FLOAT) ;
procedure ECHANGER_ENTIERS is new ECHANGER(INTEGER) ;
```

On peut contrôler les erreurs d'exécution avec la notion d'exception. Ce contrôle d'erreur augmente la taille du code exécutable de 10 à 20%. Pour des applications critiques du type embarqué où on compte le moindre octet on peut demander au compilateur de ne plus générer le code relatif aux vérifications. On procède de la sorte pour le produit final.

On a aussi la possibilité de programmer à un bas niveau en gérant les interruptions.

Enfin on dispose, grâce au *« pragma interface »* de la possibilité d'utiliser des sous-programmes écrit dans un autre langage (Assembleur, C, Fortran, Pascal, ...).

2.3 Exemples d'utilisations de Ada

Le langage Ada est utilisé de façon systématique par le département de la défense Américain et diverses armées dans le monde.

On trouve des utilisations dans les domaines cités ci dessous :

- Avionique. Avec les réalisations suivantes :

Aux Etats Unis :

Contrôle du trafic aérien, pour le Federal Aviation Authority ;

Contrôle des cockpits des 767 et 777 pour Boeing ;

Simulateur de vol, pour Ford Aerospace.

Au Canada :

Contrôle du trafic aérien, pour CSC/IBM.

En Europe :

Contrôle du trafic aérien en Europe, pour Eurocontrol ;

Contrôle du trafic aérien, pour Thompson SDC ;

Contrôle d'équipements au sol, pour Plessey Avionics ;

.....

- L'espace, avec les applications suivantes :

Logiciel d'environnement et de gestion des données pour la station spatiale Freedom ;

Contrôle de robots pour la NASA ;

Station de transmission pour les communications spatiales pour Jet Propulsion Lab ;

Logiciels pour Ariane V ;
...

- L'industrie, avec les applications suivantes :
  Processeurs électroniques pour Lockheed ;
  Logiciel d'aide à l'extraction de pétrole pour BP Oil, Total et Schlumberger Oil ;
  Exploration géophysique du pétrole pour Shell Oil ;
  Système de transmission dans le domaine des chemins de fer pour la CSEE ;
  Systèmes d'instrumentation, pour Schlumberger ;
  Automate de simulation pour 3IP/Citroën ;
  ....

On a également des applications dans les domaines de la banque et des assurances, dans le domaine commercial, médical, des télécommunications, de l'électronique, de l'intelligence artificielle et du nucléaire.

3. Utilisation du compilateur OpenAda sur micro-ordinateur

Ce compilateur, qui est validé, permet de travailler sur tout micro-ordinateur du type IBM-PC (de préférence un 386 ou 486) sous environnement MS-DOS. De plus, il est livré avec un environnement de programmation à base de menus déroulants, une bibliothèque mathématique et une bibliothèque graphique.

La dernière version permet de travailler en mode réel, 16 bits ou 32 bits. De plus une version sous Windows est disponible. Ces versions sont proposées avec un « debugger ». Dans ce qui suit on s'intéresse seulement à la version DOS.

Il est distribué par Meridian Software, cette société étant représentée en France par Cerus-Informatique.

On peut l'utiliser directement en ligne de commande, sous DOS, ou depuis l'environnement de programmation ACE (Ada Compiler Environnement). Nous décrivons tout d'abord son utilisation en ligne de commande, pour bien comprendre les étapes de création d'une bibliothèque, de compilation et d'édition de liens.

Une fois l'installation réalisée sur le disque C (ou D, ...), on dispose du répertoire nommé Ada (ou autre), contenant les sous-répertoires *ACE*, *BIN*, *PACLIB*, *INASM*, *MSC*, *ADAUTIL*, *DOSENV*, *MATH*, *AGUL*, *SBOOCH* et *TEST*. Pour la version 386 on a de plus les répertoires *ADAUTIL.32*, *DOSENV.32* et *PACLIB.32*. Ils correspondent à la version 32 bits.

Le répertoire ACE contient les fichiers nécessaires à l'environnement de programmation. BIN contient les programmes exécutables indispensables au compilateur. PACLIB contient les bibliothèques et fichiers de configuration. INASM contient des exemples de programmes interfacés avec le langage assembleur ainsi qu'une documentation sur le sujet. MSC contient des exemples de programmes interfacés avec le langage C de Microsoft. Math contient une

bibliothèque mathématique avec un exemple d'utilisation. Agul (Ada Graphic Utility Library) contient une bibliothèque graphique avec un exemple d'utilisation. Et Test est un répertoire contenant des programmes de démonstration.

Avant d'utiliser un compilateur Ada on doit créer une *bibliothèque* (en Anglais *library*). Avec OpenAda, on utilise le programme *NEWLIB.BAT* qui crée dans le répertoire de travail le fichier *ADA.LIB* et le répertoire *ADA.AUX*. C'est dans ADA.AUX que seront stockés les codes objets correspondants aux sous-programmes compilés. Cette création se fait une seule fois dans un répertoire donné.

Pour plus de détails sur la gestion d'une telle bibliothèque, on se reportera au chapitre 8 du « Compiler User's Guide » fourni avec la documentation.

Cette bibliothèque est nécessaire pour produire le code objet correspondant à un code source donné. Le processus de compilation est schématisé ci-dessous :

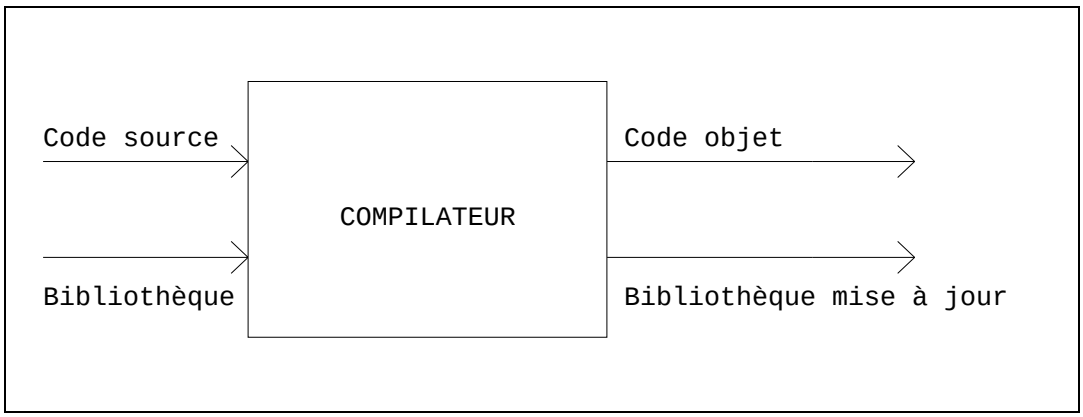


Figure 1.1

Cela permet une compilation séparée et dépendante, c'est-à-dire qu'on garde une trace de toutes les compilations et des liens avec les autres bibliothèques.

Si on compare avec ce qui se passe en Fortran où on a une compilation séparée, mais indépendante, on a une meilleure sécurité. En Fortran on peut écrire dans un programme CALL Sub(i,j,k), alors que la subroutine Sub, se trouvant dans un autre fichier, a été déclarée comme agissant seulement sur deux paramètres. Une telle erreur ne sera pas décelée à la compilation. Alors qu'en Ada, si on a compilé la procédure Sub agissant sur deux paramètres entiers i et j, l'appel Sub(i,j,k) dans n'importe quelle procédure ou fonction provoquera une erreur à la compilation

L'avantage de la compilation séparée est de pouvoir traiter et tester séparément les divers morceaux d'un programme.

Pour compiler le programme Simple.Ada, il suffit d'écrire : *ADA Simple.Ada***.**

Puis l'édition de liens se fait avec : *BAMP Simple*, où BAMP signifie *Build Ada Main Program*. Et si tout se passe normalement, le fichier Simple.Exe a été créé. Bamp doit être suivi du nom du programme principal qui peut être différent de celui du fichier qui le contient.

On dispose avec ADA et BAMP de plusieurs options. On se reportera, pour plus de détails, au fascicule « Compiler User's Guide PC DOS » faisant partie de la documentation.

En fait, il est beaucoup plus agréable d'utiliser l'environnement de programmation ACE, d'utilisation relativement simple.

Pour l'utiliser, il suffit de taper soit ACE, soit ACE suivi du nom du programme à charger.

Cet éditeur peut être configuré à sa convenance. Cette configuration se fait en utilisant le sous-menu *CHANGE BINDINGS* du sous-menu *OPTIONS* du menu principal. On peut y accéder directement avec la combinaison de touches ALT 2.

On a également la possibilité d'utiliser une « souris ». Le bouton droit permet d'accéder au menu principal et ensuite il suffit de savoir lire.

Dans le tableau 1.1 nous décrivons les options par défaut et celles que nous avons choisies.

On peut aussi utiliser le compilateur *FirstAda* de *ALSYS*. Ce compilateur est utilisable sous MS-DOS en mode 32 bits, mais nécessite au minimum un 386 avec 3 Mo. de mémoire vive. Il est livré avec une bibliothèque mathématique mais pas de bibliothèque graphique. Ce compilateur est d'aussi bonne qualité que celui de Meridian Software, mais plus onéreux et de maniement plus complexe.

Les paquetages décrits dans le chapitre suivant et utilisés dans les autres chapitres sont utilisables avec OpenAda. Pour les utiliser avec FirstAda, il faudra en réécrire certaines parties (tout ce qui est relatif au clavier et à l'écran).

ACTION	TOUCHES PAR DEFAUT	TOUCHES CHOISIES
— Menu Principal	ESC	ESC
— Quitter un menu	ESC	ESC
— Charger un fichier	CTRL N	F3
— Sauvegarder un fichier	ALT W et <ENTER>	F2 et <ENTER>
— Le sauver sous un autre nom	ALT W, Nom et <ENTER>	F2, Nom et <ENTER>
— Créer une bibliothèque		ALT F8
— Compiler le fichier édité		F9
— Faire l'édition de liens		ALT F9
— Lancer l'exécution du programme édité		CTRL F9, <ENTER>
— Exécuter un programme.		CTRL F9, Nom
— Aide sur l'éditeur	ALT H	ALT H
— Retour temporaire au DOS	ALT F2	SHIFT F10
— Chercher une chaîne de caractères et la remplacer par une autre	CTRL R	CTRL R
— Effacer une ligne	CTRL D	CTRL Y
— Quitter sans sauvegarder	ALT Z	ALT X
— Quitter en sauvegardant	CTRL Z	CTRL X
— Marquer le début d'un bloc	F1	F1
— Marquer la fin d'un bloc	Flèches de déplacement	Flèches de déplacement
— Copier le bloc marqué dans un « buffer »	F3	CTRL C
— Insérer à la position du curseur un bloc du « buffer »	F7	F7
— Imprimer le bloc marqué		CTRL P
— Supprimer le bloc marqué	F3	ALT D

Tableau 1.1

CHAPITRE 2

Bibliothèque mathématique

1. Introduction

La bibliothèque décrite dans ce chapitre est formée des paquetages CRT, MATH0, MATH1, MATH2, GRAPH et MATH3. A chacun de ces paquetages est associé le paquetage COMMON_NOM_DU_PAQUETAGE où sont déclarés des constantes, types, variables globales et exceptions.

La conception de cette bibliothèque destinée à faciliter l'écriture de programmes mathématiques est inspirée du logiciel Modulog (écrit en Pascal), qui est produit par l'ALE Sup. sous la responsabilité d'Hervé Lehning et Robert Rolland, et du livre de Ducamp et Reverchon : Mathématiques en Turbo-Pascal, Volume 1 : Analyse.

Tout programme utilisant les paquetages CRT, MATH0, et COMMON_MATH1 (par exemple) devra commencer par :

```
with CRT, MATH0, COMMON_MATH1 ;
use CRT, MATH0, COMMON_MATH1 ;
```

L'utilité de chacun de ces paquetages est décrite ci-dessous.

- Dans le paquetage *CRT*, on trouve des procédures et fonctions destinées à faciliter la gestion du clavier et de l'écran. Il est inspiré de l'unité CRT de Turbo-Pascal. Ce paquetage utilise les interruptions DOS.
- Dans le paquetage *MATH0*, on trouve des procédures destinées à faciliter la communication entre un programme et son utilisateur, ainsi que des procédures permettant de gérer les erreurs d'entrées-sorties.
- Dans le paquetage *MATH1*, on définit des fonctions mathématiques qui ne sont pas directement disponibles avec Ada. Ce paquetage utilise le paquetage non générique MATH_LIB fourni avec OpenAda.

On pourrait aussi écrire une version générique qui aurait l'avantage de permettre à l'utilisateur de jouer avec les précisions. Pour la conception d'un tel paquetage on pourra consulter le manuel « Math Library » fourni avec la documentation du compilateur OpenAda. Dans ce manuel on décrit le paquetage

GENERIC_ELEMENTARY_FUNCTIONS et des exemples d'utilisation sont donnés dans le répertoire MATH.

Dans MATH1 on travaille directement avec le type FLOAT par souci de simplicité.

- Avec le paquetage *MATH2*, on définit un nouveau type de variable, le type FONCTION, qui permet d'entrer au clavier des fonctions de une à plusieurs variables (ce nombre est défini en constante) et de les évaluer en fonction des paramètres effectifs donnés.
- Dans le paquetage *GRAPH*, on trouve des procédures permettant d'initialiser un mode graphique, de dessiner des points et des segments en mode graphique et de revenir au mode texte. Il est inspiré de l'unité Graph de Turbo-Pascal et utilise les interruptions DOS.
- Avec le paquetage *MATH3*, on dispose de procédures permettant de travailler en mode graphique dans une fenêtre graphique utilisateur en coordonnées réelles. Elle sera très utile pour le tracé de courbes.

Pour utiliser les entiers du type LONG_INTEGER, on instancie le paquetage INTEGER_IO comme suit :

```
with TEXT_IO ;
package LIO is new TEXT_IO.INTEGER_IO(LONG_INTEGER) ;
```

Dans les paragraphes qui suivent seules les spécifications des paquetages sont présentées ainsi qu'une documentation. Les codes sources des corps correspondants se trouvant sur la disquette fournie avec l'ouvrage.

2. Le paquetage CRT

Ce paquetage permettant la gestion du clavier et de l'écran n'est évidemment pas facilement portable.

La version décrite dans ce paragraphe utilise le paquetage *INTERRUPT* de OpenAda. Ce paquetage permet de manipuler les interruptions BIOS et DOS. L'action de chacune des interruptions utilisée est décrite en commentaire dans le code du corps de CRT.

Pour écrire une version équivalente avec FirstAda, on peut utiliser le paquetage INTERRUPT_MANAGER ou le paquetage DOS. Pour plus de détails, on se reportera à la documentation : Application Developer's Guide and Appendix F.

Pour certaines procédures, on peut se passer des interruptions en utilisant les séquences d'échappement ANSI.

2.1 Spécifications des paquetages COMMON_CRT et CRT

```
package COMMON_CRT is
subtype COLONNE_TEXTE is INTEGER range 0..79 ;
subtype LIGNE_TEXTE is INTEGER range 0..24 ;
end COMMON_CRT ;

with COMMON_CRT ;
use COMMON_CRT ;
package CRT is

procedure CURSOR(B : in BOOLEAN) ;
procedure CLREOL ;
procedure CLRSCR ;
```

```
procedure GOTOXY(C : in COLONNE_TEXTE ; L : in LIGNE_TEXTE) ;
function READKEY return CHARACTER ;
procedure PUT_CHAR(C : in CHARACTER) ;
procedure PUT_STRING(Msg : in STRING) ;
procedure PUT_LINE_STRING(Msg : in STRING) ;
function WHERE_X return COLONNE_TEXTE ;
function WHERE_Y return LIGNE_TEXTE ;
end CRT ;
```

2.2 Documentation du paquetage CRT

2.2.1 La procédure CLREOL

Cette procédure permet d'effacer tous les caractères d'une ligne à partir de la position du curseur.

2.2.2 La procédure CLRSCR

Cette procédure permet d'effacer l'écran.

2.2.3 La procédure GOTOXY

Permet de se positionner en colonne C et ligne L.

2.2.4 La fonction READKEY

Permet de lire un caractère au clavier sans écho à l'écran.

2.2.5 Les fonctions WHERE_X et WHERE_Y

Ces fonctions renvoient, respectivement, le numéro de colonne et de ligne de la position actuelle du curseur.

2.2.6 La procédure CURSOR

Cette procédure permet d'inhiber (B => FALSE) ou non (B => TRUE) le curseur.

2.2.7 La procédure PUT_CHAR

Cette procédure permet un affichage plus rapide et plus pratique d'un caractère, que la procédure PUT de TEXT_IO.

2.2.8 La procédure PUT_STRING

Permet d'afficher une chaîne de caractères en utilisant la procédure précédente et sans passer à la ligne suivante.

2.2.9 La procédure PUT_LINE_STRING

Même effet que la précédente en passant à la ligne suivante après affichage.

2.2.10 Partie initialisation de CRT

```
begin
    CLRSCR ;
    CURSOR(TRUE) ;
end CRT ;
```

3. Le paquetage MATH0

3.1 Spécifications des paquetages COMMON_MATH0 et MATH0

La variable IMP, de type fichier texte (i. e. fichier ASCII) est utilisée dans la procédure MODE_AFFICHAGE.

```
with TEXT_IO ;
package COMMON_MATH0 is
subtype HEURE is INTEGER range 0..23 ;
subtype MINUTE is INTEGER range 0..59 ;
subtype SECONDE is INTEGER range 0..59 ;
subtype CENTIEME is INTEGER range 0..99 ;
type TIME is record
           H : HEURE := 0 ;
           M : MINUTE := 0 ;
           S : SECONDE := 0 ;
           C : CENTIEME := 0 ;
         end record ;
type PERIPHERIQUE_DE_SORTIE is (ECRAN, IMPRIMANTE, FICHIER) ;
CHOIX_AFFICHAGE : PERIPHERIQUE_DE_SORTIE := ECRAN ;
IMP : TEXT_IO.FILE_TYPE ;
end COMMON_MATH0 ;

with TEXT_IO, COMMON_MATH0 ;
use COMMON_MATH0 ;
package Math0 is

procedure RANDOMIZE ;
function RANDOM return FLOAT ;
function RANDOM(n : in INTEGER) return NATURAL ;
function MAJUSCULE(C : in CHARACTER) return CHARACTER ;
function CHAINE_ENTIERE(n : in INTEGER) return STRING ;
function CHAINE_REELLE(x : in FLOAT) return STRING ;
procedure GET_LINE(n : out INTEGER) ;
procedure GET_LINE(x : out FLOAT) ;
procedure GET_LINE(c : out CHARACTER) ;
procedure GET_LINE(Msg : out STRING) ;
procedure PAUSE ;
procedure BIP ;
function LIRE_REPONSE(Msg : in STRING) return BOOLEAN ;
function TRUNC(x : in FLOAT) return INTEGER ;
function FRAC(x : in FLOAT) return FLOAT ;
procedure CHRONOMETRE(Top : in INTEGER) ;
procedure ENTRER_REEL(x : out FLOAT ; Msg : in STRING := " ") ;
procedure ENTRER_ENTIER(n : out INTEGER ; Msg : in STRING := " ") ;
procedure ENTRER_ENTIER_LONG(n : out LONG_INTEGER ;
                             Msg : in STRING := " ") ;
procedure ENTRER_REEL_BORNE(rMin, rMax : in FLOAT ; x : out FLOAT ;
                            Msg : in STRING := " ") ;
procedure ENTRER_ENTIER_BORNE(nMin, nMax : in INTEGER ;
                            n : out INTEGER ; Msg : in STRING := " ") ;
procedure ENTRER_ENTIER_LONG_BORNE(nMin, nMax : in LONG_INTEGER ;
                      n : out LONG_INTEGER ; Msg : in STRING := " ") ;
```

```
procedure ECHANGER_REELS(x, y : in out FLOAT) ;
procedure ECHANGER_ENTIERS(p, q : in out INTEGER) ;
procedure LECTURE_REPERTOIRE(NOM_REPERTOIRE : in STRING) ;
procedure LIRE_FICHIER(FICHIER_DONNEES : in out TEXT_IO.FILE_TYPE ;
                       NOM_DE_FICHIER : in out STRING ;
                       NOM_DE_REPERTOIRE : in STRING := "A:\DONNEES") ;
function FICHIER_EXISTE(NOM_DE_FICHIER : in STRING) return BOOLEAN ;
procedure CREER_FICHIER(FICHIER_DONNEES : in out TEXT_IO.FILE_TYPE ;
                        NOM_DE_FICHIER : in out STRING ;
                        NOM_DE_REPERTOIRE : in STRING := "A:\DONNEES") ;
procedure MODE_AFFICHAGE ;
procedure VAL(Msg : in STRING ; DEBUT, FIN : in NATURAL ;
              i : in out NATURAL ; Err : in out NATURAL) ;
procedure VAL(Msg : in STRING ; DEBUT, FIN : in NATURAL ;
              x : in out FLOAT ; Err : in out NATURAL) ;
function LONGUEUR_CHAINE(Msg : in STRING) return NATURAL ;
end MATH0 ;
```

3.2 Documentation du paquetage MATH0

Le corps de ce paquetage utilise les paquetages INTERRUPT, ERRORS et FILE_IO livrés avec OPEN_ADA.

Dans ERRORS est défini un type énuméré EXTENDED_ERRORS permettant de gérer des erreurs DOS.

Le paquetage FILE_IO permet la gestion de fichiers sous DOS.

3.2.1 La procédure GET_TIME

Cette procédure donne l'heure maintenue par le système d'exploitation.

3.2.2 La procédure RANDOMIZE

Cette procédure permet d'initialiser un générateur de nombres pseudo-aléatoires. Elle doit être appelée avant les fonctions RANDOM. Elle utilise la procédure GET_TIME qui permet de connaître l'heure maintenue par le système d'exploitation.

3.2.3 Les fonctions RANDOM

RANDOM sans argument retourne un réel aléatoire entre 0.0 et 1.0 et RANDOM(n) retourne un entier aléatoire entre 0 et $n - 1$, pour $n \leq 1$ et 0 pour $n < 1$.

Voir le paragraphe (4.1) du chapitre sur le calcul numérique des intégrales pour un algorithme de construction de nombres pseudo-aléatoires.

3.2.4 La fonction MAJUSCULE

Permet de convertir un caractère compris entre 'a' et 'z' en majuscule.

3.2.5 Les procédures GET_LINE

Ces procédures permettent l'entrée, au clavier, d'un entier, d'un réel, d'un caractère ou d'une chaîne de caractères. Si Ch est une chaîne de 78 caractères, il suffit d'écrire GET_LINE(Ch) ; et si à l'exécution on entre abc suivi d'un retour chariot alors Ch est la chaîne formée des caractères a, b et c suivis de 75 blancs. Pour les entiers et les réels on utilisera de préférence les procédures ENTRER_ENTIER, ... (Cf § 3.2.10), où on gère les erreurs de type et de contrainte.

3.2.6 La fonction LIRE_REPONSE

Cette fonction permet d'associer à la question définie par Msg une valeur booléenne qui aura la valeur TRUE si la réponse est 'o' ou 'O' et FALSE si la réponse est 'n' ou 'N'.

On pourra l'utiliser comme suit :

```
if LIRE_REPONSE("Un autre essai? ") then
   ...........
end if ;
```

3.2.7 La procédure PAUSE

Cette procédure permet de marquer une pause pendant l'exécution d'un programme, la frappe d'une touche quelconque permet ensuite de continuer.

3.2.8 La procédure BIP

Cette procédure pourra être utilisée pour signaler une erreur en émettant un son court.

3.2.9 La procédure CHRONOMETRE

Cette procédure permet de calculer le temps nécessaire à la réalisation d'une partie, ou de la totalité, d'un programme. Avant la première instruction relative à cette partie de programme, on écrira CHRONOMETRE(0) ; ce qui déclenche le chronomètre, puis après la dernière instruction, on écrira : CHRONOMETRE(1) ; ce qui arrête le chronomètre et affiche le temps écoulé en secondes et centièmes de secondes.

3.2.10 Les procédures d'entrée des entiers et des réels

Ces procédures remplacent les instructions GET(n) et GET(x) où n est une variable de type entier et x de type réel.

L'intérêt est de signaler l'erreur si la variable introduite n'est pas du bon type ou dépasse les bornes autorisées en évitant l'arrêt de l'exécution du programme. Le nom de chacune des ces procédures est assez significatif pour comprendre de quoi il s'agit.

3.2.11 Les fonctions CHAINE_ENTIERE et CHAINE_REELLE

Ces fonctions permettent de convertir un entier (Resp. un réel) en chaîne de caractères.

3.2.12 La procédure LECTURE_REPERTOIRE

Cette procédure donne une liste de fichiers contenus dans le répertoire dont le nom et le chemin d'accès sont indiqués par la variable NOM_REPERTOIRE, de type STRING. Elle équivaut à l'utilisation de l'instruction DIR du DOS. Elle utilise les procédures FIND_FIRST et FIND_NEXT de FILE_IO.

3.2.13 La procédure LIRE_FICHIER

Cette procédure permet d'ouvrir en lecture un fichier texte dans lequel sont stockées des données. Ce fichier est recherché dans le sous-répertoire NOM_DE_REPERTOIRE (par défaut le répertoire DONNEES du disque A). A l'exécution on entre NOM_DE_FICHIER et le fichier cherché aura pour nom, NOM_DE_REPERTOIRE\NOM_DE_FICHIER. En cas de non existence de ce fichier l'erreur est signalée et on peut recommencer.

Au préalable, la liste de tous les fichiers de NOM_DE_REPERTOIRE est affichée, ce qui permet de retrouver plus facilement le fichier cherché.

3.2.14 La procédure FICHIER_EXISTE

Cette procédure teste si le fichier que l'on veut créer existe déjà, si c'est le cas on demande confirmation pour en effacer le contenu.

3.2.15 La procédure CREER_FICHIER

Cette procédure permet de créer un fichier de type texte.

3.2.16 La procédure MODE_AFFICHAGE

Cette procédure permet d'afficher les résultats d'un programme sur:

- l'écran, pour CHOIX_AFFICHAGE = ECRAN ;
- l'imprimante, pour CHOIX_AFFICHAGE = IMPRIMANTE ;
- un fichier, pour CHOIX_AFFICHAGE = FICHIER (à l'exécution, il suffit d'entrer NOM_DE_FICHIER).

Un programme utilisant cette procédure devra débuter par :

```
MODE_AFFICHAGE ;
```

et se terminer par :

```
CLOSE(IMP) ;
```

Les instructions PUT et PUT_LINE, seront alors remplacées par : PUT(IMP, et PUT_LINE(IMP, .

3.2.17 La fonction TRUNC

Calcule la partie entière d'un réel.

3.2.18 La fonction FRAC

Calcule la partie fractionnaire d'un réel.

3.2.19 Les procédures de permutation de deux entiers ou deux réels

Ces procédures permettent de permuter deux entiers ou deux réels.

3.2.20 Les procédures VAL

La première procédure permet de rechercher un entier dans une chaîne de caractères Msg. i est l'entier trouvé. Si tout ce passe bien Err vaut 0, sinon il indique la position du premier caractère non entier de la chaîne.

La seconde permet la recherche d'un réel dans une chaîne de caractères. x est le réel trouvé. Si tout ce passe bien Err vaut 0, sinon il indique la position du premier caractère non correct de la chaîne.

3.2.21 La fonction LONGUEUR_CHAINE

Cette procédure permet de calculer la longueur d'une chaîne de caractères, sans compter les derniers caractères blancs.

3.2.22 Un exemple de programme utilisant MATH0

Le programme suivant permet de rechercher tous les nombres premiers inférieurs à un nombre donné en utilisant le crible d'Eratosthène.

```
with TEXT_IO, IIO, LIO, CRT, COMMON_MATH0, MATH0 ;
use TEXT_IO, IIO, LIO, CRT, COMMON_MATH0, MATH0 ;
procedure PREMIERS is

type TABLEAU_BOOLEENS is array(LONG_INTEGER range <>) of BOOLEAN ;
nMax : constant LONG_INTEGER := 60_000 ;
n : LONG_INTEGER ;

procedure MENU_PREMIER(n : out LONG_INTEGER) is
```

```
begin
    CLRSCR ;
    PUT_LINE("RECHERCHE DE NOMBRES PREMIERS") ; NEW_LINE ;
    ENTRER_ENTIER_LONG_BORNE(2,nMax,n,
                          "Borne superieure de recherche (m > 2) : ") ;
    CLRSCR ;
end MENU_PREMIER ;

procedure CALCUL_PREMIERS(n : in LONG_INTEGER) is
k, m, COMPTEUR : LONG_INTEGER ;
C : TABLEAU_BOOLEENS(1..n) ;
begin
    CHRONOMETRE(0) ;
    k := 2 ; m := k ;
    COMPTEUR := 0 ;
    C := (others => TRUE) ;
    C(1) := FALSE ;
    while (k < n)
    loop
       if c(k) then
          if ( (COMPTEUR mod 9) = 0) then
             PUT_LINE(IMP," ") ;
          end if ;
          COMPTEUR := COMPTEUR + 1 ;
          PUT(IMP,k,8) ;
          m := k ;
          while (m < n - k)
          loop
               m := m + k ;
               c(m) := FALSE ;
          end loop ;
       end if ;
       k := k + 1 ;
    end loop ;
    NEW_LINE(IMP,2) ;
```

```
    CHRONOMETRE(1) ;
    PUT(IMP,"Nombre d'entiers premiers : ") ;
    PUT(IMP,COMPTEUR) ; PUT_LINE(IMP," ") ;
end CALCUL_PREMIERS ;
begin
    MODE_AFFICHAGE ;
    MENU_PREMIER(n) ;
    CALCUL_PREMIERS(n) ;
    CLOSE(IMP) ;
end PREMIERS ;
```

4. Le paquetage MATH1

Dans ce paquetage, on définit les fonctions usuellement utilisées en mathématiques. Ce sont les fonctions :

SQRT : pour la racine carrée d'un réel positif ;
LN : pour le logarithme népérien d'un réel positif ;
LOG : pour le logarithme dans une base quelconque ;
EXP : pour l'exponentielle ;
*** :* pour la fonction puissance ;
SIN : pour le sinus ;
COS : pour le cosinus ;
TAN : pour la tangente ;
COT : pour la cotangente ;
ARCSIN : pour l'arcsinus d'un réel entre –1.0 et 1.0 ;
ARCCOS : pour l'arccosinus d'un réel entre –1.0 et 1.0 ;
ARCTAN : pour l'arctangente d'un réel ;
SH : pour le sinus hyperbolique ;
CH : pour le cosinus hyperbolique ;
TH : pour la tangente hyperbolique ;
ARGSH : pour l'argument sinus hyperbolique ;
ARGCH : pour l'argument cosinus hyperbolique d'un réel >= 1.0 ;
ARGTH : pour l'argument tangente hyperbolique d'un réel entre –1.0 et 1.0 ;
FACT : pour la factorielle d'un entier ;
GAMMA : pour la fonction Gamma d'Euler ;
LNGAMMA : pour le logarithme népérien de Gamma ;
MIN : pour le minimum de deux réels ;
MAX : pour le maximum de deux réels ;

L'exception ERREUR_ARGUMENT est levée quand l'argument fourni à une fonction ne vérifie pas les bonnes conditions (Par exemple, LN(x), avec x négatif ou nul).

Pour ce paquetage, on utilise le paquetage MATH_LIB fourni avec OPEN_ADA. Dans MATH_LIB sont définies les 6 fonctions de base : SIN, COS, EXP, SQRT, LN et ATAN agissant sur des paramètres de type FLOAT. Pour une version générique de ce paquetage on consultera le paquetage GENERIC_ELEMENTARY_FUNCTION fourni avec OpenAda.

Le compilateur FirstAda de Alsys fournit un paquetage générique, non documenté, MATH_LIB qui permet d'écrire une version générique de MATH1 pour ce compilateur.

4.1 Spécifications des paquetages COMMON_MATH1 et MATH1

```
package COMMON_MATH1 is
PI: constant FLOAT := 3.141_592_653_589_793 ;
PI_SUR_DEUX: constant FLOAT := 0.5*PI ;
PI_SUR_QUATRE : constant FLOAT := 0.25*PI ;
DEUX_PI : constant FLOAT := 2.0*PI ;
E : constant FLOAT := 2.718_281_828_459_045 ;
ERREUR_ARGUMENT : exception ;
end COMMON_MATH1 ;

package MATH1 is
 function SQRT(x : FLOAT) return FLOAT ;
 function LN(x : FLOAT) return FLOAT ;
 function LOG(x : FLOAT ; BASE : in FLOAT := 10.0) return FLOAT ;
 function EXP(x : FLOAT) return FLOAT ;
 function "**"(x : FLOAT ; n : INTEGER) return FLOAT ;
 function "**"(x,y : FLOAT) return FLOAT ;
 function SIN(x : FLOAT) return FLOAT ;
 function COS(x : FLOAT) return FLOAT ;
 function TAN(x : FLOAT) return FLOAT ;
 function COT(x : FLOAT) return FLOAT ;
 function ARCSIN(x : FLOAT) return FLOAT ;
 function ARCCOS(x : FLOAT) return FLOAT ;
 function ARCTAN(x : FLOAT) return FLOAT ;
 function SH(x : FLOAT) return FLOAT ;
 function CH(x : FLOAT) return FLOAT ;
 function TH(x : FLOAT) return FLOAT ;
 function ARGSH(x : FLOAT) return FLOAT ;
 function ARGCH(x : FLOAT) return FLOAT ;
 function ARGTH(x : FLOAT) return FLOAT ;
 function FACT(n : INTEGER) return FLOAT ;
 function LNGAMMA(x : FLOAT) return FLOAT ;
 function GAMMA(x : FLOAT) return FLOAT ;
 function MIN(x, y : FLOAT) return FLOAT ;
 function MAX(x, y : FLOAT) return FLOAT ;
end MATH1 ;
```

4.2 Documentation du paquetage MATH1

4.2.1 Les fonctions Puissances

La première associe à $x \in R$ et $n \in Z$ la puissance $n^{ème}$ de x, x^n , avec $x \neq 0$ pour $n < 0$ et la seconde associe à $x > 0$ et $y \in R$ la puissance : $x^y = e^{y \cdot Ln(x)}$

4.2.2 Les fonctions ArcTan, ArcSin et ArcCos

Si on dispose seulement de la fonction Arctangente, les fonctions ArcSin et ArcCos peuvent être définies par :

$$\text{ArcSin}(x) = \text{ArcTan}\left(\frac{x}{\sqrt{1-x^2}}\right), \text{ pour } |x| < 1.0$$

et

$$\text{ArcSin}(1) = -\text{ArcSin}(-1) = \frac{\pi}{2}$$

puis,

$$\text{ArcCos}(x) = \frac{\pi}{2} - \text{ArcSin}(x)$$

4.2.3 Les fonctions hyperboliques inverses

Elles sont définies par :

$$\text{ArgCh}(x) = \text{Ln}\left(x + \sqrt{x^2-1}\right), \text{ pour } x \geq 1$$

$$\text{ArgSh}(x) = \text{Ln}\left(x + \sqrt{x^2+1}\right), \text{ pour } x \in R$$

$$\text{ArgTh}(x) = \frac{1}{2} \cdot \text{Ln}\left(\frac{1+x}{1-x}\right), \text{ pour } |x| < 1$$

4.2.4 Les fonctions Gamma et LnGamma

La fonction Γ est définie pour $x > 0$ par :

$$\forall x > 0, \Gamma(x) = \int_0^{+\infty} t^{x-1} \cdot e^{-t} dt$$

On a alors la relation fonctionnelle :

$$\Gamma(x+1) = x \cdot \Gamma(x)$$

de laquelle on déduit que:

$$\forall n \in N, \Gamma(n+1) = n!$$

Il y a plusieurs méthodes de calcul de Γ, l'une des plus rapide étant celle donnée par une formule du type Stirling (Cf. Numerical Recipes p. 157) :

$$\frac{\Gamma(x)}{\sqrt{2 \cdot \pi}} \approx (x+4.5)^{x-0.5} \cdot e^{-(x+4.5)} \cdot \left(1 + \frac{C_1}{x} + \frac{C_2}{x+1} + \ldots + \frac{C_6}{x+5}\right)$$

L'erreur commise étant de l'ordre de 10^{-10} pour $x > 1$.

Pour éviter les débordements arithmétiques, il est préférable de calculer $\text{Ln}(\Gamma(x))$, pour $x > 1$, avec la formule ci-dessus. Le calcul pour $0 < x < 1$ se fait avec la relation fonctionnelle, puis celui de $\Gamma(x)$ avec $\text{EXP}(\text{Ln}(\Gamma(x)))$.

5. Le paquetage Math2

Dans ce paquetage, on définit un type FONCTION ainsi que des procédures permettant de manipuler des variables de ce type. Ce paquetage est inspiré du livre de Ducamp et Reverchon : Mathématiques en Turbo-Pascal, Vol. 1.

5.1 Spécifications du paquetage MATH2

```
package MATH2 is
MAX_VARIABLES : constant INTEGER := 99 ;
subtype INDICE_VARIABLES is INTEGER range 0..MAX_VARIABLES ;
type VARIABLE_2 is array(MAX_VARIABLES range <>) of FLOAT ;
type FONCTION(MAX_CHAINE : NATURAL := 80) is limited private ;
function EVALUE(f : FONCTION ; x : FLOAT ; y : VARIABLE_2)
                return FLOAT ;
function EVALUE(f : FONCTION ; x, y, z, t : FLOAT := 0.0)
                return FLOAT ;
procedure DETRUIT_FONCTION(f : in out FONCTION) ;
procedure GET_LINE(f : in out FONCTION ; Msg : in STRING := " ") ;
function TEXTE_DE_FONCTION(f : in FONCTION) return STRING ;
private
type TYPE_NOEUD is
      (CONSTANTE, VARIABLE, OPERATION_UNAIRE, OPERATION_BINAIRE) ;
type UNE_OPERATION_UNAIRE is
     (FMOINS, FPLUS, FSIN, FCOS, FTG, FARCSIN, FARCCOS,
      FARCTAN, FSH, FCH, FTH, FARGSH, FARGCH, FARGTH,
      FABS, FFRAC, FEXP, FLN, FGAMMA, FLNGAMMA, FSQRT) ;
type UNE_OPERATION_BINAIRE is (FSUM, FSUB, FMUL, FDIV, FPUI) ;
type NOEUD ;
type POINTEUR_NOEUD is access NOEUD ;
type NOEUD is
      record
         GAUCHE, DROITE : POINTEUR_NOEUD ;
         TYPE_DU_NOEUD : TYPE_NOEUD ;
         VALEUR : FLOAT ;
         RANG : INTEGER ;
         OPERATION_1 : UNE_OPERATION_UNAIRE ;
         OPERATION_2 : UNE_OPERATION_BINAIRE ;
      end record ;
type FONCTION(MAX_CHAINE : NATURAL := 80) is
      record
         TEXTE_FONCTION : STRING(1..MAX_CHAINE) := (others => ' ') ;
         ARBRE : NOEUD ;
      end record ;
end MATH2 ;
```

5.2 Documentation de Math2

5.2.1 Les fonctions EVALUE

Les fonctions d'évaluation :

```
EVALUE(f : Fonction ; x : FLOAT ; y : VARIABLE_2) return FLOAT ;
EVALUE(f : Fonction ; x, y, z, t : FLOAT := 0.0) return FLOAT ;
```

interprètent la fonction f pour retourner respectivement les valeurs réelles f(x,y) et f(x) ou f(x,y) ou f(x,y,z) ou f(x,y,z,t), où x est réelle et y est un vecteur ayant au plus MAX_VARIABLES + 1 composantes dans le premier cas et x, y, z, t sont réelles dans le second cas. Pour une fonction d'une variable, on écrira r := Evalue(f,x) ; pour une fonction de deux variables, r := Evalue(f,x,y) ; ...

5.2.2 La procédure DETRUIT_FONCTION

Cette procédure permet d'éliminer une fonction f en vue de libérer l'espace mémoire occupé.

5.2.3 La procédure GET_LINE

Cette procédure permet de lire au clavier le texte du corps d'une fonction f et de créer l'objet f de type FONCTION.

5.2.4 La fonction TEXTE_DE_FONCTION

Cette fonction retourne le texte de la fonction f sans les derniers caractères blancs.

6. Le paquetage GRAPH

Dans ce paquetage on définit des procédures graphiques de base. Ce paquetage étant lié au matériel n'est pas facilement portable. On utilise les interruptions DOS.

6.1 Spécifications des paquetages COMMON_GRAPH et GRAPH

```
package COMMON_GRAPH is
type CARTE_GRAPHIQUE is (CGA_C4_320x200, CGA_GREY4_320x200,
   CGA_C2_640x200, EGA_C16_320x200, EGA_C16_640x200, EGA_BW_640x350,
   EGA_VGA_C16_320x200, EGA_VGA_C16_640x200,EGA_VGA_BW_640x350,
   EGA_VGA_C16_640x350, MCGA_BW_640x480, MCGA_C256_320x200,
   VGA_C16_640x480, SVGA_C16_800x600, SVGA_C16_1024x768) ;
type COULEUR is (NOIR, BLEUE, VERT, CYAN, ROUGE, MAGENTA, BRUN,
                 GRIS_CLAIR, GRIS_FONCE, BLEUE_CLAIR, VERT_CLAIR,
                 CYAN_CLAIR, ROUGE_CLAIR, MAGENTA_CLAIR, JAUNE, BLANC) ;
subtype TAILLE_CARACTERE is integer range 8 .. 16 ;
subtype TEXTE_HORIZONTAL is INTEGER range 0..127 ;
subtype TEXTE_VERTICAL is INTEGER range 0..47 ;
CARTE_GRAPHIQUE_UTILISATEUR : CARTE_GRAPHIQUE ;
MAX_LIGNE_TEXTE : TEXTE_VERTICAL ;
MAX_COLONNE_TEXTE : TEXTE_HORIZONTAL ;
CLIP_ON : BOOLEAN ;
GR_X_MIN, GR_X_MAX, GR_Y_MIN, GR_Y_MAX, GET_MAX_X, GET_MAX_Y : NATURAL;
LARGEUR_PIXEL, HAUTEUR_PIXEL : TAILLE_CARACTERE ;
X_POS, Y_POS : INTEGER ;
COULEUR_PIXEL : COULEUR := GRIS_CLAIR ;
end COMMON_GRAPH ;

with COMMON_GRAPH ;
use COMMON_GRAPH ;
package GRAPH is
procedure CLEAR_SCREEN ;
procedure CLOSE_GRAPH ;
procedure INIT_GRAPH(CHOIX : CARTE_GRAPHIQUE) ;
procedure MOVE_TO(x : in INTEGER ; y : in INTEGER) ;
procedure PUT_PIXEL(x : in INTEGER ; y : in INTEGER) ;
procedure LINE(x, x1 : in INTEGER ; y, y1 : in INTEGER) ;
```

```
procedure LINE_TO(x : in INTEGER ; y : in INTEGER) ;
procedure RECTANGLE(x, x1 : in INTEGER ; y, y1 : in INTEGER ;
                    CLIP : in BOOLEAN := FALSE) ;
end GRAPH ;
```

6.2 Documentation de GRAPH

6.2.1 La procédure CLEAR_SCREEN

Cette procédure permet d'effacer l'écran en mode graphique.

6.2.2 La procédure CLOSE_GRAPH

Cette procédure permet de revenir en mode texte : 25 lignes et 80 colonnes.

6.2.3 La procédure INIT_GRAPH

Cette procédure permet d'initialiser un mode graphique.

6.2.4 La procédure MOVE_TO

Cette procédure permet de mémoriser la position d'un point.

6.2.5 La procédure PUT_PIXEL

Cette procédure permet de se déplacer en (x,y) en allumant un pixel avec la couleur spécifiée par COULEUR_PIXEL (par défaut c'est GRIS_CLAIR).

6.2.6 La procédure LINE

Cette procédure permet de tracer le segment d'extrémités (x,y) et (x1,y1). le curseur étant ensuite positionné en (x1,y1).

6.2.7 La procédure LINE_TO

Cette procédure permet de tracer un segment depuis la position actuelle du curseur jusqu'au point de coordonnées (x,y).

6.2.8 La procédure RECTANGLE

Cette procédure permet de tracer un rectangle de coin supérieur gauche (x,y) et de coin inférieur droit (x1, y1).

7. Le paquetage MATH3

7.1 Spécifications des paquetages COMMON_MATH3 et MATH3

La constante CHOIX_CARTE_GRAPHIQUE devra être modifiée en fonction du type d'écran sur lequel on travaille (CGA, EGA, ...). On se référera à la spécification de COMMON_GRAPH pour connaître la liste des choix possibles.

L'exception ERREUR_CONTRAINTES_GRAPHIQUES sera levée pour toute erreur de contrainte dans la définition d'une fenêtre ou d'un cadre graphique. L'exception ERREUR_MODE_ECRAN sera levée quand on tente d'utiliser une procédure graphique alors qu'on est en mode texte.

On pourra ajouter des procédures permettant d'utiliser une table traçante.

```
with COMMON_GRAPH ; use COMMON_GRAPH ;
package COMMON_MATH3 is
type MODE_ECRAN is (TEXTE, GRAPHIQUE) ;
CHOIX_CARTE_GRAPHIQUE :
```

```
        constant GRAPH.CARTE_GRAPHIQUE := VGA_C16_640x480 ;
x_MINIMUM, x_MAXIMUM, y_MINIMUM, y_MAXIMUM : NATURAL ;
HAUTEUR_CAR, LARGEUR_CAR, MARGE : NATURAL ;
GR_LARGEUR_ECRAN, GR_HAUTEUR_ECRAN : NATURAL ;
X_MIN, X_MAX, Y_MIN, Y_MAX, X_GRD, Y_GRD : FLOAT ;
LARGEUR_ECRAN, HAUTEUR_ECRAN : FLOAT ;
X_RAPPORT, Y_RAPPORT : FLOAT ;
INDIC_MODE : MODE_ECRAN := TEXTE ;
ERREUR_CONTRAINTES_GRAPHIQUES : exception ;
ERREUR_MODE_ECRAN : exception ;
end COMMON_MATH3 ;

package MATH3 is
procedure INIT_GRAPHIQUE ;
procedure MODE_TEXTE ;
procedure MESSAGE_ERREUR_GRAPHIQUE(Msg : in STRING) ;
procedure TEST_MODE_GRAPHIQUE(Msg : in STRING) ;
procedure FENETRE(X_FEN_MIN,X_FEN_MAX,
                  Y_FEN_MIN,Y_FEN_MAX : in FLOAT) ;
procedure CADRE_GRAPHIQUE(X_CADRE_MIN, X_CADRE_MAX,
                          Y_CADRE_MIN, Y_CADRE_MAX : in INTEGER) ;
procedure FENETRE_GRAPHIQUE(X_FEN_MIN,X_FEN_MAX,
                            Y_FEN_MIN,Y_FEN_MAX : in FLOAT) ;
procedure CONVERSION(x, y : in FLOAT ;
                     X_INFO : out INTEGER ; Y_INFO : out INTEGER) ;
procedure TEXT_MOVE_TO(x : in INTEGER ; y : in INTEGER) ;
procedure PAUSE_GRAPHIQUE ;
procedure DEPLACE(x, y : in FLOAT) ;
procedure POINT(x, y : in FLOAT) ;
procedure TRACE_SEGMENT(x1, y1, x2, y2 : in FLOAT) ;
procedure TRACE(x, y : in FLOAT) ;
procedure CROIX(x, y : in FLOAT) ;
procedure SORTIE_GRAPHIQUE ;
procedure CERCLE(x, y, r : in FLOAT) ;
procedure X_AXE(X_ORIGINE, Y_ORIGINE, X_UNITE : in FLOAT) ;
procedure Y_AXE(X_ORIGINE, Y_ORIGINE, Y_UNITE : in FLOAT) ;
procedure XY_AXES(X_ORIGINE, Y_ORIGINE, X_UNITE, Y_UNITE : in FLOAT ;
                  XY_GRILLE : in BOOLEAN := TRUE) ;
procedure TITRE(Msg : in STRING) ;
end MATH3 ;
```

7.2 Documentation de MATH3

7.2.1 La procédure INIT_GRAPHIQUE

Permet de passer en mode graphique et d'initialiser les variables du système.

7.2.2 La procédure MODE_TEXTE

Quitte le mode graphique et repasse en mode texte.

7.2.3 La procédure MESSAGE_ERREUR_GRAPHIQUE

En cas d'erreur dans une procédure graphique, cette procédure permet d'arrêter l'exécution du programme en retournant en mode texte et en donnant un message sur l'erreur ayant provoqué cet arrêt.

7.2.4 La procédure TEST_MODE_GRAPHIQUE

Cette procédure permet de vérifier que l'on est dans le bon mode pour utiliser une procédure graphique. Pour cela, on utilise la variable INDIC_MODE définie par :

- INDIC_MODE = GRAPHIQUE, si on est en mode graphique ;
- INDIC_MODE = TEXTE, si on est en mode texte.

7.2.5 La procédure TEXT_MOVE_TO

Cette procédure permet de se déplacer en un point de l'écran, en mode graphique, pour y écrire du texte.

7.2.6 La procédure PAUSE_GRAPHIQUE

Cette procédure interrompt momentanément le déroulement d'un programme jusqu'à ce qu'une touche soit pressée.

7.2.7 La procédure FENETRE

Initialisation des variables, pour définir une fenêtre graphique, permettant de travailler dans les unités de l'utilisateur. Ainsi, si on a donné l'instruction

```
FENETRE(-15.5,2.3,1.2,1.8) ;
```

le point situé en bas et à gauche a pour coordonnées X = –15.5 et Y = 1.2 et le point situé en haut et à droite a pour coordonnées X = 2.3 et Y = 1.8.

7.2.8 La procédure CADRE_GRAPHIQUE

Cette procédure qui doit suivre l'appel à FENETRE permet de préciser sur quel portion de l'écran graphique on travaille. Elle permet de travailler sur plusieurs fenêtres graphiques dans un même écran (pour une même fenêtre, on peut utiliser plusieurs cadres).

7.2.9 La procédure FENETRE_GRAPHIQUE

Cette procédure permet de définir une fenêtre graphique de travail avec un cadre par défaut qui est l'écran tout entier privé d'une bande en haut de l'écran et d'une bande en bas.

7.2.10 La procédure CONVERSION

Cette procédure permet d'associer aux coordonnées de l'utilisateur (x,y), dans un repère orienté de façon classique en mathématiques, les coordonnées écran (xInfo,yInfo) dans le repère « informatique », c'est-à-dire avec des coordonnées entières et les abscisses allant de gauche à droite et les ordonnées de haut en bas.

Cette procédure est nécessaire pour les procédures DEPLACE, TRACE, CROIX et POINT.

7.2.11 La procédure DEPLACE

Mémorise la position du curseur au point de coordonnées (X,Y) sans tracer.

7.2.12 La procédure TRACE

Trace un segment de droite joignant la position actuelle du curseur au point de coordonnées (X,Y). Le curseur est déplacé en ce dernier point. Le tracé se fait dans la couleur définie par COULEUR_PIXEL.

7.2.13 La procédure CROIX

Trace une croix au point de coordonnées (x,y).

7.2.14 La procédure TRACE_SEGMENT

Permet de tracer le segment d'extrémités (x1,y1) et (x2,y2). La position du curseur est mémorisée en (x2,y2). Le tracé se fait dans la couleur définie par COULEUR_PIXEL.

7.2.15 La procédure POINT

Positionne le curseur au point de coordonnées X,Y et dessine ce point dans la couleur précisée par COULEUR_PIXEL.

7.2.16 La procédure CERCLE

Trace un cercle de centre (x,y) et de rayon r. Le tracé se fait dans la couleur définie par COULEUR_PIXEL.

Le tracé se fait par récurrence de la façon suivante : soit un cercle de centre 0 et de rayon R et (x ,y) un point de ce cercle ; le point déduit par une rotation d'angle dθ est donné par:

$$\begin{cases} x_{n+1} = \mathrm{Cos}(d\theta)\cdot x_n - \mathrm{Sin}(d\theta)\cdot y_n \\ y_{n+1} = \mathrm{Sin}(d\theta)\cdot x_n + \mathrm{Cos}(d\theta)\cdot y_n \end{cases}$$

En prenant dθ assez petit on peut ainsi tracer rapidement un cercle.

7.2.17 La procédure TITRE

Efface le dernier titre en date puis dessine un petit rectangle, en haut de l'écran, dans lequel sera écrit le titre Msg.

7.2.18 La procédure X_AXE

Cette procédure permet de tracer un axe gradué dans la direction Ox passant par le point de coordonnées X_ORIGINE et Y_ORIGINE et de graduation X_UNITE.

7.2.19 La procédure Y_AXE

Procédure analogue à X_AXE concernant la direction Oy.

7.2.20 La procédure XY_AXES

Cette procédure permet de tracer les axes des x et y avec la possibilité de tracer une grille pour mieux visualiser les graduations.

7.2.21 La procédure SORTIE_GRAPHIQUE

Permet de sortir du mode graphique.

7.2.22 Un exemple d'utilisation de MATH2 et de MATH3

Dans ce paragraphe on décrit tout d'abord le paquetage CURVE, où on définit un type COURBE avec des procédures permettant le tracé de courbes dans le plan, puis le paquetage CARTESIENNE est utilisé pour le tracé de courbes d'équation cartésienne y = f(x) sur un intervalle [a,b].

— Spécification de CURVE

```
package CURVE is
NB_POINTS : constant NATURAL := 800 ;
type COORDONNEE is array(NATURAL range <>) of FLOAT ;
type COURBE(n : NATURAL) is
     record
         uMin, uMax, vMin, vMax : FLOAT ;
```

```
        u, v : COORDONNEE(0..n) ;
        Rep_Axes, Rep_Grille : BOOLEAN ;
      end record ;
```

Le rectangle de travail est [uMin,uMax]x[vMin,vMax], le nombre de points de la courbe est u'LENGTH = v'LENGTH = n + 1, les abscisses sont consignées dans le vecteur u et les ordonnées dans v. Si Rep_Axes = TRUE, on trace les axes et si Rep_Grille = TRUE, on trace une grille pour mieux visualiser les graduations.

```
procedure CHOIX_AXES(xOrigine , yOrigine , xUnite , yUnite ,
                  xFenMin , xFenMax , yFenMin , yFenMax : out FLOAT) ;
procedure AXES_PAR_DEFAUT(xMin , yMin , xMax , yMax : in FLOAT ;
               xOrigine , yOrigine , xUnite , yUnite ,
               xFenMin , xFenMax , yFenMin , yFenMax : in out FLOAT) ;
procedure SYSTEME_AXES(xMin , yMin , xMax , yMax : in FLOAT ;
               xOrigine , yOrigine , xUnite , yUnite ,
               xFenMin , xFenMax , yFenMin , yFenMax : in out FLOAT) ;
procedure INIT_COURBE(C : in out COURBE ; xFenMin, xFenMax, yFenMin,
          yFenMax, xOrigine, yOrigine, xUnite, yUnite : in out FLOAT) ;
procedure TRACE_COURBE(C : in COURBE) ;
procedure TRACE_UNE_COURBE(C : in out COURBE) ;
end CURVE ;
```

— Corps de CURVE

```
with TEXT_IO, CRT, MATH0, COMMON_MATH3, MATH3 ;
use TEXT_IO, CRT, MATH0, COMMON_MATH3, MATH3 ;
package body CURVE is
procedure CHOIX_AXES(xOrigine , yOrigine , xUnite , yUnite ,
                  xFenMin , xFenMax , yFenMin , yFenMax : out FLOAT) is
begin
    CLRSCR ;
    PUT_LINE("Coordonnees de l'origine:") ;
    ENTRER_REEL(xOrigine," Abscisse de l'origine : ") ;
    ENTRER_REEL(yOrigine," Ordonnee de l'origine : ") ;
    NEW_LINE ;
    PUT_LINE("Choix des unites sur les axes:") ;
    ENTRER_REEL(X_GRD," Unite de l'echelle en x : ") ;
    xUnite := X_GRD ;
    ENTRER_REEL(Y_GRD," Unite de l'echelle en y : ") ;
    yUnite := Y_GRD ; NEW_LINE ;
    PUT_LINE("Coordonnees de la fenetre graphique:") ;
    ENTRER_REEL(xFenMin," Abscisse coin inferieur gauche : ") ;
    ENTRER_REEL(yFenMin," Ordonnee coin inferieur gauche : ") ;
    ENTRER_REEL(xFenMax," Abscisse coin superieur droit : ") ;
    ENTRER_REEL(yFenMax," Ordonnee coin superieur droit : ") ;
end CHOIX_AXES ;

procedure AXES_PAR_DEFAUT(xMin , yMin , xMax , yMax : in FLOAT ;
               xOrigine , yOrigine , xUnite , yUnite ,
               xFenMin , xFenMax , yFenMin , yFenMax : in out FLOAT) is
begin
    xOrigine := (xMax + xMin)/2.0 ; yOrigine := (yMin + yMax)/2.0 ;
    xUnite := (xMax - xMin)/20.0 ; X_GRD := xUnite ;
    yUnite := (yMax - yMin)/20.0 ; Y_GRD := yUnite ;
    xFenMin := xMin - xUnite ; xFenMax := xMax + xUnite ;
```

```
    yFenMin := yMin - yUnite ;
    yFenMax := yMax + yUnite ;
end AXES_PAR_DEFAUT ;

procedure SYSTEME_AXES(xMin , yMin , xMax , yMax : in FLOAT ;
                xOrigine , yOrigine , xUnite , yUnite ,
                xFenMin , xFenMax , yFenMin , yFenMax : in out FLOAT) is
CHOIX : INTEGER ;
begin
    CLRSCR ;
    PUT_LINE(" 1 : Systeme d'axe par defaut.") ;
    PUT_LINE(" Si les abscisses varient dans l'intervalle [xMin,xMax]");
    PUT_LINE(" et les ordonnees dans [yMin,yMax], on pose alors:") ;
    PUT_LINE(" Abscisse de l'origine = (xMax + xMin)/2.0 ;") ;
    PUT_LINE(" Ordonnee de l'origine = (yMax + yMin)/2.0 ;") ;
    PUT_LINE(" Unite sur l'axe des x = (xMax - xMin)/20.0 ;") ;
    PUT_LINE(" Unite sur l'axe des y = (yMax - yMin)/20.0 ;") ;
    PUT("Abscisse du coin inferieur gauche de la fenetre graphique = ");
    PUT_LINE("xMin - xUnite ;") ;
    PUT("Ordonnee du coin inferieur gauche de la fenetre graphique = ");
    PUT_LINE("yMin - yUnite ;") ;
    PUT("Abscisse du coin superieur droit de la fenetre graphique = ") ;
    PUT_LINE("xMax + xUnite ;") ;
    PUT("Ordonnee du coin superieur droit de la fenetre graphique = ") ;
    PUT_LINE("yMax + yUnite .") ;
    NEW_LINE ;
    PUT_LINE(" 2 : Votre systeme d'axes.") ;
    NEW_LINE ;
    ENTRER_ENTIER_BORNE(1,2,CHOIX,"Votre choix : ") ;
    CLRSCR ;
    if CHOIX = 1 then
       AXES_PAR_DEFAUT(xMin,yMin,xMax,yMax,xOrigine,yOrigine,
                       xUnite,yUnite, xFenMin,xFenMax,yFenMin,yFenMax) ;
    else
       CHOIX_AXES(xOrigine,yOrigine,xUnite, yUnite,
                   xFenMin,xFenMax,yFenMin,yFenMax) ;
    end if ;
end SYSTEME_AXES ;
procedure INIT_COURBE(C : in out COURBE ; xFenMin, xFenMax,
                       yFenMin, yFenMax,  xOrigine, yOrigine,
                       xUnite, yUnite : in out FLOAT) is
begin
    C.Rep_Axes := LIRE_REPONSE("Voulez vous le trace des axes? ") ;
    if C.Rep_Axes then
       NEW_LINE ;
   C.Rep_Grille := LIRE_REPONSE("Voulez vous le trace d'une grille? ") ;
       NEW_LINE ;
       SYSTEME_AXES(C.uMin,C.vMin,C.uMax,C.vMax,xOrigine,yOrigine,
                     xUnite,yUnite, xFenMin,xFenMax,yFenMin,yFenMax) ;
    else
       xUnite := (C.uMax - C.uMin)/20.0 ;
       yUnite := (C.vMax - C.vMin)/20.0 ;
       xFenMin := C.uMin - xUnite ;
       xFenMax := C.uMax + xUnite ;
       yFenMin := C.vMin - yUnite ;
       yFenMax := C.vMax + yUnite ;
    end if ;
```

```
end INIT_COURBE ;

procedure TRACE_COURBE(C : in COURBE) is
begin
    TEST_MODE_GRAPHIQUE("TRACE_COURBE") ;
    DEPLACE(C.u(0),C.v(0)) ;
    for i in C.u'range
    loop
       TRACE(C.u(i),C.v(i)) ;
    end loop ;
end TRACE_COURBE ;

procedure TRACE_UNE_COURBE(C : in out COURBE) is
xOrigine, yOrigine, xUnite, yUnite,
xFenMin, xFenMax, yFenMin, yFenMax : FLOAT ;
begin
    INIT_COURBE(C,xFenMin,xFenMax,yFenMin,yFenMax,xOrigine,yOrigine,
                xUnite,yUnite) ;
    INIT_GRAPHIQUE ;
    FENETRE_GRAPHIQUE(xFenMin,xFenMax,yFenMin,yFenMax) ;
    TRACE_COURBE(C) ;
    if C.Rep_Axes then
       XY_AXES(xOrigine,yOrigine,xUnite,yUnite,C.Rep_Grille) ;
    end if ;
end TRACE_UNE_COURBE ;
end CURVE ;
```

— Spécification de CARTESIENNE

```
with MATH2, CURVE ;
use MATH2, CURVE ;
package CARTESIENNE is
procedure DONNEES_CARTESIENNE(f : in out FONCTION ; C : in out COURBE);
procedure TRACE_CARTESIENNE(f : in out FONCTION ; C : in out COURBE) ;
end CARTESIENNE ;
```

— Corps de CARTESIENNE

```
with TEXT_IO, FIO, MATH0, MATH1, MATH2, MATH3, CURVE ;
use TEXT_IO, FIO, MATH0, MATH1, MATH2, MATH3, CURVE ;

package body CARTESIENNE is

procedure DONNEES_CARTESIENNE(f : in out FONCTION ;
                              C : in out COURBE) is
Pas : FLOAT ;
begin
    GET_LINE(f,"Entrer la fonction f(x) : ") ;
    ENTRER_REEL(C.uMin,"Valeur minimale de l'abscisse, a = ") ;
    ENTRER_REEL_BORNE(C.uMin,1.0E+6,C.uMax,
                      "Valeur maximale de l'abscisse, b = ") ;
    NEW_LINE ;
    PAS := (C.uMax - C.uMin)/FLOAT(C.n) ;
    C.vMIN := 1.0E+37 ;  C.vMAX := -C.vMIN ;
    C.u(0) := C.uMin ; C.v(0) := Evalue(f,C.u(0)) ;
    for i in 1..C.n
    loop
```

```
      C.u(i) := C.u(i-1) + PAS ;
      C.v(i) := Evalue(f,C.u(i)) ;
      C.vMIN := MIN(C.vMIN,C.v(i-1)) ;
      C.vMAX := MAX(C.vMAX,C.v(i-1)) ;
    end loop ;
end DONNEES_CARTESIENNE ;

procedure TRACE_CARTESIENNE(f : in out FONCTION ; C : in out COURBE) is
begin
    DONNEES_CARTESIENNE(f,C) ;
    TRACE_UNE_COURBE(C) ;
    TITRE(" Graphe de f(x) = " & TEXTE_DE_FONCTION(f)) ;
    SORTIE_GRAPHIQUE ;
end TRACE_CARTESIENNE ;

end CARTESIENNE ;
```

— Un exemple d'utilisation

```
with TEXT_IO, MATH0, COMMON_MATH2, CURVE, CARTESIENNE ;
use TEXT_IO, MATH0, COMMON_MATH2, CURVE, CARTESIENNE ;

procedure ESS_CART is
NOMBRE_DE_POINTS : NATURAL ;
begin
    PUT_LINE(" TRACE D'UNE COURBE D'EQUATION Y = F(X) SUR [A,B].") ;
    NEW_LINE(3) ;
    ENTRER_ENTIER_BORNE(2,NB_POINTS,NOMBRE_DE_POINTS,
                        "Nombre de points de la courbe : ") ;
```

```
    declare
       C : COURBE(NOMBRE_DE_POINTS) ;
       f : FONCTION ;
    begin
       TRACE_CARTESIENNE(f,C) ;
    end ;
end ESS_CART ;
```

8. Exercices

8.1 Calculs sur les nombres rationnels

Le but est d'écrire des procédures permettant d'effectuer des calculs de façon exacte sur les nombres rationnels. Ces procédures seront stockées dans un paquetage.

On devra donc :

- définir un type rationnel ;
- écrire une procédure de simplification d'un rationnel (i. e. l'écrire sous forme irréductible) ;
- définir les 4 opérations élémentaires sur les rationnels.

A titre d'application on pourra écrire des procédures de manipulation des matrices à coefficients dans Q, ainsi qu'une procédure de résolution d'un système linéaire à coefficients dans Q.

On utilisera cette procédure pour calculer les coefficients qui interviennent dans les méthodes de Newton-Cotes décrites dans le chapitre sur le calcul numérique des intégrales.

On pourra aussi imaginer d'autres applications.

8.2 Approximation d'un réel par des fractions continues

Le problème est, étant donné un réel x, de construire une suite de nombres rationnels qui converge vers x.

Les fractions continues permettent de construire une telle suite de rationnels $(r_n) = (p_n/q_n)$, écrits sous forme irréductible.

Algorithme de calcul :

On définit d'abord, la fonction :

$$f : R - Z \to R$$

$$x \mapsto f(x) = \frac{1}{x - E[x]}$$

où E[x] est la partie entière de x, c'est à dire l'entier qui vérifie $E[x] \leq x < E[x] + 1$. Puis on définit les suites (x_n) et (a_n) par :

$$\begin{cases} x_0 = x;\ a_0 = E[x_0]; \\ \text{si } x_n \in Z, \text{ alors on arrête;} \\ \text{sinon, } x_n = f(x_{n-1}) \text{ et } a_n = E[x_n] \end{cases}$$

La suite d'entiers (a_n) est formée d'une infinité de termes si, et seulement si x n'est pas rationnel.

On note N_{max} , le dernier indice pour lequel a_n est défini, soit :

$$N_{max} = \begin{cases} +\infty \text{ si } x \in R - Q \\ < +\infty \text{ si } x \in Q \end{cases}$$

Pour $0 \le n \le N_{max}$,on pose alors :

$$r_n = a_0 + \cfrac{1}{a_1 + \cfrac{\cdots\cdots\cdots\cdots}{a_{n-1} + \cfrac{1}{a_n}}}.$$

et on a : $r_n = \dfrac{p_n}{q_n}$, où les entiers p et q sont premiers entre eux et vérifient la même récurrence :

$$(R)\ u_n = a_n \cdot u_{n-1} + u_{n-2}\ , \text{ pour } n > 1$$

avec les valeurs initiales :

$$\begin{cases} p_0 = a_0;\ p_1 = a_0 \cdot a_1 + 1 \\ q_0 = 1;\ q_1 = a_1 \end{cases}$$

On a alors :

$$\left| x - \frac{p_n}{q_n} \right| \le \frac{1}{q_n^2}$$

avec, pour $N_{max} = +\infty$, $\underset{n \to +\infty}{\text{Lim}}(q_n) = +\infty$, et $x = r_{N_{max}}$ si $x \in Q$.

Ecrire une procédure correspondante à l'algorithme décrit ci-dessus.

8.3 Calculs en multiprécision sur les entiers

Le but de l'exercice est de calculer, i.e. faire des additions, soustractions, multiplications et divisions avec des entiers ayant au plus N_{max} chiffres où N_{max} est aussi grand que possible.

Les procédures et fonctions écrites seront rangées dans un paquetage.

Pour ce faire, une idée est d'écrire un entier positif non nul en base 10, soit :

$$a = \sum_{k=0}^{n(a)} a_k \cdot 10^k\ , \text{ avec } 0 \le a_k < 10 \text{ et } a_{n(a)} \ne 0;$$

puis de ranger les a dans un vecteur :

$$A = \left(a_0,\ a_1,\ \ldots,\ a_{n(a)},\ 0,\ \ldots,\ 0\right) \in N^{T_{max}}$$

où T_{max} est le nombre maximum chiffres que l'on a choisi.

Si maintenant, on veut travailler sur Z, on aura intérêt à stocker les a dans un vecteur à $T_{max} + 1$ composantes $A = \left(A_i\right)_{0 \le i \le T_{max}+1}$, avec :

$$\begin{cases} A_k = a_{k-1} \text{ pour } k \ge 1; \\ |A_0| = \text{Nombre de chiffres de a, si } a \ne 0; \\ A_0 \text{ du signe de a, si } a \ne 0 \text{ et } A_0 = 0, \text{ si } a = 0 \end{cases}$$

1°) Définir un type Entier correspondant à la description ci-dessus.
2°) Ecrire une procédure de lecture d'un entier.
3°) Ecrire une procédure d'affichage d'un entier.
4°) Définir une fonction Longueur qui donne le nombre de chiffres d'un entier.
5°) Décrire des procédures d'addition, de soustraction et de multiplication qui opèrent sur des variables de type Entier.

6°) Décrire une procédure de division euclidienne qui donne le reste et le quotient dans la division euclidienne de deux entiers.

8.4 Opérations sur les polynômes

Définir un type polynôme réel, puis des procédures permettant de faire les calculs usuels : addition, soustraction, multiplication, division euclidienne, division suivant les puissances croissantes, calcul de pgcd et ppcm, évaluation d'un polynôme, dérivation, intégration, composition de deux polynômes, opérations sur les fractions rationnelles, ...

Applications **:** Ecrire des procédures de calcul des polynômes orthogonaux classiques (Cf. le chapitre sur le calcul numérique des intégrales).

8.5 Ensembles de Mandelbrot et de Julia

1°) On appelle ensemble de Mandelbrot, une partie de C définie par:

$$M = \left\{ z \in C;\ (z_n)_{n\in N} \text{ converge} \right\}$$

où (z_n) est la suite définie par :

$$\begin{cases} z_0 = z \\ z_{n+1} = z_n^2 + z \quad (n \geq 0) \end{cases}$$

Ecrire une procédure permettant de dessiner cet ensemble.

2°) Si dans l'ensemble ci-dessus, on se fixe z et on fait varier la valeur initiale z_0 , alors l'ensemble obtenu est appelé ensemble de Julia. Soit:

$$G = \left\{ z_0 \in C;\ (z_n)_{n\in N} \text{ converge} \right\}$$

où (z_n) est la suite définie par:

$$\begin{cases} z_0 \text{ donné dans C} \\ z_{n+1} = z_n^2 + z \quad (n \geq 0) \end{cases}$$

z étant donné.

Ecrire une procédure permettant de dessiner cet ensemble.

8.6 Courbe de Van Koch

La courbe de Van Koch est une courbe continue dérivable nulle part (fractale), de longueur infinie et bornée.

Le principe de la construction est le suivant :

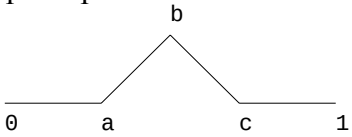


Figure 2.1

Partant de [0,1] on remplace ce segment par 4 autres disposés comme sur le schéma ci-contre, puis on recommence indéfiniment avec chacun des nouveaux segments.

1°) La courbe obtenue à l'étape n est alors polygonale. Calculer le nombre de sommets de cette courbe.

2°) Ecrire une procédure de tracé d'une telle courbe.

3°) Généralisation : au lieu d'un triangle équilatéral, on peut choisir un motif quelconque défini par un courbe polygonale joignant 0 à 1 et on reproduit ce motif sur chacun des segments du motif précédent.

Ecrire une procédure de définition du motif (c'est le générateur de la courbe), puis une procédure de tracé de la fractale obtenue.

8.7 Transformations géométriques du plan

Ecrire des procédures de transformations géométriques dans le plan qui permettent, pour une figure donnée D, d'en déduire les transformées par : une rotation, une homothétie, une symétrie, ..., et toute transformation obtenue par composition.

Une transformation affine du plan peut être décrite par une matrice 3x3 :

$$M = \begin{pmatrix} a & b & m \\ c & d & n \\ 0 & 0 & 1 \end{pmatrix}$$

où $A = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$ est la matrice de la transformation vectorielle associée et $\begin{pmatrix} m \\ n \end{pmatrix}$ est le vecteur de translation.

Les coordonnées $\begin{pmatrix} x' \\ y' \end{pmatrix}$ de l'image de $\begin{pmatrix} x \\ y \end{pmatrix}$ sont alors données par :

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = M \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

Par exemple la rotation d'angle θ autour du point (α,β) peut être décomposée en une translation de vecteur $(-\alpha,-\beta)$ (qui nous ramène à l'origine), une rotation de centre (0,0) et d'angle θ et une translation de vecteur (α,β). Ce qui donne :

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & \alpha \\ 0 & 1 & \beta \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & -\alpha \\ 0 & 1 & -\beta \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

La figure que l'on veut déformer sera définie par un fichier de points.

8.8 Géométrie dans l'espace

Pour une représentation plane de l'espace, on peut utiliser une perspective cavalière décrite par les formules :

$$\begin{cases} X = X_0 + x \cdot \cos(\alpha) + y \cdot \cos(\beta) \\ Y = Y_0 + x \cdot \sin(\alpha) + y \cdot \sin(\beta) - z \end{cases}$$

où (x,y,z) sont les coordonnées d'un point de l'espace, avec l'axe oz vertical, α est l'angle entre OX et ox, β l'angle entre OX et oy et (X_0,Y_0) sont les coordonnées de o dans le plan :

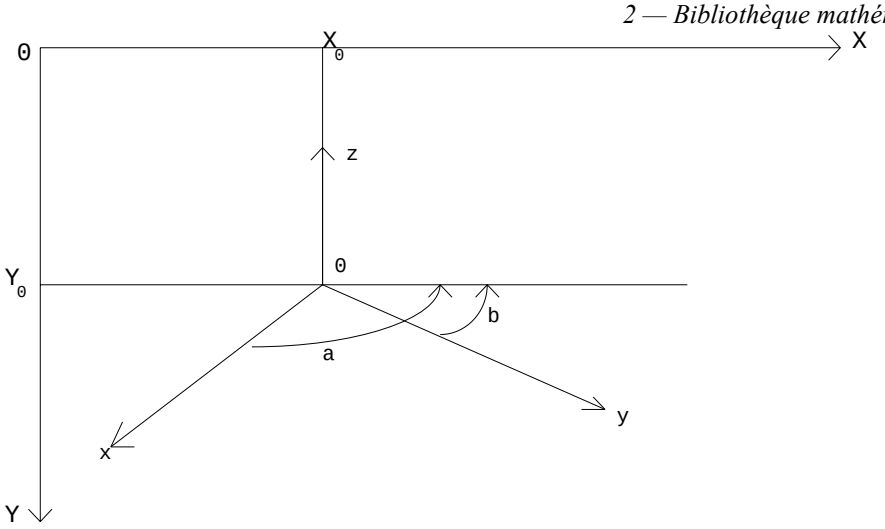


Figure 2.2

On écrira :

- des procédures permettant le tracé de courbes gauches définies par:

$$\begin{cases} x = f(t) \\ y = g(t) \\ z = h(y) \end{cases}$$

- des procédures permettant de tracer les projections de la courbe sur chacun des plans xoy, xoz et yoz ;
- des procédures permettant le tracé « en fil de fer » d'une surface d'équation:

$$z = f(x,y)$$

Si x varie de xMin à xMax et y de yMin à yMax, on pourra procéder comme suit :

— on coupe la surface par un plan d'équation x = h et on trace la courbe intersection en faisant varier y de yMin à yMax ;

— on coupe la surface par le plan y = k et on trace la courbe d'intersection en faisant varier x de xMin à xMax ;

- des procédures permettant d'éliminer les parties cachées dans le tracé précédent;
- des procédures de tracé de surfaces définies par:

$$\begin{cases} x = f(u,v) \\ y = g(u,v) \\ z = h(u,v) \end{cases}$$

avec et sans parties cachées.

8.9 Tracé de courbes paramétriques et polaires

Ecrire des paquetages permettant le tracé de courbes d'équations paramétriques x = f(t), y = g(t) et d'équation polaire r = f(θ).

8.10 Recherche des lignes de niveaux d'une fonction

Soit $f : R^2 \rightarrow R$, une fonction continue, il s'agit de tracer les courbes d'équation:

$$f(x,y) = Cste$$

On définit d'abord un rectangle de travail [a,b]x[c,d] que l'on maille en m lignes et n colonnes, en posant :

$$hx = \frac{b - a}{n} \; ; \; hy = \frac{d - c}{m}$$

$$x_i = a + i \cdot hx \; ; \; y_j = c + j \cdot hy \quad (i = 1, \ldots, n; \; j = 1, \ldots, m)$$

On pose alors :

$$F_{i,j} = f(x_i, y_j) \quad (i = 1, \ldots, n; \; j = 1, \ldots, m)$$

A partir du tableau F, on calcule :

$$fMin = Min \{ F_{i,j} , i = 1, \ldots, n, j = 1, \ldots, m \}$$
$$fMax = Max \{ F_{i,j} , i = 1, \ldots, n, j = 1, \ldots, m \}$$

Chaque petit rectangle est divisé en deux triangles et on trace les lignes de niveaux par triangle.

Soit un de ces triangles :

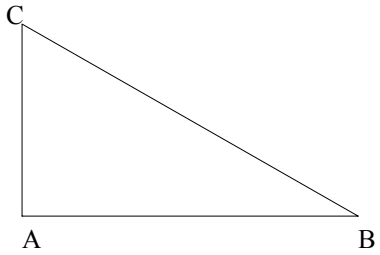


Figure 2.3

La fonction f sera alors remplacée sur chacun de ces triangles par une fonction affine :

$$f(x,y) \cong \alpha + \beta \cdot x + \gamma \cdot y$$

La fonction f étant connue en trois points A, B, C, les coefficients α, β et γ, s'en déduisent facilement.

Le problème est alors de savoir comment déterminer les morceaux de lignes de niveau qui coupent le triangle.

En programmation linéaire, on montre qu'une fonction affine sur un triangle (de manière plus générale sur un polygone) atteint ses extrema en un sommet.

Connaissant f en A, B, C, on calcule les extrema et on regarde si la valeur constante cherchée (f = Cste) est entre ses bornes.

En général la courbe de niveau coupe le triangle en deux cotés. On aura des ennuis quand elle passera par l'un des sommets (à détailler).

Exemples de courbes :

Ellipse: $a \cdot x^2 + b \cdot y^2 + \alpha \cdot x + \beta \cdot y + \gamma \ (a > 0, b > 0, \gamma - \frac{1}{4} \cdot \left(\frac{\alpha^2}{a} + \frac{\beta^2}{b} \right) < 0)$

Hyperbole: $a \cdot x^2 - b \cdot y^2 + \alpha \cdot x + \beta \cdot y + \gamma \ (a > 0, b > 0, \gamma - \frac{1}{4} \cdot \left(\frac{\alpha^2}{a} + \frac{\beta^2}{b} \right) < 0)$

Parabole: $a \cdot x^2 + \alpha \cdot x + \beta \cdot y + \gamma \ (a > 0)$

? : $x + y - \mathrm{Ln}(x) - \mathrm{Ln}(y) + C$

CHAPITRE 3

Analyse numérique linéaire

1. Introduction

1.1 Position des problèmes

Dans ce chapitre, on étudie les trois problèmes suivants :

- la résolution d'un système linéaire A.x = b ;
- l'inversion d'une matrice A ;
- la recherche des valeurs propres et des vecteurs propres d'une matrice A.

1.2 Notations

Une matrice sera notée $A = ((a_{ij}))_{1 \le i,j \le n}$, où :
i est le numéro de la ligne ;
j est le numéro de la colonne.
Un vecteur sera noté $v = (v_i)_{1 \le i \le n}$.

1.3 Remarques

(i) Si A et b sont à coefficients complexes, en écrivant : $A = C + i.D$, $b = \alpha + i \cdot \beta$, $x = \xi + i \cdot \eta$, avec C, D, α, β, ξ, η à coefficients réels, la résolution du système de n équations à n inconnues $A \cdot x = b$ dans C^n se ramène à un système de $2 \cdot n$ équations à $2 \cdot n$ inconnues, à coefficients réels :

$$\begin{pmatrix} C & -D \\ D & C \end{pmatrix} \cdot \begin{pmatrix} \xi \\ \eta \end{pmatrix} = \begin{pmatrix} \alpha \\ \beta \end{pmatrix}$$

Il nous suffit donc de considérer des systèmes à coefficients réels.
(ii) Si A est inversible, les coefficients de A^{-1} sont les n solutions des n systèmes linéaires :

$$A \cdot x = e_j \quad (j = 1 , 2 , .. , n) ;$$

où $\{e_1, ..., e_n\}$ désigne la base canonique de R^n.

C'est-à-dire que le calcul de l'inverse d'une matrice se ramène à la résolution simultanée de n systèmes linéaires de même matrice.

1.4 Problèmes numériques liés à la résolution des systèmes linéaires

La précision numérique des ordinateurs n'étant pas infinie, un algorithme de résolution numérique d'un système linéaire ne donnera pas, en général, une solution exacte mais seulement une approximation de cette solution.

D'autre part il sera important de savoir évaluer le temps de calcul nécessaire à un tel algorithme .

Les erreurs numériques sont de trois types :

- les erreurs sur les données (pour des données expérimentales) ;
- les erreurs d'arrondis (calculs en virgule flottante) ;
- les erreurs de troncature.

Les erreurs d'arrondis vont s'accumuler au cours des calculs, il est donc important de connaître le nombre d'opérations élémentaires que nécessite un tel algorithme.

1.5 Systèmes dégénérés et numériquement dégénérés

Un système $A \cdot x = b$ est dit (théoriquement) dégénéré si $\text{Dét}(A) = 0$. Mais vérifier cette condition de manière exacte nécessite une précision infinie, ce qui n'est pas le cas, il faut donc définir un concept de dégénérescence numérique d'un système linéaire.

Exemple 1 — Considérons le système :

$$\begin{cases} x + y = 2 \\ (1 + a) \cdot x + y = 2 + a \end{cases}$$

où a est « très petit ».

On a $\text{Dét}(A) = -a$, qui est supposé non nul. Mais si $a = 10^{-12}$, pour un ordinateur qui travaille avec 11 chiffres significatifs la représentation machine de A et b sera :

$$A_{mach} = \begin{pmatrix} 1.00000000000 & 1.00000000000 \\ 1.00000000000 & 1.00000000000 \end{pmatrix} \text{ et } b_{mach} = \begin{pmatrix} 2.00000000000 \\ 2.00000000000 \end{pmatrix}$$

et le système apparaîtra comme dégénéré.

Exemple 2 — Si on considère le système dégénéré :

$$\begin{cases} 3 \cdot x + 5 \cdot y = 4 \\ 3 \cdot x + 5 \cdot y = 4 \end{cases}$$

l'utilisation de la méthode de Gauss (voir paragraphe 3.4) donne $x = 0.50000000000$ et $y = 0.50000000000$ comme solutions. Ce qui n'est pas faux, mais ce n'est pas la seule solution, (il y en a une infinité) ce que ne sait pas détecter l'ordinateur.

Un système sera dit numériquement dégénéré, si la valeur de son déterminant calculé en machine est non significative (i.e. trop petite ou trop grande, c'est-à-dire en dehors des limites de la machine).

Exemple 3 — La matrice de Hilbert d'ordre n est définie par $H = ((h_{ij}))_{1\le i,j\le n}$ où $h_{ij} = \dfrac{1}{i+j-1}$ pour $1 \le i, j \le n$. C'est une matrice symétrique définie positive, donc non dégénérée. En fait, on a :

$$\text{Dét}(H) = \frac{\Phi^4(n-1)}{\Phi(2\cdot n-1)} \text{ , où } \Phi(n) = \prod_{k=1}^{n} k!$$

Pour n grand, on aura $\text{Dét}(A)_{mach} = 0$. Un tel système est donc numériquement dégénéré, mais non dégénéré.

Pour tester la performance d'un algorithme de résolution d'un système linéaire, on pourra l'essayer avec une matrice de Hilbert d'ordre élevé.

Des exemples de calculs de déterminants de matrices de Hilbert, en utilisant, d'une part la décomposition L-R (Cf. paragraphe (3.6)) et d'autre part la formule de Hilbert sont donnés dans le tableau 3.1.

Les résultats sont obtenus en utilisant les réels de base, de type FLOAT, de OpenAda.

Pour la formule de Hilbert, une méthode de calcul est basée sur les récurrences :

$$\Phi(n) = n!\cdot\Phi(n-1)$$

$$\Delta_{n+1} = \frac{\Delta_n}{(2\cdot n + 1)\cdot\left(C_{2n}^{n}\right)^2}$$

où on a noté Δ_n le déterminant de la matrice de Hilbert d'ordre n.

Si $u_n = C_{2n}^{n}$, on a $u_{n+1} = \dfrac{2\cdot(2\cdot n + 1)}{n + 1}\cdot u_n$ ce qui donne en définitive l'algorithme de calcul suivant :

D = 1 ;
u = 2 ;
Pour n allant de 1 à nMax faire
Début

$$\Delta = \frac{\Delta}{(2\cdot n + 1)\cdot u^2} \text{ ;}$$

$$u = \frac{2\cdot(2\cdot n + 1)}{n + 1}\cdot u \text{ ;}$$

Fin ;

Pour de grandes valeurs de n la matrice n'apparaît pas numériquement dégénérée mais les résultats ne sont pas fiables du fait des accumulations d'erreurs

d'arrondis, aussi bien pour le déterminant calculé en utilisant la décomposition L·R que pour le calcul utilisant la formule de Hilbert.

n	Déterminant Formule	Déterminant calculé avec L·R
1	1.00000000000000E+00	1.00000000000000E+00
2	8.33333333333333E–02	8.33333333333333E–02
3	4.62962962962963E–04	4.62962962962961E–04
4	1.65343915343915E–07	1.65343915343930E–07
5	3.74929513251509E–12	3.74929513251290E–12
6	5.36729988735869E–18	5.36729988682088E–18
7	4.83580262392612E–25	4.83580261566227E–25
8	2.73705011379151E–33	2.73705023258196E–33
9	9.72023431192500E–43	9.72026335654786E–43
10	2.16417922643149E–53	2.16436997064480E–53
11	3.01909533444935E–65	3.02449714771783E–65
12	2.63778065125355E–78	2.70874882840569E–78
13	1.44289651879114E–92	Pivot trop petit dans LR
14	4.94031491459083E–108	Pivot trop petit dans LR
15	1.05854274306972E–124	Pivot trop petit dans LR
16	1.41913921144317E–142	Pivot trop petit dans LR
17	1.19027124270300E–161	Pivot trop petit dans LR
18	6.24486061412311E–182	Pivot trop petit dans LR
19	2.04934373331812E–203	Pivot trop petit dans LR
20	4.20617895662473E–226	Pivot trop petit dans LR
21	5.39898624190004E–250	Pivot trop petit dans LR

Tableau 3.1

1.6 Problèmes de stabilité numérique

Dans la pratique, les coefficients de la matrice et du second membre ne sont connus que de façon approximative. Et un problème important est donc de savoir si de petites variations sur ces coefficients peuvent entraîner de grosses variations sur la solution. La réponse est hélas oui comme le montre l'exemple du système $A \cdot x = b$, avec :

$$A = \begin{pmatrix} 10 & 7 & 8 & 7 \\ 7 & 5 & 6 & 5 \\ 8 & 6 & 10 & 9 \\ 7 & 5 & 9 & 10 \end{pmatrix} \text{ et } b = \begin{pmatrix} 32 \\ 23 \\ 33 \\ 31 \end{pmatrix}$$

dont la solution est $x = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}$. Si on modifie le second membre en $b = \begin{pmatrix} 32.1 \\ 22.9 \\ 33.1 \\ 30.9 \end{pmatrix}$, c'est-à-dire que b est donné avec une erreur relative de 0.3%, alors la solution devient $x = \begin{pmatrix} 9.2 \\ -12.6 \\ 4.5 \\ -1.1 \end{pmatrix}$. On a donc une erreur relative sur x de l'ordre de 1000%.

De même si on perturbe la matrice en prenant $A = \begin{pmatrix} 10 & 7 & 8.1 & 7.2 \\ 7.08 & 5.04 & 6 & 5 \\ 8 & 5.98 & 9.81 & 9 \\ 6.99 & 4.99 & 9 & 9.98 \end{pmatrix}$ et en gardant le second membre initial, la solution devient $x = \begin{pmatrix} -8.1 \\ 137 \\ -34 \\ 22 \end{pmatrix}$ Un tel système est dit mal conditionné.

Le conditionnement d'une matrice n'est pas lié à un déterminant voisin de 0. Pour l'exemple ci-dessus, la matrice est symétrique définie positive de déterminant égal à 1.

Dans le paragraphe 2.2 on définira une quantité permettant de mesurer le bon ou mauvais conditionnement d'un système.

1.7 Méthodes de résolution des systèmes linéaires

On a deux catégories de méthodes de résolution d'un système linéaire :

- les méthodes directes ;
- les méthodes itératives.

Si on suppose une précision infinie, les méthodes directes conduisent à la solution exacte du système en un nombre fini d'étapes.

Alors que les méthodes itératives donnent toujours une approximation de la solution, la solution exacte étant obtenue en un nombre infini d'étapes.

Par exemple, la méthode de Cramer est une méthode directe de résolution des systèmes linéaires (mais impraticable).

Comme il n'existe pas d'algorithme direct de résolution d'une équation polynomiale, les méthodes de recherche de valeurs propres seront nécessairement des méthodes itératives.

Les méthodes directes sont plutôt adaptées au « petits systèmes » ($n < 50$) et les méthodes itératives aux « grands systèmes ».

Un aspect intéressant de la matrice A est le nombre de termes a_{ij} qui sont nuls. Une matrice avec beaucoup de zéros sera dite creuse. Les méthodes itératives

seront bien adaptées aux matrices creuses et en particulier aux matrices bandes (i.e. telles que $a_{ij} = 0$ pour $|i - j| > p$) car cette structure bande sera conservée.

Enfin, les problèmes avec des matrices symétriques seront plus faciles à traiter, en particulier les problèmes de valeurs propres.

1.8 Exemples d'applications

Dans ce cours, après avoir exposé quelques méthodes d'analyse numérique linéaire, nous les utiliseront pour la résolution numérique de certaines équations différentielles (méthode des différences finies et méthode des éléments finis pour des problèmes aux limites en dimension un) ou aux dérivées partielles (méthode

des différences finies et méthode des éléments finis pour des problèmes aux limites en dimension 2) et pour résoudre des problèmes d'interpolation et d'approximation.

En utilisant la méthode de Newton-Raphson, la résolution des systèmes non linéaires se ramène à des résolutions de systèmes linéaires. La méthode de Newton-Raphson sera utilisée avec la méthode du tir qui permet de résoudre des systèmes d'équations différentielles avec conditions aux limites.

2. Rappels et compléments sur le calcul matriciel

Dans tout ce paragraphe, on désigne par $A = ((a_{ij}))_{1 \le i,j \le n}$ une matrice dans
$M_n(R) = \{$ Matrices à n lignes et n colonnes à coefficients réels $\}$.

2.1 Normes vectorielles et matricielles

2.1.1 Norme matricielle induite par une norme vectorielle

Définition : Si $x \mapsto \|x\|$ est une norme sur R^n, en posant, pour toute matrice $A \in M_n(R)$:

$$\|A\| = \mathrm{Sup}\left\{ \frac{\|A \cdot x\|}{\|x\|} \; ; \; x \in R^n - \{0\} \right\}$$

on définit une norme sur $M_n(R)$ appelée norme matricielle induite par $\|\cdot\|$.

Remarque 1 — Avec la linéarité de A, on déduit qu'on a aussi :

$$\|A\| = \mathrm{Sup}\{ \|A \cdot x\|; \; x \in R^n \text{ et } \|x\| = 1 \}$$

Remarque 2 — De la définition, il résulte que :

$$\forall\, x \in R^n \,,\ \forall\, A \in M_n(R) \,,\ \|A \cdot x\| \le \|A\| \cdot \|x\|$$

$$\forall\, A,\, B \in M_n(R) \,,\ \|A \cdot B\| \le \|A\| \cdot \|B\|$$

$$\text{Si } I_d \text{ est la matrice identité, alors } \|I_d\| = 1$$

Remarque 3 — Il existe des normes matricielles qui ne sont pas induites par une norme vectorielle. Un exemple classique est donné par la « norme de Schur » définie par :

$$N_s(A) = \left(\mathrm{Trace}\left({}^t A \cdot A\right)\right)^{\frac{1}{2}} = \left(\sum_{i,\,j\,=\,1}^{n} a_{ij}^2 \right)^{\frac{1}{2}}$$

C'est simplement la norme euclidienne de A considérée comme vecteur de R^{n^2}.
Comme $N_s(I_d) = \sqrt{n}$ cette norme n'est pas induite par une norme vectorielle.

2.1.2 Norme utilisée dans ce livre

Nous prendrons, sauf indication contraire, la norme vectorielle définie par :

$$\|x\| = \mathrm{Max}\{ |x_i| \; ; \; 1 \le i \le n \}$$

On a alors le résultat suivant qui nous permet de calculer effectivement la norme matricielle induite :

Théorème : Pour toute matrice A dans M (R), on a :

$$\|A\| = \text{Max}\left\{ \sum_{j=1}^{n} |a_{ij}| \; ; \; 1 \le i \le n \right\}$$

pour la norme matricielle induite par $\|\cdot\|$ définie ci-dessus.

Démonstration — Posons $\alpha = \text{Max}\left\{ \sum_{j=1}^{n} |a_{ij}| \; ; \; 1 \le i \le n \right\}$.

(a) Pour x dans R^n tel que $\|x\| = 1$, on a :

$$\|A \cdot x\| = \text{Max}\left\{ \sum_{j=1}^{n} |a_{ij} \cdot x_j| \; ; \; 1 \le i \le n \right\} \le \alpha$$

(b) Pour montrer que $\alpha \le \|A\|$, il suffit de trouver un vecteur x tel que $\|A \cdot x\| = \alpha$.

Soit k tel que $\alpha = \sum_{j=1}^{n} |a_{kj}|$. On pose, pour j = 1, ..., n

$$x_j = \begin{cases} \text{Signe}(a_{kj}) \text{ si } a_{kj} \neq 0 \\ 0 \text{ si } a_{kj} = 0 \end{cases}$$

Et on a alors $\|A \cdot x\| = \alpha$.

2.2 Conditionnement d'une matrice

Soit A une matrice inversible et x solution du système A·x = b. En notant x + δx la solution du système perturbé A·y = b + δb, on a :

$$\frac{\|\delta x\|}{\|x\|} \le \|A\| \cdot \|A^{-1}\| \cdot \frac{\|\delta b\|}{\|b\|}$$

de même si x + δx est solution de (A + δA)·y = b, on a :

$$\frac{\|\delta x\|}{\|x + \delta x\|} \le \|A\| \cdot \|A^{-1}\| \cdot \frac{\|\delta A\|}{\|A\|}$$

ce qui nous amène à poser la définition suivante :

Définition : Si A est une matrice inversible et $\|\cdot\|$ une norme matricielle induite par une norme vectorielle, alors le conditionnement de A relativement à cette norme est la quantité :

$$\text{Cond}(A) = \frac{1}{\|A\| \cdot \|A^{-1}\|}$$

Le conditionnement est donc un réel compris entre 0 et 1.

Un système sera bien conditionné si Cond(A) est voisin de A. On a Cond(A) = 1 pour les matrices unitaires.

Pour l'exemple de matrice mal conditionnée du paragraphe 1.6, on a Cond(A) ≅ 1/3000.

Pour plus de détails sur ces problèmes de conditionnement, on pourra consulter Théodor et Lascaux, Chap. 3 (p. 167 à 206).

2.3 Valeurs propres et rayon spectral d'une matrice

2.3.1 Définitions

On dit que μ dans C est *valeur propre* de A s'il existe un vecteur non nul x dans C^n tel que $A \cdot x = \mu \cdot x$. On dit alors que x est un *vecteur propre* associé à A. Ce qui équivaut à dire que $A - \mu \cdot I_d$ est non inversible, soit que le *polynôme caractéristique* de A, $P(t) = Dét(A - t \cdot I_d)$, est nul pour $t = \mu$.

On dit que A est *diagonalisable*, s'il existe une base de R^n formée de vecteurs propres de A.

L'ensemble des valeurs propres de A est appelé le *spectre de A* et noté *Sp(A)*. C'est une partie finie de C ayant au plus n éléments.

Le *rayon spectral* de A est alors défini par :

$$\rho(A) = \mathrm{Max}\{\, |\mu| \,;\, \mu \in \mathrm{Sp}(A) \,\}$$

2.3.2 Propriétés du rayon spectral

Théorème : Si A est dans $M_n(R)$, on a :
$\rho(A) = \mathrm{Inf}\{\|A\| \,;\, \|\cdot\| \text{ est une norme matricielle induite par une norme vectorielle}\}$

Démonstration — (a) Pour toute valeur propre μ de vecteur propre associé x et pour toute norme matricielle induite par une norme vectorielle, on a : $\|A \cdot x\| \le \|A\| \cdot \|x\|$, avec $\|A \cdot x\| = \mu \cdot \|x\|$, on en déduit donc que $|\mu| \le \|A\|$, pour tout μ dans Sp(A), donc $\rho(A) \le \|A\|$.

(b) On peut d'autre part montrer que pour tout $\tau > 0$, il existe une norme matricielle induite, telle que $\|A\| \le \rho(A) + \tau$ (Voir Ciarlet p.19). Ce qui donne le résultat.

Corollaire — Soit A dans $M_n(R)$, les trois conditions suivantes sont équivalentes :

(i) $\lim_{k \to +\infty} (A^k) = 0$;

(ii) pour toute valeur initiale x_0, la suite définie par $x_{k+1} = A \cdot x_k$, pour $k \ge 0$, converge vers le vecteur nul ;

(iii) $\rho(A) < 1$.

Démonstration — (i) ⇒ (ii). Résulte de : $\|x_k\| = \|A^k \cdot x_0\| \le \|A^k\| \cdot \|x_0\|$.

(ii) $\Rightarrow$ (iii). Supposons qu'il existe une valeur propre μ telle que $|\mu| \geq 1$. Alors, si x_0 est un vecteur propre associé à μ, en écrivant que $x_k = A^k \cdot x_0 = \mu^k \cdot x_0$, on déduit que la suite (x_k) ne converge pas vers 0.
(iii) $\Rightarrow$ (i). Soit $\tau > 0$ tel que $\rho(A) + \tau < 1$ et $\|\cdot\|$ une norme matricielle induite telle que $\|A\| < \rho(A) + \tau$. Avec $\|A^k\| \leq \|A\|^k$, on déduit alors que $\lim_{k\to+\infty}(A^k) = 0$.

Remarque — On peut aussi montrer que, pour toute norme matricielle :

$$\rho(A) = \lim_{k\to+\infty}\left(\|A^k\|^{\frac{1}{k}}\right).$$

(Voir Ciarlet p. 22).

2.4 Quelques propriétés des matrices symétriques réelles

Le *produit scalaire euclidien* de R^n est noté :

$$(x, y) \mapsto \langle x | y \rangle = \sum_{i=1}^{n} x_i \cdot y_i$$

La norme associée est la *norme euclidienne* et notée $x \mapsto \|x\|_2$.

2.4.1 Transposée d'une matrice

La *transposée* de la matrice $A = ((a_{ij}))_{1\leq i,j\leq n}$, est la matrice : ${}^tA = ((a_{ji}))_{1\leq i,j\leq n}$
Elle est aussi définie par :

$$\forall x, y \in R^n : \langle x | {}^tA \cdot y \rangle = \langle A \cdot x | y \rangle$$

La matrice A est dite *symétrique*, si ${}^tA = A$. On dit qu'elle est symétrique *définie positive* si on a, de plus : $\forall x \in R^n - \{0\} : \langle A \cdot x | y \rangle > 0$.

2.4.2 Spectre des matrices symétriques

Pour ce qui est des valeurs propres d'une matrice symétrique réelle, la situation est plus simple que dans le cas général, ce que nous résumons par le :

Théorème : (1) Les valeurs propres d'une matrice symétrique réelle sont toutes réelles.
(2) Si, de plus, la matrice est définie positive, alors toutes les valeurs propres sont strictement positives.
(3) Une matrice symétrique se diagonalise dans une base orthonormale. Ce qui se traduit en disant qu'il existe une matrice orthogonale O (i.e. telle que $O^{-1} = {}^tO$) telle que $A = {}^tO \cdot D \cdot O$, où $D = \mathrm{Diag}(\mu_1, \ldots, \mu_n)$, les μ_i étant les valeurs propres de A.

2.5 Matrices à diagonale strictement dominante

Définition — Une matrice A est à diagonale strictement dominante, si :

$$|a_{ii}| > \sum_{j\neq i} |a_{ij}| \quad (i = 1, \ldots, n)$$

Remarque — Les matrices à diagonale strictement dominante se rencontrent dans de nombreux problèmes, par exemple dans le problème de l'interpolation par des fonctions splines cubiques ou dans les problèmes de résolutions d'équations aux dérivées partielles par des méthodes de discrétisation par différences finies.

Théorème : Une matrice à diagonale strictement dominante est inversible.

Démonstration — Montrons que toutes les valeurs propres de A sont non nulles.

Soit μ une valeur propre de A et x un vecteur propre associé on a alors :

$$\forall i \in \{1,..,n\} \quad (\mu - a_{ii}) \cdot x_i = \sum_{j \neq i} a_{ij} \cdot x_j$$

Et en prenant un indice i tel que $|x_i| = \|x\|$, on déduit que :

$$|\mu - a_{ii}| \leq \sum_{j \neq i} |a_{ij}|.$$

On ne peut donc avoir $\mu = 0$ si A est à diagonale strictement dominante. Donc A est inversible.

Théorème : Soit A une matrice symétrique réelle à diagonale strictement dominante. Alors, A est définie positive si, et seulement si $a_{ii} > 0$, pour tout $i = 1 , ... , n$.

Démonstration — Supposons que $a_{ii} > 0$ pour tout i. En reprenant la démonstration du théorème précédent, on a :

$$|\mu - a_{ii}| \leq \sum_{j \neq i} |a_{ij}| < a_{ii} \text{ , pour tout indice i}$$

On en déduit donc que $\mu > 0$. La matrice A a donc toutes ses valeurs propres strictement positives, ce qui équivaut à dire qu'elle est définie positive.

La réciproque provient de l'égalité $a_{ii} = \langle A \cdot e_i | e_i \rangle$, où e_1, ..., e_n est la base canonique de R^n

3. Résolution numérique des systèmes linéaires. Inversion des matrices

3.1 Position des problèmes. Notations

On se donne une matrice A dans $M_n(R)$, supposée inversible, un vecteur b dans R^n et on veut résoudre le système linéaire :

$$A \cdot x = b \quad (1)$$

L'inverse de A sera obtenue en prenant pour valeurs successives de b les éléments de la base canonique de R^n.

Remarque — Une matrice sera notée $A = ((a_{ij}))_{1 \leq i,j \leq n}$, où l'indice i est le numéro de la ligne et j celui de la colonne. Mais il faut être conscient du fait que la

machine va stocker un tel objet sur un seul tableau, soit par colonnes (c'est le cas en Fortran) :

$$a_{11}, a_{21},..., a_{n1}, a_{12},..., a_{n2},..., a_{1n}, ..., a_{nn}$$

soit par lignes (c'est le cas en Ada, C et Pascal) :

$$a_{11}, a_{12},..., a_{1n}, a_{21},..., a_{2n},..., a_{n1}, ..., a_{nn}$$

Le stockage en colonnes peut être exploité en programmation alors que cela est moins pratique pour le stockage en lignes.

3.2 Une méthode impraticable : la méthode de Cramer

Si on note A_j, la matrice déduite de A en remplaçant la colonne N°j par b, pour j = 1, 2, ..., n, la solution du système (1) est donnée par les formules de Cramer :

$$x_j = \frac{\text{Dét}(A_j)}{\text{Dét}(A)} \quad (j = 1, 2, ..., n)$$

Si on calcule un déterminant par la formule :

$$\text{Dét}(A) = \sum \text{Sgn}(\sigma) \cdot \prod a_{j,\sigma(j)}$$

où σ décrit l'ensemble des permutations de {1, 2, ..., n} et Sgn(σ) est la signature de σ, cela nécessite :

$$n!\cdot(n-1) \text{ multiplications et } (n!-1) \text{ additions,}$$

donc, environ $n\cdot n!$ opérations élémentaires.

Comme il y a n déterminants à calculer puis n divisions à faire, on aura un total, environ, de $n^2\cdot n!$ opérations élémentaires.

En considérant qu'une opération élémentaire demande à un micro-ordinateur une microseconde, pour un système de 20 équations à 20 inconnues, la méthode de Cramer demandera 10^{15} secondes soit plus de 10^7 années, c'est-à-dire plus de 10 millions d'années !!!

Cette méthode n'a donc qu'un intérêt théorique, ce qui est loin d'être négligeable.

3.3 Cas des systèmes triangulaires

On suppose, ici, que A est une matrice triangulaire supérieure.

3.3.1 Résolution des systèmes triangulaires

Le système (1) s'écrit alors :

$$a_{ii}\cdot x_i + a_{i,i+1}\cdot x_{i+1} + ... + a_{i,n}\cdot x_n = b_n \quad (1 \le i \le n) .$$

La résolution se fait donc « en remontant » :

$$\begin{cases} x_n = \dfrac{b_n}{a_{nn}} \\ x_i = \dfrac{\left(b_i - \displaystyle\sum_{j=i+1}^{n} a_{ij}\cdot x_j\right)}{a_{ii}} \quad (i = n-1, ..., 1) \end{cases}$$

Ce qui donne la programmation structurée :

```
PROCEDURE SystTriSup(Entrée n : Entier ; A : Matrice ; b : Vecteur ; Sortie x : Vecteur) ;
Début
   Pour i allant de n à 1 faire
   Début
      S = 0 ;
      Pour j allant de i + 1 à n faire S = S + a_ij x_j ;
      Si (a_ii = 0) Alors Stop('Dét(A) = 0) ;
      x_i = (b_i – S)/a_ii ;
   Fin ;
Fin ;
```

Pour un système triangulaire inférieur, on procède de même « en descendant ».

3.3.2 Inversion des matrices triangulaires

Les colonnes de A^{-1} sont les solutions de $A \cdot x = e_i$, où $\{e_1, ..., e_n\}$ est la base canonique de R^n. Ce qui donne, si A est triangulaire inférieure, pour les coefficients de $U = A^{-1}$:

$$\begin{cases} u_{ii} = \dfrac{1}{a_{ii}} \\ u_{ij} = 0 \quad (1 \le i < j \le n) \\ u_{ki} = -\dfrac{\sum_{j=i}^{k-1} a_{kj} \cdot u_{ji}}{a_{kk}} \quad (1 \le i < k \le n) \end{cases}$$

Si A est triangulaire supérieure, ${}^tA$ est triangulaire inférieure et il suffit d'écrire :

$$A^{-1} = {}^t({}^tA^{-1}).$$

Le déterminant d'une telle matrice se calculant facilement par :

$$\text{Dét}(A) = \prod_{i=1}^{n} a_{ii}$$

On a donc la programmation structurée :

```
PROCEDURE InvTriInf(Entrée n : Entier ; A : Matrice ; Sortie U : Matrice ) ;
Début
   Pour i allant de 1 à n faire
   Début
      Si (a_ii = 0) Alors Stop('Matrice non inversible') ;
      u_ii  = 1/a_ii ;
      Pour j allant de i + 1 à n faire u_ij = 0 ;
   Fin ;
   Pour i allant de 1 à n - 1 faire
   Début
      Pour k allant de i + 1 à n faire
      Début
         S = 0 ;
         Pour j allant de i à k - 1 faire S = S + a_kj·u_ji ;
         u_ki = –S/a_kk ;
      Fin ;
```

Fin ;
Fin ;

3.4 Méthode des pivots de Gauss

3.4.1 Principe de la méthode

En faisant des opérations élémentaires sur le système (1) (i.e. permuter deux lignes ou ajouter à une ligne un multiple d'une autre, c'est-à-dire des opérations qui ne vont pas changer l'ensemble des solutions de (1)), on transforme ce système en un système triangulaire supérieur : R·x = c. De plus comme une permutation de ligne change le déterminant de signe et que le fait d'ajouter un multiple d'une ligne ne change pas ce dernier, on aura

$$\text{Dét}(A) = \text{Dét}(R) = \pm\prod_{i=1}^{n} r_{ii}$$

L'intérêt de cette méthode est surtout pédagogique, dans la pratique on lui préférera la méthode L·R (Cf. paragraphe (3.6)) quand cette dernière peut s'appliquer, ou la méthode de Gaus-Jordan (Cf. paragraphe (3.5)).

3.4.2 Description de la méthode

On note L_i la ligne N° i d'un système linéaire.

Etape 0 — On se ramène à un système tel que a_{11} non nul.

Si pour tout i = 1, 2, ..., n, on a $a_{i1} = 0$, alors Dét(A) = 0 et c'est fini. Sinon, il existe i > 1 tel que a_{i1} non nul, et en permutant les lignes 1 et i (si i = 1, on ne fait rien), on se ramène à un système $A^{(1)} \cdot x = b^{(1)}$, avec $a_{11}^{(1)}$ non nul.

Le coefficient $a_{11}^{(1)}$ est le premier pivot.

On a alors $\text{Dét}(A) = \pm\text{Dét}\left(A^{(1)}\right)$, avec le signe moins si, et seulement si il y a une permutation des lignes 1 et i avec i > 1.

Etape 1 — Elimination de x_1 dans les équations 2, ..., n.

On fait pour cela :

$$L_i^{(1)} - \frac{a_{i1}^{(1)}}{a_{11}^{(1)}} \cdot L_1^{(1)} \quad (i = 2, 3, \ldots, n)$$

et, après une éventuelle permutation des lignes 2 et i > 3, on a le système :

$$A^{(2)} \cdot x = b^{(2)}, \text{ avec } a_{22}^{(2)} \text{ non nul.}$$

où

$$A^{(2)} = \begin{pmatrix} a_{11}^{(1)} & a_{12}^{(1)} & . & . & a_{1n}^{(1)} \\ 0 & a_{22}^{(2)} & . & . & a_{2n}^{(2)} \\ . & . & . & . & . \\ . & . & . & . & . \\ 0 & a_{n2}^{(2)} & . & . & a_{nn}^{(2)} \end{pmatrix}$$

Le coefficient $a_{22}^{(2)}$ est le deuxième pivot.

Etape k — Elimination de x_k dans les équations k + 1, ..., n.

A la fin de l'étape k – 1, on a obtenu le système :

$$A^{(k)} \cdot x = b^{(k)}, \text{ avec } a_{kk}^{(k)} \text{ non nul.}$$

et

$$A^{(k)} = \begin{pmatrix} a_{11}^{(1)} & a_{12}^{(1)} & . & . & . & a_{1n}^{(1)} \\ 0 & a_{22}^{(2)} & . & . & . & a_{2n}^{(2)} \\ . & . & . & . & . & . \\ 0 & . & 0 & a_{kk}^{(k)} & . & a_{kn}^{(k)} \\ . & . & . & . & . & . \\ 0 & . & 0 & a_{nk}^{(k)} & . & a_{nn}^{(k)} \end{pmatrix}$$

Le coefficient $a_{kk}^{(k)}$ est le pivot N° k.

On fait alors $L_i^{(k)} - \frac{a_{ik}^{(k)}}{a_{kk}^{(k)}} \cdot L_k^{(k)} \quad (i = k+1, \ldots, n)$, puis une éventuelle permutation des lignes k + 1 et j > k + 1 pour se ramener à $a_{k+1,k+1}^{(k+1)}$ non nul.

Au bout de n – 1 étapes, on est donc ramené à un système triangulaire supérieur :

$$A^{(n)} \cdot x = b^{(n)}.$$

De plus, on a Dét(A) = $(-1)^p \cdot$Dét($A^{(n)}$), où p est le nombre de permutations qui ont été nécessaires pour avoir des pivots non nuls et Dét($A^{(n)}$) est le produit des pivots.

3.4.3 Remarque sur le choix des pivots : méthode de Gauss avec pivots partiels

Pour éviter de faire une division par un nombre trop petit, dans le choix du pivot, on aura intérêt, à l'étape k – 1, à permuter la ligne k avec la ligne j > k telle que $|a_{jk}| = \text{Max}\{|a_{ik}|; i = k, \ldots, n\}$, de manière à avoir le pivot le plus grand possible en valeur absolue.

Si ce maximum est trop petit, alors le système est numériquement dégénéré et la méthode ne sera pas fiable.

3.4.4 Nombre d'opérations élémentaires dans la méthode de Gauss

Les opérations $L_i^{(k)} - \frac{a_{ik}^{(k)}}{a_{kk}^{(k)}} \cdot L_k^{(k)} \quad (i = k+1, \ldots, n)$, à l'étape N° k, demandent :

n – k divisions ;
(n – k)·(n – k) multiplications ;
(n – k)·(n – k) additions .

Soit un total de :

$$\sum_{k=1}^{n-1} n - k = \frac{n \cdot (n-1)}{2} \text{ divisions}$$

et

$$2 \cdot \sum_{k=1}^{n-1} (n-k)^2 = \frac{2 \cdot n \cdot (n-1) \cdot (2 \cdot n - 1)}{6} \text{ additions et multiplications}$$

Il nous faut donc $\frac{n \cdot (n - 1) \cdot (4n + 1)}{6} \approx \frac{2 \cdot n^3}{3}$ opérations élémentaires pour aboutir à un système triangulaire supérieur. Puis la résolution de ce dernier système nécessite n divisions et n·(n – 1) additions et multiplications, soit n^2 opérations élémentaires.

En résumé, la résolution d'un système de n équations à n inconnues par la méthode de Gauss va nécessiter un nombre d'opérations qui est un $O(n^3)$.

Par exemple, un système de 100 équations demandera moins d'une seconde.

3.4.5 Programmation structurée

La variable L utilisée dans la procédure PivotMax est le numéro de la ligne où se trouve le pivot le plus grand en valeur absolue, pour tout k = 1, 2, ..., n.

La procédure PermuterLigne, facile à écrire permet de permuter deux lignes du système.

Après exécution de la procédure la matrice A est devenue triangulaire et le vecteur b est transformé, de sorte que les valeurs initiales de A et b sont perdues. C'est-à-dire que les variables A et b sont passées par adresse (c'est le mode Entrée_Sortie).

```
PROCEDURE PivotMax(Entrée k, n : Entier ; Entrée_Sortie A : Matrice ; b : Vecteur) ;
Début
   L = k ;
   Pour i Allant de k + 1 à n Faire
     Si |a_ik| > |a_Lk| Alors L = i ;
   Si |a_Lk| < ε Alors Stop('Pivot trop petit') ;
   Si L ≠ k Alors PermuterLignes(n,k,L,A,b) ;
Fin ;

PROCEDURE Eliminer(Entrée k, n : Entier ; Entrée_Sortie A : Matrice ; b : Vecteur) ;
Début
   Pour i Allant de k + 1 à n Faire
   Début
      m = a_ik / a_kk ;
      Pour j Allant de k + 1 à n Faire a_ij = a_ij – m·a_kj ;
      a_ik = 0 ;
      b_i = b_i – m·b_k ;
   Fin ;
Fin ;

PROCEDURE Pivotage(Entrée n : Entier ; Entrée_ Sortie A : Matrice ; b : Vecteur) ;
Début
      Pour k Allant de 1 à n – 1 Faire
      Début
        PivotMax(k, n, A, b) ;
        Eliminer(k,n,A,b) ;
      Fin ;
Fin ;
```

Ensuite la résolution de $A \cdot x = b$ se fait en appelant la procédure de résolution d'un système triangulaire supérieur.

3.4.6 Remarques

(i) — Dans la programmation structurée ci-dessus, on peut aussi travailler avec une matrice à n lignes et n + 1 colonnes en prenant b pour colonne n + 1.

De manière plus générale, pour résoudre en parallèle p systèmes linéaires de même matrice A et de seconds membres respectifs b_1, ..., b_p, on travaillera avec une matrice à n lignes et n + p colonnes en prenant, pour j = 1, ..., p, b_j comme colonne n + j.

En prenant pour valeurs particulières des seconds membres les vecteurs de la base canonique de R^n, on a ainsi un procédé de calcul de l'inverse d'une matrice. Mais, en fait, pour calculer l'inverse d'une matrice on préfère utiliser la méthode Gauss-Jordan qui est une variante de la méthode de Gauss (Cf. paragraphe (3.5)).

(ii) Cas particulier des systèmes linéaires à coefficients entiers — Si A et b sont à coefficients entiers relatifs, en modifiant la méthode de Gauss, on peut calculer la solution x de façon exacte comme vecteur à coefficients dans Q, en utilisant, au besoin, un procédé de calcul en multiprécision sur les entiers.

On procède comme suit :

Etape 0 — Après une éventuelle permutation de lignes, on a le système, à coefficients dans Z, $A^{(1)} \cdot x = b^{(1)}$, avec un premier pivot non nul.

Etape 1 — Elimination de x_1 dans les équations 2, ..., n.

Si on fait la division de la ligne 1 par le premier pivot, on perdra le caractère entier du système.

On fait plutôt :

garder la ligne 1 ;

$a_{11}^{(1)} \cdot L_i^{(1)} - a_{i1}^{(1)} \cdot L_1^{(1)}$, pour i = 2, ..., n.

Ce qui donne le système, $A^{(2)} \cdot x = b^{(2)}$, avec un deuxième pivot non nul (au prix d'une éventuelle permutation).

Etape 2 — Elimination de x_2 dans les équations 3, ..., n.

Si on procède de la même façon pour cette étape, on constate que les coefficients de la matrice $A^{(3)}$ et du vecteur $b^{(3)}$ sont divisibles par le premier pivot. On peut donc s'autoriser une division par ce premier pivot pour cette étape 2, ce qui aura l'avantage de diminuer les coefficients obtenus.

Etape k — Elimination de x_k dans les équations k + 1, ... n.

Cette étape est décrite par les formules :

$$a_{ij}^{(k+1)} = \frac{\left(a_{kk}^{(k)} \cdot a_{ij}^{(k)} - a_{ik}^{(k)} \cdot a_{kj}^{(k)}\right)}{a_{k-1,k-1}^{(k-1)}}$$

pour i = k + 1, ..., n et j = k + 1, ..., n + 1 (en prenant b comme colonne N° n + 1).

Ce qui donne au bout de n – 1 étapes un système triangulaire supérieur à coefficients entiers qui peut se résoudre de façon exacte dans Q^n.

Calcul du déterminant — Au signe près, on a :

$$\text{Dét}\left(A^{(2)}\right) = \left(a_{11}^{(1)}\right)^{n-1} \cdot \text{Dét}(A)$$

$$\text{Dét}\left(A^{(3)}\right) = \left(\frac{a_{22}^{(2)}}{a_{11}^{(1)}}\right)^{n-2} \cdot \text{Dét}\left(A^{(2)}\right) = \left(a_{22}^{(2)}\right)^{n-2} \cdot a_{11}^{(1)} \cdot \text{Dét}\left(A^{(1)}\right)$$

De proche en proche, on a alors, $\text{Det}(A^{(n)}) = (-1)^p \cdot \left(\prod_{k=1}^{n} a_{kk}^{(k)}\right) \cdot \text{Dét}(A)$, ce qui donne, en tenant compte de l'expression de $\text{Det}(A^{(n)})$ comme produit des pivots, $\text{Dét}(A) = (-1)^p \cdot a_{nn}^{(n)}$.

3.5 Méthode de Gauss-Jordan

3.5.1 Méthode d'élimination de Gauss-Jordan

La méthode de Gauss-Jordan est une variante de la méthode de Gauss, qui consiste à diagonaliser la matrice A, à l'aide de transformations élémentaires (c.a.d. des permutations et des combinaisons linéaires de lignes).

A l'étape k, on ne se contente pas seulement d'éliminer le coefficient de x_k dans les lignes k + 1 à n, mais on l'élimine aussi dans les lignes du dessus soit 1 à k – 1. Ce qui donnera au bout de n étapes un système diagonal.

Description de la k^ième étape du calcul (1 < k < n) — A la k^ième étape, on est au départ dans la situation suivante :

$$\begin{cases} x_1 + a_{1k}^{(k)} \cdot x_k + \ldots\ldots\ldots\ldots\ldots + a_{1n}^{(k)} \cdot x_n = b_1^{(k)} \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ x_{k-1} + a_{k-1,k}^{(k)} \cdot x_k + \ldots\ldots\ldots\ldots\ldots + a_{k-1,n}^{(k)} \cdot x_n = b_{k-1}^{(k)} \\ a_{kk}^{(k)} \cdot x_k + \ldots\ldots\ldots\ldots\ldots + a_{kn}^{(k)} \cdot x_n = b_k^{(k)} \\ a_{k+1,k}^{(k)} \cdot x_k + \ldots\ldots\ldots\ldots\ldots + a_{k+1,n}^{(k)} \cdot x_n = b_{k+1}^{(k)} \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ a_{nk}^{(k)} \cdot x_k + \ldots\ldots\ldots\ldots\ldots + a_{nn}^{(k)} \cdot x_n = b_n^{(k)} \end{cases}$$

On peut supposer que $a_{kk}^{(k)}$ est le pivot maximum, sinon, il suffit de permuter avec une des lignes suivantes.

On commence par tirer x_k de la k^ième équation :

$$x_k = \frac{\left(b_k^{(k)} - a_{k,k+1}^{(k)} \cdot x_{k+1} - \ldots - a_{kn}^{(k)} \cdot x_n\right)}{a_{k,k}^{(k)}}$$

que l'on reporte ensuite dans les autres équations, ce qui donne :

$$\begin{cases} x_1 + a_{1,k+1}^{(k+1)} \cdot x_{k+1} + \ldots\ldots\ldots\ldots\ldots + a_{1n}^{(k+1)} \cdot x_n = b_1^{(k+1)} \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ x_{k-1} + a_{k-1,k+1}^{(k+1)} \cdot x_{k+1} + \ldots\ldots\ldots\ldots\ldots + a_{k-1,n}^{(k+1)} \cdot x_n = b_{k-1}^{(k+1)} \\ x_k + a_{k,k+1}^{(k+1)} \cdot x_{k+1} + \ldots\ldots\ldots\ldots\ldots + a_{kn}^{(k+1)} \cdot x_n = b_k^{(k+1)} \\ a_{k+1,k+1}^{(k+1)} \cdot x_{k+1} + \ldots\ldots\ldots\ldots\ldots + a_{k+1,n}^{(k+1)} \cdot x_n = b_{k+1}^{(k+1)} \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ a_{n,k+1}^{(k+1)} \cdot x_{k+1} + \ldots\ldots\ldots\ldots\ldots + a_{nn}^{(k+1)} \cdot x_n = b_n^{(k+1)} \end{cases}$$

avec, pour j = k + 1, ..., n :

$$a_{kj}^{(k+1)} = \frac{a_{kj}^{(k)}}{a_{kk}^{(k)}} \quad \text{et} \quad b_k^{(k+1)} = \frac{b_k^{(k)}}{a_{kk}^{(k)}}$$

et, pour i dans {1, ..., n} – {k}, j dans {k+1,...,n} :

$$a_{ij}^{(k+1)} = a_{ij}^{(k)} - a_{ik}^{(k)} \cdot a_{kj}^{(k+1)} \quad \text{et} \quad b_i^{(k+1)} = b_i^{(k)} - a_{ik}^{(k)} \cdot b_k^{(k+1)}.$$

On obtient alors directement la solution, après la $n^{ème}$ étape :

$$\begin{cases} x_1 = b_1^{(n)} \\ . \\ . \\ x_n = b_n^{(n)} \end{cases}$$

Pour la recherche du pivot maximum, on utilisera la procédure PivotMax du paragraphe (3.4.5).

Ce qui donne la programmation structurée suivante, le vecteur b contenant en fin d'opération la solution du système.

```
PROCEDURE GaussJordan(Entrée n : Entier ;  Entrée_Sortie A : Matrice ; b :Vecteur) ;

PROCEDURE EliminerGaussJordan(Entrée k, n : Entier ;
                              Entrée_Sortie A : Matrice ; b : Vecteur) ;
Début
  Pour j Allant de k + 1 à n Faire a_kj = a_kj / a_kk ;
  b_k = b_k / a_kk ; a_kk = 1 ;
  Pour i Allant de 1 à k – 1 Faire
  Début
    Pour j Allant de k + 1 à n Faire a_ij = a_ij – a_ik*a_kj ;
    b_i = b_i – a_ik*b_k ;
```

```
      aik = 0 ;
   Fin ;
   Pour i Allant de k + 1 à n Faire
   Début
      Pour j Allant de k + 1 à n Faire aij = aij – aik*akj ;
      bi = bi – aik*bk ;
      aik = 0 ;
   Fin ;
Fin ;

Début
   Pour k Allant de 1 à n Faire
   Début
      PivotMax(k,n,A,b) ;
      EliminerGaussJordan(k,n,A,b) ;
   Fin ;
Fin ;
```

3.5.2 Inversion et calcul du déterminant d'une matrice

On adaptera ce qui précède en remplaçant b par la matrice identité, ce qui revient à résoudre en parallèle les n systèmes linéaires de même matrice A et de seconds membres respectifs e_1, ..., e_n formant la base canonique de R^n.

On obtient alors la programmation structurée suivante :

```
PROCEDURE InversionGaussJordan(Entrée n : Entier ; Entrée_Sortie A, Inv : Matrice) ;

PROCEDURE InitInverse(Entrée : n : Entier ; Sortie Inv : Matrice) ;
Début
   Pour i Allant de 1 à n Faire
   Début
      Pour j Allant de 1 à n Faire
      Début
         Si j ≠ i
         Alors Invij  = 0
         Sinon Invij = 1
      Fin ;
    Fin ;
Fin ;

PROCEDURE PivotMaxInv(Entrées k, n : Entiers ;  Entrée_Sortie  A, Inv : Matrice) ;
Début
   L = k ;
   Pour i Allant de k + 1 à n Faire
    Si |aik| > |aLk|
    Alors L = i ;
   Si |aLk| < ε
   Alors Arréter('Pivot trop petit') ;
   Si L ≠ k
   Alors Début
          Pour j Allant de k à n Echanger(akj,aLj) ;
          Pour j Allant de 1 à n Echanger(Invkj,InvLj) ;
       Fin ;
Fin ;
```

```
PROCEDURE EliminerInvGaussJordan(Entrées k, n : Entiers ; Entrée_Sortie A, Inv : Matrice) ;
Début
   Pour j Allant de k + 1 à n Faire a_kj = a_kj/a_kk ;
   Pour j Allant de 1 à n Faire Inv_kj = Inv_kj/a_kk ;
   a_kk = 1 ;
   Pour i Allant de 1 à k – 1 Faire
   Début
      Pour j Allant de k + 1 à n Faire a_ij = a_ij – a_ik*a_kj ;
      Pour j Allant de 1 à n Faire Inv_ij = Inv_ij – a_ik*Inv_kj ;
      a_ik = 0 ;
   Fin
   Pour i Allant de k + 1 à n Faire
   Début
      Pour j Allant de k + 1 à n Faire a_ij = a_ij – a_ik*a_kj ;
      Pour j Allant de 1 à n Faire Inv_ij = Inv_ij – a_ik*Inv_kj ;
      a_ik = 0 ;
   Fin
Fin ;
Début
   InitInverse(n,Inv) ;
   Pour k Allant de 1 à n Faire
   Début
      PivotMaxInv(k,n,A,Inv) ;
      EliminerInvGaussJordan(k,n,A,Inv) ;
   Fin ;
Fin ;
```

La matrice A étant passée par adresse, on a perdu sa valeur initiale en fin d'opération.

3.6 Interprétation algébrique de la méthode de Gauss. Décomposition L·R ou méthode de Crout

3.6.1 Condition nécessaire et suffisante pour qu'il n'y ait pas de permutations dans la méthode de Gauss

Définition : Les sous-matrices principales de la matrice A sont les matrices :

$$A_k = \left(\left(a_{ij}\right)\right)_{1\le i,j\le k}, (k = 1, ..., n).$$

Et les déterminants principaux sont les $\Delta_k = \text{Dét}\left(A_k\right)$.

Si tous les déterminants principaux de A sont non nuls alors, dans la méthode de Gauss, tous les pivots seront non nuls et il ne sera pas nécessaire de faire des permutations (on ne s'occupe pas de la taille des pivots). Réciproquement, si tous les pivots sont non nuls, alors tous les déterminants principaux sont non nuls.

Exemple 1 — Si A est à diagonale strictement dominante alors toutes les sous-matrices principales de A sont aussi à diagonale strictement dominante et donc

inversibles. On en déduit donc que pour A à diagonale strictement dominante, la méthode de Gauss ne nécessite pas de permutations.

Exemple 2 — Si A est symétrique définie positive, il en est de même de toutes les sous-matrices principales de A. Donc, pour A symétrique définie positive, la méthode de Gauss ne nécessite pas de permutations.

3.6.2 Interprétation algébrique de la méthode Gauss

On suppose que A est une matrice ayant tous ses déterminants principaux non nuls.

Pour k = 1, ..., n, on note E_k la matrice déduite de la matrice identité en remplaçant la colonne N° k par :

$$C_k = \begin{pmatrix} 0 \\ . \\ 0 \\ 1 \\ c_{k+1,k} \\ . \\ C_{n,k} \end{pmatrix}, \text{ avec } c_{ik} = -\frac{a_{ik}^{(k)}}{a_{kk}^{(k)}}, \text{ pour } k < i < n.$$

en gardant les notations de la méthode de Gauss.

On vérifie facilement que :

$$A^{(k)} = E_{k-1} \cdot A^{(k-1)} = E_{k-1} \cdot E_{k-2} \cdot \ldots \cdot E_1 \cdot A, \text{ pour } k = 2, \ldots, n-1,$$

c'est-à-dire que l'opération d'élimination de x_{k-1} dans les équations k, ..., n revient à multiplier à gauche $A^{(k-1)}$ par E_{k-1} .

Au bout des n – 1 étapes, on a donc $A^{(n)} = (E_{n-1} \cdot \ldots \cdot E_1).A$ qui est triangulaire supérieure.

Ce qui donne la décomposition de A : A = L·R, où L est triangulaire inférieure de diagonale unité et R est triangulaire supérieure ($R = A^{(n)}$ et L est l'inverse du produit des E_k).

De plus une telle décomposition est unique : en effet si A = L·R et A = L'·R' avec L et L' triangulaires inférieures de diagonale unité et R et R' triangulaires supérieures, alors $L'^{-1} \cdot L = R' \cdot R^{-1}$ est à la fois triangulaire inférieure et triangulaire supérieure de diagonale unité, c'est nécessairement l'identité.

On a donc le :

Théorème : La matrice A possède une décomposition A = L·R, avec L triangulaire inférieure à diagonale unité et R triangulaire supérieure si, et seulement si tous les déterminants principaux de A sont non nuls. Dans ce cas une telle décomposition est unique et les coefficients diagonaux de R sont donnés par :

$$\begin{cases} r_{11} = a_{11} \\ r_{kk} = \dfrac{\text{Dét}(A^{(k)})}{\text{Dét}\left(A^{(k-1)}\right)}, \text{ pour } k = 2, \ldots, n \end{cases}$$

Démonstration — Voir Lascaux et Théodor, Vol. 1 p. 228.

3.6.3 Détermination pratique de la décomposition L·R

On procède par coefficients indéterminés, c'est-à-dire qu'on écrit A = L·R, on effectue le produit et on identifie, ce qui donne, en effectuant les calculs dans l'ordre indiqué :

$$\begin{cases} r_{1j} = a_{1j} \quad (j = 1, \ldots, n) \\ L_{i1} = \dfrac{a_{i1}}{a_{11}} \quad (i = 2, \ldots, n) \end{cases}$$

puis, pour k = 2, ..., n

$$r_{kj} = a_{kj} - \sum_{i=1}^{k-1} L_{ki} \cdot r_{ij} \quad (j = k, \ldots, n)$$

et

$$L_{ik} = \frac{\left(a_{ik} - \sum_{j=1}^{k-1} L_{ij} \cdot r_{jk} \right)}{r_{kk}} \quad (i = k+1, \ldots, n)$$

avec :

$$L_{ii} = 1$$

Ce qui donne la programmation structurée :

```
PROCEDURE L·R(Entrée n : Entier ; A : Matrice ; Sortie : L, R : Matrice) ;
Début
    Pour i allant de 1 à n faire
    Début
        Lii = 1 ;
        Pour j allant de i + 1 à n faire Lij = 0 ;
        Pour j allant de 1 à i – 1 faire rij = 0 ;
    Fin ;
    Pour j allant de 1 à n faire r1j = a1j ;
    Pour i allant de 2 à n faire Li1 = ai1 / a11 ;
    Pour k allant de 2 à n faire
    Début
      Pour j allant de k à n faire
      Début
        S = 0 ;
        Pour i allant de 1 à k - 1 faire S = S + rij·Lki ;
        rkj = akj – S ;
      Fin ;
      Si k ≠ n
      Alors Début
            Pour i allant de k + 1 à n faire
            Début
              S = 0 ;
              Pour j allant de 1 à k – 1 faire S = S + Lij·rjk ;
              Lik = (aik – S)/rkk ;
            Fin ;
```

```
        Fin ;
    Fin ;
Fin ;
```

3.6.4 Résolution d'un système linéaire par la méthode de Crout

Comme A = L·R, l'équation A·x = b équivaut aux deux systèmes triangulaires :

$$\begin{cases} L \cdot y = b \ \text{(triangulaire inférieur)} \\ R \cdot x = y \ \text{(triangulaire supérieur)} \end{cases}$$

Cette méthode est intéressante pour résoudre en parallèle plusieurs systèmes linéaires de même matrice A, car la décomposition L·R ne fait pas intervenir le second membre.

3.6.5 Calcul de l'inverse d'une matrice par la méthode de Crout

Avec A = L·R, on a $A^{-1} = R^{-1} \cdot L^{-1}$, et il suffit alors d'utiliser les procédures d'inversion des matrices triangulaires.

3.7 Cas des matrices tridiagonales

3.7.1 Notations et hypothèses

On suppose que A est tridiagonale, inversible et telle qu'il n'y ait pas de permutations dans la méthode de Gauss (par exemple si A est à diagonale strictement dominante ou symétrique définie positive).

On note :

$$A = \begin{pmatrix} a_1 & c_1 & 0 & . & . & 0 \\ b_2 & a_2 & c_2 & 0 & . & 0 \\ . & . & . & . & . & . \\ 0 & . & 0 & b_{n-1} & a_{n-1} & c_{n-1} \\ 0 & . & . & 0 & b_n & a_n \end{pmatrix}$$

3.7.2 Décomposition L·R d'une matrice tridiagonale

On cherche les matrices L et R sous la forme bidiagonale, soit :

$$L = \begin{pmatrix} 1 & 0 & 0 & . & . & 0 \\ L_2 & 1 & 0 & 0 & . & 0 \\ . & . & . & . & . & . \\ 0 & . & 0 & L_{n-1} & 1 & 0 \\ 0 & . & . & 0 & L_n & 1 \end{pmatrix} \text{ et } R = \begin{pmatrix} d_1 & r_1 & 0 & . & . & 0 \\ 0 & d_2 & r_2 & 0 & . & 0 \\ . & . & . & . & . & . \\ 0 & . & 0 & 0 & d_{n-1} & r_{n-1} \\ 0 & . & . & 0 & & d_n \end{pmatrix}$$

On effectuant le produit L·R et en identifiant avec les coefficients de A, on obtient alors :

$$\begin{cases} d_1 = a_1 \\ r_j = c_j \ \ (j = 1, \ldots, n - 1) \end{cases}$$

puis :

$$\begin{cases} L_{j+1} = \dfrac{b_{j+1}}{d_j} \\ d_{j+1} = a_{j+1} - r_j \cdot L_{j+1} \end{cases} \quad (j = 1, \ldots, n - 1)$$

La programmation structurée est alors évidente.

3.7.3 Méthode de résolution d'un système tridiagonal

Soit à résoudre A·x = e, pour A tridiagonale, vérifiant les hypothèses précédentes.

Première méthode : utilisation de la décomposition L·R — Tout d'abord, on résout L·y = e, ce qui donne :

$$\begin{cases} y_1 = e_1 \\ y_j = e_j - L_j \cdot y_{j-1} \quad (j = 2, \ldots, n) \end{cases}$$

Puis on résout R·x = y, ce qui donne :

$$\begin{cases} x_n = \dfrac{y_n}{d_n} \\ x_j = \dfrac{\left(y_j - r_j \cdot x_{j+1}\right)}{d_j} \quad (j = n - 1, \ldots, 1) \end{cases}$$

Là encore la préparation structurée est évidente.

Deuxième méthode : utilisation de la méthode de Gauss — On obtient alors une méthode aussi appelée méthode du double balayage de Cholesky.

En supposant qu'il n'y a pas de permutations dans la méthode de Gauss (une telle permutation ferait perdre le caractère tridiagonal), on n'aura à chaque étape qu'une ligne à traiter et pour chaque ligne seulement deux opérations.

A l'étape k de la méthode, la ligne L_k du système devient :

$$L_k - m \cdot L_{k-1} \ (k = 2, \ldots, n)$$

avec :

$$m = \frac{b_k}{a_{k-1}}$$

Ce qui donne les formules de transformations :

$$b_k = 0 \ ;$$
$$a_k = a_k - m \cdot c_{k-1}$$
$$c_k \text{ inchangé}$$
$$e_k = e_k - m \cdot e_{k-1}$$

Le système triangulaire supérieur obtenu sera bidiagonal et aura pour solution :

$$\begin{cases} x_n = \dfrac{e_n}{a_n} \\ x_i = \dfrac{\left(e_i - c_i \cdot x_{i+1}\right)}{a_i} \quad (i = n - 1, \ldots, 1) \end{cases}$$

Remarque — Le produit des a_i donnera, en fin d'opération, le déterminant de A.

Pour la programmation structurée, en vue d'économiser la place mémoire, on aura intérêt à stocker une matrice tridiagonale dans trois vecteurs. Une telle matrice sera donc notée :

$$T = \left(T_{ij}\right)_{\substack{1\le i\le n\\ 1\le j\le 3}}$$

où i est le numéro de la ligne et j celui de la diagonale, c'est-à-dire que $\left(T_{i1}\right)_{1\le i\le n}$ représente la diagonale inférieure, $\left(T_{i2}\right)_{1\le i\le n}$ la diagonale et $\left(T_{i3}\right)_{1\le i\le n}$ la diagonale supérieure.

Ce qui donne la procédure :

```
PROCEDURE SystTridiag(Entrée n : Entier ;
                      Entrée_Sortie T : MatriceTridiagonale ; b : Vecteur) ;
Début
  Pour i Allant de 2 à n Faire
  Début
    Si (T_{i–1,2} = 0)
    Alors Arréter('Déterminant nul') ;
    m = T_{i,1}/T_{i–1,2} ;
    T_{i,2} = T_{i,2} – m*T_{i–1,3} ;
    b_i = b_i – m*b_{i–1} ;
  Fin ;
  Si (T_{n,2} = 0)
  Alors Arréter('Déterminant nul') ;
  b_n = b_n/T_{n,2} ;
  Pour i Allant de n – 1 à 1 Faire b_i = (b_i – T_{i,3}*b_{i+1})/T_{i,2} ;
Fin ;
```

Remarque — En fin d'opération les valeurs initiales de la matrice T et du second membre b sont perdues, le vecteur b contenant la solution du système.

3.7.4 Déterminant d'une matrice tridiagonale

On reprend les notations du paragraphe précédent, en notant D_n le déterminant d'une matrice tridiagonale d'ordre n.

En développant suivant la dernière ligne, on a :

$$D_n = a_n \cdot D_{n-1} - b_n \cdot c_{n-1} \cdot D_{n-2}$$

Ce qui donne, avec les valeurs initiales $D_0 = 1$ et $D_1 = a_1$, un algorithme très simple de calcul.

3.7.5 Exemples d'applications

- Interpolation spline cubique.
- Résolution de certaines équations différentielles linéaires par la méthode des différences finies (problème de Dirichlet linéaire en dimension 1).
- Résolution de l'équation de la chaleur à une dimension par la méthode des différences finies.

3.8 Résolution des systèmes symétriques

On suppose que A est symétrique et qu'il n'y a pas de permutations dans la méthode de Gauss.

3.8.1 Décomposition $L \cdot D \cdot {}^tL$

Dans la décomposition L·R de A, on a Det(R) = Det(A) qui est non nul, donc tous les termes diagonaux de R sont non nuls et on peut écrire R sous la forme : R = D·R', où D est diagonale et R' est triangulaire supérieure à diagonale unité (il suffit de diviser chaque ligne de R par son terme diagonal). On a donc A = L·D·R', puis en écrivant que ${}^tA = A$ et en utilisant le fait que la décomposition L·R est unique, on déduit que $R' = {}^tL$.

On a donc pour toute matrice symétrique A, dont tous les déterminants principaux sont non nuls, la décomposition unique $A = L \cdot D \cdot {}^tL$, la matrice L étant triangulaire inférieure et à diagonale unité et la matrice D étant diagonale.

Remarque : Calcul de la signature d'une matrice symétrique — On sait qu'une matrice symétrique a toutes ses valeurs propres réelles et que, pour A inversible, le couple d'entiers (p,q) formé du nombre p de valeurs propres strictement positives et du nombre q = n – p de valeurs propres strictement négatives est uniquement déterminé par A.

Ce couple d'entiers est la signature de A et la décomposition ci-dessus nous permet de la calculer.

On dit que deux matrices A et B sont congruentes, s'il existe une matrice inversible L telle que $A = L \cdot B \cdot {}^tL$. Et le théorème de Sylvester nous dit que deux matrices congruentes ont même signature.

Avec les notations ci-dessus, on a donc :

$$\text{Signature}(A) = \text{Signature}(D) = (p,q)$$

où p est le nombre de termes strictement positifs de la diagonale de D.

De plus A est définie positive, si et seulement si, tous les coefficients de D sont strictement positifs.

3.8.2 Calculs pratiques

Comme pour la décomposition L·R, on trouve les coefficients de L et D par identification, ce qui donne, $d_1 = a_{11}$ et pour i = 2, ..., n :

$$\begin{cases} L_{ij} = \dfrac{\left(a_{ij} - \displaystyle\sum_{k=1}^{j-1} L_{ik} \cdot d_k \cdot L_{jk} \right)}{d_j} \quad (j = 1, \ldots, i - 1) \\ d_i = \left(a_{ii} - \displaystyle\sum_{k=1}^{i-1} L_{ik}^2 \cdot d_k \right) \end{cases}$$

3.8.3 Programmation structurée

PROCEDURE L·R_Sym(Entrée n : Entier ; A : Matrice ; Sortie : L : Matrice ; D : Vecteur) ;
Début

```
    Pour i allant de 1 à n faire
    Début
      Pour j allant de 1 à i – 1 faire
      Début
        S = 0 ;
        Pour k allant de 1 à j – 1 faire S = S + L_ik·d_k·L_jk ;
        L_ij = (a_ij – S)/d_j ;
      Fin ;
      S = 0 ;
      Pour k allant de 1 à i – 1 faire S = S + L²_ik ·d_k ;
      d_i= a_ii·S ;
    Fin ;
Fin ;
```

3.8.4 Cas particulier des matrices symétriques définies positives. Décomposition $B\cdot{}^tB$ de Cholesky

Si A est symétrique définie positive, dans la décomposition précédente, on a alors $d_i > 0$ pour tout i = 1, ..., n. On peut donc écrire $D = D'^2$, et en posant : $B = L\cdot D'$, on a $A = B\cdot{}^tB$ avec B triangulaire inférieure, la diagonale de B étant formée des $\pm\sqrt{d_i}$. Si on impose la positivité des coefficients diagonaux de B une telle décomposition est unique. on a donc le :

Théorème : Une matrice A est symétrique définie positive si, et seulement si il existe une matrice B triangulaire inférieure inversible telle que $A = B\cdot{}^tB$. De plus une telle décomposition est unique si on impose la positivité des coefficients diagonaux de B.

Le calcul effectif des coefficients de B se fait par identification ce qui donne :

$$\begin{cases} b_{ij} = \dfrac{\left(a_{ij} - \displaystyle\sum_{k=1}^{j-1} b_{ik}\cdot b_{jk}\right)}{b_{jj}} & (j = 1, \ldots, i-1) \\ b_{ii}^2 = a_{ii} - \displaystyle\sum_{k=1}^{i-1} b_{ik}^2 \end{cases} \quad (i = 1, \ldots, n)$$

La programmation structurée étant évidente.

Le nombre d'opérations, dans la décomposition de Cholesky est alors un $O(n^3)$.

Pour calculer l'inverse de A il suffit d'inverser la matrice triangulaire B.

3.9 Résolution des systèmes linéaires par des méthodes itératives

3.9.1 Remarques préliminaires

(i) Les méthodes directes pour résoudre les systèmes linéaires ont un caractère exact, mais pour les « grands systèmes », (i.e. pour n > 100) la propagation des erreurs d'arrondis en diminue l'efficacité.

(ii) Les méthodes itératives sont bien adaptées au cas des matrices creuses (i.e. avec beaucoup de zéros), car ces méthodes ne transforment pas la matrice de

départ, contrairement à la méthode de Gauss qui peut faire apparaître des termes non nuls à la place de zéros.

3.9.2 Principe des méthodes itératives

L'idée des méthodes itératives est la suivante : construire une suite $(x^{(k)})$ d'éléments de R^n qui va converger vers x solution de $A \cdot x = b$, où Dét(A) est non nul ; le calcul de chaque $x^{(k)}$ devant être plus facile que la résolution directe de $A \cdot x = b$.

On écrira la matrice A sous la forme A = M - N, où M est « facilement inversible », et la résolution de $A \cdot x = b$ est ramenée au *« problème de point fixe »* :

$$\text{trouver } x \text{ dans } R^n \text{ solution de : } x = M^{-1} \cdot N \cdot x + M^{-1} \cdot b.$$

Pour résoudre ce problème, on utilise la *« méthode des approximations successives »*, c'est-à-dire qu'on considère la suite $(x^{(k)})$ définie par :

$$(1) \quad \begin{cases} x^{(0)} \text{ donné dans } R^n \\ x^{(k+1)} = M^{-1} \cdot N \cdot x^{(k)} + M^{-1} \cdot b \end{cases}$$

Si cette suite converge, c'est nécessairement vers la solution de $A \cdot x = b$.

Problèmes posés par les méthodes itératives

- A quelles conditions sur la matrice M, la suite définie par (1) converge ?
- Comment choisir M facilement inversible ?
- Quelle est la rapidité de la convergence ?

Convergence de la suite définie par (1)

On suppose que A est inversible et x désigne la solution de $A \cdot x = b$.

Si A = M – N, avec M inversible, on pose $B = M^{-1} \cdot N$. On a alors le :

Théorème : La suite définie par (1) est convergente, quelle que soit la valeur initiale $x^{(0)}$, si et seulement si, $\rho(B) < 1$, où $\rho(B)$ désigne le rayon spectral de B.

Démonstration — En écrivant que $A \cdot x = b$ équivaut à $x = B \cdot x + M^{-1} \cdot b$, on déduit que :

$$x^{(k+1)} - x = B \cdot \left(x^{(k)} - x \right) = B^{k+1} \cdot \left(x^{(0)} - x \right), \text{ pour tout } k \in N,$$

et donc $(x^{(k)})$ va converger vers x pour toute valeur initiale si, et seulement si $(B^{(k)})$ tend vers 0, ce qui équivaut à $\rho(B) < 1$.

3.9.3 Méthode de Jacobi

On prend M = D, matrice diagonale définie par $d_{ii} = a_{ii}$ pour tout i. Si on veut que M soit inversible il faut donc supposer que tous les coefficients diagonaux de A sont non nuls.

L'algorithme de construction des $x^{(k)}$, pour un tel choix de M est alors le suivant :

$$a_{ii} \cdot x_i^{(k+1)} = -\left(a_{i1} \cdot x_1^{(k)} + \ldots + a_{i,i-1} \cdot x_{1-1}^{(k)}\right) - \left(a_{i,i+1} \cdot x_{i+1}^{(k)} + \ldots + a_{i,n} \cdot x_n^{(k)}\right) + b_i \quad (i = 1, \ldots, n)$$

Remarques — (i) Le calcul des composantes de $x^{(k+1)}$ nécessite de garder en mémoire le vecteur $x^{(k)}$. Une itération va donc immobiliser 2·n cases mémoires.
(ii) On peut décider d'arrêter les itérations à un rang MaxIter donné et lorsque $\left\|x^{(k+1)} - x^{(k)}\right\| < \varepsilon \cdot \left\|x^{(k+1)}\right\|$ (ε une précision Donnée).

(iii) Comme valeur initiale, on peut prendre $x^{(0)} = \left(\left(\frac{b_i}{a_{ii}}\right)\right)$.

Ce qui donne la programmation structurée :

```
PROCEDURE Jacobi(Entrée n : Entier ; A : Matrice ; b : Vecteur ; Sortie x : Vecteur) ;
Début
  Pour i allant de 1 à n faire
  Début
     Si a_ii = 0 Alors Stop Sinon x_i = b_i/a_ii ;
   Fin ;
  p = 0 ; Continuer = Vrai ;
  Tant que (p < MaxIter) et (Continuer) faire
  Début
    p = p + 1 ;
    Pour i allant de 1 à n faire
    Début
      S = b_i ;
      Pour j allant de 1 à n faire
      Début
        Si j ≠ i Alors S = S – a_ij·x_j ;
      Fin ;
      y_i = S/a_ii ;
    Fin ;
    Continuer = ||y – x|| > ε·||y|| ;
    x = y ;
 Fin ;
 Si Non Continuer
 Alors afficher('Solution approchée = ',x)
 Sinon arrêter (convergence trop lente ou divergence) ;
Fin ;
```

Cas des matrices à diagonale strictement dominante

Théorème : Si A est à diagonale strictement dominante, alors la méthode de Jacobi converge.

Démonstration — Si $J = D^{-1} \cdot N = ((b_{ij}))$, avec A = D - N, on a :

$$\|J\| = \text{Max}\left\{\sum_{j=1}^{n} \left|b_{ij}\right| , \ i = 1, \ldots, n\right\} = \text{Max}\left\{\sum_{j \neq i} \left|\frac{a_{ij}}{a_{ii}}\right| , \ i = 1, \ldots, n\right\} < 1$$

c'est-à-dire que $\rho(J) < 1$ et la méthode de Jacobi est convergente.

3.9.4 Méthode de Gauss-Seidel

On prend M définie par :

$$\begin{cases} m_{ij} = 0 \ , \text{ pour } 1 \le i < j \le n; \\ m_{ij} = a_{ij} \ , \text{ pour } 1 \le j \le i \le n; \end{cases}$$

En notant A = M - N, $x^{(k+1)}$ est solution du système triangulaire inférieur $M \cdot x^{(k+1)} = N \cdot x^{(k)} + b$ qui se résout en « descente », d'où l'algorithme de Gauss-Seidel :

$$a_{ii} \cdot x_i^{(k+1)} = -\left(a_{i1} \cdot x_1^{(k+1)} + \ldots + a_{i,i-1} \cdot x_{i-1}^{(k+1)}\right) - \left(a_{i,i+1} \cdot x_{i+1}^{(k)} + \ldots + a_{i,n} \cdot x_n^{(k)}\right) + b_i \quad (i = 1, \ldots, n)$$

Remarque 1 — Cet algorithme est en fait une amélioration de l'algorithme de Jacobi : dans le calcul de $x_i^{(k+1)}$, on utilise les composantes 1, ..., i – 1 de $x^{(k+1)}$ (alors que dans la méthode de Jacobi ce sont celles de $x^{(k)}$ qui sont utilisées) et les composantes i + 1, ..., n de $x^{(k)}$ (comme dans la méthode de Jacobi). Cet algorithme sera donc en général plus performant que celui de Jacobi.

Remarque 2 — Pour une itération on ne garde donc que n termes en mémoire.

Ce qui donne la programmation structurée :

```
PROCEDURE Gauss-Seidel(Entrée n : Entier ; A : Matrice ; b : Vecteur ; Sortie x : Vecteur) ;
Début
    Continuer = Vrai ; p = 0 ;
    Pour i allant de 1 à n faire
    Début
        Si a_ii = 0 Alors Stop Sinon x_i = b_i/a_ii ;
    Fin ;
    Tant que (p < MaxIter) et (Continuer) faire
    Début
      p = p + 1 ;
      DeltaMax = 0 ;
      Pour i allant de 1 à n faire
      Début
          S = b_i ;
          Pour j allant de 1 à n faire
          Début
              Si j ≠ i Alors S = S – a_ij·x_j ;
          Fin ;
          S = S/a_ii ;
          Si |x_i – S| > DeltaMax
          Alors DeltaMax = |x_i – S| ;
          x_i = S ;
      Fin ;
      Continuer = DeltaMax > Epsilon ;
  Fin ;
  Si Non Continuer
  Alors Afficher('Solution approchée',x)
  Sinon Arrêter('convergence trop lente ou divergence') ;
Fin ;
```

Théorème : La méthode de Gauss-Seidel converge si A vérifie l'une des conditions suivantes :
(i) A à diagonale strictement dominante ;
(ii) A symétrique définie positive.

Démonstration — (i) Soit A à diagonale strictement dominante, $G = M^{-1} \cdot N$, μ une valeur propre de G et x, non nul, un vecteur propre associé. On a alors : $N \cdot x = \mu \cdot M \cdot x$, c'est-à-dire :

$$\begin{cases} -\sum_{j=i+1}^{n} a_{ij} \cdot x_j = \mu \cdot \sum_{j=1}^{i} a_{ij} \cdot x_j \quad (i = 1, \ldots, n - 1) \\ 0 = \mu \cdot \sum_{j=1}^{n} a_{nj} \cdot x_j \end{cases}$$

Si on suppose que $|\mu| \geq 1$, en prenant i tel que $\|x\| = |x_i|$, on déduit que : $|\mu| \cdot |a_{ii}| \leq \sum_{j \neq i} |a_{ij}| \cdot |\mu|$, ce qui contredit A à diagonale strictement dominante.

(ii) Soit A symétrique définie positive, μ un vecteur propre de G et x, non nul, un vecteur propre associé. En écrivant A = D + E + F, où D est la diagonale de A, E le triangle inférieur strict et F le triangle supérieur strict, on en déduit alors que :

$$\langle x | A \cdot x \rangle = (1 - \mu) \cdot \langle x | D \cdot x \rangle + (1 - \mu) \cdot \langle x | E \cdot x \rangle$$
$$\langle x | A \cdot x \rangle = (1 - \mu) \cdot \langle x | D \cdot x \rangle + (1 - \mu) \cdot \langle x | F \cdot x \rangle$$
$$\langle x | A \cdot x \rangle = \langle x | D \cdot x \rangle + \langle x | E \cdot x \rangle + \langle x | F \cdot x \rangle$$

D'où :

$$\left(1 - |\mu|^2\right) \cdot \langle x | A \cdot x \rangle = |1 - \mu|^2 \cdot \langle x | D \cdot x \rangle$$

avec $\langle x | A \cdot x \rangle > 0$ et $\langle x | D \cdot x \rangle > 0$ car A et D sont définies positives, donc $|\mu| < 1$. C'est-à-dire que $\rho(G) < 1$ et la méthode est convergente.

Remarque — Quand les méthodes de Jacobi et de Gauss-Seidel sont convergentes il vaut mieux choisir celle de Gauss-Seidel. Mais il se peut que la méthode de Gauss-Seidel diverge alors que celle de Jacobi converge comme le montre l'exemple suivant :

$$A = \begin{pmatrix} 1 & 2 & -2 \\ 1 & 1 & 1 \\ 2 & 2 & 1 \end{pmatrix}$$

3.9.5 Comparaison des méthodes de Jacobi et de Gauss-Seidel dans le cas des matrices tridiagonales d'ordre n ≥ 3

On se donne :

$$A = \begin{pmatrix} a_1 & c_1 & 0 & . & . & 0 \\ b_2 & a_2 & c_2 & 0 & . & 0 \\ . & . & . & . & . & . \\ 0 & . & 0 & b_{n-1} & a_{n-1} & c_{n-1} \\ 0 & . & . & 0 & b_n & a_n \end{pmatrix}$$

avec $n \geq 3$ et tous les a_i non nuls.

On a alors le

Lemme — 1°) Si μ est valeur propre de J, alors $-\mu$ est aussi valeur propre de J.
2°) Le polynôme caractéristique de J s'écrit : $P_J(x) = P(x^2) \cdot x^q$ où P est un polynôme de degré p et $2 \cdot p + q = n$.
Si P_G est le polynôme caractéristique de G, alors : $P_G(x^2) = x \cdot P_J(x)$.

On en déduit donc le

Théorème : Si A est tridiagonale d'ordre $n > 3$ alors : $\rho(G) = (\rho(J))^2$.
Donc les deux méthodes convergent ou divergent simultanément, et dans le cas de la convergence, c'est la méthode de Gauss-Seidel qui est la plus rapide.

3.9.6 Méthode de relaxation

En vue d'accélérer la convergence de la méthode de Gauss-Seidel, on introduit un paramètre dans la matrice $G = M^{-1} \cdot N$. C'est-à-dire qu'on pose $M_\omega = \left(\left(m_{ij}\right)\right)$, avec :

$$\begin{cases} m_{ij} = 0 \text{ , pour } 1 \leq i < j \leq n \\ m_{ii} = \dfrac{a_{ii}}{\omega} \text{ , pour } i = 1, \ldots, n \text{ , avec :} \\ m_{ij} = a_{ij} \text{ , pour } 1 \leq j < i \leq n \end{cases}$$

où ω est non nul à préciser.

On pose $L_\omega = M^{-1} \cdot N_\omega$, si $A = M_\omega - N_\omega$.

Si D est la diagonale de A, E le triangle inférieur strict et F le triangle supérieur strict, on a :

$$M_\omega = \omega^{-1} \cdot D + E \text{ et } A = D + E + F$$

On a alors :

$$L_\omega = \left(D + \omega \cdot E\right)^{-1} \cdot \left(\left(1 - \omega\right) \cdot D - \omega \cdot F\right)$$

Le vecteur $x^{(k+1)}$ est alors solution du système triangulaire :

$$\left(D + \omega \cdot E\right) \cdot x^{(k+1)} = \left(\left(1 - \omega\right) \cdot D - \omega \cdot F\right) \cdot x^{(k)} + \omega \cdot b$$

et l'algorithme de calcul des $x^{(k)}$ est :

$$x_i^{(k+1)} = \left(1 - \omega\right) \cdot x_i^{(k)} + \omega \cdot \hat{x}_i^{(k+1)} \quad (i = 1, \ldots, n)$$

où $\hat{x}_i^{(k+1)}$ est donné par les formules de Gauss-seidel en fonction des composantes 1, ... i - 1 de $x^{(k+1)}$ et des composantes i + 1, ..., n de $x^{(k)}$, soit :

$$a_{ii} \cdot \hat{x}_i^{(k+1)} = -\left(a_{i1} \cdot x_1^{(k+1)} + \ldots + a_{i,i-1} \cdot x_{1-1}^{(k+1)}\right) - \left(a_{i,i+1} \cdot x_{i+1}^{(k)} + \ldots + a_{i,n} \cdot x_n^{(k)}\right) + b_i \quad (i = 1, \ldots, n)$$

Ce qui peut encore s'écrire :

$$x^{(k+1)} = x^{(k)} - \omega \cdot \xi^{(k+1)} \; (k \geq 0)$$

où $\xi^{(k+1)}$ est le *« vecteur résidu »* défini par :

$$\xi_i^{(k+1)} = \frac{\left(\sum_{j=1}^{i-1} a_{ij} \cdot x_j^{(k+1)} + \sum_{j=i}^{n} a_{ij} \cdot x_j^{(k)} - b_i\right)}{a_{ii}} \quad (i = 1, \ldots, n)$$

Un test de convergence sera alors $\|\xi\| < \varepsilon$, où ε est une précision donnée.

La programmation structurée est alors la suivante :

```
PROCEDURE Relaxation(Entrée n : Entier ; A : Matrice ; b : Vecteur ; ω : Réel ;
                     Sortie : x : Vecteur) ;
Début
   p = 0 ;
   Continuer = Vrai ;
   Pour i Allant de 1 à n Faire
  Début
     Si a_ii = 0 Alors Stop Sinon x_i = b_i/a_ii ;
   Fin ;
   Tant que (p < MaxIter) et (Continuer) Faire
   Début
     p = p + 1 ;
     NormeResidu = 0 ;
     Pour i Allant de 1 à n Faire
     Début
        Residu = -b_i ;
        Pour j Allant de 1 à n Faire
          Residu = Residu + a_ij·x_j ;
        Residu = Residu/a_ii ;
        x_i = x_i – ω·Residu ;
        Si |Residu| > NormeResidu
        Alors NormeResidu = |Residu| ;
     Fin ;
     Continuer = (NormeResidu > ε) ;
   Fin ;
   Si (p = MaxIter)
   Alors Arrêter('convergence trop lente ou divergence') ;
Fin ;
```

Une condition nécessaire sur ω pour que la méthode converge est donnée par le

Théorème : Si, μ_1, ..., μ_n sont les valeurs propres de L_ω , on a alors $\prod_{i=1}^{n} \mu_i = (1-\omega)^n$. Donc $\rho(L_\omega) \geq |\omega - 1|$. On en déduit donc que la méthode de relaxation ne peut converger que si $\omega \in]0,2[$.

Démonstration — Voir Stoer et Burlisch p. 547.

Cas des matrices symétriques définies positives — Dans ce cas la condition précédente est aussi suffisante :

Théorème (Ostrowski , Reich) :
(a) Pour toute valeur propre μ de L , on a : $\mu \neq 1$ et :

$$\frac{\left(1-|\mu|^2\right)}{|1-\mu|^2} \cdot \langle x | A \cdot x \rangle = (2-\omega) \cdot \frac{\langle x | A \cdot x \rangle}{\omega}$$

où x est un vecteur propre associé à μ.
(b) Donc une condition nécessaire et suffisante de convergence de la méthode de relaxation, pour une matrice symétrique définie positive, est $\omega \in$]0,2[.

Démonstration — Voir Stoer et Burlisch p. 547.

Remarque — Si de plus A est tridiagonale, on a le

Théorème (Young , Varga) : Si A est symétrique définie positive et tridiagonale, alors :
(i) les méthodes de Jacobi et de relaxation pour $\omega \in$]0,2[sont convergentes ;
(ii) la fonction $\omega \mapsto \rho(L_\omega)$ a l'allure indiquée ci-dessous :

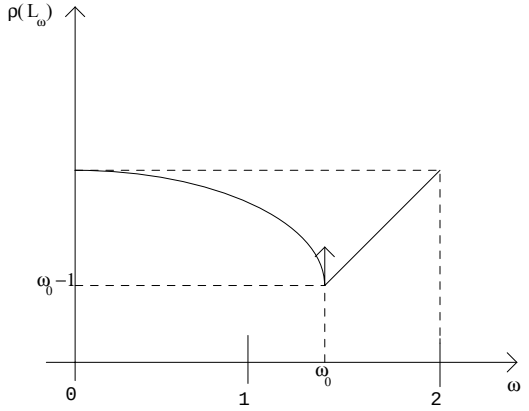


Figure 3.1

La valeur du paramètre de relaxation optimal étant :

$$\omega_\circ = \frac{2}{1+\sqrt{1-\rho(J)^2}}$$

avec

$$\rho\left(L_{\omega_0}\right) = \omega_0 - 1.$$

Démonstration — Voir Stoer et Burlisch p.553.

Remarque 1 : Procédé d'accélération de la convergence de Tchébycheff — Si on sait calculer la valeur optimale de ω, ce choix n'est pas judicieux dès la première itération. L'idée de Tchébycheff est de changer de paramètre ω à chaque itération de la manière suivante : au départ, on prend $\omega_1 = 1$, puis à l'étape suivante on prend

$$\omega_2 = \frac{1}{1 - \frac{\rho(J)^2}{2}}$$

et aux étapes suivantes on prendra :

$$\omega_{k+1} = \frac{1}{1 - \frac{\rho(J)^2 \cdot \omega_k}{4}}$$

La suite (ω_k) converge en fait vers ω_0 (la valeur optimale) et le procédé de Tchébycheff va diminuer, en général, le nombre d'itérations.

Remarque 2 : Méthode de relaxation par blocs — Lorsque la dimension du système est très grande, on aura intérêt à partitionner la matrice A en blocs, ce découpage étant adapté à la forme particulière de A (Pour la résolution de certaines équations aux dérivées partielles, on doit résoudre des systèmes tridiagonaux par blocs.).

On a donc :

$$A = \begin{pmatrix} A_{11} & A_{12} & . & . & . & A_{1m} \\ A_{21} & A_{22} & . & . & . & . \\ . & . & . & . & . & . \\ A_{m1} & A_{m2} & . & . & . & A_{mm} \end{pmatrix}$$

où A_{ij} est une matrice à n_i lignes et n_j colonnes, avec $\sum_{i=1}^{n} n_i = n$.

De plus, les matrices A_{ii} sont supposées inversibles.

Le second membre b et l'inconnue x sont alors partitionnés de manière analogue, soit :

$$b = \begin{pmatrix} B_1 \\ B_2 \\ . \\ . \\ B_m \end{pmatrix} \text{ et } x = \begin{pmatrix} X_1 \\ X_2 \\ . \\ . \\ X_m \end{pmatrix}$$

Le système à résoudre peut alors s'écrire par blocs :

$$\sum_{j=1}^{m} A_{ij} \cdot X_j = B_i \quad (i = 1, ..., m)$$

Et les formules pour la méthode de relaxation s'écrivent :

$$X^{(k+1)} = X^{(k)} - \omega \cdot \Xi^{(k+1)} \quad (k \geq 0)$$

où $\Xi^{(k+1)}$ est le *« vecteur résidu »* défini par :

$$\Xi^{(k+1)} = A_{ii}^{-1} \cdot \left(\sum_{j=1}^{i-1} A_{ij} \cdot X_j^{(k+1)} + \sum_{j=i}^{n} A_{ij} \cdot X_j^{(k)} - B_i \right) \quad (i = 1, ..., m)$$

Cette méthode sera intéressante si les A_{ii} sont facilement inversibles.

3.9.7 Exemple d'application

La méthode de relaxation sera utilisée pour la résolution d'une équation aux dérivées partielles de type elliptique par la méthode des différences finies : Cf. le chapitre sur la résolution des équations aux dérivées partielles paragraphe (5.5). On aura à résoudre des systèmes tridiagonaux par blocs, les matrices blocs de la diagonale étant tridiagonales et les autres diagonales.

Dans le cas particulier de l'équation de Poisson, $\Delta \cdot u = f$, on est ramené à résoudre un système tridiagonal par blocs :

$$T \cdot u = v \quad (1)$$

où la matrice T, tridiagonale par blocs est définie par :

$$T = \begin{pmatrix} A & I_d & 0 & . & . & 0 \\ I_d & A & I_d & 0 & . & 0 \\ . & . & . & . & . & . \\ 0 & . & . & 0 & I_d & A \end{pmatrix}$$

où I_d désigne la matrice identité de R^n et A, d'ordre n est définie par :

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

les vecteurs étant notés :

$$u = \begin{pmatrix} u_1 \\ u_2 \\ . \\ . \\ u_m \end{pmatrix}, \text{ avec } u_j = \begin{pmatrix} u_{1j} \\ u_{2j} \\ . \\ . \\ u_{nj} \end{pmatrix}, \text{ pour } j = 1, ..., m$$

La composante vectorielle N°j de M·u est alors :

$$u_{j-1} + A \cdot u_j + u_{j+1} \quad (j = 1, ..., m)$$

avec la convention $u_0 = u_{m+1} = 0$.

C'est-à-dire que la composante N° (i,j) de M·u est :

$$u_{i,j-1} + u_{i-1,j} - 4 \cdot u_{i,j} + u_{i+1,j} + u_{i,j+1} \quad (i = 1, ..., n)$$

avec les conventions $u_{0,j} = u_{n+1,j} = 0$, pour tout j = 1, ..., m.

La suite $\left(u^{(k)}\right)_{k\in N}$ construite par la méthode de relaxation, avec paramètre ω dans]0,2[est alors définie par :

$$\begin{cases} u^{(0)} \text{ donné dans } R^{n\cdot m} \\ u^{(k+1)} = u^{(k)} - \omega\cdot\xi^{(k+1))} \quad (k \geq 0) \end{cases}$$

où le vecteur résidu $\xi^{(k+1)}$ est défini par :

$$\xi_{ij}^{(k+1)} = \frac{1}{-4}\cdot\left(u_{i,j-1}^{(k+1)} + u_{i-1,j}^{(k+1)} - 4\cdot u_{i,j}^{(k)} + u_{i+1,j}^{(k)} + u_{i,j+1}^{(k)} - v_{i,j}\right) \quad (1\leq i\leq n,\ 1\leq j\leq m)$$

pour $1 < i < n$ et $1 < j < m$.

Dans ce cas on connaît explicitement le rayon spectral de la matrice de Jacobi :

$$\rho(J) = \frac{\mathrm{Cos}\left(\frac{\pi}{n+1}\right) + \mathrm{Cos}\left(\frac{\pi}{m+1}\right)}{2}$$

ce qui permet d'appliquer le procédé d'accélération de Tchébycheff.

Pour la programmation structurée, on se reportera au paragraphe (5.5.4) du chapitre sur la résolution des équations aux dérivées partielles par la méthode des différences finies.

4. Calcul des valeurs propres et des vecteurs propres de certaines matrices réelles

4.1 Introduction

Le problème de la détermination des valeurs propres d'une matrice est, en général, beaucoup plus difficile que celui de la résolution des systèmes linéaires.

Tout d'abord nous allons voir la méthode de la puissance itérée qui permet de calculer la valeur propre de plus grand module sous certaines hypothèses. Puis en itérant ce procédé on peut en déduire toutes les autres quand elles sont toutes distinctes en modules. Cette méthode, peu performante, est à utiliser pour les matrices ayant toutes les valeurs propres distinctes et si on en cherche seulement quelques-unes.

D'autres méthodes sont les méthodes de Rutishauser, Givens et Householder et de Jacobi qui reposent sur le principe suivant : on construit une suite (A_k) de matrices semblables à A, donc ayant les mêmes valeurs propres, et qui « converge » vers une matrice plus simple (diagonale, pour Jacobi, ou tridiagonale, pour Givens et Householder, ou triangulaire ; pour Rutishauser) encore semblable à A. Les valeurs propres cherchées sont alors les termes diagonaux de la matrice limite, dans le cas des méthodes de Jacobi et de Rutishauser.

Les problèmes posés par ces méthodes sont :
(i) savoir décrire le passage de A_k à A_{k+1} aussi simplement que possible c'est-à-dire avec un coût de calcul peu élevé ;
(ii) les valeurs propres doivent être bien conservées (stabilité numérique de la méthode).

Enfin une autre idée est d'essayer de déterminer le polynôme caractéristique de A puis d'utiliser un procédé numérique de recherche des racines d'une équation polynomiale. On verra, par exemple, les méthodes de Leverrier, de Krylov et de Souriau. Mais de telles méthodes sont déconseillées pour des grandes matrices à cause des problèmes d'instabilité numérique (évaluation du polynôme, extraction des racines d'un polynôme mal déterminé qui peut être de degré élevé ...).

4.2 Calcul du rayon spectral d'une matrice. Méthode de la puissance itérée

4.2.1 Hypothèses et notations

On se donne une matrice A à coefficients réels telle que la valeur propre dominante soit unique, c'est-à-dire que si μ_1, ..., μ_n sont les valeurs propres de A dans C, alors on suppose que :

$$|\mu_1| > |\mu_2| \geq |\mu_3| \geq \ldots \geq |\mu_n|.$$

Dans ce cas la valeur propre μ_1 est nécessairement réelle et simple. De plus, le rayon spectral de A est donné par : $\rho(A) = |\mu_1|$.

Remarque — Il existe un théorème qui donne une condition suffisante sur les coefficients de la matrice A, pour que la valeur propre dominante soit unique :

Théorème : Si $a_{ij} > 0$ pour tous i, j alors $\rho(A) = \mu$, avec $\mu > 0$ unique valeur propre dominante de la matrice A.

4.2.2 Méthode de la puissance itérée pour calculer le rayon spectral

On note e_1 un vecteur propre, de norme 1, associé à μ_1 et on définit la suite de vecteurs (v_p) de la façon suivante :

$$\begin{cases} v_0 \text{ donné tel que } \|v_0\| = 1 \\ v_p = \dfrac{1}{\|A \cdot v_{p-1}\|} \cdot A \cdot v_{p-1} \text{ , pour } p > 0 \end{cases}$$

Lemme **:** Pour tout $p \geq 0$, on a :

$$v_p = \frac{1}{\|A^p \cdot v_0\|} \cdot A^p \cdot v_0$$

Démonstration — Pour p = 1 le résultat est vrai et en le supposant vrai pour p, on a :

$$v_{p+1} = \frac{1}{\|A \cdot v_p\|} \cdot A \cdot v_p,$$

avec :

$$A \cdot v_p = A\left(\frac{1}{\|A^p \cdot v_0\|} \cdot A^p \cdot v_0\right) = \frac{1}{\|A^p \cdot v_0\|} \cdot A^{p+1} \cdot v_0$$

d'où le résultat.

Théorème : Si v_0 n'est pas orthogonal à e_1 (i.e. si la composante de v_0 suivant e_1 dans une base de Jordan associée à A est non nulle) alors la suite (v_p) converge vers un vecteur propre associé à la valeur propre dominante μ_1 et la suite $\left(\|A \cdot v_p\|\right)$ converge vers le rayon spectral $\rho(A)$.

Démonstration — Avec le théorème de Jordan , on peut écrire que $A = P^{-1} \cdot A' \cdot P$, où :

$$A' = \begin{pmatrix} \mu_1 & 0 \ldots 0 \\ 0 & \\ . & \quad B \\ 0 & \end{pmatrix}$$

la matrice B étant triangulaire supérieure de valeurs propres $\mu_2,\ldots,\mu_n$.En écrivant $B = \mu_1 \cdot B'$, où B' a pour valeurs propres les μ_j/μ_1 pour j = 2, ..., n, on a donc :

$$A^p \cdot v_0 = \mu_1^p \cdot \begin{pmatrix} 1 & 0 \ldots 0 \\ 0 & \\ . & (B')^p \\ 0 & \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ \\ v_0' \\ \\ \end{pmatrix} = \mu_1^p \cdot \begin{pmatrix} x_1 \\ \\ u_p \\ \\ \end{pmatrix}$$

si $v_0 = \begin{pmatrix} x_1 \\ \\ v_0' \\ \\ \end{pmatrix}$, avec $u_p \to 0$. On a alors :

$$v_p = \text{Signe}(x_1).(\text{Signe}(\mu_1)^p) \cdot \begin{pmatrix} 1 \\ \\ u_p \\ \\ \end{pmatrix}.$$

Donc :

$$v_p \underset{p \to +\infty}{\longrightarrow} \left(-\text{Signe}(x_1)\right)^p \cdot e_1$$

et

$$\left\| A \cdot v_p \right\| \underset{p \to +\infty}{\longrightarrow} |\mu_1|.$$

Remarque 1 — En général on n'a pas d'informations sur le sous-espace propre associé à μ_1 et en prenant v_0 quelconque dans R^n on pourrait avoir $x_1 = 0$ et, a priori la méthode décrite ci-dessus ne donnera pas le résultat espéré. En fait $V = \{ v \in R^n ; x_1 = 0 \}$ est un hyperplan de R^n, donc de mesure nulle, et en prenant v_0 quelconque dans R^n on a très peu de chances de tomber sur V. Et même si v_0 est dans V, à cause des erreurs d'arrondis, au bout de quelques itérations on sortira de l'hyperplan et on aura les bonnes hypothèses mais la convergence peut être lente. Une autre façon d'éviter les problèmes est de prendre deux valeurs différentes de v_0 et de garder celle qui donne la convergence la plus rapide.
Remarque 2 — La méthode de la puissance itérée est une adaptation de la méthode de Bernouilli pour calculer la racine de plus grand module d'un polynôme, quand cette dernière est unique.
Remarque 3 — En appliquant la méthode de la puissance itérée à la matrice A^{-1}, on a un moyen de calculer la valeur propre de plus petit module de A (quand cette dernière est unique) car $\rho(A^{-1}) = \dfrac{1}{|\mu_n|}$ où μ_n est la valeur propre de plus petit module de A. La méthode obtenue est aussi appelée méthode de la puissance inverse.
Remarque 4 — Quand la méthode converge, pour p grand, v_p peut être considéré comme un vecteur propre de A associé à μ_1, de sorte que $A \cdot v_p \approx \mu_1 \cdot v_p$, ce qui permet de calculer μ_1 avec :

$$\mu_1 \approx \frac{(A \cdot v_p)(i)}{v_p(i)}, \text{ où i est tel que } |v_p(i)| = \|v_p\|.$$

4.3 Calcul des autres valeurs propres par la méthode de déflation

4.3.1 Hypothèses et notations

On suppose maintenant que les valeurs propres de A vérifient :

$$|\mu_1| > |\mu_2| > |\mu_3| > \ldots > |\mu_n|.$$

Soit X un vecteur propre associé à μ_1 de norme euclidienne égale à 1.

On va alors vérifier que les valeurs propres de la matrice

$$B = A - \mu_1 \cdot X \cdot {}^tX = ((a_{ij} - \mu_1 \cdot x_i \cdot x_j))$$

sont 0, μ_2, ..., μ_n. On appliquera alors la méthode de la puissance itérée à cette nouvelle matrice pour avoir μ_2, puis on continue ainsi de suite.

4.3.2 Un lemme

Lemme : Soit α une valeur propre de A différente de μ_1, alors
(i) α est aussi valeur propre de ${}^tA$;
(ii) si Y est un vecteur propre de ${}^tA$ associé à α alors Y est orthogonal à X, c'est-à-dire $\langle X | Y \rangle = 0$;
(iii) Y est aussi valeur propre de $B = A - \mu_1 \cdot X \cdot {}^tX$;
(iv) 0 est valeur propre de B.

Démonstration — Il suffit de vérifier.

Remarque — Pour calculer la deuxième valeur propre de B on peut se passer de calculer la matrice B.

Le procédé de calcul est le suivant :

$$u_p = B \cdot v_{p-1} ;$$

puis

$$v_p = \frac{1}{\|u_p\|} \cdot u_p .$$

Mais on a :

$$u_p = A \cdot v_{p-1} - \mu_1 \cdot X \cdot {}^tX \cdot v_{p-1} = A \cdot (v_{p-1} - \langle X | v_{p-1} \rangle \cdot X) ;$$

car $\mu_1 \cdot X = A \cdot X$.

L'algorithme de calcul est donc :

calcul de $\beta_p = \langle X | v_{p-1} \rangle$;

calcul de $u_p = A \cdot (v_{p-1} - \beta_p \cdot X)$;

calcul de $v_p = \frac{1}{\|u_p\|} \cdot u_p$.

Mais pour calculer toutes les valeurs propres il est préférable d'utiliser une procédure de calcul de la matrice B.

4.3.3 Programmation structurée

Les valeurs propres peuvent être stockées dans un vecteur Valeurs_Propres et les vecteurs propres dans une matrice Vecteurs_Propres.

La procédure ci-dessous calcule les p premières valeurs propres et les vecteurs propres associés pour p = 1, ..., n. Les valeurs propres sont calculées dans l'ordre des valeurs absolues décroissantes, ce qui suppose qu'elles sont toutes distinctes.

```
PROCEDURE Déflation(Entrée n, p : Entier ; A : Matrice ;
                    Sortie Valeurs_Propres : Vecteur ; Vecteurs_Propres : Matrice) ;

Début
  Pour k allant de 1 à p faire
  Début
    Se donner Vk aléatoire ;
    Iter = 0 ;
    Répéter
      Iter = Iter + 1 ;
      v = A.Vk ;
      i0 = 1 ;
      Pour i allant de 2 à n faire
      Début
        Si |Vk_i| > |Vk_i0|
        Alors i0 = i ;
      Fin ;
      Valeurs_Propres(k) = v_i0/Vk_i0;
      Aux = NormeEuclidienne(v) ;
      v = (1/Aux)·v ;
      uAncien = Vk ;
      Vk = v ;
    Jusqu'à (||Vk − uAncien|| < ε) ou (Iter = MaxIter) ;
     Pour i allant de 1 à n faire Vecteurs_Propres_ik = v_i ;
    Si Iter = MaxIter
    Alors Arrêter('Convergence trop lente ou divergence') ;
    Pour i allant de 1 à n faire
    Début
      Pour j allant de 1 à n faire
      Début
        a_ij = a_ij – Valeurs_Propres(k)*v_i*v_j ;
      Fin ;
    Fin ;
  Fin ;
Fin ;
```

4.4 Méthode de Rutishauser

4.4.1 Un lemme

Lemme : Si A admet une décomposition L·R, alors la matrice $A_1 = R \cdot L$ est semblable à A, donc ces deux matrices ont les mêmes valeurs propres.

Démonstration — On a : $R \cdot L = R \cdot (L \cdot R) \cdot R^{-1}$.

4.4.2 Hypothèses

On suppose que la matrice A admet une décomposition L·R et que toutes ses valeurs propres sont réelles. Un exemple de telle matrice est donné par A symétrique définie positive.

4.4.3 Principe de la méthode de Rutishauser

Etape 1 — A partir de la décomposition $A = L_1 \cdot R_1$, on pose $A_1 = R_1 \cdot L_1$;
Etape 2 — on écrit la décomposition L·R de A_1 , soit $A_1 = L_2 \cdot R_2$, puis on pose $A_2 = R_2 \cdot L_2$;
Etape p — ayant obtenu $A_{p-1} = R_{p-1} \cdot L_{p-1}$, on écrit sa décomposition L·R, soit $A_{p-1} = L_p \cdot R_p$, puis on pose $A_p = R_p \cdot L_p$.

Si à chaque étape la décomposition L·R est possible, alors A_p est semblable à A.

Sous certaines conditions, la suite de matrices ainsi obtenue va converger vers une matrice A' qui est triangulaire supérieure et semblable à A. Donc les valeurs propres de A seront les termes diagonaux de A'.

4.4.4 Une condition suffisante de convergence de la méthode de Rutishauser

Théorème : Si la matrice A est diagonalisable avec ses valeurs propres réelles et si les matrices A_p admettent toutes des décompositions L·R, alors la suite (A_p) va converger vers une matrice triangulaire supérieure A' dont les termes diagonaux sont les valeurs propres de A.

Démonstration — Voir Stoer et Burlisch Chap. 6.

Remarque 1 — Si on ajoute $\alpha \cdot I_d$ à la matrice A, du point de vue spectral, cela revient à ajouter α aux valeurs propres de A.

Donc, dans le cas où des pivots trop petits apparaissent à l'étape p, avant de factoriser la matrice A_p on lui ajoute $\alpha_p \cdot I_d$ avec α_p choisi tel que $A_p + \alpha \cdot I_d$ soit à diagonale strictement dominante. Ensuite on effectue la décomposition L·R. A la fin il ne faudra pas oublier de retrancher des valeurs propres obtenues, la somme de tous les α_p qu'il a fallu ajouter.

Remarque 2 — Comme test d'arrêt on pourra choisir de s'arrêter pour p < MaxIter et quand :

$$\sum_{i=1}^{n} \left| a_{ii}^{(p+1)} - a_{ii}^{(p)} \right| < \varepsilon \quad \text{et} \quad \sum_{1 \le j < i \le n} \left| a_{ij}^{(p+1)} \right| < \varepsilon .$$

La programmation structurée s'écrit alors :

```
PROCEDURE Rutishauser(Entrée n : Entier ; A : Matrice ; Sortie Valeurs_Propres : Vecteur) ;
Début
   Iter = 0 ;
   Répéter
      L-R(n,A,L,R) ;
```

Pour i Allant de 1 à n Faire Valeurs_Propres(i) = a_{ii} ;
A = R·L ; c = 0 ; d = 0 ;
Pour i Allant de 1 à n Faire
Début
$c = c + \left| a_{ii} - Valeurs\_Propres(i) \right|$;
Pour j Allant de 1 à i – 1 Faire $d = d + \left| a_{ij} \right|$;
Fin ;
Iter = Iter + 1 ;
Jusqu'à (c < ε) et (d < ε) ou (Iter = MaxIter) ;
Fin ;

4.5 Méthode de Jacobi pour calculer les valeurs et vecteurs propres d'une matrice symétrique

On se donne, dans ce paragraphe, une matrice A symétrique réelle. Ses valeurs propres sont donc toutes réelles.

Le principe de la méthode de Jacobi consiste à construire une suite de matrices de rotations planes (R_k) telle que pour k tendant vers l'infini, la suite de matrices (A_k) définie par $A_0 = A$ et $A_k = {}^tR_k \cdot A_{k-1} \cdot R_k$ tend vers une matrice diagonale. Comme chacune des matrices A_k est semblable à A (puisque ${}^tR_k = R_k^{-1}$), on en déduit que les valeurs propres de A sont les termes diagonaux de la matrice limite.

Cette méthode est bien adaptée aux matrices symétriques de taille au plus 10. Pour les matrices de taille supérieure, on lui préférera la méthode Q·R (Cf. Exercice 9).

Pour tous θ dans [0,2·π[et $1 \le p < q \le n$, on note $R_{p,q}(\theta)$ la matrice de rotation d'angle θ dans le plan défini par les vecteurs e_p et e_q de la base canonique de R^n. Elle est donc définie par :

$$R_{p,q}(\theta) = \begin{pmatrix} 1 & 0 & . & . & . & . & 0 \\ . & . & . & . & . & . & \\ . & . & c. & . & -s & & \\ . & . & . & 1 & . & & \\ . & . & s & . & c & & \\ . & . & . & . & . & . & \\ 0 & . & . & . & . & 0 & 1 \end{pmatrix} \begin{matrix} \\ \\ \leftarrow p \\ \\ \leftarrow q \\ \\ \\ \end{matrix}$$

$$\begin{matrix} & & \uparrow & & \uparrow & & \\ & & p & & q & & \end{matrix}$$

où c = Cos(θ) et s = Sin(θ).

Pour toute matrice symétrique A, le produit $A' = R_{p,q}^{-1}(\theta) \cdot A \cdot R_{p,q}(\theta)$ est alors facile à calculer. La multiplication à gauche par $R^{-1} = {}^tR$ modifie seulement les

lignes p et q de A et la multiplication à droite par R change seulement les colonnes p et q de A. Ce qui donne pour les coefficients de A' :

$$\begin{aligned}
a'_{ij} &= a_{ij} \quad (i \neq p, i \neq q, j \neq p, j \neq q)\\
a'_{ip} &= c\cdot a_{ip} + s\cdot a_{iq} \ (i \neq p, i \neq q)\\
a'_{pp} &= c^2\cdot a_{pp} + s^2\cdot a_{qq} + 2\cdot s\cdot c\cdot a_{pq}\\
a'_{iq} &= c\cdot a_{iq} - s\cdot a_{ip} \ (i \neq p, i \neq q)\\
a'_{qq} &= s^2\cdot a_{pp} + c^2\cdot a_{qq} - 2\cdot s\cdot c\cdot a_{pq}\\
a'_{pq} &= (c^2 - s^2)\cdot a_{pq} - s\cdot c\cdot (a_{pp} - a_{qq})
\end{aligned}$$

la matrice A' étant symétrique.

L'idée de Jacobi est alors de déterminer θ de manière à annuler les coefficients a'_{pq} et a'_{qp} de A', ce qui donne pour l'angle θ :

$$2\cdot\cos(2\cdot\theta)\cdot a_{pq} - \sin(2\cdot\theta)\cdot(a_{pp} - a_{qq}) = 0$$

pour $a_{pq} \neq 0$, on a nécessairement $\sin(2\cdot\theta) \neq 0$ et :

$$\mathrm{Cotg}(2\cdot\theta) = \frac{a_{pp} - a_{qq}}{2\cdot a_{pq}} = b$$

(si $a_{pq} = 0$, le problème ne se pose pas).

Le calcul de c = Cos(θ) et s = Sin(θ) peut se faire de façon purement algébrique, ce qui est plus rapide que de faire appel aux fonctions trigonométriques.

En posant t = tg(θ), on a :

$$b = \frac{1}{2}\cdot\left(\frac{1}{\mathrm{tg}(\theta)} - \mathrm{tg}(\theta)\right)$$

c'est-à-dire que t est solution de l'équation du second degré :

$$t^2 + 2\cdot b\cdot t - 1 = 0$$

Cette équation admet deux racines réelles :

$$t = -b + \sqrt{1 + b^2} = \frac{1}{b + \sqrt{1 + b^2}} \text{ et } t' = -\frac{1}{t} = \frac{-1}{\sqrt{1 + b^2} - b}$$

Comme $|t\cdot t'| = 1$, l'une de ces racines est de valeur absolue inférieure ou égale à 1. Pour minimiser les problèmes d'erreurs d'arrondis, on choisit la racine de valeur absolue inférieure à 1, ce qui donne un angle θ entre $-\frac{\pi}{4}$ et $\frac{\pi}{4}$. Cette racine est donnée par :

$$t = \frac{\mathrm{signe}(b)}{\sqrt{b^2 + 1} + |b|}$$

Avec $\cos(\theta) = \frac{1}{\sqrt{1 + \mathrm{tg}^2(\theta)}}$ et $\sin(\theta) = \cos(\theta)\cdot\mathrm{tg}(\theta)$, on déduit que :

$$c = \frac{1}{\sqrt{1 + t^2}} \text{ et } s = t\cdot c$$

Les formules donnant les coefficients de A' se simplifient alors pour donner finalement :

$$a'_{pq} = 0$$

$$a'_{pp} = a_{pp} + t\cdot a_{pq}$$
$$a'_{qq} = a_{qq} - t\cdot a_{pq}$$
$$a'_{ip} = a_{ip} + s\cdot(a_{iq} - \tau\cdot a_{ip}) \quad (i \neq p, i \neq q)$$
$$a'_{iq} = a_{iq} - s\cdot(a_{ip} + \tau\cdot a_{iq}) \quad (i \neq p, i \neq q)$$

où τ est donné par :

$$\tau = \mathrm{tg}\left(\frac{\theta}{2}\right) = \frac{s}{1+c}$$

La méthode de Jacobi classique consiste à partir de la matrice A à construire la matrice $A^{(1)}$ par le procédé décrit ci-dessus où les indices p et q sont définis par :

$$\left|a_{pq}\right| = \mathrm{Max}\left\{\left|a_{ij}\right|;\ 1 \le i < j \le n\right\}$$

puis on recommence avec $A^{(1)}$ pour construire $A^{(2)}$ et en continuant ainsi de suite on construit une suite de matrices $A^{(k)}$ toutes semblables à A.

On peut alors montrer le :

Théorème : Pour toute matrice symétrique réelle, la suite $(A^{(k)})$ converge vers une matrice D diagonale et semblable à A.
Les valeurs propres de A sont donc les éléments diagonaux de D.
De plus, si on écrit $D = {}^tV\cdot A\cdot V$, alors pour tout j = 1, ..., n, la colonne j de V est un vecteur propre associé à la valeur propre D_{jj}.

Démonstration — Voir Lascaux et Théodor p. 697.

Remarque 1 — D'un point de vue numérique, le calcul du maximum des termes non diagonaux de $A^{(k)}$ à chaque étape n'est pas intéressant car il augmente le temps de calcul. On préfère procéder de la manière suivante : à l'étape k du calcul, la matrice $A^{(k-1)}$ étant construite on construit la matrice $A^{(k)}$ en effectuant $\frac{n\cdot(n-1)}{2}$ transformations de Jacobi en prenant pour valeurs successives de (p,q), les valeurs (1,2), (1,3), ..., (1,n) ; (2,3), ..., (2,n) ; ... ; (n – 1,n). Un tel calcul est appelé un « balayage ».

En notant :

$$S_k = \sum_{1\le i<j\le n}\left|a_{ij}\right|$$

on arrête les itérations quand $S < \varepsilon$, où ε est une précision donnée.

On peut alors montrer que la convergence devient très rapidement quadratique.

Remarque 2 — En pratique on n'effectuera pas de transformation de Jacobi si $\left|a_{pq}\right| < \sigma$, où σ est un seuil donné. Usuellement, on prend pour valeur de σ : $\sigma = \frac{S_k}{5\cdot n^2}$ seulement pour les trois premiers balayages, puis à partir du quatrième, on décide $a_{pq} = 0$ si $\left|a_{pq}\right| < \varepsilon'\cdot\left|a_{pp}\right|$ et $\left|a_{pq}\right| < \varepsilon'\cdot\left|a_{qq}\right|$ avec ε' assez petit.

Remarque 3 — La matrice V, donnant les vecteurs propres de A, est donnée par :

$$V = R_1 \cdot R_2 \cdot ....$$

où les R_i sont les rotations successives. Cette matrice est donc définie par la récurrence :

$$V' = V \cdot R_i$$

si $A' = {}^tR_1 \cdot A \cdot R_1$ est une étape de calcul. Ce qui donne les relations :

$$v'_{ij} = v_{ij} \ (j \neq p, j \neq q)$$
$$v'_{ip} = v_{ip} + s \cdot (v_{iq} - \tau \cdot v_{ip}) \ (i = 1, ..., n)$$
$$v'_{iq} = v_{iq} - s \cdot (v_{ip} + \tau \cdot v_{iq}) \ (i = 1, ..., n)$$

Ce qui nous donne en définitive la programmation structurée suivante, où D est un vecteur qui contiendra les valeurs propres et V une matrice dont les colonnes seront des vecteurs propres associés.

```
PROCEDURE Jacobi(Entrée n : Entier ; A : Matrice ; Sorties D : Vecteur ; V : Matrice) ;
Début
   V = Id ; Correct = Faux ; Iteration = 0 ; D = 0 ;
   Pour i Allant de 1 à n Faire d_i = a_ii ;
   Répéter
      Iteration = Iteration + 1 ;
      Sk = O ;
      Pour i allant de 1 à n – 1 Faire
      Début
         Pour j allant de i + 1 à n Faire
         Début
            Sk = Sk + |a_ij| ;
         Fin ;
      Fin ;
      Correct = (Sk < ε) Ou (Iteration = MaxIteration) ;
      Si Iteration < 4
      Alors Seuil = Sk/(5·n^2)
      Sinon Seuil = ε^2 ;
      Pour p allant de 1 à n – 1 Faire
      Début
         Pour q allant de p + 1 à n Faire
         Début
            Si (Iteration ≥ 4) Et (10^8·|a_pq| < |d_p|) Et (10^8·|a_pq| < |d_q|)
            Alors a_pq = O
            Sinon Si |a_pq| > Seuil
               Alors Début
                  b = 0.5·(d_p – d_q)/a_pq ;
                  t = 1 / (sqrt(b^2 + 1) + |b|) ;
                  Si b < 0 Alors t = –t ;
                  c = 1 / sqrt(1 + t^2) ;
                  s = t·c ;
```

```
                        τ = s/(1 + c) ;
                        Aux = t·a_pq ;
                        d_p = d_p + Aux ;
                        d_q = d_q – Aux ;
                        a_pq = 0 ;
                        Pour j Allant de 1 à p – 1 Faire
                        Début
                            Aux = a_jp ;
                            a_jp = a_jp + s·(a_jq – τ·a_jp) ;
                            a_jq = a_jq – s·(Aux + τ·a_jq) ;
                        Fin ;
                        Pour j Allant de p + 1 à q – 1 Faire
                        Début
                            Aux = a_pj ;
                            a_pj = a_pj + s·(a_jq – τ·a_pj) ;
                            a_jq = a_jq – s·(Aux + τ·a_jq) ;
                        Fin ;
                        Pour j Allant de q + 1 à n Faire
                        Début
                            Aux = a_pj  ;
                            a_pj  = a_pj + s·(a_qj – τ·a_pj) ;
                            a_qj  = a_qj – s·(Aux + τ·a_qj) ;
                        Fin ;
                        Pour j Allant de 1 à n Faire
                        Début
                            Aux = v_jp ;
                            v_jp = v_jp + s·(v_jq – τ·v_jp) ;
                            v_jq = v_jq – s·(Aux + τ·v_jq) ;
                        Fin ;
                    Fin ;
                Fin ;
            Fin ;
        Jusqu'à Correct ;
        Si Iteration = MaxIteration
        Alors Arréter('Nombre d'itérations trop grand') ;
    Fin ;
```

4.6 Méthodes de calcul du polynôme caractéristique

Ces méthodes peuvent être utilisées pour des matrices de petite taille en relation avec des méthodes de résolution d'équations algébriques. On pourra, par exemple, utiliser la méthode de Bairstow pour chercher les racines du polynôme caractéristique.

4.6.1 Méthode de Souriau

Le polynôme caractéristique de A est noté ici :

$$P(x) = Dét(x \cdot I_d - A) = x^n + p_1 \cdot x^{n-1} + ... + p_{n-1} \cdot x + p_n$$

avec les coefficients p_k dans R inconnus a priori.

On note Tr(A) la trace de la matrice A, c'est-à-dire la somme de tous les termes diagonaux.

On définit les suites (A_p), (t_p) et (B_p) de la façon suivante :

$$\begin{cases} A_1 = A & t_1 = -\mathrm{Tr}(A_1) & B_1 = A_1 + t_1 \cdot I_d \\ A_2 = B_1 \cdot A & t_2 = -\dfrac{\mathrm{Tr}(A_2)}{2} & B_2 = A_2 + t_2 \cdot I_d \\ . & . & . \\ A_n = B_{n-1} \cdot A & t_n = -\dfrac{\mathrm{Tr}(A_n)}{n} & B_n = A_n + t_n \cdot I_d \end{cases}$$

Si $\lambda_1, \lambda_2, ..., \lambda_n$ sont les valeurs propres, dans C, de la matrice A, en notant :

$$S_p = \sum_{i=1}^{n} \lambda_i^p \quad (p \geq 1)$$

on a alors les *formules de Newton* :

$$(1) S_k + p_1 \cdot S_{k-1} + ... + p_{k-1} \cdot S_1 + p_k \cdot k = 0 \quad (1 \leq k \leq n)$$

D'où le résultat suivant qui permet de calculer très simplement le polynôme caractéristique de A :

Théorème : Les coefficients t_k, k = 1, 2, ..., n, définis ci-dessus sont les coefficients du polynôme caractéristique de A.

Démonstration — On procède par récurrence sur k.

Pour k = 1 :

$$t_1 = -\mathrm{Tr}(A_1) = -\sum_{i=1}^{n} \lambda_i = p_1$$

Supposons le résultat vrai pour j = 1, ..., k – 1, avec k > 1 : par définition, on a :

$$A_k = B_{k-1} \cdot A = A_{k-1} \cdot A + t_{k-1} \cdot A$$

et par récurrence, il vient :

$$A_k = A^k + t_1 \cdot A^{k-1} + ... + t_{k-1} \cdot A$$

donc :

$$(1) A_k = A^k + p_1 \cdot A^{k-1} + ... + p_{k-1} \cdot A \quad (k = 2, ..., n)$$

On déduit alors que :

$$\mathrm{Tr}(A_k) = \mathrm{Tr}(A^k) + p_1 \cdot \mathrm{Tr}(A^{k-1}) + ... + t_{k-1} \cdot \mathrm{Tr}(A)$$

C'est-à-dire :

$$t_k = -\frac{1}{k} \cdot (S_k + p_1 \cdot S_{k-1} + ... + p_{k-1} \cdot S_1)$$

Ce qui s'écrit, en tenant compte des formules de Newton :

$$t_k = p_k$$

Remarque : Calcul de l'inverse de A — En faisant k = n dans l'identité (1) ci-dessus, on déduit que :

$$B_n = A_n + t_n \cdot I_d = P(A) = 0$$

d'après le théorème de Cayley-Hamilton.

Ce qui permet d'en déduire l'inverse de A :

$$A^{-1} = \frac{-1}{t_n} \cdot B_{n-1}.$$

Ce qui donne, pour la programmation structurée, en définissant un type polynôme comme un tableau de réels numérotés de 0 à DimMax, où DimMax est la dimension maximale des matrices :

```
PROCEDURE Souriau(Entrée n : Entier ; A : Matrice ;
                  Sortie t : Polynome ; Inverse : Matrice) ;
Début
   Pour i Allant de 1 à n Faire
   Début
      Pour j Allant de 1 à n Faire B_ij = 0 ;
      B_ii = 1 ;
   Fin ;
   Pour i  Allant de  1 à n Faire
   Début
       B = B·A ;
       t_i = –Trace(B)/i ;
      Pour j Allant de 1 à n Faire B_jj := B_jj + t_i ;
       Si i = n – 1  Alors Inverse = B ;
   Fin ;
   Pour i Allant de 1 à n Faire
    Pour j Allant de 1 à n Faire Inverse_ij := –Inverse_ij/t_n ;
Fin ;
```

4.6.2 Méthode de Krylov

Avec les notations de (4.6.1), en notant, pour tout vecteur u de R^n : $u_k = A^k \cdot u$ ($k \in N$) on déduit du théorème de Cayley-Hamilton que les coefficients p_i sont solutions du système linéaire :

$$p_1 \cdot u_{n-1} + p_2 \cdot u_{n-2} + ... + p_n \cdot u = -u_n$$

En prenant u au hasard, on a toutes les chances que le système ci-dessus soit non dégénéré de sorte que les coefficients du polynôme caractéristique s'obtiennent comme solution d'un système linéaire que l'on peut résoudre par la méthode de Gauss-Jordan (par exemple).

La programmation structurée étant alors évidente.

On notera « Krylov(Entrée n : Entier ; A : Matrice ; Sortie P : Vecteur) ; » une telle procédure de calcul.

De plus, toujours avec le théorème de Cayley-Hamilton, on déduit que l'inverse de A est donné par :

$$A^{-1} = -\frac{1}{p_n} \cdot \left(A^{n-1} + p_1 \cdot A^{n-2} + ... + p_{n-1} \cdot I_d \right)$$

Ce qui donne, en s'inspirant de l'algorithme de Horner pour l'évaluation d'un polynôme, la procédure suivante de calcul de l'inverse :

```
PROCEDURE InversionKrylov(Entrée n : Entier ; A : Matrice ; Sortie Inverse : Matrice) ;
Début
   Krylov(n,A,P) ;
   Inv = A + p_1·I_d ;
   Pour i Allant de 2 à n – 1 Faire Inverse = A·Inverse + p_i·I_d ;
   Inverse = -(1/p_n) · Inverse ;
```

Fin ;

4.6.3 Méthode de Leverrier

On conserve toujours les notations de (4.6.3).

En utilisant les formules de Newton, les coefficients p_k apparaissent comme solutions d'un système triangulaire inférieur.

La matrice de ce système se calcule facilement avec les $S_k = Tr(A^k)$ (Cf Exercice 5).

5. Exercices

5.1 Etude d'un système tridiagonal

On se donne un entier $k \geq 1$ et un réel a tel que $|a| > 2$.

Si $n = 2^k - 1$, on considère le système linéaire de n équations à n inconnues :

$$A \cdot x = b \qquad (1)$$

avec :

$$\begin{cases} a_{ii} = a \ (i = 1, \ldots, n) \\ a_{i,i-1} = a_{i,i+1} = 1 \ (i = 2, \ldots, n - 1) \\ a_{ij} = 0 \ \text{dans les autres cas} \end{cases}$$

Un tel système s'écrit donc :

$$x_{i-1} + a \cdot x_i + x_{i+1} = b_i \ (i = 1, \ldots, n)$$

avec la convention : $x_0 = x_{n+1} = 0$.

On va étudier une méthode de résolution qui va consister à diviser n par 2 à chaque étape.

1°) Cas particulier k = 2.

(a) Ecrire le système obtenu.

(b) Comment calculer x_1 et x_3 connaissant x_2?

(c) Montrer que x_2 est solution d'un système 1x1 : $a^{(1)} \cdot x_2 = b_2^{(1)}$.

2°) Cas particulier k = 4, $b_i = 1$ (i = 1, ..., 15). On note $A^{(0)} = A$, $b^{(0)} = b$.

(a) Comment déterminer les x_{2p+1} connaissant les x_{2j} , pour p = 1, ..., 7.

(b) On pose $y_p = x_{2p}$, pour p = 1, ..., 7. Et y est le vecteur de R^7 de composantes y_p.

Montrer que y est solution d'un système tridiagonal : $A^{(1)} \cdot y = b^{(1)}$.

(c) On pose $z_q = y_{2q}$, pour q = 1, 2, 3. Et z est le vecteur de R^3 de composantes z_q .

Montrer que z est solution d'un système tridiagonal : $A^{(2)} \cdot y = b^{(2)}$.

On suppose de plus que a = 4.

(d) Calculer w_1, z_2, y_4, x_8.

(e) En déduire les valeurs de z_1 et z_3.

(f) En déduire celles de y_1, y_3, y_5 et y_7.

(g) Donner la solution du système (1).

3°) En s'inspirant du 2°) écrire les procédures permettant de résoudre (1) dans le cas général.

5.2 Valeurs propres d'une matrice tridiagonale

Soient a, b dans R. Trouver les valeurs propres et les vecteurs propres de la matrice :

$$A = \begin{pmatrix} a & b & 0 & . & . & 0 \\ b & a & b & 0 & . & 0 \\ . & . & . & . & . & . \\ 0 & . & 0 & b & a & b \\ 0 & . & . & 0 & b & a \end{pmatrix}$$

Cas particulier : a = 2 et b = -1.

5.3 Matrices de Vandermonde

Si $t_1, t_2, ..., t_n$ sont des réels, on note :

$$V(t_1, t_2, ..., t_n) = \left(\left(a_{ij}\right)\right)_{1\le i,j\le n} \text{ , avec } a_{ij} = t_i^{j-1}$$

une matrice de Vandermonde.

On veut résoudre le système $A\cdot x = b$, où A est de Vandermonde. Ce problème étant lié à l'interpolation polynomiale, on introduit les notations suivantes :

$$\begin{cases} P_i(t) \text{ est le polynôme de degré n - 1 dans } R[t] \text{ défini par } P_i(t_j) = \delta_{ij} \\ P_i(t) = \sum_{i=1}^{n} c_{ij} \cdot t^{j-1} \end{cases}$$

1°) Calculer le déterminant d'une matrice de Vandermonde.

2°) Montrer que, si les t_i sont deux à deux distincts, alors :

$$V(t_1, ..., t_n)^{-1} = {}^t((c_{ij}))$$

3°) En déduire une procédure de résolution d'un système de Vandermonde.

5.4 Polynôme caractéristique d'une matrice tridiagonale

Soit $A = \begin{pmatrix} d_1 & e_1 & 0 & . & . & 0 \\ c_2 & d_2 & e_2 & 0 & . & 0 \\ . & . & . & . & . & . \\ 0 & . & 0 & c_{n-1} & d_{n-1} & e_{n-1} \\ 0 & . & . & 0 & c_n & d_n \end{pmatrix}$ une matrice tridiagonale de polynôme caractéristique :

$$P(t) = \alpha_1\cdot t^n + \alpha_2\cdot t^{n-1} + ... + \alpha_n\cdot t + \alpha_{n+1}$$

Pour tout k = 1, 2, ... , n – 1, on pose $b_k = c_{k+1}\cdot e_k$, on note $A^{(k)}$ la $k^{\text{ième}}$ sous-matrice principale de A et son polynôme caractéristique est noté :

$$P_k(t) = \alpha_1^{(k)}\cdot t^k + ... + \alpha_{k+1}^{(k)}.$$

1°) Trouver une relation de récurrence sur les polynômes P_k.

2°) En déduire un algorithme qui permet de calculer les coefficients de P.

5.5 Méthode de Levérrier et calcul du rayon spectral d'une matrice

Soit A une matrice à n lignes et n colonnes à coefficients réels de valeurs propres $\lambda_1, \lambda_2, ..., \lambda_n$ dans C. On note :

$$S_p = \sum_{i=1}^{n} \lambda_i^p \quad (p \geq 1)$$

et :

$$P(x) = Dét(A - x \cdot I_d) = p_0 \cdot x^n + p_1 \cdot x^{n-1} + ... + p_{n-1} \cdot x + p_n$$

le polynôme caractéristique de A, avec $p_0 = (-1)^n$.

On a alors les *formules de Newton* :

$$(1) \quad p_0 \cdot S_k + p_1 \cdot S_{k-1} + ... + p_{k-1} \cdot S_1 + p_k \cdot k = 0 \quad (1 \leq k \leq n)$$

1°) Déduire avec (1), une procédure de calcul des coefficients p_k.

2°) On suppose que A admet des valeurs propres toutes distinctes dans R, avec :

$$0 < |\lambda_1| < |\lambda_2| < ... < |\lambda_n|.$$

(a) Calculer $\lim_{k \to +\infty} \left(\frac{S_k}{S_{k-1}} \right)$.

(b) Calculer $\lim_{k \to +\infty} \left(\sqrt[k]{|S_k|} \right)$.

3°) Montrer que pour $k \geq n$:

$$p_0 \cdot S_k + p_1 \cdot S_{k-1} + ... + p_n \cdot S_{k-n} = 0$$

4°) Donner une procédure de calcul de (S_k) pour k dans N et en déduire une procédure de calcul de λ_n.

5.6 Tridiagonalisation d'une matrice symétrique.

On se donne une matrice symétrique A.

On pose $A_1 = A$ et on suppose qu'on a construit la matrice A_k, pour k = 1, ..., n – 1, de la forme :

$$A_k = \begin{pmatrix} * & * & & & & & & \\ * & * & * & & & & & \\ . & . & . & . & . & . & . & \\ & & * & * & . & . & * & \leftarrow \text{Ligne k} \\ & & & * & . & . & . & \\ & & & . & . & . & . & \\ & & & * & . & . & * & \end{pmatrix}$$

$\uparrow$

colonne k

où les éléments non nuls (a priori) sont repérés par *.

Montrer qu'il existe un réel r, de même signe que $a_{k,k+1}^{(k)}$ tel que si v_k est le vecteur de coordonnées $\left(0, \quad ..., \quad 0, \quad r + a_{k,k+1}^{(k)}, \quad a_{k,k+2}^{(k)}, \quad . \quad . \quad ., \quad a_{k,n}^{(k)}\right)$ et H_k la matrice de Householder définie par :

$$H_k = I_d - \frac{2}{\|v_k\|^2} \cdot v_k \cdot {}^t v_k$$

alors la matrice $A_{k+1} = {}^tH_k \cdot A_k \cdot H_k$ vérifie bien la condition voulue.

En déduire que A est semblable à une matrice tridiagonale.

Ecrire une procédure de tridiagonalisation d'une matrice symétrique, puis une procédure de calcul du polynôme caractéristique.

Proposer une méthode de recherche des valeurs propres d'une matrice symétrique.

5.7 Méthode de Givens et Householder pour les matrices tridiagonales

Soit $A = \begin{pmatrix} a_1 & b_1 & 0 & . & . & 0 \\ b_1 & a_2 & b_2 & 0 & . & 0 \\ . & . & . & . & . & . \\ 0 & . & 0 & b_{n-2} & a_{n-1} & b_{n-1} \\ 0 & . & . & 0 & b_{n-1} & a_n \end{pmatrix}$ une matrice tridiagonale et symétrique. Les valeurs propres de A sont alors réelles et notées :

$$\lambda_1 \le \lambda_2 \le \ldots \le \lambda_n$$

On suppose, en outre, que les b_i sont tous non nuls (sinon on découpe A en deux blocs et on travaille séparément sur chacun d'eux).

On construit la suite de polynômes $(P_k)_{0 \le k \le n}$ de la manière suivante :

$$\begin{cases} P_0 = 1; \; P_1 = a_1 - x; \\ P_k = (a_k - x) \cdot P_{k-1} - (b_{k-1})^2 \cdot P_{k-2} \quad (k = 2, \ldots, n) \end{cases}$$

1°) Montrer que la suite (P_k) vérifie les propriétés suivantes :

(a) P_k est de degré égal à k ;

(b) Pour k = 1, ..., n :

$$\lim_{k \to -\infty} P_k(x) = +\infty$$

(c) P_k possède k racines réelles distinctes séparées par les k – 1 racines de P_{k-1} ;

(d) P_k est le polynôme caractéristique de la matrice A_k déduite de A en supprimant les n – k dernières lignes et colonnes.

2°) On note, pour tout x réel, $s_k(x) = \pm 1$ = le signe de $P_k(x)$ (avec la convention $s_k(x) = s_{k-1}(x)$, si $P_k(x) = 0$). N(x) désigne le nombre de changements de signes entre deux termes consécutifs de la suite $(s_0(x), s_1(x), \ldots, s_n(x))$.

(a) Montrer que N(x) est égal au nombre de racines de P_n qui sont inférieures à x.

(b) Ecrire une procédure de calcul de N(x).

3°) Toutes les valeurs propres de A sont dans l'intervalle [–M,M], où

$$M = \text{Max}\left\{ \sum_{i=1}^{n} |a_{ij}| \; ; \; 1 \le j \le n \right\}$$

(a) Ecrire une procédure de calcul de M.

(b) Ecrire une procédure de calcul de la $k^{ème}$ valeur propre de A, pour tout k = 1, 2, ..., n.

5.8 Factorisation Q·R

Si v est un vecteur non nul, on définit la *matrice de Householder* par :

$$H(v) = I_d - \frac{2}{{}^t v \cdot v} \cdot v \cdot {}^t v$$

1°) Calculer le déterminant et l'inverse de H(v).

2°) Soit a un vecteur de R^n non colinéaire à e_1 (premier vecteur de base). Montrer qu'il existe un unique vecteur v tel que :

(i) $v = a + \lambda \cdot e_1$;

(ii) $H(v) \cdot a = \mu \cdot e_1$;

(iii) $\|v\| \geq \sqrt{2} \cdot \|a\|$

Ce vecteur sera noté $v = J_n(a)$. Si a est proportionnel à e_1, on pose $J_n(a) = 0$.

3°) On construit la suite de matrices (A_k), de la manière suivante :

$A_1 = A$;

Pour k = 1, ..., n, on note $A_k = \left(\left(a_{ij}^{(k)}\right)\right)_{1 \leq i,j \leq n}$ et a'_k est le vecteur de R^{n-k+1} de composantes $a_{ik}^{(k)}$ avec i = k, ..., n.

On pose alors $v'_k = J_{n-k+1}(a'_k)$ et v_k est le vecteur de R^n dont les k – 1 premières composantes sont nulles, les autres étant égales à celles de v'_k.

Enfin $H_k = H(v_k)$.

On pose alors $H_{k+1} = H_k \cdot A_k$, pour k = 1, ..., n – 1.

(a) Montrer que A_n est triangulaire supérieure.

(b) Montrer qu'il existe une matrice diagonale D telle que si on pose :

$$Q = H_1 \cdot \ldots \cdot H_{n-1} \cdot D \text{ et } R = D \cdot A_n$$

alors Q est une matrice orthogonale, R est triangulaire supérieure à coefficients diagonaux positifs et :

$$A = Q \cdot R$$

4°) Ecrire une procédure permettant de réaliser la factorisation Q·R d'une matrice réelle inversible. En déduire un moyen de calculer l'inverse.

5°) En déduire une procédure de résolution du système $A \cdot x = b$.

5.9 Matrices complexes

Ecrire un paquetage permettant de manipuler des variables de type complexe. Puis un paquetage permettant de manipuler des matrices à coefficients complexes (voir le paragraphe 6.1).

5.10 Paquetage générique de manipulation des matrices

Ecrire un paquetage générique de manipulation des matrices (voir le paragraphe 6.1). On l'instanciera ensuite sur les réels, les rationnels et les complexes.

6. Programmation Ada

6.1 Spécifications des paquetages COMMON_MATRIX et MATRIX

Ce paquetage est formé d'utilitaires pour travailler avec des matrices carrées et des vecteurs de dimension au plus DIM_MAX déclarée en constante.

Le nom de chaque procédure est assez explicite pour comprendre de quoi il s'agit.

```
package COMMON_MATRIX is
DIM_MAX : constant NATURAL := 80 ;
type MATRICE is array(NATURAL range <>,NATURAL range <>) of FLOAT ;
type VECTEUR is array(NATURAL range <>) of FLOAT ;
type MATRICE_TRIDIAGONALE is array(NATURAL range <>) of VECTEUR(1..3) ;
ERREUR_DIMENSION : exception ;
ERREUR_DETERMINANT_NUL : exception ;
end COMMON_MATRIX ;

with TEXT_IO, COMMON_MATRIX ;
use COMMON_MATRIX ;
package MATRIX is
procedure GET(A : out MATRICE) ;
procedure MATRICE_ALEATOIRE(A : out MATRICE) ;
procedure MATRICE_DU_FICHIER(A : out MATRICE ;
            NOM_DE_REPERTOIRE : in STRING := "A :\DONNEES") ;
procedure MATRICE_HILBERT(A : out MATRICE) ;
procedure MATRICE_TEST(A : in out MATRICE) ;
procedure MATRICE_VAN_DER_MONDE(v : in VECTEUR ; A : in out MATRICE) ;
procedure GET(A : out MATRICE_TRIDIAGONALE) ;
procedure MATRICE_TRIDIAG_ALEATOIRE(A : out MATRICE_TRIDIAGONALE) ;
procedure MATRICE_TRIDIAG_DU_FICHIER(A : out MATRICE_TRIDIAGONALE ;
            NOM_DE_REPERTOIRE : in STRING := "A :\DONNEES") ;
procedure MATRICE_SYMETRIQUE(A : in out MATRICE) ;
procedure MATRICE_SYM_ALEATOIRE(A : in out MATRICE) ;
procedure MATRICE_SYM_DU_FICHIER(A : in out MATRICE ;
            NOM_DE_REPERTOIRE : in STRING := "A :\DONNEES") ;
procedure PUT_LINE(A : in MATRICE ; FORE : in TEXT_IO.FIELD := 6 ;
          AFT : in TEXT_IO.FIELD := 4 ; EXP : in TEXT_IO.FIELD := 0) ;
procedure PUT_LINE(A : in MATRICE_TRIDIAGONALE ;
          FORE : in TEXT_IO.FIELD := 6 ; AFT : in TEXT_IO.FIELD := 4 ;
          EXP : in TEXT_IO.FIELD := 0) ;
procedure GET(v : out VECTEUR) ;
procedure VECTEUR_ALEATOIRE(v : out VECTEUR) ;
procedure VECTEUR_DU_FICHIER(v : out VECTEUR ;
            NOM_DE_REPERTOIRE : in STRING := "A :\DONNEES") ;
procedure PUT_LINE(v : in VECTEUR ; FORE : in TEXT_IO.FIELD := 6 ;
          AFT : in TEXT_IO.FIELD := 4 ; EXP : in TEXT_IO.FIELD := 0) ;
function TRACE(A : in MATRICE) return FLOAT ;
function TRANSPOSE(A : in MATRICE) return MATRICE ;
function "+"(A, B : in  MATRICE) return  MATRICE ;
```

```
function "-"(A, B : in  MATRICE) return  MATRICE ;
function "*"(x : in FLOAT ; A : in MATRICE) return MATRICE ;
function "*"(A : in MATRICE ; v : in VECTEUR) return VECTEUR ;
function "*"(A : in MATRICE_TRIDIAGONALE ; v : in VECTEUR)
            return VECTEUR ;
function "+"(u, v : in  VECTEUR) return  VECTEUR ;
function "-"(u, v : in  VECTEUR) return  VECTEUR ;
function "*"(x : in FLOAT ; u : in  VECTEUR) return  VECTEUR ;
function NORME_DU_SUP(v : in VECTEUR) return FLOAT ;
function NORME_EUCLIDIENNE(v : in VECTEUR) return FLOAT ;
function PRODUIT_SCALAIRE(u, v : in VECTEUR) return FLOAT ;
end MATRIX ;
```

6.2 Spécification du paquetage ALGEBLIN

Dans ce paquetage, on trouvera des procédures permettant de résoudre des systèmes linéaires par des méthodes directes et itératives.

En commentaire, on indique à quel paragraphe de ce chapitre on devra se reporter.

```
with COMMON_MATRIX, MATRIX ;
use COMMON_MATRIX, MATRIX ;
package ALGEBLIN is
procedure SYSTEME_TRIANGULAIRE_SUPERIEUR(A : in MATRICE ;
            b : in VECTEUR ; x : out VECTEUR) ; -- § 3.3
procedure PIVOTAGE(A : in out MATRICE ; b : in out VECTEUR) ; -- § 3.4
function DET_MATRICE_TRIANGULAIRE(A : in MATRICE)
            return FLOAT ; -- § 3.3
function INV_MATRICE_TRIANG_INF(A : in MATRICE)
            return MATRICE ; -- § 3.3
function INV_MATRICE_TRIANG_SUP(A : in MATRICE)
            return MATRICE ; -- § 3.3
procedure DECOMPOSITION_LR(A : in MATRICE ; L, R : in out MATRICE) ;
            -- § 3.6
function DETERMINANT_LR(A : in MATRICE) return FLOAT ; -- § 3.6
function INVERSE_LR(A : in MATRICE) return MATRICE ; -- § 3.6
procedure GAUSS_JORDAN(A : in out MATRICE ; b : in out VECTEUR) ;
            -- § 3.5
function INVERSE_GAUSS_JORDAN(A : in MATRICE) return MATRICE ; -- § 3.5
procedure SYSTEME_TRIDIAGONAL(A : in out MATRICE_TRIDIAGONALE ;
            b : in out VECTEUR) ; -- § 3.7
procedure CHOLESKY(A : in MATRICE ; C : in out MATRICE) ; -- § 3.8
function INVERSE_VAN_DER_MONDE(x : in VECTEUR) return MATRICE ;
            -- Ex. 3
procedure DECOMPOSITION_QR(A : in MATRICE ; Q, R : in out MATRICE) ;
            -- Ex. 8
procedure JACOBI(A : in MATRICE ;b : in VECTEUR ;x : in out VECTEUR) ;
            -- § 3.9.3
procedure RELAXATION(A : in MATRICE ; b : in VECTEUR ;
            OMEGA : in FLOAT := 1.0 ; x : in out VECTEUR) ; -- § 3.9.6
ERREUR_DIAGONALE_NON_DOMINANTE : exception
ERREUR_NON_CONVERGENCE : exception ;
```

```
end ALGEBLIN ;
```

6.3 Spécification du paquetage DONNEES_ALGEBLIN

Dans ce paquetage on trouvera des procédures faciltant l'entrée de matrices et vecteurs de divers types. Il est utilisé dans les procédures de démonstrations du paragraphe suivant.

```
with COMMON_MATRIX ;
use COMMON_MATRIX ;
package DONNEES_ALGEBLIN is
procedure DONNEE_MATRICE(A : in out MATRICE) ;
procedure DONNEE_VECTEUR(b : in out VECTEUR) ;
procedure DONNEE_MATRICE_SYMETRIQUE(A : in out MATRICE) ;
procedure DONNEE_MATRICE_TRIDIAGONALE(A : in out MATRICE_TRIDIAGONALE);
end DONNEES_ALGEBLIN ;
```

6.4 Démonstration de la méthode des pivots de Gauss

```
with TEXT_IO, CRT, COMMON_MATH0, MATH0, COMMON_MATRIX, MATRIX,
     ALGEBLIN, DONNEES_ALGEBLIN ;
use TEXT_IO, CRT, COMMON_MATH0, MATH0, COMMON_MATRIX, MATRIX,
     ALGEBLIN, DONNEES_ALGEBLIN ;
procedure DEM_PIV is
n : NATURAL ;
begin
    MODE_AFFICHAGE ;
    PUT_LINE(IMP,"    RESOLUTION D'UN SYSTEME LINEAIRE") ;
    PUT_LINE(IMP,"       METHODE DES PIVOTS DE GAUSS") ;
    NEW_LINE(IMP) ;
    ENTRER_ENTIER_BORNE(2,DIM_MAX,n,"Dimension du systeme : ") ;
    declare
        A : MATRICE(1..n,1..n) ;
        b : VECTEUR(1..n) ;
    begin
        DONNEE_MATRICE(A) ; DONNEE_VECTEUR(b) ;
        PUT_LINE(IMP,"    SYSTEME AVANT PIVOTAGE") ;
        PUT_LINE(IMP,"MATRICE A :") ; PUT_LINE(A) ;
        PAUSE ;
        PUT_LINE(IMP,"VECTEUR b") ; PUT_LINE(b) ;
        PAUSE ;
        PIVOTAGE(A,b) ;
        CLRSCR ;
        PUT_LINE(IMP,"    SYSTEME APRES PIVOTAGE") ;
        PUT_LINE(IMP,"MATRICE A :") ; PUT_LINE(A) ;
        PAUSE ;
        PUT_LINE(IMP,"VECTEUR b") ; PUT_LINE(b) ;
        PAUSE ;
        CLRSCR ;
        PUT_LINE(IMP,"      SOLUTION DU SYSTEME") ;
        SYSTEME_TRIANGULAIRE_SUPERIEUR(A,b,b) ;
        PUT_LINE(b) ;
        CLOSE(IMP) ;
    end ;
```

```
end DEM_PIV ;
```

6.5 Démonstration de la méthode L·R

```
with TEXT_IO, FIO, CRT, COMMON_MATH0, MATH0, COMMON_MATRIX,
     MATRIX, ALGEBLIN, DONNEES_ALGEBLIN ;
use TEXT_IO, FIO, CRT, COMMON_MATH0, MATH0, COMMON_MATRIX,
     MATRIX, ALGEBLIN, DONNEES_ALGEBLIN ;
procedure DEM_LR is
n : NATURAL ;
begin
    MODE_AFFICHAGE ;
    PUT_LINE(IMP,"    DECOMPOSITION A = L.R DE CROUT") ;
    PUT_LINE(IMP,"    CALCUL DE L'INVERSE DE A") ;
    NEW_LINE(IMP) ;
   PUT_LINE("Attention : La decomposition L.R est possible s.s.i ") ;
    PUT_LINE("il n'y a pas de permutations dans la methode de GAUSS") ;
    NEW_LINE ;
    ENTRER_ENTIER_BORNE(2,DIM_MAX,n,"Dimension de la matrice : ") ;
    declare
        A, L, R, Inv : MATRICE(1..n,1..n) ;
        DET : FLOAT ;
    begin
        DONNEE_MATRICE(A) ;
        DECOMPOSITION_LR(A,L,R) ;
        PUT_LINE(IMP,"    DECOMPOSITION L.R") ;
        PUT_LINE(IMP,"MATRICE L :") ; PUT_LINE(L) ;
        PAUSE ;
        CLRSCR ;
        PUT_LINE(IMP,"MATRICE R") ; PUT_LINE(R) ;
        PAUSE ;
        CLRSCR ;
        PUT_LINE(IMP,"    VERIFICATION : L*R = ") ;
        A := L*R ;
        PUT_LINE(A) ;
        PAUSE ;
        CLRSCR ;
        PUT_LINE(IMP,"   CALCUL DU DETERMINANT") ;
        PUT("Det(A) = ") ;
        DET := DET_MATRICE_TRIANGULAIRE(R) ;
        FIO.PUT(DET,6,4,0) ; NEW_LINE ;
        PAUSE ;
        CLRSCR ;
        PUT_LINE(IMP,"   CALCUL DE L'INVERSE DE A") ;
        INV := INV_MATRICE_TRIANG_SUP(R)*INV_MATRICE_TRIANG_INF(L) ;
        PUT_LINE(INV) ;
        PAUSE ; CLRSCR ;
        PUT_LINE(IMP,"    VERIFICATION : A*INV(A) =") ;
        A := A*INV ;
        PUT_LINE(A) ;
        PAUSE ;
        CLRSCR ;
        CLOSE(IMP) ;
    end ;
```

```
end DEM_LR ;
```

6.6 Démonstration de la méthode du double balayage de Cholesky

```
with TEXT_IO, CRT, COMMON_MATH0, MATH0, COMMON_MATRIX, MATRIX,
     ALGEBLIN, DONNEES_ALGEBLIN ;
use TEXT_IO, CRT, COMMON_MATH0, MATH0, COMMON_MATRIX, MATRIX,
     ALGEBLIN, DONNEES_ALGEBLIN ;

procedure DEM_TRID is
n : NATURAL ;
begin
    MODE_AFFICHAGE ;
    PUT_LINE(IMP,"    RESOLUTION D'UN SYSTEME LINEAIRE TRIDIAGONAL") ;
    PUT_LINE(IMP,"       METHODE DES PIVOTS DE GAUSS") ;
    NEW_LINE(IMP) ;
    PUT_LINE("Attention : la methode est valable si il n'y a pas de") ;
    PUT_LINE("permutations dans la methode de Gauss.") ;
    NEW_LINE ;
    ENTRER_ENTIER_BORNE(2,DIM_MAX,n,"Dimension du systeme : ") ;
    declare
        A : MATRICE_TRIDIAGONALE(1..n) ;
        b : VECTEUR(1..n) ;
    begin
        DONNEE_MATRICE_TRIDIAGONALE(A) ;
        DONNEE_VECTEUR(b) ;
        PUT_LINE(IMP,"    SYSTEME AVANT PIVOTAGE") ;
        PUT_LINE("MATRICE A :") ; PUT_LINE(A) ;
        PAUSE ;
        PUT_LINE(IMP,"VECTEUR b") ; PUT_LINE(b) ;
        PAUSE ;
        CLRSCR ;
        SYSTEME_TRIDIAGONAL(A,b) ;
        PUT_LINE(IMP,"      SOLUTION DU SYSTEME") ;
        PUT_LINE(b) ;
        CLOSE(IMP) ;
    end ;
end DEM_TRID ;
```

6.7 Démonstration de la méthode de Gauss-Jordan

```
with TEXT_IO, CRT, COMMON_MATH0, MATH0, COMMON_MATRIX, MATRIX,
     ALGEBLIN, DONNEES_ALGEBLIN ;
use TEXT_IO, CRT, COMMON_MATH0, MATH0, COMMON_MATRIX, MATRIX,
     ALGEBLIN, DONNEES_ALGEBLIN ;
procedure DEM_GJ is
n : NATURAL ;
begin
    MODE_AFFICHAGE ;
    PUT_LINE(IMP,"      Resolution d'un systeme lineaire.") ;
    PUT_LINE(IMP,"      Calcul de l'inverse d'une matrice.") ;
    PUT_LINE(IMP,"       Methode de Gauss-Jordan.") ;
    NEW_LINE(IMP) ;
    ENTRER_ENTIER_BORNE(1,DIM_MAX,n,"Dimension du systeme : ") ;
```

```
    declare
       A, INV : MATRICE(1..n,1..n) ;
       b : VECTEUR(1..n) ;
    begin
       DONNEE_MATRICE(A) ;
       DONNEE_VECTEUR(b) ;
       PUT_LINE(IMP,"RESOLUTION DE A.x = b, AVEC :") ;
       PUT_LINE(IMP," A = ") ; PUT_LINE(A) ;
       PAUSE ; CLRSCR ;
       PUT_LINE(IMP," et b = ") ; PUT_LINE(b) ;
       PAUSE ; CLRSCR ;
       GAUSS_JORDAN(A,b) ;
       PUT_LINE(IMP,"       Solution, de A.x = b") ;
       NEW_LINE(IMP) ;
       PUT_LINE(b) ;
       PAUSE ; CLRSCR ;
       PUT_LINE(IMP,"Inversion d'une matrice") ;
       NEW_LINE(IMP) ;
       DONNEE_MATRICE(A) ;
       PUT_LINE(IMP,"INVERSION DE LA MATRICE A = ") ;
       NEW_LINE(IMP) ;
       PUT_LINE(A) ;
       PAUSE ; CLRSCR ;
       PUT_LINE(IMP,"Inv(A) = ") ;
       NEW_LINE(IMP) ;
       PUT_LINE(INVERSE_GAUSS_JORDAN(A)) ;
       PAUSE ; CLRSCR ;
       CLOSE(IMP) ;
    end ;
end DEM_GJ ;
```

6.8 Démonstration de la méthode de Cholesky

```
with TEXT_IO, FIO, CRT, COMMON_MATH0, MATH0, COMMON_MATRIX,
     MATRIX, ALGEBLIN, DONNEES_ALGEBLIN ;
use TEXT_IO, FIO, CRT, COMMON_MATH0, MATH0, COMMON_MATRIX,
     MATRIX, ALGEBLIN, DONNEES_ALGEBLIN ;

procedure DEM_CHOL is
n : NATURAL ;
begin
    MODE_AFFICHAGE ;
    PUT_LINE(IMP,"    DECOMPOSITION DE CHOLESKY A = B.Transpose(B)") ;
    NEW_LINE(IMP) ;
    PUT_LINE("Attention : La decomposition est valable pour") ;
    PUT_LINE("A symetrique definie positive.") ;
    NEW_LINE ;
    ENTRER_ENTIER_BORNE(2,DIM_MAX,n,"Dimension de la matrice : ") ;
    declare
        A, B, INV : MATRICE(1..n,1..n) ;
        DET : FLOAT ;
    begin
        DONNEE_MATRICE(A) ;
```

```
        CHOLESKY(A,B) ;
        PUT_LINE(IMP,"    MATRICE B") ;
        PUT_LINE(B) ;
        PAUSE ; CLRSCR ;
        PUT_LINE(IMP,"    VERIFICATION : B*Transpose(B) = ") ;
        A := B*TRANSPOSE(B) ;
        PUT_LINE(A) ;
        PAUSE ; CLRSCR ;
        PUT_LINE(IMP,"   CALCUL DU DETERMINANT") ;
        PUT(IMP,"Det(A) = ") ;
        DET := DET_MATRICE_TRIANGULAIRE(B) ;
        FIO.PUT(IMP,DET,6,4,0) ;
        NEW_LINE(IMP) ;
        PAUSE ; CLRSCR ;
        PUT_LINE(IMP,"   CALCUL DE L'INVERSE DE A") ;
        INV :=
      INV_MATRICE_TRIANG_SUP(TRANSPOSE(B))*INV_MATRICE_TRIANG_INF(B) ;
        PUT_LINE(INV) ;
        PAUSE ; CLRSCR ;
        PUT_LINE(IMP,"    VERIFICATION : A*INV(A) =") ;
        A := A*INV ;
        PUT_LINE(A) ;
        PAUSE ; CLRSCR ;
        CLOSE(IMP) ;
    end ;
end DEM_CHOL ;
```

6.9 Démonstration de la méthode de Jacobi

```
with TEXT_IO, CRT, COMMON_MATH0, MATH0, COMMON_MATRIX, MATRIX,
     ALGEBLIN, DONNEES_ALGEBLIN ;
use TEXT_IO, CRT, COMMON_MATH0, MATH0, COMMON_MATRIX, MATRIX,
     ALGEBLIN, DONNEES_ALGEBLIN ;

procedure DEM_JAC is
n : NATURAL ;
begin
    MODE_AFFICHAGE ;
    TEXT_IO.PUT_LINE(IMP,"Resolution d'un systeme lineaire.") ;
    TEXT_IO.PUT_LINE(IMP,"Methode iterative de Jacobi.") ;
    TEXT_IO.NEW_LINE(IMP) ;
    ENTRER_ENTIER_BORNE(2,DIM_MAX,n,"Dimension du systeme : ") ;
    declare
       A : MATRICE(1..n,1..n) ;
       b, x : VECTEUR(1..n) ;
    begin
       DONNEE_MATRICE(A) ;
       DONNEE_VECTEUR(b) ;
       TEXT_IO.PUT_LINE(IMP,"RESOLUTION DE A.x = b, AVEC :") ;
       TEXT_IO.PUT_LINE(IMP," A = ") ; PUT_LINE(A) ;
       PAUSE ; CLRSCR ;
       TEXT_IO.PUT_LINE(IMP," et b = ") ; PUT_LINE(b) ;
       PAUSE ; CLRSCR ;
```

```
       CHRONOMETRE(0) ;
       JACOBI(A,b,x) ;
       CHRONOMETRE(1) ;
       TEXT_IO.PUT_LINE(IMP,"La solution est :") ; PUT_LINE(x) ;
       PAUSE ; CLRSCR ;
       CLOSE(IMP) ;
    end ;
end DEM_JAC ;
```

6.10 Démonstration de la méthode de relaxation

```
with TEXT_IO, CRT, COMMON_MATH0, MATH0, COMMON_MATRIX, MATRIX,
     ALGEBLIN, DONNEES_ALGEBLIN ;
use TEXT_IO, CRT, COMMON_MATH0, MATH0, COMMON_MATRIX, MATRIX,
     ALGEBLIN, DONNEES_ALGEBLIN ;

procedure DEM_REL is
n : NATURAL ;
begin
    MODE_AFFICHAGE ;
    TEXT_IO.PUT_LINE(IMP,"Resolution d'un systeme lineaire.") ;
    TEXT_IO.PUT_LINE(IMP,"Methode iterative de Relaxation.") ;
    TEXT_IO.NEW_LINE(IMP) ;
    ENTRER_ENTIER_BORNE(2,DIM_MAX,n,"Dimension du systeme : ") ;
    declare
       A : MATRICE(1..n,1..n) ;
       b, x : VECTEUR(1..n) ;
       Omega : FLOAT ;
    begin
       DONNEE_MATRICE(A) ;
       DONNEE_VECTEUR(b) ;
       TEXT_IO.PUT_LINE(IMP,"RESOLUTION DE A.x = b, avec :") ;
       TEXT_IO.PUT_LINE(IMP," A = ") ; PUT_LINE(A) ;
       PAUSE ; CLRSCR ;
       TEXT_IO.PUT_LINE(IMP," et b = ") ; PUT_LINE(b) ;
       PAUSE ; CLRSCR ;
       loop
          ENTRER_REEL_BORNE(0.0001,1.9999,OMEGA,
                            "Parametre de relaxation (entre 0 et 2) : ") ;
          CHRONOMETRE(0) ;
          RELAXATION(A,b,Omega,x) ;
          CHRONOMETRE(1) ;
          TEXT_IO.PUT_LINE(IMP,"La solution est :") ;
          PUT_LINE(x) ;
          PAUSE ; CLRSCR ;
          exit when (not LIRE_REPONSE("Un autre essai : ")) ;
       end loop ;
    CLOSE(IMP) ;
    end ;
end DEM_REL ;
```

6.11 Démonstration de la décomposition Q·R

```
with TEXT_IO, FIO, CRT, COMMON_MATH0, MATH0, COMMON_MATRIX,
     MATRIX, ALGEBLIN, DONNEES_ALGEBLIN ;
use TEXT_IO, FIO, CRT, COMMON_MATH0, MATH0, COMMON_MATRIX, MATRIX,
     ALGEBLIN, DONNEES_ALGEBLIN ;

procedure DEM_QR is
n : NATURAL ;
begin
    MODE_AFFICHAGE ;
    TEXT_IO.PUT_LINE(IMP,"Decomposition A = Q*R, avec :
                          Q orthogonale R triangulaire superieure") ;
    NEW_LINE(IMP) ;
 ENTRER_ENTIER_BORNE(2,DIM_MAX,n,"Entrer la dimension de la matrice") ;
    declare
       A, Q, R : MATRICE(1..n,1..n) ;
       b : VECTEUR(1..n) ;
    begin
       DONNEE_MATRICE(A) ;
       TEXT_IO.PUT_LINE(IMP,"Matrice A :") ; PUT_LINE(A) ;
       PAUSE ; CLRSCR ;
       TEXT_IO.PUT_LINE(IMP,"Calcul de la decomposition Q*R ...") ;
       DECOMPOSITION_QR(A,Q,R) ;
       TEXT_IO.PUT_LINE(IMP,"Matrice Q :") ; PUT_LINE(Q) ;
       PAUSE ; CLRSCR ;
       TEXT_IO.PUT_LINE(IMP,"Matrice R :") ; PUT_LINE(R) ;
       PAUSE ; CLRSCR ;
       TEXT_IO.PUT_LINE(IMP,"Determinant de A.") ;
       PUT(IMP,"Det(A) = Det(R) = ") ;
       FIO.PUT(IMP,DET_MATRICE_TRIANGULAIRE(R),6,4,0) ;
       PAUSE ; CLRSCR ;
       TEXT_IO.PUT_LINE(IMP,"Resolution de A*x = b.") ;
       DONNEE_VECTEUR(b) ;
       b := TRANSPOSE(Q)*b ;
       SYSTEME_TRIANGULAIRE_SUPERIEUR(R,b,b) ;
       TEXT_IO.PUT_LINE(IMP,"Solution x :") ;
       PUT_LINE(b) ;
       PAUSE ; CLRSCR ;
       TEXT_IO.PUT_LINE(IMP,"Verification A*x = ") ;
       b := A*b ;
       PUT_LINE(b) ;
       PAUSE ; CLRSCR ;
       TEXT_IO.PUT_LINE(IMP,"Inverse de A") ;
       R := INV_MATRICE_TRIANG_SUP(R) ;
       Q := R*TRANSPOSE(Q) ;
       PUT_LINE(Q) ;
       PAUSE ; CLRSCR ;
       TEXT_IO.PUT_LINE(IMP,"Verification A*Inv(A) = ") ;
       A := A*Q ;
       PUT_LINE(A) ;
       PAUSE ; CLRSCR ;
       CLOSE(IMP) ;
```

```
    end ;
end DEM_QR ;
```

6.12 Démonstration du calcul de l'inverse d'une matrice de Van Der Monde

```
with TEXT_IO, CRT, COMMON_MATH0, MATH0, COMMON_MATRIX, MATRIX,
     ALGEBLIN, DONNEES_ALGEBLIN ;
use TEXT_IO, CRT, COMMON_MATH0, MATH0, COMMON_MATRIX, MATRIX,
     ALGEBLIN, DONNEES_ALGEBLIN ;

procedure DEM_VAND is
n : NATURAL ;
begin
    MODE_AFFICHAGE ;
    PUT_LINE(IMP,"    INVERSE D'UNE MATRICE DE VAN DER MONDE") ;
    NEW_LINE(IMP,5) ;
    ENTRER_ENTIER_BORNE(2,10,
           "Dimension du vecteur definissant la matrice (< 10) : ") ;
    declare
        A, INV : MATRICE(1..n,1..n) ;
        x : VECTEUR(1..n) ;
    begin
        DONNEE_VECTEUR(x) ;
        MATRICE_VAN_DER_MONDE(x,A) ;
        PUT_LINE(IMP,"    MATRICE DE VAN DER MONDE ASSOCIEE") ;
        PUT_LINE(A) ;
        PAUSE ; CLRSCR ;
        PUT_LINE(IMP,"    INVERSE DE LA MATRICE") ;
        INV := INVERSE_VAN_DER_MONDE(x) ;
        PUT_LINE(INV) ;
        PAUSE ; CLRSCR ;
        PUT_LINE(IMP,"VERIFICATION : A*INV = ") ;
        PUT_LINE(A*INV) ;
        PAUSE ; CLRSCR ;
    end ;
    CLOSE(IMP) ;
end DEM_VAND ;
```

6.13 Spécification du paquetage SPECTRE

Dans ce paquetage, on trouvera des procédures permettant de calculer les élements propres d'une matrice.

En commentaire, on indique à quel paragraphe de ce chapitre on devra se reporter.

```
with COMMON_MATRIX ;
use COMMON_MATRIX ;
package SPECTRE is
procedure DEFLATION(A : in MATRICE ; VALEURS_PROPRES : in out VECTEUR ;
            VECTEURS_PROPRES : in out MATRICE) ; -- § 4.3
procedure RUTISHAUSER(A : in MATRICE ;
            VALEURS_PROPRES : in out VECTEUR) ; -- § 4.4
```

```
procedure JACOBI(A : in MATRICE ; D : in out VECTEUR ;
             V : in out MATRICE) ; -- § 4.5
procedure SOURIAU(A : in MATRICE ; T : in out VECTEUR ;
             INVERSE : in out MATRICE) ; -- 4.6.1
end SPECTRE ;
```

6.14 Démonstration de la méthode de déflation

```
with TEXT_IO, IIO, FIO, CRT, COMMON_MATH0, MATH0, COMMON_MATRIX,
     MATRIX, DONNEES_ALGEBLIN, SPECTRE ;
use TEXT_IO, IIO, FIO, CRT, COMMON_MATH0, MATH0, COMMON_MATRIX,
     MATRIX, DONNEES_ALGEBLIN, SPECTRE ;

procedure DEM_DEFL is
n, p : NATURAL ;
begin
    MODE_AFFICHAGE ;
    PUT_LINE(IMP,"Valeurs propres d'une matrice : Methode de deflation")
;
    NEW_LINE(IMP) ;
    ENTRER_ENTIER_BORNE(2,DIM_MAX,n,"Dimension de la matrice : ") ;
 ENTRER_ENTIER_BORNE(1,n,p,"Nombre de valeurs propres a calculer : ") ;
    declare
       A : MATRICE(1..n,1..n) ;
       VALEURS_PROPRES : VECTEUR(1..p) ;
       VECTEURS_PROPRES : MATRICE(1..n,1..p) ;
       S : FLOAT ;
    begin
       DONNEE_MATRICE(A) ;
       S := TRACE(A) ;
       PUT_LINE(IMP,"Valeurs propres de la matrice A = ") ;
       PUT_LINE(A) ; NEW_LINE(IMP) ;
       PAUSE ; CLRSCR ;
       CHRONOMETRE(0) ;
       DEFLATION(A,VALEURS_PROPRES,VECTEURS_PROPRES) ;
       CHRONOMETRE(1) ; NEW_LINE(IMP) ;
       for j in 1..p
       loop
          PUT(IMP,"Valeur propre Numero ") ;
          PUT(IMP,j) ; PUT(IMP," = ") ;
           PUT(IMP,VALEURS_PROPRES(j),5,5,0) ; NEW_LINE(IMP) ;
          PUT_LINE(IMP,"Vecteur propre associe :") ;
          PUT(IMP,"(") ;
          for i in 1..n - 1
          loop
             PUT(IMP,VECTEURS_PROPRES(i,j),5,5,0) ;
             PUT(IMP,",") ;
          end loop ;
          PUT(IMP,VECTEURS_PROPRES(n,j),5,5,0) ; PUT_LINE(IMP,")") ;
          if (j mod 5 = 0) then
             PAUSE ;
          end if ;
       end loop ;
```

```
      if (p = n) then
         PAUSE ; CLRSCR ;
         PUT_LINE(IMP,"Verification") ;
         NEW_LINE(IMP) ;
         PUT(IMP,"Trace de A = ") ;
         PUT(IMP,S,5,5,0) ; NEW_LINE(IMP) ;
         PUT(IMP,"Somme des valeurs propres = ") ;
         S := 0.0 ;
         for i in 1..n
         loop
            S := S + VALEURS_PROPRES(i) ;
         end loop ;
         PUT(IMP,S,5,5,0) ; NEW_LINE(IMP) ;
      end if ;
   end ;
   CLOSE(IMP) ;
end DEM_DEFL ;
```

6.15 Démonstration de la méthode de Rutishauser

```
with TEXT_IO, IIO, FIO, CRT, COMMON_MATH0, MATH0, COMMON_MATRIX,
     MATRIX, DONNEES_ALGEBLIN, SPECTRE ;
use TEXT_IO, IIO, FIO, CRT, COMMON_MATH0, MATH0, COMMON_MATRIX,
     MATRIX, DONNEES_ALGEBLIN, SPECTRE ;

procedure DEM_RUTI is
n : NATURAL ;
begin
    MODE_AFFICHAGE ;
    PUT_LINE(IMP,
             "Valeurs propres d'une matrice : Methode de Rutishauser") ;
    NEW_LINE(IMP) ;
    ENTRER_ENTIER_BORNE(2,DIM_MAX,n,"Dimension de la matrice : ") ;
    declare
       A : MATRICE(1..n,1..n) ;
       V : VECTEUR(1..n) ;
       S : FLOAT ;
    begin
       DONNEE_MATRICE(A) ;
       S := TRACE(A) ;
       PUT_LINE(IMP,"Valeurs propres de la matrice A = ") ;
       PUT_LINE(A) ; NEW_LINE(IMP) ;
       PAUSE ; CLRSCR ;
       CHRONOMETRE(0) ;
       RUTISHAUSER(A,V) ;
       CHRONOMETRE(1) ; NEW_LINE(IMP) ;
       PUT_LINE(IMP,"Les valeurs propres sont : ") ;
       PUT_LINE(V) ;
       PAUSE ; CLRSCR ;
       PUT_LINE(IMP,"Verification") ;
       NEW_LINE(IMP) ;
       PUT(IMP,"Trace de A = ") ;
       PUT(IMP,S,5,5,0) ; NEW_LINE(IMP) ;
```

```
       PUT(IMP,"Somme des valeurs propres = ") ;
       S := 0.0 ;
       for i in V'range
       loop
          S := S + V(i) ;
       end loop ;
       PUT(IMP,S,5,5,0) ;
       NEW_LINE(IMP) ;
    end ;
    CLOSE(IMP) ;
end DEM_RUTI ;
```

6.16 Démonstration de la méthode de Jacobi

```
with TEXT_IO, IIO, FIO, CRT, COMMON_MATH0, MATH0, COMMON_MATRIX,
     MATRIX, DONNEES_ALGEBLIN, SPECTRE ;
use TEXT_IO, IIO, FIO, CRT, COMMON_MATH0, MATH0, COMMON_MATRIX,
     MATRIX, DONNEES_ALGEBLIN, SPECTRE ;

procedure DEM_VPJA is
n : NATURAL ;
begin
    MODE_AFFICHAGE ;
    PUT_LINE(IMP,
          "Valeurs propres et vecteurs propres : Methode de Jacobi") ;
    NEW_LINE(IMP) ;
    ENTRER_ENTIER_BORNE(2,DIM_MAX,n,"Dimension de la matrice : ") ;
    declare
       A, V : MATRICE(1..n,1..n) ;
       D : VECTEUR(1..n) ;
    begin
       DONNEE_MATRICE(A) ;
       PUT_LINE(IMP,"Spectre de la matrice A = ") ;
       PUT_LINE(A) ;
       NEW_LINE(IMP) ;
       PAUSE ; CLRSCR ;
       CHRONOMETRE(0) ;
       JACOBI(A,D,V) ;
       CHRONOMETRE(1) ;
       NEW_LINE(IMP) ;
       for j in D'range loop
          PUT(IMP,"Valeur propre Numero ") ;
          PUT(IMP,j) ;
          PUT(IMP," = ") ;
          PUT(IMP,D(j),5,6,0) ;
          NEW_LINE(IMP) ;
          PUT_LINE(IMP,"Vecteur propre associe") ;
          PUT(IMP,"(") ;
          for i in V'range(1)
          loop
             PUT(IMP,V(i,j),5,6,0) ;
             if (i < D'LAST) then
                PUT(IMP,",") ;
```

```
            end if ;
         end loop ;
         PUT_LINE(IMP,")") ;
         if (j mod 5 = 0) then
            PAUSE ;
         end if ;
         NEW_LINE(IMP) ;
      end loop ;
   end ;
   CLOSE(IMP) ;
end DEM_VPJA ;
```

6.17 Démonstration de la méthode de Souriau

```
with TEXT_IO, FIO, CRT, COMMON_MATH0, MATH0, COMMON_MATRIX,
   MATRIX, DONNEES_ALGEBLIN, SPECTRE ;
use TEXT_IO, FIO, CRT, COMMON_MATH0, MATH0, COMMON_MATRIX, MATRIX,
   DONNEES_ALGEBLIN, SPECTRE ;

procedure DEM_SOU is
n : NATURAL ;
begin
    MODE_AFFICHAGE ;
    PUT_LINE(IMP,"Calcul du polynome caracteristique : Methode de
Souriau") ;
    NEW_LINE(IMP) ;
    ENTRER_ENTIER_BORNE(2,DIM_MAX,n,"Dimension de la matrice : ") ;
    declare
       A, INVERSE : MATRICE(1..n,1..n) ;
       t : VECTEUR(0..n) ;
    begin
       DONNEE_MATRICE(A) ;
       PUT(IMP,"Polynome caracteristique (puissances decroissantes)") ;
       PUT_LINE(IMP," de la matrice A = ") ;
       PUT_LINE(A) ;
       NEW_LINE(IMP) ;
       PAUSE ; CLRSCR ;
       CHRONOMETRE(0) ;
       SOURIAU(A,t,INVERSE) ;
       CHRONOMETRE(1) ;
       PUT(IMP,"P(x) = ") ;
       PUT_LINE(T) ;
       NEW_LINE(IMP) ;
       PAUSE ;
       PUT_LINE(IMP,"Inverse de A : ") ;
       PUT_LINE(INVERSE) ;
    end ;
    CLOSE(IMP) ;
end DEM_SOU ;
```

CHAPITRE 4

Résolution numérique des systèmes non linéaires

1. Introduction

1.1 Position des problèmes

Dans ce chapitre, on s'intéresse au problème de la résolution d'un système d'équations non linéaires :

$$\begin{cases} f_1(x_1,\ldots,x_n) = 0 \\ \ldots\ldots\ldots\ldots\ldots \\ f_n(x_1,\ldots,x_n) = 0 \end{cases}$$

où les f_j sont des fonctions à valeurs réelles définies sur une partie de R^n.

Le système précédent sera noté de façon plus compacte :

$$(1) \quad f(x) = 0 \qquad (x \in \Omega)$$

où Ω est une partie de R^n et $f : \Omega \to R^n$.

On étudiera tout d'abord le cas particulier des équations numériques, c'est-à-dire le cas n = 1, puis le cas des systèmes d'équations (n > 1).

1.2 Remarques

(i) Dans le cas des systèmes linéaires ($f(x) = A \cdot x - b$), on a distingué deux types de méthodes : les méthodes directes et les méthodes itératives.

Dans le cas des systèmes non linéaires, il y a peu d'espoir d'aboutir avec des méthodes directes. Les méthodes de résolution seront donc directes, c'est-à-dire qu'on cherchera à construire une suite $(x^{(k)})_{k \in N}$ qui converge vers la solution quand cette dernière est unique.

(ii) Deux problèmes importants vont alors se poser :

(i) le problème de l'existence et de l'unicité d'une solution ;
(ii) le problème de la localisation de la racine dans un domaine Ω donné. Cette localisation est nécessaire pour démarrer une méthode itérative.

Dans le cas des fonctions continues d'une variable réelle, on dispose du théorème des valeurs intermédiaires qui permet d'apporter une réponse à cette question, mais dans le cas des fonctions de plusieurs variables la situation est plus délicate.

Considérons, par exemple, un système de deux équations à deux inconnues :

$$\begin{cases} f(x,y) = 0 \\ g(x,y) = 0 \end{cases} \quad (x,y) \in \Omega \subset R^2$$

les solutions sont les points d'intersection des courbes f(x,y) = 0 et g(x,y) = 0. Mais la connaissance des signes de f et de g n'est pas d'un grand secours pour localiser les racines :

Pour n ≥ 2 la situation est encore plus compliquée, il s'agit de chercher les points d'intersection de n hypersurfaces.

2. Cas des équations numériques

2.1 Notations

On se donne un intervalle réel I, une fonction f : I → R et on cherche des solutions de f(x) = 0 sur I.

Dans le cas général on cherchera une solution réelle dans un intervalle donné. Mais dans le cas particulier des polynômes on pourra chercher toutes les solutions réelles ou complexes.

2.2 La méthode dichotomie ou de bissection

Si la fonction f : [a,b] → R est continue et strictement monotone avec f(a)·f(b) < 0, le théorème des valeurs intermédiaires nous dit que l'équation f(x) = 0 admet une unique solution dans cet intervalle.

L'idée de la méthode de dichotomie est de découper l'intervalle en deux parties égales et de garder celle qui contient la solution. On répète ensuite le processus.

Avec les hypothèses ci-dessus la méthode converge toujours, mais cette convergence peut être lente. De plus elle ne se généralise pas au cas des systèmes.

L'algorithme est donc le suivant :

$[a_0,b_0] = [a,b]$

Si $[a_k,b_k]$ contient la racine de f(x) = 0, on pose $c_k = \dfrac{a_k + b}{2}$ et

si $f(a_k)\cdot f(c_k) \le 0$
alors $[a_{k+1},b_{k+1}] = [a_k,c_k]$
sinon $[a_{k+1},b_{k+1}] = [c_k,b_k]$.

Les suites (a_k) et (b_k)sont donc adjacentes et convergent vers la solution x de f(x) = 0.

De plus, pour les erreurs d'approximation, on a :

$$|x - a_k| \le \frac{b-a}{2^k},\ |x - b_k| \le \frac{b-a}{2^k}\ (k \ge 0)$$

ce qui permet de connaître le nombre d'itération suffisant pour atteindre une précision ε donnée. Ce nombre est $k \ge \log_2\left(\frac{b-a}{\varepsilon}\right)$.

Le test d'arrêt que nous utiliserons est $|a_k - b_k| < \varepsilon$.

La programmation structurée est alors la suivante, en supposant $f(a)\cdot f(b) \le 0$.

```
PROCEDURE Dichotomie(Entrée f : Fonction ; a, b, ε : Réel ;  Sortie x : Réel) ;
Début
   y = f(a) ;
   Répéter
      c = (a + B) / 2 ;
      z = f(c) ;
      Si y·z ≤ 0 Alors
         b = c ;
      Sinon Début
               a  = c ;
               y = z ;
            Fin ;
   Jusqu'à (|b − a| < ε) ou (z = 0) ;
```

Remarque — Pour trouver toutes les racines sur un intervalle [a,b], on peut le découper en un nombre suffisamment grand de sous-intervalles $[a_i,b_i]$ et appliquer la procédure précédente sur chacun de ces intervalles.

2.3 La méthode de Newton-Raphson

L'idée de cette méthode est de se ramener à un problème de point fixe qui peut se traiter par la méthode des approximations successives.

Soit x la solution de $f(x) = 0$ et x_k une approximation de x, on cherche $h > 0$ tel que $x_k + h$ soit une meilleure approximation. En supposant que f est continûment dérivable, on peut écrire :

$$f(x_k + h) = f(x_k) + h\cdot f'(x_k) + O(h^2)$$

et pour h assez petit, $f(x_k + h) \cong 0$ entraîne :

$$f(x_k) + h\cdot f'(x_k) \cong 0$$

soit :

$$h = -\frac{f(x_k)}{f'(x_k)}$$

Ces considérations heuristiques nous amènent à considérer la suite définie par :

$$(1)\quad \begin{cases} x_0 \in I \\ x_{k+1} = x_k - \dfrac{f(x_k)}{f'(x_k)} \quad (k \ge 0) \end{cases}$$

ce qui peut s'illustrer comme ci-dessous :

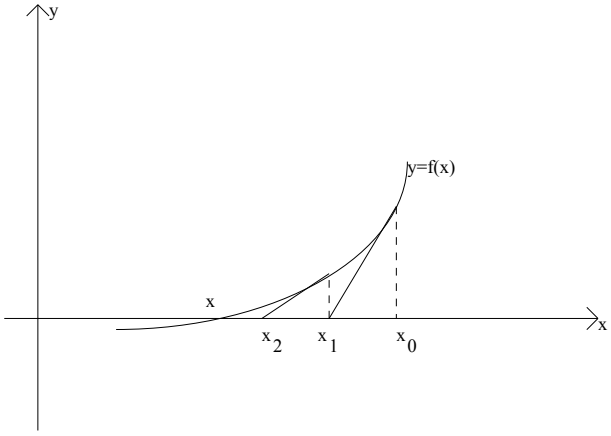


Figure 4.1

Cette méthode s'applique si x est racine simple de f, c'est-à-dire si f(x) = 0 et f′(x) ≠ 0. Dans ce cas on aura f′(t) ≠ 0 pour t voisin de la racine x. De manière plus précise, une condition suffisante de convergence est donnée par le :

Théorème : Si f : [a,b] → R est de classe C^2 avec f(a)·f(b) ≤ 0, f strictement monotone (soit f′ de signe constant) et f″(x) ≠ 0 sur [a,b] (concavité dans le même sens), alors pour tout x_0 dans [a,b] tel que $f(x_0)\cdot f''(x_0) > 0$, la suite définie par (1) converge vers l'unique solution de f(x) = 0.

Une majoration de l'erreur est donnée par :

$$\left|x_k - x\right| \le \left|x_0 - x\right|^{2^n} \cdot \left(\frac{M_2}{2\cdot m_1}\right)^{2^n-1}$$

où :

$$m_1 = \mathrm{Inf}\left\{\left|f'(x)\right|;\ x \in [a,b]\right\};\ M_2 = \mathrm{Sup}\left\{\left|f''(x)\right|;\ x \in [a,b]\right\}$$

Démonstration — Cf. Théodor : P. 26.

Remarques — (i) Dans la pratique, la méthode de Newton est très efficace si le point de départ est choisi proche de la solution.
(ii) Cette méthode se généralise au cas des systèmes d'équations.
(iii) L'évaluation de la dérivée peut se faire en utilisant l'approximation par différence finie centrée soit :

$$f'(x) \cong \frac{f(x+h) - f(x-h)}{2\cdot h}$$

ce qui donne une approximation de l'ordre de h^2.
(iv) Comme test d'arrêt, on prendra $\left|f(x_k)\right| < \varepsilon$ où ε est une précision donnée.

La programmation structurée est alors la suivante :

PROCEDURE Newton_Raphson(Entrée f : Fonction ; x_0, ε : Réel ; Sortie x : Réel) ;
Début
x = x_0 ; k = 0 ; h = 10^{-3} ;

```
Répéter
    k = k + 1 ;
```

$df = \dfrac{f(x+h) - f(x-h)}{2 \cdot h}$;

$\delta = \dfrac{f(x)}{df}$;

$x = x - \delta$;

Jusqu'à (k = MaxIter) ou ($|\delta| < \varepsilon$) ;

Fin ;

3. Cas des équations algébriques

3.1 Introduction

Dans ce paragraphe, on considère le cas où f est un polynôme de degré n. On le notera :

$$P(x) = a_0 + a_1 \cdot x + ... + a_n \cdot x^n$$

où $a_n \neq 0$.

Le théorème de D'Alembert-Gauss nous dit que l'équation P(x) = 0 admet n solutions dans C. Mais ce théorème n'est pas constructif.

Pour certaines applications, on sait que toutes les racines de P sont réelles distinctes et on sait les localiser. C'est le cas pour les polynômes orthogonaux classiques (Cf. § (3.6) du chapitre sur le calcul numérique des intégrales).

Dans ce paragraphe nous décrirons la méthode de Newton-Maehly qui est une adaptation de celle de Newton. Ensuite, après avoir décrit la méthode de Newton pour les systèmes non linéaires, nous l'utiliserons pour décrire la méthode de Bairstow qui permet d'obtenir toutes les racines réelles ou complexes d'un polynôme (Cf. § 5).

3.2 La méthode Newton_Maehly

On suppose dans ce paragraphe que toutes les racines du polynôme P sont réelles. C'est le cas pour les polynômes orthogonaux classiques ou pour le polynôme caractéristique d'une matrice symétrique.

La méthode de Newton-Maehly permet d'obtenir toutes les racines de P avec leur multiplicité.

On note $\alpha_1, \alpha_2, ..., \alpha_p$, les racines de P de multiplicités respectives $m_1, m_2, ..., m_p \geq 1$.

On a alors le résultat suivant :

Théorème : Soit $(x_k)_{k \in N}$ la suite définie par :

$$\begin{cases} x_0 > \alpha_p \\ x_{k+1} = x_k - \dfrac{P(x_k)}{P'(x_k)} \end{cases}$$

est strictement décroissante et $\lim_{k\to+\infty}(x_k) = \alpha_p$. Une majoration de l'erreur étant donnée par :

$$|x_k - \alpha_p| \leq \left(1 - \frac{1}{n}\right)^k \cdot (x_0 - \alpha_p)$$

Démonstration — D. Monasse : RMS N° 5, Janvier 1987. Stoer et Burlisch : p. 272.

Remarque 1 — La convergence est très rapide dans le cas de racines simples.
Remarque 2 — Pour avoir une méthode convenable, on devra prendre $x_0 > \alpha_p$, en étant assez proche de α_p. Un tel choix est donné par :

$$x_0 = \text{Max}\left\{\frac{|a_0|}{|a_n|};\ 1 + \frac{|a_i|}{|a_n|};\ 1 \leq i \leq n-1\right\}$$

(Cf. Stoer et Burlisch, p. 273).

Le calcul des autres racines peut se faire en appliquant ce qui précède à $P_1(x) = \frac{P(x)}{(x - \alpha_p)}$ et en remarquant que :

$$\frac{P_1(x)}{P_1'(x)} = \frac{P(x)}{P'(x) - \frac{P(x)}{x - \alpha_p}}$$

La limite de la nouvelle suite $(x_k)_{k\in N}$ sera encore α_p si $m_p > 1$ et α_{p-1} si $m_p = 1$.
L'algorithme de calcul de toutes les racines est alors le suivant :
Etape m — On dispose des racines $x_1 \geq ... \geq x_{m-1}$ et la racine x_m est donnée comme limite de la suite définie par :

$$\begin{cases} x_0^{(m)} = x_0 \\ x_{k+1}^{(m)} = x_k^{(m)} - \dfrac{P\left(x_k^{(m)}\right)}{P'\left(x_k^{(m)}\right) - P\left(x_k^{(m)}\right) \cdot \sum_{i=1}^{m-1} \dfrac{1}{x_k^{(m)} - x_i}} \end{cases}$$

Comme test d'arrêt, on pourra prendre $\left|P\left(x_k^{(m)}\right)\right| < \varepsilon$.

Remarque 3 — Accélération de la convergence (Cf. Stoer et Burlisch, p. 277).
On considère la suite définie par :

$$\begin{cases} y_0 > \alpha_p \\ y_{k+1} = y_k - 2 \cdot \dfrac{P(y_k)}{P'(y_k)} \end{cases}$$

et on calcule les y_k tant que $y_{k+1} < y_k$ (la suite est théoriquement croissante, mais dès qu'on est très proche de la solution cela peut ne plus être vrai à cause des erreurs d'arrondis), puis dès que $y_{k0+1} \geq y_{k0}$, on utilise la suite (x_k) avec $x_0 = y_{k0}$.

On utilise la même idée pour les autres racines.

On a donc en définitive la programmation structurée suivante :

PROCEDURE Newton_Maehly(Entrée n : Entier ; P : Polynôme ; Sortie x : Vecteur) ;
Début

$$x_0 = Max\left\{\frac{|a_0|}{|a_n|};\ 1+\frac{|a_i|}{|a_n|};\ 1 \le i \le n-1\right\}$$

Pour m Allant de 1 à n Faire
Début
xkP1 = x_0 *;*
Pour j Allant de 2 à 1 Faire
Début
Répéter
xk = xkP1 ; U = P(xk) ; S = 0 ;
Si $|U| > \varepsilon$ *Alors*
Début
V = P'(xk) ;
Pour i Allant de 1 à m - 1 Faire $S = S + \frac{1}{xk - x_i}$ *;*
$V = \frac{U}{V - U \cdot S}$ *; xkP1 = xk - j·V ;*
Fin ;
Jusqu'à (xkP1 $\ge$ *xk) ou (*$|U| < \varepsilon$*) ;*
Fin ;
x_m *= xk ;*
Fin ;
Fin ;

4. Résolution des systèmes non linéaires par la méthode de Newton-Raphson

4.1 Introduction

On suppose ici que f : $\Omega \subset R^n \rightarrow R$ est de classe C^1 et que l'équation f(x) = 0 admet une solution α dans R^n.

On suppose de plus que, dans un voisinage de α la *matrice jacobienne* de f en x :

$$df(x) = \begin{pmatrix} \frac{\partial f_1}{\partial x_1}(x) & .... & \frac{\partial f_1}{\partial x_n}(x) \\ .. & .. & .. \\ \frac{\partial f_n}{\partial x_1}(x) & .... & \frac{\partial f_n}{\partial x_n}(x) \end{pmatrix}$$

est inversible.

4.2 Algorithme de Newton-Raphson

La méthode de Newton étudiée dans le cas n = 1 se généralise en considérant la suite $(x^k)_{k\in N}$ définie par :

$$(1)\quad \begin{cases} x^{(0)} \in \Omega \\ x^{(k+1)} = x^{(k)} - \delta^{(k)} \quad (k \geq 0) \end{cases}$$

où δ^k est solution du système linéaire : $f(x^{(k)})\cdot\delta^k = f(x^{(k)})$.

En général, si $x^{(0)}$ est choisi assez près de la solution α alors la méthode converge de façon quadratique. De manière plus précise, on a le

Théorème : Si f est de classe C^1 sur Ω convexe avec Dét(df(α)) $\neq 0$ et s'il existe une constante L > 0 telle que $\|df(x) - df(\alpha)\| \leq L\cdot\|x-\alpha\|$ pour tout x dans Ω, alors la suite définie par (1) converge vers la solution α de f(x) = 0. De plus la convergence est quadratique, c'est-à-dire que :

$$\|x^{(k+1)} - \alpha\| \leq \text{Cste}\cdot\|x^{(k)} - \alpha\|^2$$

Demonstration — Ortega et Rheinboldt : p. 312.

Remarque — La convergence est locale, c'est-à-dire que le point de départ de la suite doit être choisi assez proche de la solution.

Par exemple, si on considère le système :

$$\begin{cases} x - \text{tg}(y) = 0 \\ x - y = 0 \end{cases}$$

de solution (4.49, 4.49), en prenant $(x^{(0)}, y^{(0)}) = (4.1, 4.0)$, la suite diverge.

4.3 Calcul de la matrice jacobienne

Le calcul de la matrice jacobienne en $x^{(k)}$ se faisant par les *différences finies centrées* ; soit :

$$\frac{\partial f_i}{\partial x_j}\left(x^{(k)}\right) \cong \frac{f_i\left(\beta^{k,j}\right) - f_i\left(\gamma^{k,j}\right)}{2\cdot h}$$

où : $\beta^{k,j} = \left(x_1^{(k)},\ldots,x_{j-1}^{(k)},x_j^{(k)}+h,x_{j+1}^{(k)},\ldots,x_n^{(k)}\right)$,

$\gamma^{k,j} = \left(x_1^{(k)},\ldots,x_{j-1}^{(k)},x_j^{(k)}-h,x_{j+1}^{(k)},\ldots,x_n^{(k)}\right)$ et h est un pas assez petit.

Comme test d'arrêt dans les itérations, on pourra prendre $\left\|f\left(x^{(k)}\right)\right\| \leq \varepsilon$, où ε est une précision donnée.

Enfin le système linéaire donnant δ^k pourra être résolu par la méthode des pivots de Gauss-Jordan.

4.4 Programmation structurée

On écrit tout d'abord une procédure de calcul de la matrice jacobienne.

```
PROCEDURE MatriceJacobienne(Entrée n : Entier ; f : Tableau_Fonctions ; x : Vecteur ;
                                        Sortie df : Matrice) ;
Constante h = 10^-3 ;
Début
    Pour i Allant de 1 à n Faire
    Début
        Pour j Allant de 1 à n Faire
        Début
            x_j = x_j + h ;
            a = f_i(x) ;
            x_j = x_j – 2·h ;
            b = f_i(x) ;
            x_j = x_j + h ;
            df(i,j) = (a – b) / (2·h) ;
        Fin ;
    Fin ;
Fin ;
```

Puis la procédure principale peut s'écrire :

```
PROCEDURE Newton_Raphson(Entrée n : Entier ; f : Tableau_Fonctions ;
                                        Entrée_Sortie x : Vecteur) ;
Début
    k = 0 ;

    Répéter
        k = k + 1 ;
        MatriceJacobienne(n,f,x,A) ;
        b = f(x) ;
        Gauss_Jordan(n,A,b) ;
        x = x – b ;
    Jusqu'à (k = MaxIter) ou (||f(x)|| < ε) ;
Fin ;
```

5. Racines d'un polynôme. Méthode de Bairstow

5.1 Principe de la méthode

L'idée est d'écrire le polynôme $P(x) = a_0 + a_1 \cdot x + ... + a_n \cdot x^n$ comme un produit de polynômes de degré au plus 2.

La première étape consiste donc à trouver p et q dans R tels que :

$$(1) \quad P(x) = Q(x) \cdot (x^2 + p \cdot x + q)$$

On itère ensuite le procédé.

5.2 Recherche des coefficients p et q

Pour p, q donnés dans R, on peut écrire par division euclidienne :

$$(2) \quad P(x) = B(x) \cdot (x^2 + p \cdot x + q) + R \cdot x + S$$

les coefficients R et S dépendants de p et q. Il s'agit donc de déterminer p et q solutions du système de deux équations à deux inconnues :

$$(3)\quad \begin{cases} R(p,q) = 0 \\ S(p,q) = 0 \end{cases}$$

Le premier problème est donc de calculer R et S en fonction de p, q et des coefficients du polynôme P. Ensuite le système (3) sera résolu par la méthode de Newton-Raphson.

Pour ce faire, on pose :

$$\begin{cases} B(x) = b_0 + b_1 \cdot x + \ldots + b_{n-2} \cdot x^{n-2} \\ R(x) = R \cdot x + S = b_{-1} \cdot (x + p) + b_{-2} \end{cases}$$

On va d'abord calculer les b_i pour $i = -2, -1, \ldots, n-2$.

Par identification, on a :

$$\begin{cases} a_n = b_{n-2} \\ a_{n-1} = p \cdot b_{n-2} + b_{n-3} \\ a_k = b_{k-2} + p \cdot b_{k-1} + q \cdot b_k \quad (k = n-2, \ldots, 0) \end{cases}$$

d'où :

$$(R1)\quad \begin{cases} b_n = b_{n-1} = 0 \\ b_{k-2} = a_k - p \cdot b_{k-1} - q \cdot b_k \quad (k = n, \ldots, 0) \end{cases}$$

en particulier on a b_{-1} et b_{-2} donc R et S.

Le deuxième problème est de calculer les dérivées partielles de R et S par rapport à p et q.

On a :

$$\begin{cases} R = b_{-1} \\ S = b_{-1} \cdot p + b_{-2} \end{cases}$$

donc :

$$\begin{cases} \dfrac{\partial R}{\partial p} = \dfrac{\partial b_{-1}}{\partial p} \\ \dfrac{\partial R}{\partial q} = \dfrac{\partial b_{-1}}{\partial q} \\ \dfrac{\partial S}{\partial p} = b_{-1} + p \cdot \dfrac{\partial b_{-1}}{\partial p} + \dfrac{\partial b_{-2}}{\partial p} \\ \dfrac{\partial S}{\partial q} = p \cdot \dfrac{\partial b_{-1}}{\partial q} + \dfrac{\partial b_{-2}}{\partial q} \end{cases}$$

tout revient à calculer les $\frac{\partial b_k}{\partial p}$ et $\frac{\partial b_k}{\partial q}$ pour k = –2 et k = –1. Ce qui se fait en dérivant les relations (R1). On obtient alors la récurrence :

$$\begin{cases} \dfrac{\partial b_n}{\partial p} = \dfrac{\partial b_{n-1}}{\partial p} = \dfrac{\partial b_n}{\partial q} = \dfrac{\partial b_{n-1}}{\partial q} = 0 \\ \dfrac{\partial b_{k-2}}{\partial p} = -b_{k-1} - p \cdot \dfrac{\partial b_{k-1}}{\partial p} - q \cdot \dfrac{\partial b_k}{\partial p} \quad (k = n,\ldots,0) \\ \dfrac{\partial b_{k-2}}{\partial q} = -p \cdot \dfrac{\partial b_{k-1}}{\partial q} - b_k - q \cdot \dfrac{\partial b_k}{\partial q} \end{cases}$$

En posant :

$$\begin{cases} c_k = -\dfrac{\partial b_{k-1}}{\partial p} \\ c'_k = -\dfrac{\partial b_{k-2}}{\partial q} \end{cases}$$

on constate que ces deux suites vérifient la même récurrence :

$$\text{(R2)} \quad \begin{cases} u_{n+1} = u_n = 0 \\ u_{k-1} = b_{k-1} + p \cdot u_k + q \cdot u_{k+1} \quad (k = n,\ldots,0) \end{cases}$$

et en particulier $c_k = c'_k$.

Le troisième problème est alors de résoudre le système linéaire intervenant dans la méthode de Newton-Raphson.

A l'étape k + 1 du calcul, on a :

$$(p_{k+1},q_{k+1}) = (p_k,q_k) + (\delta p,\delta q)$$

où $(\delta p,\delta q)$ est solution du système de deux équations à deux inconnues :

$$d(R,S)(p_k,q_k)\cdot(\delta p,\delta q) = -(R(p_k,q_k),S(p_k,q_k))$$

soit :

$$\begin{cases} \dfrac{\partial b_{-1}}{\partial p} \cdot \delta p + \dfrac{\partial b_{-1}}{\partial q} \cdot \delta q = -b_{-1} \\ \left(b_{-1} + p_k \cdot \dfrac{\partial b_{-1}}{\partial p} + \dfrac{\partial b_{-2}}{\partial p} \right) \cdot \delta p + \left(p_k \cdot \dfrac{\partial b_{-1}}{\partial q} + \dfrac{\partial b_{-2}}{\partial q} \right) \cdot \delta q = -\left(b_{-1} \cdot p_k + b_{-2} \right) \end{cases}$$

avec les notations précédentes, ce système va s'écrire :

$$\begin{cases} c_0 \cdot \delta p + c_1 \cdot \delta q = b_{-1} \\ \left(-b_{-1} + p_k \cdot c_0 + c_{-1} \right) \cdot \delta p + \left(p_k \cdot c_1 + c_0 \right) \cdot \delta q = \left(b_{-1} \cdot p_k + b_{-2} \right) \end{cases}$$

la solution est :

$$\begin{cases} \delta p = \dfrac{U}{D} \\ \delta q = \dfrac{V}{D} \end{cases}$$

où on a noté :

$$\begin{cases} D = c_0^2 - c_1 \cdot c_{-1} + c_1 \cdot b_{-1} \\ U = c_0 \cdot b_{-1} - c_1 \cdot b_{-2} \\ V = c_0 \cdot b_{-2} + b_{-1}^2 - b_{-1} \cdot c_{-1} \end{cases}$$

5.3 Programmation structurée

On écrit tout d'abord la procédure de division qui donne les coefficients p et q et le polynôme B tels que $P(x) = Q(x)\cdot(x^2 + p\cdot x + q)$.

PROCEDURE Division(Entrée n : Entier ; P : Polynôme ; Sortie B : Vecteur ; p, q : Réels) ;
Début
Se donner p_0 et q_0 ;
k = 0 ;
p = p_0 ; q = q_0 ;
Répéter
Calcule_Coeff_bk(n,P,p,q,B) ; { Récurrence R1 }
Calcule_Coeff_ck(n,B,p,q,C) ; { Récurrence R2 }
$D = c_0^2 - c_1 \cdot c_{-1} + c_1 \cdot b_{-1}$;
$U = c_0 \cdot b_{-1} - c_1 \cdot b_{-2}$;
$V = c_0 \cdot b_{-2} + b_{-1}^2 - b_{-1} \cdot c_{-1}$;
$\delta p = \frac{U}{D}$; $\delta q = \frac{V}{D}$;
p = p + δp ; q = q + δq ;
Jusqu'à $\left(|\delta p| < \varepsilon\right)$ *et* $\left(|\delta q| < \varepsilon\right)$ *;*
Fin ;

Les procédures de calcul des coefficients b_k et c_k s'écrivant :

PROCEDURE Calcule_Coeff_bk(Entrée n : Entier ; P : Polynôme ; p,q : Réel ;
Sortie B : Vecteur) ;
Début
b_n = 0 ; b_{n-1} = 0 ;
Pour k allant de n à 0 faire $b_{k-2} = a_k - p\cdot b_{k-1} - q\cdot b_k$;
Fin ;

PROCEDURE Calcule_Coeff_ck(Entrée n : Entier ; B : Vecteur ; p,q : Réel ;
Sortie C : Vecteur) ;
Début
c_{n+1} = 0 ; c_n = 0 ;
Pour k allant de n à 0 faire $c_{k-1} = b_{k-1} + p\cdot c_k + q\cdot c_{k+1}$;
Fin ;

La procédure principale peut alors s'écrire :

PROCEDURE Bairstow(Entrée n : Entier ; P : Polynôme ; Sortie x : VecteurComplexe) ;
Début
Pour i allant de 0 à n faire $a_i = a_i / a_n$;
k = 0 ;
Répéter
Division(n,P,B,p,q) ;

n = n – 2 ;
Pour i allant de 0 à n faire $a_i = b_i$ *;*
RésolEquDegre2(p,q,x1,x2) ;
$k = k + 1$ *;* $x_k = x1$ *;*
$k = k + 1$ *;* $x_k = x2$ *;*

```
    Jusqu'à (n ≤ 2) ;
    Si n = 2 Alors
    Début
        RésolEquDegre2(a1,a0,x1,x2) ;
        k = k + 1 ; xk = x1 ;
        k = k + 1 ; xk = x2 ;
    Fin ;
    Si n = 1 Alors
    Début
        k = k + 1 ; xk = –a0/a1 ;
    Fin ;
Fin ;
```

6. Exercices

6.1 Méthode des approximations successives

On veut résoudre l'équation g(x) = 0 sur [a,b], où g est continûment dérivable et admet une unique racine dans [a,b]. Cette équation équivaut à $f_\alpha(x) = x - \alpha \cdot g(x)=0$, où α est un réel non nul à choisir. Pour approcher la solution de cette dernière équation, on considère la suite des approximations successives définie par :

$$\begin{cases} x_0 \in [a,b] \\ x_{n+1} = f_\alpha(x_n) \quad (n \geq 0) \end{cases}$$

1°) Montrer que si $|f'_\alpha(x)| < 1$ sur [a,b], alors la suite $(x_k)_{k\in N}$ converge vers l'unique solution de f(x) = 0.

2°) On pose $m = \text{Inf}\{g'(x); x \in [a,b]\}$, $M = \text{Sup}\{g'(x); x \in [a,b]\}$ et on suppose que m > 0.

(a) Déterminer α tel que $|f'_\alpha(x)| \leq \dfrac{M-m}{M+m}$ sur [a,b].

(b) En déduire que pour un tel choix de α la méthode est convergente.

3°) Soit à résoudre g(x) = tg(x) - x = 0 sur $\left]\dfrac{3\cdot\pi}{2}, \dfrac{5\cdot\pi}{2}\right[$. On pose a = 7.65, b = 7.75.

(a) Montrer que g admet une unique racine dans [a,b].

(b) Déterminer le coefficient α du 2°) (a).

(c) Décrire un algorithme correspondant à la méthode décrite ci-dessus.

6.2 Méthode de Bernouilli

Soit $P(x) = x^n + a_{n-1}\cdot x^{n-1} + ... + a_1\cdot x + a_0$ un polynôme à coefficients réels admettant une unique racine de module maximal λ_1.

On définit la suite $(x_k)_{k\in N}$ par :

$$x_k + a_{n-1}\cdot x_{k-1} + ... + a_0\cdot x_{k-n} = 0 \quad (k \geq n)$$

les valeurs initiales $x_0, ..., x_{n-1}$ étant arbitraires.

1°) Montrer que $\lim_{k\to+\infty}\left(\frac{x_{k+1}}{x_k}\right)=\lambda_1$.

2°) Montrer que $\lim_{k\to+\infty}\left(\frac{\frac{x_{k+1}}{x_k}-\lambda_1}{\frac{x_{k1}}{x_{k-1}}-\lambda_1}\right)=\frac{\lambda_2}{\lambda_1}$.

3°) En déduire l'évaluation de l'erreur : $\left|\frac{x_{k+1}}{x_k}-\lambda_1\right|\cong\left|\frac{\lambda_2}{\lambda_1}\right|^n$.

4°) Comment procéder pour déterminer les autres racines (méthode de déflation).

6.3 Méthode de la fausse position

Décrire la méthode obtenue, si dans la méthode de Newton-Raphson on utilise comme approximation de la dérivée en x_k la quantité $f'(x_k)\cong\frac{f(x_k)-f(x_{k-1})}{x_k-x_{k-1}}$.

Donner une interprétation géométrique.

6.4 Méthode de Muller

Dans la méthode de la fausse position, on fait une interpolation de y = f(x) par la droite passant par $(x_{k-1},f(x_{k-1}))$ et $(x_k,f(x_k))$.

En faisant une interpolation par un polynôme de degré 2 passant par $(x_{k-2},f(x_{k-2}))$, $(x_{k-1},f(x_{k-1}))$ et $(x_k,f(x_k))$ et en prenant comme point d'intersection des deux courbes celui qui donne la valeur la plus proche de x_k, donner la valeur du point x_{k+1} obtenu.

7. Programmation Ada

7.1 Spécification du paquetage EQUATIONS_GENERIQUE

```
generic
  with function f(x : in FLOAT) return FLOAT ;
package EQUATIONS_GENERIQUE is
procedure DICHOTOMIE(a, b : in FLOAT ; x : out FLOAT ;
             Eps : in FLOAT := 1.0E-6) ; -- § (2.2)
procedure NEWTON_RAPHSON(x0 : in FLOAT ; x : out FLOAT ;
             Eps : in FLOAT := 1.0E-6) ; -- § (2.3)
ERREUR_CONVERGENCE : exception ;
end EQUATIONS_GENERIQUE ;
```

7.2 Démonstration du paquetage EQUATIONS_GENERIQUE

```
with TEXT_IO, IIO, FIO, CRT, COMMON_MATH0, MATH0, MATH1, MATH2,
     EQUATIONS_GENERIQUE ;
use TEXT_IO, IIO, FIO, CRT, COMMON_MATH0, MATH0, MATH1, MATH2 ;
```

```
procedure DEM_EQU is

FF : FONCTION ;
Choix : INTEGER range 0..2 ;

function f(x : in FLOAT) return FLOAT is
begin
    return EVALUE(FF,x) ;
end f ;

package EQUATIONS is new EQUATIONS_GENERIQUE(f) ;
use EQUATIONS ;

procedure DEMO_DICHOTOMIE is
a, b, ak, bk, x, y, z, h : FLOAT ;
n, k : INTEGER ;
begin
    MODE_AFFICHAGE ;
    PUT_LINE(
        "RACINES DE f(x) = 0 SUR [a,b] PAR LA METHODE DE DICHOTOMIE.") ;
    NEW_LINE ;
    PUT_LINE("L'intervalle [a,b] est decoupe en n sous-intervalle") ;
    PUT_LINE(
   "et on cherche une racine dans chaque [ai,bi] si f(ai)*f(bi) <= 0") ;
    NEW_LINE ;
    GET_LINE(FF,"Entrez l'equation : f(x) = ") ; NEW_LINE ;
    ENTRER_REEL(a,"Valeur de a : ") ;
    ENTRER_REEL_BORNE(a,1.0E+6,b,"Valeur de b : ") ; NEW_LINE ;
    ENTRER_ENTIER(n,"Nombre de sous-intervalles : ") ; NEW_LINE ;
    h := (b - a)/FLOAT(n) ;
    ak := a ;
    k := 0 ;
    for i in 0..n - 1
    loop
       y := f(ak) ;
       z := f(ak + h) ;
       if (y = 0.0) then
          k := k + 1 ;
          PUT(IMP,"Solution ") ; PUT(IMP,k,2) ; PUT(IMP," = ") ;
          PUT(IMP,ak,5,6,0) ; NEW_LINE(IMP) ;
       end if ;
       if (y*z < 0.0) then
          DICHOTOMIE(ak,ak + h,x) ;
          k := k + 1 ;
          PUT(IMP,"Solution ") ; PUT(IMP,k,2) ; PUT(IMP," = ") ;
          PUT(IMP,x,5,6,0) ; NEW_LINE(IMP) ;
       end if ;
       ak := ak + h ;
    end loop ;
    CLOSE(IMP) ;
end DEMO_DICHOTOMIE ;
```

```
procedure DEMO_NEWTON_RAPHSON is
x0, x: FLOAT ;
k : INTEGER ;
begin
    MODE_AFFICHAGE ;
PUT_LINE("RACINE DE f(x) = 0 [a,b] PAR LA METHODE DE NEWTON-RAPHSON.");
    NEW_LINE ;
    GET_LINE(FF,"Entrez l'equation : f(x) = ") ;
    NEW_LINE ;
    ENTRER_REEL(x0,"Valeur initiale x0 : ") ;
    NEW_LINE ;
    NEWTON_RAPHSON(x0,x) ;
    PUT("Solution x = ") ; PUT(x,5,6) ;
    NEW_LINE ;
    CLOSE(IMP) ;
end DEMO_NEWTON_RAPHSON ;

begin
  loop
    CLRSCR ;
    PUT_LINE("-0- FIN") ;
    PUT_LINE("-1- Recherches des racines de f(x) = 0 par dichotomie") ;
    PUT_LINE(
    "-2- Recherche d'une racine de f(x) = 0 par la methode de Newton") ;
    NEW_LINE ;
    ENTRER_ENTIER_BORNE(0,2,Choix,"Votre choix : ") ;
    case Choix is
      when 0 => exit ;
      when 1 => DEMO_DICHOTOMIE ; PAUSE ;
      when 2 => DEMO_NEWTON_RAPHSON ; PAUSE ;
    end case ;
  end loop ;
end DEM_EQU ;
```

7.3 Spécification du paquetage SYSTEME_EQUATIONS_GENERIQUE

```
with COMMON_MATRIX, MATH2 ;
use COMMON_MATRIX, MATH2 ;

generic
  with function f(j : in POSITIVE ; x : in VARIABLE_2) return FLOAT ;

package SYSTEME_EQUATIONS_GENERIQUE is

procedure MATRICE_JACOBIENNE(x : in VARIABLE_2 ; df : out MATRICE) ;
             -- § (4.3)
procedure NEWTON_RAPHSON(x : in out VARIABLE_2) ; -- § (4.2)

ERREUR_CONVERGENCE : exception ;

end SYSTEME_EQUATIONS_GENERIQUE ;
```

7.4 Démonstration du paquetage SYSTEME_EQUATIONS_GENERIQUE

```
with TEXT_IO, IIO, FIO, CRT, COMMON_MATH0, MATH0, MATH2,
     SYSTEME_EQUATIONS_GENERIQUE ;
use TEXT_IO, IIO, FIO, CRT, COMMON_MATH0, MATH0, MATH2 ;

procedure DEM_SYST is
MAX_FONCTIONS : constant INTEGER := 10 ;
subtype INDICE_FCT is INTEGER range 1..MAX_FONCTIONS ;
type SYSTEME_EQUATIONS is array(INDICE_FCT range <>) of FONCTION ;
n : INTEGER ;
begin
    MODE_AFFICHAGE ;
    PUT_LINE ("      SYSTEME DE N EQUATIONS NON LINEAIRES") ;
    NEW_LINE ;
    PUT_LINE("           METHODE DE NEWTON_RAPHSON") ;
    NEW_LINE ;
    ENTRER_ENTIER_BORNE(1,MAX_FONCTIONS,n,"Nombre d'equations : ") ;
    NEW_LINE ;
    declare
       FF : SYSTEME_EQUATIONS(1..n) ;
       x : VARIABLE_2(1..n) ;
       function f(j : in POSITIVE ; x : in VARIABLE_2) return FLOAT is
       begin
           return EVALUE(FF(j),0.0,x) ;
       end f ;
       package SYSTEME_FF is new SYSTEME_EQUATIONS_GENERIQUE(F) ;
       use SYSTEME_FF ;
    begin
       for j in x'range
       loop
          PUT("Equation Numero ") ; PUT(j,2) ; NEW_LINE ;
          PUT("f(") ;
          for k in 1..n - 1
          loop
             PUT("y") ; PUT(k,2) ; PUT(",") ;
          end loop ;
          PUT("y") ; PUT(n,2) ; PUT_LINE(") = ") ;
          GET_LINE(ff(j),"   ") ;
       end loop ;
       NEW_LINE ;
       PUT_LINE(" Valeur initiale x") ;
       NEW_LINE ;
       for j in x'range
       loop
          PUT("Valeur initiale x") ; PUT(j,2) ; NEW_LINE ;
          ENTRER_REEL(x(j),"  x = ") ;
       end loop ;
       NEW_LINE ;
       NEWTON_RAPHSON(x) ;
       PUT_LINE("Solution") ;
       for i in x'range
```

```
       loop
           PUT(IMP,"x") ; PUT(IMP,i,2) ; PUT(IMP," = ") ;
           PUT(IMP,x(i),5,6,0) ;
           NEW_LINE(IMP) ;
       end loop ;
    end ;
    CLOSE(IMP) ;
end DEM_SYST ;
```

7.5 Spécifications des paquetages COMMON_POLY et POLY

Dans ces paquetages on définit un type POLYNOME avec des procédures d'entrée-sortie, une fonction d'évaluation (algorithme de Horner), une procédure de tracé d'une courbe polynomiale et des procédures de recherche des racines d'un polynôme.

On utilise le paquetage NOMBRES_COMPLEXES de spécification :

```
with TEXT_IO ;
use TEXT_IO ;
package NOMBRES_COMPLEXES is
type INDICE is (REEL, IMAGINAIRE) ;
type COMPLEXE is array(INDICE) of FLOAT ;

function "+"(A, B : COMPLEXE) return COMPLEXE ;
function "-"(A, B : COMPLEXE) return COMPLEXE ;
function "*"(A, B : COMPLEXE) return COMPLEXE ;
function "*"(A : FLOAT ; B : COMPLEXE) return COMPLEXE ;
function "/"(A, B : COMPLEXE) return COMPLEXE ;
function MODULE(A : COMPLEXE) return FLOAT ;
procedure PUT(z : in COMPLEXE ; FORE : in INTEGER := 3 ;
              AFT : in INTEGER := 3 ; EXP : in INTEGER := 0) ;
procedure PUT(FILE : in FILE_TYPE ; z : in COMPLEXE ;
              FORE : in INTEGER := 3 ; AFT : in INTEGER := 3 ;
              EXP : in INTEGER := 0) ;
end NOMBRES_COMPLEXES ;

package COMMON_POLY is
MAX_DEGRE : constant INTEGER := 30 ;
subtype INDICE_COEFFICIENTS is INTEGER range 0..MAX_DEGRE ;
type POLYNOME is array(INDICE_COEFFICIENTS range <>) of FLOAT ;
end COMMON_POLY ;

with TEXT_IO, COMMON_POLY, COMMON_MATRIX, NOMBRES_COMPLEXES ;
use COMMON_POLY, COMMON_MATRIX, NOMBRES_COMPLEXES ;
package POLY is
type VECTEUR_COMPLEXE  is array(INDICE_COEFFICIENTS range <>)
                               of COMPLEXE ;

procedure GET(P : out POLYNOME) ;
procedure POLYNOME_ALEATOIRE(P : out POLYNOME) ;
procedure PUT_LINE(P : in POLYNOME ; FORE : in TEXT_IO.FIELD := 6 ;
                   AFT : in TEXT_IO.FIELD := 4 ;
```

```
                    EXP : in TEXT_IO.FIELD := 0) ;
function EVALUE(P : in POLYNOME ; x : in FLOAT) return FLOAT ;
function EVALUE_DERIVEE(P : in POLYNOME ; x : in FLOAT) return FLOAT ;
procedure TRACE_POLYNOME(P : in POLYNOME ; xMin, xMax : in FLOAT) ;
procedure NEWTON_MAEHLY(P : in POLYNOME ; x : in out VECTEUR) ;
             -- § (3.2)
procedure BAIRSTOW(P : in POLYNOME ; x : out VECTEUR_COMPLEXE) ; -- § 5
ERREUR_CONVERGENCE : exception ;
end POLY ;
```

7.6 Démonstration du paquetage POLY

```
with TEXT_IO, IIO, FIO, CRT, COMMON_MATH0, MATH0, COMMON_POLY, POLY,
     NOMBRES_COMPLEXES, COMMON_MATRIX ;
use TEXT_IO, IIO, FIO, CRT, COMMON_MATH0, MATH0, COMMON_POLY, POLY,
    NOMBRES_COMPLEXES, COMMON_MATRIX ;

procedure DEM_POL is
Choix : INTEGER range 0..2 ;
procedure DEMO_NEWTON_MAEHLY is
n : INDICE_COEFFICIENTS ;
begin
    MODE_AFFICHAGE ;
    PUT_LINE("Calcul de toutes les racines reelles d'un polynome") ;
    PUT_LINE("Methode de Newton-Maehly") ;
    PUT_LINE(
        "On suppose que toutes les racines du polynome sont reelles") ;
    NEW_LINE ;
    ENTRER_ENTIER_BORNE(1,MAX_DEGRE,n,"Degre du polynome : ") ;
    declare
       P : POLYNOME(0..n) ;
       x : VECTEUR(1..n) ;
    begin
       PUT_LINE("Entrer le polynome P") ;
       GET(P) ;
       NEW_LINE ;
       NEWTON_MAEHLY(P,x) ;
       for i in x'range
       loop
          PUT(IMP,"Solution ") ; PUT(IMP,i,3) ; PUT(IMP," = ") ;
          PUT(IMP,x(i),5,6,0) ; NEW_LINE(IMP) ;
       end loop ;
   end ;
   CLOSE(IMP) ;
end DEMO_NEWTON_MAEHLY ;

procedure DEMO_BAIRSTOW is
n : INDICE_COEFFICIENTS ;
begin
    MODE_AFFICHAGE ;
    PUT_LINE("Calcul de toutes les racines complexes d'un polynome") ;
    PUT_LINE("Methode de Bairstow") ;
    NEW_LINE ;
```

```
    ENTRER_ENTIER_BORNE(1,MAX_DEGRE,n,"Degre du polynome : ") ;
    declare
       P : POLYNOME(0..n) ;
       x : VECTEUR_COMPLEXE(1..n) ;
    begin
       PUT_LINE("Entrer le polynome P") ;
       GET(P) ;
       NEW_LINE ;
       BAIRSTOW(P,x) ;
       for i in x'range
       loop
          PUT(IMP,"Solution ") ; PUT(IMP,i,3) ; PUT(IMP," = ") ;
          PUT(IMP,x(i),5,6,0) ; NEW_LINE(IMP) ;
       end loop ;
   end ;
   CLOSE(IMP) ;
end DEMO_BAIRSTOW ;

begin
  loop
    CLRSCR ;
    PUT_LINE("-0- FIN") ;
    PUT_LINE("-1- Recherches des racines reelles d'un polynome.") ;
    PUT_LINE("     Methode de Newton-Maehly") ;
    PUT_LINE("-2- Recherche des racines complexes d'un polynome") ;
    PUT_LINE("    Methode de Bairstow") ;
    NEW_LINE ;
    ENTRER_ENTIER_BORNE(0,2,Choix,"Votre choix : ") ;
    case Choix is
      when 0 => exit ;
      when 1 => DEMO_NEWTON_MAEHLY ; PAUSE ;
      when 2 => DEMO_BAIRSTOW ; PAUSE ;
    end case ;
  end loop ;
end DEM_POL ;
```

CHAPITRE 5

Approximation et interpolation

1. Introduction

Etant donné un nuage de points, $S = \{ (x_i,y_i) ; 1 \leq i \leq n \}$, on cherche une courbe qui passe, « au mieux », par ces points.

On peut demander à cette courbe de passer exactement par ces points, (c'est le cas, par exemple, pour l'interpolation polynomiale de Lagrange ou l'interpolation spline) ou alors on peut demander à cette courbe d'approcher, dans un sens à préciser, le nuage de points, (c'est le cas, par exemple, pour les méthodes des moindres carrés ou pour l'approximation polynomiale uniforme).

Dans un premier temps on étudiera dans ce chapitre les problèmes d'approximation avec les méthodes de moindres carrés, les approximants de Bézier et B-Splines. Puis on s'intéressera aux problèmes d'interpolation avec l'interpolation polynomiale de Lagrange et l'interpolation spline cubique.

Les idées sur l'interpolation polynomiale seront exploitées dans le chapitre sur le calcul numérique des intégrales.

2. Problèmes d'approximation. Méthode des moindres carrés

2.1 Introduction

Dans le cas de nuages de points expérimentaux, on n'a aucune raison d'exiger que la courbe approximative passe exactement par ces points. En effet, on peut savoir que la courbe en question sera une droite, un cercle, ... et à cause des erreurs de mesures, aucune courbe de ce type ne pourra passer par tous les points du nuage.

Un modèle de fonction étant choisi, dépendant d'un nombre fini de paramètres, il s'agira alors de déterminer ces derniers. Usuellement cette détermination se ramène à calculer le minimum d'une fonction de plusieurs variables : il s'agira de

minimiser la distance du nuage de points à la courbe définie par le modèle de fonction choisi.

Dans certains cas, comme pour les exemples du paragraphe (2.2), on a une idée du modèle qui va décrire la situation. Mais le problème est beaucoup plus délicat si on n'a aucune idée a priori de la loi décrivant le phénomène étudié.

Cet aspect des choses a été étudié par *Rufener* en 1945 dans son livre *« Mise en équation des résultats d'expériences »*.

L'idée de Rufener est, à partir d'une famille de fonctions données a priori, de rechercher, pour un nuage de points donné, le modèle qui décrira au mieux la situation. Cette recherche se faisant à partir d'une série de critères et en utilisant des tests statistiques. Une fois le modèle trouvé il s'agit de déterminer les coefficients qui donneront la courbe de meilleure approximation.

2.2 Exemples

Dans les exemples qui suivent des lois de la physique nous disent quel est le modèle à rechercher.

2.2.1 La loi d'Ohm

Si x est une intensité aux bornes d'un circuit et y la tension correspondante, on aura une relation du type $y = R \cdot x$, où la résistance R est à déterminer. Le nuage de points est alors une série de mesures (x_i,y_i) des intensités et tensions.

2.2.2 Elongation d'un ressort

Si x représente le temps et y l'élongation d'un ressort à l'instant x, on aura une relation du type $y = A \cdot \cos(\omega \cdot t + \phi)$ où A, ω et ϕ sont à déterminer.

2.2.3 Problème du fil chaud

Pour mesurer la vitesse d'écoulement d'un fluide, on utilise la « méthode du fil chaud » due à *L. V. King* en 1914.

La loi de King, qui relie la tension (en Volts) appliquée au fil à la vitesse d'écoulement du fluide (en m/s), est donnée par :

$$T^2 = a \cdot \sqrt{v} + b$$

A partir de mesures expérimentales de v et T, on peut déduire les coefficients a et b en se ramenant à un modèle affine. En effet, en posant $y = T^2$ et $x = \sqrt{v}$, la loi de King s'écrit $y = a \cdot x + b$.

2.2.4 Corrélation statistique

Il se peut aussi qu'il n'y ait aucune relation déterministe entre x et y, mais seulement une corrélation statistique.

Par exemple, en économie, si x représente une année et y le nombre de Kw/h consommés cette année-là, il n'y aura aucune relation simple entre x et y.

2.3 Détermination des paramètres

Pour chaque problème, on commence d'abord par choisir le modèle qui doit rendre compte des observations, ce qui mettra en évidence les paramètres à déterminer.

Un tel modèle de fonction sera noté $y = f(x,a_1,...,a_p)$, où les a_i sont les paramètres inconnus.

Le modèle étant choisi, on définit le *résidu* en x_i, comme l'écart entre l'observation et le modèle, soit :

$$e_i = \left| y_i - f\left(x_i, a_1, \ldots, a_p\right) \right|, \text{ pour } i = 1, \ldots, n.$$

L'idée est alors de trouver les paramètres qui vont minimiser ces résidus. Pour ce faire, on doit travailler de façon globale, c'est-à-dire définir un écart entre le nuage de points S et la fonction f choisie comme modèle. Cet écart doit tenir compte de tous les e_i. Les choix les plus simples sont :

$$E_\infty = \text{Max}\left\{ \left|e_i\right|;\ 1 \le i \le n \right\}$$

$$E_1 = \sum_{i=1}^{n} \left|e_i\right|$$

$$E_2 = \left(\sum_{i=1}^{n} e_i^2 \right)^{\frac{1}{2}}$$

qui correspondent aux trois normes classiques de R^n.

Le problème étant de minimiser l'écart global E, la première idée est d'utiliser les dérivées partielles, puisqu'une condition nécessaire d'extremum pour une fonction dérivable est la nullité de ces dernières. Cette idée va exclure les quantités E_ω et E_1 qui ont peu de chances d'être dérivables, ce qui nous conduit à utiliser E_2.

Ce choix de E_2 est donc lié à des considérations pratiques et non à des considérations théoriques.

2.4 Principes des méthodes des moindres carrés

L'idée des méthodes de moindres carrés est de trouver les paramètres a_i qui minimisent *l'écart quadratique :*

$$E\left(a_1, \ldots, a_p\right) = \sum_{i=1}^{n} e_i^2 = \sum_{i=1}^{n} \left(y_i - f\left(x_i, a_1, \ldots, a_p\right) \right)^2$$

Si f est dérivable, une condition nécessaire d'extremum est donc donnée par le système de p équations à p inconnues :

$$\frac{\partial E}{\partial a_i}\left(a_1, \ldots, a_p\right) = 0 \ (i = 1, \ldots, p) \quad (1)$$

Mais il se peut que ce système n'ait pas de solution ou qu'il ait une solution qui ne corresponde pas à un extremum pour E.

Le cas agréable sera celui où on a un système linéaire. Les dérivées partielles donneront des expressions linéaires si E est une expression quadratique des a_i, ce qui sera réalisé si les e_i sont des expressions linéaires des a_i. On parlera alors de modèles linéaires.

De manière plus générale, on peut être amené à introduire des coefficients de pondération liés aux incertitudes sur les mesures des x_i et des y_i, ce qui conduit à minimiser :

$$E = \sum_{i=1}^{n} w_i \cdot e_i^2$$

2.5 Les modèles linéaires

2.5.1 Généralités

Dans ces cas, on cherche une fonction y qui est combinaison linéaire de fonctions y_1, ..., y_p données, soit :

$$y(x) = \sum_{k=1}^{p} a_k \cdot y_k(x)$$

Les y_k pouvant être des monômes (y est alors un polynôme de degré p), des sinus et cosinus (y est alors une série harmonique), ...

2.5.2 La régression affine

On cherche, ici, le modèle sous la forme :

$$y(x) = f(x,a,b) = a \cdot x + b$$

avec le nuage de points $S = \{ (x_i,y_i) ; 1 \leq i \leq n \}$.

L'un des tests de Rufener, pour rechercher un tel modèle, est de dire que nécessairement les quotients $\frac{y_{i+1} - y_i}{x_{i+1} - x_i}$ devraient être sensiblement constants.

Remarque — Si les quotients ci-dessus sont très grands, il sera plus judicieux de chercher une droite presque verticale, c'est-à-dire chercher un modèle du type $x = \alpha \cdot y + \beta$.

Il s'agit donc de minimiser :

$$E(a,b) = \sum_{i=1}^{n} (a \cdot x_i + b - y_i)^2$$

Une condition nécessaire d'extremum est alors :

$$\begin{cases} \dfrac{\partial E}{\partial a} = 0 \\ \dfrac{\partial E}{\partial b} = 0 \end{cases} \quad (2)$$

Soit le système :

$$\begin{cases} \displaystyle\sum_{i=1}^{n} (a \cdot x_i + b - y_i) \cdot x_i = 0 \\ \displaystyle\sum_{i=1}^{n} (a \cdot x_i + b - y_i) = 0 \end{cases} \quad (3)$$

ou encore :

$$\begin{cases} \left(\displaystyle\sum_{i=1}^{n} x_i^2 \right) \cdot a + \left(\sum_{i=1}^{n} x_i \right) \cdot b = \sum_{i=1}^{n} x_i \cdot y_i \\ \left(\displaystyle\sum_{i=1}^{n} x_i \right) \cdot a + n \cdot b = \sum_{i=1}^{n} y_i \end{cases} \quad (4)$$

En notant X la variable aléatoire prenant les valeurs x_i de manière équiprobable et Y celle prenant les valeurs y_i, le système (4) s'écrit :

$$\begin{cases} E(X^2)\cdot a + E(X)\cdot b = E(X\cdot Y) \\ E(X)\cdot a + b = E(Y) \end{cases} \quad (5)$$

où on a noté :

$$E(X^k) = \frac{1}{n}\cdot\sum_{i=1}^{n} x_i^k \quad (\textit{moment d'ordre } k)$$

$$E(X\cdot Y) = \frac{1}{n}\cdot\sum_{i=1}^{n} x_i\cdot y_i$$

Le déterminant de ce système est :

$$V(X) = E(X^2) - E(X)^2 = E((X - E(X))^2) \quad (\textit{variance} \text{ de X})$$

Si $V(X) = 0$, tous les x_i valent $E(X)$ et les points sont sur une droite verticale.

En général on a donc $V(X) \neq 0$ et la solution de (5) est :

$$\begin{cases} a = \dfrac{\sigma_{XY}}{\sigma_X^2} \\ b = E(Y) - a\cdot E(X) \end{cases} \quad (6)$$

où on a noté :

$$V(X) = \sigma_X^2 \quad (\textit{écart type} \text{ de X})$$

$$\sigma_{XY} = E(X\cdot Y) - E(X)\cdot E(Y) = \frac{1}{n}\cdot\sum_{i=1}^{n}\left(x_i - E(X)\right)\cdot\left(y_i - E(Y)\right)$$

(*Covariance* des séries statistiques X et Y)

Remarque — On a $E(a',b') = E(a,b) + \sum_{i=1}^{n}\left((a'-a)\cdot x_i + (b'-b)\right)^2$, pour tous a', b' dans R et on a donc bien un minimum.

En considérant le modèle $x = \alpha\cdot y + \beta$, on aurait de la même façon :

$$\begin{cases} \alpha = \dfrac{\sigma_{XY}}{\sigma_Y^2} \\ \beta = E(X) - \alpha\cdot E(Y) \end{cases} \quad (7)$$

Les points seront effectivement alignés si ces deux droites sont identiques c'est-à-dire si, et seulement si, elles ont la même pente, ce qui se traduit par $\alpha = \frac{1}{a}$, ou encore :

$$\rho_{XY} = \frac{\sigma_{XY}}{\sigma_X\cdot\sigma_Y} = 1$$

ρ_{XY} est le *coefficient de corrélation* de X et Y, son importance étant résumée par le :

Théorème **:** (i) Si X et Y sont indépendantes, alors $\rho_{XY} = 0$.
(ii) On a toujours $|\rho_{XY}| \leq 1$, avec égalité si, et seulement si, les points sont alignés.

En pratique, si $|\rho_{XY}| > 0.75$ on recherche un alignement statistique.

Programmation structurée — On a :

$$a = \frac{1}{S(X)} \cdot \sum_{i=1}^{n} \left(x_i - E(X)\right) \cdot \left(y_i - E(Y)\right)$$

où :

$$S(x) = \sum_{i=1}^{n} \left(x_i - E(X)\right)^2$$

et en posant $t_i = x_i - E(X)$, on a aussi :

$$a = \frac{1}{S(X)} \cdot \sum_{i=1}^{n} t_i \cdot \left(y_i - E(Y)\right)$$

et avec $\sum_{i=1}^{n} t_i = n \cdot E(X - E(X)) = 0$, on déduit que :

$$a = \frac{1}{S(X)} \cdot \sum_{i=1}^{n} t_i \cdot y_i$$

Et de la même façon, on a :

$$\alpha = \frac{1}{S(Y)} \cdot \sum_{i=1}^{n} t_i' \cdot x_i$$

où $t'_i = y_i - E(Y)$ et $S(Y) = \sum_{i=1}^{n} \left(y_i - E(Y)\right)^2$.

Avec les relations $a = \frac{\sigma_{XY}}{\sigma_X^2}$ et $\rho_{XY} = \frac{\sigma_{XY}}{\sigma_X \cdot \sigma_Y}$, on déduit que

$$\sigma_{XY} = \frac{\sigma_X}{\sigma_Y} \cdot a .$$

```
PROCEDURE Regression_Affine(Entrée : n : Entier ; x, y : Vecteur ; Sortie : a, b, r : Réel) ;
Début
  a = Ex = Ey = Sx= Sy = 0 ;
  Pour i allant de 1 à n faire
  Début
    Ex = Ex + xi ; Ey = Ey + yi ;
  Fin ;
  Ex = Ex/n ;
  Ey = Ey/n ;
  Pour i allant de 1 à n faire
  Début
    t  = xi – Ex ;
    Sx = Sx + t·xi ;
    a = a + t·yi ;
    t = yi – Ey ;
    Sy = Sy + t·yi ;
  Fin ;
  a = a/Sx ;
```

$r = a \cdot \sqrt{\frac{Sx}{Sy}}$;
$b = Ey - a \cdot Ex$;
Si $|r| < 0.75$
Alors Arrêter('Modèle affine mal adapté') ;
$\{ \alpha = r^2/a ; \beta = Ex - \alpha \cdot Ey \}$
Fin ;

2.5.3 Exemple : le problème du fil chaud

Les mesures sont contenues dans $S = \{ (v_i,T_i) ; 1 \le i \le 14 \}$, avec les vitesses et tensions données par :

```
v : 5.00; 5.91; 7.07; 8.36; 9.78; 11.54; 13.47; 15.05; 16.98;
19.14; 20.81; 23.45; 26.23; 27.68.
T : 4.11; 4.22; 4.34; 4.42; 4.52; 4.63; 4.73; 4.81; 4.89; 4.98;
5.03 ; 5.12 ; 5.20 ; 5.25.
```

Tableau 5.1

La loi cherchée est de la forme : $T^2 = a \cdot \sqrt{v} + b$. En posant $y = T^2$ et $x = \sqrt{v}$, on se ramène à un modèle affine et on trouve :

Coefficients de régression : a = 3.46 ; b = 9.52

Coefficient de corrélation : $\rho = 0.9988$

2.5.4 Régression polynomiale

Position du problème — De manière plus générale, on peut parfois être amené à chercher un modèle polynomial, soit :

$$f(t,a_0,...,a_p) = a_0 + a_1 \cdot t + ... + a_p \cdot t^p$$

Un critère pour chercher un tel modèle, avec p = 2 et dans le cas de points x_i équidistants, est donné par Rufener en considérant les *différences finies d'ordre 2*.

En effet, si les points sont près d'une courbe du second degré d'équation $y = a \cdot x^2 + b \cdot x + c$, nécessairement on aura $y_{i-1} - 2 \cdot y_i + y_{i+1} = a \cdot (2 \cdot dx)^2$, où $dx = x_{i+1} - x_i$ ne dépend pas de i, c'est-à-dire que toutes ces différences finies d'ordre 2 devraient être sensiblement constantes.

De manière générale, pour un modèle polynomiale de degré p, une condition nécessaire pour que le modèle soit acceptable est que les différences finies d'ordre p soient sensiblement constantes.

On suppose donc une telle condition vérifiée et il s'agit donc de minimiser :

$$E(a_0,...,a_p) = \sum_{i=1}^{n} \left(y_i - \left(a_0 + a_1 \cdot x_i + ... + a_p \cdot x_i^p \right) \right)^2$$

Si on note y, le vecteur de R^n de coordonnées y_i, v un vecteur de coordonnées a_i dans R^{p+1} et B la matrice à n lignes et p + 1 colonnes définie par $b_{ij} = x_i^{j-1}$, pour $1 \le i \le n$ et $1 \le j \le p + 1$, alors :

$$E(a_0,...,a_p) = \| y - B \cdot v \|^2$$

où $\|z\|^2 = \sum_{i=1}^{n} z_i^2$ est la norme euclidienne de R^n.

On doit donc minimiser la fonction :

$$E:\ \begin{array}{ccc} R^{p+1} & \to & R^{+} \\ v & \mapsto & E(v) = \|y - B\cdot v\|^2 \end{array}$$

Recherche du minimum — Toujours avec la même condition nécessaire d'extremum, on aboutit au système linéaire de p + 1 équations à p + 1 inconnues :

$$\frac{\partial E}{\partial a_i} = 2\cdot\sum_{k=1}^{n}\left(a_j\cdot x_k^j\cdot - y_k\right)\cdot x_k^i = 0 \ (i = 0, 1, ..., p) \qquad (1)$$

soit, en permutant l'ordre des sommations :

$$\sum_{j=1}^{p+1} S\left(X^{j+i-2}\right)\cdot a_{j-1} = S\left(X^{i-1}\cdot Y\right) \ (i = 1, 2, ..., p + 1) \quad (2)$$

en notant :

$$S(X^{j+i-2}) = \sum_{k=1}^{n} x_k^{j+i-2} \text{ et } S(X^{i-1}\cdot Y) = \sum_{k=1}^{n} x_k^{i-1}\cdot y_k$$

Le système (2) est appelé *système d'équations normales*.

En notant :

$$\beta_{ij} = S(X^{j+i-2}) = \sum_{k=1}^{n} x_k^{j+i-2} \ ; \gamma_i = S(X^{i-1}\cdot Y) = \sum_{k=1}^{n} x_k^{i-1}\cdot y_k \quad (1 \le i, j \le p + 1)$$

le système (2) s'écrit :

$$\beta\cdot v = \gamma$$

On peut en fait montrer que la condition (2) est aussi suffisante. Pour ce faire, on remarque que :

$$(3)\ \beta = {}^tB\cdot B \text{ et } \gamma = {}^tB\cdot y$$

avec :

$$B = \begin{pmatrix} 1 & x_1 & x_1^2 & .... & x_1^p \\ 1 & x_2 & x_2^2 & .... & x_2^p \\ . & . & . & . & . \\ . & . & . & . & . \\ 1 & x_n & x_n^2 & .... & x_n^p \end{pmatrix} ; \quad {}^tB = \begin{pmatrix} 1 & 1 & 1 & .... & 1 \\ x_1 & x_2 & x_3 & .... & x_n \\ . & . & . & . & . \\ . & . & . & . & . \\ x_1^p & x_2^p & x_3^p & .... & x_n^p \end{pmatrix}$$

puis en écrivant que pour tous u, v dans R^{p+1} , on a :

$$E(u + v) = E(v) + 2\cdot\left\langle {}^tB\cdot B\cdot v - {}^t B\cdot y \middle| u\right\rangle + \|B\cdot u\|^2$$

on déduit que si v est solution de (2), alors :

$$\forall u \in R^{p+1}: E(u + v) = E(v) + \|B\cdot u\|^2 \ge E(v)$$

l'égalité étant réalisée si, et seulement si, u = 0. Ce qui prouve que E admet un minimum en v.

Donc (2) est bien une condition nécessaire et suffisante d'extremum, ce qui se résume par le :

Théorème **:** Un vecteur v de R^{p+1} est solution du problème de minimisation de E, si et seulement si, il est solution du système linéaire : ${}^tB\cdot B\cdot v = {}^tB\cdot y$.

Remarque 1 — On peut vérifier que, si n > p et au moins p + 1 x_i sont distincts, alors la matrice ${}^tB \cdot B$ est symétrique définie positive (cf. [Ciarlet] p. 69) et donc le problème (1) admet une unique solution.

Remarque 2 — Souvent on constate que Dét(${}^tB \cdot B$) est proche de 0, ce qui signifie que le système (1) est mal conditionné, c'est-à-dire qu'une petite variation des coefficients de B peut fortement perturber la solution numérique trouvée. Par exemple, si les x_i sont régulièrement espacés sur [0,1] alors la matrice ${}^tB \cdot B$ est proportionnelle à une matrice de Hilbert d'ordre p + 1, dont le déterminant décroît très vite vers zéro quand n augmente.

De manière pratique, le système donnera des résultats médiocres dès que p > 7. On aura donc intérêt à travailler en double précision.

Remarque 3 — Si p + 1 = n et si B est inversible, alors les équations normales vont s'écrire B·u = y.

Résolution du système d'équations normales — Pour résoudre le système d'équations normales on pourra utiliser la méthode des pivots de Gauss ou la méthode de Cholesky vues dans le chapitre « Analyse Numérique Linéaire ».

Ce qui donne la procédure ci-dessous où v est le vecteur contenant les coefficients du polynôme de régression. La matrice du système d'équations normales est notée β et le second membre γ.

Tout d'abord, on a une procédure de construction du système linéaire :

```
PROCEDURE SystemeRegression(Entrée n, p : Entiers ; x, y : Vecteurs ;
                                    Sortie β : Matrice ; γ : Vecteur) ;
Début
   Pour i Allant de 1 à p + 1 faire
   Début
      Pour j Allant de 1 à p + 1 Faire β_ij = 0 ;
      γ_i = 0 ;
   Fin ;
   β_11 = n ;
   γ_1 = y_1 ;
   Pour k allant de 2 à n Faire γ_1 = γ_1 + y_k ;
   Pour i allant de 2 à p + 1 Faire
   Début
      γ_i = y_i*x_1^(i–1) ;
      Pour k allant de 2 à n Faire γ_i = γ_i + y_k*x_k^(i–1) ;
      Pour j allant de 1 à i Faire
      Début
         β_ij = x_1^(i+j–2) ;
         Pour k allant de 2 à n Faire β_ij = β_ij + x_k^(i+j–2) ;
      Fin ;
   Fin ;
   Pour i Allant de 1 à p Faire
   Début
      Pour j Allant de i + 1 à p + 1 Faire β_ij  = β_ji ;
   Fin ;
```

```
Fin ;

PROCEDURE RegressionPolynomiale(Entrée : p, n : Entier ; x, y : Vecteur ;
                                Sortie v : Vecteur) ;
Début
   SystemeRegression(n,p,x,y,β,γ) ;
   Pivotage(p + 1,β,γ) ;
   SystTriSup(p + 1,β,γ,v) ;
Fin ;
```

Il suffit ensuite de définir une fonction polynôme à partir de u, en posant :

$$P(t) = u_1 + u_2 \cdot t + ... + u_{p+1} \cdot t^p$$

2.5.5 Cas général

Les fonctions de base, linéairement indépendantes, étant données, on cherche à minimiser E. De la même façon que dans le cas polynomiale, on peut vérifier qu'un vecteur u de composantes $(a_1,...,a_p)$ est solution du problème de minimisation si, et seulement si, il est solution du système d'équations normales : ${}^tB \cdot B \cdot u = {}^tB \cdot y$ où $B = \left(\left(f_j(x_i)\right)\right)_{\substack{1 \le i \le n \\ 1 \le j \le p}}$, ce qui donne :

$$\begin{cases} \beta = {}^tB \cdot B = \left(\left(\beta_{ij}\right)\right)_{1 \le i,j \le p} \\ \gamma = {}^t B \cdot y = \left(\left(\gamma_i\right)\right)_{1 \le i \le p} \end{cases} \text{ avec : } \begin{cases} \beta_{ij} = \sum_{k=1}^{n} f_j(x_k) \cdot f_i(x_k) \\ \gamma_i = \sum_{k=1}^{n} y_k \cdot f_i(x_k) \end{cases}$$

La matrice β étant symétrique. Si B est de rang p, alors β est définie positive et le système d'équations normales a une solution et une seule. Dans tous les cas, le système a au moins une solution.

On a le résultat suivant :

> *Théorème* **:** Si les fonctions de base $\left(y_i\right)_{1 \le i \le p}$ sont linéairement indépendantes et s'il y a au moins p points distincts dans les x_i alors B est de rang p et le système d'équations normales admet une unique solution v. C'est-à-dire que le problème d'extremum admet une unique solution.

Remarque **—** S'il y a p points distincts parmi les n, il peut exister plusieurs points de même abscisse.

Pour la programmation structurée, on procédera alors comme dans le cas polynomiale en remplaçant les x^k, pour k = 0, ..., p par les $y_k(x)$, pour k = 1, ..., p.

2.5.6 Régression trigonométrique

On considère ici le modèle :

$$f(x,a_0,a_1,b_1,...,a_p,b_p) = a_0 + a_1 \cdot \cos(\omega \cdot x) + b_1 \cdot \sin(\omega \cdot x) + ... + a_p \cdot \cos(p \cdot \omega \cdot x) + b_p \cdot \sin(p \cdot \omega \cdot x)$$

où ω est connu (i.e. la période $T = \frac{2 \cdot \pi}{\omega}$ est connue).

La méthode décrite précédemment donne donc un moyen de calculer les coefficients de Fourier d'une fonction périodique échantillonnée.

Dans le cas particulier où les x_i sont équidistants, on peut déterminer assez facilement les a_k et b_k.

Par changement de variable, on peut toujours se ramener à une fonction $2\cdot\pi$-périodique, c'est-à-dire à $\omega = 1$.

On suppose donc que les x_k sont donnés par :

$$x_k = \frac{k \cdot \pi}{n} \quad (k = -n + 1, ..., n)$$

avec $n \geq p$ (ce sont donc des points équidistants sur $]-\pi,\pi]$).

Il s'agit donc de minimiser :

$$E(a_0,...,a_p,b_1,...,b_p) = \sum_{i=-n+1}^{n} \left(y_i - a_0 - \sum_{k=1}^{p} \left(a_k \cdot \cos(k \cdot x_i) + b_k \cdot \sin(k \cdot x_i) \right) \right)^2$$

En annulant les dérivées partielles, on aboutit alors au système linéaire de $2\cdot n + 1$ équations à $2\cdot n + 1$ inconnues :

$$\sum_{i=-n+1}^{n} \left(y_i - a_0 - \sum_{k=1}^{p} \left(a_k \cdot \cos(k \cdot x_i) + b_k \cdot \sin(k \cdot x_i) \right) \right) \cdot \cos(j \cdot x_i) = 0$$

$$\sum_{i=-n+1}^{n} \left(y_i - a_0 - \sum_{k=1}^{p} \left(a_k \cdot \cos(k \cdot x_i) + b_k \cdot \sin(k \cdot x_i) \right) \right) \cdot \sin(j \cdot x_i) = 0$$

avec j variant de 0 à p dans les premières équations et de 1 à p dans les dernières.

En exploitant les propriétés des fonctions trigonométriques, on déduit alors que :

$$a_0 = \frac{1}{2 \cdot n} \cdot \sum_{i=-n+1}^{n} y_i$$

$$a_k = \frac{1}{n} \cdot \sum_{i=-n+1}^{n} y_i \cdot \cos(k \cdot x_i) \quad (k = 1, ..., n)$$

$$b_k = \frac{1}{n} \cdot \sum_{i=-n+1}^{n} y_i \cdot \sin(k \cdot x_i) \quad (k = 1, ..., n)$$

(Cf. Pichat, ..., p.118).

Ces formules sont simplement les approximations des coefficients de Fourier en utilisant la méthode des rectangles à droite pour le calcul des intégrales. On pourrait utiliser la méthode de Simpson ou de Romberg (Cf. le chapitre « Calcul Numérique des Intégrales ») pour avoir une meilleure précision, mais cela augmentera le temps de calcul qui est déjà important du fait de la présence des fonctions trigonométriques.

En écrivant les choses sous forme complexe, on aura un algorithme beaucoup plus performant de calcul, c'est l'algorithme FFT (pour Fast Fourier Transform) de Cooley et Tukey (Cf. le chapitre sur le calcul numérique des intégrales).

2.6 Les modèles non linéaires

Ce sont les modèles qui vont se ramener au cas linéaire par changement de variable. On a déjà vu un exemple avec le problème du fil chaud.

Exemple 1 — $f(x) = a\cdot\cos(\omega\cdot x + \phi) + b$, où ω est connue et a, ϕ et b sont à déterminer.

En écrivant $f(x) = a\cdot\cos(\phi)\cdot\cos(\omega\cdot x) - a\cdot\sin(\phi)\cdot\sin(\omega\cdot x) + b$, on se ramène à la régression trigonométrique.

Exemple 2 — $f(x) = a\cdot e^{-\alpha\cdot t}$, en utilisant le logarithme népérien on se ramène au cas linéaire :

$$\mathrm{Ln}(f(x)) = -\alpha\cdot t + \mathrm{Ln}(a)$$

Pour les modèles non linéaires qui résistent à ce genre d'astuce, il existe des méthodes de linéarisation. On écrit le système $\frac{\partial E}{\partial a_i} = 0$, que l'on essaye de résoudre par une méthode de type Newton.

2.7 Un exemple d'application : détermination du coefficient de traînée d'une particule sphérique

On considère une particule sphérique de masse m, de diamètre d et de densité volumique ρ qu'on laisse tomber, à partir de t = 0 et sans vitesse initiale, dans un fluide au repos de densité volumique ρ_f et de viscosité cinématique ν.

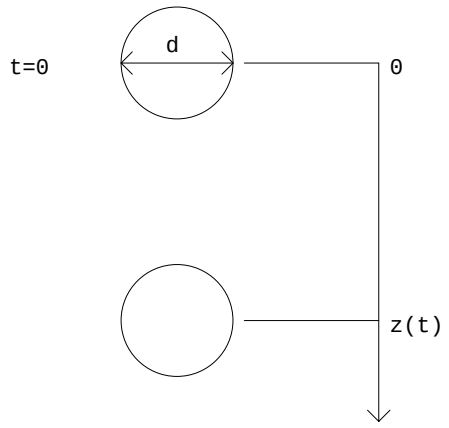


Figure 5.1

La *force de réaction*, due à la viscosité du fluide, est alors donnée par :

$$F_r = -C_d\left(v(t)\right)\cdot\frac{1}{2}\cdot\rho_f\cdot v(t)\cdot\left|v(t)\right|\cdot A$$

où, v(t) est la vitesse de la particule à l'instant t, $A = \pi\cdot\left(\frac{d}{2}\right)^2$ et $C_d(v(t))$ est le *coefficient de traînée* de la particule.

Le coefficient de traînée peut s'exprimer en fonction du nombre de Reynolds donné, dans le cas d'une particule sphérique, par :

$$R_e(t) = \frac{v(t) \cdot d}{\nu}$$

L'expression de C_d en fonction de R_e est connue seulement de façon empirique, avec des formules valables seulement dans des régions déterminées des R_e.

Dans le cas d'une sphère, on a les lois classiques suivantes :

(i) loi de Stokes — $C_d = \frac{24}{R_e} \quad (0 \le R_e \le 0.1)$ (certains auteurs prennent R_e dans [0,1]) ;

(ii) loi d'Olson — $C_d = \frac{24}{R_e} \cdot \left(1 + \frac{3}{16} \cdot R_e\right)^{\frac{1}{2}} \quad (0.1 \le R_e \le 100)$

De manière un peu plus générale, Morsi propose des formules du type :

$$C_d = K_0 + \frac{K_1}{R_e} + \frac{K_2}{R_e^2} \quad (a \le R_e \le b)$$

avec des valeurs spécifiques des coefficients K_i dans les intervalles suivants :
[0,0.1], [0.1,1], [1,10], [10,100], [100,1000], [1000,5000], [5000,10000] et [10000,50000].

Une idée pour trouver ces coefficients est de partir d'un tableau de mesures expérimentales du type ci-dessous et de procéder par interpolations polynomiales.

R_e :	0.1	0.2	0.3	0.5	0.7	1
C_d :	240	120	80	49	36.5	26.5
R_e :	1	2	3	5	7	10
C_d :	26.5	14.4	10.4	6.9	5.4	4.1
R_e :	10	20	30	50	70	100
C_d :	4.1	2.55	2	1.5	1.27	1.07
R_e :	100	200	300	500	700	1000
C_d :	1.07	0.77	0.65	0.55	0.5	0.46
R_e :	1000	2000	3000	5000		
C_d :	0.46	0.42	0.4	0.385		
R_e :	5000	7000	10000			
C_d :	0.385	0.39	0.405			
R_e :	10000	20000	30000	50000		
C_d :	0.405	0.45	0.47	0.49		

Tableau 5.2

Dans le programme ci-dessous, on adopte les notations suivantes :

Dans l'intervalle N° j, les mesures de Re et Cd sont notées Re(j,i) et Cd(j,i) où $1 \leq j \leq$ Nombre_Intervalles = 7 et $1 \leq i \leq n(j)$, avec n(j) égal au nombre de mesures dans cet intervalle et $n(j) \leq$ Nombre_Mesures = 6.

Dans l'intervalle N° j, les coefficients de Morsi sont notés K(j,i), où i = 1, 2, 3.

La procédure DONNEES_DU_FICHIER permet de lire les données du problème depuis un fichier.

On notant, pour j donné, u(i) = 1/Re(j,i) et v(i) = Cd(j,i), avec $1 \leq i \leq n(j) = n$, on est amené à chercher un polynôme de régression de degré au plus 2 défini par les vecteurs u et v.

La formation du système d'équations normales est réalisée par la procédure Equations_Normales. Ce système est défini par :

$$M = \begin{pmatrix} n & a & b \\ a & b & c \\ b & c & d \end{pmatrix} \text{ et } \gamma = \begin{pmatrix} \gamma_1 \\ \gamma_2 \\ \gamma_3 \end{pmatrix}$$

avec :

$$a = \sum_{k=1}^{n} u_k ;\ b = \sum_{k=1}^{n} u_k^2 ;\ c = \sum_{k=1}^{n} u_k^3 ;\ d = \sum_{k=1}^{n} u_k^4 ;$$

$$\gamma_1 = \sum_{k=1}^{n} v_k ;\ \gamma_2 = \sum_{k=1}^{n} u_k \cdot v_k ;\ \gamma_3 = \sum_{k=1}^{n} u_k^2 \cdot v_k ;$$

La solution de ce système est alors donnée par :

$$w_1 = \frac{\delta_1}{\delta} ;\ w_2 = \frac{\delta_2}{\delta} ;\ w_3 = \frac{\delta_3}{\delta} ;$$

avec :

$$\delta = n\cdot(b\cdot d - c^2) + 2\cdot a\cdot b\cdot c - b^3 - a^2\cdot d$$
$$\delta_1 = \gamma_1\cdot(b\cdot d - c^2) + \gamma_2\cdot(b\cdot c - a\cdot d) + \gamma_3\cdot(a\cdot c - b^2)$$
$$\delta_2 = \gamma_1\cdot(b\cdot c - a\cdot d) + \gamma_2\cdot(n\cdot d - b^2) + \gamma_3\cdot(a\cdot b - n\cdot c)$$
$$\delta_3 = \gamma_1\cdot(a\cdot c - b^2) + \gamma_2\cdot(b\cdot a - n\cdot c) + \gamma_3\cdot(n\cdot b - a^2)$$

Pour ce qui est du tracé graphique, étant donné que R_e prendra de très grandes valeurs et C_d des valeurs beaucoup plus petites, il est préférable de tracer le graphe de $Ln(C_d)$ en fonction de $Ln(R_e)$.

```
with TEXT_IO, IIO, FIO, CRT, COMMON_MATH0, MATH0, MATH1, MATH3, CURVE ;
procedure BILLE1 is
use TEXT_IO, IIO, FIO, CRT, COMMON_MATH0, MATH0, MATH1, MATH3, CURVE ;
RE_MAX : constant FLOAT := 50000.0 ;
NOMBRE_INTERVALLES : constant NATURAL := 7 ;
NOMBRE_MESURES : constant NATURAL := 6 ;
type MATRICE_MESURES is array(  POSITIVE range 1..NBR_INTERVALLES,
                                POSITIVE range 1..NBR_MESURES) of FLOAT ;
type VECTEUR_COEFF is array(POSITIVE range 1..NBR_INTERVALLES,
                            NATURAL range 0..2) of FLOAT ;
type VECTEUR_ENTIER is array(POSITIVE range 1..NBR_INTERVALLES)
                            of NATURAL ;
type VECTEUR is array(POSITIVE range 1..NBR_MESURES) of FLOAT ;
```

```
type VECTEUR_3 is array(POSITIVE range 1..3) of FLOAT ;
Re, Cd : MATRICE_MESURES ;
nn : VECTEUR_ENTIER ;
K : VECTEUR_COEFF ;
Rho, Rhof, g, Nu, d, h, t0, tMax, z0, v0 : FLOAT ;



procedure DONNEES_DU_FICHIER(t0,tMax,z0,v0,Rho,Rhof,Nu,g,
                         d : out FLOAT ;   nn : in out VECTEUR_ENTIER ;
                         Re, Cd : out MATRICE_MESURES) is
Nom : STRING(1..50) ;
FichierDonnees : FILE_TYPE ;
begin
    PUT_LINE("   Donnees du fichier") ;
    LIRE_FICHIER(FichierDonnees,Nom) ;
    GET(FichierDonnees,t0) ;
    GET(FichierDonnees,tMax) ;
    GET(FichierDonnees,z0) ;
    GET(FichierDonnees,v0) ;
    GET(FichierDonnees,Rho) ;
    GET(FichierDonnees,Rhof) ;
    GET(FichierDonnees,Nu) ;
    GET(FichierDonnees,g) ;
    GET(FichierDonnees,d) ;
    SKIP_LINE(FichierDonnees) ;
    for j in 1..NOMBRE_INTERVALLES loop
       GET(FichierDonnees,nn(j)) ;
       SKIP_LINE(FichierDonnees) ;
       for i in 1..nn(j) loop
          GET(FichierDonnees,Re(j,i)) ;
       end loop ;
       SKIP_LINE(FichierDonnees) ;
       for i in 1..nn(j) loop
          GET(FichierDonnees,Cd(j,i)) ;
       end loop ;
       SKIP_LINE(FichierDonnees) ;
    end loop ;
    CLOSE(FichierDonnees) ;
end DONNEES_DU_FICHIER ;

procedure AFFICHAGE_DONNEES is
begin
    CLRSCR ;
    PUT_LINE(IMP,"            Donnees du probleme") ; NEW_LINE(IMP) ;
    PUT_LINE(IMP,"Conditions initiales :") ;
    PUT(IMP,"                       t0 = ") ;
    PUT(IMP,t0,6,3,0) ; NEW_LINE(IMP) ;
    PUT(IMP,"                       tMax = ") ;
    PUT(IMP,tMax,6,3,0) ; NEW_LINE(IMP) ;
    PUT(IMP,"                       z0 = ") ;
    PUT(IMP,z0,6,3,0) ; NEW_LINE(IMP) ;
    PUT(IMP,"                       v0 = ") ;
```

```
    PUT(IMP,v0,6,3,0) ; NEW_LINE(IMP,2) ;
    PUT_LINE(IMP,"Caracteristtiques de la bille et du fluide :") ;
    NEW_LINE(IMP) ;
    PUT(IMP,"                      Rho = ") ;
    PUT(IMP,Rho,6,3,0) ; NEW_LINE(IMP) ;
    PUT(IMP,"                      Rhof = ") ;
    PUT(IMP,Rhof,6,3,0) ; NEW_LINE(IMP) ;
    PUT(IMP,"                      Nu = ") ;
    PUT(IMP,Nu,6,6,0) ; NEW_LINE(IMP) ;
    PUT(IMP,"                      g = ") ;
    PUT(IMP,g,6,3,0) ; NEW_LINE(IMP) ;
    PUT(IMP,"                      d = ") ;
    PUT(IMP,d,6,3,0) ; NEW_LINE(IMP) ;
    PAUSE ; CLRSCR ;
end AFFICHAGE_DONNEES ;

function Cd_j(K : in VECTEUR_COEFF ; j : in INTEGER ; x : in FLOAT)
          return FLOAT is
-- Evaluation du polynome definissant la fonction Cd sur l'intervalle j
begin
    return K(j,0) + K(j,1)*x + K(j,2)*x*x ;
end Cd_j ;

procedure EQUATIONS_NORMALES(n : in INTEGER ; u, v : in VECTEUR ;
                             a, b, c, d : in out FLOAT ;
                             GAMMA in out VECTEUR_3) is
e : FLOAT ;
begin
    a := u(1) ; b := a*u(1) ; c := b*u(1) ; d := c*u(1) ;
    Gamma(1) := v(1) ;
    Gamma(2) := Gamma(1)*u(1) ;
    Gamma(3) := Gamma(2)*u(1) ;
    for k in 2..n loop
        e := u(k) ;
        a := a + e ;
        e := e*u(k) ;
        b := b + e ;
        e := e*u(k) ;
        c := c + e ;
        e := e*u(k) ;
        d := d + e ;
        e := v(k) ;
        Gamma(1) := Gamma(1) + e ;
        e := e*u(k) ;
        Gamma(2) := Gamma(2) + e ;
        e := e*u(k) ;
        Gamma(3) := Gamma(3) + e ;
    end loop ;
end EQUATIONS_NORMALES ;

procedure RESOLUTION_EQU_NORMALES(n : in INTEGER ;
                                  a, b, c, d : in FLOAT ;
                                  Gamma : in out VECTEUR_3) is
```

```
DD, D1, D2, D3 : FLOAT ;
begin
    DD := FLOAT(n)*(b*d - c*c) + 2.0*a*b*c - b*b*b - a*a*d ;
    D1 := Gamma(1)*(b*d - c*c) + Gamma(2)*(b*c - a*d)
          + Gamma(3)*(a*c - b*b) ;
    D2 := Gamma(1)*(b*c - a*d) + Gamma(2)*(FLOAT(n)*d - b*b)
          + Gamma(3)*(a*b - FLOAT(n)*c) ;
    D3 := Gamma(1)*(a*c - b*b) + Gamma(2)*(a*b - FLOAT(n)*c)
          + Gamma(3)*(FLOAT(n)*b - a*a) ;
    Gamma(1) := D1/DD ; Gamma(2) := D2/DD ; Gamma(3) := D3/DD ;
end RESOLUTION_EQU_NORMALES ;

procedure CALCUL_COEFF_MORSI(Re, Cd: in MATRICE_MESURES ;
                             K : out VECTEUR_COEFF) is
a, b, c, d : FLOAT ;
Gamma : VECTEUR_3 ;
u, v : VECTEUR ;
begin
    for j in 1..NOMBRE_INTERVALLES loop
       for i in 1..nn(j) loop
            u(i) := 1.0/Re(j,i) ;
            v(i) := Cd(j,i) ;
       end loop ;
       EQUATIONS_NORMALES(nn(j),u,v,a,b,c,d,Gamma) ;
       RESOLUTION_EQU_NORMALES(nn(j),a,b,c,d,Gamma) ;
       for i in 0..2 loop
         K(j,i) := Gamma(i + 1) ;
       end loop ;
    end loop ;
end CALCUL_COEFF_MORSI ;

procedure AFICCHE_COEFF_MORSI(Re, Cd : in MATRICE_MESURES ;
                              K : in VECTEUR_COEFF) is
begin
    for j in 1..NOMBRE_INTERVALLES
    loop
       PUT_LINE(IMP,"Intervalle Numero " & INTEGER'IMAGE(j)) ;
       for i in 0..2 loop
   PUT(IMP,"K(" & INTEGER'IMAGE(j) & "," & INTEGER'IMAGE(i) & ") = ") ;
          PUT(IMP,k(j,i),3,3,0) ; NEW_LINE(IMP) ;
       end loop ;
      PUT_LINE(IMP,"Re             Cd (Experimental)   Cd (Calcule)") ;
       for i in 1..nn(j) loop
          PUT(IMP,Re(j,i),6,3,0) ;
          PUT(IMP,"      ") ;
          PUT(IMP,Cd(j,i),6,3,0) ;
          PUT(IMP,"     ") ;
          PUT(IMP,Cd_j(K,j,1.0/Re(j,i)),6,3,0) ; NEW_LINE(IMP) ;
       end loop ;
       Pause ;
    end loop ;
end AFICCHE_COEFF_MORSI ;
```

```
procedure COURBE_CD(Re, Cd : in MATRICE_MESURES ;
                    K : in VECTEUR_COEFF ; CC : in out Courbe) is
-- Definition de la courbe Ln(Cd) en fonction de Ln(Re).
Pas, Pas_r, r : FLOAT ;
i : INTEGER ;
begin
    i := 0 ;
    r := 0.01 ;
    CC.uMin := Ln(r) ;
    CC.uMax := Ln(RE_MAX) ;
    CC.vMin := 1.0E+37 ;
    CC.vMax := -1.0E+37 ;
    CC.u(0) := CC.uMin ;
    CC.v(0) := Ln(Cd_j(K,1,1.0/r)) ;
    Pas := (CC.uMax - CC.uMin)/FLOAT(CC.n) ;
    Pas_r := Exp(Pas) ;
    r := r*Pas_r ;
    for j in 1..NOMBRE_INTERVALLES loop
        while (r <= Re(j,nn(j))) loop
             i := i + 1 ;
             CC.u(i) := CC.u(i-1) + Pas ;
             CC.v(i) := Ln(Cd_j(K,j,1.0/r)) ;
             CC.vMin := Min(CC.vMin,CC.v(i)) ;
             CC.vMax := Max(CC.vMax,CC.v(i)) ;
             r := r*Pas_r ;
        end loop ;
    end loop ;
end COURBE_CD ;

procedure TRACE_CD(Re, Cd : in MATRICE_MESURES) is
CC : Courbe(200) ;
begin
    COURBE_CD(Re,Cd,K,CC) ;
    Trace_Une_Courbe(CC) ;
    for j in 1..NOMBRE_INTERVALLES loop
        for i in 1..nn(j) loop
            CROIX(Ln(Re(j,i)),Ln(Cd(j,i))) ;
        end loop ;
    end loop ;
    SORTIE_GRAPHIQUE ;
end TRACE_CD ;

begin
    MODE_AFFICHAGE ;
    DONNEES_DU_FICHIER(t0,tMax,z0,v0,Rho,Rhof,Nu,g,d,nn,Re,Cd) ;
    AFFICHAGE_DONNEES ;
    PUT_LINE("Calcul des coefficients de regression") ;
    CALCUL_COEFF_MORSI(Re,Cd,K) ;
    AFICCHE_COEFF_MORSI(Re,Cd,K) ;
    CLRSCR ;
    PUT_LINE(" Trace de la courbe Ln(Cd) en fonction de Ln(Re)") ;
    PUT_LINE(" Patience je calcul ...") ;
    TRACE_CD(Re,Cd) ;
```

```
    CLOSE(IMP) ;
end BILLE1 ;
```

Les résultats obtenus sont alors les suivants :

Conditions initiales —

t0 = 0.000
tMax = 10.000
z0 = 0.000
v0 = 0.000

Caractéristiques de la bille et du fluide —

Rho = 8000.000
Rhof = 1.220
Nu = 0.000015
g = 9.800
d = 0.010

Calculs des coefficients de régression

Intervalle Numero 1 — K(1,0) = 3.486 K(1,1) = 22.782 K(1,2) = 0.087

Re	Cd (Expérimental)	Cd (Calculé)
0.100	240.000	240.051
0.200	120.000	119.583
0.300	80.000	80.398
0.500	49.000	49.400
0.700	36.500	36.211
1.000	26.500	26.356

Intervalle Numero 2 — K(2,0) = 1.538 K(2,1) = 26.943 K(2,2) = –1.995

Re	Cd (Expérimental)	Cd (Calculé)
1.000	26.500	26.486
2.000	14.400	14.511
3.000	10.400	10.297
5.000	6.900	6.847
7.000	5.400	5.346
10.000	4.100	4.213

Intervalle Numero 3 — K(3,0) = 0.687 K(3,1) = 41.253 K(3,2) = –71.552

Re	Cd (Expérimental)	Cd (Calculé)
10.000	4.100	4.097
20.000	2.550	2.571
30.000	2.000	1.983
50.000	1.500	1.484
70.000	1.270	1.262
100.000	1.070	1.093

Intervalle Numero 4 — K(4,0) = 0.377 K(4,1) = 88.621 K(4,2) = –1929.682

Re	Cd (Expérimental)	Cd (Calculé)
100.000	1.070	1.070
200.000	0.770	0.771
300.000	0.650	0.650
500.000	0.550	0.546
700.000	0.500	0.499
1000.000	0.460	0.463

Intervalle Numero 5 — K(5,0) = 0.357 K(5,1) = 146.799
K(5,2) = –43665.282

Re	Cd (Expérimental)	Cd (Calculé)
1000.000	0.460	0.460
2000.000	0.420	0.419
3000.000	0.400	0.401
5000.000	0.385	0.385

Intervalle Numero 6— K(6,0) = 0.478 K(6,1) = –987.500
K(6,2) = 2625000.000

Re	Cd (Expérimental)	Cd (Calculé)
5000.000	0.385	0.385
7000.000	0.390	0.390
10000.000	0.405	0.405

Intervalle Numero 7 — K(7,0) = 0.522 K(7,1) = -1725.126
K(7,2) = 5556952.339

Re	Cd (Expérimental)	Cd (Calculé)
10000.000	0.405	0.405
20000.000	0.450	0.450
30000.000	0.470	0.471
50000.000	0.490	0.490

Enfin le graphe de Ln(C) en fonction de Ln(R) est donné sur la figure 5.2 :

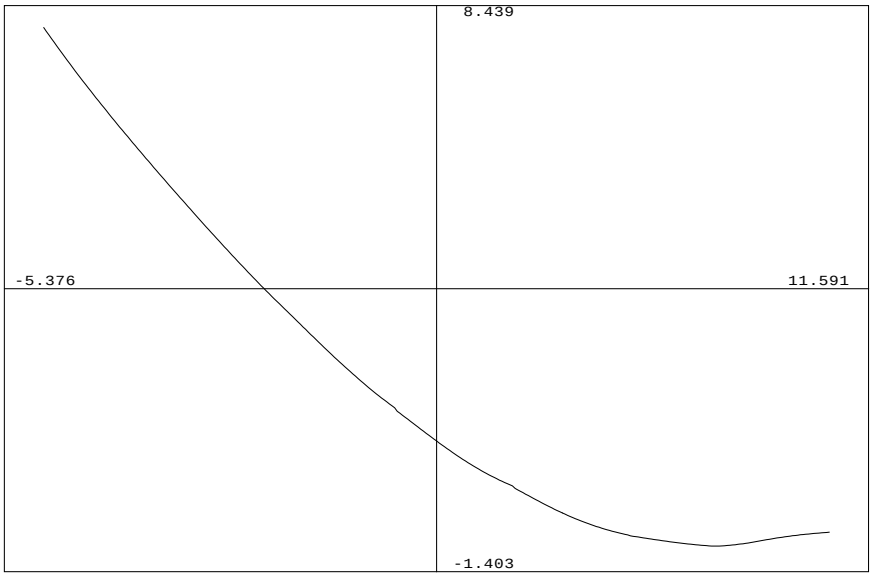


Figure 5.2

2.8 Calcul du cercle des moindres carrés

2.8.1 Notations et hypothèses

On dispose d'un nuage de points expérimentaux :

$$S = \{ M_i = (x_i,y_i) \; ; \; 1 \le i \le n \}$$

qui sont supposés être sur un cercle de centre et de rayon inconnus.

Un cercle sera décrit par une équation implicite de la forme :

$$(1) \qquad x^2 + y^2 + a \cdot x + b \cdot y + c = 0$$

C'est le cercle de centre Ω et de rayon R, avec :

$$\Omega = \left(\frac{-a}{2}, \frac{-b}{2} \right) \text{ et } R = \sqrt{\frac{a^2 + b^2}{4} - c}$$

On prendra pour écart entre le nuage S et un cercle d'équation (1), la quantité :

$$E(a,b,c) = \sum_{i=1}^{n} \left(x_i^2 + y_i^2 + a \cdot x_i + b \cdot y_i + c \right)^2$$

2.8.2 Calcul du cercle des moindres carrés

En écrivant les conditions nécessaires d'extremum, on est alors amené à résoudre le système linéaire :

$$\begin{cases} E\left(X^2\right) \cdot a + E\left(X \cdot Y\right) \cdot b + E\left(X\right) \cdot c = -E\left(X^3\right) - E\left(X \cdot Y^2\right) \\ E\left(X \cdot Y\right) \cdot a + E\left(Y^2\right) \cdot b + E\left(Y\right) \cdot c = -E\left(X^2 \cdot Y\right) - E\left(Y^3\right) \\ E\left(X\right) \cdot a + E\left(Y\right) \cdot b + c = -E\left(X^2\right) - E\left(Y^2\right) \end{cases}$$

où on a noté :

$$E\left(X^p \cdot Y^q\right) = \frac{1}{n} \cdot \sum_{i=1}^{n} x_i^p \cdot y_i^q \quad (p,\, q \ge 0)$$

En retranchant à la ligne 1, la ligne 3 multipliée par E(X) et à la ligne 2, la ligne 3 multipliée par E(Y), on obtient le système :

$$\begin{cases} V(X)\cdot a + Cov(X,Y)\cdot b = -Cov\left(X, X^2+Y^2\right) \\ Cov(X,Y)\cdot a + V(Y)\cdot b = -Cov\left(X^2+Y^2, Y\right) \end{cases}$$

le déterminant de ce système étant :

$$\Delta = V(X)\cdot V(Y)\cdot(1-\rho(X,Y)^2)$$

où $\rho(X,Y)$ est le coefficient de corrélation de X et Y.

Ce déterminant est donc nul si, et seulement si, les points sont alignés, ce qui n'est pas le cas en général pour ce type de problème.

Notre système aura donc une unique solution donnée par :

$$\begin{cases} a = \dfrac{\alpha}{\Delta} \\ b = \dfrac{\beta}{\Delta} \\ c = -\left(E(X)\cdot a + E(Y)\cdot b + E\left(X^2+Y^2\right)\right) \end{cases}$$

où :

$$\begin{cases} \alpha = Cov(X,Y)\cdot Cov\left(X^2+Y^2, Y\right) - V(Y)\cdot Cov\left(X^2+Y^2, X\right) \\ \beta = Cov(X,Y)\cdot Cov\left(X^2+Y^2, X\right) - V(X)\cdot Cov\left(X^2+Y^2, Y\right) \end{cases}$$

Programmation structurée — En reprenant les idées utilisées dans le cas de la régression affine, on aboutit à la procédure suivante, E est la covariance de X et $X^2 + Y^2$, F celle de Y et $X^2 + Y^2$ et G est la moyenne de $X^2 + Y^2$:

```
PROCEDURE Regression_Cercle(Entrée n : Entier ; x, y : Vecteur ; Sortie a, b, R : Réels) ;
Début
  Ex = Ey = Vx = Vy = Cov_xy = E = F = G = 0 ;
  Pour i allant de 1 à n Faire
  Début
     Ex = Ex + x_i ; Ey = Ey + y_i ;
  Fin ;
  Ex = Ex/n ; Ey = Ey/n ;
  Pour i allant de 1 à n Faire
  Début
    t = x_i – Ex ; s = x_i^2 + y_i^2 ;
    Vx = Vx + t·x_i ;
    Cov_xy = Cov_xy + t·y_i ;
    E = E + t·s ;
    t = y_i – Ey ;
    Vy = Vy + t·y_i ;
    F = F + t·s ;
    G = G + s ;
  Fin ;
  Δ = Vx·Vy – (Cov_xy)^2 ;
  G = G/n ;
  α = Cov_xy·F – Vy·E ;
```

β = Cov_xy·E – Vx·F ;
a = α/Δ ; b = β/Δ ; R = G – a·Ex – b·Ey ;
a = –a/2 ; b = –b/2 ; R = $\sqrt{a^2 + b^2 - R}$ *;*
Fin ;

2.8.3 Application à un calcul de rayon de courbure

On se donne, ici, un nuage de points :

$$S = \{ M_i = (x_i,y_i) \; ; -n \le i \le n \}$$

et on voudrait calculer une approximation du rayon de courbure en M_0.

Une première idée serait d'utiliser la formule :

$$R_0 = \frac{\left(1 + y_0'^2\right)^{\frac{3}{2}}}{y_0''}$$

et d'utiliser des approximations des dérivées par différences finies centrées.

Cette idée donnera des résultats médiocres si les points M_i sont éloignés pour i = –1, 0 et 1.

Une autre idée, plus efficace en général, est de prendre pour approximation de R_0 le rayon du cercle des moindres carrés défini par les M_i, avec $-p \le i \le p$, où p est à choisir, en fonction de la forme de S, entre 1 et n. Pour favoriser les points proches de M_0, on cherchera plutôt à minimiser la quantité :

$$E(a,b,c) = \sum_{i=-p}^{p} \frac{1}{i^2} \cdot \left(x_i^2 + y_i^2 + a \cdot x_i + b \cdot y_i + c\right)^2$$

où on pose 1/0 = 1.

3. Approximation uniforme des fonctions continues. Courbes de Bernstein, Bézier et B-Splines

3.1 Position du problème

Etant donnée une fonction continue, f : [a,b] → R, dont on connaît les valeurs en un nombre fini de points x_0, x_1, ..., x_n, on voudrait calculer, pour tout x dans [a,b], une valeur approchée de f(x). De plus on voudrait que la précision de cette approximation augmente avec n.

Le *théorème de Weirstrass* (1855), qui nous dit que toute fonction continue sur un intervalle fermé borné peut être approchée uniformément par des polynômes, nous permet de répondre à la question. Une démonstration de ce théorème peut se faire de façon constructive en utilisant les *polynômes de Bernstein* (Cf § 3.2).

Les restrictions de cette construction sont, d'une part que le modèle de courbe est de type cartésien, i. e. de la forme y = f(x), ce qui exclut les courbes fermées, et d'autre part qu'il est nécessaire de connaître f en des points équidistants.

Si on s'intéresse au cas, plus général, des courbes définies par des équations paramétriques dans R^2 ou R^3, on aboutit aux procédés d'approximations de Bézier et aux approximants B-Splines.

L'idée de base de ces méthodes est, étant donné un nuage de points :
$S = \{M_i = (x_i,y_i) \; ; \; i \le i \le n \}$, de chercher des approximations polynomiales d'une paramétrisation (x(t),y(t)) en prenant [0,1] comme espace des paramètres, avec :

$$\begin{cases} x\left(\dfrac{i}{n}\right) = x_i \\ y\left(\dfrac{i}{n}\right) = y_i \end{cases} \quad (0 \le i \le n)$$

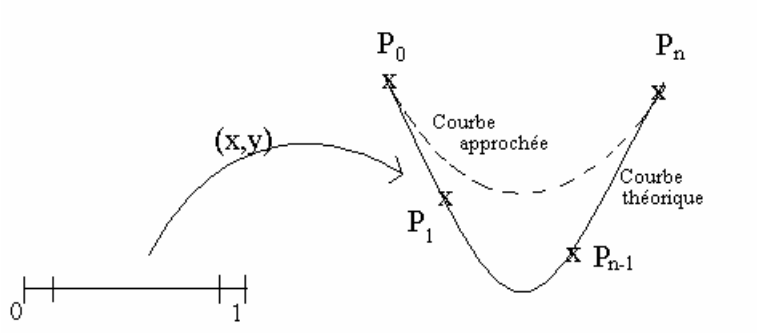


Figure 5.3

Ces procédés vont aussi s'appliquer à la définition de surfaces à partir d'une série de points de R^3 donnés par l'utilisateur en fonction d'expérimentations ou toute autre considération.

Ces idées sont utilisées en *conception assistée par ordinateur* (*CAO*) pour définir des carrosseries de voiture ou des fuselages d'avions.

En France, c'est *P. Bezier* qui est à l'origine de ce genre d'idées avec le logiciel *UNISURF* qu'il développa pour le compte de la régie Renault à partir de 1962 (Cf. Math et C.A.O., vol. 4, Chap. 4).

La technique des courbes B-Splines est une généralisation des courbes de Bézier qui ont le désavantage d'avoir une nature globale, c'est-à-dire que la modification d'un point de contrôle M_i va modifier toute la courbe, contrairement aux B-Splines qui ont un caractère local.

3.2 Les bases de Bernstein. Polynômes de Bernstein

3.2.1 Les bases de Bernstein

Pour tout $n \geq 1$, on note $R_n[t]$ l'espace vectoriel des polynômes de degré au plus n.

Définition **:** On appelle *base de Bernstein d'ordre n*, la base de $R_n[t]$ formée des polynômes $B_{n,k}$ définis par :

$$B_{n,k}(t) = C_n^k \cdot t^k \cdot (1-t)^{n-k} \quad (0 \leq k \leq n)$$

Remarque — En écrivant que, pour tout k = 0, 1, ..., n :

$$t^k = (t + (1-t))^{n-k} \cdot t^k = \sum_{j=0}^{n-k} C_{n-k}^j \cdot t^j \cdot (1-t)^{n-k-j} \cdot t^k = \sum_{i=k}^{n} \lambda_{i,k} \cdot B_{n,i}(t)$$

on déduit que $\{ B_{n,k} ; 0 \leq k \leq n \}$ engendre $R_n[t]$ et c'est donc bien une base.

Il est facile de vérifier les :

Propriétés :

(i) $\forall t \in]0,1[, B_{n,k}(t) > 0$;

(ii) $B_{n,k}(0) = \begin{cases} 0 \text{ si } 1 \leq k \leq n \\ 1 \text{ si } k = 0 \text{ et } n \geq 1 \end{cases}$

(iii) $B_{n,k}(t) = B_{n,n-k}(1-t) \quad (0 \leq k \leq n)$

(iv) $\sum_{k=0}^{n} B_{n,k}(t) = 1$

(v) On a la récurrence qui permet de calculer les $B_{n,k}(t)$:

$B_{n,n}(t) = t \cdot B_{n-1,n-1}(t)$

$B_{n,0}(t) = (1-t) \cdot B_{n-1,0}(t)$

$B_{n,k}(t) = (1-t) \cdot B_{n-1,k}(t) + t \cdot B_{n-1,k-1}(t) \quad (1 \leq k \leq n-1)$

Remarque — Les fonctions $B_{n,k}(t)$ sont aussi appelées *fonctions mélanges* ou *fonctions pondérantes.*

Sur la figure 5.4, on trouvera les graphes des fonctions $B_{n,k}$ pour $0 \leq k \leq n = 5$.

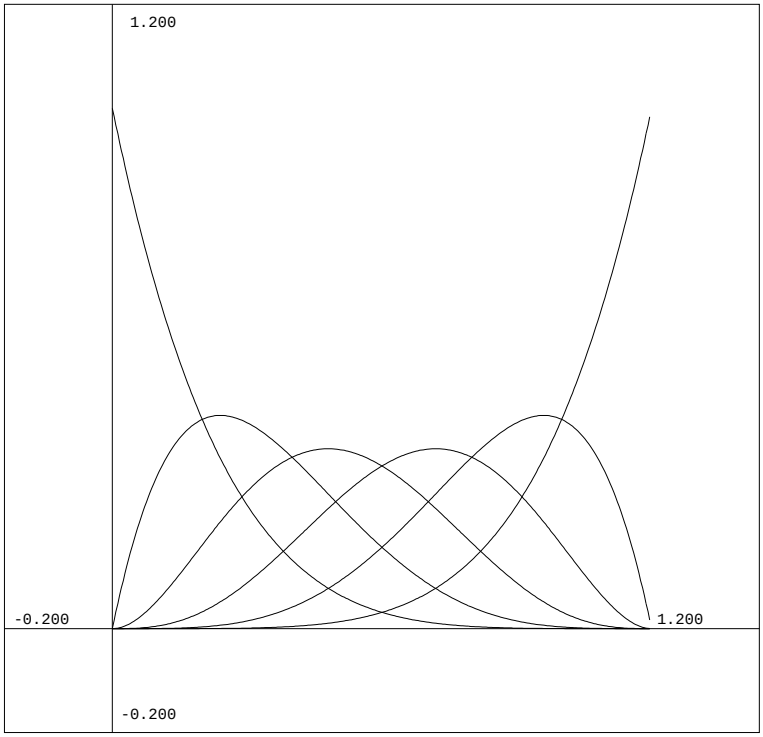


Figure 5.4

Les $B_{n,k}(t)$ peuvent être calculées de façon récursive avec la procédure suivante.

```
FONCTION BaseBernsteien_R(Entrée n, k : Entier ; t : Réel) : Réel ;
Début
   Si n = 0
   Alors Début
         BaseBernstein_R = 1 ;
         Sortir ;
       Fin ;
   Si k = 0
   Alors BaseBernstein_R = (1 – t)^n
   Sinon Début
         Aux1 = BaseBernstein_R(n – 1,k,t) ;
         Aux2 = BaseBernstein(n – 1,k – 1,t) ;
         BaseBernstein_R = (1 – t)·Aux1 + t·Aux2 ;
       Fin ;
Fin ;
```

Une autre façon de calculer est d'utiliser la récurrence (V) ci-dessus ce qui donnera, pour n grand, une procédure de calcul beaucoup plus rapide.

L'algorithme de calcul est le suivant :

Etape 0 — $B_{n,i}^{(0)} = \begin{cases} 1 \text{ si } i = k \\ 0 \text{ sinon} \end{cases}$ (i = 0, 1, ...n)

Etape j — $B_{n,i}^{(j)} = (1-t)\cdot B_{n,i}^{(j-1)} + t \cdot B_{n,i+1}^{(j-1)}$ (i = 0, 1, ..., n - j) pour j = 1, 2, ..., n.

Et alors :

$$B_{n,k}(t) = B_{n,0}^{(n)}(t)$$

Cet algorithme est connu sous le nom d'algorithme de *De Casteljeau*. On le retrouvera pour calculer les polynômes de Bernstein et de Bézier.

3.2.2 Polynômes de Bernstein associés à une fonction continue

Dans ce paragraphe, on se donne une fonction : f : [0,1] → R.

Définition **:** Pour tout n ≤ 1, le n^{ème} *polynôme de Bernstein* associé à f est défini par :

$$B_n(f,t) = \sum_{k=0}^{n} f\left(\frac{k}{n}\right) \cdot B_{n,k}(t)$$

L'intérêt de cette suite de polynômes est lié au :

Théorème **:** La suite $(B_n)_{n\in N}$ converge uniformément vers f sur [0,1].

Démonstration **—** Cf. Davis p. 109.

Remarque **—** Malheureusement la convergence n'est pas très rapide, on peut montrer que l'erreur est un O(1/n).

De plus, du point de vue numérique il ne sera pas très pratique de manipuler des polynômes de degré trop élevé.

Un algorithme de calcul est celui de De Casteljeau qui est décrit dans le paragraphe suivant dans le cas des courbes de Bézier.

3.3 Les courbes de Bézier

3.3.1 Données du problème

On suppose que l'on dispose d'un nuage de points expérimentaux (ou définis par l'utilisateur) :

$$S = \{ P_i = (x_i,y_i) ; 0 \le i \le n \}$$

et on voudrait trouver une courbe d'équations paramétriques :

$$\begin{cases} x = x(t) \\ y = y(t) \end{cases} \quad t \in [0,1]$$

qui vérifie :

$$\begin{cases} x\left(\frac{i}{n}\right) = x_i \\ y\left(\frac{i}{n}\right) = y_i \end{cases} \quad (0 \le i \le n)$$

En approximant les fonctions x et y par leurs approximants de Bernstein respectifs, on aboutit à la définition des courbes de Bézier.

Remarque — Les points P_i sont appelés *points de contrôles* ou *pôles* et le polygone de R^2 qu'ils définissent est le *polygone descripteur*.

Les courbes de Bézier font partie d'une famille de courbes appelées *courbes à pôles*.

3.3.2 Les courbes de Bézier

Définition **:** La *courbe de Bézier* associée aux n pôles P_i est la courbe polynomiale de degré n définie par les équations paramétriques :

$$\begin{cases} P_x(t) = B_n(x,t) \\ P_y(t) = B_n(y,t) \end{cases} \quad t \in [0,1]$$

ou encore par :

$$\forall t \in [0,1], P(t) = \sum_{k=0}^{n} B_{n,k}(t) \cdot P_k$$

Les propriétés de ces courbes sont résumées dans le :

Théorème **:** (i) $P(0) = P_0$; $P(1) = P_n$ (P_0 et P_n sont donc les extrémités de la courbe) ;
(ii) la courbe de Bézier est contenue dans l'enveloppe convexe des points de contrôles ;
(iii) la tangente en P_0 à la courbe de Bézier est dirigée par $\overrightarrow{P_0P_1}$ et la tangente en P_n est dirigée par $\overrightarrow{P_{n-1}P_n}$;
(iv) chaque point de contrôle exerce sur la courbe une attraction qui est proportionnelle à $B_{n,k}(t)$.

Démonstration — Dony p. 111.

On a donc l'allure de courbe suivante :

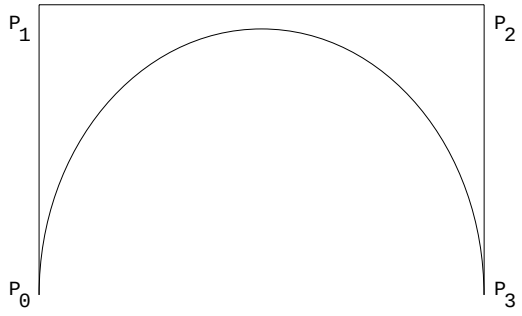


Figure 5.5

Remarque — Avec la récurrence (v) du paragraphe (3.2.1) sur les polynômes de Bernstein, en notant $P^{(u,v)}$ la courbe de Bézier associée au polygone $\{P_i ; u \le i \le v\}$, on déduit que :

$$P^{(0,n)}(t) = (1-t)\cdot P^{(0,n-1)}(t) + t\cdot P^{(1,n)}(t)$$

c'est-à-dire que la courbe de Bézier associée au polygone $\{P_i ; 0 \le i \le n\}$ se déduit des courbes associées aux polygones $\{P_i ; 0 \le i \le n-1\}$ et $\{P_i ; 1 \le i \le n\}$, ce qui permet de donner un algorithme de construction de telles courbes.

3.3.3 Algorithme de De Casteljeau

De ce qui précède, on déduit que si on pose, pour $t \in [0,1]$:

$$P_k^{(0)}(t) = P_k \quad (0 \le k \le n)$$

$$P_k^{(j)}(t) = (1-t)\cdot P_k^{(j-1)}(t) + t\cdot P_{k+1}^{(j-1)}(t) \quad (0 \le k \le n-j;\ 1 \le j \le n)$$

on a alors :

$$P^{(0,n)}(t) = \sum_{k=0}^{n-j} B_{n-j,k}(t)\cdot P_k^{(j)}(t) \ (j = 0, 1, ..., n)$$

Ce qui se démontre par récurrence sur j en utilisant la remarque précédente.

Et pour j = n, on a alors :

$$P^{(0,n)}(t) = P_0^{(n)}$$

Ce qui nous donne la programmation structurée :

```
PROCEDURE Bézier(Entrée n : Entier ; P : TableauDePoints ; t : Réel ; Sortie B : Point) ;
Début
   Q = P ; { Q_k = P_k^(0) }
   Pour j allant de 1 à n faire
   Début
      Pour k allant de 0 à n – j faire
      Début
         P_k = (1 – t)·Q_k + t·Q_k+1 ; { P_k = P_k^(j), Q_k = P_k^(j-1) }
      Fin ;
      Pour k allant de 0 à n – j + 1 faire Q_k = P_k ;
   Fin ;
   B = P_0 ;
Fin ;
```

Remarque 1 — Cet algorithme peut aussi être utilisé pour calculer l'approximant de Bernstein, les calculs se faisant seulement sur la deuxième composante y.

Remarque 2 — Cet algorithme est très performant car il évite le calcul récursif des fonctions de base, qui peut être très lent pour n grand.

Remarque 3 — Si on désire que la courbe de Bézier passe plus près de l'un des points de contrôle, il suffit de le compter plusieurs fois, on dit que l'on augmente sa multiplicité.

Remarque 4 — Dans le cas des courbes fermées, on aura en général la situation de la figure 5.6 :

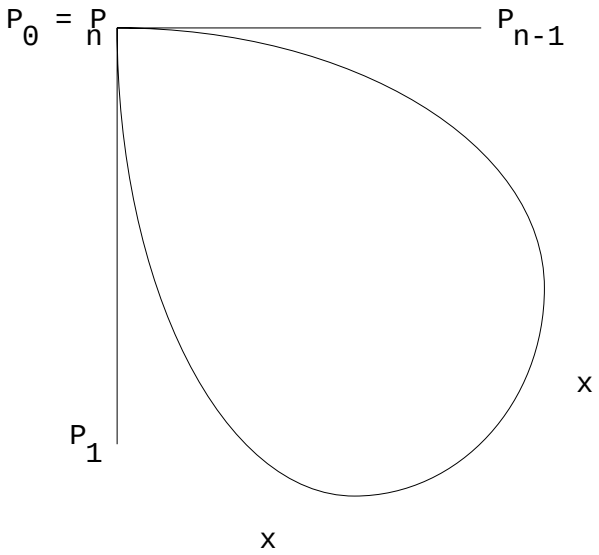


Figure 5.6

Des exemples de tracés sont donnés sur la figure 5.7 où on travaille avec le même fichier de points, en affectant successivement à tous les points les multiplicités 1, 2, 3 et 4.

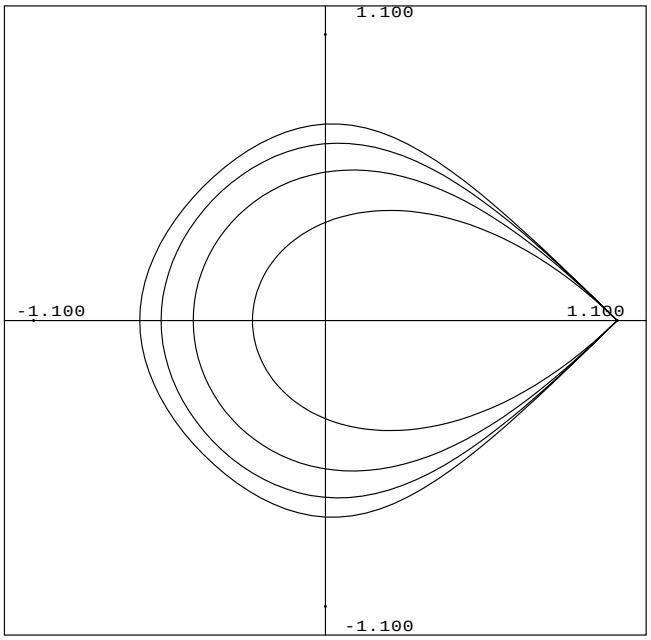


Figure 5.7

3.4 Les surfaces de Bézier

3.4.1 Définition

Les définitions précédentes s'étendent au cas des surfaces de R^3.

Les données sont un nuage de points de contrôle de R^3, formant le *polyèdre descripteur* :

$$S = \{ P_{i,j} ; 0 \le i \le n ; 0 \le j \le m \}$$

et le *carreau de Bézier* correspondant est la surface d'équations paramétriques :

$$\begin{array}{rccc} B: & [0,1]^2 & \longrightarrow & R^3 \\ & (u,v) & \longmapsto & \displaystyle\sum_{\substack{0\le i\le n \\ 0\le j\le m}} B_{n,i}(u)\cdot B_{m,j}(v)\cdot P_{i,j} \end{array}$$

Une carrosserie automobile peut être définie avec une vingtaine de carreaux de Bézier correctement raccordés.

Remarque — On peut écrire que $B(u,v) = \sum_{i=0}^{n} B_{n,i}(u)\cdot P_i(v)$, où on a noté $P_i(v) = \sum B_{m,j}(v)\cdot P_{i,j}$. C'est-à-dire que pour tout v fixé, u $\rightarrow$B(u,v) est la courbe de Bézier associée au polygone descripteur $\{ P_i(v) ; 1 \le i \le n \}$. On a un résultat analogue à u fixé.

3.4.2 Propriétés

Les propriétés de ces surfaces sont résumées dans le théorème ci-dessous.

Théorème **:** (i) Le bord de la surface de Bézier définie par le polyèdre descripteur S est formé des courbes de Bézier définies par les polygones $\Gamma_0 = \{P_{0,0},P_{0,1}, ..., P_{0,m}\}$, $\Gamma_n = \{P_{n,0},P_{n,1}, ..., P_{n,m}\}$, $\Gamma_0' = \{P_{0,0},P_{1,0}, ..., P_{n,0}\}$ et $\Gamma_m' = \{P_{0,m},P_{1,m}, ..., P_{n,m}\}$. Soit :

$$B(0,v) = \sum_{j=0}^{m} B_{m,j}(v)\cdot P_{0,j}$$

$$B(1,v) = \sum_{j=0}^{m} B_{m,j}(v)\cdot P_{n,j}$$

$$B(u,0) = \sum_{i=0}^{n} B_{n,i}(u)\cdot P_{i,0}$$

$$B(u,1) = \sum_{i=0}^{n} B_{n,i}(u)\cdot P_{i,m}$$

(ii) la surface de Bézier est contenue dans l'enveloppe convexe des points de contrôle ;

(iii) le plan tangent à la surface de Bézier en $P_{0,0}$ est défini par $\left(\overrightarrow{P_{0,0}P_{0,1}}, \overrightarrow{P_{0,0}P_{1,0}}\right)$; en $P_{n,0}$ par $\left(\overrightarrow{P_{n,0}P_{n,1}}, \overrightarrow{P_{n,0}P_{n-1,0}}\right)$; en $P_{n,m}$, par $\left(\overrightarrow{P_{n,m}P_{n,m-1}}, \overrightarrow{P_{n,m}P_{n-1,m}}\right)$; et en $P_{0,m}$ par $\left(\overrightarrow{P_{0,m}P_{0,m-1}}, \overrightarrow{P_{0,m}P_{1,m}}\right)$;

3.4.3 Algorithme de De-Casteljeau

En utilisant la remarque du paragraphe (3.4.1), il suffit d'utiliser l'algorithme de De-Casteljeau n + 1 fois pour calculer les $P_i(v)$, puis de nouveau cet algorithme pour les B(u,v).

3.5 Les courbes B-Splines

3.5.1 Introduction

Les courbes de Bézier présentent les inconvénients suivants :

- ce sont des courbes polynomiales de degré n pour un polygone à n + 1 points, ce qui entraînera une forte instabilité numérique pour n grand ;
- la modification d'un seul point de contrôle va se répercuter sur toute la courbe (caractère global des courbes de Bézier) ;
- comme on peut le voir sur les exemples de tracés du paragraphe précédent, la courbe peut passer assez loin des points de contrôle ; pour s'en rapprocher on devra augmenter la multiplicité des points ce qui entraînera une augmentation du degré des fonctions polynomiales à traiter.

C'est en relation avec ces problèmes que dans les années 1970, *M.G. Cox* et C. *De Boor* ont introduit la notion de fonction B-Spline comme généralisation des courbes de Bézier.

Les courbes B-Splines sont définies localement par des expressions polynomiale de degré j assez faible (usuellement 3 ou 4), ce dernier étant indépendant du nombre de points et la modification d'un point de contrôle ne se fait ressentir qu'au voisinage de ce dernier. De plus elles ont les mêmes propriétés que celles de Bézier.

3.5.2 Fonctions de base B-Splines

Définition **:** Soient $n \geq 1$ et j dans N, avec $2 \leq j \leq n + 1$. On appelle alors *vecteur nodal* associé à (n,j), le vecteur v de N^{n+j+1} défini par :

$$\begin{cases} v_i = 0; \ 0 \leq i \leq j - 1 \\ v_i = i - j + 1; \ j \leq i \leq n \\ v_i = n - j + 2; \ n + 1 \leq i \leq n + j \end{cases}$$

Définition **:** Soient $n \geq 1$ et j dans N, avec $2 \leq j \leq n + 1$. En notant v le vecteur nodal associé à (n,j), on appelle *base B-Spline d'ordre j*, la suite de fonctions $\{ N_{i,j} ; 0 \leq i \leq n \}$, définie sur $[0, t_{max}]$, avec $t_{max} = n - j + 2$, par la récurrence :

$$N_{i,1}(t) = \begin{cases} 1 \text{ si } v_i \le t < v_{i+1} \\ 0 \text{ sinon} \end{cases} \quad (0 \le i \le n)$$

$$N_{i,k}(t) = \alpha_{i,k} \cdot (t - v_i) \cdot N_{i,k-1}(t) + \beta_{i,k} \cdot (v_{i+k} - t) \cdot N_{i+1,k-1}(t)$$

pour $2 \le k \le j$, $0 \le i \le n$, où :

$$\alpha_{i,k} = \begin{cases} \dfrac{1}{v_{i+k-1} - v_i} & \text{si } v_{i+k-1} \ne v_i \\ 0 \text{ sinon} \end{cases}$$

$$\beta_{i,k} = \begin{cases} \dfrac{1}{v_{i+k} - v_{i+1}} & \text{si } v_{i+k} \ne v_{i+1} \\ 0 \text{ sinon} \end{cases}$$

Remarque 1 — L'algorithme de construction des $N_{i,j}$ défini ci-dessus est dû à Cox et De Boor.

Remarque 2 — Les fonctions $N_{i,j}$ sont aussi appelées *fonctions mélanges* ou *fonctions pondérantes*.

Exemple — Pour n = 3, j = 2, le vecteur nodal est v = (0,0,1,2,3,3) et :

$$N_{0,2}(t) = \begin{cases} 1 - t \text{ si } t \in [0,1] \\ 0 \text{ sinon} \end{cases}$$

$$N_{1,2}(t) = \begin{cases} t \text{ si } t \in [0,1] \\ 2 - t \text{ si } t \in [1,2] \\ 0 \text{ si } t \in [2,3] \end{cases}$$

$$N_{2,2}(t) = \begin{cases} 0 \text{ si } t \in [0,1] \\ t - 1 \text{ si } t \in [1,2] \\ 3 - t \text{ si } t \in [2,3] \end{cases}$$

$$N_{3,2}(t) = \begin{cases} t - 2 \text{ si } t \in [2,3] \\ 0 \text{ sinon} \end{cases}$$

Les propriétés de ces fonctions sont résumées dans le :

Théorème : (i) $\forall t \in [0, t_{max}]$, $N_{i,j}(t) \ge 0$, avec, pour $j \ge 2$:

$$\begin{cases} N_{i,j}(t) > 0 \text{ si } t \in]v_i, v_{i+j}[\\ 0 \text{ sinon} \end{cases}$$

et sur $]v_i, v_{i+j}[$, $N_{i,j}$ est polynomiale de degré $j - 1$;

(ii) $\forall t \in]0, t_{max}[$, $\sum_{i=0}^{n} N_{i,j}(t) = 1$;

(iii)

$$\begin{cases} N_{i,j}(0) = 0 \quad (1 \le i \le n) \\ N_{0,j}(0) = 1 \end{cases}$$

$$\begin{cases} N_{i,j}(t_{max}) = 0 \quad (0 \le i \le n - 1) \\ N_{n,j}(t_{max}) = 1 \end{cases}$$

(iv) $\{ N_{i,n+1} ; 0 \le i \le n \}$ est la base de Bernstein de degré n sur [0,1] ($t_{max} = 1$).

Tracé des fonctions de base B-Spline — Tout d'abord on définit une procédure qui calcule les composantes du vecteur nodal v, soit :

```
PROCEDURE VecteurNodal(Entrée n, j : Entier ; Sortie v : VecteurEntier) ;
Début
   Pour i allant de 0 à j – 1 faire v_i = 0 ;
   Pour i allant de j à n faire v_i = i – j + 1 ;
   Pour i allant de n + 1 à n + j faire v_i = n – j + 2 ;
Fin ;
```

Ensuite, on peut définir les fonctions de base de manière récursive de la manière suivante :

```
FONCTION BaseBSpline(Entrée n, i, j : Entier ; v : VecteurEntier ; t : Réel) : Réel ;
Début
   Si (t < v_i) ou (t ≥ v_{i+j})
   Alors Début
         BaseBSpline = 0 ;
         Sortir ;
      Fin
   Sinon Début
         Si j = 1
         Alors Début
               BaseBSpline = 1 ; { puisque t ∈ [v_i,v_{i+1}[ }
               Sortir ;
            Fin
         Sinon Début
               Aux1 = BaseBSpline(n,i,j – 1,v,t) ;
               Si (Aux1 = 0) ou (v_{i+j–1} = v_i)
               Alors S1 = 0
               Sinon S1 = (t – v_i) / (v_{i+j–1} – v_i) ·Aux1 ;
               Aux2 = BaseBSpline(n,i+1,j – 1,v,t) ;
               Si (Aux2 = 0) ou (v_{i+j} = v_{i+1})
               Alors S2 = 0
               Sinon S2 = (v_{i+j} – t) / (v_{i+j} – v_{i+1}) ·Aux2 ;
               BaseBSpline = S1 + S2 ;
            Fin ;
      Fin ;
Fin ;
```

En fait, comme on le verra dans le paragraphe suivant, on peut se passer de la récursivité qui peut donner un calcul très long pour de grandes valeurs de j.

Sur la figure 5.8, on donne les graphes des fonctions de base pour n = 5 et j = 3.

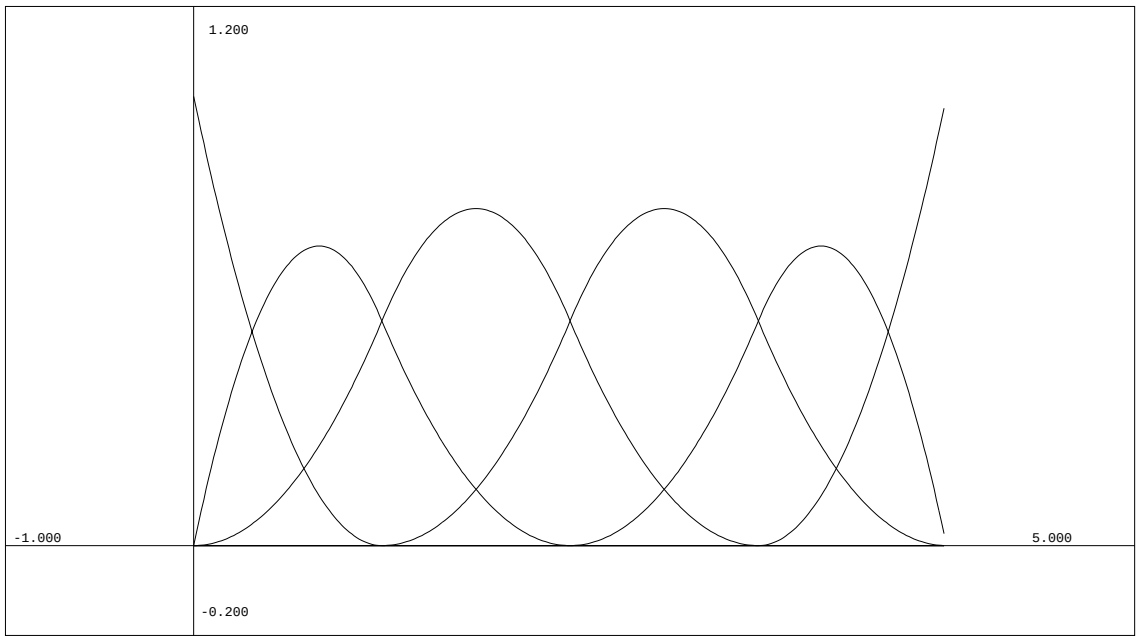


Figure 5.8

3.5.3 Courbes B-Splines

On reprend les notations du paragraphe (3.3.1) et on se donne $n \geq 1$ et $2 \leq j \leq n + 1$ dans N.

Définition **:** On appelle *courbe B-Spline d'ordre j* (ou de degré j – 1) associée au polygone descripteur S, la courbe d'équation paramétrique :

$$B(t) = \sum_{i=0}^{n} N_{i,j}(t) \cdot P_i \quad \text{si } t \in [0, t_{max}] = [0, n - j + 2]$$

où $\{ N_{i,j}(t) ; 0 \leq i \leq n \}$ est la base B-Spline associée à (n,j).

Remarque — On a $[0,t_{max}] = [v_{j-1},v_{n+1}]$ et pour k = j – 1, ..., n, si t est dans $[v_k,v_{k+1}]$, on aura $t \notin]v_i, v_{i+j}[$ pour $i \leq k - j$ ou $i \geq k + 1$, ce qui donnera $N_{i,j}(t) = 0$ pour de tels i. En définitive on a donc :

$$B(t) = \sum_{i=k-j+1}^{k} N_{i,j}(t) \cdot P_i \quad (k = j - 1, \ldots, n;\ t \in [v_k, v_{k+1}])$$

On a donc seulement j termes à calculer dans la somme et non pas n + 1 comme dans le cas des courbes de Bézier. De plus les polynômes que l'on manipule sont de degré au plus j. Tout cela étant indépendant du nombre de points n + 1.

Exemple — Pour n = 3, j = 2 : $B(t) = \sum_{i=0}^{3} N_{i,3}(t) \cdot P_i \quad t \in [0,3]$.

Sur [0,1] = [v_1,v_2] (k = j – 1 = 1), on a :

$$B(t) = N_{0,2}(t)\cdot P_0 + N_{1,2}(t)\cdot P_1 = (1 - t)\cdot P_0 + t\cdot P_1 ;$$

sur [1,2] = [v_2,v_3] (k = j = 2), on a :

$$B(t) = N_{1,2}(t)\cdot P_1 + N_{2,2}(t)\cdot P_2 = (2 - t)\cdot P_1 + (t - 1)\cdot P_2 ;$$

Sur [2,3] = [v_3,v_4] (k = n), on a :

$$B(t) = N_{2,2}(t)\cdot P_2 + N_{3,2}(t)\cdot P_3 = (3 - t)\cdot P_2 + (t - 2)\cdot P_3 ;$$

En utilisant la procédure de calcul des fonctions de base B-Spline, on en déduit immédiatement un algorithme de calcul des courbes B-Splines.

L'indice k ∈ {j – 1, ..., n} tel que t ∈ [v_k,v_{k+1}[, pour t dans [0,t_{tmax}[, est simplement donné par : k = PartieEntière(t) + j – 1

De plus, pour t = 0, on a :

$$B(0) = \sum_{i=0}^{n} N_{i,j}(0) \cdot P_i = 1 \cdot P_0 = P_0$$

et pour t = t_{max}, on a :

$$B(t_{max}) = \sum_{i=0}^{n} N_{i,j}(t_{max}) \cdot P_i = 1 \cdot P_n = P_n$$

Si on ne veut pas utiliser la récursivité (qui donnera un calcul assez lent pour de grandes valeurs de j et même très lent pour j = n + 1), on peut s'inspirer de l'algorithme de De Casteljeau. L'algorithme obtenu est connu sous le nom d'algorithme d'Oslo. (Cf. Math. et C. A. O., Vol. 6, p. 316 et 317).

La procédure de calcul est alors la suivante :

```
PROCEDURE B-Spline(Entrée n, j : Entier ; v : VecteurEntier ;  P : TableauDePoints ; t : Réel ;
                   Sortie B : Point) ;
Début
   Si t ≥ n – j + 2
   Alors Début
         B = P_n ;
         Sortir ;
      Fin ;
   k = PartieEntière(t) + j – 1 ;
   Pour i allant de k – j + 1 à k faire Q_i = P_i ;
   Pour p allant de j à 2 faire
   Début
      Pour i allant de k – p + 1 à k faire
      Début
         Si v_{i+p–1} = v_i
         Alors Début
               S1 = 0 ; S2 = 0 ;
            Fin
         Sinon Début
               S1 = (t – v_i) / (v_{i+p–1} – v_i) · Q_i ;
               S2 = (v_{i+p–1} – t) / (v_{i+p–1} – v_i) · Q_{i-1} ;
```

```
                Fin ;
            P  = S1 + S2 ;
        Fin ;
        Pour i Allant de k – p + 1 à k faire Q_i = P_i ;
    Fin ;
    B = P_k ;
Fin ;
```

Sur la figure 5.9, on donne un exemple de tracé, avec n = 17, j = 3 et le fichier de points suivant : 1 –2 ; 1 –1 ; 1.4 0.6 ; 1 1.8 ; –0.6 1.8 ; –1 1.6 ; –1 1 ; –0.6 0.8 ; –1.6 0 ; –1.4 –0.2 ; –1 0 ; –1 –1 ; –0.6 –1.2 ; –1 –1.2 ; –1 –1.8; –0.4 –1.6 ; –0.4 –2 ; 1 –2

Les temps de calcul respectifs, sur un micro-ordinateur équipé d'un coprocesseur 80387, sont : en utilisant la récursivité : 82 centièmes de secondes ; en utilisant l'algorithme d'Oslo : 33 centièmes de secondes.

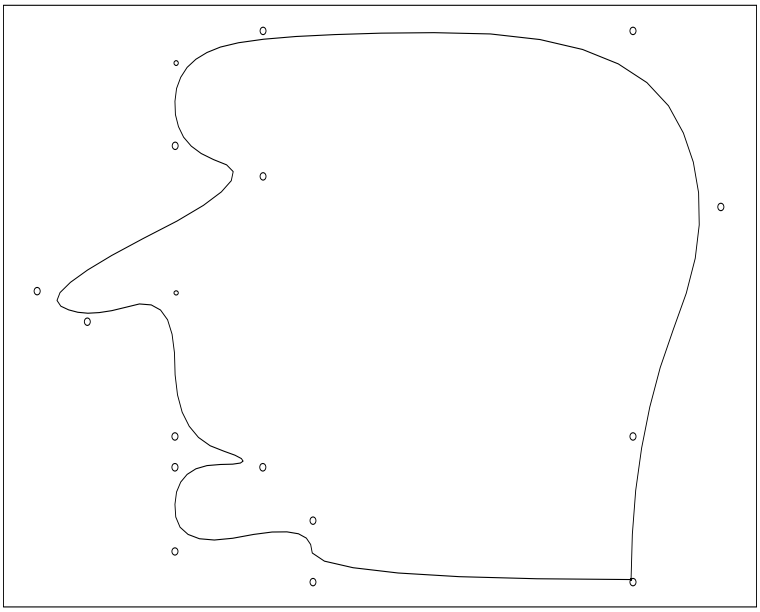


Figure 5.9

4. Problèmes d'interpolation

4.1 Introduction. Position des problèmes

On se place toujours dans l'hypothèse où une fonction continue f, définie sur un intervalle [a,b] est connue en un nombre fini de points x_i avec i = 0, 1, ..., n. Mais dans ce paragraphe, on cherche une fonction ϕ qui coïncide avec f en tous les points x_i. Cette fonction ϕ sera cherchée aussi simple que possible (polynomiale ou polynomiale par morceaux). On dira alors que ϕ *interpole* f.

Historiquement ce sont les problèmes de *tabulations numériques* des fonctions transcendantes élémentaires, puis à partir du 17ème siècle le calcul approché des dérivées et des intégrales qui sont à l'origine des méthodes d'interpolation.

Un problème important qui se posera pour toute méthode d'interpolation sera d'évaluer *l'erreur d'interpolation* $R(x) = \left|f(x) - \Phi(x)\right|$ en tout point x de [a,b].

On peut aussi se demander s'il est possible de reconstituer f comme limite des fonctions ϕ quand le module de la subdivision :

$$\Delta(S) = \mathrm{Sup}\left\{\left|x_{j+1} - x_j\right|;\ 0 \le j \le n-1\right\}$$

tend vers 0.

Par exemple dans le cas de l'interpolation polynomiale (Cf. (4.2)), en prenant une subdivision à pas constant (i.e. $x_{j+1} - x_j = \Delta = \frac{b-a}{n}$) et en notant P_n le polynôme de degré n qui coïncide avec f en tous les x_i, peut-on espérer que la suite $(P_n)_{n\in N}$ converge uniformément vers f sur [a,b]?

Depuis Lagrange en 1795 jusqu'à Runge en 1901 on pensait que la réponse était positive. Mais en fait il n'en est rien comme le montre le contre-exemple de Runge $f(x) = \frac{1}{1+8\cdot x^2}$ sur [0,1] : en traçant les graphes de f et des polynômes interpolateurs successifs, on constate des oscillations aux bords de l'intervalle (effets de bord ou phénomène de Runge : Cf. pararagraphe (4.2)).

Dans le Calcul Infinitésimal de Dieudonné (p. 319), on montre que $\lim_{n\to+\infty}\left|f(1) - P_n(1)\right| = +\infty$. Ce qui peut se constater sur la figure 5.10.

Pour résoudre ce genre de problème on aura recours à un procédé plus élaboré qui consiste à approcher f par des fonctions polynomiales (usuellement de degré 3) sur chacun des intervalles $[x_i,x_{i+1}]$ en imposant des conditions de régularités aux points communs à deux intervalles. C'est le principe de l'interpolation spline cubique (Cf. pararagraphe (4.3)).

4.2 L'interpolation polynomiale de Lagrange

4.2.1 L'interpolation linéaire

L'interpolation linéaire permet de calculer approximativement une valeur intermédiaire dans une table numérique. Par exemple, pour trouver une valeur approchée de la fonction f en x = 1.5 sachant que f(1) = 2.434 et f(2) = 5.842, le principe consiste à remplacer le graphe de f sur [1,2] par la droite passant par les deux points A = (1,f(1)) et B = (2,f(2)) (on dit qu'on a linéarisé f), ce qui donne y = 4.138 comme valeur approchée de f(x).

Cette valeur approchée s'écrit sous la forme :

$$y = \frac{x - x_1}{x_0 - x_1}\cdot y_0 + \frac{x - x_0}{x_1 - x_0}\cdot y_1$$

On peut penser que cette approximation sera meilleure si au lieu d'utiliser des polynômes de degré 1 on utilise des polynômes de degré p > 1. C'est le principe des polynômes d'interpolation de Lagrange.

4.2.2 Le théorème d'interpolation de Lagrange

Théorème **:** Etant donnés n + 1 réels distincts $x_0 < x_1 < ... < x_n$ et n + 1 réels y_i avec $0 \le i \le n$, il existe un unique polynôme P de degré au plus n tel que :

$$P(x_i) = y_i \ (0 \le i \le n)$$

Ce polynôme est donné par :

$$P(x) = \sum_{i=0}^{n} y_i \cdot L_i(x)$$

où :

$$L_i(x) = \frac{\prod_{\substack{j=0 \\ j\neq i}}^{n} (x - x_j)}{\prod_{\substack{j=0 \\ j\neq i}}^{n} (x_i - x_j)} \quad (0 \leq i \leq n)$$

Démonstration — En notant E = R[x] l'espace vectoriel des polynômes à coefficients réels et $u : E \to R^{n+1}$ définie par :

$$\forall P \in E,\ u(P) = (P(x_0),P(x_1),...,P(x_n))$$

tout revient à montrer que u est surjective et que sa restriction à $E_n = R_n[x]$ est bijective.

Le noyau de u est formé des multiples du polynôme $\omega(x) = \prod_{i=0}^{n} (x - x_i)$ et avec la division euclidienne, on peut écrire tout polynôme P sous la forme :

$$P = Q\cdot\omega + R \qquad (R = 0 \text{ ou } d°(R) \leq n)$$

On a donc :

$$E = E_n \oplus \mathrm{Ker}(u)$$

ce qui suffit à prouver que u est un isomorphisme de E_n sur R^{n+1}.

Remarque — En prenant $\{1,x,...,x^n\}$ comme base de E_n et la base canonique sur R^{n+1}, la matrice de la restriction de u à E_n est une matrice de *Van Der Monde* dont le déterminant est $\prod_{j<i} (x - x_i) \neq 0$. On a donc ainsi une deuxième preuve de théorème.

Dans le cas où $y_i = f(x_i)$, avec f assez régulière, on peut donner une majoration de l'erreur d'interpolation :

Théorème **:** Soit $f : [x_0,x_n] \to R$ de classe C^{n+1} et P le polynôme d'interpolation de Lagrange défini par $P(x_i) = f(x_i)$, pour i = 0, 1, ..., n, avec $x_0 < x_1 < ... < x_n$. L'erreur d'interpolation vérifie alors :

$$R(x) = |f(x) - P(x)| \leq M_{n+1} \cdot \frac{|\omega(x)|}{(n+1)!}$$

où : $\omega(x) = \prod_{i=0}^{n} (x - x_i)$ et $M_{n+1} = \mathrm{Sup}\left\{ \left| f^{(n+1)}(x) \right| ;\ x \in [x_0, x_n] \right\}$

Démonstration — Théodor p. 104.

4.2.3 Algorithme de Neville pour calculer le polynôme de Lagrange

Une programmation directe du polynôme de Lagrange est élémentaire, mais peu performante. Des algorithmes plus performants ont été donnés par Aitken (Cf. Théodor p. 94) puis par Neville. Nous utiliserons celui de Neville qui est plus performant. Le principe en est résumé dans le :

Théorème **:** Soit (P_{ik}) la suite de polynômes définie par la récurrence :

$$P_{i0}(x) = y_i \; (0 \le i \le n)$$

$$P_{ik}(x) = \frac{(x - x_{i-k}) \cdot P_{i,k-1}(x) - (x - x_i) \cdot P_{i-1,k-1}(x)}{x_i - x_{i-k}} \qquad (1 \le k \le i \le n)$$

Alors $P_{n,n}$ est le polynôme de Lagrange.

Démonstration — Cf. Stoer et Burlisch, par. (2.1.2).

Une construction peut être visualisée par la disposition triangulaire ci-dessous :

P_{00}
P_{11}
P_{10} P_{22}
P_{21} $P_{33} = P$
P_{20} P_{32}
P_{31}
P_{30}

L'algorithme de Neville peut aussi s'écrire sous la forme :

$$P_{i,k} = P_{i,k-1} + (x - x_i) \cdot \frac{P_{i,k-1} - P_{i-1,k-1}}{x_i - x_{i-k}} \; (1 \le k \le i \le n)$$

Ce qui nous donne la programmation structurée :

```
FONCTION LagrangeNeville(Entrée n : Entier ; x, y : Vecteur ; t : Réel) : Réel ;
Début
   Pour i allant de 0 à n faire
   Début
      P_i = y_i ;
      Pour j allant de i – 1 à 0 faire
      Début
         P_j = P_j+1 + (P_j+1 - P_j) / (x_i - x_j) ·(t - x_i) ;
      Fin ;
   Fin ;
   LagrangeNeville = P_0 ;
Fin ;
```

4.2.4 Forme de Newton du polynôme de Lagrange

L'algorithme de Neville est pratique pour calculer quelques valeurs du polynôme de Lagrange, mais les calculs sont relativement lents. Si on doit calculer un grand nombre de valeurs (par exemple en vue du tracé graphique), on lui préférera un algorithme du même type déduit de la forme de Newton du polynôme de Lagrange.

L'idée de Newton est d'écrire le polynôme d'interpolation sous la forme :

$$P(x) = P_0 + P_1 \cdot (x - x_0) + \ldots + P_n \cdot (x - x_0) \cdot \ldots \cdot (x - x_{n-1})$$

et les conditions $P(x_i) = y_i$ nous conduisent à la résolution d'un système triangulaire inférieur aux inconnues P_i.

La encore la résolution directe est élémentaire, mais en s'inspirant de l'algorithme de Neville on aboutit à l'algorithme plus performant décrit ci-dessous (Cf. Stoer et Burlisch : § (2.1.3)).

PROCEDURE PolynômeLagrange(Entrée n : Entier ; x, y : Vecteur ; Sortie P : Vecteur) ;
Début
  Pour i allant de 0 à n faire
  Début
    $Q_i = y_i$;
    Pour j allant de i - 1 à 0 faire $Q_j = \dfrac{Q_{j+1} - Q_j}{x_i - x_j}$;
    $P_i = Q_0$;
  Fin ;
Fin ;

L'évaluation du polynôme P se fait alors en utilisant l'algorithme de Horner, soit :

FONCTION LagrangeNewton(Entrée n : Entier ; x, P : Vecteur ; t : Réel) : Réel ;
Début
  Aux = P ;
  Pour i allant de n – 1 à 0 faire $Aux = P_i + Aux \cdot (t - x_i)$;
  LagrangeNewton = Aux ;
Fin ;

Remarque — Les coefficients P_k qui interviennent dans la forme de Newton sont aussi appelés les *différences divisées* associées aux x_i.

La figure 5.10 donne un exemple de tracé du polynôme de Lagrange. Il s'agit de l'interpolée de la fonction $f(x) = \dfrac{1}{1+x^2}$ sur [–5,5] en prenant 11 points équidistants (n = 10). On peut constater le phénomène de Runge aux bords de l'intervalle.

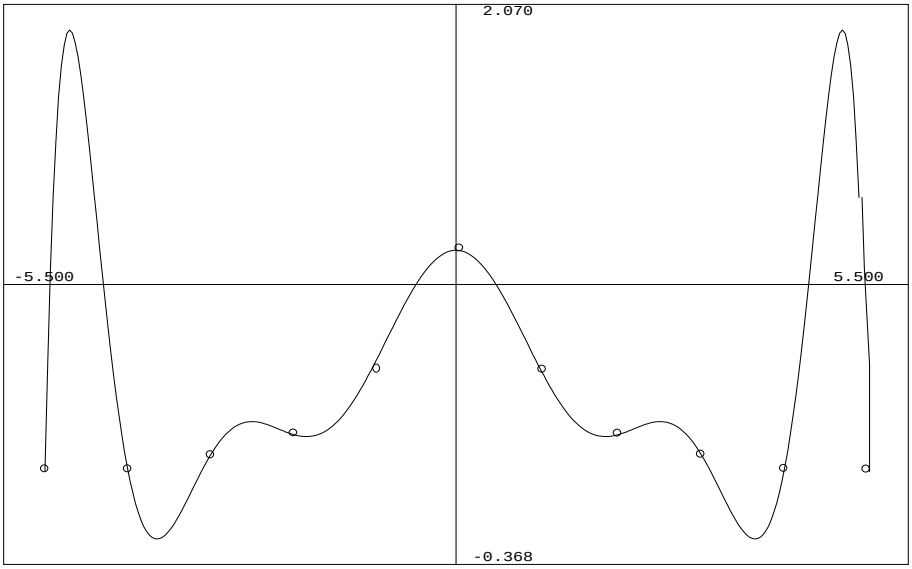


Figure 5.10

4.2.5 Sur le choix des abscisses d'interpolation. Les polynômes de Tchébychev

Le problème est de savoir s'il existe un choix optimal des abscisses x_i, c'est-à-dire un choix qui minimisera l'erreur d'interpolation.

Une réponse positive à ce problème est donnée par les zéros des *polynômes de Tchébychev* sur [–1,1].

Les polynômes de Tchébychev, que nous étudierons plus en détail dans le chapitre sur le calcul numérique des intégrales, peuvent être définis sur [–1,1] par :

$$\forall\, x \in [-1,1],\ T_n(x) = \mathrm{Cos}(n \cdot \mathrm{ArcCos}(x)) \quad (n \geq 0)$$

ou encore :

$$\forall\, x \in [-1,1],\ T_n(x) = \sum_{k=0}^{\left[\frac{n}{2}\right]} (-1)^k \cdot C_n^{2k} \cdot x^{n-2k} \cdot \left(1 - x^2\right)^k$$

Pour $n \geq 1$, T_n admet n racines distinctes dans [–1,1] données par :

$$x_k = \mathrm{Cos}\left(\frac{2 \cdot k - 1}{2 \cdot n} \cdot \pi\right) \quad (1 \leq k \leq n)$$

On a alors le :

Théorème **:** Sur [–1,1] le choix optimal des abscisses d'interpolation est donné par les racines du polynôme de Tchébychev de degré n.

Démonstration — Théodor par. (3.5).

Remarque — Dans le cas d'un intervalle [a,b] quelconque, on se ramène facilement à [–1,1] en faisant le changement de variable :

$$t = \frac{2}{b-a} \cdot x - \frac{b+a}{b-a} \quad (x \in [a,b])$$

Ci-dessous on reprend l'exemple de tracé du paragraphe précédent mais avec les abscisses de Tchébychev. Et on constate que les effets de bords sont moins importants.

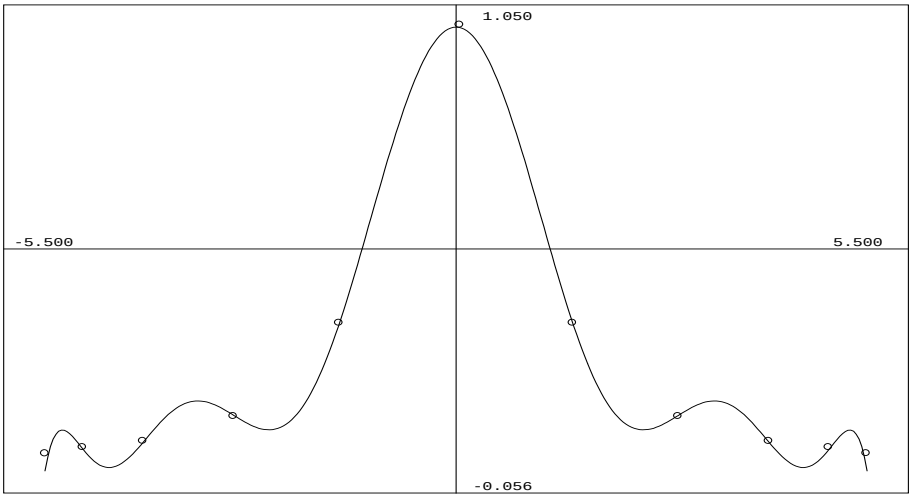


Figure 5.11

4.3 Interpolation spline cubique

4.3.1 Introduction

L'interpolation polynomiale présente deux gros inconvénients :

- le coût des calculs est trop élevé dès que le nombre de points est important ;
- pour un grand intervalle d'interpolation, on maîtrise mal les effets de bords (phénomène de Runge).

L'idée des fonctions splines est de travailler localement, c'est-à-dire qu'on cherchera plutôt à approcher le nuage de points expérimentaux par des fonctions polynomiales par morceaux, de degré assez faible. L'exemple le plus simple étant l'interpolation par une fonction affine par morceaux (c'est cette interpolation que l'on utilise pour tracer le graphe d'une fonction sur l'écran d'un ordinateur ou sur une table traçante) :

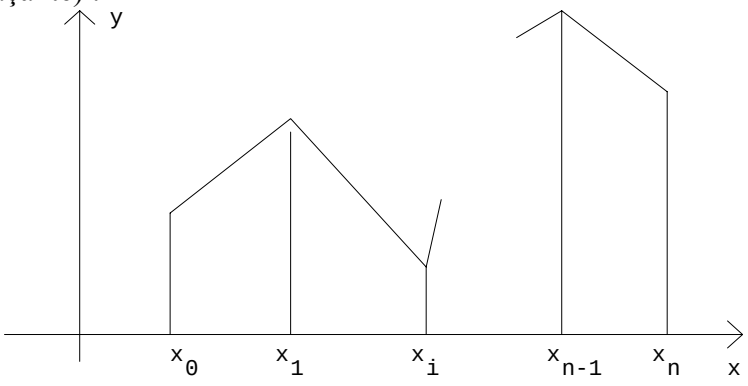


Figure 5.12

Du point de vue du calcul approché des intégrales, cette méthode aboutit à la méthode des trapèzes (Cf. le chapitre « Calcul Numérique des Intégrales »).

Dans ce paragraphe on étudiera le cas des fonctions *splines cubiques*, c'est-à-dire celui où les morceaux de polynômes sont de degré au plus 3.

Le mot anglais « spline » désigne une latte de bois flexible qui était utilisée pour tracer des courbes passant par des points donnés (les noeuds).

Vers 1946 *I. J. Schoenberg* a appelé fonctions splines des fonctions formées de morceaux de polynômes se raccordant correctement entre eux. Ces fonctions sont solutions du problème d'élasticité correspondant à la tige flexible du dessinateur. On a ensuite appelé fonctions splines des solutions approchées des extensions et généralisations du problème précédent. Le résultat n'ayant plus rien à voir avec les polynômes.

4.3.2 Position du problème. Notations

Etant donné le nuage de points :

$$S = \{ P_i = (x_i,y_i) \; ; \; 1 \leq i \leq n \}$$

on cherche une fonction polynomiale par morceaux, de degré ≤ 3, et de classe C^2, qui passe exactement par ces points.

De manière plus précise une telle courbe sera définie par une paramètrisation :

$$\begin{array}{rccl} \gamma: & [a,b] & \longrightarrow & R^2 \\ & t & \longrightarrow & \gamma(t) = (x(t), y(t)) \end{array}$$

avec une subdivision $a = t_1 < t_2 < ... < t_n = b$, telle que la restriction de γ à $[t_i,t_{i+1}]$ est polynomiale de degré au plus 3.

En raisonnant séparément sur x(t) et y(t), on est donc ramené à résoudre le problème suivant :

Etant donné un nuage de points :

$$S = \{ P_i = (t_i,x_i) \; ; \; 1 \leq i \leq n \}$$

avec :

$$t_1 < t_2 < ... < t_n$$

on cherche une fonction s de classe C^2, polynomiale de degré au plus 3 sur chaque $[t_i,t_{i+1}]$ telle que $s(t_i) = x_i$, pour tout i = 1, ..., n.

Remarque 1 — Le choix de l'espacement des t_i va influencer la forme de la courbe.

Remarque 2 — Dans le cas des courbes paramétriques, on pourra prendre [1,n] comme espace des paramètres avec $t_i = i$, si on dispose seulement des $P_i = (x_i,y_i)$ sans autre indication. Un autre choix judicieux sera fourni par les angles polaires.

On note $h_i = t_{i+1} - t_i$ pour $1 \leq i \leq n$.

On cherche $s : [t_1,t_n] \longrightarrow R$ telle que :

$$\begin{cases} s \text{ est de classe } C^2 \text{ sur } [t_1, t_n] \\ s(t) = s_i(t) = a_i \cdot (t - t_i)^3 + b_i \cdot (t - t_i)^2 + c_i \cdot (t - t_i) + d_i \text{ sur } [t_i, t_{i+1}] \\ s(t_i) = x_i \text{ , pour tout } i = 1, \ldots, n \end{cases}$$

Le fait que l'on cherche s de classe C^2 impose les conditions de recollement :

$$s_i^{(k)}(t_{i+1}) = s_{i+1}^{(k)}(t_{i+1}) \text{ pour } k = 0, 1, 2 \text{ et } i = 1, ..., n - 2$$

ce qui revient à imposer que la pente et la courbure de deux cubiques voisines soient identiques.

On va d'abord calculer les coefficients a_i, b_i, c_i et d_i en fonction des x_i et des $x''_i = x''(t_i)$, puis les x''_i apparaîtront comme solutions d'un système linéaire tridiagonal.

4.3.3 Calcul des coefficients d_i

On a $s(t_i) = d_i = x_i$ pour $i = 1, ..., n - 1$. Soit :

$$d_i = x_i \quad (i = 1, ..., n - 1)$$

4.3.4 Calcul des b_i et des a_i

On a, sur $[t_i,t_{i+1}]$, $s''(t) = 6 \cdot a\,(t - t_i) + 2 \cdot b_i$, ce qui donne :

$$\begin{cases} x''_i = s''(t_i) = 2 \cdot b_i \\ x''_{i+1} = s''(t_{i+1}) = 6 \cdot a_i \cdot h_i + 2 \cdot b_i \end{cases}$$

d'où :

$$\begin{cases} b_i = \dfrac{x''_i}{2} \\ a_i = \dfrac{x''_{i+1} - x''_i}{6 \cdot h_i} \end{cases} \qquad (i = 1, ..., n - 1)$$

4.3.5 Calcul des c_i

On a $x_{i+1} = a_i \cdot h_i^3 + b_i \cdot h_i^2 + c_i \cdot h_i + d_i$, ce qui donne avec (4.3.3) et (4.3.4) :

$$c_i = \frac{x_{i+1} - x_i}{h_i} - \frac{x''_i}{2} \cdot h_i - \frac{x''_{i+1} - x''_i}{6} \cdot h_i$$

soit :

$$c_i = \frac{x_{i+1} - x_i}{h_i} - \frac{h_i}{2} \cdot \frac{x''_{i+1} + 2 \cdot x''_i}{3}$$

On calcule donc les coefficients des polynômes s_i grâce aux formules ci-dessus en fonction des x''_i. Il reste donc à déterminer ces derniers.

4.3.6 Calcul des $s_i = x''_i$

Avec les conditions $y'_i = s'(t_i) = s'_{i-1}(t_i) = s'_i(t_i)$, on déduit que les x''_i sont solutions du système :

$$h_{i-1} \cdot x''_{i-1} + 2 \cdot (h_{i-1} + h_i) \cdot x''_i + h_i \cdot x''_{i+1} = 6 \cdot \left(\frac{x_{i+1} - x_i}{h_i} - \frac{x_i - x_{i-1}}{h_{i-1}} \right)$$

pour $i = 2, ..., n - 1$.

Soit un système de $n - 2$ équations à n inconnues. Il nous manque donc deux équations. On complète alors en posant des *conditions aux limites* qui devront fournir deux équations indépendantes des $n - 2$ précédentes. Classiquement, on a les conditions suivantes :

Splines naturels — On pose $x''_1 = x''_n = 0$;

Splines périodiques — Si la fonction considérée est périodique, on aura $x''(t_1) = x''(t_n)$ et il nous manque seulement une équation que l'on peut obtenir en disant que dans le système ci-dessus on peut faire i = n (Cf. § (4.3.10)) ;
Conditions de De Boor — On impose une condition de continuité C^3 en t_2 et t_{n-1}, ce qui se traduit par $a_1 = a_2$ et $a_{n-1} = a_n$ et fournit les deux équations manquantes sur les s_i.

Remarque — Ces conditions peuvent aussi porter sur des points intérieurs, mais dans tous les cas des conditions différentes donneront des courbes d'allures différentes. On choisira ces conditions en fonction de la nature du problème.

4.3.7 L'interpolation spline naturelle

On pose donc, dans ce paragraphe : $x''_1 = x''_n = 0$ et les inconnues sont donc $x''_2, ..., x''_{n-1}$.

Pour se ramener à des notations plus conventionnelles (inconnues numérotées de 1 à p), on pose :

$$u_i = x''_{i+1} \ (i = 1, 2, ..., p = n - 2)$$

avec :

$$u_0 = u_{p+1} = 0$$

Alors le vecteur u de composantes u_i dans R^p est solution du système tridiagonal :

$$T \cdot u = v$$

avec :

$$T = \begin{pmatrix} T_{12} & T_{13} & 0 & ... & 0 & 0 \\ T_{21} & T_{22} & T_{23} & 0 & ... & 0 \\ . & . & . & . & . & . \\ . & . & . & . & . & . \\ 0 & . & . & 0 & T_{p1} & T_{p2} \end{pmatrix} \quad v = \begin{pmatrix} v_1 \\ . \\ . \\ . \\ v_p \end{pmatrix}$$

où :

$$\begin{cases} T_{i1} = h_i \ \ (i = 2, ..., p) \\ T_{i2} = 2 \cdot (h_i + h_{i+1}) \ \ (i = 1, ..., p) \\ T_{i3} = h_{i+1} \ \ (i = 1, ..., p - 1) \end{cases}$$

$$v_i = 6 \cdot \left(\frac{x_{i+2} - x_{i+1}}{h_{i+1}} - \frac{x_{i+1} - x_i}{h_i} \right) \ \ (i = 1, ..., p)$$

En remarquant que, pour tout i = 1, ..., p (où on a posé : $T_{12} = T_{p3} = 0$) :

$$|T_{i2}| > |T_{i1}| + |T_{i3}|$$

on déduit que la matrice T est à diagonale strictement dominante, ce qui entraîne que le système admet une unique solution. De plus si on utilise la méthode de résolution de Gauss, il n'est pas nécessaire de faire des permutations de lignes de sorte que l'on pourra utiliser la méthode de double balayage de Cholesky pour résoudre ce système (Cf. le chapitre « Analyse Numérique Linéaire », § (3.7.3)).

4.3.8 Programmation structurée pour l'interpolation spline naturelle

Tout d'abord, on écrit une procédure de formation du système donnant les x''_i.

PROCEDURE SystèmeSplineNaturel(Entrée n ; Entier ; x, y : Vecteur ;
Sortie T : MatriceTridiagonale ; v : Vecteur) ;
Début
Pour i allant de 1 à n – 2 Faire
Début
$T_{i1} = x_{i+1} - x_i$;
$T_{i3} = x_{i+2} - x_{i+1}$;
$T_{i2} = 2 \cdot (T_{i1} + T_{i3})$;
$v_i = 6 \cdot \left(\frac{y_{i+2} - y_{i+1}}{T_{i3}} - \frac{y_{i+1} - y_i}{T_{i1}} \right)$;
Fin ;
Fin ;

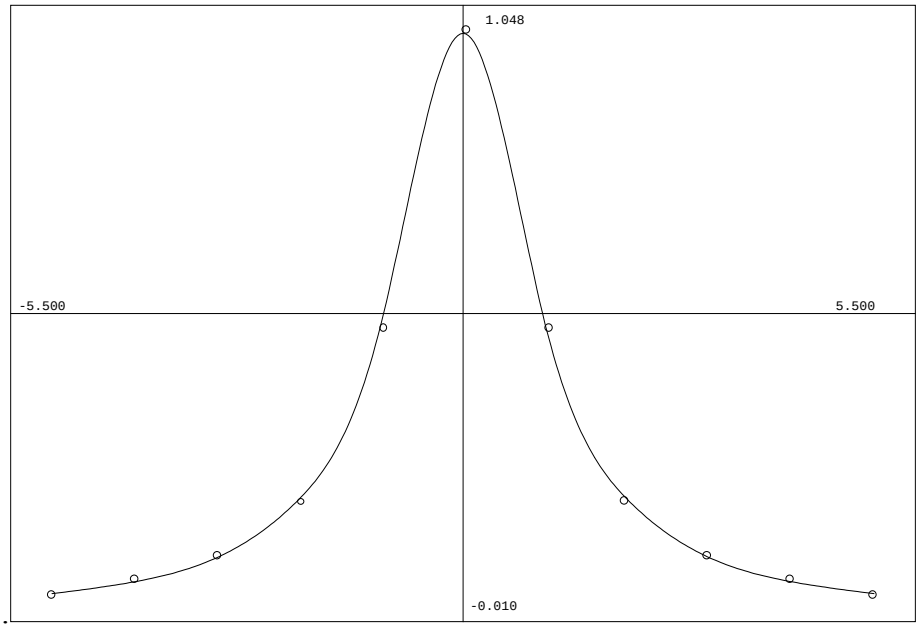
Enfin on écrit une procédure qui donne les coefficients des polynômes s_i en fonction des x''_i.

PROCEDURE CalculCoeff(Entrée n : Entier ; x, y,s : Vecteur ; Sortie a, b, c, d : Vecteur) ;
Début
Pour i allant de 1 à n – 1 Faire
Début
$h = x_{i+1} - x_i$;
$a_i = \frac{s_{i+1} - s_i}{6 \cdot h}$;
$b_i = \frac{s_i}{2}$;
$c_i = \frac{y_{i+1} - y_i}{h} - h \cdot \frac{2 \cdot s_i + s_{i+1}}{6}$;
$d_i = y_i$;
Fin ;
Fin ;

Le calcul des coefficients, dans le cas des splines naturels, se fait alors simplement avec la :

PROCEDURE CalculCoeffSplineNaturel(Entrée n : Entier ; x, y : Vecteur ;
Sortie a, b, c, d : Vecteur) ;
Début
SystemeSplineNaturel(n,x,y,T,v) ;
SystTridiag(n–2,T,v,s) ;
Pour i allant de n – 1 à 2 faire $s_i = s_{i-1}$;
$s_1 = s_n = 0$;
CalculCoeff(n,x,y,s,a,b,c,d) ;
Fin ;

En reprenant l'exemple du paragraphe (4.2.4), on obtient le résultat de la figure 5.13.



5.13.

Figure 5.13

4.3.9 Application à un calcul d'intégrale

En notant S_i une primitive de s_i, une approximation de l'intégrale de la fonction interpolée est donnée par :

$$I = \sum_{i=1}^{n-1} \left(S_i(t_{i+1}) - S_i(t_i) \right)$$

avec :

$$S_i(t) = \frac{a_i}{4} \cdot (t - t_i)^4 + \frac{b_i}{3} \cdot (t - t_i)^3 + \frac{c_i}{2} \cdot (t - t_i)^2 + d_i \cdot (t - t_i)$$

Ce qui donne, en tenant compte des calculs précédents, la formule d'intégration approchée :

$$I = \sum_{i=1}^{n-1} \frac{x_{i+1} + x_i}{2} \cdot h_i - \frac{1}{24} \cdot \sum \left(x_i'' + 2 \cdot x_{i+1}'' \right) \cdot h_i^3$$

On peut alors compléter la procédure CalculCoeff en ajoutant une variable de sortie Int, qui donnera la valeur de l'intégrale du spline. Le calcul se faisant comme suit :

Int1 = 0 ; Int2 = 0 ;

Dans la boucle sur i, on ajoute les instructions :

h3 = h^3 ;

Int1 = Int1 + $(y_i + y_{i+1}) \cdot h$;

Int2 = $(s_i + 2 \cdot s_{i+1}) \cdot h3$;

Ce qui donne, après la fin de la boucle sur i : Int = Int1/2 – Int2/24 ;

4.3.10 Cas des fonctions périodiques

On suppose donc ici que le nuage de points provient d'une fonction périodique de période $P = t_n - t_1$ et de classe C^2.

Le système de n – 2 équations à n inconnues du paragraphe (4.3.6) :

$$h_{i-1}\cdot x''_{i-1} + 2\cdot(h_{i-1} + h_i)\cdot x''_i + h_i\cdot x''_{i+1} = 6\cdot\left(\frac{x_{i+1} - x_i}{h_i} - \frac{x_i - x_{i-1}}{h_{i-1}}\right)$$

est encore valable pour i = 2, ..., n – 1.

Comme $x''_1 = x''_n$, il nous manque ici seulement une équation. Pour obtenir cette dernière, on demande que (4.3.6) soit encore valable pour i = n, en posant, du fait de la périodicité :

$$x_n = x_1,\ x''_1 = x''_n,\ h_n = h_1,\ x''_{n+1} = x''_2,\ h_{n+1} = h_2 \text{ et } x_{n+1} = x_2$$

L'équation N° (n – 1) (correspondante à i = n) va donc s'écrire :

$$h_1 \cdot x''_2 + h_{n-1} \cdot x''_{n-1} + 2\cdot(h_{n-1} + h_1)\cdot x''_n = 6\cdot\left(\frac{x_2 - x_1}{h_1} - \frac{x_1 - x_{n-1}}{h_{n-1}}\right)$$

et l'équation N° 1 s'écrit :

$$2\cdot(h_1 + h_2)\cdot x''_2 + h_2 \cdot x''_3 + h_1 \cdot x''_n = 6\cdot\left(\frac{x_3 - x_2}{h_2} - \frac{x_2 - x_1}{h_1}\right)$$

Là encore, en posant $u_i = x''_{i+1}$ pour i = 1, ..., p = n - 1, le vecteur u est solution d'un système de p équations à p inconnues du type :

$$T\cdot u = v \qquad (*)$$

avec :

$$T = \begin{pmatrix} T_{12} & T_{13} & 0 & \dots & 0 & T_{11} \\ T_{21} & T_{22} & T_{23} & 0 & \dots & 0 \\ . & . & . & . & . & . \\ . & . & . & . & . & . \\ T_{p3} & 0 & . & 0 & T_{p1} & T_{p2} \end{pmatrix} \quad v = \begin{pmatrix} v_1 \\ . \\ . \\ . \\ v_p \end{pmatrix}.$$

où :

$$\begin{cases} T_{i1} = h_i \\ T_{i2} = 2\cdot(h_i + h_{i+1}) \quad (i = 1, \dots, p) \\ T_{i3} = h_{i+1} \end{cases}$$

$$v_i = 6\cdot\left(\frac{x_{i+2} - x_{i+1}}{h_{i+1}} - \frac{x_{i+1} - x_i}{h_i}\right) \quad (i = 1, \dots, p)$$

Là encore la matrice A est symétrique définie positive et le système (*) admet une unique solution.

Avec ces notations, la construction de la matrice T est identique à celle utilisée dans le cas du spline naturel, à la différence que les coefficients T_{11} et T_{p3} interviendront dans la résolution du système linéaire.

Pour résoudre un tel système, on considère l'inconnue u_p comme un paramètre et on retire un instant la dernière ligne pour se ramener au cas du paragraphe (3.7).

La résolution de ce système se fait en utilisant la méthode des pivots de Gauss.

En notant γ la dernière colonne de T, soit :

$$\gamma_1 = T_{11},\ \gamma_0 = 0\ (i = 2, ..., p-2),\ \gamma_{p-1} = T_{p-1,3},\ \gamma_p = T_{p2}$$

on a alors les formules de résolution suivantes :

$$m = \frac{T_{i,1}}{T_{i-1,2}}\ ;\ T_{i,1} = 0\ ;$$

$T_{i,2} = T_{i,2} - m \cdot T_{i-1,3}$;

$T_{i,3}$ inchangé

$\gamma_i = \gamma_i - m \cdot \gamma_{i-1}$;

$v_i = v_i - m \cdot v_{i-1}$

$T_{p,2} = T_{p,2} - m \cdot \gamma_{i-1}$

pour $i = 2, ..., p-1$.

Ce qui transforme le système en système triangulaire supérieur qui se résout « en remontée », ce qui donne :

$$\begin{cases} u_p = \dfrac{v_p}{T_{p,2}} \\ u_i = \dfrac{v_i - T_{i,3} \cdot u_{i+1} - \gamma_i \cdot u_p}{T_{i,2}} \quad (i = p-1, ..., 1) \end{cases}$$

D'où la programmation structurée :

```
PROCEDURE SystemeSplinePeriodique(Entrée n : Entier ; P : Réel ; x, y : Vecteur ;
                                  Sortie T : MatriceTridiagonale ; v : Vecteur) ;
{ T n'est pas vraiment tridiagonale, c'est la donnée de trois vecteurs }
Début
   x_{n+1} = x_2 + P ;
   y_{n+1} = y_2 ;
   SystemeSplineNaturel(n + 1,x,y,T,v) ;
Fin ;

PROCEDURE ResolutionSystemeSplinePeriodique(
                  Entrée p : Entier ; T : MatriceTridiagonale ; v : Vecteur ;
                  Sortie u : Vecteur) ;
Début
   γ_1 = T_11, γ_{p-1} = T_{p-1,3}, γ_p = T_p2 ;
   Pour i allant de 2 à p – 2 faire γ_i = 0 ;
   Pour i allant de 2 à p – 1 faire
   Début
      m = T_{i,1} / T_{i-1,2} ;
      T_{i,2} = T_{i,2} – m·T_{i-1,3} ;
      γ_i = γ_i – m·γ_{i-1} ;
      v_i = v_i – m·v_{i-1} ;
      T_{p,2} = T_{p,2} – m·γ_{i-1}
   Fin ;
   Pour i allant de p – 1 à 1 faire u_i = (v_i – T_{i,3}·u_{i+1} – γ_i·u_p) / T_{i,2}
Fin ;
```

```
PROCEDURE CalculCoeffSplinePeriodique(Entrée n : Entier ; P : Réel ; x, y : Vecteur ;
                                      Sortie a, b, c, d : Vecteur) ;
Début
   SystemeSplinePeriodique(n,P,x,y,T,v) ;
   ResolutionSystemeSplinePeriodique(n – 1,T,v,s) ;
   Pour i allant de n à 2 faire s_i = s_i-1 ;
   s_1 = s_n ;
   CalculCoeff(n,x,y,s,a,b,c,d) ;
Fin ;
```

4.3.11 Interpolation spline pour les courbes fermées

Si le nuage de points correspond à une courbe fermée, soit $(x_1,y_1) = (x_n,y_n)$, une idée pour réaliser l'interpolation spline est alors de passer en coordonnées paramétriques. Si la courbe est paramétrée par x = x(t) et y = y(t) où t est l'angle polaire avec l'origine ramenée au centre de gravité du nuage de points, alors il suffit d'utiliser l'interpolation spline pour chacune des fonctions périodiques de période $2\cdot\pi$ x(t) et y(t), si on suppose que la courbe est de classe C^2 et sans points doubles.

On calcule donc d'abord les coordonnées du centre de gravité :

$$(xg,yg) = \frac{1}{n-1}\cdot\left(\sum_{i=1}^{n-1} x_i, \sum_{i=1}^{n-1} y_i\right)$$

Puis on écrit une procédure de passage en coordonnées polaires, en prenant soin de ranger ensuite les points par angles croissants.

Ensuite on utilise la procédure d'interpolation spline pour x et y en fonction de l'angle polaire t, ce qui donne deux fonctions polynomiales par morceaux s_x et s_y.

Et enfin on trace la courbe d'équation polaire :

$$\begin{cases} x = xg + s_s(t) \\ y = yg + s_y(t) \end{cases}$$

Une autre façon est de prendre $t \in [0,n]$ comme paramètre en posant $x(i) = x_i$ et $y(i) = y_i$, puis on utilise la procédure d'interpolation spline périodique pour x et y séparément, ce qui donne une autre représentation paramétrique de l'interpolée spline (et une courbe d'allure différente en général). Cela permet de traiter le cas des courbes avec points doubles.

La figure 5.14 donne un exemple de tracé correspondant au nuage de points du paragraphe (3.4.3).

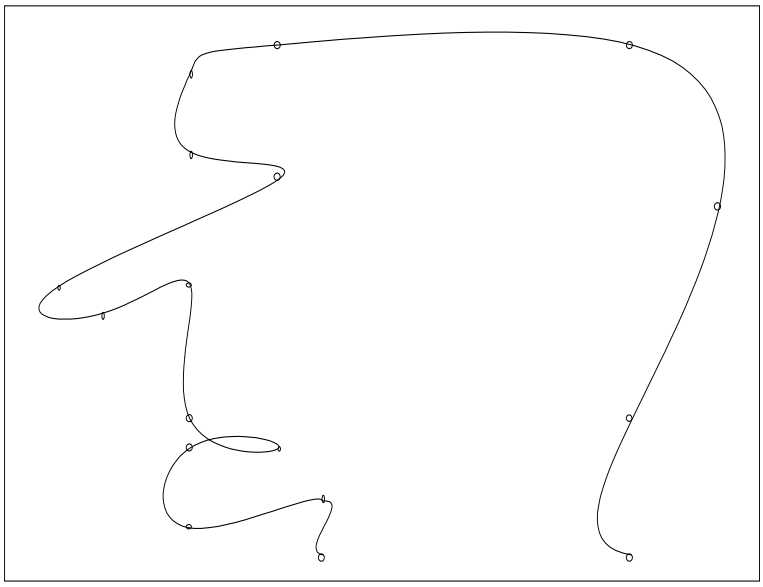


Figure 5.14

4.3.12 Majoration de l'erreur d'interpolation

Si le nuage de points S provient d'une fonction f de classe C^4 sur [0,1], avec les x_i dans [0,1] (par changement de variable, on peut toujours se ramener à ce cas), on peut alors montrer que, si s est la fonction spline obtenue, on a les majorations suivantes, où $\|\cdot\|$ désigne la norme du sup. pour les fonctions bornées sur [0,1] :

$$\|f - s\| \leq C_1 \cdot h^4$$
$$\|f' - s'\| \leq C_2 \cdot h^3$$
$$\|f'' - s''\| \leq C_3 \cdot h^2$$

avec :

$$C_1 = \frac{13}{64} \cdot \left\|f^{(4)}\right\| , \quad C_2 = C_3 = \frac{13}{8} \cdot \left\|f^{(4)}\right\|$$

(Cf. Alhlberg, Nilson et Walsh, § (2.3)).

5. Exercices

5.1 Ellipse, plan et sphère des moindres carrés

Ecrire les procédures correspondantes aux problèmes suivants :
(i) Recherche d'une ellipse des moindres carrés ;
(ii) Recherche d'un plan des moindres carrés ;
(iii) Recherche d'une sphère des moindres carrés.

5. 2 Interpolation par fractions continues

Première partie

Si $(a_j)_{j\geq 1}$ et $(b_j)_{j\geq 1}$ sont deux suites de fonctions, les fractions continues associées sont définies par :

$$f_1(x) = a_1(x)$$

$$f_n(x) = a_1(x) + \cfrac{b_1(x)}{a_2(x) + \cfrac{b_2(x)}{a_3(x) + \cfrac{b_3(x)}{\ddots \cfrac{}{a_{n-1}(x) + \cfrac{b_{n-1}(x)}{a_n(x)}}}}}$$

Cette fraction peut aussi s'écrire :

$$f_n(x) = \frac{P_n(x)}{Q_n(x)} \quad (n \geq 1)$$

1°) Donner P_n et Q_n pour : (a) n = 1 ; (b) n = 2 ; (c) n = 3.
2°) Montrer par récurrence que les suites (P_n) et (Q_n) peuvent être définies par :

$$\begin{cases} P_0(x) = 1 \\ P_1(x) = a_1(x) \end{cases} ; \begin{cases} Q_0(x) = 0 \\ Q_1(x) = 1 \end{cases}$$

$$\begin{cases} P_n(x) = a_n(x) \cdot P_{n-1}(x) + b_{n-1}(x) \cdot P_{n-2}(x) \\ Q_n(x) = a_n(x) \cdot Q_{n-1}(x) + b_{n-1}(x) \cdot Q_{n-2}(x) \end{cases} \quad (n \geq 2)$$

3°) En probabilités, la fonction de répartition de la loi Gamma de paramètre $\alpha > 0$ est définie par :

$$\forall\, x > 0,\ P(\alpha,x) = 1 - Q(\alpha,x)$$

où :

$$Q(\alpha,x) = \frac{1}{\Gamma(\alpha)} \cdot \int_x^{+\infty} e^{-t} \cdot t^{\alpha-1} dt \text{, avec : } \Gamma(\alpha) = \int_0^{+\infty} e^{-t} \cdot t^{\alpha-1} dt$$

On peut montrer que la fonction Q admet le développement :

$$Q(\alpha,x) = \frac{e^{-x}\cdot x^{\alpha}}{\Gamma(\alpha)}\cdot F(x) \text{ où : } F(x) = \lim_{n\to+\infty}\left(f_n(x)\right)$$

les f_n étant les fractions continues définies par :

$$\begin{cases} a_1 = 0 \\ a_{2k} = x \quad (k \geq 1) \\ a_{2k+1} = 1 \quad (k \geq 1) \end{cases} ; \quad \begin{cases} b_1 = 1 \\ b_{2k} = k - \alpha \quad (k \geq 1) \\ b_{2k+1} = k \quad (k \geq 1) \end{cases}$$

Ecrire une fonction qui permet de calculer une valeur approchée de $P(\alpha,x)$ à une précision $\varepsilon > 0$ donnée.

Deuxième partie

Soit $S = \{ P_i = (x_i,y_i) ; 1 \leq i \leq n \}$ un nuage de points expérimentaux.

On cherche à interpoler S par une fonction rationnelle définie par la fraction continue :

$$P(x) = a_1 + \cfrac{x - x_1}{a_2 + \cfrac{x - x_2}{a_3 + \cfrac{x - x_3}{\cdots\cdots + \cfrac{x - x_{n-1}}{a_n}}}}$$

En vue de calculer les a_i, on définit la suite (b_{ij}) par :

$$b_{i1} = y_i \ (1 \leq i \leq n)$$

$$b_{i,j+1} = \frac{x_i - x_j}{b_{i,j} - b_{j,j}} \quad (1 \leq j < i \leq n) \ (1 < j < i < n)$$

1°) (a) Calculer les b_{ij} pour n = 3.
(b) Quel lien a-t-on entre les a_i et les b_{ij} , pour n = 3?
2°) Ecrire une procédure de calcul des b_{ij} .
3°) Montrer par récurrence sur i que $a_i = b_{ii}$ (i = 1, ..., n).
4°) Ecrire une fonction qui permet de calculer P(x) pour tout x.
5°) Ecrire un programme permettant de tracer le graphe d'une fonction f et de son interpolée en prenant des abscisses x_i équidistantes.

5.3 Interpolation spline d'ordre 5

Ecrire les procédures correspondantes à l'interpolation spline de degré au plus 5.

6. Programmation Ada

6.1 Spécification du paquetage REGRESSIONS

Dans ce paquetage on trouve la programmation des procédures d'approximation du paragraphe 2.

```
with COMMON_MATRIX, COMMON_POLY ;
use COMMON_MATRIX, COMMON_POLY ;
package REGRESSIONS is
procedure REGRESSION_AFFINE(x,y : in VECTEUR ; a,b,Rho : in out FLOAT);
procedure REGRESSION_POLYNOMIALE(x, y : in VECTEUR ; U : out POLYNOME);
procedure REGRESSION_CIRCULAIRE(x,y : in VECTEUR ;
             a,b,r : in out FLOAT) ;
end REGRESSIONS ;
```

6.2 Le paquetage DONNEES_POINTS

Dans ce paquetage on décrit des procédures permettant l'entrée de nuages de points.

```
with COMMON_MATRIX ;
use COMMON_MATRIX ;
package DONNEES_POINTS is
procedure GET(x, y : in out VECTEUR ; ORDRE : in BOOLEAN := FALSE) ;
procedure POINTS_DU_FICHIER(x, y : in out VECTEUR ;
             ORDRE : in BOOLEAN := FALSE;
             NOM_DE_REPERTOIRE : in STRING := "A :\DONNEES") ;
procedure ENTRER_POINTS(x, y : in out VECTEUR ;
             ORDRE : in BOOLEAN := FALSE) ;
procedure EXTREMA_POINTS(x, y : in VECTEUR ;
             xMin, xMax, yMin, yMax : in out FLOAT) ;
end DONNEES_POINTS ;
```

6.3 Démonstration de la régression affine

```
with TEXT_IO, FIO, COMMON_MATH0, MATH0, MATH3, CURVE,
     COMMON_MATRIX,  REGRESSIONS, DONNEES_POINTS ;
use TEXT_IO, FIO, COMMON_MATH0, MATH0, MATH3, CURVE,
     COMMON_MATRIX, REGRESSIONS, DONNEES_POINTS ;
procedure DEM_AFF is
n : NATURAL ;
procedure TRACE_REGRESSION_AFFINE(x, y : in VECTEUR ;
             a, b : in FLOAT) is
xMin,yMin,xMax,yMax,xOrigine,yOrigine,xUnite,yUnite : FLOAT ;
xFenMin, xFenMax, yFenMin, yFenMax : FLOAT ;
begin
    EXTREMA_POINTS(x,y,xMin,xMax,yMin,yMax) ;
    AXES_PAR_DEFAUT(xMin,yMin,xMax,yMax,xOrigine,yOrigine,
                    xUnite,yUnite,xFenMin, xFenMax, yFenMin, yFenMax) ;
    INIT_GRAPHIQUE ;
    FENETRE_GRAPHIQUE(xFenMin,xFenMax,yFenMin,yFenMax) ;
    XY_AXES(xOrigine,yOrigine,xUnite,yUnite,FALSE) ;
```

```
    for j in x'range loop
       CROIX(x(j),y(j)) ;
    end loop ;
    DEPLACE(xMin,a*xMin + b) ; TRACE(xMax,a*xMax + b) ;
    SORTIE_GRAPHIQUE ;
end TRACE_REGRESSION_AFFINE ;
begin
    MODE_AFFICHAGE ;
    ENTRER_ENTIER_BORNE(2,DIM_MAX,n,"Nombre de points du nuage : ") ;
    declare
       x, y : VECTEUR(1..n) ;
       a, b, Rho : FLOAT ;
    begin
       ENTRER_POINTS(x,y) ;
       REGRESSION_AFFINE(x,y,a,b,Rho) ;
       PUT_LINE(IMP,"DROITE DES MOINDRES CARREES") ;
       PUT(IMP,"   y = ") ; FIO.PUT(IMP,a,6,4,0) ;
       PUT(IMP,"*x + ") ; FIO.PUT(IMP,b,6,4,0) ; NEW_LINE(IMP) ;
       PUT(IMP,"Coefficient de regression : Rho = ") ;
       FIO.PUT(IMP,Rho,6,4,0) ; NEW_LINE(IMP) ; PAUSE ;
       TRACE_REGRESSION_AFFINE(x,y,a,b) ;
    end ;
    CLOSE(IMP) ;
end DEM_AFF ;
```

6.4 Démonstration de la régression polynomiale

```
with TEXT_IO, COMMON_MATH0, MATH0, MATH3, CURVE, COMMON_MATRIX,
     COMMON_POLY, POLY, REGRESSIONS, DONNEES_POINTS ;
use TEXT_IO, COMMON_MATH0, MATH0, MATH3, CURVE, COMON_MATRIX,
     COMMON_POLY, POLY, REGRESSIONS, DONNEES_POINTS ;
procedure DEM_RPOL is
n : NATURAL ;
p : DEGRE_MAX ;
begin
    MODE_AFFICHAGE ;
    ENTRER_ENTIER_BORNE(2,DIM_MAX,n,"Nombre de points du nuage : ") ;
    ENTRER_ENTIER_BORNE(1,n - 1,p,
                        "Degre du polynome des moindres carres : ") ;
    declare
       x, y : VECTEUR(1..n) ;
       U : POLYNOME(0..p) ;
       xMin, xMax, yMin, yMax : FLOAT ;
    begin
       ENTRER_POINTS(x,y) ;
       REGRESSION_POLYNOMIALE(x,y,U) ;
       TEXT_IO.PUT_LINE(IMP,"POLYNOME DES MOINDRES CARREES") ;
       TEXT_IO.PUT(IMP,"P(x) = ") ; PUT_LINE(U) ; NEW_LINE(IMP) ;
       PAUSE ;
       EXTREMA_POINTS(x,y,xMin,xMax,yMin,yMax) ;
       TRACE_POLYNOME(U,xMin,xMax) ;
       for j in x'range loop
          CROIX(x(j),y(j)) ;
```

```
       end loop ;
       SORTIE_GRAPHIQUE ;
    end ;
    CLOSE(IMP) ;
end DEM_RPOL ;
```

6.5 Démonstration de la régression circulaire

```
with TEXT_IO, FIO, COMMON_MATH0, MATH0, MATH1, MATH3, CURVE,
     COMMON_MATRIX,  REGRESSIONS, DONNEES_POINTS ;
use TEXT_IO, FIO, CMMON_MATH0, MATH0, MATH1, MATH3, CURVE,
    COMMON_MATRIX,  REGRESSIONS, DONNEES_POINTS ;
procedure DEM_CIRC is
n : NATURAL ;
procedure TRACE_REGRESSION_CIRCULAIRE(x, y : in VECTEUR ;
                                      a, b, r : in FLOAT) is
xMin,yMin,xMax,yMax : FLOAT ;
begin
    EXTREMA_POINTS(x,y,xMin,xMax,yMin,yMax) ;
    xMin := MIN(xMin,a - r) - 0.1*r ; xMax := MAX(xMax,a + r) + 0.1*r ;
    yMin := MIN(yMin,b - r) - 0.1*r ; yMax := MAX(yMax,b + r) + 0.1*r ;
    INIT_GRAPHIQUE ;
    FENETRE(xMin,xMax,yMin,yMax) ;
    CADRE_GRAPHIQUE(0,Y_MAXIMUM - 2*MARGE,MARGE, Y_MAXIMUM - MARGE) ;
    XY_AXES(a,b,0.1*r,0.1*r,FALSE) ;
    for j in x'range loop
       CROIX(x(j),y(j)) ;
    end loop ;
    CERCLE(a,b,r) ;
    SORTIE_GRAPHIQUE ;
end TRACE_REGRESSION_CIRCULAIRE ;
begin
    MODE_AFFICHAGE ;
    ENTRER_ENTIER_BORNE(2,DIM_MAX,n,"Nombre de points du nuage : ") ;
    declare
       x, y : VECTEUR(1..n) ;
       a, b, r : FLOAT ;
    begin
       ENTRER_POINTS(x,y) ;
       REGRESSION_CIRCULAIRE(x,y,a,b,r) ;
       PUT_LINE(IMP,"CERCLE DES MOINDRES CARREES") ;
       PUT(IMP,"   Centre, (a,b) = (") ; FIO.PUT(IMP,a,6,4,0) ;
       PUT(IMP,",") ; FIO.PUT(IMP,b,6,4,0) ; PUT_LINE(IMP,")") ;
       PUT(IMP,"Rayon, r = ") ; FIO.PUT(IMP,r,6,4,0) ; NEW_LINE(IMP) ;
       PAUSE ;
       TRACE_REGRESSION_CIRCULAIRE(x,y,a,b,r) ;
    end ;
    CLOSE(IMP) ;
end DEM_CIRC ;
```

6.6 Spécification du paquetage BEZIER

```
with CURVE, COMMON_MATRIX ;
```

```
use CURVE, COMMON_MATRIX ;
package BEZIER is
function POLYNOME_BERNSTEIN(P : in VECTEUR ; t : in FLOAT)
             return FLOAT ; -- § 3.2
procedure CALCUL_BEZIER(x,y : in VECTEUR ; t : in FLOAT ;
             Bx,By : out FLOAT) ; -- § 3.2
procedure TRACE_BASE_BERNSTEIN(n : in NATURAL) ; -- § 3.2
procedure COURBE_BEZIER(x, y : in VECTEUR ;
             C : in out COURBE) ; -- § 3.3
end BEZIER ;
```

6.7 Démonstration de l'approximation de Bézier

```
with TEXT_IO, MATH0, MATH3, CURVE, COMMON_MATRIX,
     DONNEES_POINTS, BEZIER ;
use TEXT_IO, MATH0, MATH3, CURVE, COMMON_MATRIX,
    DONNEES_POINTS, BEZIER ;
procedure DEM_BEZI is
CHOIX : NATURAL range 0..2 ;
n, j : NATURAL ;
procedure MENU_BEZIER(CHOIX : out NATURAL) is
begin
    PUT_LINE("-0- FIN") ;
    PUT_LINE("-1- Trace des fonctions de base de Bernstein") ;
    PUT_LINE(
     "-2- Tracer une courbe de Bezier a partir d'un nuage de points") ;
    NEW_LINE ;
    ENTRER_ENTIER_BORNE(0,2,CHOIX,"Votre choix : ") ;
end MENU_BEZIER ;
begin
   loop
      MENU_BEZIER(CHOIX) ;
      case CHOIX is
        when 0 => exit ;
        when 1 =>
   ENTRER_ENTIER_BORNE(2,DIM_MAX,n,"Nombre de fonctions a tracer : ") ;
           n := n - 1 ;
           TRACE_BASE_BERNSTEIN(n) ;
        when 2 =>
ENTRER_ENTIER_BORNE(3,DIM_MAX,n,"Nombre de points du nuage (> 2) : ") ;
             n := n - 1 ;
             declare
               x, y : VECTEUR(0..n) ;
               C : COURBE(300) ;
             begin
               ENTRER_POINTS(x,y,FALSE) ;
               COURBE_BEZIER(x, y, C) ;
               TRACE_UNE_COURBE(C) ;
               SORTIE_GRAPHIQUE ;
             end ;
      end case ;
   end loop ;
end DEM_BEZI ;
```

6.8 Spécification du paquetage B_SPLINE

```
with CURVE, COMMON_MATRIX ;
use CURVE, COMMON_MATRIX ;
package B_SPLINE is
type VECTEUR_NODAL is array(NATURAL range <>) of NATURAL ;
function BASE_BSPLINE(i, j : NATURAL ; Ne : VECTEUR_NODAL ; t : FLOAT)
            return FLOAT ; -- § 3.4.2
procedure CALCUL_BSPLINE(j : in NATURAL ; Ne : in VECTEUR_NODAL ;
   x, y : in VECTEUR ; t : in FLOAT ; Bx, By : out FLOAT) ; -- § 3.4.3
procedure COURBE_BSPLINE(j : in NATURAL ; x, y : in VECTEUR ;
            C : in out COURBE) ; -- § 3.4.3
procedure TRACE_BASE_BSPLINE(n, j : in NATURAL) ; -- § 3.4.2
end B_SPLINE ;
```

6.9 Démonstration de l'approximation B_Spline

```
with TEXT_IO, MATH0, MATH3, CURVE, COMMON_MATRIX,
     DONNEES_POINTS, B_SPLINE ;
use TEXT_IO, MATH0, MATH3, CURVE, COMMON_MATRIX,
    DONNEES_POINTS, B_SPLINE ;
procedure DEM_BSPL is
CHOIX : NATURAL range 0..2 ;
n, j : NATURAL ;
procedure MENU_BSPLINE(CHOIX : out NATURAL) is
begin
    PUT_LINE("-0- FIN") ;
    PUT_LINE("-1- Trace des fonctions de base B_Splines") ;
    PUT_LINE(
     "-2- Tracer une courbe B_Spline a partir d'un nuage de points") ;
    NEW_LINE ;
    ENTRER_ENTIER_BORNE(0,2,CHOIX,"Votre choix : ") ;
end MENU_BSPLINE ;
begin
   loop
      MENU_BSPLINE(CHOIX) ;
      case CHOIX is
        when 0 => exit ;
        when 1 =>
   ENTRER_ENTIER_BORNE(2,DIM_MAX,n,"Nombre de fonctions a tracer : ") ;
           n := n - 1 ;
           ENTRER_ENTIER_BORNE(2,n + 1,j,"Ordre des B_Splines : ") ;
           TRACE_BASE_BSPLINE(n,j) ;
        when 2 =>
ENTRER_ENTIER_BORNE(3,DIM_MAX,n,"Nombre de points du nuage (> 2) : ") ;
             n := n - 1 ;
             ENTRER_ENTIER_BORNE(2,n + 1,j,"Ordre de la B_Spline : ");
             declare
               x, y : VECTEUR(0..n) ;
               C : COURBE(300) ;
             begin
               ENTRER_POINTS(x,y,FALSE) ;
```

```
               COURBE_BSPLINE(j, x, y, C) ;
               TRACE_UNE_COURBE(C) ;
               SORTIE_GRAPHIQUE ;
             end ;
      end case ;
   end loop ;
end DEM_BSPL ;
```

6.10 Spécification du paquetage LAGRANGE

```
with CURVE, COMMON_MATRIX ;
use CURVE, COMMON_MATRIX ;
package LAGRANGE is
function LAGRANGE_NEVILLE(x, y : VECTEUR ; t : FLOAT)
             return FLOAT ; -- § 4.2.3
procedure POLYNOME_LAGRANGE(x, y : in VECTEUR ;
             P : out VECTEUR) ; -- § 4.2.3
function LAGRANGE_NEWTON(x, P : in VECTEUR ; t : in FLOAT)
             return FLOAT ; -- § 4.2.4
procedure COURBE_INTERPOLATION_LAGRANGE(x,y : in VECTEUR ;
             xMin, xMax : in FLOAT ; C : in out COURBE) ;
end LAGRANGE ;
```

6.11 Démonstration de l'interpolation de Lagrange

```
with TEXT_IO, MATH0, COMMON_MATH1, MATH1, MATH2, MATH3, CURVE,
     COMMON_MATRIX, LAGRANGE, DONNEES_POINTS ;
use TEXT_IO, MATH0, COMMON_MATH1, MATH1, MATH2, MATH3, CURVE,
    COMMON_MATRIX, LAGRANGE, DONNEES_POINTS ;
procedure DEM_LAGR is
CHOIX : NATURAL range 0..2 ;
n : NATURAL ;
f : FONCTION ;
procedure MENU_LAGRANGE(CHOIX : out NATURAL) is
begin
    PUT_LINE("-0- FIN") ;
    PUT_LINE(
        "-1- Interpolation d'une fonction connue en quelques points") ;
    PUT_LINE("-2- Interpolation d'une fonction donnee analytiquement");
    NEW_LINE ;
    ENTRER_ENTIER_BORNE(0,2,CHOIX,"Votre choix : ") ;
end MENU_LAGRANGE ;
begin
   loop
      MENU_LAGRANGE(CHOIX) ;
      case CHOIX is
        when 0 => exit ;
        when 1 =>
ENTRER_ENTIER_BORNE(3,DIM_MAX,n,"Nombre de points du nuage (> 2) : ") ;
             declare
               x, y : VECTEUR(0..n) ;
               C : COURBE(300) ;
             begin
```

```
               ENTRER_POINTS(x,y,TRUE) ;
               COURBE_INTERPOLATION_LAGRANGE(x, y, x(1), x(n), C) ;
               TRACE_UNE_COURBE(C) ;
               SORTIE_GRAPHIQUE ;
             end ;
        when 2 =>
             GET_LINE(f,"Fonction a interpoler, f(x) = ") ;
ENTRER_ENTIER_BORNE(3,DIM_MAX,n,"Nombre de points du nuage (> 2) : ") ;
             n := n - 1 ;
             declare
               x, y : VECTEUR(0..n) ;
               C : COURBE(300) ;
               Pas, a, b : FLOAT ;
             begin
          PUT_LINE("L'intervalle de definition de f est note [a,b]") ;
               ENTRER_REEL(a,"a = ") ;
               ENTRER_REEL_BORNE(a,1.0E+10,b,"b = ") ;
               if LIRE_REPONSE(
               "Interpolation avec les abscisses de Tchebychev? ") then
                  for i in x'range loop
       x(i) := 0.5*((b-a)*Cos(FLOAT(2*i+1)/FLOAT(2*n+2)*Pi) + b + a) ;
                    y(i) := EVALUE(f,x(i)) ;
                  end loop ;
               else
                  Pas := (b - a)/FLOAT(n) ;
                  x(0) := a ; y(0) := EVALUE(f,a) ;
                  for i in 1..n loop
                    x(i) := x(i-1) + Pas ;
                    y(i) := EVALUE(f,x(i)) ;
                  end loop ;
               end if ;
               COURBE_INTERPOLATION_LAGRANGE(x, y, a, b, C) ;
               TRACE_UNE_COURBE(C) ;
               Pas := (b - a)/FLOAT(C.n) ;
               for i in 0..C.n loop
                  CROIX(a,EVALUE(f,a)) ;
                  a := a + Pas ;
               end loop ;
               SORTIE_GRAPHIQUE ;
             end ;
      end case ;
   end loop ;
end DEM_LAGR ;
```

6.12 Spécification du paquetage INTERPOLATION_SPLINE

```
with COMMON_MATRIX ;
use COMMON_MATRIX ;
package INTERPOLATION_SPLINE is
procedure CALCUL_COEFFICIENTS_SPLINE_NATUREL(x, y : in VECTEUR ;
             a,b,c,d : out VECTEUR) ; -- § 4.3.7
function SPLINE(x,y,a,b,c,d : in VECTEUR ; t : in FLOAT)
             return FLOAT ; -- § 4.3.2
```

```
procedure TRACE_SPLINE_NATUREL(x, y : in VECTEUR) ;
procedure CALCUL_COEFFICIENTS_SPLINE_PERIODIQUE(PERIODE : in FLOAT ;
            x, y : in VECTEUR ; a,b,c,d : out VECTEUR) ; -- § 4.3.10
procedure TRACE_SPLINE_PERIODIQUE(PERIODE : in FLOAT ;
            x, y : in VECTEUR) ;
procedure TRACE_SPLINE_PARAMETRIQUE(x, y : in VECTEUR) ; -- § 4.3.11
end SPLINE ;
```

6.13 Démonstration de l'interpolation spline

```
with TEXT_IO, MATH0, COMMON_MATRIX, DONNEES_POINTS,
     INTERPOLATION_SPLINE ;
use TEXT_IO, MATH0, COMMON_MATRIX, DONNEES_POINTS,
    INTERPOLATION_SPLINE ;
procedure DEM_SPL is
CHOIX : NATURAL range 0..2 ;
procedure DEMO_SPLINE_CARTESIENNE is
n : NATURAL ;
begin
 ENTRER_ENTIER_BORNE(4,DIM_MAX,n,"Nombre de points du nuage (> 3) : ");
    declare
       x, y : VECTEUR(1..n) ;
    begin
       ENTRER_POINTS(x,y,TRUE) ;
       TRACE_SPLINE_NATUREL(x,y) ;
    end ;
end DEMO_SPLINE_CARTESIENNE ;
procedure DEMO_SPLINE_PARAMETRIQUE is
n : NATURAL ;
begin
 ENTRER_ENTIER_BORNE(4,DIM_MAX,n,"Nombre de points du nuage (> 3) : ");
    declare
       x, y : VECTEUR(1..n) ;
    begin
       ENTRER_POINTS(x,y,FALSE) ;
       TRACE_SPLINE_PARAMETRIQUE(x,y) ;
    end ;
end DEMO_SPLINE_PARAMETRIQUE ;
procedure MENU_SPLINE(CHOIX : out NATURAL) is
begin
    PUT_LINE("-1- Interpolation spline d'une courbe cartesienne") ;
    PUT_LINE("-2- Interpolation spline d'une courbe parametrique") ;
    NEW_LINE ;
    ENTRER_ENTIER_BORNE(0,2,CHOIX,"Votre choix : ") ;
end MENU_SPLINE ;

begin
   loop
      MENU_SPLINE(CHOIX) ;
      case CHOIX is
        when 0 => exit ;
        when 1 => DEMO_SPLINE_CARTESIENNE ;
        when 2 => DEMO_SPLINE_PARAMETRIQUE ;
```

```
      end case ;
   end loop ;
end DEM_SPL ;
```

CHAPITRE 6

Calcul numérique des intégrales

1. Introduction. Position des problèmes

1.1 Remarques préliminaires sur le calcul des primitives

1.1.1 Calcul direct des intégrales

On sait que, si f : [a,b] → R est continue, alors elle admet des primitives et si F est l'une d'elles, l'intégrale de f sur [a,b] est :

$$\int_a^b f(x)dx = F(b) - F(a)$$

Puis, pour f : [a,b[→ R continue avec a < b ≤ +∞, en cas de convergence, l'intégrale impropre de f sur [a,b[est définie par :

$$\int_a^b f(x)dx = \lim_{x \to b} \int_a^x f(t)dt$$

Enfin, le théorème de Fubini va ramener le calcul des intégrales multiples à celui des intégrales simples.

Tout semble donc résolu. Mais en réalité, les cas où l'on sait calculer explicitement les primitives sont rares. De plus on a souvent affaire à des fonctions f qui ne sont connues explicitement qu'en un nombre fini de points. Il faudra donc développer des méthodes de calcul adaptées à ces cas.

1.1.2 A propos des primitives élémentaires

On dira qu'une fonction f admet des primitives élémentaires si elle admet des primitives qui peuvent s'exprimer à l'aide des fonctions polynomiales, des fonctions transcendantes usuelles (Log, trigonométriques et leurs inverses) et des quatre opérations usuelles sur R.

En ce sens, la fonction $f: x \mapsto f(x) = \frac{1}{\mathrm{Ln}(x)}$ n'admet pas de primitives élémentaires sur]1,+∞[. Cette dernière étant continue, on peut poser pour x > 1, $L_i(x) = \int_2^x \frac{dt}{\mathrm{Ln}(t)}$, ce qui définit une fonction appelée *« logarithme intégral »*.

De manière plus générale, Liouville a montré le résultat suivant :

Théorème **:** Si f et g sont deux fonctions rationnelles, alors $f \cdot e^g$ admet des primitives élémentaires si, et seulement si, il existe une fonction rationnelle h telle que $f = h' + h \cdot g'$.

Démonstration — B. Randé : Primitives élémentaires. Revue des Mathématiques spéciales, Mai 1984.

Application — En prenant $f(x) = \frac{1}{x}$ et $g(x) = x$, on déduit que $\frac{e^x}{x}$ n'admet pas de primitives élémentaires et en faisant des changements de variables, on déduit qu'il en est de même pour les fonctions $\frac{1}{\mathrm{Ln}(x)}$, $\frac{e^x}{x^2}$, $e^{\frac{1}{x}}$ et Ln(Ln(x)).

1.2 Elaboration de méthodes numériques

Il est donc nécessaire de développer des méthodes numériques de calcul approché des intégrales définies, impropres et multiples. On parle aussi de méthodes de *« quadrature numérique »*, pour les intégrales simples et de *« cubature numérique »* pour les intégrales doubles.

Une façon de résoudre le problème qui nous intéresse est de se ramener à une équation différentielle avec conditions initiales, soit à calculer y(b), où y est solution de :

$$\begin{cases} y'(x) = f(x) \quad (a \le x \le b) \\ y(a) = 0 \end{cases}$$

et utiliser les méthodes de résolution des équations différentielles (Cf. le chapitre sur la résolution numérique des équations différentielles).

On pourra aborder le problème de ce point de vue, par exemple, pour des fonctions ayant de nombreux pics, mais en général, vue la forme particulière de l'équation différentielle, on aura intérêt à utiliser une méthode spécifique.

Comme dans le cas des équations différentielles, à la base on a une idée de *« discrétisation »* de l'intervalle [a,b], à partir de laquelle on calculera des sommes qui vont approximer l'intégrale.

Dans ce chapitre, on s'intéressera d'abord au calcul numérique des intégrales définies en utilisant des interpolations polynomiales, puis on étudiera des méthodes utilisant les polynômes orthogonaux, qui permettent également de calculer certaines intégrales impropres et enfin on dira quelques mots sur le calcul numérique des intégrales doubles.

Comme toujours, à chaque fois on essayera de donner une évaluation des erreurs de méthode.

2. Méthodes de calcul par interpolation polynomiale. Schémas d'intégration classiques

2.1 Idée des méthodes par interpolation

On se donne, pour tout le paragraphe 2, une fonction f : [a,b] $\to$ R supposée continue et connue en un nombre fini de points $x_0 < ... < x_n$ dans [a,b].

L'idée est alors d'utiliser les valeurs connues $(f(x_i))_{0\le i\le n}$ pour calculer une valeur approchée de $I(f) = \int_a^b f(x)dx$.

Dans un premier temps, on cherchera des formules d'intégration du type :

$$(1) \quad I(f) \cong \hat{I}(f) = \sum_{i=0}^{n} a_i \cdot f(x_i)$$

où les constantes a_i ne dépendent que des x_i et pas de la fonction f.

On parle alors de « formule d'intégration élémentaire ».

Dans les méthodes par interpolation polynomiale, on demandera que la formule (1) soit exacte pour les polynômes de degré au plus n, soit en notant $R_n[x]$ l'espace des polynômes de degré au plus n :

$$(2) \quad \forall P \in R_n[x] \ , \ \int_a^b P(x)dx = \sum_{i=0}^{n} a_i \cdot P(x_i)$$

Cette idée est justifiée par le théorème de Stone-Weirstrass qui nous dit que toute fonction continue sur [a,b] est limite uniforme d'une suite de polynômes.

Donc si pour une précision $\varepsilon > 0$ donnée, P_n est un polynôme de degré n tel que $\mathrm{Sup}\{|f(x) - P_n(x)|;\ a \le x \le b\} < \varepsilon$ et si $(a_i)_{0\le i\le n}$ est une suite de coefficients vérifiant (2), on écrira :

$$\int_a^b f(x)dx \cong \int_a^b P_n(x)dx = \sum_{i=0}^{n} a_i \cdot P_n(x_i) \cong \sum_{i=0}^{n} a_i \cdot f(x_i)$$

soit la formule (1).

En fait, numériquement il n'est pas pratique de raisonner comme indiqué ci-dessus.

L'idée que l'on retiendra est de remplacer f par son polynôme d'interpolation en $(x_i)_{0\le i\le n}$, ce qui théoriquement est moins satisfaisant, car quand n tend vers $+\infty$ une telle suite ne converge pas vers f et donc la convergence de la méthode n'est pas assurée dans tous les cas. Mais d'un point de vue pratique cela est en général satisfaisant.

Il sera nécessaire d'évaluer l'erreur de méthode :

$$(3) \quad R(f) = |I - \hat{I}|$$

2.2 Résolution du problème (2)

2.2.1 Existence et unicité d'une solution

L'existence et l'unicité des coefficients a_i est assurée par le :

Théorème **:** Il existe des constantes uniques $(a_i)_{0\le i\le n}$ telles que :

$$(2) \qquad \forall P \in R_n[x],\ \int_a^b P(x)dx = \sum_{i=0}^{n} a_i \cdot P(x_i)$$

Démonstration — En utilisant la base canonique $\{1, x, ..., x^n\}$ de $R_n[x]$, on voit que (2) équivaut à trouver une solution au système linéaire :

$$(4) \quad \sum_{j=0}^{n} x_j^i \cdot a_j = \frac{b^{i+1} - a^{i+1}}{b-a} \quad (i = 0, ..., n)$$

La matrice de ce système étant du type Vandermonde de déterminant $\Delta_i = \prod_{0\le i<j\le n}(x_j - x_i) \neq 0$, on déduit que (4) et donc (2) admet une unique solution.

2.2.2 Détermination pratique des coefficients a_i

Exemple — Prenons a = 0, b = 2 et $x_0 = 0$, $x_1 = 1$, $x_2 = 2$.

Le système à résoudre est alors :

$$\begin{cases} a_0 + a_1 + a_2 = 2 \\ a_1 + 2\cdot a_2 = 2 \\ a_1 + 4\cdot a_2 = \dfrac{8}{3} \end{cases}$$

de solution $a_0 = \frac{1}{3},\ a_1 = \frac{4}{3},\ a_2 = \frac{1}{3}$.

On a donc pour tout polynôme $P = a\cdot x^2 + b\cdot x + c$:

$$\int_0^2 P(x)dx = \frac{1}{3}\cdot\left(P(0) + 4\cdot P(1) + P(2)\right)$$

et la formule d'intégration associée sera alors :

$$\int_0^2 f(x)dx \cong \frac{1}{3}\cdot\left(f(0) + 4\cdot f(1) + f(2)\right)$$

Il s'agit en fait de la règle de Simpson que nous décrirons plus loin.

Pour f(x) = Ch(x), on a I = Sh(2) ≅ 3,6269 et $\hat{I} \cong 3.6449$.

De manière générale, le système (4) étant mal conditionné, sa résolution posera des problèmes pour n grand.

La matrice de (4) étant symétrique définie positive, on peut penser à utiliser la méthode de Cholesky, mais pour n grand les résultats ne sont pas satisfaisants.

En fait, dans la pratique, on ne considère pas ces formules pour de grandes valeurs de n comme on le verra plus loin.

D'autre part, on peut calculer les a_i d'une autre façon.

Soit $\{L_0, ..., L_n\}$ la base de Lagrange de $R_n[x]$ définie par :

$$L_i(x) = \frac{\prod_{\substack{j=0 \\ j\neq i}}^{n} (x - x_j)}{\prod_{\substack{j=0 \\ j\neq i}}^{n} (x_i - x_j)} \quad (i = 0, ..., n)$$

on a alors :

$$\forall\, P \in R_n[x]\ ,\ P(x) = \sum_{i=0}^{n} P(x_i) \cdot L_i(x)$$

et :

$$\int_a^b P(x)dx = \sum_{i=0}^{n} \left(\int_a^b L_i(x)dx \right) \cdot P(x_i)$$

et de l'unicité des coefficients a_i, on déduit que :

$$a_i = \int_a^b L_i(x)dx,\ i = 0, ..., n$$

ce qui détermine de façon « explicite » les a_i.

2.2.3 Majoration de l'erreur

Pour f : [a,b] $\rightarrow$ R, continue, on pose :

$$E(f) = \int_a^b f(x)dx - \sum_{i=0}^{n} a_i \cdot f(x_i)$$

où les a_i sont définis par E(P) = 0, pour tout P dans $R_n[x]$.

Pour t fixé dans R, on note :

$$\varphi_t : \mapsto \varphi_t(x) = (x-t)_+^n = \begin{cases} (x-t)^n & \text{si } x \geq t \\ O & \text{si } x < t \end{cases}$$

et :

$$K_n(t) = E(\varphi_t)$$

La fonction K_n ainsi définie s'appelle le *« noyau de Péano »* de la méthode.

On peut alors montrer le :

Théorème (Péano) : Si f est de classe C^{n+1}, alors :

$$E(f) = \frac{1}{n!} \cdot \int_a^b K_n(t) \cdot f^{(n+1)}(t)dt$$

et donc :

$$R(f) = |E(f)| \leq C_n \cdot M_{n+1} \cdot \frac{1}{n!}$$

où $C_n = \int_a^b |K_n(t)|dt$ et $M_{n+1} = \text{Sup}\left\{ \left| f^{(n+1)}(x) \right| \ ;\ a \leq x \leq b \right\}$.

Si de plus, K_n garde un signe constant sur [a,b], alors il existe ξ dans [a,b] tel que :

$$E(f) = \frac{E(x^{n+1})}{(n+1)!} \cdot f^{(n+1)}(\xi)$$

où :

$$E(x^{n+1}) = \frac{b^{n+2} - a^{n+2}}{b-a} - \sum_{i=0}^{n} a_i \cdot x_i^{n+1}$$

Démonstration — Voir Crouzeix et Mignot p.39.

2.3 Cas particulier où les x_i sont équidistants. Méthodes de Newton et Cotes

2.3.1 Calcul des coefficients a_i

On suppose ici que les x_i sont équidistants dans [a,b], soit : en posant $h = \frac{b-a}{n}$:

$$x_i = a + i \cdot h \quad (i = 0, 1, ..., n)$$

La méthode obtenue est appelée *« méthode fermée de Newton-Cotes à n + 1 points »*. On parle de méthode fermée car les extrémités a et b font partie des points utilisés. On peut aussi considérer des méthodes qui n'utilisent que les points x_1, ..., x_{n-1} et on parle de « méthode ouverte de Newton-Cotes à n – 1 points », mais ces méthodes sont peu utilisées.

En prenant pour base de $R_n[x]$, $\{1, (x - a), ..., (x - a)^n\}$, on déduit que les coefficients a_i sont solutions du système linéaire :

$$(5) \quad \begin{cases} a_0 + a_1 + \ldots\ldots\ldots\ldots\ldots + a_n = n \cdot h \\ a_1 + 2 \cdot a_2 + \ldots\ldots\ldots + n \cdot a_n = \frac{n^2}{2} \cdot h \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ a_1 + 2^n \cdot a_2 + \ldots + n^n \cdot a_n = \frac{n^{n+1}}{n+1} \cdot h \end{cases}$$

En écrivant les coefficients a_i sous la forme :

$$a_i = \int_a^b L_i(x)dx, \; i = 0, ..., n$$

avec les notations de (2.2.2), puis en faisant le changement de variable $x = a + t \cdot h$, on déduit que :

$$a_i = \frac{b-a}{n} \cdot b_i, \text{ avec } b_i = \int_0^n \prod_{\substack{j=0 \\ j \neq i}}^{n} \frac{t-j}{i-j} dt \quad (i = 0, ..., n)$$

la suite $(b_i)_{0 \leq i \leq n}$ ne dépendant que de n et pas de l'intervalle [a,b], on peut en calculer ces valeurs une fois pour toutes. Les b_i sont aussi solutions de (5) avec $h = 1$.

En utilisant un programme de résolution des systèmes linéaires à coefficients entiers (Cf. (3.4.6.2) du chapitre sur l'analyse numérique linéaire), on obtient les résultats du tableau 6.1 :

n	b_0	b_1	b_2	b_3	b_4	b_5
1	1/2	1/2				
2	1/3	4/3	1/3			
3	3/8	9/8	9/8	3/8		
4	14/45	64/45	8/15	64/45	14/45	
5	95/288	125/96	125/144	125/144	125/96	95/28

Tableau 6.1

2.3.2 Majoration de l'erreur dans les méthodes de Newton_Cotes

On peut montrer les résultats suivants, qui dépendent de la parité de n :

Théorème : (1) Si n est pair et f de classe C^{n+2}, alors il existe x dans [a,b] tel que :

$$E(f) = K_n \cdot f^{(n+2)}(\xi) \cdot \frac{h^{n+3}}{(n+2)!}, \text{ où } K_n = \int_0^n t^2 \cdot (t-1)\cdot\ldots\cdot(t-n)dt$$

(2) Si n est impair et f de classe C^{n+2}, alors il existe x dans [a,b] tel que :

$$E(f) = K_n \cdot f^{(n+1)}(\xi) \cdot \frac{h^{n+2}}{(n+1)!}, \text{ où } K_n = K_n = \int_0^n t \cdot (t-1)\cdot\ldots\cdot(t-n)dt$$

Démonstration — Cf Stoer et Burlisch p. 123.

En fait, comme la suite des polynômes d'interpolation de f ne tend pas vers f quand n tend vers +∞, il y a peu de chances que les méthodes de Newton-Cotes convergent quand n tend vers +∞. Le bon point de vue sera, non pas de travailler sur tout l'intervalle [a,b], mais de découper cet intervalle en un grand nombre de petits intervalles et d'utiliser une méthode de Newton-Cotes sur chacun de ces intervalles, ce qui donne des résultats très satisfaisants en général.

Nous allons maintenant regarder ce qui se passe pour quelques valeurs classiques de n.

2.4 Méthodes classiques d'intégration

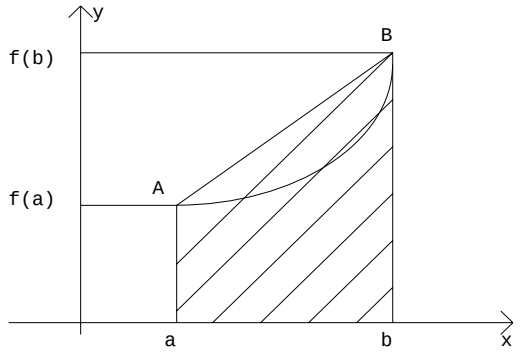
2.4.1 Cas n = 1. Méthode des trapèzes

Dans ce cas, on a h = b – a, $x_0 = a$ et $x_1 = b$, ce qui donne :

$$a_0 = a_1 = \frac{b-a}{2}$$

et la formule d'intégration approchée :

$$\int_a^b f(x)dx \cong \frac{b-a}{2} \cdot (f(a) + f(b))$$



Cette valeur approchée représente l'aire du trapèze hachuré.
Ce qui consiste aussi à remplacer l'arc AB par sa corde.

Figure 6.1

Méthode composite des trapèzes — L'erreur de méthode est alors donnée par :

$$R(f) \le \frac{M_2}{12} \cdot (b-a)^3$$

$$\text{où } M_2 = \text{Sup}\left\{ \left| f''(x) \right| \ ; \ a \le x \le b \right\}.$$

Pour un intervalle [a,b] trop grand, la précision est donc médiocre.

La première idée est donc de découper [a,b] en un grand nombre de sous-intervalles et d'appliquer la méthode des trapèzes sur chacun de ces sous-intervalles.

On pose donc, pour n > 0 dans N :

$$h = \frac{b-a}{n},\ x_i = a + i \cdot h \ (i = 0, ..., n)$$

ces notations étant totalement indépendantes des précédentes.

Et en écrivant :

$$I(f) = \int_a^b f(x)dx = \sum_{i=0}^{n-1} \int_{x_i}^{x_{i+1}} f(x)dx$$

on aboutit à la formule d'intégration approchée :

$$I(f) \cong T(h) = h \cdot \left(\frac{f(a)+f(b)}{2} + \sum_{i=0}^{n-1} f(x_i) \right)$$

Une majoration de l'erreur étant :

$$R(f) \le \frac{M_2 \cdot (b-a)}{12} \cdot h^2$$

On a donc une méthode d'ordre 2.

Remarque — A l'aide de la « formule d'Euler-Maclaurin », on peut préciser le reste dans la méthode des trapèzes. Cette formule s'écrit, pour une fonction f de classe C^{2n+2} :

$$\int_a^b f(x)dx =$$

$$T(h) - \sum_{k=1}^{n} \frac{B_k}{(2 \cdot k)!} \left(f^{(2k-1)}(b) - f^{(2k-1)}(a) \right) \cdot h^{2k} - \frac{B_{2n+2}}{(2 \cdot n+1)!} \cdot (b-a) \cdot h^{2n+2} \cdot f^{(2n+2)}(\xi)$$

où ξ est dans [a,b] et les B_{2k} sont les nombres de Bernoulli définis par le développement en série entière :

$$\frac{t}{e^t - 1} = \sum_{k=0}^{+\infty} B_n \cdot \frac{t^n}{n!}$$

soit, pour les premières valeurs :

$B_0 = 1, B_1 = -\frac{1}{2}, B_2 = \frac{1}{6}, B_4 = -\frac{1}{30}, B_6 = \frac{1}{42}, B_8 = -\frac{1}{30}, B_{10} = \frac{5}{66}, B_{12} = -\frac{691}{2730}, \ldots$

et pour $k \geq 1$, $B_{2k+1} = 0$.

Méthode des trapèzes dichotomique — On peut remarquer, dans la méthode composite des trapèzes, que si l'on double le nombre de points, soit avec $n' = 2 \cdot n$, $h' = \frac{h}{2}$, alors dans le calcul de T(h'), on peut exploiter les calculs utilisés pour T(h), soit :

$$T(h') = \frac{h}{2} \cdot \left(\frac{f(a) + f(b)}{2} + \sum_{i=0}^{n'-1} f(x'_i) \right)$$

et dans $\sum_{i=1}^{n'-1} f(x'_i)$, tous les x'_i avec i pair sont les termes précédents, il suffit donc de calculer les n termes $f(x'_{2k-1})$, pour k = 1, ..., n. Et à ce stade l'erreur est un $O\left(\left(\frac{h}{2}\right)^2\right)$.

D'où l'algorithme de calcul :

Etape 1 — $n = 1$, $h = (b - a)$ et $T_1 = \frac{b-a}{2} \cdot (f(a) + f(b))$

Etape 2 — $n = 2$, $h = \frac{b-a}{2}$ et $T_2 = \frac{1}{2} \cdot (T_1 + (b - a) \cdot f(x_1))$

Etape 3 — $n = 4$, $h = \frac{b-a}{4}$ et $T_4 = \frac{1}{2} \cdot (T_2 + \frac{b-a}{2} \cdot (f(x_1) + f(x_3))$

Dans les programmations structurées qui suivent, la procédure Trapèze_Dichotomique décrit une itération de la méthode, puis la procédure Integration_Trapèze décrit le calcul de l'intégrale, un test d'arrêt étant :

$$\text{Iter} = \text{MaxIteration ou } \left|T_{\text{Iter}} - T_{\text{Iter}-1}\right| < \text{Eps}$$

où le nombre maximum d'itérations MaxIter et la précision Eps sont donnés en constantes. MaxIter ne devra pas être trop grand car $n = 2^p$ avec $p \leq$ MaxIter. MaxIter = 10 est plus que suffisant.

```
PROCEDURE Trapèze_Dichotomique(Entrée f : Fonction, a, b : Réel ; Iter : Entier ;
                               Entrée_Sortie n : Entier ; T : Réel) ;
Début
   Si Iter = 1
   Alors Début
         T = (b - a)/2 · (f(a) + f(b)) ;
```

```
        n = 1 ;
  Fin
  Sinon Début
        h = (b − a) / n ; x = a + h / 2 ;
        Somme = 0 ;
        Pour j allant de 1 à n Faire
        Début
            Somme = Somme + f(x) ;
            x = x + h ;
        Fin ;
        T = 1/2 ·(T + h·Somme) ;
        n = 2·n ;
      Fin ;
Fin ;

PROCEDURE Integration_Trapèze(Entrée f : Fonction ; a, b : Réel ;
                              Entrée_Sortie T : Réel) ;
Début
   Iter = 0 ;
   T1 = –10^38 ;
   Répéter
      Iter = Iter + 1 ;
      Trapèze_Dichotomique(f,a,b,Iter,n,T) ;
      Correct = (Iter = MaxIter) ou ( |T_Iter − T_Iter−1| < Eps) ;
      T1 = T ;
   Jusqu'à Correct ;
Fin ;
```

2.4.2 Cas n = 2. Méthode de Simpson

Dans ce cas, on a $h = \frac{b-a}{2}$, $x_0 = a$, $x_1 = \frac{a+b}{2}$ et $x_2 = b$, ce qui donne :

$$a_0 = \frac{1}{3}\cdot h,\ a_1 = \frac{4}{3}\cdot h \text{ et } a_2 = \frac{1}{3}\cdot h$$

et la formule d'intégration approchée :

$$\int_a^b f(x)dx \cong \frac{b-a}{6}\cdot\left(f(a) + 4\cdot f\left(\frac{a+b}{2}\right) + f(b)\right)$$

L'erreur de méthode est alors donnée par :

$$R(f) \le \frac{M_4}{90}\cdot h^5 = \frac{M_4}{2880}\cdot(b-a)^5$$

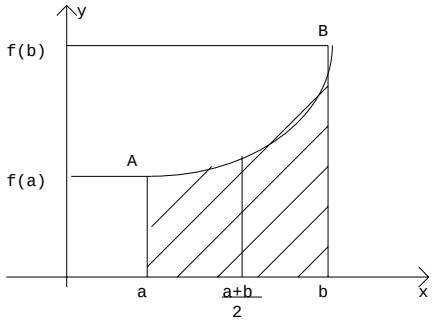


Figure 6.2

Cette valeur approchée représente l'aire du trapèze parabolique hachuré.
Ce qui consiste aussi à remplacer l'arc AB par l'arc de parabole passant par a, $\frac{a+b}{2}$ *et b.*

Méthode de Simpson composite — Avec les notations utilisées pour la méthode composite des trapèzes, en découpant [a,b] en n sous-intervalles de même longueur et en utilisant la formule de Simpson sur chacun de ces sous-intervalles, on aboutit à la formule approchée :

$$\int_a^b f(x)dx \cong S(h) = \frac{h}{6}\cdot\left(f(a) + f(b) + 2\cdot\sum f(x_i) + 2\cdot f(m_i) \right)$$

où $m_i = \frac{x_i + x_{i+1}}{2}$] est le point milieu et $h = \frac{b-a}{n}$.

Une majoration de l'erreur étant :

$$R(f)\ \frac{M_4\cdot(b-a)}{2880}\cdot h^4$$

Pour la programmation, on peut exploiter les considérations précédentes en remarquant que :

$$S(h) = \frac{4}{3}\cdot T\left(\frac{h}{2}\right) - \frac{1}{3}\cdot T(h)$$

Comme pour la méthode des trapèzes, on peut utiliser une méthode dichotomique, ce qui donne la procédure :

```
PROCEDURE Integration_Simpson(Entrée f : Fonction ; a, b : Réel ;
                              Entrée_Sortie S : Réel) ;
Début
   Iter = 1 ;
   Trapèze_Dichotomique(f,a,b,Iter,n,T) ;
   T1 := T ;
   S1 = –10^38 ; { Valeur précédente de S(h } 
   Répéter
      Iter = Iter + 1 ;
      Trapèze_Dichotomique(f,a,b,Iter,n,T) ; { T = T(h/2) }
      S = (4·T – T1)/3 ; { S = S(h) }
      Correct = (Iter = MaxIter) ou (|S – S1| < Eps) ;
      T1 = T ;
      S1 = S ;
   Jusqu'à Correct ;
Fin ;
```

2.4.3 Méthode de Romberg

En utilisant un procédé, due à *Richardson* en 1927, d'accélération de la convergence de certaines suites, on peut améliorer la méthode dichotomique des trapèzes. La méthode obtenue est due à Romberg (1955) et le principe consiste à construire la suite double $(T(k,m))_{(k,m)\in N^2}$ définie par la récurrence :

$$(1)\quad \begin{cases} T(0,m) = T\left(\dfrac{b-a}{2^m}\right),\ m \geq 0 \\ T(k,m) = \dfrac{1}{4^k-1}\cdot\left(4^4\cdot T(k-1,m+1) - T(k-1,m)\right) \end{cases}$$

où $T\left(\dfrac{b-a}{2^m}\right)$ est l'approximation de l'intégrale obtenue par la méthode des trapèzes avec le pas $\dfrac{b-a}{2^m}$.

Les propriétés de la suite (T(k,m)) sont résumées dans le :

Théorème **:** (1) Il existe une suite double $(a(k,m))_{(k,m)\in N^2}$ telle que :

$$T(k,m) = \sum_{p=0}^{k} a(k,p)\cdot T(0,p+m)$$

où :

$$a(k,m) = (-1)^{k+m}\cdot\frac{b(k,m)}{d_k}$$

avec :

$$d_k = 1\cdot 3\cdot 15\cdot\ldots\cdot(4k-1) \cong Cste\cdot 2^{k(k+1)}$$

et les b(k,m) définis par :

$$\begin{cases} b(k,0) = 1 \\ b(k,m) = 4^k\cdot b(k-1,m-1) + b(k-1,m) \end{cases}$$

en particulier :

$$\begin{cases} b(k,m) > 0, \text{ pour } m = 0, 1, \ldots, k \\ b(k,m) = 0, \text{ pour } m > k \end{cases}$$

$$(2)\qquad \lim_{m\to+\infty} T(0,m) = I(f) = \int_a^b f(x)dx$$

et avec (1), on déduit que :

$$\forall k \in N,\ \lim_{m\to+\infty} T(k,m) = I(f)$$

$$(3)\quad T(k,m) - I(f) = O(4^{-(k+1)m})$$

$$(4)\quad \forall m \in N,\ \lim_{k\to+\infty} T(k,m) = I(f)$$

Démonstration — Cf Warusfel, R. M. S. Janvier 1986.

Remarque 1 — Avec (3), on déduit que la convergence de T(k,m) vers I(f), à k fixé, est d'autant plus rapide que k est grand.

Pour k = 0, on retrouve la méthode des trapèzes dichotomiques, pour k = 1, c'est la méthode de Simpson, pour k = 2, c'est la méthode de Boole-Villarceau et pour k ≥ 3 il ne s'agit plus d'une méthode de Newton-Cotes.

Remarque 2 — La suite qui est généralement utilisée pour approcher I(f) est celle définie par : $R_k = T(k,0)$.

La convergence de (R_k) vers I(f) est, en général très rapide, mais on ne sait rien sur la vitesse de convergence (dixit Warusfel).

La relation (3) ne permet pas de montrer cette convergence, la démonstration se faisant par une étude directe (Cf. Warusfel).

L'algorithme de calcul des R_k est alors le suivant :

k = 0 — Calcul de $T(0,0) = R_0$ par Trapèze-Dichotomique ;
k = 1 — Calcul de T(0,1) par Trapèze-Dichotomique,
Calcul de $T(1,0) = R_1$ en fonction de T(0,1) et T(0,0) ;
k = 2 — Calcul de T(0,2) par Trapèze-Dichotomique,
Calcul de T(1,1) en fonction de T(0,2) et T(0,1),
Calcul de $T(2,0) = R_2$ en fonction de T(1,1) et T(1,0) ;
k quelconque — Calcul de T(0,k) par Trapèze-Dichotomique,
Calcul de T(1,k - 1) en fonction de T(0,k) et T(0,k-1),
.....
Calcul de T(k-j,j) en fonction de T(k-j-1,j+1) et T(k-j-1,j),
.....
Calcul de $T(k,0) = R_k$ en fonction de T(k - 1,1) et T(k - 1,0).

En fait, on peut remarquer qu'il est inutile de travailler avec un tableau à deux dimensions, ce qui revient à conserver en mémoire des valeurs qui ne sont pas utilisées par la suite.

Il est plus astucieux d'utiliser un seul tableau :

$$T = (T(0),...,T(kMax))$$

et à l'étape k du calcul de stocker T(k – j,j) dans T(j), pour j allant de k à 0. Ce qui donne, pour l'étape k :

Calcul de T(k) par Trapèze-Dichotomique ;

Pour j allant de k – 1 à 0 T(j) est fonction de T(j + 1) = T(k – (j + 1),j+1), qui vient d'être calculé, et de T(j) = T((k – 1) – j,j), qui a été calculé à l'étape k – 1.

Ce qui donne en définitive, la procédure ci-dessous, où kMax est une constante (par ex. kMax = 15) ; T est un tableau de réels numérotés de 0 à kMax ; pour k = 0, T(0) contient T(k,0), c'est-à-dire la valeur R de l'intégrale obtenue par la méthode de Romberg, et T(k) = T(0,k) est la valeur calculée par la méthode des trapèzes.

Comme test d'arrêt, on prendra : $\left|T(k,0) - T(k-1,0)\right| < Eps$, où Eps est une précision donnée en constante.

```
PROCEDURE Intégration_Romberg(Entrée f : Fonction ; a, b : Réel ;
                              Entrée_Sortie R : Réel) ;
Début
```

Trapèze_Dichotomique(f,a,b,1,n,T1) ;

```
    T(0) = T1 ; { T1 est réinjecté à l'étape suivante d'où la nécessité de cette variable. }
    R1 = –MaxRéel ;
    k = 0 ;
    Répéter
        k = k + 1 ;
        Trapèze_Dichotomique(f,a,b,k,n,T1) ;
        T(k) = T1 ;
        p = 1 ;
        Pour j allant de k – 1 à 0 Faire
        Début
            p = 4p ;
            T(j) = (p·T(j + 1) – T(j))/(p - 1) ;
        Fin ;
        R = T(0) ;
        Correct = (k = kMax) ou ( |R – R1| < Eps) ;
        R1 = R ;
    Jusqu'à Correct ;
Fin ;
```

3. Utilisation des polynômes orthogonaux. Quadratures de Gauss

3.1 Introduction. Idées des méthodes

Pour calculer, de façon approchée, $I(f) = \int_a^b f(x)dx$, l'idée des méthodes de Newton-Cotes est, étant données des abscisses x_1, ..., x_n dans [a,b], de trouver des coefficients a_1, ..., a_n tels que

$$(1) \quad \int_a^b P(x)dx = \sum_{i=1}^{n} a_i \cdot P(x_i), \text{ pour tout P dans } R_{n-1}[x]$$

et de prendre comme formule de quadrature :

$$(2) \quad I(f) = \int_a^b f(x)dx \cong \hat{I}(f) = \sum_{i=1}^{n} a_i \cdot f(x_i)$$

La question que l'on peut se poser est alors : peut-on choisir les x_i de façon optimale ?

Pour répondre à la question ci-dessus, l'idée des méthodes de Gauss est de demander que (1) soit vraie pour des polynômes de degré maximum. Pour ce faire, on prendra aussi comme inconnues les abscisses x_i, ce qui va doubler le nombre de degrés de liberté et, de ce fait va aussi augmenter le temps de calcul.

Une fois les x_i et a_i déterminés, pour le même nombre d'évaluations que dans une méthode de Newton-Cotes correspondante au même nombre de points, on aura une formule d'intégration exacte pour des polynômes de degré élevé, ce qui peut faire espérer à une meilleure approximation de I(f), du moins pour les fonctions « lisses », c'est-à-dire qui peuvent être bien approchées par des polynômes. Mais pour d'autres cas il se pourra que l'approximation ne soit pas meilleure.

Un autre intérêt des méthodes de Gauss est qu'elles permettent aussi de calculer certaines intégrales impropres du type

$$I(f) = \int_a^b f(x) \cdot \pi(x) dx$$

où π :]a,b[$\rightarrow R^+$ est une fonction continue positive appelée *« fonction poids »* et $-\infty \leq a < b \leq +\infty$. La fonction poids contiendra, en général, les singularités de l'intégrale à calculer.

On pourra, par exemple, considérer des intégrales impropres du type :

$$\int_{-1}^{+1} \frac{f(x)}{\sqrt{1-x^2}} dx, \ \int_0^{+\infty} f(x) \cdot e^{-x} dx \text{ ou } \int_{-\infty}^{+\infty} f(x) \cdot e^{-x^2} dx$$

Dans la mesure où les abscisses x_i seront imposées ces méthodes s'appliqueront aux fonctions définies analytiquement, mais pas aux fonctions connues en quelques points si ces derniers ne sont pas les points privilégiés trouvés.

3.2 Notations. Position du problème

3.2.1 Notations

On note R[x] l'espace vectoriel des polynômes à coefficients réels et pour $n \geq 0$, $R_n[x]$ est le sous-espace vectoriel formé des polynômes de degré au plus n.

On se donne une fonction poids continue :

$$\pi :]a,b[\rightarrow R^+$$

où $-\infty \leq a < b \leq +\infty$ et telle que :

$$\forall k \in N, \int_a^b \left| \pi(x) \cdot x^k \right| dx < +\infty$$

Enfin, on pose :

$$E = \{ f :]a,b[\rightarrow R \text{ ; continue ; } \int_a^b \left| f(x) \right|^2 \cdot \pi(x) dx < +\infty \}$$

alors, E est un espace vectoriel sur R, contenant R[x] et qui peut être muni d'un produit scalaire en posant :

$$\forall f, g \in E, \langle f | g \rangle = \int_a^b f(x) \cdot g(x) \cdot \pi(x) dx$$

3.2.2 Position du problème

Pour $n \geq 1$ donné, on cherche des coefficients x_1, ..., x_n dans]a,b[, a_1, ..., a_n dans R et m dans N aussi grand que possible tels que :

$$\forall P \in R_m, \int_a^b P(x) \cdot \pi(x) dx = \sum_{i=1}^{n} a_i \cdot P(x_i) \qquad (3)$$

Ce qui équivaut encore au système de (m + 1) équations à $2 \cdot n$ inconnues :

$$\sum_{i=1}^{n} a_i \cdot x_i^k = \int_a^b \pi(x) \cdot x^k dx \qquad (k = 0, 1, ..., m) \quad (4)$$

Si on veut avoir une chance de résoudre ce système, il est raisonnable de prendre $m = 2 \cdot n - 1$.

Ensuite, pour f dans E, on utilisera la formule de quadrature :

$$I(f) = \int_a^b f(x) \cdot \pi(x) dx \cong \hat{I}(f) = \sum_{i=1}^{n} a_i \cdot f(x_i) \qquad (5)$$

Enfin, comme toujours, il faudra évaluer l'erreur :

$$E(f) = \left| I(f) - \hat{I}(f) \right| \quad (6)$$

3.3 Calcul des abscisses x_i et des coefficients de Gauss a_i

3.3.1 Remarque

Calculer les x_i équivaut à trouver un polynôme P_n de degré n qui s'annule en tous les x_i. Et il s'agit alors de traduire (4) sur P_n.

On pose donc :

$$P_n(x) = \alpha_n \cdot \prod_{i=1}^{n} (x - x_i) \quad (7)$$

où $\alpha_n \in R$, $x_1 < ... < x_n$ dans]a,b[sont les inconnues.

3.3.2 Condition nécessaire et suffisante sur P

Lemme : Les abscisses x_i vérifient (4) si, et seulement si :

$$\forall Q \in R_{n-1}, \int_a^b P_n(x) \cdot Q(x) \cdot \pi(x) dx = 0 \quad (8)$$

Ce qui peut se traduire en disant que P_n est orthogonal à $R_{n-1}[x]$.

Démonstration — C. N. Si les x_i vérifient (4), comme $P_n \cdot Q$ est dans $R_{2n-1}[x]$, pour tout Q dans $R_{n-1}[x]$, on a nécessairement :

$$\int_a^b P_n(x) \cdot Q(x) \cdot \pi(x) dx = \sum_{i=1}^{n} a_i \cdot P_n(x_i) \cdot Q(x_i) = 0$$

C. S. Supposons que P_n vérifie (8). Soit P dans $R_{2n-1}[x]$; par division euclidienne, on peut écrire :

$$P = Q \cdot P_n + R_n \quad (d°(Q) \le n - 1, d°(R_n) \le n - 1)$$

et avec (8) :

$$\int_a^b P(x) \cdot \pi(x) dx = \int_a^b \pi(x) \cdot R_n(x) dx$$

Il suffit donc de vérifier (4) pour k = 0, 1, ..., n – 1, car $R_n(x_i) = P(x_i)$ pour tout i = 1, ..., n.

Ce qui donne le système linéaire, aux inconnues a_1, ..., a_n :

$$\sum_{i=1}^{n} x_i^k \cdot a_i = \int_a^b \pi(x) \cdot x^k dx \qquad (k = 0, 1, ..., n - 1)$$

Le déterminant de ce dernier étant du type Vandermonde, il est non nul et il existe une unique solution a_1, ..., a_n.

3.3.3 Détermination explicite de P_n

Il s'agit de trouver un polynôme P_n orthogonal à $R_{n-1}[x]$ et n'ayant que des racines simples. Puis la recherche des racines de P_n nous donnera les x_i et enfin la résolution d'un système de Vandermonde nous donnera les a_i.

En fait, on verra que pour les méthodes de Gauss classiques, les a_i seront donnés explicitement et il est inutile de résoudre le système linéaire.

Lemme : Si P_n est orthogonal à $R_{n-1}[x]$, alors il n'a que des racines simples.

Démonstration — On a :

$$\forall Q \in R_{n-1}[x], \int_a^b P_n(x) \cdot Q(x) \cdot \pi(x) dx = 0$$

Pour Q = 1, on a donc $\int_a^b P_n(x) \cdot \pi(x) dx = 0$, avec π continue strictement positive et P_n continue, donc P_n ne peut garder un signe constant c'est-à-dire que P_n admet au moins une racine d'ordre impair.

En notant $x_1 < ... < x_k$ toutes les racines d'ordre impair de P_n, avec $k \geq 1$, on a :

$$P_n(x) = (x - x_1) \cdot ... \cdot (x - x_k) \cdot Q(x)$$

où Q est dans $R_{n-1}[x]$ et garde un signe constant sur]a,b[.

Si $k \leq n - 1$, alors $R(x) = (x - x_1) \cdot ... \cdot (x - x_k)$ est dans $R_{n-1}[x]$ et $\int_a^b P_n(x) \cdot R(x) \cdot \pi(x) dx = 0$ avec $R(x) \cdot P_n(x)$ de signe constant, ce qui est impossible, donc k = n.

On a donc : $P_n(x) = \alpha_n \cdot \prod_{i=1}^{n} (x - x_i)$.

Remarque — Localisation des zéros de P_n.

Pour localiser les racines de P_n, on dispose du :

Théorème : En notant $x_1, ..., x_n$ les zéros de P_n et $y_1, ..., y_{n-1}$ ceux de P_{n-1}, on a :

$$0 < x_1 < y_1 < x_2 < y_2 < ... < x_{n-1} < y_{n-1} < x_n$$

Ce qui permet de les localiser de proche en proche.

Conclusion — Pour construire P_n, il suffit de se donner une base orthogonale de $R_n[x]$, $\{P_0, P_1, ..., P_n\}$.

Le procédé d'orthogonalisation de Gram-Schmidt nous permet toujours de déterminer une telle base à partir de la base canonique de $R_n[x]$.

En pratique, on verra que dans les cas classiques, une telle base est connue. Il suffira alors de trouver les n racines de P_n, en utilisant la méthode de Newton (par exemple), puis de trouver les coefficients a_n.

3.4 Propriétés des polynômes orthogonaux. Calcul des coefficients de Gauss

Dans ce paragraphe, on se donne une famille orthonormée de polynôme $(P_n)_{n \geq 0}$, relativement au produit scalaire défini par une fonction poids π sur E.

On a donc :

$$\langle P_k | P_j \rangle = \int_a^b P_k(x) \cdot P_j(x) dx = \begin{cases} 1 \text{ si } k = j \\ 0 \text{ si } k \neq j \end{cases} \quad (0 \leq j, k \leq n)$$

et $d^\circ(P_k) = k$.

En particulier, on en déduit que :

$$\langle P_k | x^j \rangle = 0, \text{ pour } j = 0, 1, ..., k - 1$$

et si $P_k(x) = \alpha_k \cdot x^k + \beta_k \cdot x^{k-1} + ... + \gamma_0$ alors : $\left\langle P_k \middle| x^k \right\rangle = \dfrac{1}{\alpha_k}$

3.4.1 Récurrence vérifiée par les polynômes orthogonaux

Un algorithme de calcul des polynômes orthogonaux est donné par le :

Théorème : Si (P_n) est une famille orthonormée de polynômes dans E, on a alors la relation de récurrence :

$$x \cdot P_n(x) = a_n \cdot P_{n+1}(x) + b_n \cdot P_n(x) + c_n \cdot P_{n-1}(x) \quad (n \geq 1)$$

avec :

$$a_n = \frac{\alpha_n}{\alpha_{n+1}},\ b_n = \frac{\beta_n}{\alpha_n} - \frac{\beta_{n+1}}{\alpha_{n+1}} \text{ et } c_n = \frac{\alpha_{n-1}}{\alpha_n} = a_{n-1}$$

où $P_n(x) = \alpha_n \cdot x^n + \beta_n \cdot x^{n-1} + ... + \gamma_0$.

Démonstration — Cf Ayant et Borg p. 117.

3.4.2 Formule de Darboux-Christoffel

Théorème **:** Si (P_n) est une famille orthonormée de polynômes , avec les notations ci-dessus, pour x, y dans R, on a :

$$(x-y) \cdot \sum_{k=0}^{n} P_k(x) \cdot P_k(y) = \frac{\alpha_n}{\alpha_{n+1}} \cdot \left(P_{n+1}(x) \cdot P_n(y) - P_n(x) \cdot P_{n+1}(y) \right)$$

Démonstration — Cf. Ayant et Borg p. 119.

3.4.3 Calcul des coefficients de Gauss a_i

En (2.2.2), on a vu que les coefficients a_k sont donnés par :

$$a_k = \int_a^b L_k(x) \cdot \pi(x) dx \quad (k = 1, ..., n)$$

où :

$$L_k(x) = \prod_{\substack{j=1 \\ j \neq k}}^{n} \frac{x - x_j}{x_k - x_j} = \frac{P_n(x)}{x - x_k} \cdot \frac{1}{P_n'(x_k)}$$

donc,

$$a_k = \frac{1}{P_n'(x_k)} \cdot \int_a^b \frac{P_n(x)}{x - x_k} \cdot \pi(x) dx$$

Avec l'identité de Darboux-Christofell, on a :

$$(x - x_k) \cdot \sum_{j=0}^{n} P_j(x) \cdot P_j(x_k) = -\frac{\alpha_n}{\alpha_{n+1}} \cdot P_{n+1}(x_k) \cdot P_n(x)$$

et avec $\int_a^b P_j(x) \cdot \pi(x) dx = \left\langle P_j \middle| P_0 \right\rangle = 0$, pour $j \geq 1$, on a :

$$P_0(x_k)\cdot\int_a^b P_0(x)\cdot\pi(x)dx = -\frac{\alpha_n}{\alpha_{n+1}}\cdot P_{n+1}(x_k)\cdot\int_a^b \frac{P_n(x)}{x-x_k}\cdot\pi(x)dx$$

puis, avec $P_0(x_k) = P_0(x) = P_0$, on déduit que :

$$P_0(x_k)\cdot\int_a^b P_0(x)\cdot\pi(x)dx = \langle P_0 | P_0\rangle = 1$$

d'où :

$$a_k = -\frac{\alpha_{n+1}}{\alpha_n}\cdot\frac{1}{P_n'(x_k)\cdot P_{n+1}(x_k)}$$

Enfin, avec la récurrence vérifiée par les P_n, on a :

$$0 = a_n\cdot P_{n+1}(x_k) + c_n\cdot P_{n-1}(x_k)$$

soit,

$$a_k = \frac{\alpha_n}{\alpha_{n-1}}\cdot\frac{1}{P_n'(x_k)\cdot P_{n-1}(x_k)}$$

Remarque — Les polynômes orthogonaux classiques sont, en général, plus faciles à manipuler non normalisés.

Si (P_n) est une famille de polynômes orthogonaux, on les normalise en posant :

$$P_n^* = \frac{1}{\lambda_n}\cdot P_n \text{ , où } \lambda_n = \sqrt{\langle P_n | P_n\rangle}$$

et la formule ci-dessus devient :

$$a_k = (\lambda_{n-1})^2\cdot\frac{\alpha_n}{\alpha_{n-1}}\cdot\frac{1}{P_n'(x_k)\cdot P_{n-1}(x_k)}$$

3.5 Majoration de l'erreur de quadrature

On peut montrer le résultat suivant :

Théorème (Markov) : Si f est de classe C^{2n} sur]a,b[, alors l'erreur de quadrature dans la méthode de Gauss est donnée par :

$$E(f) = \frac{f^{(2n)}(\xi)}{(2n)!}\cdot\frac{\langle P_n | P_n\rangle}{\alpha_n^2}$$

où ξ est dans]a,b[.

Démonstration — (Cf. Stoer et Burlisch p. 150, ou Deheuvels p. 217)

3.6 Exemples classiques de polynômes orthogonaux et formules de quadrature correspondantes

Pour les propriétés des polynômes orthogonaux, on pourra se reporter à Ayant et Borg.

La programmation des méthodes correspondantes se trouve dans le paquetage INTEGRATION_GENERIQUE sur la disquette.

3.6.1 Les polynômes de Legendre

Ce sont les polynômes définis, à une constante multiplicative près, par :

$$[a,b] = [-1,1] \text{ et } \pi(x) = 1$$

Ils sont classiquement définis par la formule de Rodrigues :

$$P_n(x) = \frac{1}{2^n \cdot n!} \cdot \left(\frac{d}{dx}\right)^n \left(\left(x^2 - 1\right)^n\right)$$

soit :

$$P_n(x) = \frac{1}{2^n} \cdot \sum_{k=0}^{\left[\frac{n}{2}\right]} (-1)^k \cdot C_n^k \cdot C_{2(n-k)}^n \cdot x^{n-2k}$$

où [ ] désigne la fonction partie entière.

On peut aussi les définir par la récurrence :

$$\begin{cases} P_0(x) = 1,\ P_1(x) = x \\ (n+1) \cdot P_{n+1}(x) = (2 \cdot n + 1) \cdot x \cdot P_n(x) - n \cdot P_{n-1}(x) \quad (n \geq 1) \end{cases}$$

On peut encore définir P_n comme la solution polynomiale de l'équation différentielle :

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

ou par :

$$\varphi(x,z) = \left(1 - 2 \cdot x \cdot z + z^2\right)^{\frac{-1}{2}} = \sum_{n=0}^{+\infty} P_n(x) \cdot z^n \quad (|z| < 1)$$

On dit que φ est la *« fonction génératrice »* des P_n.

On a la majoration :

$$\forall x \in [-1,1],\ |P_n(x)| \leq 1$$

Enfin, on peut localiser les zéros de P_n grâce aux *inégalités de Brun* (Cf. Ayant et Borg p. 138) :

$$\text{Cos}\left(\frac{2k}{2n+1} \cdot \pi\right) \leq x_k \leq \text{Cos}\left(\frac{2k-1}{2n+1} \cdot \pi\right)$$

valables pour tous les zéros strictement positifs de P_n, et :

$$x_k = -x_{n+1-k} \ (k = 1, ..., n)$$

ce qui résulte de $P_n(-x) = (-1)^n \cdot P_n(x)$ (donc les racines vont former un ensemble symétrique par rapport à 0).

On déduit alors de ce qui précède que les a_k sont donnés par :

$$a_k = \frac{2 \cdot \left(1 - x_k^2\right)}{\left(n \cdot P_{n-1}(x_k)\right)^2} \qquad (n = 1, ..., n)$$

avec aussi : $a_k = a_{n+1-k}$ $(k = 1, ..., n)$.

La programmation du calcul des P_n, de leurs racines x_k et des coefficients a_k est alors élémentaire. L'exécution d'un tel programme nous donne, par exemple, les résultats suivants :

Polynôme L_{24}

$$L24(x) = 0.16118 - 48.35408 \cdot x^2 + 2393.52683 \cdot x^4$$

$$- 46274.85201\cdot x^6 + 461095.84685\cdot x^8$$
$$- 2705095.63485\cdot x^{10} + 10041642.88695\cdot x^{12} - 24497194.73519\cdot x^{14}$$
$$+ 39807941.44469\cdot x^{16} - 42669950.30673\cdot x^{18}$$
$$+ 28970650.47141\cdot x^{20} - 11287266.41743\cdot x^{22} + 1922106.96239\cdot x^{24}$$

Zéros du polynôme L_{24}

$x_1 = -x_{24} = -0.995187220000022$	$x_2 = -x_{23} = -0.974728555978706$
$x_3 = -x_{22} = -0.938274551976997$	$x_4 = -x_{21} = -0.886415527020260$
$x_5 = -x_{20} = -0.820001985964673$	$x_6 = -x_{19} = -0.740124191575882$
$x_7 = -x_{18} = -0.648093651937057$	$x_8 = -x_{17} = -0.545421471388688$
$x_9 = -x_{16} = -0.433793507626036$	$x_{10} = -x_{15} = -0.315042679696163$
$x_{11} = -x_{14} = -0.191118867473616$	$x_{12} = -x_{13} = -0.064056892862606$

Coefficients de Gauss du polynôme L_{24}

$a_1 = a_{24} = 0.012341229289022$	$a_2 = a_{23} = 0.028531388532063$
$a_3 = a_{22} = 0.044277438545067$	$a_4 = a_{21} = 0.059298584856864$
$a_5 = a_{20} = 0.073346481465724$	$a_6 = a_{19} = 0.086190161546685$
$a_7 = a_{18} = 0.097618652106859$	$a_8 = a_{17} = 0.107444270116811$
$a_9 = a_{16} = 0.115505668053805$	$a_{10} = a_{15} = 0.121670472927805$
$a_{11} = a_{14} = 0.125837456346828$	$a_{12} = a_{13} = 0.127938195346752$

Les polynômes de Legendre peuvent être utilisés pour calculer une intégrale définie sur un intervalle [a,b], le changement de variable :

$$x = \frac{b+a}{2} + \frac{b-a}{2}\cdot t$$

nous ramenant à [–1,1].

Ce qui donne la formule de quadrature de Legendre :

$$I(f) \cong \hat{I}(f) = \frac{b-a}{2}\cdot\sum_{k=1}^{n} a_k \cdot f(\xi_k)$$

où :

$$\xi_k = \frac{b+a}{2} + \frac{b-a}{2}\cdot x_k \quad (k = 1, ..., n)$$

Là encore la programmation est évidente.

Pour l'évaluation de l'erreur, avec :

$$\langle P_n | P_n \rangle = \int_{-1}^{1} P_n(x)^2\,dx = \frac{2}{2\cdot n+1}$$

et :

$$\alpha_n = \frac{(2n)!}{2^n\cdot(n!)^2}$$

on déduit que :

$$E(f) = \frac{2^{2n+1}\cdot(n!)^2\cdot f^{(2n)}(\xi)}{(2\cdot n+1)\cdot((2n)!)^3}$$

où ξ est dans]–1,1[.

3.6.2 Polynômes de Tchébychev

Ils sont définis, à une constante multiplicative près par :

$$[a,b] = [-1,1] \text{ et } \pi(x) = \frac{1}{\sqrt{1-x^2}}$$

Une définition classique étant :

$$T_n(x) = \text{Cos}(n \cdot \text{ArcCos}(x))$$

soit

$$T_n(x) = \frac{n}{2} \cdot \sum_{k=0}^{\left[\frac{n}{2}\right]} (-1)^k \cdot \frac{(n-k-1)!}{k! \cdot (n-2k)!} \cdot (2x)^{n-2k}$$

On peut aussi les définir par la récurrence :

$$\begin{cases} T_0(x) = 1,\ T_1(x) = x \\ T_{n+1}(x) = 2 \cdot x \cdot T_n(x) + T_{n-1}(x) \quad (n \geq 1) \end{cases}$$

On peut encore définir T_n comme la solution polynomiale de l'équation différentielle :

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

ou par :

$$(1 - t^2) \cdot (1 - 2 \cdot x \cdot z + z^2)^{-1} = T_0(x) + 2 \cdot \sum_{n=1}^{+\infty} T_n(x) \cdot z^n \quad (|z| < 1)$$

Les racines de T_n sont données par :

$$x_k = \text{Cos}\left(\frac{2 \cdot k - 1}{2 \cdot n} \cdot \pi\right) \qquad (k = 1, ..., n)$$

Là encore, on a :

$$|T_n(x)| \leq 1$$

Le calcul des a_k est ici très simple car ils sont tous égaux, soit :

$$a_k = \frac{\pi}{n} \ (k = 1, ..., n)$$

La formule d'intégration approchée est alors :

$$I(f) = \int_{-1}^{1} \frac{f(x)}{\sqrt{1-x^2}} dx \cong \frac{\pi}{n} \cdot \sum_{i=1}^{n} f(x_i)$$

Avec $\langle T_n | T_n \rangle = \frac{\pi}{2}$, et $\alpha_n = 2^{n-1}$, on déduit que l'erreur de quadrature est :

$$E(f) = \frac{f^{(2n)}(\xi)}{(2 \cdot n)! \cdot 2^{2n-2}} \cdot \frac{\pi}{2}$$

3.6.3 Polynômes de Laguerre

Ils sont définis, à une constante multiplicative près par :

$$[a,b[= [0,+\infty[\text{ et } \pi(x) = e^{-x}$$

Soit classiquement par :

$$L_n(x) = \frac{e^x}{n!} \cdot \left(\frac{d}{dx} \right)^n \left(x^n \cdot e^{-x} \right)$$

donc :

$$L_n(x) = \sum_{k=0}^{n} (-1)^k \cdot C_n^{n-k} \cdot \frac{x^k}{k!}$$

Ils sont aussi définis par la récurrence :

$$\begin{cases} L_0(x) = 1,\ L_1(x) = 1 - x \\ (n+1) \cdot L_{n+1}(x) = (2 \cdot n + 1 - x) \cdot L_n(x) - n \cdot L_{n-1}(x) \quad (n \geq 1) \end{cases}$$

En utilisant le fait qu'entre deux racines de L_{n-1} il y a une seule racine de L_n, on peut localiser, de proche en proche les racines de L_n et les calculer par la méthode de Newton.

Les coefficients a_k sont donnés par :

$$a_k = \frac{x_k}{\left(n \cdot L_{n-1}(x_k) \right)^2} \ (k = 1, ..., n)$$

Ce qui donne la formule de quadrature de Laguerre :

$$I(f) = \int_0^{+\infty} f(x) \cdot e^{-x} dx \cong \hat{I}(f) = \sum_{i=1}^{n} a_i \cdot f(x_i)$$

L'erreur de quadrature étant donnée par :

$$E(f) = \frac{(n!)^2}{(2 \cdot n)!} \cdot f^{(2n)}(\xi)$$

où $\xi > 0$.

3.6.4 Polynômes d'Hermite

Ils sont définis, à une constante multiplicative près par :

$$]a,b[= R \text{ et } \pi(x) = e^{-x^2}$$

Soit classiquement par :

$$H_n(x) = (-1)^n \cdot e^{x^2} \cdot \left(\frac{d}{dx} \right)^n \left(e^{-x^2} \right)$$

C'est-à-dire :

$$H_n(x) = n! \cdot \sum_{k=0}^{\left[\frac{n}{2}\right]} (-1)^k \cdot \frac{(2 \cdot x)^{n-2k}}{k! \cdot (n - 2 \cdot k)!}$$

Ils sont aussi définis par la récurrence :

$$\begin{cases} H_0(x) = 1,\ H_1(x) = 2 \cdot x \\ H_{n+1}(x) = 2 \cdot x \cdot H_n(x) - 2 \cdot n \cdot H_{n-1}(x) \quad (n \geq 1) \end{cases}$$

Comme $H_n(-x) = (-1)^n \cdot H_n(x)$, les zéros forment un ensemble symétrique par rapport à 0, soit :

$$x_i = -x_{n+1-i} \ (i = 1, ..., n)$$

En utilisant le fait qu'entre deux racines de H_{n-1} il y a une seule racine de H_n, on peut localiser, de proche en proche les racines de H_n et les calculer par la méthode de Newton.

Les coefficients a_k étant donnés par :

$$a_k = \frac{\sqrt{\pi} \cdot 2^{n-1} \cdot (n-1)!}{\left(H_{n-1}(x_k)\right)^2 \cdot n}$$

L'erreur de quadrature étant :

$$E(f) = \frac{f^{(2n)}(\xi) \cdot \sqrt{\pi} \cdot n!}{(2 \cdot n)! \cdot 2^n}$$

4. La méthode probabiliste de Monte-Carlo

4.1 Nombres pseudo-aléatoires

Un ordinateur ne peut pas véritablement générer des nombres aléatoires, il ne peut générer que des nombres *« pseudo-aléatoires »* en ce sens que la connaissance d'un ou plusieurs premiers d'entre eux va entraîner la connaissance de tous les suivants. Les nombres obtenus auront une bonne qualité aléatoire si leur comportement, dans des simulations d'expériences aléatoires, est proche de véritables nombres aléatoires. On peut, par exemple, simuler une partie de « pile ou face » avec des nombres aléatoires entiers entre 0 et 1, et pour un grand nombre de tirages le nombre de 1 doit être sensiblement équivalent au nombre de 0.

Une méthode classique pour engendrer une suite de tels nombres est d'utiliser l'algorithme suivant :

$$\begin{cases} x_0 \text{ est donné impair} \\ x_n = a \cdot x_{n-1} \ (\text{Mod. } 2^p) \end{cases}$$

avec p donné en général entre 30 et 40 (p dépend de la machine, on prend souvent p = 32), le réel a donné comme puissance impaire élevée de 5 et la valeur initiale x_0 (en anglais *« initial seed »*) également donnée.

L'avantage de cette congruence linéaire est d'être assez rapide.

x_n étant le reste dans la division euclidienne de $a \cdot x_{n-1}$ par 2^p, on a : $x_n < 2^p$, donc la quantité $r = \frac{x_n}{2^p - 1}$ va donner un nombre aléatoire entre 0 et 1.

On peut donc obtenir par cette méthode un moyen de faire un tirage aléatoire sur [0,1[et ce tirage peut être considéré comme uniforme.

On peut également simuler un tirage aléatoire suivant une loi de Gauss de moyenne et d'écart type donné ou toute autre loi classique (Cf Nougier, p. 305).

4.2 La méthode de Monte-Carlo. Première version : tirage par « noir ou blanc »

4.2.1 Equiprobabilité sur un segment, ou un pavé de R^n

On appelle tirage uniforme sur un intervalle [a,b] un tirage aléatoire de réels x dans [a,b] tel que la probabilité que x soit dans l'intervalle I est proportionnelle à la longueur de I.

Pour un tel tirage on a donc :

$$\text{Prob}(x \in I) = \frac{|I|}{b - a}$$

De la même façon pour un tirage uniforme sur un pavé $P = \prod_{i=1}^{n} [a_i, b_i]$ de R^n, on aura :

$$\text{Prob}(x \in D) = \frac{m(D)}{m(P)}$$

4.2.2 Calcul d'une intégrale simple

Premières hypothèses — On se donne une fonction continue f de [a,b] dans [0,1].

La mesure du domaine hachuré est alors : $m(D) = \int_a^b f(x)dx$

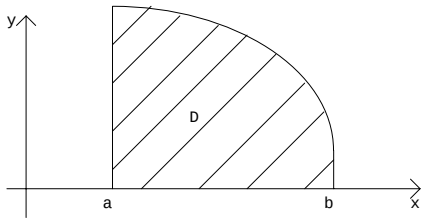


Figure 6.3

Et pour un tirage uniforme sur le rectangle P, on aura :

$$\text{Prob}(\ (x,y) \in D\ ) = \frac{\int_a^b f(x)dx}{b-a}$$

Donc, si on effectue n tirages dans P et si k est le nombre de fois où le point est tiré dans D, pour n grand on aura :

$$\text{Prob}(\ (x,y) \in D\ ) = \frac{k}{n} \cong \frac{\int_a^b f(x)dx}{b-a}$$

Cas général — Si la fonction f n'est pas à valeurs dans [0,1], on s'y ramène facilement en faisant le changement de fonction :

$$g(x) = \frac{f(x) - yMin}{yMax - yMin}$$

où yMin [Resp. yMax] est la borne inférieure [Resp. Supérieure] de f sur [a,b].

4.2.3 Calcul d'une intégrale multiple

Soit à calculer $\int_\Delta f(x)dx$, où $\Delta = \prod_{i=1}^{n} [a_i, b_i]$ est un pavé de R^p et f est continue de Δ dans [0,1].

En reprenant les idées précédentes, une méthode de calcul approché consistera donc à faire n tirages aléatoires d'un p-uplet $x = (x_1,...,x_n)$ et d'un réel y dans [0,1] puis à compter le nombre k de fois où $y \le f(x)$. Alors $\frac{k}{n} \cdot m(\Delta)$ donnera une valeur approchée de l'intégrale où $m(\Delta) = \prod_{i=1}^{n} (b_i - a_i)$.

Comme dans le cas d'une variable on peut aussi traiter le cas où f n'est pas à valeurs dans [0,1].

4.2.4 Remarques

(i) — Cette méthode est compétitive pour les intégrales multiples car pour chaque tirage aléatoire on ne fait qu'une évaluation de f (opération coûteuse en temps de calcul). Le nombre d'opérations étant proportionnel à la dimension de l'intégrale.
(ii) — Comme il s'agit de tirages aléatoires, deux tirages différents donneront des résultats différents. Il est donc conseillé de faire plusieurs expériences et de faire la moyenne des résultats obtenus. Si I_k est la valeur obtenue à l'expérience k et m le nombre d'expériences, alors on prendra pour valeur approchée :

$$I = \frac{\sum_{i=1}^{m} I_k}{m}$$

Une mesure de la dispersion des I_k autour de I est alors donnée par l'écart type σ avec :

$$\sigma^2 = \frac{\sum_{i=1}^{m} (I_k - I)^2}{m}$$

La méthode est d'autant plus efficace que σ est petit. En augmentant le nombre d'expériences, on diminuera σ mais on augmentera le temps de calcul T. Une mesure de l'efficacité de la méthode est donnée par :

$$E = \frac{1}{T \cdot \sigma^2}$$

4.3 Méthode de Monte Carlo, deuxième version

4.3.1 Remarques préliminaires

Si $f : [a,b] \to R$ est continue, sa valeur moyenne est :

$$m(f) = \frac{1}{b-a} \cdot \int_a^b f(x)dx = f(c)$$

pour un point c de [a,b].

L'idée est alors de trouver cette valeur c « au hasard ».

Pour ce faire, on effectue n tirages, suivant une même loi uniforme, sur [a,b] et si au tirage n° k, x_k est le point obtenu, on prend comme valeur approchée de m(f) la quantité :

$$m_{k,n} = f(x_k) \ (k = 1, ..., n)$$

Une valeur approchée de m(f) sera alors donnée par la valeur moyenne :

$$m_n = \frac{1}{n} \cdot \sum_{k=1}^{n} m_{k,n}$$

L'idée de cette approximation est justifiée par la loi des grands nombres.

En effet, $(x_k)_{k\geq 1}$ peut être considérée comme une suite de variables aléatoires suivant une même loi uniforme sur [a,b] et $(f(x_k))_{k\geq 1}$ est alors une suite de

variables aléatoires suivant une même loi de moyenne m(f) et d'écart-type $\sigma(f)$ définie par :

$$\sigma(f)^2 = \int_a^b \left(f(x) - m(f)\right)^2 \cdot \frac{dx}{b-a}$$

la loi des grands nombres nous dit alors que la suite (m_n) va converger en probabilité vers la moyenne m(f) et la différence m_n - m(f) a pour écart-type $\frac{\sigma(f)}{\sqrt{n}}$, c'est-à-dire que pour tout $\lambda > 0$, on a :

$$\lim_{n\to+\infty} \text{Prob}\left(\left|m_n - m(f)\right| > \lambda\right) = 0$$

l'inégalité de Bienaymé-Tchebychev nous donnant la majoration :

$$\text{Prob}\left(\left|m_n - m(f)\right| > \lambda\right) < \frac{\sigma(f)^2}{n \cdot \lambda^2}$$

4.3.2 Méthode de Monte Carlo avec échantillonnage simple

Au tirage n° k, on dispose de x_k dans [a,b], ce qui donne $y_k = f(x_k)$, puis on fait la moyenne des y_k, ce qui donne m_n et enfin l'intégrale de f est approximée par $(b - a)\cdot m_n$, soit :

$$I(f) = \int_a^b f(x)dx \cong \frac{b-a}{n} \cdot \sum_{k=1}^{n} f(x_k)$$

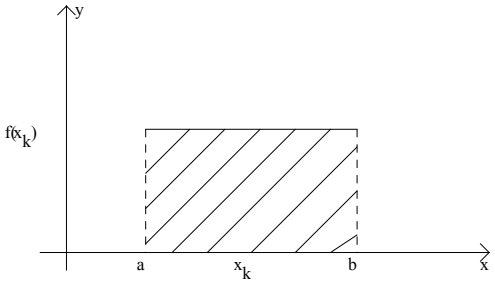


Figure 6.4

Ce qui revient à remplacer le domaine délimité par les droites x = a, x = b, l'axe des x et le graphe de f sur [a,b] par le rectangle hachuré.

Remarque : Echantillonnage stratifié — Pour améliorer la précision on peut découper l'intervalle [a,b] en sous-intervalles plus petits et appliquer la méthode sur chacun de ces sous-intervalles. On parle alors d'échantillonnage stratifié. Pour un découpage très fin, on retrouve la définition d'une intégrale à l'aide des sommes de Riemann.

La programmation structurée est alors la suivante :

```
FONCTION MonteCarlo(Entrée a, b : Réel ; n : Entier ; f : Fonction) ;
Début
    S = 0
    Pour k allant de 1 à n faire
    Début
        r = Nombre aléatoire dans [0,1] ;
        x = a + (b – a)·r ;
        S = S + f(x) ;
```

Fin ;
S = (b – a)·S/n ;
MonteCarlo = S ;
Fin ;

4.3.3 Utilisation de transformations antithétiques

Au tirage n° k, on dispose d'un nombre aléatoire r_k dans [0,1[auquel on va associer deux points de [a,b[:

$$x'_k = a + (b - a)\cdot r_k \text{ et } x''_k = a + (b - a)\cdot(1 - r_k)$$

ce qui donne deux approximations de la valeur moyenne :

$$y'_k = f(x'_k) \text{ et } y''_k = f(x''_k)$$

on prend alors comme nouvelle valeur approchée de la valeur moyenne :

$$y_k = \frac{1}{2}\cdot\left(y_k' + y_k''\right)$$

La méthode obtenue se trouve alors être plus efficace que la précédente.

On peut encore augmenter l'efficacité de cette méthode en généralisant ce qui précède de la façon suivante : au tirage n°k, on construit à partir de r_k dans [0,1[les réels $x_{k,j}$ de [a,b[en posant :

$$x_{k,j} = a + (b - a)\cdot\frac{r_k + j}{m} \quad (j = 0, ..., m - 1)$$

m étant donné. Ce qui va donner les approximations de m(f) :

$$y_{k,j} = f(x_{k,j}) \quad (j = 0, ..., m - 1)$$

on prend alors, à l'étape k, comme nouvelle approximation de m(f) la moyenne des $y_{k,j}$, soit :

$$y_k = \frac{1}{m}\cdot\sum_{j=0}^{m} y_{k,j} \quad (k = 1, ..., n)$$

En combinant cette méthode avec un échantillonnage stratifié, on gagne encore en efficacité.

La méthode Monte Carlo que nous retiendrons, pour calculer des intégrales simples, est finalement la suivante :

(a) on coupe [a,b] en deux sous-intervalles [a,c] et [c,b] avec $c = \frac{a+b}{2}$;

(b) sur chacun de ces deux sous-intervalles, on applique ce qui précède, c'est-à-dire qu'à partir de r_k dans [0,1[, on pose $r_{k,j} = \frac{r_k + j}{m}$ pour j = 0, 1, ..., m – 1 et

$$y'_{k,j} = f(a + (c - a)\cdot r_{k,j}) \text{ , } y''_{k,j} = f(c + (b - c)\cdot(1 - r_{k,j}))$$

puis :

$$y'_k = \frac{1}{m}\cdot\sum_{j=0}^{m-1} y'_{k,j}\text{ , } y''_k = \frac{1}{m}\cdot\sum_{j=0}^{m-1} y''_{k,j}$$

$$y_k = \frac{1}{2}\cdot\left(y'_k + y''_k\right)$$

enfin, une valeur approchée de l'intégrale est :

$$S = \frac{b - a}{n}\cdot\sum_{k=1}^{m} y_k$$

D'où la programmation structurée :

```
FONCTION MonteCarlo1(Entrée a, b : Réel ; m, n : Entier ; f : Fonction) ;
Début
    S = 0 ; L = (b – a)/2 ;
    Pour k allant de 1 à n Faire
    Début
        r = Nombre aléatoire dans [0,1] ;
        Sk = 0 ;
        Pour j Allant de 0 à m – 1 Faire
        Début
            Sk = Sk + f(a + r·L) + f(c + (1 – r)·L) ;
        Fin ;
        Sk = Sk/(2·m) ;
        S = S + Sk ;
    Fin ;
    S = (b – a)·S/n ;
Fin ;
```

4.3.4 Calcul d'intégrales multiples

Les méthodes utilisant les variables antithétiques et l'échantillonnage stratifié sont difficiles à mettre en oeuvre pour les intégrales multiples. On utilisera donc la méthode avec échantillonnage simple.

Tout d'abord, on se ramène à l'intégration d'une fonction sur un pavé $P = \prod_{i=1}^{n} [a_i , b_i]$ de R^p aussi petit que possible et contenant le domaine d'intégration Ω et on prolonge la fonction à intégrer en posant $f = 0$ sur $P - \Omega$.

Au tirage n° k, on extrait au hasard p valeurs $x_{i,k}$ de $[a_i,b_i]$ (i = 1, ..., p), ce qui donne :

$$y_k = f(x_{1,k},...,x_{p,k}) \quad (k = 1, ..., n)$$

et l'intégrale à calculer est approchée par :

$$\frac{m(P)}{n} \cdot \sum_{k=1}^{n} y_k$$

où :

$$m(P) = \prod_{i=1}^{p} (b_i - a_i)$$

4.3.5 Exemple : calcul des coordonnées du centre de gravité d'un corps

On considère le volume Ω défini comme l'intersection du tore de centre 0 et de rayons 4 et 2 et du pavé [1,4]x[-3,4]x[–1,1]. Il est donc défini par les relations :

$$\begin{cases} \left(\sqrt{x^2 + y^2} - 3\right)^2 + z^2 \le 1 \\ 2 \le x \le 4, \ -3 \le y \le 4, \ -1 \le z \le 1 \end{cases}$$

Les coordonnées du centre de gravité sont alors données par :

$$x_G = \frac{I_x}{I}, \ y_G = \frac{I_y}{I}, \ z_G = \frac{I_z}{I}$$

où :

$$I = \int_{\Omega} \rho(x,y,z)dxdydz;\ I_x = \int_{\Omega} x \cdot \rho(x,y,z)dxdydz$$

$$I_y = \int_{\Omega} y \cdot \rho(x,y,z)dxdydz;\ I_z = \int_{\Omega} z \cdot \rho(x,y,z)dxdydz$$

ρ étant la densité volumique définie, pour notre exemple par :

$$\rho(x,y,z) = e^{5\cdot z}$$

En posant $e^{5\cdot z}dz = ds$, soit $s = \frac{1}{5}\cdot e^{5\cdot z}$, on se ramène à :

$$\rho(x,y,z)dxdydz = dxdyds$$

ce qui va faciliter les calculs numériques.

Le nouveau domaine d'intégration est alors :

$$P = [1,4]x[-3,4]x[s_0,s_1]$$

où $s_0 = \frac{1}{5}\cdot e^{-5\cdot} = 0.00135$ et $s_1 = \frac{1}{5}\cdot e^{5} = 29.682$.

Le volume V de ce domaine étant $V = 3x7x0.2x(e^5 - e^{-5})$.

La fonction à intégrer valant 1 dans Ω et 0 ailleurs.

Remarque — Dans la pratique, il faudra toujours essayer par changement de variable ou autre de simplifier l'intégrale multiple à calculer.

5. Transformation de Fourier rapide

5.1 Position du problème. Notations

Soit $f : R \to C$ un signal dépendant du temps vérifiant les propriétés suivantes :

$$\begin{cases} f(t) = 0 \text{ pour } t < 0 \\ f \text{ continue sur }]0,+\infty[\\ f \in L^1(R) \\ \lim\limits_{t\to+\infty} f(t) = 0 \end{cases}$$

En général ce signal n'est pas connu de façon explicite mais seulement en quelques points équidistants t_j.

Pour $\varepsilon > 0$ assez petit, on peut trouver $t_{max} > 0$ assez grand tel que :

$$|f(t)| < \varepsilon \text{ pour } t > t_{max}$$

Pour un tel t_{max}, on pose alors :

$$\begin{cases} \delta = \dfrac{t_{max}}{n} \\ t_j = j\cdot\delta \\ \qquad\qquad (j = 0,\ 1,\ \ldots,\ n) \\ f_j = f(t_j) \end{cases}$$

On définit la *fréquence de Nyquist* par :

$$x_c = \frac{1}{2\cdot\delta}$$

Comme la transformée de Fourier $\hat{f}$ est nulle à l'infini, pour t_{max} donné on pourra trouver n assez grand pour que :

$$\left|\hat{f}(x)\right| < \varepsilon \text{ pour } |x| > x_c$$

C'est-à-dire que f est négligeable en dehors de $[0,t_{max}]$ et $\hat{f}$ est négligeable en dehors de $[-x_c,x_c]$:

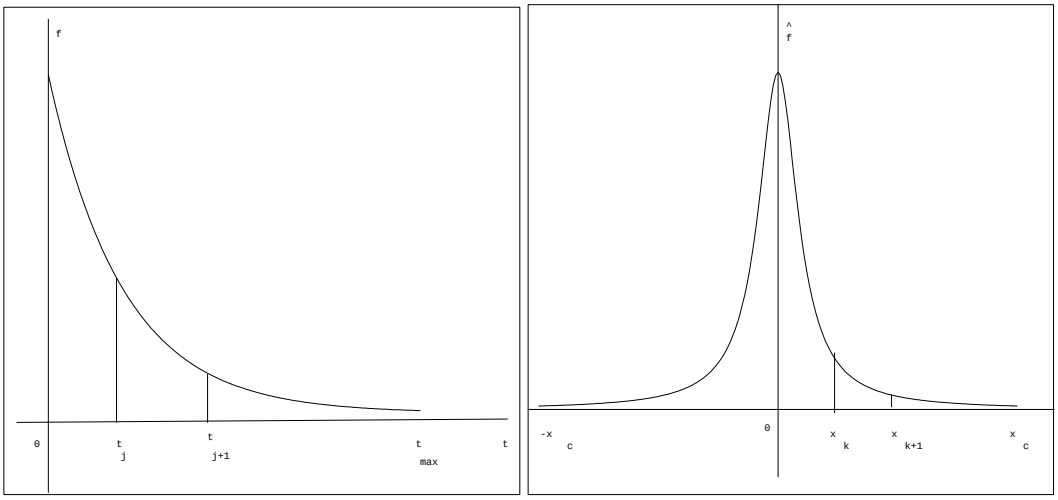


Figure 6.5

Le problème sera donc de calculer de façon approchée les valeurs de la transformée de Fourier de f en n points équidistants x_k dans l'intervalle $[-x_c,x_c]$.

On note donc :

$$x_k = k \cdot \frac{2 \cdot x_c}{n} = \frac{k}{n \cdot \delta} \quad \left(k = -\frac{n}{2},\ldots,\frac{n}{2}\right)$$

5.2 Approximation des $\hat{f}(x_k)$

Le calcul approché des intégrales de Fourier se fera en utilisant la méthode des rectangles à gauche, soit :

$$\hat{f}(x_k) = \int_{-\infty}^{+\infty} f(t) \cdot e^{-2i\pi x_k t} dt \cong \int_0^{t_{max}} f(t) \cdot e^{-2i\pi x_k t} dt$$

et :

$$\hat{f}(x_k) \cong \frac{t_{max}}{n} \cdot \sum_{j=0}^{n-1} f(t_j) \cdot e^{-2i\pi x_k t_j}$$

en tenant compte de :

$$x_k \cdot t_j = \frac{k}{n \cdot \delta} \cdot j \cdot \delta = \frac{k \cdot j}{n}$$

et en posant :

$$\omega = e^{-\frac{2i\pi}{n}} \quad \text{(racine } n^{\text{ème}} \text{ de l'unité)}$$

il s'agit de calculer les quantités :

$$\hat{y}_k = \delta \cdot \sum_{j=0}^{n-1} \omega^{jk} \cdot f_j = \delta \cdot y_k \quad \left(k = -\frac{n}{2},\ldots,\frac{n}{2}\right)$$

Remarque — Avec la 2π-périodicité de la fonction $t \to e^{it}$, on voit que les y_k sont définis pour tout k dans Z et qu'ils forment une suite n-périodique, c'est-à-dire que :$y_{k+n} = y_k$ $(k \in Z)$

Il nous suffit donc, pour déterminer complètement cette suite de calculer les y_k pour k = 0, 1, ..., n – 1.

Du point de vue pratique, les valeurs approchées des $\hat{f}(x_k)$ sont données par :

$$\begin{cases} k = 0 & \text{pour } x_k = 0 \\ k = 1,\ 2,\ \ldots,\ \dfrac{n}{2} - 1 & \text{pour } x_k \in \left]0, x_c\right[\\ k = \dfrac{n}{2} + 1,\ \ldots,\ n - 1 & \text{pour } x_k \in \left]-x_c, 0\right[\\ k = \dfrac{n}{2} & \text{pour } x_k = x_c \text{ et } x_k = -x_c \end{cases}$$

5.3 La transformation de Fourier discrète

5.3.1 Définition, propriétés

Définition : Soit n un entier positif non nul et $\omega = e^{-\frac{2i\pi}{n}}$ une racine $n^{ème}$ de l'unité. La *transformation de Fourier discrète* est l'application linéaire :

$$F: \begin{array}{ccc} C^n & \to & C^n \\ x & \mapsto & y = F(x) \end{array}$$

qui associe au vecteur x de C^n de composantes $x_0, x_1, \ldots, x_{n-1}$ le vecteur y de composantes $y_0, y_1, \ldots, y_{n-1}$ définies par :

$$y_k = \sum_{j=0}^{n-1} \omega^{j\cdot k} \cdot x_j \qquad (k = 0, 1, \ldots, n-1)$$

On peut aussi l'écrire sous forme matricielle : $y = F \cdot x$ où F est la matrice à coefficients dans C :

$$F = \begin{pmatrix} 1 & 1 & \ldots. & 1 \\ 1 & \omega & \ldots. & \omega^{n-1} \\ . & . & \ldots. & . \\ 1 & \omega^{n-1} & \ldots. & \omega^{(n-1)^2} \end{pmatrix}$$

Comme pour la transformation intégrale de Fourier, on a une formule d'inversion et un théorème de Plancherel, ce qui est résumé dans le :

Théorème : (i) Formule d'inversion — La transformation de Fourier discrète est un isomorphisme de C^n, d'inverse F^{-1} définie par :

$$(x = F^{-1}(y)) \Leftrightarrow \left(x_j = \frac{1}{n} \cdot \sum_{k=0}^{n-1} \omega^{-j\cdot k} \cdot y_k \right) \qquad (j = 0, 1, \ldots, n-1))$$

(ii) Formule de Plancherel — Si y = F(x), alors :

$$\sum_{j=0}^{n-1} \left| x_j \right|^2 = \frac{1}{n} \cdot \sum_{k=0}^{n-1} \left| y_k \right|^2$$

Démonstration — Théodor, p. 561.

5.3.2 Calcul direct de la transformée de Fourier discrète

Tout d'abord, on écrit une procédure de calcul des ω^r, pour r = 0, 1, ..., n – 1, les valeurs de ω^r étant stockées dans un vecteur W de C^n.

Ce qui donne, en disposant d'opérations sur les nombres complexes :

```
PROCEDURE PuissancesOmega(Entrée n : Entier ; Sortie W : VecteurComplexe) ;
Début
   W0 = 1 ; W1  = Cos(2π/n) – i·Sin(2π/n) ;
   Pour r allant de 2 à n - 1 Faire Wr = Wr–1·W1 ;
Fin ;
```

Ce calcul nécessitant n – 2 multiplications complexes.

En remarquant que pour $0 \leq j, k \leq n - 1$, on a : $\omega^{k \cdot j} = \omega^r$, où r est le reste dans la division euclidienne de j·k par n, on en déduit la procédure de calcul du vecteur y :

```
PROCEDURE FourierDiscret_1(Entrée n : Entier ; W, x : VecteurComplexe ;
                                   Sortie y : VecteurComplexe) ;
Début
   Pour k Allant de 0 à n – 1 Faire yk = x0 ;
   Pour j Allant de 1 à n – 1 Faire y0 = y0 + xj ;
   Pour k Allant de 1 à n – 1 Faire
   Début
      Pour j Allant de 1 à n – 1 Faire
      Début
         r = j·k Modulo n ;
         yk = yk + ωr·xj ;
      Fin ;
   Fin ;
Fin ;
```

Cette procédure de calcul très simple à le gros inconvénient d'être lente. En effet, le calcul de y_0 nécessite n – 1 additions complexes et pour k = 1, ..., n – 1, le calcul de chaque y_k nécessite n - 1 multiplications complexes et n - 1 additions complexes, ce qui donne en ajoutant le nombre d'opérations nécessaires au calcul de W un total de $(n - 2) + (n - 1) + 2 \cdot (n - 1)^2$ opérations complexes. Soit un nombre d'opérations de l'ordre de n^2, ce qui sera peu performant pour n grand (par exemple $n = 10^6$ demandera plusieurs jours de calcul).

Un algorithme très performant est celui de Cooley et Tukey.

5.4 L'algorithme FFT de Cooley et Tukey

5.4.1 Introduction

Cet algorithme, aussi connu sous le nom de *FFT* (*Fast Fourier Transform*), a été mis au point dans les années 1960 par Cooley et Tukey sur une idée de Danielson et Lanczos en 1942. Il demandera, pour le calcul d'une transformée de Fourier à n points, un nombre d'opérations de l'ordre de n·Ln(n), ce qui est beaucoup plus performant que l'algorithme direct, le prix à payer étant une programmation plus délicate.

L'idée de base est d'écrire une transformée à n points comme somme de deux transformées à n/2 points et ainsi de suite. Pour ce faire on supposera que n est une puissance de 2.

On notera donc pour tout ce qui suit :

$$n = 2^p \ (p \geq 1) \text{ et } \omega_p = e^{-\frac{2i\pi}{n}}$$

5.4.2 Cas particulier n = 2

Dans ce cas, on a p = 1 et $\omega_1 = -1$, avec les formules :

$$\begin{cases} y_0 = x_0 + x_1 \\ y_1 = x_0 - x_1 \end{cases}$$

ce qui s'écrit globalement :

$$(1)\, y_{k_0} = \sum_{j_0=0}^{1} (-1)^{k_0 \cdot j_0} \cdot x_{j_0} \quad (k_0 = 0, 1)$$

5.4.3 Cas particulier n = 4

Dans ce cas, on a p = 2 et $\omega_2 = -i$, avec les formules :

$$\begin{cases} y_0 = x_0 + x_1 + x_2 + x_3 \\ y_1 = x_0 + \omega_2 \cdot x_1 + \omega_2^2 \cdot x_2 + \omega_2^3 \cdot x_3 \\ y_2 = x_0 + \omega_2^2 \cdot x_1 + \omega_2^4 \cdot x_2 + \omega_2^6 \cdot x_3 \\ y_3 = x_0 + \omega_2^3 \cdot x_1 + \omega_2^6 \cdot x_2 + \omega_2^9 \cdot x_3 \end{cases}$$

en séparant dans chaque somme les termes d'indices pairs et impairs et en tenant compte de la périodicité, on a :

$$\begin{cases} y_0 = (x_0 + x_2) + (x_1 + x_3) \\ y_1 = (x_0 - x_2) + \omega_2 \cdot (x_1 - x_3) \\ y_2 = (x_0 + x_2) + \omega_2^2 \cdot (x_1 + x_3) \\ y_3 = (x_0 - x_2) + \omega_2^3 \cdot (x_1 - x_3) \end{cases}$$

En décomposant tout nombre entier $r \in \{0, 1, ..., n-1\}$ en base deux sous la forme :

$$r = r_0 + r_1 \cdot 2 + ... + r_{p-2} \cdot 2^{p-2} + r_{p-1} \cdot 2^{p-1} \quad (r_j = 0 \text{ ou } 1)$$

et en notant, pour j = 0, 1, ..., n – 1 :

$$x_j = x_{j_0, j_1, ..., j_{p-1}}$$

les formules vont s'écrire :

$$\begin{cases} y_{0,0} = (x_{0,0} + x_{0,1}) + (x_{1,0} + x_{1,1}) \\ y_{1,0} = (x_{0,0} - x_{0,1}) + \omega_2 \cdot (x_{1,0} - x_{1,1}) \\ y_{0,1} = (x_{0,0} + x_{0,1}) + \omega_2^2 \cdot (x_{1,0} + x_{1,1}) \\ y_{1,1} = (x_{0,0} - x_{0,1}) + \omega_2^3 \cdot (x_{1,0} - x_{1,1}) \end{cases}$$

ce qui peut s'écrire globalement :

$$y_k = y_{k_0,k_1} = \sum_{j_0=0}^{1} \omega^{2\cdot k_1 \cdot j_0} \cdot \sum_{j_1=0}^{1} \omega^{k_0\cdot(j_0+2\cdot j_1)} \cdot x_{j_0,j_1} \qquad (k_0, k_1 = 0, 1)$$

5.4.4 Cas général $n = 2^p$

Pour tout $k = 0, 1, ..., n - 1$, y_k est une somme du type :

$$S = \sum_{j=0}^{n-1} S_j = \sum_{j=0}^{n-1} S_{j_0,...,j_{p-1}}$$

qui peut s'écrire sous la forme :

$$S = \sum_{j_0=0}^{1} \sum_{j_1=0}^{1} .... \sum_{j_{p-1}=0}^{1} S_{j_0,...,j_{p-1}}$$

ce qui donne :

$$y_k = \sum_{j_0=0}^{1} \sum_{j_1=0}^{1} .... \sum_{j_{p-1}=0}^{1} \omega_p^{k\cdot j} \cdot x_{j_0,...,j_{p-1}}$$

En remarquant que pour $j, k = 0, 1, ..., n - 1$, on a :

$j\cdot k \equiv k_0\cdot(j_0 + ... j_{p-1}\cdot 2^{p-1}) + 2\cdot k_1\cdot(j_0 + ... j_{p-2}\cdot 2^{p-2}) + ... + 2^{p-1}\cdot k_{p-1}\cdot j_0$ (Mod. n)

on déduit que :

$$\omega_p^{k\cdot j} = \omega_p^{2^{p-1}\cdot k_{p-1}\cdot j_0} \cdot \omega_p^{2^{p-2}\cdot k_{p-2}\cdot(j_0+2\cdot j_1)} \cdot .... \cdot \omega_p^{k_0\cdot(j_0+...+2^{p-1}\cdot j_{p-1})}$$

et donc que la transformation de Fourier discrète peut s'écrire :

$$(p) \qquad y_k = \sum_{j_0=0}^{1} \omega_p^{2^{p-1}\cdot k_{p-1}\cdot j_0} \cdot \sum_{j_1=0}^{1} \omega_p^{2^{p-2}\cdot k_{p-2}\cdot(j_0+2\cdot j_1)} \cdot .... \sum_{j_{p-1}=0}^{1} \omega_p^{k_0\cdot(j_0+...+2^{p-1}\cdot j_{p-1})} \cdot x_{j_0,...,j_{p-1}}$$

5.4.5 Utilisation de (p) pour calculer la transformation de Fourier discrète

On décrit tout d'abord l'algorithme pour $n = 4$.

Etape 0 — On pose $x^{(0)} = x$.

Les composantes de $x^{(0)}$ sont donc les x_{j_0,j_1} pour $j_0, j_1 = 0, 1$.

Etape 1 — On construit $x^{(1)}$ à partir de $x^{(0)}$ en s'inspirant de la récurrence (p), soit en notant $x^{(1)}_{j_0,k_0}$ les coordonnées de $x^{(1)}$:

$$x^{(1)}_{j_0,k_0} = \sum_{j_1=0}^{1} \omega_2^{k_0\cdot(j_0+2\cdot j_1)} \cdot x^{(0)}_{j_0,j_1}$$

Etape 2 — On construit de la même façon $x^{(2)}$ en fonction de $x^{(1)}$ en notant $x^{(2)}_{k_1,k_0}$ ses composantes, soit :

$$x^{(2)}_{k_0,k_0} = \sum_{j_0=0}^{1} \omega_2^{2\cdot k_1\cdot j_0} \cdot x^{(1)}_{j_0,k_0}$$

Le vecteur $x^{(2)}$ donne donc les composantes de y, mais elles sont dans l'ordre binaire inverse :

$$x^{(2)}_{k_1,k_0} = y_{k_0,k_1}$$

Dans le cas général, on procède comme suit :

Etape 0 — On pose $x^{(0)} = x$.

Etape r + 1 (r = 0, 1, ..., p – 1) — L'étape précédente a donné le vecteur $x^{(r)}$ de composantes $x^{(r)}_{j_0,\ldots,j_{p-r-1},k_{r-1},\ldots,k_0}$ et on veut calculer celles de $x^{(r+1)}$ notées $x^{(r+1)}_{j_0,\ldots,j_{p-r-2},k_r,\ldots,k_0}$

Avec la récurrence (p), on déduit alors que :

$$x^{(r+1)}_{j_0,\ldots,j_{p-r-2},k_r,\ldots,k_0} = \sum_{j_{p-r-1}=0}^{1} \omega_p^{2^r\cdot k_r\cdot\left(j_0+\ldots+2^{p-r-1}\cdot j_{p-r-1}\right)} \cdot x^{(r)}_{j_0,\ldots,j_{p-r-1},k_{r-1},\ldots,k_0}$$

On pose alors :

$$m = \frac{n}{2^{r+1}} = 2^{p-r-1}$$

$$L = j_0 + \ldots + j_{p-r-2}\cdot 2^{p-r-2} \in \{0,\ldots,m-1\}$$

$$h = k_{r-1} + \ldots + k_0\cdot 2^{r-1} \in \{0,\ldots,2^r - 1 = \frac{n}{2\cdot m} - 1\}$$

$$i = j_0 + \ldots + j_{p-r-2}\cdot 2^{p-r-2} + 0\cdot 2^{p-r-1} + k_{r-1}\cdot 2^{p-r} + \ldots + k_0\cdot 2^{p-1} = L + 2\cdot m\cdot h$$

$$j = j_0 + \ldots + j_{p-r-2}\cdot 2^{p-r-2} + 1\cdot 2^{p-r-1} + k_{r-1}\cdot 2^{p-r} + \ldots + k_0\cdot 2^{p-1} = i + m$$

et les formules précédentes vont s'écrire :

$$\begin{cases} x_i^{(r+1)} = x_i^{(r)} + x_j^{(r)} & (k_r = 0) \\ x_j^{(r+1)} = \omega_p^{2^r\cdot L}\cdot x_i^{(r)} + \omega_p^{2^r\cdot\left(L+2^{p-r-1}\right)}\cdot x_j^{(r)} & (k_r = 1) \end{cases}$$

Avec :

$$\omega_p^{2^{r+p-r-1}} = -1 \text{ et } \omega_p^{2^r} = \omega_{p-r}$$

on déduit que :

$$\begin{cases} x_i^{(r+1)} = x_i^{(r)} + x_j^{(r)} \\ x_j^{(r+1)} = \left(\omega_{p-r}\right)^L\cdot\left(x_i^{(r)} - x_j^{(r)}\right) \end{cases}$$

Et au bout de p étapes, on a :

$$x^{(p)}_{k_{p-1},\ldots,k_0} = y_{\sigma(k)}$$

où σ est *l'inversion binaire* définie par :

$$\sigma(k_0 + \ldots + k_{p-1}\cdot 2^{p-1}) = k_{p-1} + \ldots + k_0\cdot 2^{p-1}$$

Pour retrouver les composantes de y dans le bon ordre on devra donc faire une inversion binaire sur celles de $x^{(p)}$.

5.4.6 Nombre d'opérations élémentaires dans l'algorithme de Cooley et Tukey

En notant, pour p ≥ 1 :

$$\begin{cases} A_p = \text{Nombre d'additions complexes;} \\ M_p = \text{Nombre de multiplications complexes;} \end{cases}$$

on a :

$$\begin{cases} A_1 = 2 \\ M_1 = 0 \end{cases}$$

et la récurrence :

$$\begin{cases} A_p = 2 \cdot A_{p-1} + 2^p \\ M_p = 2 \cdot M_{p-1} + 2^{p-1} \end{cases}$$

d'où :

$$\begin{cases} M_p = A_{p-1} \\ A_p = p \cdot 2^p \end{cases}$$

Ce qui donne un nombre d'opérations élémentaires de l'ordre de $n \cdot Log_2(n)$. (Cf. Théodor, p. 563)

5.4.7 Programmation structurée

```
PROCEDURE FFT(Entrée p : Entier ; x : VecteurComplexe ; Sortie y : VecteurComplexe) ;
Début
   n = 2^p ;
   m = n/2 ; { Etape r + 1 = 1 }
   Pour r allant de 0 à p – 1 Faire
   Début
      w1 = 1 ; {w1 = ω^L_{p-r}, avec L = 0}
      w2 = Cos(π/m) – i·Sin(π/m) ; {w2 = ω_{p-r} }
      Pour L allant de 0 à m – 1 Faire
      Début
         Pour h Allant de 0 à n/(2·m) – 1 Faire
         Début
            i = L + 2·m·h ;
            j = i + m ;
            Aux = (x_i – x_j)·w1 ;
            x_i = x_i + x_j ;
            x_j = Aux ;
         Fin ;
         w1 = w1·w2 ; {w1 = ω^L_{p-r} }
      Fin ;
      m = m/2 ;
   Fin ;
   InversionBinaire(p,x,y) ;
Fin ;
```

Pour retrouver les $\hat{y}_k$, il ne faudra pas oublier de multiplier les y_k par le coefficient δ.

En notant :

$$k = k_0 + k_1 \cdot 2 + ... + k_{p-1} \cdot 2^{p-1}$$

le calcul de l'inverse binaire $h = \sigma(k)$, se fait en utilisant les remarques suivantes :

$$k_r = (k \text{ Divise } 2^r) \text{ Modulo } 2 \; ; \; h_{p-r-1} = k_r \quad (r = 0, 1, ... \; p - 1)$$

Ce qui donne la procédure :

```
PROCEDURE InversionBinaire(Entrée p : Entier ; x : VecteurComplexe ;
                                Sortie y : VecteurComplexe) ;
Début
   n = 2^p ;
   Pour k Allant de 0 à n – 1 Faire
   Début
      h = 0 ;
      Aux = n/2 ; {Aux = 2^(p–1)}
      Pour r Allant de 0 à p – 1 Faire
      Début
         c = (k Divise 2^r) Modulo 2 ; { c = k_r }
         h = h + c·Aux ; {c·Aux = k_r·2^(p–r–1)}
         Aux = Aux/2 ; { = 2^(p–1–(r+1)) }
      Fin ;
      y_h = x_k ;
   Fin ;
Fin ;
```

5.5 Application au calcul des coefficients de Fourier d'une fonction périodique

Soit $f : R \to C$, continue par morceaux et t_{max}-périodique. Ces coefficients de Fourier complexes sont définis par :

$$c_k = \frac{1}{t_{max}} \cdot \int_0^{t_{max}} f(t) \cdot e^{-\frac{2ik\pi}{t_{max}} \cdot t} dt = \frac{1}{t_{max}} \cdot \int_0^{t_{max}} f(t) \cdot e^{-2i\pi x_k \cdot t} dt$$

Soit en notant encore f la fonction coïncidant avec la précédente sur $[0,t_{max}]$ et nulle ailleurs :

$$c_k = \frac{1}{t_{max}} \cdot \hat{f}(x_k) \qquad (k = -\frac{n}{2}, ..., 0, ..., \frac{n}{2})$$

C'est-à-dire que si $y = \left(y_{-\frac{n}{2}}, ..., y_0, ..., y_{\frac{n}{2}} \right)$ est la transformée de Fourier discrète de $f = (f_0, ..., f_{n-1})$, alors :

$$c_k = \frac{\delta}{t_{max}} \cdot y_k = \frac{1}{n} \cdot y_k \qquad (k = -\frac{n}{2}, ..., 0, ..., \frac{n}{2})$$

Soit avec la n-périodicité de la suite (y_k) :

$$\begin{cases} c_k = \frac{1}{n} \cdot y_k & \left(k = 0, \ldots, \frac{n}{2} - 1 \right) \\ c_k = \frac{1}{n} \cdot y_{k+n} & \left(k = -\frac{n}{2}, \ldots, -1 \right) \end{cases}$$

On retrouve les coefficients de Fourier trigonométriques avec :

$$a_0 = c_0 = \mathrm{Re}(c_0)$$

$$a_k = c_k + c_{-k} = c_k + \bar{c}_k = 2 \cdot \mathrm{Re}(c_k) \qquad \left(k = 1, \ldots, \frac{n}{2} - 1\right)$$

$$b_k = i \cdot (c_k - c_{-k}) = i \cdot (c_k - \bar{c}_k) = -2 \cdot \mathrm{Im}(c_k) \qquad \left(k = 1, \ldots, \frac{n}{2} - 1\right)$$

En cas de convergence de la série de Fourier, on a alors :

$$f(t) \cong a_0 + \sum_{k=1}^{\frac{n}{2}-1} \left\{ a_k \cdot \mathrm{Cos}\left(k \cdot \frac{2 \cdot \pi}{t_{max}} \cdot t \right) + b_k \cdot \mathrm{Sin}\left(k \cdot \frac{2 \cdot \pi}{t_{max}} \cdot t \right) \right\}$$

ou encore :

$$f(t) \cong a_0 + \sum_{k=1}^{\frac{n}{2}-1} \left\{ a_k \cdot \mathrm{Cos}(2 \cdot \pi \cdot x_k \cdot t) + b_k \cdot \mathrm{Sin}(2 \cdot \pi \cdot x_k \cdot t) \right\}$$

En posant :

$$A_0 = a_0$$

$$A_k = 2 \cdot |c_k| = |a_k + i \cdot b_k|$$

$$\varphi_k = \mathrm{Arg}(a_k + i \cdot b_k) = -\mathrm{Arctg}\left(\frac{\mathrm{Im}(c_k)}{\mathrm{Re}(c_k)} \right)$$

on a aussi :

$$f(t) \cong A_0 + \sum_{k=1}^{\frac{n}{2}-1} A_k \cdot \mathrm{Cos}(2 \cdot \pi \cdot x_k \cdot t - \varphi_k)$$

6. Exercices

6.1 Aire intérieure à une courbe fermée

On considère une courbe définie par une équation polaire $\rho = f(\theta)$, avec $\theta \in [0,2 \cdot \pi[$.

On veut calculer $S = \mathrm{Aire}\{ (\rho,\theta) \in Rx[0,2 \cdot \pi[\ ; \rho < f(\theta) \}$

1°) En s'inspirant de la méthode des trapèzes, écrire un algorithme de calcul de S.

2°) Application : aire de la cardïode $\rho = 1 + \cos(\theta)$.

6.2 Séries de Fourier

1°) Ecrire une procédure Fourier, qui permet de calculer les coefficients du développement en série de Fourier d'une fonction périodique f (Les intégrales donnant les coefficients de Fourier pourront être calculées successivement par les

méthodes des trapèzes, de Simpson et de Romberg, puis on comparera les temps de calcul).
2°) Ecrire un programme qui permet de comparer la fonction T-périodique f avec son développement en série de Fourier. On pourra tracer, sur [–T,T], les courbes y = f(x) et

$$y = a_0 + \sum_{i=1}^{n}\left(a_i \cdot \mathrm{Cos}\left(i \cdot \frac{2 \cdot \pi}{T} \cdot x \right) + b_i \cdot \mathrm{Sin}\left(i \cdot \frac{2 \cdot \pi}{T} \cdot x \right) \right)$$

les a_i et b_i étant les coefficients de Fourier de f.
3°) *Application* — Appliquer ce qui précède à la fonction f, $2 \cdot \pi$-périodique définie par :

$$f_p(x) = \frac{\mathrm{Sin}(p \cdot x)}{x}, \text{ pour } x \in [-\pi,\pi]$$

6.3 Circuit R-C

On considère le circuit R-C suivant, soumis à une entrée e(t) :

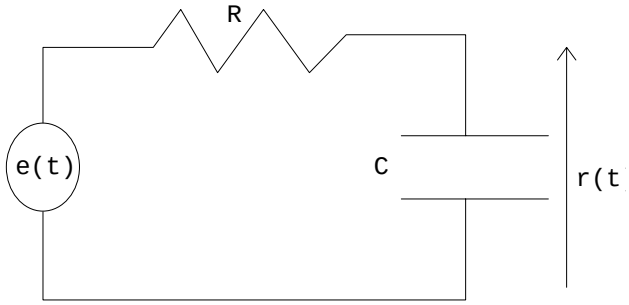


Figure 6.6

On pose $R \cdot C = \tau$.

On a alors : r = h*e, avec h(t) = 0 pour t < 0 et $h(t) = \frac{1}{\tau} \cdot e^{-\frac{t}{\tau}}$, pour t > 0, le signe * désignant le *produit de convolution* défini dans le cas où f(x) = g(x) = 0 pour x < 0 par :

$$f * g(x) = \int_0^x f(t) \cdot g(x-t)\,dt$$

1°) Ecrire une procédure de calcul de r(t), pour tout t dans R.
Applications

(a) $e(t) = \begin{cases} 0 \text{ si } t < 0 \\ 1 \text{ si } 0 \le t \le 1. \\ 0 \text{ si } t > 1 \end{cases}$

(b) e(t) = Sin(t).
(c) e est T-périodique, affine par morceaux sur [0,T], avec e(0) = e(T) = 0 et $e(\frac{T}{2}) = 1$.
2°) Ecrire une procédure de tracé des graphes de e et de r.
3°) Ecrire l'équation différentielle vérifiée par r(t) sous la forme : r'(t) = f(t,r(t)).

Résoudre cette équation de façon approchée en utilisant la méthode de Runge-Kutta d'ordre 4 (Cf. le chapitre : « Résolution numérique des équations différentielles »).

Tracer les graphes de e et r.

Comparer à la méthode précédente.

6.4 Intégrales elliptiques

Une intégrale elliptique de deuxième espèce est définie par :

$$e_2(y,k_c,a,b) = \int_0^{+\infty} \frac{a + b \cdot x^2}{\left(1 + x^2\right) \cdot \sqrt{\left(1 + x^2\right) \cdot \left(1 + k_c^2 \cdot x^2\right)}} dx$$

où $y > 0$, a, b, c, $k_c \in R$.

1°) Ecrire e_2 en utilisant le changement de variable $x = tg(\phi)$ et en notant $k^2 = 1 - k_c^2$.

2°) L'intégrale elliptique complète est obtenue en faisant tendre y vers $+\infty$ (ou ϕ vers $\frac{\pi}{2}$). De manière plus générale, l'intégrale elliptique complète est définie par :

$$e(k_c,p,a,b) = \int_0^{+\infty} \frac{a + b \cdot x^2}{\left(1 + p \cdot x^2\right) \cdot \sqrt{\left(1 + x^2\right) \cdot \left(1 + k_c^2 \cdot x^2\right)}} dx$$

Ecrire e en utilisant le changement de variable $x = tg(\varpi)$ et en notant $k^2 = 1 - k_c^2$.

3°) Ecrire un programme permettant de calculer chacune de ces intégrales.

6.5 Calcul d'intégrales doubles par la méthode des éléments finies

Soit Ω un domaine polygonal à cotés parallèles aux axes dans R^2 :

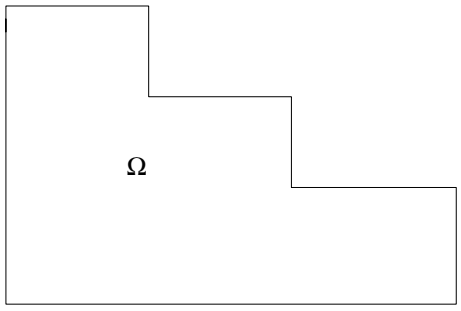


Figure 6.7

et $f : \Omega \rightarrow R$, une fonction continue.

Le but est de calculer, de façon approchée : $I(f) = \iint_{\Omega} f(x,y) dxdy$.

I. On suppose dans cette partie que Ω est le carré unité, noté Ω_1 :

$$\Omega_1 = \{ (x,y) \in R^2 ; 0 \leq x \leq 1, 0 \leq y \leq 1 \}$$

Ce carré est appelé élément de référence.

On note $A_1 = (0,0)$, $A_2 = (1,0)$, $A_3 = (0,1)$, $A_4 = (1,1)$.

E_1 désigne l'ensemble des fonctions polynomiales du type :

$$g(x,y) = a + b\cdot x + c\cdot y + d\cdot x\cdot y$$

E_1 est appelé l'ensemble des fonctions de base.

1°) Déterminer des fonctions g_1, g_2, g_3 et g_4 dans E_1 telles que :

$$g_i(A_j) = \begin{cases} 1 \text{ si } j = i \\ 0 \text{ si } j \neq i \end{cases}$$

2°) Calculer $\iint_\Omega g_j(x,y)\,dxdy$, pour j = 1, 2, 3, 4.

3°) Déterminer g dans E_1 qui interpole f aux sommets de Ω_1, c'est-à-dire telle que :

$$g(A_j) = f(A_j), \text{ pour } j = 1, 2, 3, 4$$

4°) En déduire une approximation $I_1(f)$ de I(f) sur Ω_1.

II. On suppose dans cette partie que Ω est un rectangle à cotés parallèles aux axes, noté Ω_2 :

$$\Omega_2 = \{ (x,y) \in R^2 ; a \leq x \leq b, c \leq y \leq d \}$$

1°) Transformer I(f) en une intégrale sur Ω_1.

2°) En déduire une formule d'approximation de I(f), faisant intervenir les valeurs de f aux sommets de Ω_2.

III. Dans cette partie Ω est un domaine polygonal à cotés parallèles aux axes.

On se donne un maillage de Ω par des rectangles à cotés parallèles aux axes. Si n est le nombre de noeuds de ce maillage, on note S_i chacun des sommets (on pourra les numéroter de haut en bas et de gauche à droite).

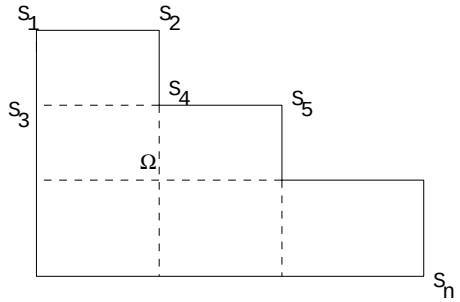


Figure 6.8

Donner une formule d'approximation de I(f) du type :

$$I(f) \cong \sum_{i=1}^{n} \mu_i \cdot f(S_i)$$

où les μ_i sont à préciser.

IV. Reprendre les questions I, II et III avec pour élément de référence le carré unité à 9 noeuds : les 4 sommets, les 4 milieux de chaque coté et le centre du carré, et pour espace de fonctions de base l'espace E des fonctions polynomiales engendré par 1, x, y, $x\cdot y$, x^2, $x^2\cdot y$, $x^2\cdot y^2$, y^2 et $x\cdot y^2$.

7. Programmation Ada

7.1 Spécification du paquetage INTEGRATION_GENERIQUE

```
generic
   with function f(x : in FLOAT) return FLOAT ;
package INTEGRATION_GENERIQUE is
function INTEGRATION_TRAPEZES(a, b : in FLOAT) return FLOAT; -- § 2.4.1
function INTEGRATION_SIMPSON(a, b : in FLOAT) return FLOAT ; -- § 2.4.2
function INTEGRATION_ROMBERG(a,b : in FLOAT) return FLOAT ; -- § 2.4.3
function INTEGRATION_LEGENDRE(a, b : in FLOAT) return FLOAT; -- § 3.6.1
function INTEGRATION_LAGUERRE return FLOAT ; -- § 3.6.3
function INTEGRATION_HERMITE return FLOAT ; -- § 3.6.4
function INTEGRATION_TCHEBYCHEV return FLOAT ; -- § 3.6.2
end INTEGRATION_GENERIQUE ;
```

7.2 Démonstration du paquetage INTEGRATION_GENERIQUE

```
with TEXT_IO, FIO, CRT, COMMON_MATH0, MATH0, MATH2,
     INTEGRATION_GENERIQUE ;
use TEXT_IO, FIO, CRT, COMMON_MATH0, MATH0, MATH2 ;

procedure DEM_INT is
CHOIX : NATURAL range 0..7 ;
ff : FONCTION ;

function f(x : in FLOAT) return FLOAT is
begin
   return EVALUE(ff,x) ;
end ;

package INTEGRATION_F is new INTEGRATION_GENERIQUE(f) ;
use INTEGRATION_F ;

procedure DONNEES_INTEGRATION(a, b : in out FLOAT) is
begin
   PUT_LINE("               Calculs d'integrales definies") ;
   NEW_LINE ;
   GET_LINE(ff,"Donner la fonction a integrer, f(x) = ") ;
   ENTRER_REEL(a,"Donner la valeur minimale de l'abscisse : ") ;
   ENTRER_REEL_BORNE(a,1.0E+12,b,
                     "Donner la valeur maximale de l'abscisse : ") ;
end DONNEES_INTEGRATION ;

procedure DEMO_TRAPEZES is
a, b, T : FLOAT ;
begin
    MODE_AFFICHAGE ;
    DONNEES_INTEGRATION(a,b) ;
    T := INTEGRATION_TRAPEZES(a,b,) ;
    PUT(IMP,"  Integrale de f(x) = ") ;
    PUT(TEXTE_DE_FONCTION(ff)) ;
```

```
    PUT(IMP," sur [") ;
    PUT(IMP,a,4,2,0) ;
    PUT(IMP,",") ;
    PUT(IMP,b,4,2,0) ;
    PUT(IMP,"] = ") ;
    PUT(IMP,T,6,4,0) ; NEW_LINE(IMP) ;
    CLOSE(IMP) ;
    PAUSE ;
end DEMO_TRAPEZES ;

procedure DEMO_SIMPSON is
a, b, S : FLOAT ;
begin
    MODE_AFFICHAGE ;
    DONNEES_INTEGRATION(a,b) ;
    S := INTEGRATION_SIMPSON(a,b) ;
    PUT(IMP,"  Integrale de f(x) = ") ;
    PUT(TEXTE_DE_FONCTION(ff)) ;
    PUT(IMP," sur [") ;
    PUT(IMP,a,4,2,0) ;
    PUT(IMP,",") ;
    PUT(IMP,b,4,2,0) ;
    PUT(IMP,"] = ") ;
    PUT(IMP,S,6,4,0) ;
    NEW_LINE(IMP) ;
    CLOSE(IMP) ;
    PAUSE ;
end DEMO_SIMPSON ;

procedure DEMO_ROMBERG is
a, b, R : FLOAT ;
begin
    MODE_AFFICHAGE ;
    DONNEES_INTEGRATION(a,b) ;
    R := INTEGRATION_ROMBERG(a,b) ;
    PUT(IMP,"  Integrale de f(x) = ") ;
    PUT(TEXTE_DE_FONCTION(ff)) ;
    PUT(IMP," sur [") ;
    PUT(IMP,a,4,2,0) ;
    PUT(IMP,",") ;
    PUT(IMP,b,4,2,0) ;
    PUT(IMP,"] = ") ;
    PUT(IMP,R,6,4,0) ;
    NEW_LINE(IMP) ;
    CLOSE(IMP) ;
    PAUSE ;
end DEMO_ROMBERG ;

procedure DEMO_LEGENDRE is
a, b, R : FLOAT ;
begin
    MODE_AFFICHAGE ;
    DONNEES_INTEGRATION(a,b) ;
```

```
    R := INTEGRATION_LEGENDRE(a,b) ;
    PUT(IMP,"  Integrale de f(x) = ") ;
    PUT(TEXTE_DE_FONCTION(ff)) ;
    PUT(IMP," sur [") ;
    PUT(IMP,a,4,2,0) ;
    PUT(IMP,",") ;
    PUT(IMP,b,4,2,0) ;
    PUT(IMP,"] = ") ;
    PUT(IMP,R,6,4,0) ;
    NEW_LINE(IMP) ;
    CLOSE(IMP) ;
    PAUSE ;
end DEMO_LEGENDRE ;

procedure DEMO_LAGUERRE is
S : FLOAT ;
begin
   MODE_AFFICHAGE ;
   PUT_LINE(" CALCUL DE L'INTEGRALE IMPROPRE DE EXP(-x).f(x) SUR R+") ;
   NEW_LINE ;
   PUT_LINE(" Utilisation du polynome de Laguerre Numero 12") ;
   NEW_LINE ;
   GET_LINE(ff," Entrer la fonction a integrer f(x) = ") ;
   NEW_LINE ;
   S := INTEGRATION_LAGUERRE ;
   PUT(IMP," Valeur approchee de l'integrale, I = ") ;
   PUT(IMP,S,6,4,0) ;
   NEW_LINE(IMP) ;
   CLOSE(IMP) ;
   PAUSE ;
end DEMO_LAGUERRE ;

procedure DEMO_HERMITE is
S : FLOAT ;
begin
   MODE_AFFICHAGE ;
  PUT_LINE(" CALCUL DE L'INTEGRALE IMPROPRE DE EXP(-x**2).f(x) sur R");
   NEW_LINE ;
   PUT_LINE(" Utilisation du polynome d'Hermite Numero 10") ;
   NEW_LINE ;
   GET_LINE(ff," Entrer la fonction a integrer f(x) = ") ;
   NEW_LINE ;
   S := INTEGRATION_HERMITE ;
   PUT(IMP," Valeur approchee de l'integrale, I = ") ;
   PUT(IMP,S,6,4,0) ;
   NEW_LINE(IMP) ;
   CLOSE(IMP) ;
   PAUSE ;
end DEMO_HERMITE ;

procedure DEMO_TCHEBYCHEV is
S : FLOAT ;
begin
```

```
   MODE_AFFICHAGE ;
   PUT_LINE(
  " CALCUL DE L'INTEGRALE IMPROPRE de f(x)/SQRT(1 - x*x) sur [-1,1]") ;
   NEW_LINE ;
   PUT_LINE(" Utilisation du polynome de Tchebychev Numero 30") ;
   NEW_LINE ;
   GET_LINE(ff," Entrer la fonction a integrer f(x) = ") ;
   NEW_LINE ;
   S := INTEGRATION_TCHEBYCHEV ;
   PUT(IMP," Valeur approchee de l'integrale, I = ") ;
   PUT(IMP,S,6,4,0) ;
   NEW_LINE(IMP) ;
   CLOSE(IMP) ;
   PAUSE ;
end DEMO_TCHEBYCHEV ;

procedure MENU_INTEGRATION(CHOIX : out NATURAL) is
begin
    CLRSCR ;
    PUT_LINE("Calcul d'integrales") ;
    PUT_LINE("-0- FIN") ;
    PUT_LINE("-1- Methode des trapezes") ;
    PUT_LINE("-2- Methode de Simpson") ;
    PUT_LINE("-3- Methode de Romberg") ;
    PUT_LINE("-4- Methode de Legendre") ;
    PUT_LINE("-5- Methode de Tchebychev (sur [-1,1])") ;
    PUT_LINE("-6- Methode de Laguerre ( sur [0,+8[)") ;
    PUT_LINE("-7- Methode d'Hermite (Sur R )") ;
    ENTRER_ENTIER_BORNE(0,7,CHOIX,"Votre choix : ") ;
end MENU_INTEGRATION ;

begin
    loop
      MENU_INTEGRATION(CHOIX) ;
      case CHOIX is
        when 0 => exit ;
        when 1 => DEMO_TRAPEZES ;
        when 2 => DEMO_SIMPSON ;
        when 3 => DEMO_ROMBERG ;
        when 4 => DEMO_LEGENDRE ;
        when 5 => DEMO_TCHEBYCHEV ;
        when 6 => DEMO_LAGUERRE ;
        when 7 => DEMO_HERMITE ;
      end case ;
    end loop ;
end DEM_INT ;
```

7.3 Spécifications des paquetages COMMON_FOURIER et FOURIER_GENERIQUE

```
with NOMBRES_COMPLEXES ;
use NOMBRES_COMPLEXES ;
package COMMON_FOURIER is
```

```
P_MAX : constant INTEGER := 8 ;
N_MAX : constant INTEGER := 2**P_MAX ;
subtype INDICE is INTEGER range 0..N_MAX - 1 ;
type VECTEUR_COMPLEXE is array(INDICE range <>) of COMPLEXE ;
end COMMON_FOURIER ;

with COMMON_MATRIX, COMMON_FOURIER ;
use COMMON_MATRIX, COMMON_FOURIER ;
generic
   with function f(x : in FLOAT) return FLOAT ;
package FOURIER_GENERIQUE is
procedure ECHANTILLONNAGE(tMax : in FLOAT ;
             x : out VECTEUR_COMPLEXE ; D : out FLOAT) ; -- § 5.1
-- x = (x(0),...,x(n-1)), avec x(j) = f(j*D) et D = tMax/n
function SERIE_DE_FOURIER(a, b : in VECTEUR ; x : in FLOAT)
             return FLOAT ; -- § 5.5
procedure TRACE_UNE_SERIE_DE_FOURIER(tMax : in FLOAT ;
        a, b : in VECTEUR ; NB_POINTS : in INTEGER := 300) ; -- § 5.5
procedure FFT(x : in VECTEUR_COMPLEXE ;
             y : out VECTEUR_COMPLEXE) ; -- § 5.4
procedure CALCUL_COEFFICIENTS_DE_FOURIER(y : in VECTEUR_COMPLEXE ;
             a, b : out VECTEUR) ; -- § 5.5
function ENERGIE(x : in VECTEUR_COMPLEXE) return FLOAT ; -- § 5.3
end FOURIER_GENERIQUE ;
```

7.4 Démonstration du paquetage FOURIER_GENERIQUE

```
with TEXT_IO, IIO, FIO, CRT, COMMON_MATH0, MATH0, MATH1, GRAPH, MATH2,
     MATH3, CURVE, NOMBRES_COMPLEXES, COMMON_FOURIER,
     FOURIER_GENERIQUE, COMMON_MATRIX, MATRIX ;
use TEXT_IO, IIO, FIO, CRT, COMMON_MATH0, MATH0, MATH1, GRAPH, MATH2,
     MATH3, CURVE, NOMBRES_COMPLEXES, COMMON_FOURIER,
     COMMON_MATRIX, MATRIX ;

procedure DEM_FFT is

MAX_FONCTIONS : constant INTEGER := 10 ;
subtype INDICE_FONCTION is INTEGER range 0..MAX_FONCTIONS ;
type TABLEAU_DE_FONCTIONS is array(INDICE_FONCTION range <>)
        of FONCTION ;

NBR_INT, p, n, m : INTEGER ;
procedure AFFICHER_FFT(D : in FLOAT ; y : in VECTEUR_COMPLEXE) is
--               ^      ^
-- Affichage des y(k) = f(k/tMax) = D*y(k), ou des c(k) = y(k)/n, ou :
-- n = 2^p, k = -n/2, ..., O, ..., n/2 - 1.

m : INTEGER ;
begin
    m := y'LENGTH/2 ;
    for i in m..y'LENGTH - 1
    loop
        PUT(IMP,"x(") ; PUT(IMP,i - n,3) ; PUT(IMP,") = ") ;
```

```
        PUT(D*y(i)) ; NEW_LINE ;
        if (i + 1) mod 20 = 0 then
           PAUSE ;
        end if ;
    end loop ;
    for i in 0..m - 1
    loop
        PUT(IMP,"x(") ; PUT(i,3) ; PUT(") = ") ;
        PUT(D*y(i)) ;
        NEW_LINE ;
        if (i + 1) mod 20 = 0 then
           PAUSE ;
        end if ;
    end loop ;
end AFFICHER_FFT ;

begin
-- DONNEES POUR LE CALCUL D'UNE F.F.T. ET DES COEFFICIENTS DE FOURIER
-- La fonction f(t) est defini comme une fonction continue par morceaux
-- sur un intervalle [0,tn], avec : 0 < t1 < ... < tn et :
-- f(t) = f(k-1)(t), si t dans [t(k-1),t(k)] (k = 1, ..., n),
-- les fonctions f(k) etant continues pour k = 0, ..., n - 1.
    CLRSCR ;
    PUT_LINE(
 "Calculs de la FFT et des coefficients de Fourier d'une fonction f") ;
    NEW_LINE ;
    PUT_LINE("La fonction est definie sur [0,tMax], de periode tMax,");
    PUT_LINE("par des fonctions continues fk sur [tk,t(k+1)], avec") ;
    PUT_LINE("       0 < t1 < ... < tn = tMax") ;
    NEW_LINE ;
    ENTRER_ENTIER_BORNE(1,MAX_FONCTIONS,NBR_INT,
                        "Nombre d'intervalles : ") ;
    NEW_LINE ;
PUT_LINE("Le nombre de points de l'echantillonnage de f est n = 2^p.");
    ENTRER_ENTIER_BORNE(1,p_Max,p,"p = ") ;
    n := 2**p ;
    m := n/2 ;
    declare
       ff : TABLEAU_DE_FONCTIONS(0..NBR_INT - 1) ;
       t : VECTEUR(0..NBR_INT) ;
       x, y : VECTEUR_COMPLEXE(0..n - 1) ;
       a : VECTEUR(0..m- 1) ;
       b : VECTEUR(1..m - 1) ;
       C : COURBE(n) ;
       PAS, D, xOrigine, yOrigine, xUnite, yUnite,
       xFenMin, xFenMax, yFenMin, yFenMax : FLOAT ;

       function f(x : in FLOAT) return FLOAT is
       k : INTEGER := 0 ;
       begin
           loop
             k := k + 1 ;
             exit when (t(k) >= x) or (k = NBR_INT) ;
```

```
         end loop ; -- x est dans [t(k-1),t(k)]
         return EVALUE(ff(k - 1),x) ;
      end f ;

      package FOURIER_F is new FOURIER_GENERIQUE(f) ;
      use FOURIER_F ;

   begin
      t(0) := 0.0 ;
      PUT_LINE("Entree des bornes des intervalles : t1, t2, ..., tn");
      for k in 1..NBR_INT
      loop
        ENTRER_REEL_BORNE(t(k-1),1.0E+32,t(k),"t = ") ;
      end loop ;
      NEW_LINE ;
      PUT_LINE("Definition de la fonction f = (f0,...,f(n-1))") ;
      for k in 0..NBR_INT - 1
      loop
         GET_LINE(ff(k),"f(x) = ") ;
      end loop ;
      NEW_LINE ;
      MODE_AFFICHAGE ;
      PUT_LINE(IMP,"Calcul de l'echantillonnage : Vecteur x :") ;
      ECHANTILLONNAGE(t(NBR_INT),x, D) ;
      for i in x'range
      loop
        PUT(IMP,"x") ; PUT(IMP,i,3) ; PUT(IMP," = ") ;
        PUT(IMP,x(i)) ;
        NEW_LINE(IMP) ;
        if (i + 1) mod 20 = 0 then
           PAUSE ;
        end if ;
      end loop ;
      NEW_LINE(IMP) ;
      PUT_LINE(IMP,"Calcul de la transformee de Fourier discrete") ;
      NEW_LINE(IMP) ;
      FFT(x,y) ;
      PUT_LINE(IMP,"VERIFICATION : FORMULE DE PLANCHEREL") ;
      PUT(IMP,"Ex = ") ; PUT(IMP,ENERGIE(x),6,6) ;
      PUT(IMP," ;   Ey/n = ") ; PUT(IMP,ENERGIE(y)/FLOAT(n),6,6) ;
      PAUSE ;
      CLRSCR ;
  PUT_LINE(Imp,"Calcul des coefficients de Fourier trigonometriques") ;
      CALCUL_COEFFICIENTS_DE_FOURIER(y,a,b) ;
      CLRSCR ;
      PUT_LINE(IMP,"Transformee de Fourier ") ;
      AFFICHER_FFT(D,y) ;
      PAUSE ;
      CLRSCR ;
      PUT_LINE(IMP,"Coefficients de Fourier complexes") ;
      AFFICHER_FFT(1.0/FLOAT(n),y) ;
      PAUSE ;
      CLRSCR ;
```

```
      PUT_LINE(IMP,"Coefficients de Fourier trigonometriques") ;
      PUT_LINE("Coefficients a(k), pour k = 0, ..., n/2 - 1") ;
      PUT_LINE(a) ;
      NEW_LINE ;
      PUT_LINE("Coefficients b(k), pour k = 1, ..., n/2 - 1") ;
      PUT_LINE(b) ;
      NEW_LINE ;
      PAUSE ;
      CLRSCR ;
      C.uMin := 0.0 ;
      C.uMax := t(NBR_INT) ;
      PAS := (C.uMax - C.uMin)/FLOAT(C.n) ;
      C.vMIN := 1.0E+37 ;
      C.vMAX := -C.vMIN ;
      C.u(0) := C.uMin ;
      C.v(0) := f(C.u(0)) ;
      for i in 1..C.n
      loop
         C.u(i) := C.u(i-1) + PAS ;
         C.v(i) := f(C.u(i)) ;
         C.vMIN := MIN(C.vMIN,C.v(i-1)) ;
         C.vMAX := MAX(C.vMAX,C.v(i-1)) ;
      end loop ;
      TRACE_UNE_COURBE(C) ;
      TITRE("Signal initial") ;
      PAUSE_GRAPHIQUE ;
      CLEAR_SCREEN ;
      TITRE("Reconstitution par les series de Fourier") ;
      TRACE_UNE_SERIE_DE_FOURIER(t(NBR_INT),a,b,n) ;
      XY_AXES(xOrigine,yOrigine,xUnite,yUnite,TRUE) ;
      PAUSE_GRAPHIQUE ;
   end ;
   MODE_TEXTE ;
end DEM_FFT ;
```

CHAPITRE 7

Résolution numérique des équations différentielles

1. Introduction. Origines des problèmes d'équations différentielles

Les équations différentielles interviennent naturellement dans l'étude des *problèmes d'évolution*, que l'on peut formuler de la façon suivante :

- l'état d'un système (mécanique, électrique, économique, ...) est complètement décrit par p degrés de liberté y_1, ..., y_p qui sont fonctions du temps t ;
- les y_i sont tous connus à un instant donné t ;
- la loi d'évolution du système est connue, c'est-à-dire qu'à chaque instant $t > t_0$, on connaît une relation entre les dérivées $y_i^{(k)}(t)$ et les valeurs des paramètres $y_j(t)$.

Dans le cas des systèmes d'ordre 1, les y_i sont alors solutions d'un *système différentiel avec conditions initiales :*

$$\begin{cases} f_i\left(t, y_1(t), \ldots, y_p(t), y_1'(t), \ldots, y_p'(t)\right) = 0 \quad (t \in I) \\ y_i(t_0) \text{ sont donnés pour } i = 1, \ldots, p \end{cases}$$

où I est un intervalle réel contenant t_0.

Exemples — En *mécanique*, l'exemple classique est donné par la loi de Newton : $f(t,x(t),y(t),z(t)) = m \cdot \gamma(t)$, où $\gamma(t) = (x''(t),y''(t),z''(t))$ et le point de masse m et de coordonnées (x,y,z) est soumis à un instant t à une force f(t,x,y,z).

En *électricité*, les équations différentielles interviennent, entre autres, dans l'étude des phénomènes transitoires.

Considérons, par exemple, une résistance métallique de coefficient de température α, de capacité calorique C et telle que la résistance soit reliée à la température par une loi du type : $R = R_0 \cdot (1 + \alpha \cdot (T - T_0))$, où R_0 et T_0 sont les

résistance et température initiales. On suppose, de plus, qu'elle transmet au milieu extérieur, à la température T_0, la puissance thermique

$$P_T = G\cdot(T - T_0).$$

Alors $\theta = T - T_0$ est solution de l'équation différentielle :

$$\begin{cases} \theta'(t) = -\dfrac{G}{C}\cdot\theta(t) + \dfrac{u^2}{C\cdot R_0\cdot(1+\alpha\cdot\theta(t))} \\ \theta(0) = 0 \end{cases}$$

En *théorie mathématique des populations* (Cf. A. Hillion), on peut montrer que l'évolution de la taille x(t) d'une population simple, en fonction du temps, suit une loi du type :

$$\begin{cases} x'(t) = r(t, x(t))\cdot x(t) \\ x(0) \text{ donné} \end{cases}$$

où r(t,x) représente un taux d'accroissement.

Quand le taux d'accroissement est constant, la solution est $x(t) = x(0)\cdot e^{r\cdot t}$, c'est-à-dire qu'on a une croissance exponentielle de la population. Mais dans la réalité une telle situation ne se produit pas et l'équation différentielle obtenue ne peut se résoudre explicitement.

Dans le cas de populations en interaction, on aboutit à un système d'équations différentielles. Un exemple classique étant le modèle proie-prédateur qui peut se décrire de la façon suivante : supposons qu'à un instant initial $t_0 = 0$, il y ait sur un territoire x(0) prédateurs et y(0) proies. Le problème est alors de connaître l'évolution de ces populations (si, par exemple, il y a trop de prédateurs alors les proies disparaîtront, ce qui entraînera aussi la disparition des prédateurs faute de nourriture).

On peut alors montrer que x et y sont solutions d'un système différentiel du type :

$$\begin{cases} x'(t) = r_1(t,x)\cdot x(t) - b_1(t,x)\cdot x(t)\cdot y(t) \\ y'(t) = -r_2(t,x)\cdot y(t) + b_2(t,x)\cdot x(t)\cdot y(t) \end{cases}$$

La résolution de ce système permet alors de prévoir la situation à chaque instant.

2. Problème de Cauchy

2.1 Position du problème

Soient $\begin{array}{rccc} f_j: & [a,b]\times R^p & \to & R \\ & (t, y_1, ..., y_p) & \mapsto & f_j(t, y_1, ..., y_p) \end{array}$ des fonctions continues, avec j = 1, ..., p.

Le *problème de Cauchy* associé aux f_j, consiste à trouver des fonctions y_1, y_2, ..., y_p de classe C^1 (i. e. continuement dérivable) de [a,b] dans R telles que :

$$(1)\quad \begin{cases} y_1'(t) = f_1\left(t, y_1(t), \ldots, y_p(t)\right) \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad \left(t \in [a,b]\right) \\ y_p'(t) = f_p\left(t, y_1(t), \ldots, y_p(t)\right) \\ \\ y_j(a) = y_{a,j} \text{ donnés dans R (conditions initiales)} \end{cases}$$

Ce qui peut se noter de façon plus compacte :

$$(2)\quad \begin{cases} y'(t) = f\left(t, y(t)\right) \;\; t \in [a,b] \\ y(a) = y_a \text{ donné dans } R^p \end{cases}$$

Remarque 1 — Equations différentielles d'ordre supérieur à 1.

Une équation différentielle d'ordre $q \geq 1$, avec conditions initiales, de la forme :

$$\begin{cases} y_1^{(q)}(t) = g_1\left(t, y_1(t), \ldots, y_1^{(q-1)}(t), \ldots, y_p(t), \ldots, y_p^{(q-1)}(t)\right) \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad \left(t \in [a,b]\right) \\ y_p^{(q)}(t) = g_p\left(t, y_1(t), \ldots, y_1^{(q-1)}(t), \ldots, y_p(t), \ldots, y_p^{(q-1)}(t)\right) \\ \\ y_j^{(k)}(a) \text{ donnés dans R pour } j = 1, \ldots, p \text{ et } k = 0, \ldots, q-1 \end{cases}$$

où $g : [a,b] \times R^{p \cdot q} \to R^p$ est continue, peut se ramener à un système différentiel d'ordre un en faisant le changement de fonctions inconnues :

$$\begin{cases} z_1 = y_1 \\ . \\ . \\ z_q = y_1^{(q-1)} \\ z_{q+1} = y_2 \\ . \\ . \\ z_{p \cdot q} = y_p^{(q-1)} \end{cases}$$

ce qui donne le système de $p \cdot q$ équations d'ordre 1 :

$$\begin{cases} z'(t) = f\left(t, z(t)\right) \;\; \left(t \in [a,b]\right) \\ z(a) = z_a \end{cases}$$

où :

$$f(t,z)=\begin{pmatrix} z_2 \\ . \\ . \\ z_q \\ g_1(t,z_1,\ldots,z_{p\cdot q}) \\ z_{q+2} \\ . \\ . \\ z_{p\cdot q} \\ g_p(t,z_1,\ldots,z_{p\cdot q}) \end{pmatrix}$$

Remarque 2 — On peut aussi considérer les *problèmes aux limites*, où les conditions initiales ne sont pas nécessairement données en a. Les exemples classiques sont :

- le *problème de Dirichlet* : $\begin{cases} y''(t)=f(t,y(t),y'(t)) \text{ sur } [a,b] \\ y(a) \text{ et } y(b) \text{ donnés} \end{cases}$
- le *problème de Neumann* : $\begin{cases} y''(t)=f(t,y(t),y'(t)) \text{ sur } [a,b] \\ y'(a) \text{ et } y'(b) \text{ donnés} \end{cases}$
- le *problème de Dirichlet-Neumann* : $\begin{cases} y''(t)=f(t,y(t),y'(t)) \text{ sur } [a,b] \\ y(a) \text{ et } y'(b) \text{ donnés} \end{cases}$

Remarque 3 — On peut aussi considérer des équations différentielles sous forme « non résolue » : $\varphi(t,y(t),y'(t),\ldots,y^{(q)}(t))=0$. Mais dans ce cours on s'intéressera surtout au cas résolu, le cas général étant beaucoup plus difficile.

2.2 Problème de l'existence et l'unicité de solutions

Définition **:** On dit que f est *Lipschitzienne* en y, s'il existe une constante $L \geq 0$, telle que :

$$\forall t \in [a,b],\ \forall y,\ z \in R^p,\ \|f(t,y)-f(t,z)\| \leq L \cdot \|y-z\|$$

Remarque 1 — Si L = 0, dans la définition ci-dessus, alors f est constante. On supposera donc L > 0.

Remarque 2 — *Critère pour vérifier qu'une application est Lipschitzienne en y*

Avec le théorème des accroissements finis, on voit que si f est continûment dérivable par rapport à y et s'il existe $L \geq 0$ tel que :

$$\left\| \frac{\partial f}{\partial y_j}(t,y) \right\| \leq L \quad (t \in [a,b],\ y \in R^p,\ j = 1, ..., p)$$

alors f est lipschitzienne en y.

Théorème (Cauchy-Lipschitz) : Si f est continue et L-Lipschitzienne en y, alors le problème de Cauchy (1) admet une unique solution y de classe C^1.

Démonstration — L'idée est de se ramener à un problème de point fixe que l'on résout par la méthode des approximations successives. Le problème (1) équivaut à trouver y dans $E = C^0([a,b],R^p)$ (espace des fonctions continues de [a,b] dans R^p) solution de :

$$(2) \quad y(t) = \int_a^t f(x, y(x))dx + y_a$$

En notant Φ l'application de E dans E définie par :

$$(\forall y \in E,\ z = \Phi(y)) \Leftrightarrow \left(z(t) = \int_a^t f(x, y(x)dx) + y_a \right)$$

alors (2) équivaut à : $y = \Phi(y)$. C'est-à-dire que y est un point fixe de Φ.

On veut alors écrire y comme limite de la suite d'approximations successives (y_k) définie par :

$$\begin{cases} y_0 \in E \\ y_{k+1} = \Phi(y_k) \quad (k \geq 0) \end{cases}$$

Comme E est un espace de Banach, on sait qu'une telle suite converge vers l'unique point fixe de Φ, y, si Φ ou l'une de ses itérées Φ^n est contractante (théorème du point fixe de Banach).

Du fait que Φ est L-lipschitzienne en y, on déduit que pour tout $n > 0$ et tous y, z dans E, on a :

$$\left\| \Phi^n(y) - \Phi^n(z) \right\| \leq \frac{(L \cdot (b-a))^n}{n!} \cdot \|y - z\|$$

et tenant compte de $\lim_{n \to +\infty} \frac{(L \cdot (b-a))^n}{n!} = 0$, on a le résultat.

Exemple 1 — Considérons le problème, sur R :

$$\begin{cases} y'(t) = y(t) \quad (t \in R) \\ y(0) = 1 \end{cases}$$

La méthode du point fixe consiste à étudier la suite de fonctions définie par :

$$\begin{cases} y_0(t) = 1 \\ y_{n+1}(t) = \int_0^t y_n(x)dx + 1 \quad (n \geq 0) \end{cases}$$

Et, par récurrence, on a : $y_n(t) = \sum_{k=0}^{n} \frac{t^k}{k!}$. On retrouve ainsi une définition de l'exponentielle.

Exemple 2 — Considérons le problème de Cauchy :

$$\begin{cases} y'(t) = y(t)^{\alpha} \quad (t \in [0,1]) \\ y(0) = 0 \end{cases}$$

où $\alpha \in R$.

Pour $\alpha = 1$, il admet une unique solution donnée par $y = 0$, mais pour $\alpha \in]0,1[$, il admet une infinité de solutions données par :

$$y(t) = \begin{cases} 0 \text{ si } t \in [0,a] \\ \left((1-\alpha)\cdot(t-a)\right)^{\frac{1}{1-\alpha}} \text{ si } t \in [a,1] \end{cases}$$

où $0 < a < 1$.

On déduit donc que l'application $f : (t,y) \to y^{\alpha}$ ne vérifie pas la condition de Lipschitz.

Corollaire (Equations différentielles linéaires) : Si $f(t,y) = A(t)\cdot y + b(t)$, avec $A : [a,b] \to End(R^p)$ et $b : [a,b] \to R^p$ continues, alors le problème de Cauchy linéaire :

$$\begin{cases} y'(t) = A(t)\cdot y(t) + b(t) \quad t \in [a,b] \\ y(a) = y_a \end{cases}$$

admet une unique solution.

Remarque — Le corollaire est encore valable en remplaçant [a,b] par un intervalle quelconque.

2.3 Approximation de la solution d'un problème de Cauchy par discrètisation

On suppose que f est L-Lipschitzienne par rapport à y, de sorte que le problème de Cauchy associé admet une unique solution.

Pour trouver une approximation de la solution y, on utilisera une *méthode de discrètisation* dont le principe est le suivant : on se donne une subdivision de l'intervalle [a,b] $a = t_0 < t_1 < ... < t_n = b$ et pour tout k = 1, ..., n, on cherche une valeur approchée y_k de la valeur exacte $y(t_k)$.

La quantité $h = \text{Max}\{ t_{i+1} - t_i ; 0 \le i \le n-1 \}$ est le *pas d'intégration*.

Quand h sera très petit, on espère que la fonction affine par morceaux définie par $\varphi(t_k) = y_k$ pour k = 0, 1, ..., n nous donnera une bonne approximation de y.

En général, on utilisera une subdivision à pas constant, soit :

$$t_k = a + k\cdot\frac{b-a}{n} \text{ , pour } k = 0, 1, ..., n.$$

L'idée sera alors de calculer les y_k par des formules de récurrence.
On distingue deux types de schémas d'intégration :

- les *schémas à un pas* (ou à pas séparés) : y_k est fonction de y_{k-1} ;
- les *schémas à plusieurs pas* (ou à pas liés) : y_k est fonction de y_{k-i} où i = 1, 2, ..., p et p est le nombre de pas.

3. Généralités sur les méthodes d'intégration à un pas

3.1 Définitions

Une méthode à un pas est définie par les formules de récurrence :

$$(1)\quad \begin{cases} y_0 \text{ donné} \\ y_{k+1} = y_k + h \cdot \Phi(t_k, y_k, h), \text{ pour } k = 0, 1, \ldots, n-1 \end{cases}$$

où Φ est une fonction continue qui ne dépend que de f.

Définition : L'erreur de consistance en t_k de la méthode est la quantité :

$$\tau_k = y(t_{k+1}) - y(t_k) - h\cdot\Phi(t_k,y(t_k),h)$$

Et on dira que la méthode est consistante avec l'équation différentielle si $\Phi(t,y,0) = f(t,y)$ pour tous t et y.

Remarque — L'erreur de consistance permet de mesurer de combien la solution exacte est éloignée du schéma d'intégration.

Définition : L'erreur de discrètisation en t_k de la méthode est la quantité :

$$e_k = \left\| y(t_k) - y_k \right\|$$

3.2 Lien entre l'erreur de consistance et l'erreur de discrètisation

3.2.1 Hypothèses

On fait les hypothèses suivantes :

- p = 1 (pour simplifier) ;
- f est lipschitzienne en y ;
- Φ est lipschitzienne en y ;
- la méthode est consistante avec l'équation différentielle ;
- f et Φ sont de classe C^1.

3.2.2 Première évaluation de l'erreur de consistance

L'erreur de consistance en t_k est :

$$\tau_k(h) = y(t_{k+1}) - y(t_k) - h\cdot\Phi(t_k,y(t_k),h)$$

En faisant un développement limité au voisinage de 0, on a :

$$\Phi(t_k,y(t_k),h) = \Phi(t_k,y(t_k),0) + h \cdot \frac{\partial\Phi}{\partial h}(t_k, y(t_k), \xi_k) \qquad (\xi_k \text{ entre 0 et h})$$

Avec (iv), on déduit alors que :

$$\tau_k(h) = y(t_{k+1}) - y(t_k) - h\cdot f(t_k,y(t_k)) - h^2 \cdot \frac{\partial\Phi}{\partial h}(t_k, y(t_k), \xi_k)$$

avec f(t,y(t)) = y'(t) et f de classe C^1, on déduit que y est de classe C^2, puis avec la formule de Taylor, on a :

$$\tau_k(h) = h^2 \cdot \left\{ \frac{1}{2} \cdot y''(\nu_k) - \frac{\partial\Phi}{\partial h}(t_k, y(t_k), \xi_k) \right\} = 0(h^2)$$

3.2.3 Hypothèses supplémentaires

Si on suppose que f et Φ sont de classe C^p, on peut faire des développements limités à l'ordre p.

3.2.4 Deuxième évaluation de l'erreur de consistance

Un développement limité en h = 0, nous donne :

$$\Phi\left(t_k, y(t_k), h\right) = \Phi\left(t_k, y(t_k), 0\right)$$

$$+ h \cdot \frac{\partial \Phi}{\partial h}\left(t_k, y(t_k), 0\right) + \ldots + \frac{h^{p-1}}{(p-1)!} \cdot \frac{\partial^{p-1} \Phi}{\partial h^{p-1}}\left(t_k, y(t_k), 0\right) + \frac{h^p}{p!} \cdot \frac{\partial^p \Phi}{\partial h^p}\left(t_k, y(t_k), \xi_k\right)$$

Pour calculer un développement limité de y en t_k, il nous faut d'abord calculer les dérivées successives de y en fonction de f.

Au départ, on a : y'(t) = f(t,y(t)). Donc :

$$y''(t) = \frac{\partial f}{\partial t}\left(t, y(t)\right) + \frac{\partial f}{\partial y}\left(t, y(t)\right) \cdot y'(t) = \frac{\partial f}{\partial t}\left(t, y(t)\right) + \frac{\partial f}{\partial y}\left(t, y(t)\right) \cdot f\left(t, y(t)\right)$$

Et par récurrence, on a :

$$y^{(k)}(t) = f^{(k-1)}(t, y(t)) \quad (k = 1, \ldots, p+1)$$

où les $f^{(j)}$ sont définis par :

$$\begin{cases} f^{(0)}(t,y) = f(t,y) \\ f^{(k)}(t,y) = \dfrac{\partial f^{(k-1)}}{\partial t}(t,y) + \dfrac{\partial f^{(k-1)}}{\partial y}(t,y) \cdot f(t,y) \quad (k = 1, \ldots, p+1) \end{cases}$$

Un développement de Taylor, au voisinage de t_k, va alors nous donner :

$$y(t_{k+1}) = y(t_k) + h \cdot f^{(0)}(t_k, y(t_k)) + \ldots + \frac{h^p}{p!} \cdot f^{(p-1)}\left(t_k, y(t_k)\right) + O\left(h^{p+1}\right)$$

Ce qui donne alors, pour $\tau_k(h)$:

$$\tau_k(h) = h \cdot \left\{ f^{(0)}\left(t_k, y(t_k)\right) - \Phi\left(t_k, y(t_k), 0\right) \right\}$$

$$+ \ldots + \frac{h^p}{p!} \cdot \left\{ f^{(p-1)}\left(t_k, y(t_k)\right) - p \cdot \frac{\partial^{p-1} \Phi}{\partial h^{p-1}}\left(t_k, y(t_k), 0\right) \right\} + O\left(h^{p+1}\right)$$

On déduit alors le :

Théorème : Avec les hypothèses ci-dessus, si de plus, on a :

$$\begin{cases} 2 \cdot \dfrac{\partial \Phi}{\partial h}(t,y,0) = f^{(1)}(t,y) \\ p \cdot \dfrac{\partial^{p-1} \Phi}{\partial h^{p-1}}(t,y,0) = f^{(p-1)}(t,y) \end{cases}$$

alors, $\tau_k(h) = O(h^{p+1})$.

Remarque — La réciproque est aussi vraie. (Cf. Crouzeix et Mignot, p. 95 et 96, Théorème (5.6))

3.2.5 Majoration de l'erreur de discrétisation

On note :

$$E_k = y(t_k) - y_k \text{ et } e_k = \|E_k\|$$

Avec :

$$\begin{cases} y_{k+1} = y_k + h \cdot \Phi(t_k, y_k, h) \\ y(t_{k+1}) = y(t_k) + h \cdot \Phi(t_k, y(t_k), h) + \tau_k(h) \end{cases}$$

on déduit que :

$$E_{k+1} = E_k + h \cdot \{\Phi(t_k,y_k,h) - \Phi(t_k,y(t_k),h)\} - \tau_k(h)$$

En supposant les hypothèses du théorème (2.2.4) vérifiées, on a :

$$|\tau_k(h)| \leq C_1 \cdot h^{p+1}$$

et avec Φ C_2-Lipschitzienne, on déduit que :

$$e_{k+1} < (1 + h \cdot C_2) \cdot e_k + C_1 \cdot h^{p+1}$$

Mais on a le :

Lemme (Gronwall) : Si (e_k) est une suite de réels positifs telle que :

$$e_{k+1} < (1 + \alpha) \cdot e_k + \beta \quad (\alpha, \beta > 0)$$

Alors : $e_k \leq (e_0 + \frac{\beta}{\alpha}) \cdot e^{k \cdot \alpha}$, pour tout $k > 0$.

Démonstration — Par récurrence, on déduit que :

$$e_k \leq (1 + \alpha)^k \cdot e_0 + \beta \cdot (1 + (1 + \alpha) + ... + (1 + \alpha)^{k-1})$$

soit :

$$e_k \leq (1 + \alpha)^k \cdot e_0 + \beta \cdot \frac{(1+\alpha)^k - 1}{\alpha}$$

et :

$$e_k \leq (e_0 + \frac{\beta}{\alpha}) \cdot (1 + \alpha)^k$$

Comme $\alpha = h \cdot C_2$ est petit, on majore $(1 + \alpha)$ par e^{α}, ce qui donne le résultat.

Théorème **:** Si l'erreur de consistance est un $O(h^{p+1})$, pour tout k, alors l'erreur de discrétisation est un $O(h^p)$.

Démonstration — Avec ce qui précède et la condition initiale $e_0 = 0$, on déduit que

$$e_k \leq C \cdot e^{k \cdot h \cdot C_2} \cdot h^p$$

et avec $k \cdot h \leq (b - a)$, on déduit que :

$$e_k \leq C \cdot e^{(b-a) \cdot C_2} \cdot h^p$$

C'est-à-dire que $e_k = O(h^p)$.

Remarques — (i) En général, on a seulement $e_0 \cong 0$, car y_a est seulement connu de façon approchée. Et cela peut parfois poser des problèmes.

(ii) Si on fixe le pas h et si la longueur de l'intervalle tend vers l'infini, alors le majorant de l'erreur ci-dessus va croître de façon exponentielle avec (b – a).

Définition : Avec les notations et conditions du théorème ci-dessus, on dit que la méthode est d'ordre p.

Exemple — La méthode d'Euler est définie par $\Phi(x,y) = y$ et si on considère l'équation :

$$\begin{cases} y' = y \text{ sur } [0,b] \\ y(0) = 1 \end{cases}$$

le schéma d'Euler s'écrit :

$$\begin{cases} y_0 = 1 \\ y_{k+1} = (1+h)\cdot y_k \end{cases}$$

donc $y_k = (1+h)^k = \left(1+\frac{b}{n}\right)^k$.

Et pour k = n, on a :

$$y_n = \left(1+\frac{b}{n}\right)^n \underset{n\to+\infty}{\to} e^b = y(b)$$

L'erreur de discrétisation en b est alors :

$$e_n = e^b - \left(1+\frac{b}{n}\right)^n = e^{n\cdot h}\cdot\left\{1-\left(\frac{1+h}{e^h}\right)^n\right\} \underset{n\to+\infty}{\to} +\infty$$

pour un pas h fixé.

3.3 Les méthodes de Runge-Kutta

On suppose ici que f est de classe C^2 et L-lipschitzienne en y.

3.3.1 Principe des méthodes de Runge-Kutta

A partir de l'équation différentielle y'(t) = f(t,y(t)) sur [a,b], on peut écrire que :

$$y(t_{k+1}) = y(t_k) + \int_{t_k}^{t_{k+1}} f(x, y(x))dx \qquad (0 \le k \le n-1)$$

Et l'idée est alors de calculer une valeur approchée de y_{k+1} en fonction de y_k à partir d'une formule d'intégration approchée.

3.3.2 Exemples classiques

(i) Méthode des rectangles à gauche — $\int_a^b g(t)dt \cong (b-a)\cdot g(a)$.

On obtient alors le schéma :

$$y_{k+1} = y_k + h\cdot f(t_k,y_k) \text{ pour } k = 0, 1, ..., n-1$$

C'est la *méthode d'Euler* qui est d'ordre 1.

(ii) Méthode du point milieu — $\int_a^b g(t)dt \cong (b-a)\cdot g\left(\frac{a+b}{2}\right)$.

On obtient alors le schéma :

$$y_{k+1} = y_k + h\cdot f\left(t_k + \frac{h}{2}, y_k + \frac{h}{2}\cdot f(t_k,y_k)\right)$$

C'est la *méthode d'Euler-Cauchy* qui est d'ordre 2.

(iii) Méthode des trapèzes — $\int_a^b g(t)dt \cong (b - a)\cdot\frac{g(a)+g(b)}{2}$.

On obtient alors le schéma ;

$$y_{k+1} = y_k + \frac{h}{2}\cdot\left\{f(t_k, y_k) + f(t_k + h, y_k + h\cdot f(t_k, y_k))\right\}$$

C'est la *méthode de Heun* qui est d'ordre 2.

(iv) Méthode de Simpson — $\int_a^b g(t)dt \cong \frac{b-a}{6}\cdot\left\{g(a) + 4\cdot g\left(\frac{a+b}{2}\right) + g(b)\right\}$.

On peut alors écrire :

$$y(t_{k+1}) \cong y(t_k) +$$

$$\frac{h}{6}\cdot\left\{f(t_k, y(t_k)) + 2\cdot f\left(t_k + \frac{h}{2}, y\left(t_k + \frac{h}{2}\right)\right) + 2\cdot f\left(t_k + \frac{h}{2}, y\left(t_k + \frac{h}{2}\right)\right) + f(t_k + h, y(t_k + h))\right\}$$

Calcul approché de $f(t_k,y(t_k))$ — On a :

$$f(t_k,y(t_k)) \cong K_1 = f(t_k,y_k)$$

Premier calcul approché de $f\left(t_k + \frac{h}{2}, y\left(t_k + \frac{h}{2}\right)\right)$.

On a : $y(t_k + \frac{h}{2}) = y(t_k) + \int_{t_k}^{t_k+\frac{h}{2}} f(x, y(x))dx$ et la première idée est de calculer cette intégrale avec la méthode des rectangles à gauche, soit :

$$\int_{t_k}^{t_k+\frac{h}{2}} f(x, y(x))dx \cong \frac{h}{2}\cdot f(t_k, y(t_k)) \cong \frac{h}{2}\cdot f(t_k, y_k)$$

ce qui donne :

$$(1) \quad y\left(t_k + \frac{h}{2}\right) \cong y_k + \frac{h}{2}\cdot K_1$$

et

$$f\left(t_k + \frac{h}{2}, y\left(t_k + \frac{h}{2}\right)\right) \cong K_2 = f\left(t_k + \frac{h}{2}, y_k + \frac{h}{2}\cdot K_1\right)$$

Deuxième calcul approché de $f\left(t_k + \frac{h}{2}, y\left(t_k + \frac{h}{2}\right)\right)$.

On utilise la méthode des rectangles à droite pour le calcul d'une intégrale, ce qui donne :

$$y(t_k + \frac{h}{2}) = y(t_k) + \int_{t_k}^{t_k+\frac{h}{2}} f(x, y(x))dx \cong y_k + \frac{h}{2}\cdot f\left(t_k + \frac{h}{2}, y\left(t_k + \frac{h}{2}\right)\right)$$

et en utilisant (1), il vient :

$$y(t_k + \frac{h}{2}) \cong y_k + \frac{h}{2}\cdot f\left(t_k + \frac{h}{2}, y_k + \frac{h}{2}\cdot K_1\right)$$

c'est-à-dire :

$$(2) \quad y\left(t_k + \frac{h}{2}\right) \cong y_k + \frac{h}{2} \cdot K_2$$

et :

$$f\left(t_k + \frac{h}{2}, y\left(t_k + \frac{h}{2}\right)\right) \cong K_3 = f\left(t_k + \frac{h}{2}, y_k + \frac{h}{2} \cdot K_2\right)$$

Calcul approché de $f(t_k + h, y(t_k + h))$.

On a :

$$y(t_k + h) = y(t_k) + \int_{t_k}^{t_k+h} f\big(x, y(x)\big)dx$$

l'intégrale se calculant avec la méthode du point milieu, soit :

$$y(t_k + h) \cong y_k + h \cdot f\left(t_k + \frac{h}{2}, y\left(t_k + \frac{h}{2}\right)\right)$$

ce qui donne avec (2) :

$$y(t_k + h) \cong y_k + h \cdot f\left(t_k + \frac{h}{2}, y_k + \frac{h}{2} \cdot K_2\right)$$

c'est-à-dire :

$$(3) \quad y(t_k + h) \cong y_k + h \cdot K_3$$

et :

$$f(t_k + h, y(t_k + h)) \cong K_4 = f(t_k + h, y_k + h \cdot K_3)$$

La méthode obtenue est d'ordre 4. On l'appelle méthode *RK4* (pour méthode de Runge-Kutta d'ordre 4). C'est cette méthode que l'on utilise le plus souvent. Elle est donc définie par :

$$\Phi(t,y,h) = \frac{1}{6} \cdot \{K_1 + 2 \cdot K_2 + 2 \cdot K_3 + K_4\}$$

où :

$$\begin{cases} K_1 = f(t, y) \\ K_2 = f\left(t + \frac{h}{2}, y + \frac{h}{2} \cdot K_1\right) \\ K_3 = f\left(t + \frac{h}{2}, y + \frac{h}{2} \cdot K_2\right) \\ K_4 = f(t + h, y + h \cdot K_3) \end{cases}$$

Les calculs des K_j devant se faire dans l'ordre indiqué.

3.3.3 Remarque

Les méthodes de Runge-Kutta plus générales sont basées sur le même principe. On calcule tout d'abord $\int_{t_k}^{t_k+h} f\big(x, y(x)\big)dx$ avec une méthode du type :

$$\int_{t_k}^{t_k+h} f\big(x, y(x)\big)dx \cong \sum_{i=1}^{q} b_i \cdot f\big(t_{i,k}, y(t_{i,k})\big)$$

avec $t_k \le t_{1,k} \le \ldots \le t_{q,k} \le t_{k+1}$.

Le calcul de chaque $f(t_{i,k},y(t_{i,k}))$ se faisant avec : $y(t_{i,k}) = y(t_k) + \int_{t_k}^{t_{i,k}} f(x,y(x))dx$ en utilisant une formule d'intégration du type :

$$\int_{t_k}^{t_{i,k}} f(x,y(x))dx \cong \sum_{j=1}^{i} a_{i,j} \cdot f\left(t_{j,k}, y\left(t_{j,k}\right)\right)$$

et en exploitant les calculs précédents.

3.3.4 Les schémas RK2

L'idée est de généraliser la méthode d'Euler-Cauchy en cherchant tous les quadruplés a_1, a_2, a_3, a_4 tels que le schéma d'intégration défini par :

$$\Phi(t,y,h) = a_1 \cdot f(t,y) + a_2 \cdot f(t + a_3 \cdot h, y + a_4 \cdot h \cdot f(t,y))$$

soit d'ordre 2, ce qui équivaut à dire que l'erreur de consistance est d'ordre 3.

On a :

$$\begin{cases} \Phi(t,y,0) = (a_1 + a_2) \cdot f(t,y) \\ \dfrac{\partial \Phi}{\partial h}(t,y,0) = a_2 \cdot a_3 \cdot \dfrac{\partial f}{\partial t}(t,y) + a_2 \cdot a_4 \cdot \dfrac{\partial f}{\partial y}(t,y) \cdot f(t,y) \end{cases}$$

Donc la méthode sera d'ordre 2 si, et seulement si :

$$\begin{cases} a_1 + a_2 = 1 \\ a_2 \cdot a_3 = a_2 \cdot a_4 = \dfrac{1}{2} \end{cases}$$

En posant $\lambda = a_2$, on déduit alors que les méthodes de Runge-Kutta d'ordre 2 sont définies par :

$$\Phi(t,y,h) = (1 - \lambda) \cdot f(t,y) + \lambda \cdot f\left(t + \frac{h}{2 \cdot \lambda}, y + \frac{h}{2 \cdot \lambda} \cdot f(t,y)\right)$$

Pour $\lambda = 1$, on a le schéma d'Euler Modifié, pour $\lambda = 1/2$, on a le schéma d'Euler-Cauchy et pour $\lambda = 3/4$, on a le schéma de Heun.

3.4 Programmation structurée de la méthode RK4 pour les équations différentielles d'ordre 1

Dans ce paragraphe, on décrit une itération de la méthode de Runge-Kutta d'ordre 4 pour résoudre une équation différentielle d'ordre 1 :

$$\begin{cases} y'(t) = f(t,y(t)) \text{ sur } [a,b] \\ y(a) = y_a \end{cases}$$

où $f : [a,b] \rightarrow R$ est continue et lipschitzienne en y.

A l'étape i du calcul, on dispose d'une abscisse initiale t_Initial et d'une ordonnée y_Initial, valeur approchée de y(t_Initial), on déduit alors les valeurs t_Final = t_Initial + h, où h est un pas d'intégration, et la valeur approchée de y(t_Final), y_Final, obtenue par la méthode de Runge_Kutta d'ordre 4.

Il suffit ensuite d'exécuter cette procédure sur chaque intervalle $[t_i,t_{i+1}]$, $(0 \le i \le n - 1$, où $h = \dfrac{b-a}{n})$ pour avoir une solution approchée sur [a,b].

La variable f, de type Fonction, représente une fonction de deux variables réelles x et y définissant l'équation différentielle.

PROCEDURE RK4_11(Entrée f : Fonction ; t_Initial, y_Initial, h : Réel ;
Sortie t_Final, y_Final : Réel) ;
Début
k1 = h·f(t_Initial,y_Initial) ;
$t\_Final = t\_Initial + \frac{h}{2}$;
$k2 = h \cdot f\left(t\_Final, y\_Initial + \frac{k1}{2}\right)$;
$k3 = h \cdot f\left(t\_Final, y\_Initial + \frac{k2}{2}\right)$;
t_Final = t_Initial + h ;
k4 = h·f(t_Final,y_Initial + k3) ;
$y\_Final = y\_Initial + \frac{k1 + 2 \cdot (k2 + k3) + k4}{6}$;
Fin ;

3.5 Programmation structurée de la méthode RK4 pour les systèmes de p équations différentielles d'ordre 1

Dans ce paragraphe, on s'intéresse à un système de p équations différentielles d'ordre 1 :

$$\begin{cases} y_1' = f_1(t, y_1, \ldots, y_p) \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad t \in [a, b] \\ y_p' = f_p(t, y_1, \ldots, y_p) \\ y_k(a) = y_{k,a} \text{ donnés dans R pour } k = 1, \ldots, p \end{cases}$$

Il suffit de reprendre la procédure précédente de « façon vectorielle ».

Pour ce faire on définit un type SystemeDiff comme un tableau de fonctions des variables t de type réel et y de type Variable_2 lui même défini comme un tableau de réels. Ainsi f(j)(t,y) représentera $f_j(t,y_1,\ldots,y_p)$, pour tout j = 1, ..., p. Ce qui donne la procédure ci-dessous décrivant une itération de la méthode :

PROCEDURE RK4_p1(Entrée f : SystemeDiff ; t_Initial, h : Réel ;
y_Initial : Variable_2 ; p : Entier ;
Sortie t_Final : Réel ; y_Final : Variable_2) ;
Début
Pour j allant de 1 à p faire k1(j) = h·f(j)(t_Initial,y_Initial) ;
$t\_Final = t\_Initial + \frac{h}{2}$;
Pour j allant de 1 à p faire $y\_Final(j) = y\_Initial(j) + \frac{k1(j)}{2}$;
Pour j allant de 1 à p faire k2(j) = h·f(j)(t_Final,y_Final) ;
Pour j allant de 1 à p faire $y\_Final(j) = y\_Initial(j) + \frac{k2(j)}{2}$;

Pour j allant de 1 à p faire k3(j) = h·f(j)(t_Final,y_Final) ;
t_Final = t_Initial + h ;
Pour j allant de 1 à p faire y_Final(j) = y_Initial(j) + k3(j) ;
Pour j allant de 1 à p faire k4(j) = h·f(j)(t_Final,y_Final) ;
Pour j allant de 1 à p faire

$$y\_Final(j) = y\_Initial(j) + \frac{k1(j) + 2\cdot(k2(j) + k3(j)) + k4(j)}{6} \ ;$$

Fin ;

3.6 Programmation structurée pour les équations différentielles d'ordre q

Dans ce paragraphe, on s'intéresse aux équations différentielles d'ordre $q \geq 1$.

$$\begin{cases} y^{(q)}(t) = f\left(t, y(t), y'(t), \ldots, y^{(q-1)}(t)\right) \text{ sur } [a,b] \\ y(a) = y_a, \ldots, y^{(q-1)}(a) = y_a^{q-1} \end{cases}$$

Les conditions initiales sont stockées dans un vecteur y_Initial de composantes $y^{(j)}(a)$ pour j = 0, ..., q – 1.

Comme indiqué au paragraphe (2.1), on se ramène, par changement de fonction inconnue, à un système de q équations différentielles d'ordre 1 en posant $z_i = y^{(i)}$ pour i = 0, ..., q – 1, ce qui donne le système :

$$\begin{cases} z_0' = z_1 \\ \ldots\ldots\ldots \\ z_{q-2}' = z_{q-1} \\ z_{q-1}' = f\left(t, z_0, \ldots, z_{q-1}\right) \end{cases}$$

noté u' = g(t,u') que l'on résout par la méthode de Runge-Kutta pour les systèmes.

La variable f, de type Fonction, représente une fonction des variables t, de type réel et y, de type Variable_2 définissant l'équation différentielle. De sorte que f(t,y), avec y de composantes $y_0,\ldots,y_{q-1}$ représente $f(t,y,y',\ldots,y^{(q-1)})$.

PROCEDURE RK4_1q(Entrée f : Fonction ; t_Initial, h : Réels ;
y_Initial : Variable_2 ; q : Entier ;
Sortie t_Final : Réel ; y_Final : Variable_2 ;
Début
Pour j allant de 0 à q – 2 Faire k1(j) = h·y_Initial(j+1) ;
k1(q-1) = h·f(t_Initial,y_Initial) ;

$$t\_Final = t\_Initial + \frac{h}{2} \ ;$$

$$Pour\ j\ allant\ de\ 0\ à\ q-1\ faire\ y\_Final(j) = y\_Initial(j) + \frac{k1(j)}{2} \ ;$$

Pour j allant de 0 à q – 2 Faire k2(j) = h·y_Final(j+1) ;
k2(q-1) = h·f(t_Final,y_Final) ;

$$Pour\ j\ allant\ de\ 0\ à\ q-1\ faire\ y\_Final(j) = y\_Initial(j) + \frac{k2(j)}{2} \ ;$$

Pour j allant de 0 à q – 2 Faire k3(j) = h·y_Final(j+1) ;

k3(q-1) = h·f(t_Final,y_Final) ;
Pour j allant de 0 à q – 1 faire y_Final(j) = y_Initial(j) + k3(j) ;
t_Final = t_Initial + h ;
Pour j allant de 0 à q – 2 Faire k4(j) = h·y_Final(j+1) ;
k4(q-1) = h·f(t_Final,y_Final) ;
Pour j allant de 0 à q – 1 Faire

$$y\_Final(j) = y\_Initial(j) + \frac{k1(j) + 2\cdot(k2(j) + k3(j)) + k4(j)}{6} ;$$

Fin ;

3.7 Programmation structurée pour les systèmes de p équations différentielles d'ordre q

On s'intéresse ici à la résolution d'un système de p équations différentielles d'ordre q du type :

$$y_j^{(q)} = f_j\left(t, y_1, \ldots, y_1^{(q-1)}, \ldots, y_p, \ldots, y_p^{(q-1)}\right) \quad (j = 1, \ldots, p)$$

avec les conditions initiales : $y_j^{(k)}(a)$ donnés pour j = 1, ..., p et k = 0, ..., q – 1.

Un tel système sera représenté par un tableau de fonctions f. Pour tous j = 1,...,p et i = 0, ..., q – 1, la dérivée $y_j^{(i)}$ sera représentée par la composante $y_{j,i}$ d'une matrice y à p lignes et q colonnes. La fonction f_j a pour paramètre un réel x et un vecteur y de composantes $(y_1,...,y_{p\cdot q})$ où $y_{q\cdot(j-1)+i+1}$ représente la dérivée numéro i de la fonction numéro j. La représentation matricielle de ces dérivées est faite pour faciliter l'entrée des données et la représentation vectorielle pour faciliter la programmation.

Ce qui donne la procédure suivante, toujours pour une itération de la méthode :

```
PROCEDURE RK4_pq(Entrée f : SystemeDiff ; t_Initilal, h : Réel ;
                                 y_Initial : Matrice ; p, q : Entier ;
                            Sortie t_Final : Réel ; y_Final : Matrice) ;
Début
    Pour j allant de 1 à p faire
    Début
        Pour i allant de 0 à q – 1 faire
        Début
            yiq·(j–1)+i+1 = y_Initialj,i ;
        Fin ;
    Fin ;
    Pour j allant de 1 à p faire
    Début
        Pour i allant de 0 à q – 2 faire
        Début
            k1q·(j–1)+i+1 = h·yiq·(j–1)+i+2 ;
        Fin ;
        k1q·j = h·f(j)(t_Initial,yi) ;
    Fin ;
    t_Final = t_Initial + h/2 ;
    Pour j allant de 1 à p·q faire
```

Début
$yf_j = yi_j + 0.5 \cdot k1_j$;
Fin ;
Pour j allant de 1 à p faire

```
Début
    Pour i allant de 0 à q – 2 faire
    Début
        k2q·(j–1)+i+1 = h·yfq·(j–1)+i+2 ;
    Fin ;
    k2q·j = h·f(j)(t_Initial,yf) ;
Fin ;
Pour j allant de 1 à p·q faire
Début
        yfj = yij + 0.5·k2j ;
Fin ;
Pour j allant de 1 à p faire
Début
    Pour i allant de 0 à q – 2 faire
    Début
        k3q·(j–1)+i+1 = h·yfq·(j–1)+i+2 ;
    Fin ;
    k3q·j = h·f(j)(t_Initial,yf) ;
Fin ;
t_Final = t_Initial + h ;
Pour j allant de 1 à p·q faire
Début
        yfj = yij + k3j ;
Fin ;
Pour j allant de 1 à p faire
Début
    Pour i allant de 0 à q – 2 faire
    Début
        k4q·(j–1)+i+1 = h·yfq·(j–1)+i+2 ;
    Fin ;
    k4q·j = h·f(j)(t_Initial,yf) ;
Fin ;
Pour j allant de 1 à p faire
Début
    Pour i allant de 0 à q – 1 faire
    Début
        y_Final(j,i) = yiq·(j–1)+i+1
                + (k1q·(j–1)+i+1 + 2·(k2q·(j–1)+i+1 + k3q·(j–1)+i+1) + k4q·(j–1)+i+1) / 6
    Fin ;
Fin ;
Fin ;
```

4. Contrôle du pas d'intégration

4.1 Position du problème

A l'étape i du calcul, dans une méthode à un pas de résolution d'une équation différentielle on voudrait modifier le pas d'intégration h_i de façon à atteindre une précision déterminée $\varepsilon > 0$. L'idée de prendre un pas h très petit présente

l'inconvénient de nécessiter beaucoup de calculs et de ce fait va entraîner un cumul trop important des erreurs d'arrondis.

On prendra donc, au départ, un pas h_0 « raisonnable » et ce dernier sera modifié en cours de route (diminué ou augmenté) de façon à avoir la précision désirée.

Pour ce faire, il est nécessaire de connaître une estimation de l'erreur de discrètisation en fonction du pas h_i à chaque étape i du calcul. Il est clair que cette estimation va entraîner une augmentation du temps de calcul.

4.2 Choix du pas d'intégration à l'étape i du calcul

Considérons une méthode d'intégration à un pas définie par :

$$\begin{cases} y_0 = y(t_0) \text{ donné} \\ y_{i+1} = y_i + h \cdot \Phi(t_i, y_i, h_i) \quad (i = 0, \ldots, n - 1) \end{cases}$$

où $h_i = t_{i+1} - t_i$ est un pas non constant.

En supposant que, à l'étape i du calcul, y_i est une approximation de $y(t_i)$ avec une précision inférieure ou égale à ε_0 , on voudrait modifier le pas h_i de sorte que y_{i+1} donne une approximation de $y(t_{i+1})$ avec une précision $\leq \varepsilon_0$.

Notons h_Initial le pas utilisé à l'étape i du calcul (au départ, on prendra un pas h_0 arbitraire).

En reprenant le raisonnement du paragraphe (2.2.4), on voit que pour une méthode d'ordre p et un pas h donné, l'erreur de consistance est donnée par :

$$\tau(t,h) = y(t + h) - y(t) - h \cdot \Phi(t,y(t),h) = C_x \cdot h^{p+1} + O(h^{p+2})$$

où C_x est une constante qui ne dépend que de $y^{(p+1)}(t)$ et non de h (on suppose évidemment qu'il existe une seule solution assez régulière).

En appliquant la méthode une fois avec le pas h = h_Initial et deux fois avec le pas h/2 on obtient :

$$\begin{cases} y(t_i + h) \cong y_2 + C_1 \cdot h^{p+1} + O(h^{p+2}) \\ y(t_1 + h) \cong y_2 + \left(C_1 + C_{i+\frac{1}{2}} \right) \cdot \left(\frac{h}{2} \right)^{p+1} + O(h^{p+2}) \end{cases}$$

où y_1 et y_2 sont les valeurs approchées de $y(t_1 + h)$ obtenues avec chacune des deux méthodes.

En supposant h assez petit, on peut écrire que $C_{i+\frac{1}{2}} \cong C_1$, de sorte que l'on a :

$$\begin{cases} y(t_i + h) \cong y_2 + C_1 \cdot h^{p+1} + O(h^{p+2}) \\ y(t_1 + h) \cong y_2 + 2 \cdot C_1 \cdot \left(\frac{h}{2} \right)^{p+1} + O(h^{p+2}) \end{cases}$$

Si on pose : $\varepsilon = y_2 - y_1$ on a :

$$\varepsilon \cong 2 \cdot C_1 \cdot (2^p - 1) \cdot \left(\frac{h}{2} \right)^{p+1} + O(h^{p+2})$$

de sorte que

$$\frac{\varepsilon}{2^p - 1} \cong 2 \cdot C_1 \cdot \left(\frac{h}{2}\right)^{p+1} + O\left(h^{p+2}\right)$$

est une estimation de l'erreur que l'on commet en remplaçant $y(t_i + h)$ par y_2.

De plus, on a :

$$y(t_i + h) \cong \left(y_2 + \frac{\varepsilon}{2^p - 1}\right) + O\left(h^{p+2}\right)$$

c'est-à-dire que $\left(y_2 + \frac{\varepsilon}{2^p - 1}\right)$ est une meilleure approximation de $y(t_i + h)$.

D'autre part, comme $|\varepsilon|$ est proportionnel à h^{p+1}, en notant ε_1 l'approximation de l'erreur correspondante à un autre pas h_1, on aura :

$$\frac{|\varepsilon_1|}{|\varepsilon|} = \left(\frac{h_1}{h}\right)^{p+1}$$

Si pour h = h_Initial, on a $|\varepsilon| = |y_2 - y_1| > \varepsilon_0$, cela signifie que le pas h ne convient pas et il faudra alors le remplacer par le pas h_Final qui va donner une erreur de l'ordre de ε_0. En définitive, on prendra :

$$h\_Final = h\_Initial \cdot \left(\frac{\varepsilon_0}{|\varepsilon|}\right)^{\frac{1}{p+1}}$$

c'est-à-dire qu'on diminue le pas.

Dans le cas contraire, on augmente le pas en prenant : h_Final = 2·h_Initial

4.3 Programmation structurée de la méthode de Runge-Kutta d'ordre 4 avec contrôle du pas

On note h_0 la valeur maximale autorisée du pas et la précision minimale ε_0 est supposée donnée en constante.

On décrit la procédure dans le cas d'une équation différentielle d'ordre 1 à une variable, les trois autres cas s'en déduisant facilement.

PROCEDURE RK4_11_Controle(Entrée f : Fonction ; t_initial, y_Initial, h0 : Réel ;
Sortie t_Final, y_Final, Entrée_Sortie h_Initial : Réel) ;
Début
Répeter
RK4_11(f,t_Initial,y_Initial,h_Initial/2,t_Final,y1) ;
RK4_11(f,t_Initial + h_Initial/2,y1,h_Initial/2,t_Final,y2) ;
RK4_11(f,t_Initial,y_Initial,h_Initial,t_Final,y1) ;
$\varepsilon = \frac{|y2 - y1|}{\varepsilon_0}$;
Si $\varepsilon > 1$
Alors h_Initial = $\frac{h\_Initial}{\varepsilon^{\frac{1}{5}}}$
Sinon Si h_Initial < $h_0/2$ *Alors h_Initial = h_Initial·2 ;*
Jusqu'à ($\varepsilon \leq 1$) ;

$$y\_Final = \frac{16 \cdot y2 - y1}{15};$$

Fin ;

5. Problèmes avec conditions aux limites. Méthode du tir

5.1 Introduction : position du problème

Dans les problèmes que nous avons considérés jusqu'à présent, les conditions aux limites étaient des « conditions initiales », c'est-à-dire que pour un système différentiel y' = f(t,y) on connaît toutes les valeurs initiales $y_j(a)$, pour j = 1, 2, ..., p.

Mais on peut aussi considérer des problèmes différentiels où les *conditions aux limites* portent sur les deux bornes de l'intervalle, ou sur ces deux bornes et des points intérieurs.

Par exemple, la distribution de température à l'intérieur d'un cylindre de rayon 1, en mode stationnaire, est décrite par l'équation différentielle, avec conditions aux limites :

$$\begin{cases} y''(t) = -\dfrac{y'(t)}{t} - \alpha \cdot e^{y(t)} \quad (t \in]0,1[) \\ y'(1) = y(1) = 0 \end{cases}$$

où $\alpha \in]0,0.8]$ est une constante proportionnelle à l'inverse de la conductivité.

Dans ce paragraphe, on s'occupera de problèmes aux limites du type :

$$(1) \quad \begin{cases} y'(t) = f(t, y(t)) \quad (t \in]a,b[) \\ y_j(a) = y_{j,a} \quad (j \in I_1) \\ y_j(b) = y_{j,b} \quad (j \in I_2) \end{cases}$$

où $I_1 \cup I_2 = \{1, ..., p\}$ et I_1, I_2 disjoints.

On notera p_1 le nombre d'éléments de I_1 et p_2 celui de I_2. De plus, les indices dans I_2 seront notés : $j_1 < j_2 < ... < j_{p_2}$.

Une équation différentielle d'ordre $q \geq 2$, où les conditions aux limites portent sur les deux bornes de l'intervalle peut toujours se ramener à (1) par le procédé classique.

L'idée de la méthode du tir, pour résoudre (1) est de démarrer en posant les conditions initiales manquantes de façon arbitraire, c'est-à-dire qu'on se donne un vecteur $\alpha = (\alpha_1,...,\alpha_{p_2})$, et on pose :

$$y_{j_k}(a) = \alpha_k \ (k = 1, ..., p_2)$$

On résout alors le système différentiel par une méthode à un pas, ce qui va donner une valeur en b de la solution que l'on note $y(b,\alpha)$.

Le problème est alors de trouver $\alpha \in R^{p^2}$ solution du système de p_2 équations à p_2 inconnues :

$$e_k(\alpha) = y_{j_k}(b,\alpha) - y_{j_k,b} = 0 \ (k = 1, ..., p)$$

Remarque : Cas des systèmes linéaires — Dans le cas où le système est linéaire, la méthode peut être simplifiée. En prenant p valeurs distinctes α^1, ..., α^p de sorte que les valeurs initiales $y^1(a)$, ..., $y^p(a)$ correspondantes soient linéairement indépendantes, on construit p solutions linéairement indépendantes y^1, ..., y^p et la solution cherchée va s'écrire :

$$y(t) = \sum_{i=1}^{p} \lambda_i \cdot y^i(t)$$

les constantes λ_i étant uniquement déterminées par les conditions initiales $y_{j,a}$ et $y_{k,b}$, pour j dans I_1 et k dans I_2.

5.2 La méthode du tir

5.2.1 Rappels

Il s'agit donc de résoudre le problème (1).

On démarre avec α choisi arbitrairement dans R^{p^2} , ce qui va donner une erreur sur b, $e(\alpha) = \left(\left(e_k(\alpha)\right)\right)_{1 \le k \le p_2}$, définie par :

$$e_k(\alpha) = y_{j_k}(b, \alpha) - y_{j_k, b} \qquad (k = 1, ..., p)$$

En utilisant la méthode de Newton-Raphson, on construit donc la suite de vecteurs (α^k), de la façon suivante :

$$\begin{cases} \alpha^0 = \alpha \\ \alpha^{k+1} = \alpha^k - \delta^k \quad (k \ge 0) \end{cases}$$

où δ^k est solution de :

$$A_k \cdot \delta^k = e(\alpha^k)$$

avec $A_k = \left(\left(a_{i,j}\right)\right)_{1 \le i,j \le p_2}$, $a_{i,j} = \dfrac{e_i\left(\beta^{k,j}\right) - e_i\left(\alpha^k\right)}{h}$ et

$\beta^{k,j} = \left(\alpha_1^k, ..., \alpha_{j-1}^k, \alpha_j^k + h, \alpha_{j+1}^k, ..., \alpha_j^n\right)$, h etant un pas assez petit.

5.2.2 Programmation structurée de la méthode du tir

Pour reconnaître les indices, on introduit un tableau de booléens B défini par :

$$\begin{cases} B_j = \text{Faux si } j \in I_1 \quad \left(\text{i. e. } y_j(a) \text{ connue}\right) \\ B_j = \text{Vrai si } j \in I_2 \quad \left(\text{i. e. } y_j(a) \text{ inconnue}\right) \end{cases}$$

Les $y_j(a)$, pour j = 1, ..., p, c'est-à-dire ceux qui sont connus et ceux qui sont donnés arbitrairement, seront stockés dans la variable y_Initial de type Variable_2 (tableau de réels numérotés de 1 à p). Les $y_j(b)$ connus seront stockés dans la variable y_Final de type Variable_2.

On suppose que l'on dispose d'une procédure de résolution d'un système de p équations différentielles définie par :

PROCEDURE RésolSystDiff(Entrée f : SystemeDiff ; a, b : Réel ;

```
                                    y_Initial : Variable_2 ; p : Entier ; Sortie y : Matrice) ;
```

où $y_{j,k}$ est une approximation de $y_j(t_k)$, pour j allant de 1 à p et k de 0 à n, avec $t_k = a + k \cdot \frac{b-a}{n}$.

En particulier, $y_{j,n}$ est l'approximation trouvée de $y_j(b)$.

On suppose également que l'on dispose d'une procédure de résolution d'un système linéaire définie par :

```
PROCEDURE SystLin(Entrée n : Entier ; A : Matrice ; b : Variable_2 ;
                  Sortie x : Variable_2) ;
```

On se reportera au chapitre sur l'analyse numérique linéaire pour l'écriture d'une telle procédure.

On définit tout d'abord une procédure de calcul de l'erreur commise à l'issue d'un tir. Soit :

```
PROCEDURE Calcul_Erreur(Entrée y_Final : Variable_2 ;
                                B : TableauBooléens ;  p : Entier ; y : Matrice ;
                        Sortie e : Variable_2) ;
Début
   i = 0 ;
   Pour j allant de 1 à p Faire
   Début
      Si Bj
      Alors Début
            i = i + 1 ;
            ei = yj,n – y_Finalj ;
         Fin ;
   Fin ;
Fin ;
```

On écrit ensuite une procédure de correction du tir, qui réalise une itération dans l'algorithme décrit ci-dessus, c'est-à-dire qu'on calcule α^{k+1} en fonction de α^{k}. Ce qui revient à remplacer y_Initial par sa nouvelle valeur définie par :

$$y\_Initial_j = y\_Initial_j - \delta_j \quad (j \in I_2)$$

où δ est solution du système : $M \cdot \delta = e$, e étant l'erreur précédemment calculée.

```
PROCEDURE Correction_Tir(Entrée f : SystemeDiff ; a, b : Réel ; p : Entier ;
                                y_Final : Variable_2 ; B : TableauBooléens ;
                        Sortie y_Initial : Variable_2 ; y : Matrice ; Arret : Booléen) ;
Début
   p2 = 0 ;
   Pour j allant de 1 à p Faire
   Début
      Si Bj Alors p2 = p2 + 1 ;
   Fin ;
   ResolSystDiff(f,a,b,y_Initial,p,y) ;
   Calcul_Erreur(y_Final,B,p,y,e) ;
   Arret = ( ||e|| < ε ) ;
   Si Arret
```

Alors Sortir de la procédure ;

```
{ Formation de la matrice jacobienne }
   k = 0 ;
   Pour j Allant de 1 à p Faire
   Début
      Si B_j
      Alors Début
            k = k + 1 ;
            Aux = y_Initial_j ;
            y_Initial_j = y_Initial_j + ε ;
            ResolSystDiff(f,a,b,y_Initial,p,y) ;
            Calcul_Erreur(y_Final,B,p,y,e1) ;
            Pour i Allant de 1 à p_2 Faire M_i,k = (e1_i − e_i) / ε ;
            y_Initialj = Aux ;
         Fin ;
   Fin ;
   SystLin(p2,M,e,δ) ;
   { Calcul de la nouvelle valeur de y_Initial }
   k = 0 ;
   Pour j Allant de 1 à p Faire
   Début
      Si B_j
      Alors Début
            k = k + 1 ;
            y_Initialj = y_Initial_j - δ_j ;
         Fin ;
   Fin ;
Fin ;
```

La procédure finale de résolution d'un système de p équations différentielles d'ordre 1 va alors s'écrire :

```
PROCEDURE Résolution_Tir(Entrée f : SystemeDiff ; a, b : Réel ; p : Entier ;
                                        y_Initial, y_Final : Variable_2 ; B : TableauBooléens ;
                                Sortie y : Matrice) ;
Début
   Répéter
      Correction_Tir(f,a,b,p,y_Final,B,y_Initial,y,Arret) ;
   Jusqu'à Arret ;
   ResolSystDiff(f,a,b,y_Initial,p,y) ;
Fin ;
```

6. Un exemple d'application. Mouvement de translation d'un corps sphérique pesant dans un fluide au repos

6.1 Position du problème et notations

On considère de nouveau le problème étudié au paragraphe (2.7) du chapitre sur l'approximation et l'interpolation.

On a un corps sphérique que l'on laisse tomber à l'instant t = 0, sans vitesse initiale, dans un fluide au repos et on veut calculer la distance à l'origine z(t) et la vitesse v(t) = z'(t) du corps à chaque instant t.

On utilisera les notations suivantes :

- m est la *masse* du corps sphérique ;
- g est *l'accélération due à la pesanteur*, dans la direction de l'axe Oz ;
- à l'instant t = 0, le centre de la sphère est à z = 0 et la vitesse est v = 0 ;
- d est le *diamètre* de la sphère ;
- ρ est la *densité volumique de la sphère* ;
- ρ_f est la *densité volumique du fluide* ;
- ν est la *viscosité cinématique du fluide* ;
- C_d est le *coefficient de traînée* de la sphère en régime stationnaire ;

Remarque — Le coefficient C_d est lié au *nombre de Reynolds* de la sphère lui même relié à la vitesse de cette dernière.

Dans le paragraphe (2.7) du chapitre 3, on a donné un programme qui permet de calculer C_d en fonction de R_e.

6.2 Equations du mouvement de la sphère

On sait que dans le vide la loi du mouvement de la sphère est :

$$z(t) = \frac{1}{2} \cdot g \cdot t^2$$

Dans un fluide plus général, on doit prendre en compte les forces suivantes :

- la *poussée d'Archimède*, égale au poids du fluide déplacé par la particule, soit :

$$-m_f \cdot g = -\frac{1}{6} \cdot \pi \cdot d^3 \cdot \rho_f \cdot g \qquad \uparrow -m_f \cdot g$$

- la *force induite par l'accélération* :

quand un corps plongé dans un liquide au repos est brusquement mis en mouvement avec une accélération $\frac{dv}{dt}$, il en résulte une force de réaction :

$$-\lambda \cdot \frac{dv}{dt} = -\frac{1}{2} \cdot m_f \cdot \frac{dv}{dt} \qquad \uparrow -\lambda \cdot \frac{dv}{dt}$$

où λ est la *« masse virtuelle »* ou « masse induite par accélération ».

- la *force de réaction due à la viscosité du fluide* qui s'écrit sous la forme :

$$-C_p \cdot \frac{1}{2} \cdot \rho_f \cdot v \cdot |v| \cdot A_p$$

où

$$A_p = \frac{1}{4} \cdot \pi \cdot d^2$$

et C_d est le *« coefficient de traînée »*.

La loi de Newton nous permet d'écrire l'équation du mouvement de la particule :

$$m \cdot \frac{dv}{dt} = m \cdot g - m_f \cdot g - \frac{1}{2} \cdot m_f \cdot \frac{dv}{dt} - \frac{1}{2} \cdot \rho_f \cdot v \cdot |v| \cdot \frac{\pi}{4} \cdot d^2 \cdot C_d$$

soit l'équation différentielle du second ordre :

$$(2)\quad \left(m+\frac{1}{2}\cdot m_f\right)\cdot\frac{d^2z}{dt^2}=(m-m_f)\cdot g-\frac{\pi}{8}\cdot\rho_f\cdot\frac{dz}{dt}\cdot\left|\frac{dz}{dt}\right|\cdot d^2\cdot C_d\left(\frac{dz}{dt}\right)$$

encore équivalente au système de deux équations différentielles d'ordre 1 :

$$(3)\quad \begin{cases} v(t)=\dfrac{dz}{dt}(t) \\ \left(m+\dfrac{1}{2}\cdot m_f\right)\cdot\dfrac{dv}{dt}=(m-m_f)\cdot g-\dfrac{\pi}{8}\cdot\rho_f\cdot v\cdot|v|\cdot d^2\cdot C_d(v) \end{cases}$$

Remarque — Le terme $m+\frac{1}{2}\cdot m_f$ exprime que la particule se comporte comme si sa masse avait été augmentée (dans un mouvement accéléré ou décéléré).

En écrivant que :

$$m=\frac{1}{6}\cdot\pi\cdot d^3\cdot\rho$$

et en posant :

$$\begin{cases} A=1+\dfrac{1}{2}\cdot\dfrac{\rho_f}{\rho} \\ B=\left(1-\dfrac{\rho_f}{\rho}\right)\cdot g \\ C=\dfrac{3}{4}\cdot\dfrac{\rho_f}{d\cdot\rho} \end{cases}$$

le système d'équations différentielles va s'écrire :

$$\begin{cases} v=\dfrac{dz}{dt} \\ \dfrac{dv}{dt}=\dfrac{1}{A}\cdot\left(B-C\cdot v\cdot|v|\cdot C_d(v)\right) \end{cases}$$

Le programme du paragraphe (2.7) du chapitre sur l'interpolation et l'approximation peut donc se compléter comme suit.

```
with TEXT_IO, IIO, FIO, CRT, COMMON_MATH0, MATH0, MATH1,
     COMMON_MATH3, MATH3, CURVE ;
use TEXT_IO, IIO, FIO, CRT, COMMON_MATH0, MATH0, MATH1,
    COMMON_MATH3, MATH3, CURVE ;
procedure BILLE is

NBR_INTERVALLES : constant NATURAL := 7 ;
  -- Nombre maximale d'intervalles dans [0,50000]
  -- Dans chaque intervalle on a des mesures experimentales de Re et Cd
NBR_MESURES : constant NATURAL := 6 ;
  -- Nombre maximale de mesure des Re et Cd dans chaque intervalle
type MATRICE_MESURES is array(  POSITIVE range 1..NBR_INTERVALLES,
                                POSITIVE range 1..NBR_MESURES) of FLOAT ;
  -- Dans l'intervalle N° j, les mesures de Re et Cd sont notees
```

```
  -- Re(j,i) et Cd(j,i) ou 1 <= i <= NBR_MESURES
type VECTEUR_COEFF is array(POSITIVE range 1..NBR_INTERVALLES,
                            NATURAL range 0..2) of FLOAT ;
  -- Dans l'intervalle N° j, les coefficients de Morsi sont notes
  -- K(j,i), ou i = 1, 2, 3.
type VECTEUR_ENTIER is array(POSITIVE range 1..NBR_INTERVALLES)
       of NATURAL ;
  -- Dans l'intervalle N° j, le nombre de mesures est note n(j)
Re, Cd : MATRICE_MESURES ;
nn : VECTEUR_ENTIER ;
K : VECTEUR_COEFF ;
Rho, Rhof, g, Nu, d, t0, tMax, z0, v0, A, B, C : FLOAT ;

procedure DONNEES_DU_FICHIER(
        t0, tMax, z0, v0, Rho, Rhof, Nu, g, d, A, B, C : in out FLOAT ;
        nn : in out VECTEUR_ENTIER ;
        Re, Cd : out MATRICE_MESURES ; K : out VECTEUR_COEFF) is
Nom : STRING(1..50) ;
FichierDonnees : FILE_TYPE ;
RhoBarre : FLOAT ;
begin
    PUT_LINE("   Donnees du fichier") ;
    LIRE_FICHIER(FichierDonnees,Nom) ;
    GET(FichierDonnees,t0) ;
    GET(FichierDonnees,tMax) ;
    GET(FichierDonnees,z0) ;
    GET(FichierDonnees,v0) ;
    GET(FichierDonnees,Rho) ;
    GET(FichierDonnees,Rhof) ;
    GET(FichierDonnees,Nu) ;
    GET(FichierDonnees,g) ;
    GET(FichierDonnees,d) ;
    SKIP_LINE(FichierDonnees) ;
    for j in 1..NBR_INTERVALLES
    loop
       GET(FichierDonnees,nn(j)) ;
       SKIP_LINE(FichierDonnees) ;
       for i in 1..nn(j)
       loop
          GET(FichierDonnees,Re(j,i)) ;
       end loop ;
       SKIP_LINE(FichierDonnees) ;
       for i in 1..nn(j)
       loop
          GET(FichierDonnees,Cd(j,i)) ;
       end loop ;
       SKIP_LINE(FichierDonnees) ;
    end loop ;
    for i in 1..NBR_INTERVALLES
    loop
       for j in 0..2
       loop
          GET(FichierDonnees,K(i,j)) ;
```

```
       end loop ;
       SKIP_LINE(FichierDonnees) ;
    end loop ;
    CLOSE(FichierDonnees) ;
    RhoBarre := Rhof/Rho ;
    A := 1.0 + 0.5*RhoBarre ;
    B := g*(1.0 - RhoBarre) ;
    C := 0.75*RhoBarre/d ;
end DONNEES_DU_FICHIER ;

procedure AFFICHAGE_DONNEES is
begin
    CLRSCR ;
    PUT_LINE(IMP,"            Donnees du probleme") ; NEW_LINE(IMP) ;
    PUT_LINE(IMP,"Conditions initiales :") ;
    PUT(IMP,"                       t0 = ") ;
    PUT(IMP,t0,6,3,0) ; NEW_LINE(IMP) ;
    PUT(IMP,"                       tMax = ") ;
    PUT(IMP,tMax,6,3,0) ; NEW_LINE(IMP) ;
    PUT(IMP,"                       z0 = ") ;
    PUT(IMP,z0,6,3,0) ; NEW_LINE(IMP) ;
    PUT(IMP,"                       v0 = ") ;
    PUT(IMP,v0,6,3,0) ; NEW_LINE(IMP,2) ;
    PUT_LINE(IMP,"Caracteristiques de la bille et du fluide :") ;
    NEW_LINE(IMP) ;
    PUT(IMP,"                       Rho = ") ;
    PUT(IMP,Rho,6,3,0) ; NEW_LINE(IMP) ;
    PUT(IMP,"                       Rhof = ") ;
    PUT(IMP,Rhof,6,3,0) ; NEW_LINE(IMP) ;
    PUT(IMP,"                       Nu = ") ;
    PUT(IMP,Nu,6,6,0) ; NEW_LINE(IMP) ;
    PUT(IMP,"                       g = ") ;
    PUT(IMP,g,6,3,0) ; NEW_LINE(IMP) ;
    PUT(IMP,"                       d = ") ;
    PUT(IMP,d,6,3,0) ; NEW_LINE(IMP) ;
    PAUSE ; CLRSCR ;
end AFFICHAGE_DONNEES ;

function Cd_j(K : in VECTEUR_COEFF ; j : in INTEGER ; x : in FLOAT)
             return FLOAT is
-- Evaluation du polynome definissant la fonction Cd sur l'intervalle j
begin
    return K(j,0) + K(j,1)*x + K(j,2)*x*x ;
end Cd_j ;

function F(v : in FLOAT) return FLOAT is
r, C_d : FLOAT ;
j : INTEGER ;
begin
    r := v*d/Nu ;
    if r = 0.0 then
       return (B - C*K(1,2)*Nu/(d*d))/A ;
    else
```

```
      j := 0 ;
      loop
         j := j + 1 ;
         exit when (r >= Re(j,1)) and (r < Re(j,nn(j))) ;
      end loop ;
      C_d := Cd_j(K,j,1.0/r) ;
      return (B - C*v*ABS(v)*C_d)/A ;
   end if ;
end F ;

procedure RK4(AFFICHER : in BOOLEAN ; Cz, Cv : in out COURBE) is
Pas, z_k1, z_k2, z_k3, z_k4, v_k1, v_k2, v_k3, v_k4 : FLOAT ;
begin
   Cz.uMin := t0 ; Cz.uMax := tMax ;
   Cv.uMin := t0 ; Cv.uMax := tMax ;
   Cz.vMin := 1.0E+38 ; Cz.vMax := -1.0E+38 ;
   Cv.vMin := 1.0E+38 ; Cv.vMax := -1.0E+38 ;
   Cz.u(0) := t0 ;
   Cv.u(0) := t0 ;
   Cz.v(0) := z0 ;
   Cv.v(0) := v0 ;
   Pas := (tMax - t0)/FLOAT(Cz.n) ;
   for i in 1..Cz.n
   loop
      Cz.u(i) := Cz.u(i - 1) + Pas ;
      Cv.u(i) := Cz.u(i) ;
      z_k1 := Pas*Cv.v(i - 1) ;
      v_k1 := Pas*F(Cv.v(i - 1)) ;
      z_k2 := Pas*(Cv.v(i - 1) + 0.5*v_k1) ;
      v_k2 := Pas*F(Cv.v(i - 1) + 0.5*v_k1) ;
      z_k3 := Pas*(Cv.v(i - 1) + 0.5*v_k2) ;
      v_k3 := Pas*F(Cv.v(i - 1) + 0.5*v_k2) ;
      z_k4 := Pas*(Cv.v(i - 1) + v_k3) ;
      v_k4 := Pas*F(Cv.v(i - 1) + v_k3) ;
    Cz.v(i) := Cz.v(i - 1) + (z_k1 + 2.0*z_k2 + 2.0*z_k3 + z_k4)/6.0 ;
      Cz.vMin := MIN(Cz.vMin,Cz.v(i)) ;
      Cz.vMax := MAX(Cz.vMax,Cz.v(i)) ;
    Cv.v(i) := Cv.v(i - 1) + (v_k1 + 2.0*v_k2 + 2.0*v_k3 + v_k4)/6.0 ;
      Cv.vMin := MIN(Cv.vMin,Cv.v(i)) ;
      Cv.vMax := MAX(Cv.vMax,Cv.v(i)) ;
      if (AFFICHER) then
         PUT(IMP,"t = ") ;
         PUT(IMP,Cz.u(i),6,3,0) ;
         PUT(IMP,"   z = ") ;
         PUT(IMP,Cz.v(i),6,3,0) ;
         PUT(IMP,"   v = ") ;
         PUT(IMP,Cv.v(i),6,3,0) ;
         NEW_LINE(IMP) ;
         if (i mod 20 = 0) then
            PAUSE ;
         end if ;
      end if ;
   end loop ;
```

```
end RK4 ;

procedure TRACE_POSITION_VITESSE is
tFenMin, tFenMax, zFenMin, zFenMax, vFenMin, vFenMax, tUnite, zUnite,
vUnite : FLOAT ;
Afficher : BOOLEAN ;
Cz, Cv : COURBE(200) ;
begin
  Afficher := LIRE_REPONSE("Voulez vous l'affichage des resultats? ") ;
    RK4(AFFICHER,Cz,Cv) ;
    tUnite := (tMax - t0)/20.0 ;
    zUnite := (Cz.vMax - Cz.vMin)/20.0 ;
    vUnite := (Cv.vMax - Cv.vMin)/20.0 ;
    tFenMin := Cz.uMin - tUnite ;
    tFenMax := Cz.uMax + tUnite ;
    zFenMin := Cz.vMin - zUnite ;
    zFenMax := Cz.vMax + zUnite ;
    vFenMin := Cv.vMin - vUnite ;
    vFenMax := Cv.vMax + vUnite ;
    INIT_GRAPHIQUE ;
    FENETRE(tFenMin,tFenMax,zFenMin,zFenMax) ;
    CADRE_GRAPHIQUE(
        X_MINIMUM,X_MINIMUM + ((X_MAXIMUM - X_MINIMUM)/2) - 1,
        Y_MINIMUM + MARGE,Y_MAXIMUM - MARGE) ;
    DEPLACE(Cz.u(0),Cz.v(0)) ;
    for i in 1..Cz.n
    loop
       TRACE(Cz.u(i),Cz.v(i)) ;
    end loop ;
    XY_AXES(0.0,0.0,tUnite,zUnite,TRUE) ;
    FENETRE(tFenMin,tFenMax,vFenMin,vFenMax) ;
    CADRE_GRAPHIQUE(
        X_MINIMUM + ((X_MAXIMUM - X_MINIMUM)/2) + 1,
        X_MAXIMUM, Y_MINIMUM + MARGE, Y_MAXIMUM - MARGE) ;
    DEPLACE(Cv.u(0),Cv.v(0)) ;
    for i in 1..Cv.n
    loop
        TRACE(Cv.u(i),Cv.v(i)) ;
    end loop ;
    XY_AXES(0.0,0.0,tUnite,vUnite,TRUE) ;
    TITRE("Graphe de z(t)                              Graphe de v(t)") ;
    SORTIE_GRAPHIQUE ;
end TRACE_POSITION_VITESSE ;

begin
    MODE_AFFICHAGE ;
  DONNEES_DU_FICHIER(t0,tMax,z0,v0,Rho,Rhof,Nu,g,d,A,B,C,nn,Re,Cd,K) ;
    AFFICHAGE_DONNEES ;
  PUT_LINE("Resolution du systeme differentiel donnant z(t) et v(t)") ;
    NEW_LINE ;
    TRACE_POSITION_VITESSE ;
    CLOSE(IMP) ;
end BILLE ;
```

7. Exercices

7.1 Le modèle « proies-prédateurs »

On considère le modèle « proies-prédateurs » du paragraphe 1, avec $r_1(t) = 2$; $r_2(t) = 1$; $b_1(t) = 0.5$; $b_2(t) = 0.25$; et les conditions initiales : $x(0) = y(0) = 1$ (sous-entendu une unité).

On tracera les graphes d'évolution de x et y en fonction de t et de y en fonction de x.

7.2 Le problème des trois corps

On considère trois corps C_1, C_2 et C_3 isolés dans l'univers et soumis à leurs attractions mutuelles. On veut déterminer le mouvement de deux de ces corps par rapport au troisième pris comme référence.

Dans un un repère orthonormé d'origine C_1, on note (X_2, Y_2, Z_2) les coordonnées de C_2 et (X_3, Y_3, Z_3) celles de C_3.

On note m_i la masse du corps C_i et $d_{i,j}$ la distance de C_i à C_j. Enfin G désigne la constante de gravitation universelle.

1°) En utilisant la loi fondamentale de la dynamique et la loi de la gravitation universelle, montrer que les équations du mouvement de C_2 et de C_3 sont :

$$\begin{cases}
X_2''(t) = G\cdot m_3\cdot\left\{\dfrac{X_3(t)-X_2(t)}{(d_{2,3})^3}-\dfrac{X_3(t)}{(d_{1,3})^3}\right\}-G\cdot(m_1+m_2)\cdot\dfrac{X_2(t)}{(d_{1,2})^3}\\
Y_2''(t) = G\cdot m_3\cdot\left\{\dfrac{Y_3(t)-Y_2(t)}{(d_{2,3})^3}-\dfrac{Y_3(t)}{(d_{1,3})^3}\right\}-G\cdot(m_1+m_2)\cdot\dfrac{Y_2(t)}{(d_{1,2})^3}\\
Z_2''(t) = G\cdot m_3\cdot\left\{\dfrac{Z_3(t)-Z_2(t)}{(d_{2,3})^3}-\dfrac{Z_3(t)}{(d_{1,3})^3}\right\}-G\cdot(m_1+m_2)\cdot\dfrac{Z_2(t)}{(d_{1,2})^3}\\
X_3''(t) = G\cdot m_2\cdot\left\{\dfrac{X_2(t)-X_3(t)}{(d_{2,3})^3}-\dfrac{X_2(t)}{(d_{1,2})^3}\right\}-G\cdot(m_1+m_3)\cdot\dfrac{X_3(t)}{(d_{1,3})^3}\\
Y_3''(t) = G\cdot m_2\cdot\left\{\dfrac{Y_2(t)-Y_3(t)}{(d_{2,3})^3}-\dfrac{Y_2(t)}{(d_{1,2})^3}\right\}-G\cdot(m_1+m_3)\cdot\dfrac{Y_3(t)}{(d_{1,3})^3}\\
Z_3''(t) = G\cdot m_2\cdot\left\{\dfrac{Z_2(t)-Z_3(t)}{(d_{2,3})^3}-\dfrac{Z_2(t)}{(d_{1,2})^3}\right\}-G\cdot(m_1+m_3)\cdot\dfrac{Z_3(t)}{(d_{1,3})^3}
\end{cases}$$

2°) On pose $y_1 = X_2$, $y_2 = X'_2$, ..., $y_{11} = Z_3$, $y_{12} = Z'_3$.

Ecrire le système différentiel que vérifient les fonctions y_1, ..., y_{12}.

3°) Ecrire un programme de résolution de ce système en utilisant la méthode de Runge-Kutta d'ordre 4.

On pourra utiliser les valeurs numériques suivantes correspondantes au système {Terre, Lune, Vaisseau spatial} : $m_1 = 5.98 \cdot 10^{24}$ Kg ; $m_2 = 7.35 \cdot 10^{22}$ Kg ; $m_3 = 10\,000$ Kg ; $X_2(0) = 383\,000$ Km ; $X'_2(0) = 0$; $Y_2(0) = 0$; $Y'_2(0) = 1.025$; $Z_2(0) = 0$; $Z'_2(0) = 0$; $X_3(0) = 1197$ Km ; $X'_3(0) = 8.49$ Km/s ; $Y_3(0) = -9928$; $Y'_3(0) = -2.455$; $Z_3(0) = 0$; $Z'_3(0) = 0.1$; $G = 6.672 \cdot 10^{-11}$ $m^3 \cdot Kg^{-1} \cdot s^{-2}$ (constante de gravitation).

7.3 Méthode de Stormer-Cowell

Dans ce problème, on étudie la méthode de Stormer-Cowell, qui est une méthode multipas explicite de résolution d'une équation différentielle du second ordre où y'(x) ne figure pas.

On considère l'équation différentielle, avec conditions initiales :

$$(1) \quad \begin{cases} y''(t) = f(x, y(x)) \quad (a < x < b) \\ y(a) = y_a \text{ et } y'(a) = y'_a \end{cases}$$

où y_a et y'_a sont donnés.

On pose $h = \dfrac{b-a}{n+1}$ et pour $i = 0, 1, ..., n+1$, $x_i = a + i \cdot h$.

On veut alors calculer des valeurs approchées y_i des $y(x_i)$ pour $i = 1, ..., n$.

1°) Montrer que si y définie sur un intervalle I est assez régulière, alors pour tout x dans I et h assez petit, on a :

$$(2) \quad y''(x) = \frac{y(x+h) - 2 \cdot y(x) + y(x-h)}{h^2} + O(h^2)$$

2°) A l'étape i du calcul, on suppose que l'on connaît des valeurs approchées y_{i-1} de $y(x_{i-1})$ et y_i de $y(x_i)$.

(a) Déduire de (2) une valeur approchée y_{i+1} de $y(x_{i+1})$.

(b) En déduire un algorithme de résolution de (1).

3°) (a) Donner la valeur de y"(a).

(b) En déduire une approximation d'ordre 2, y_1 de $y(x_1)$.

4°) Appliquer la méthode en question au problème :

$$(3) \quad \begin{cases} y''(x) = y(x) \quad (0 < x < 1) \\ y(0) = 1,\ y'(0) = 1 \end{cases}$$

On prendra h = 0.1.

Pour tester le comportement de ce schéma numérique sur de grands intervalles de temps, on l'applique au problème :

$$(4) \quad \begin{cases} y''(x) = -y(x) \quad (x > 0) \\ y(0) = \alpha,\ y'(0) = \beta \end{cases}$$

On se donne un pas $h > 0$ et on note $x_n = n \cdot h$, pour tout n dans N.

5°) Ecrire la relation de récurrence vérifiée par la suite $(y_n)_{n \in N}$.

6°) Pour quelles valeurs de q dans C, la suite définie par $u_n = q^n$, vérifie-t-elle cette récurrence ?

7°) En notant q_1 et q_2 les racines de l'équation obtenue en 2°), montrer que la suite (y_n) est bornée si, et seulement si $|q_i| \leq 1$, pour i = 1, 2.
8°) En déduire que la solution numérique trouvée ne peut être bornée que si h est inférieur à une valeur critique que l'on précisera.

7.4 La méthode du tir pour les équations différentielles d'ordre 2

On considère l'équation différentielle :

$$(1) \quad y'' = f(x,y,y') \qquad (x \in [a,b])$$

avec les conditions aux limites :

$$y(a) = A,\ y(b) = B$$

On suppose que cette équation admet une unique solution y deux fois continûment dérivable.

En posant, de façon arbitraire, $y'(a) = \alpha$, on note $y(\alpha,x)$ la solution de (1) avec les conditions initiales :

$$y(a) = A,\ y'(a) = \alpha$$

1°) Expliquer comment construire une suite $(\alpha_n)_{n\geq 0}$ permettant « d'ajuster le tir », c'est-à-dire, sous de bonnes conditions, qui converge vers α solution de $y(\alpha,b) - B = 0$.

Si cette suite converge vers α, la solution $y(\alpha,x)$ est alors solution de problème posé.
2°) Ecrire les 2 premières itérations de la méthode pour :

$$f(x,y,y') = \frac{2}{x^3}$$

sur [1,2] avec y(1) = 1, y(2) = 0.5 et $\alpha_0 = -0.5$.
3°) Ecrire, une procédure de résolution d'un problème du type (1) par la méthode du tir.

8. Programmation Ada

Ci-dessous on décrit quatre paquetages génériques permettant de résoudre une équation différentielle d'ordre 1, un système de p équations différentielles d'ordre 1, une équation différentielle d'ordre q et un système de p équations différentielles d'ordre q.

En commentaire on indique comment utiliser les fonctions définissant les équations différentielles.

8.1 Spécification du paquetage EQUADIFF_11_GENERIQUE

Une équation différentielle d'ordre 1 s'écrit sous la forme :

y' = f(x,y) x dans (a,b)
y(a) donné dans R

```
with COMMON_MATRIX ;
use COMMON_MATRIX ;
```

```
generic
  with function f11(x, y : in FLOAT) return FLOAT ;
package EQUADIFF_11_GENERIQUE is
procedure RESOLUTION_RK4_11(a, b, ya : in FLOAT ;
             AFFICHER : in BOOLEAN ; x, y : in out VECTEUR) ; -- § 3.4
procedure RESOLUTION_RK4_11_CONTROLE(a, b, ya : in FLOAT ;
             AFFICHER : in BOOLEAN ; x, y : in out VECTEUR) ; -- § 4.3
end EQUADIFF_11_GENERIQUE ;
```

8.2 Spécification du paquetage EQUADIFF_P1_GENERIQUE

Un système de p équations différentielles d'ordre 1 s'écrit sous la forme :

y(1)' = f(1)(x,y(1),...,y(p))
..... x dans (a,b)
y(p)' = f(p)(x,y(1),...,y(p))
y(1)(a), ..., y(p)(a) donnes dans R

```
with COMMON_MATRIX, MATH2 ;
use COMMON_MATRIX, MATH2 ;
generic
  with function fp1(j : in POSITIVE ; x : in FLOAT ; y : VARIABLE_2)
return FLOAT ;
package EQUADIFF_P1_GENERIQUE is
procedure RESOLUTION_RK4_P1(a, b : in FLOAT ;
             Y_INITIAL : in VARIABLE_2 ; AFFICHER : in BOOLEAN ;
             x : in out VECTEUR ; y : in out MATRICE) ; -- § 3.5
procedure RESOLUTION_RK4_P1_CONTROLE(a, b : in FLOAT ;
             Y_INITIAL : in VARIABLE_2 ; AFFICHER : in BOOLEAN ;
             x : in out VECTEUR ; y : in out MATRICE) ; -- § 4.3
end EQUADIFF_P1_GENERIQUE ;
```

8.3 Spécification du paquetage EQUADIFF_1Q_GENERIQUE

Une équation différentielle d'ordre q s'écrit sous la forme :

$y^{(q)} = f(x,y(0),y(1),...,y(q-1))$ x dans (a,b)
y(0)(a), ..., y(q–1)(a) donnés dans R

y(j) représente la dérivée d'ordre j de y pour j = 0, ..., q – 1.

```
with COMMON_MATRIX, MATH2 ;
use COMMON_MATRIX, MATH2 ;
generic
  with function f1Q(x : in FLOAT ; y : VARIABLE_2) return FLOAT ;
package EQUADIFF_1Q_GENERIQUE is

procedure RESOLUTION_RK4_1Q(a, b : in FLOAT ;
             Y_INITIAL : in VARIABLE_2 ; AFFICHER : in BOOLEAN ;
             x, y : in out VECTEUR) ; -- § 3.6
procedure RESOLUTION_RK4_1Q_CONTROLE(a, b : in FLOAT ;
             Y_INITIAL : in VARIABLE_2 ; AFFICHER : in BOOLEAN ;
             x, y : in out VECTEUR) ; -- § 4.3
end EQUADIFF_1Q_GENERIQUE ;
```

8.4 Spécification du paquetage EQUADIFF_PQ_GENERIQUE

Un système différentielle de p équations d'ordre q s'écrit sous la forme :

$y(1)^{(q)} = f(1)(x,y(1,0),...,y(1, q -1),y(2,0),...,y(2, q - 1),..., y(p,0),...,y(p,q - 1))$

.....

$y(p)^{(q)} = f(p)(x,y(1,0),...,y(1,q -1),y(2,0),...,y(2,q - 1),..., y(p,0),...,y(p,q - 1))$

Où y(j,i) représente la dérivée d'ordre i de y(j).

FPQ(j,x,y) représente équation N° j avec : y(q*(j – 1) + i + 1) représentant la dérivée d'ordre i de y(j).

```
with COMMON_MATRIX, MATH2 ;
use COMMON_MATRIX, MATH2 ;
generic
   with function FPQ(j : in POSITIVE ; x : in FLOAT ; y : VARIABLE_2)
           return FLOAT ;
package EQUADIFF_PQ_GENERIQUE is
procedure RESOLUTION_RK4_PQ(a, b : in FLOAT ; Y_INITIAL : in MATRICE ;
             AFFICHER : in BOOLEAN ; -- § 3.7
             x : in out VECTEUR ; y : in out MATRICE) ;
procedure RESOLUTION_RK4_PQ_CONTROLE(a, b : in FLOAT ;
             Y_INITIAL : in MATRICE ; AFFICHER : in BOOLEAN ;
             x : in out VECTEUR ; y : in out MATRICE) ; -- 4.3
end EQUADIFF_PQ_GENERIQUE ;
```

8.5 Démonstration du paquetage EQUADIFF_11_GENERIQUE

```
with TEXT_IO, IIO, FIO, CRT, COMMON_MATH0, MATH0, MATH1, MATH2,
     MATH3, COMMON_MATRIX, EQUADIFF_11_GENERIQUE ;
use TEXT_IO, IIO, FIO, CRT, COMMON_MATH0, MATH0, MATH1, MATH2,
     MATH3, COMMON_MATRIX ;

procedure DEM_RK11 is
N_MAX : constant INTEGER := 500 ;
n : INTEGER ;
a, b, ya : FLOAT ;
AFFICHER, CONTROLE_DU_PAS : BOOLEAN ;
x, y : VECTEUR(0..N_MAX) ;
X_UNITE, Y_UNITE, X_FEN_MIN, X_FEN_MAX, Y_FEN_MIN, Y_FEN_MAX : FLOAT ;
FF : FONCTION ;

function f(x, y : in FLOAT) return FLOAT is
begin
    return EVALUE(FF,x,y) ;
end f ;

package EQUADIFF_11_F is new EQUADIFF_11_GENERIQUE(f) ;
use EQUADIFF_11_F ;
begin
    MODE_AFFICHAGE ;
    GET_LINE(FF,"Entrez l'equation :  y' = f(x,y) = ") ;
    NEW_LINE ;
    ENTRER_REEL(a,"Valeur de a : ") ;
```

```
    ENTRER_REEL_BORNE(a,1.0E+6,b,"Valeur de b : ") ;
    ENTRER_REEL(ya,"Valeur initiale en a : ") ;
    NEW_LINE ;
    NEW_LINE ;
  AFFICHER := LIRE_REPONSE("Voulez-vous l'affichage des resultats? ") ;
    CONTROLE_DU_PAS := LIRE_REPONSE("Methode avec controle du pas? ") ;
    CLRSCR ;
    if (CONTROLE_DU_PAS) then
       RESOLUTION_RK4_11_CONTROLE(a,b,ya,AFFICHER,x,y) ;
    else
       RESOLUTION_RK4_11(a,b,ya,AFFICHER,x,y) ;
    end if ;
    PAUSE ;
    CLRSCR ;
  if LIRE_REPONSE("Voulez vous le trace de la solution trouvee? ") then
       n := x'LAST ;
       X_UNITE := (b - a)/20.0 ;
       X_FEN_MIN := a - X_UNITE ;
       X_FEN_MAX := b + X_UNITE ;
       Y_FEN_MIN := 1.0E+38 ;
       Y_FEN_MAX := -1.0E+38 ;
       for i in y'range
       loop
          Y_FEN_MIN := MIN(Y_FEN_MIN,y(i)) ;
          Y_FEN_MAX := MAX(Y_FEN_MAX,y(i)) ;
          if (x(i) > b) then
             n := i - 1 ;
             exit ;
          end if ;
       end loop ;
       Y_UNITE := (Y_FEN_MAX - Y_FEN_MIN)/20.0 ;
       Y_FEN_MIN := Y_FEN_MIN - Y_UNITE ;
       Y_FEN_MAX := Y_FEN_MAX + Y_UNITE ;
       INIT_GRAPHIQUE ;
       FENETRE_GRAPHIQUE(X_FEN_MIN,X_FEN_MAX,Y_FEN_MIN,Y_FEN_MAX) ;
       XY_AXES(a,Y_FEN_MIN + Y_UNITE,X_UNITE,Y_UNITE,TRUE) ;
       DEPLACE(x(x'FIRST),y(y'FIRST)) ;
       for i in x'FIRST + 1..n
       loop
          TRACE(x(i),y(i)) ;
       end loop ;
       TITRE("Graphe de la solution approchee y = y(x)") ;
       SORTIE_GRAPHIQUE ;
    end if ;
    CLOSE(IMP) ;
end DEM_RK11 ;
```

8.6 Démonstration du paquetage EQUADIFF_P1_GENERIQUE

```
with TEXT_IO, IIO, CRT, COMMON_MATH0, MATH0, MATH1, GRAPH,
     MATH2, MATH3, COMMON_MATRIX, EQUADIFF_P1_GENERIQUE ;
use TEXT_IO, IIO, CRT, COMMON_MATH0, MATH0, MATH1, GRAPH,
    MATH2, MATH3, COMMON_MATRIX ;
```

```
procedure DEM_RKP1 is
N_MAX: constant INTEGER := 500 ;
MAX_FONCTIONS : constant INTEGER := 10 ;
subtype INDICE_FCT is INTEGER range 1..MAX_FONCTIONS ;
type SYSTEME_DIFF is array(INDICE_FCT range <>) of FONCTION ;
a, b : FLOAT ;
AFFICHER, CONTROLE_DU_PAS : BOOLEAN ;
p : INTEGER ;
X_UNITE, Y_UNITE, X_FEN_MIN, X_FEN_MAX, Y_FEN_MIN, Y_FEN_MAX : FLOAT ;
x : VECTEUR(0..N_MAX) ;
n : INTEGER ;
begin
    MODE_AFFICHAGE ;
    PUT_LINE(" Le systeme d'equations est ecrit sous la forme :") ;
    NEW_LINE ;
    PUT_LINE("   y' = f (x,y , ...,y   )") ;
    PUT_LINE("    j    j    1       p") ;
    NEW_LINE ;
    PUT(" pour j = 1, ..., p, ou 1 <= p <= ") ;
    PUT(MAX_FONCTIONs) ; NEW_LINE ;
    ENTRER_ENTIER_BORNE(1,MAX_FONCTIONS,p,"Nombre d'equations : ") ;
    NEW_LINE ;
    declare
       FF : SYSTEME_DIFF(1..p) ;
       ya : VARIABLE_2(1..p) ;
       y : MATRICE(0..N_MAX,1..p) ;
       function f(j : in POSITIVE ; x : in FLOAT ; y : in VARIABLE_2)
          return FLOAT is
       begin
          return EVALUE(FF(j),x,y) ;
       end f ;
       package EQUADIFF_P1_F is new EQUADIFF_P1_GENERIQUE(F) ;
       use EQUADIFF_P1_F ;
    begin
       for j in YA'range
       loop
          PUT("Equation Numero ") ; PUT(j,2) ; NEW_LINE ;
          PUT("f(x,") ;
          for k in 1..p - 1
          loop
             PUT("y") ; PUT(k,2) ; PUT(",") ;
          end loop ;
          PUT("y") ; PUT(p,2) ; PUT_LINE(") = ") ;
          GET_LINE(ff(j),"   ") ;
       end loop ;
       NEW_LINE ;
       PUT_LINE(" Bornes de l'intervalle") ;
       NEW_LINE ;
       ENTRER_REEL(a,"Valeur de a : ") ;
       ENTRER_REEL_BORNE(a,1.0E+6,b,"Valeur de b : ") ;
       for j in ya'range
       loop
          PUT("Valeur initiale y") ; PUT(j,2) ; PUT_LINE("(a)") ;
```

```
          ENTRER_REEL(ya(j),"  y = ") ;
       end loop ;
       NEW_LINE ;
  AFFICHER := LIRE_REPONSE("Voulez-vous l'affichage des resultats? ") ;
  CONTROLE_DU_PAS := LIRE_REPONSE("Methode avec controle du pas? ") ;
       CLRSCR ;
       if (CONTROLE_DU_PAS) then
          RESOLUTION_RK4_P1_CONTROLE(a,b,ya,AFFICHER,x,y) ;
       else
          RESOLUTION_RK4_P1(a,b,ya,AFFICHER,x,y) ;
       end if ;
       PAUSE ; CLRSCR ;
  if LIRE_REPONSE("Voulez vous le trace de la solution trouvee? ") then
     n := x'LAST ;
     INIT_GRAPHIQUE ;
     X_UNITE := (b - a)/20.0 ;
     X_FEN_MIN := a - X_UNITE ;
     X_FEN_MAX := b + X_UNITE ;
     for j in YA'range
     loop
          Y_FEN_MIN := 1.0E+38 ;
          Y_FEN_MAX := -1.0E+38 ;
          for i in y'range(1)
          loop
             Y_FEN_MIN := MIN(Y_FEN_MIN,y(i,j)) ;
             Y_FEN_MAX := MAX(Y_FEN_MAX,y(i,j)) ;
             if (x(i) > b) then
                n := i - 1 ;
                exit ;
             end if ;
          end loop ;
          Y_UNITE := (Y_FEN_MAX - Y_FEN_MIN)/20.0 ;
          Y_FEN_MIN := Y_FEN_MIN - Y_UNITE ;
          Y_FEN_MAX := Y_FEN_MAX + Y_UNITE ;
          FENETRE_GRAPHIQUE(X_FEN_MIN,X_FEN_MAX,Y_FEN_MIN,Y_FEN_MAX) ;
          XY_AXES(a,Y_FEN_MIN + Y_UNITE,X_UNITE,Y_UNITE,TRUE) ;
          DEPLACE(x(x'FIRST),y(y'FIRST,j)) ;
          for i in x'FIRST + 1..n
          loop
             TRACE(x(i),y(i,j)) ;
          end loop ;
          if (j < p) then
             PAUSE_GRAPHIQUE ;
             CLEAR_SCREEN ;
          else
             SORTIE_GRAPHIQUE ;
          end if ;
     end loop ;
  end if ;
    end ;
    CLOSE(IMP) ;
end DEM_RKP1 ;
```

8.7 Démonstration du paquetage EQUADIFF_1Q_GENERIQUE

```
with TEXT_IO, IIO, CRT, COMMON_MATH0, MATH0, MATH1, MATH2,
     MATH3, COMMON_MATRIX, EQUADIFF_1Q_GENERIQUE ;
use TEXT_IO, IIO, CRT, COMMON_MATH0, MATH0, MATH1, MATH2,
    MATH3, COMMON_MATRIX ;

procedure DEM_RK1Q is
N_MAX: constant INTEGER := 500 ;
MAX_ORDRE : constant INTEGER := 10 ;
a, b : FLOAT ;
q : INTEGER ;
AFFICHER, CONTROLE_DU_PAS : BOOLEAN ;
FF : FONCTION ;
x, y : VECTEUR(0..N_MAX) ;
n : INTEGER := x'LAST ;
X_UNITE, Y_UNITE, X_FEN_MIN, X_FEN_MAX, Y_FEN_MIN, Y_FEN_MAX : FLOAT ;

function f(x : in FLOAT ; y : in VARIABLE_2) return FLOAT is
begin
    return EVALUE(FF,x,y) ;
end f ;

package EQUADIFF_1Q_F is new EQUADIFF_1Q_GENERIQUE(F) ;
use EQUADIFF_1Q_F ;

begin
    MODE_AFFICHAGE ;
    PUT_LINE(" L'equation est ecrite sous la forme :") ;
    PUT_LINE("    (q)               (q-1)") ;
    PUT_LINE("   y    = f(x,y,...,y     )") ;
    PUT_LINE(" ") ;
PUT_LINE("La derivee d'ordre i est notee yi, pour i = 0, ..., q - 1") ;
    NEW_LINE ;
    ENTRER_ENTIER_BORNE(1,MAX_ORDRE,q,"Ordre de l'equation : ") ;
    NEW_LINE ;
    declare
       ya : VARIABLE_2(0..q-1) ;
    begin
       PUT("Equation f(x,y0,") ;
       for k in 1..q - 2
       loop
          PUT("y") ; PUT(k,2) ; PUT(",") ;
       end loop ;
       PUT("y") ; PUT(q-1,2) ; PUT_LINE(") = ") ;
       GET_LINE(FF,"   ") ;
       NEW_LINE ;
       ENTRER_REEL(a,"Valeur de a : ") ;
       ENTRER_REEL_BORNE(a,1.0E+6,b,"Valeur de b : ") ;
       for k in 0..q - 1
       loop
          PUT("Valeur initiale de la derivee d'ordre ") ; PUT(k,2) ;
          NEW_LINE ;
```

```
            ENTRER_REEL(ya(k),"   ") ;
         end loop ;
         NEW_LINE ;
   AFFICHER := LIRE_REPONSE("Voulez-vous l'affichage des resultats? ") ;
   CONTROLE_DU_PAS := LIRE_REPONSE("Methode avec controle du pas? ") ;
         CLRSCR ;
         if (CONTROLE_DU_PAS) then
            RESOLUTION_RK4_1Q_CONTROLE(a,b,ya,AFFICHER,x,y) ;
         else
            RESOLUTION_RK4_1Q(a,b,ya,AFFICHER,x,y) ;
         end if ;
         PAUSE ; CLRSCR ;
   if LIRE_REPONSE("Voulez vous le trace de la solution trouvee? ") then
      X_UNITE := (b - a)/20.0 ;
      X_FEN_MIN := a - X_UNITE ;
      X_FEN_MAX := b + X_UNITE ;
      Y_FEN_MIN := 1.0E+38 ;
      Y_FEN_MAX := -1.0E+38 ;
      for i in y'range
      loop
         Y_FEN_MIN := MIN(Y_FEN_MIN,y(i)) ;
         Y_FEN_MAX := MAX(Y_FEN_MAX,y(i)) ;
         if (x(i) > b) then
            n := i - 1 ;
            exit ;
         end if ;
      end loop ;
      Y_UNITE := (Y_FEN_MAX - Y_FEN_MIN)/20.0 ;
      Y_FEN_MIN := Y_FEN_MIN - Y_UNITE ;
      Y_FEN_MAX := Y_FEN_MAX + Y_UNITE ;
      INIT_GRAPHIQUE ;
       FENETRE_GRAPHIQUE(X_FEN_MIN,X_FEN_MAX,Y_FEN_MIN,Y_FEN_MAX) ;
      XY_AXES(a,Y_FEN_MIN + Y_UNITE,X_UNITE,Y_UNITE,TRUE) ;
      DEPLACE(x(x'FIRST),y(y'FIRST)) ;
      for i in x'FIRST + 1..n
      loop
         TRACE(x(i),y(i)) ;
      end loop ;
      TITRE("Graphe de la solution approchee y = y(x)") ;
      SORTIE_GRAPHIQUE ;
   end if ;
     end ;
     CLOSE(IMP) ;
end DEM_RK1Q ;
```

8.8 Démonstration du paquetage EQUADIFF_PQ_GENERIQUE

```
with TEXT_IO, IIO, CRT, COMMON_MATH0, MATH0, MATH1, GRAPH,
     MATH2, MATH3, COMMON_MATRIX, EQUADIFF_PQ_GENERIQUE ;
use TEXT_IO, IIO, CRT, COMMON_MATH0, MATH0, MATH1, GRAPH,
    MATH2, MATH3, COMMON_MATRIX ;
```

```
procedure DEM_RKPQ is
N_MAX : constant INTEGER := 500 ;
MAX_FONCTIONS : constant INTEGER := 10 ;
subtype INDICE_FCT is INTEGER range 1..MAX_FONCTIONS ;
type SYSTEME_DIFF is array(INDICE_FCT range <>) of FONCTION ;
a, b : FLOAT ;
AFFICHER, CONTROLE_DU_PAS : BOOLEAN ;
p, q, n : INTEGER ;
X_UNITE, Y_UNITE, X_FEN_MIN, X_FEN_MAX, Y_FEN_MIN, Y_FEN_MAX : FLOAT ;
x : VECTEUR(0..N_MAX) ;
begin
    MODE_AFFICHAGE ;
    PUT_LINE(" Le systeme d'equations est ecrit sous la forme :") ;
    NEW_LINE ;
    PUT_LINE("    (q)                (q-1)           (q-1) ") ;
    PUT_LINE("   y    = f (x,y , ...,y ,......,y ,...,y )") ;
    PUT_LINE("    j      j    1       1         p      p ") ;
    NEW_LINE ;
    PUT_LINE(" Pour j = 1, ..., p et i = 0, ..., q - 1, ") ;
    PUT_LINE("la derivee d'ordre i de y(j) est notee y(10*j + i)") ;
    NEW_LINE ;
    ENTRER_ENTIER_BORNE(1,MAX_FONCTIONS,p,"Nombre d'equations : ") ;
    NEW_LINE ;
    ENTRER_ENTIER_BORNE(1,MAX_FONCTIONS,q,"Ordre du systeme : ") ;
    NEW_LINE ;
    declare
       FF : SYSTEME_DIFF(1..p) ;
-- Pour p = q = 2, FF(j,x,y10,y11,y12,y20,y21) represente l'equation
-- y(j)''    = f(j)(x,y(1),y(1)',y(2),y(2)')
       ya : MATRICE(1..p,0..q-1) ;
       y : MATRICE(0..N_MAX,1..p) ;
       yy : VARIABLE_2(10..10*p + q - 1) ;
       function f(j : in POSITIVE ; x : in FLOAT ; y : in VARIABLE_2)
          return FLOAT is
       begin
          for j in 1..p
          loop
            for i in 0..q-1
            loop
               yy(10*j + i) := y(q*(j - 1) + i + 1) ;
          end loop ;
          end loop ;
          return EVALUE(FF(j),x,yy) ;
       end f ;

       package EQUADIFF_PQ_F is new EQUADIFF_PQ_GENERIQUE(F) ;
       use EQUADIFF_PQ_F ;

    begin
       for j in 1..p
       loop
          PUT("Equation Numero ") ; PUT(j,2) ; NEW_LINE ;
          PUT("f(x,") ;
```

```
         for k in 1..p
         loop
            for i in 0..q - 1
            loop
               PUT("y") ; PUT(10*k + i,2) ;
               if ((k /= p) or (i /= q - 1)) then
                  PUT(",") ;
               else
                  PUT_LINE(") = ") ;
               end if ;
            end loop ;
         end loop ;
         GET_LINE(ff(j),"   ") ;
      end loop ;
      NEW_LINE ;
      PUT_LINE(" Bornes de l'intervalle") ;
      NEW_LINE ;
      ENTRER_REEL(a,"Valeur de a : ") ;
      ENTRER_REEL_BORNE(a,1.0E+6,b,"Valeur de b : ") ;
      for j in 1..p
      loop
         for i in 0..q - 1
         loop
            PUT("Valeur initiale de la derivee d'ordre") ; PUT(i,2) ;
            PUT(" de y(") ; PUT(j,2) ; PUT_LINE(")") ;
            ENTRER_REEL(ya(j,i),"  y = ") ;
         end loop ;
      end loop ;
      NEW_LINE ;
AFFICHER := LIRE_REPONSE("Voulez-vous l'affichage des resultats? ") ;
CONTROLE_DU_PAS := LIRE_REPONSE("Methode avec controle du pas? ") ;
      CLRSCR ;
      if (CONTROLE_DU_PAS) then
         RESOLUTION_RK4_PQ_CONTROLE(a,b,ya,AFFICHER,x,y) ;
      else
         RESOLUTION_RK4_PQ(a,b,ya,AFFICHER,x,y) ;
      end if ;
      PAUSE ; CLRSCR ;
if LIRE_REPONSE("Voulez vous le trace de la solution trouvee? ") then
   n := x'LAST ;
   INIT_GRAPHIQUE ;
   X_UNITE := (b - a)/20.0 ;
   X_FEN_MIN := a - X_UNITE ;
   X_FEN_MAX := b + X_UNITE ;
   for j in 1..p
   loop
     Y_FEN_MIN := 1.0E+38 ;
     Y_FEN_MAX := -1.0E+38 ;
     for i in y'range(1)
     loop
       Y_FEN_MIN := MIN(Y_FEN_MIN,y(i,j)) ;
       Y_FEN_MAX := MAX(Y_FEN_MAX,y(i,j)) ;
       if (x(i) > b) then
```

```
          n := i - 1 ;
          exit ;
        end if ;
      end loop ;
      Y_UNITE := (Y_FEN_MAX - Y_FEN_MIN)/20.0 ;
      Y_FEN_MIN := Y_FEN_MIN - Y_UNITE ;
      Y_FEN_MAX := Y_FEN_MAX + Y_UNITE ;
       FENETRE_GRAPHIQUE(X_FEN_MIN,X_FEN_MAX,Y_FEN_MIN,Y_FEN_MAX) ;
      XY_AXES(a,Y_FEN_MIN + Y_UNITE,X_UNITE,Y_UNITE,TRUE) ;
      DEPLACE(x(x'FIRST),y(y'FIRST,j)) ;
      for i in x'FIRST + 1..n
      loop
         TRACE(x(i),y(i,j)) ;
       end loop ;
       if (j < p) then
          PAUSE_GRAPHIQUE ;
          CLEAR_SCREEN ;
       else
          SORTIE_GRAPHIQUE ;
       end if ;
    end loop ;
  end if ;
    end ;
    CLOSE(IMP) ;
end DEM_RKPQ ;
```

CHAPITRE 8

Méthode des différences finies

1. Problème de Dirichlet linéaire en dimension un

1.1 Introduction

On s'intéresse ici aux équations différentielles linéaires d'ordre 2, avec conditions aux limites, du type (*problème de Dirichlet*) :

$$(1)\quad \begin{cases} y''(x)+\alpha(x)\cdot y'(x)+\beta(x)\cdot y(x)=f(x) \quad (a < x < b) \\ y(a)=y_a \text{ et } y(b)=y_b \end{cases}$$

où a, b, f sont des applications continues de [a,b] dans R et la fonction inconnue y est de classe C^2 sur [a,b].

Remarques — (i) On sait que l'ensemble des solutions de l'équation différentielle $y''(x)+\alpha(x)\cdot y'(x)+\beta(x)\cdot y(x)=f(x)$ est un espace affine de dimension deux. En particulier, une solution est uniquement déterminée par les conditions initiales y(a) et y'(a) donnés. Mais pour le problème qui nous intéresse, les conditions aux limites portent sur y(a) et y(b), ce qui est différent et le résultat précédent n'assure pas l'existence et l'unicité de solutions.

Exemple — Considérons le problème :

$$\begin{cases} y''(x)+r^2\cdot y(x)=1 \quad (0 < x < 1) \\ y(0)=y_0 \text{ et } y(1)=y_1 \end{cases}$$

La solution générale est donnée par :

$$y(x) = c\cdot\cos(r\cdot x) + d\cdot\sin(r\cdot x) + \frac{1}{r^2}$$

où les constantes c et d sont définies par :

$$\begin{cases} c = y_0 - \dfrac{1}{r^2} \\ c \cdot \cos(r) + d \cdot \sin(r) = y_1 - \dfrac{1}{r^2} \end{cases}$$

Le déterminant de ce système étant D = sin(r), pour r = 2·k·π avec k dans Z, il y aura une infinité de solutions (si $y_0 = y_1$) ou pas de solutions du tout (si y_0 différent de y_1).
(ii) Sauf dans des cas très particuliers, on ne sait pas résoudre de façon explicite les équations du type (1). On cherchera donc des procédés numériques qui permettent de donner des valeurs approchées d'une solution en certains points.

Problèmes posés par (1)
- Existence de solutions ;
- Elaboration de méthodes numériques ;
- Qualité de l'approximation d'une solution par une méthode donnée (conditions de convergence ; vitesse de convergence).

1.2 Théorème d'existence et d'unicité de solutions du problème de Dirichlet linéaire

On peut montrer le :

Théorème **:** Si $\beta(x) < 0$ pour tout x dans [a,b], alors le problème de Dirichlet admet une unique solution y dans $C^2([a,b])$.

Contrairement au problème de Cauchy l'existence et l'unicité n'est pas assurée dans tous les cas.

On supposera donc, dans tout ce qui suit que $\beta(x) < 0$ sur [a,b].

1.3 Forme canonique de l'équation de Dirichlet. Le problème de Poisson

On suppose que α est de classe C^1 et on note A(x) une primitive de $\alpha/2$ sur [a,b].

Soit y solution de (1). En posant, pour tout x dans [a,b] :

$$z(x) = y(x) \cdot e^{A(x)} \qquad (\textit{transformation de Liouville})$$

la fonction z est solution du problème de Poisson :

$$\begin{cases} z''(x) + p(x) \cdot z(x) = g(x) \quad (a < x < b) \\ z(a) = z_a \text{ et } z(b) = z_b \end{cases}$$

où :

$$\begin{cases} p(x) = \beta(x) - \dfrac{\alpha'(x)}{2} - \dfrac{\alpha^2(x)}{4} \\ g(x) = f(x) \cdot e^{A(x)} \\ z_a = y_a \cdot e^{A(a)} \text{ et } z_b = y_b \cdot e^{A(b)} \end{cases}$$

On peut donc se contenter d'étudier l'équation de Poisson, mais si on ne connaît pas de primitive de α la transformation de Liouville n'a pas d'intérêt pratique.

1.4 Résolution approchée du problème de Dirichlet par discrétisation

En vue de discrétiser l'équation (1), on coupe l'intervalle [a,b] en n + 1 intervalles égaux (on dit aussi qu'on fait un *maillage uniforme* de [a,b]) en posant :

$$h = \frac{b-a}{n+1},\ x_k = a + k \cdot h \quad \text{pour } k = 0, 1, ..., n+1.$$

On dit que les x_k sont les *noeuds du maillage*.

On cherche alors, pour tout k = 1 , 2 ,... , n une valeur approchée de $y(x_k)$. Ces valeurs seront consignées dans un vecteur

$$u = \begin{pmatrix} u_1 \\ . \\ . \\ u_n \end{pmatrix}$$

Une fois ces valeurs approchées trouvées, on approxime la fonction y solution de (1) par la fonction affine par morceaux définie sur [a,b] par $\varphi(x_k) = u_k$ et φ affine sur $[x_k, x_{k+1}]$ pour tout k = 0,... ,n, où on a posé $u_0 = y_a$ et $u_{n+1} = y_b$.

Le graphe de φ sera alors une approximation de la courbe intégrale de (1).

On cherchera également à donner une majoration de l'erreur :

$$\text{Sup}\left\{ \left| y(x) - \varphi(x) \right|;\ 0 \le x \le 1 \right\}$$

2. Approximations des dérivées d'une fonction par différences finies

Soit f : [a,b] → R une fonction de classe C^4 et x_0 un point de [a,b].

2.1 Approximation de f'(x_0) par différence finie centrée

La première idée pour approximer $f'(x_0)$ est de prendre :

$$f'(x_0) \cong \frac{f(x_0 + h) - f(x_0)}{h} \qquad (\textit{différence finie progressive})$$

ou

$$f'(x_0) \cong \frac{f(x_0) - f(x_0 - h)}{h} \qquad (\textit{différence finie régressive})$$

où $h > 0$ est choisi assez petit.

Avec la formule de Taylor-Lagrange, on a :

$$\left| f'(x_0) - \frac{f(x_0+h)-f(x_0)}{h} \right| = \left| \frac{f''(x_0+\theta\cdot h)}{2} \right| \cdot h \quad (0 < \theta < 1)$$

c'est-à-dire que l'erreur commise est d'ordre 1 en h.

De même avec la différence finie régressive.

Mais en regardant de plus près, on peut obtenir une approximation qui sera d'ordre 2 en h.

Avec la formule de Taylor à l'ordre 3, on peut écrire :

$$\frac{f(x_0+h)-f(x_0-h)}{2\cdot h} = f'(x_0) + \frac{f'''(x_0+\theta_1\cdot h)+f'''(x_0-\theta_2\cdot h)}{12}\cdot h^2 \quad (0 < \theta_1, \theta_2 < 1)$$

De plus, en appliquant le théorème des valeurs intermédiaires à la fonction :

$$\varphi:\ t \rightarrow \varphi(t) = f'''(x_0+\theta_1\cdot h) + f'''(x_0-\theta_2\cdot h) - 2\cdot f'''(t)$$

on peut écrire :

$$\frac{f(x_0+h)-f(x_0-h)}{2\cdot h} = f'(x_0) + \frac{f'''(x_0+\theta\cdot h)}{6}\cdot h^2 \quad (-1 < \theta < 1)$$

Donc la différence finie centrée $\dfrac{f(x_0+h)-f(x_0-h)}{2\cdot h}$ donne une approximation d'ordre 2 en h de $f'(x_0)$.

2.2 Approximation de f''(x_0) par différence finie centrée

Si, pour approximer la dérivée seconde, on applique deux fois ce qui précède on n'obtient qu'une approximation d'ordre 1 de cette dérivée.

Là encore, en regardant de plus près, on peut obtenir une approximation d'ordre 2.

Avec la formule de Taylor à l'ordre 4, on peut écrire :

$$\frac{f(x_0+h)-2\cdot f(x_0)+f(x_0-h)}{h^2} = f''(x_0) + \frac{f^{(4)}(x_0+\theta\cdot h)}{12}\cdot h^2 \quad (-1 < \theta < 1)$$

Donc la différence finie centrée $\dfrac{f(x_0+h)-2\cdot f(x_0)+f(x_0-h)}{h^2}$ donne une approximation d'ordre 2 en h de $f''(x_0)$.

3. Résolution approchée du problème de Dirichlet par la méthode des différences finies

3.1 Discrétisation du problème de Dirichlet

On reprend les notations du paragraphe (1.4).

En écrivant l'équation (1) en tous les points x_k (k = 1 , ..., n) et en utilisant les résultats du (2.2), on voit que le vecteur y de coordonnées $y_k = y(x_k)$ est solution du système :

$$A \cdot y = v + \varepsilon \qquad (3)$$

où A est la matrice tridiagonale donnée par :

$$A = \begin{pmatrix} a_1 & c_1 & 0 & .... & .... & 0 \\ b_2 & a_2 & c_2 & 0 & .... & 0 \\ . & . & . & . & . & . \\ 0 & ... & 0 & b_{n-1} & a_{n-1} & c_{n-1} \\ 0 & ... & ... & 0 & b_n & a_n \end{pmatrix}$$

et les vecteurs v et ε sont donnés par :

$$v = \begin{pmatrix} v_1 \\ v_2 \\ . \\ . \\ v_{n-1} \\ v_n \end{pmatrix} = \begin{pmatrix} f(x_1) - b_1 \cdot y_a \\ f(x_2) \\ . \\ . \\ f(x_{n-1}) \\ f(x_n) - c_n \cdot y_b \end{pmatrix} \text{ et } \varepsilon = \begin{pmatrix} \varepsilon_1 \\ \varepsilon_2 \\ . \\ . \\ \varepsilon_{n-1} \\ \varepsilon_n \end{pmatrix}$$

avec, pour tout k = 1, 2, ..., n :

$$a_k = \beta(x_k) - \frac{2}{h^2}$$

$$b_k = \frac{1}{h^2} - \frac{\alpha(x_k)}{2 \cdot h}$$

$$c_k = \frac{1}{h^2} + \frac{\alpha(x_k)}{2 \cdot h}$$

$$\varepsilon_k = \frac{f^{(4)}(\theta_k) + 2 \cdot \alpha(x_k) \cdot f'''(\eta_k)}{12} \cdot h^2$$

où θ_k et η_k sont compris entre x_k et x_{k+1}.

On a donc : $\|\varepsilon\| < C \cdot h^2$ où C est une constante qui dépend de α et des dérivées troisième et quatrième de f.

En négligeant le terme ε, le système discrétisé associé au problème de Dirichlet est alors le système tridiagonal :

$$A \cdot u = v \qquad (4)$$

Remarque — Le système (4) sera à diagonale strictement dominante si :

$$|a_k| > |b_k| + |c_k| \qquad (k = 1 , ... , n)$$

soit si :

$$\left|h^2 \cdot \beta(x_k) - 2\right| > \left|1 + \frac{\alpha(x_k)\cdot h}{2}\right| + \left|1 - \frac{\alpha(x_k)\cdot h}{2}\right|$$

En prenant h > 0 tel que $h < \frac{2}{M}$, où $M = \text{Sup}\{\ |\alpha(x)|\ ;\ x \in [a,b]\ \}$, la condition ci-dessus s'écrit : $\left|h^2 \cdot \beta(x_k) - 1\right| > 2$, ce qui est réalisé si β ne prend que des valeurs négatives sur [a,b].

En résumé la condition suffisante du théorème (1.2) nous assure, que pour h assez petit, le système discrétisé associé au problème de Dirichlet est un système linéaire tridiagonal à diagonale strictement dominante, il admet donc une unique solution.

Cette dernière peut se trouver par la méthode du double balayage de Cholesky.

La programmation structurée ne pose pas de problème, il suffit de décrire une procédure qui donne les coefficients de la matrice tridiagonale A, puis de faire appel à une procédure de résolution d'un système tridiagonal.

3.2 Discrétisation de l'équation de Poisson

Dans le cas de l'équation de Poisson, on a $\alpha = 0$ et la matrice du système est symétrique définie positive (car à diagonale strictement dominante et à termes diagonaux strictement positifs).

En multipliant par $-h^2$, on est en fait ramené à résoudre le système tridiagonal particulier $A \cdot u = v$, de matrice :

$$A = \begin{pmatrix} a_1 & -1 & 0 & .... & .... & 0 \\ -1 & a_2 & -1 & 0 & .... & 0 \\ . & . & . & . & . & . \\ 0 & ... & 0 & -1 & a_{n-1} & -1 \\ 0 & ... & ... & 0 & -1 & a_n \end{pmatrix}$$

avec :

$$a_k = 2 - h^2 \cdot \beta(x_k) > 2 \qquad (k = 1, 2, ..., n)$$

et de second membre :

$$v = \begin{pmatrix} v_1 \\ v_2 \\ . \\ . \\ v_{n-1} \\ v_n \end{pmatrix} = \begin{pmatrix} -f(x_1)\cdot h^2 + y_a \\ -f(x_2)\cdot h^2 \\ . \\ . \\ -f(x_{n-1})\cdot h^2 \\ -f(x_n)\cdot h^2 + y_b \end{pmatrix}$$

On peut résoudre ce système en adaptant la méthode des pivots de Gauss, ce qui donne le système triangulaire :

$$\begin{cases} u_j - d_j \cdot u_{j+1} = e_j \quad (j = 1, ..., n-1) \\ u_n = e_n \end{cases}$$

de solution :

$$\begin{cases} u_n = e_n \\ u_k = e_k + d_k \cdot u_{k+1} \quad (k = n - 1, \ldots, 1) \end{cases}$$

où on a posé :

$$\begin{cases} d_1 = \dfrac{1}{a_1};\ e_1 = v_1 \cdot d_1 \\ d_k = \dfrac{1}{a_k - d_{k-1}} \qquad (k = 2, \ldots, n) \\ e_k = (v_k + e_{k-1}) \cdot d_k \end{cases}$$

Remarque — Comme $a_k > 2$ pour tout k = 1 , ... , n, on en déduit que d_k est dans]0,1[pour tout k = 1, ... , n.

En notant $\|v\| = \mathrm{Sup}\{|v_k|;\ k = 1, \ldots, n\}$, on en déduit alors que $e_k < k \cdot \|v\|$, pour tout k = 1, ..., n. Puis :

$$u_k < \|v\| \cdot \frac{(n+1)^2}{2} \qquad (k = 1 , \ldots , n).$$

3.3 Majoration de l'erreur

On reprend les notations du paragraphe (1.4) et le problème est de majorer l'erreur commise g(x) = y(x) – φ(x), en remplaçant la solution exacte y(x) de l'équation de Dirichlet par la fonction affine par morceaux φ(x) qui passe par les points de coordonnées (x_k,u_k) pour tout k = 0, ..., n + 1.

Grâce à la transformation de Liouville (Cf. paragraphe (1.3)), on peut se limiter au cas de l'équation de Poisson, et un changement de variable affine nous ramène facilement à l'intervalle [0,1]. On fait donc ces hypothèses dans ce paragraphe.

Notons ψ la fonction affine par morceaux qui passe par les points de coordonnées $(x_k,y(x_k))$ pour k = 0 , ... , n + 1.

On écrit alors, g(x) = (y(x) – ψ(x)) + (ψ(x) – φ(x)).

3.3.1 Majoration de h(x) = y(x) – ψ(x)

Pour k = 0 , ... , n + 1, on a $h(x_k) = 0$ et h est deux fois continûment dérivable sur $]x_k,x_{k+1}[$ avec h" = y" car ψ est affine par morceaux (voir figure 8.1).

On peut alors montrer le :

Lemme — On notant $M_2 = \mathrm{Sup}\{\ |y''(x)|\ ;\ x \in [a,b]\ \}$, on a :

$$|h(x)| < \frac{(x - x_k) \cdot (x_{k+1} - x) \cdot M_2}{2} \qquad \text{sur } [x_k,x_{k+1}].$$

Et avec $(x - x_k) \cdot (x_{k+1} - x) \le \dfrac{(x_{k+1} - x)^2}{4} \le \dfrac{h^2}{4}$ sur $[x_k,x_{k+1}]$, on en déduit que :

$$|h(x)| \le \frac{M^2 \cdot h^2}{8} \qquad \text{sur } [0,1].$$

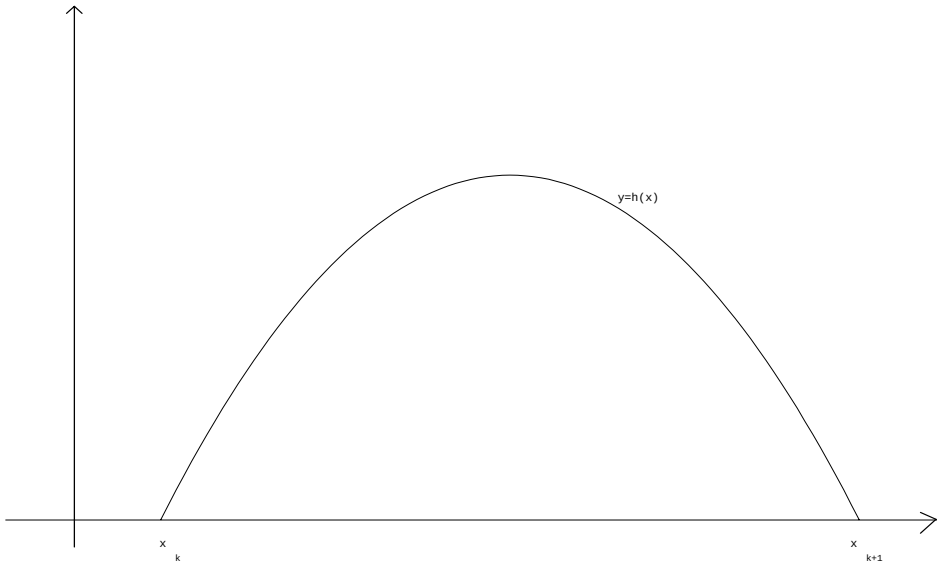


Figure 8.1

3.3.2 Majoration de $\theta(x) = \psi(x) - \varphi(x)$

La fonction θ est affine par morceaux, on a donc sur [0,1] :

$$\left|\theta(x)\right| \le \mathrm{Max}\left\{\left|\theta(x_k)\right|;\ k = 1, \ldots, n\right\}$$

avec $\theta(x_k) = y(x_k) - u_k$ pour k = 1 , ... , n ($\theta(0) = \theta(1) = 0$).

Mais, avec :

$$\begin{cases} A \cdot y = v + \varepsilon \\ A \cdot u = v \end{cases}$$

on déduit que :

$$A \cdot (y - u) = \varepsilon$$

soit, en multipliant par $-h^2$:

$$-h^2 \cdot A \cdot (y - u) = -h^2 \cdot \varepsilon.$$

Avec la remarque du paragraphe (3.2), on déduit alors que :

$$\|y - u\| \le h^2 \cdot \|\varepsilon\| \cdot \frac{(n+1)^2}{2}$$

puis avec $\|\varepsilon\| \le C \cdot h^2$, $h = \dfrac{1}{n+1}$ et $\left|\theta(x)\right| \le \|y - u\|$, on déduit que :

$$\left|\theta(x)\right| \le M_4 \cdot \frac{h^2}{24} \text{ sur } [0,1], \text{ où } M_4 = \mathrm{Sup}\left\{\left|y^{(4)}(x)\right|;\ x \in [0,1]\right\}$$

3.3.3 Majoration de $g(x) = y(x) - \varphi(x)$

De ce qui précède, on déduit donc que :

$$\left|y(x) - \varphi(x)\right| \le K \cdot h^2 \text{ sur } [0,1], \text{ où } K = \frac{M_2 + M_4}{8}$$

Il en résulte que la méthode converge toujours et est d'ordre 2 en h.

Remarque — On a en fait montré que la suite de fonctions affines par morceaux φ_n converge uniformément vers la solution exacte y sur [0,1].

4. Exemples d'application

4.1 Dissipation de la chaleur dans un disque

Position du problème — On se donne un disque plat de rayon R et d'épaisseur e négligeable devant R. On suppose qu'il est constitué d'un matériau de conductivité thermique λ, de masse volumique μ et de capacité calorifique massique C.

La température en un point M(r), situé à la distance r de l'axe du disque est supposée ne dépendre que de r, on la note T(r).

Un corps cylindrique de rayon $R_1 < R$, maintenu à la température T_1 est appliqué au centre du disque sur les deux faces. Donc les points à une distance $r \in [0,R_1]$ sont à la température constante T_1.

Pour $r \in]R_1,R]$, les deux faces du disque sont au contact avec un milieu ambiant à une température T_0.

La puissance thermique dissipée par unité de surface est $A\cdot(T(r) - T_0)$pour $r \in [R_1,R]$.

Enfin, le bord du disque est maintenu à la température constante T_0.

Les données numériques sont les suivantes :

$\lambda = 100\cdot W\cdot m^{-1}\cdot K^{-1}$; $A = 100\cdot W\cdot K^{-1}\cdot m^{-2}$; $R = 8\cdot cm$; $R_1 = 2\cdot cm$; $e = 2\cdot mm$ $T_1 = 500.K$; $T_0 = 300.K$.

Analyse du problème — En régime permanent, le bilan thermique d'une couronne de rayons r et r + dr nous permet d'écrire :

$$2\cdot r\cdot e\cdot\lambda\cdot T'(r) - 2\cdot(r + dr)\cdot e\cdot\lambda\cdot T'(r + dr) + 2\cdot(2\cdot r\cdot dr)\cdot A\cdot(T(r) - T_0) = 0.$$

Soit en écrivant que $T'(r + dr) \cong T'(r) + dr\cdot T''(r)$, et en négligeant $(dr)^2$:

$$-2\cdot dr\cdot e\cdot\lambda\cdot T'(r) - 2\cdot r\cdot dr\cdot e\cdot\lambda\cdot T''(r) + 2\cdot(2\cdot r\cdot dr)\cdot A\cdot(T(r) - T_0) = 0.$$

Soit, en divisant par $-2r\cdot dr\cdot e\cdot\lambda$:

$$T''(r) + \frac{1}{r}\cdot T'(r) - \frac{2\cdot A}{e\cdot\lambda}\cdot(T(r) - T_0) = 0.$$

En conclusion, l'équation différentielle vérifiée par $y(x) = T(x) - T_0$, où x = r et T_0 = 300 degrés Kelvin est :

$$\begin{cases} y''(x) + \alpha(x)\cdot y'(x) + \beta(x)\cdot y(x) = f(x) \quad (0.02 < x < 0.08) \\ y(0.02) = 0;\ y(0.08) = 200 \end{cases}$$

avec :

$$\alpha(x) = \frac{1}{x}\ ;\ \beta(x) = -\frac{2\cdot A}{\lambda\cdot e} = -1000\ ;\ f(x) = 0.$$

4.2 Fléchissement d'une poutre

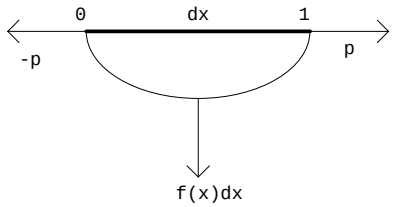


Figure 8.2

Considérons une poutre de longueur 1, appuyée à ses extrémités 0 et 1, soumise à une charge transversale f(x)dx par unité de longueur, et étirée selon son axe par une force P.

Alors, le moment fléchissant u(x), au point d'abscisse $x \in [0,1]$ est solution de l'équation de Poisson :

$$\begin{cases} -u''(x) + c(x) \cdot u(x) = f(x) \quad (0 < x < 1) \\ u(0) = u(1) = 0 \end{cases}$$

où $c(x) = \dfrac{P}{E \cdot I(x)}$, avec E désignant le module de Young du matériau constituant la poutre et I(x) désignant le moment principal d'inertie de la section de la poutre au point d'abscisse x.

5. Résolution approchée d'équations aux dérivées partielles par la méthode des différences finies

5.1 Intervention des équations aux dérivées partielles en physique

Les équations aux dérivées partielles interviennent dans la description de nombreux phénomènes physiques. Ci-dessous on décrit certains exemples parmi les plus connus.

5.1.1 L'équation des cordes vibrantes ou équation des ondes en dimension un

Position du problème — Considérons une corde de section constante, de longueur L, fixée en ses extrémités sur un axe Ox. La corde est tendue avec une tension T. On fait, de plus, les hypothèses suivantes :

- le poids par unité de longueur dx est w(x) ;
- la corde est soumise à une force verticale f(x,t) par unité de longueur dx ;
- la déformation de la corde influe de manière négligeable sur sa longueur, donc en tout point de la corde l'angle α de cette dernière avec l'axe des x est supposé petit de sorte que $tg(\alpha) \cong \alpha$;
- la position de la corde et la distribution des vitesses à t = 0 sont supposées connues, c'est-à-dire qu'on connaît y(x,0) = y (x) et $\dfrac{\partial y}{\partial t}(x,0) = y_1(x)$ (on aura $y_1(x) = 0$ si la corde est lâchée à partir d'une position donnée).

On se propose alors de décrire les petits mouvements verticaux de la corde en fonction de l'abscisse x et du temps t. Pour ce faire, On note y(x,t) le déplacement vertical de la corde pour $x \in [0,L]$ et $t > 0$.

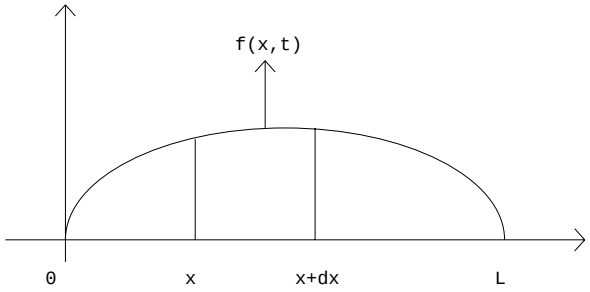


Figure 8.3

Analyse du problème — On travaille sur un élément de corde [x,x+dx] de masse $dm = \frac{w(x)\cdot dx}{g}$.

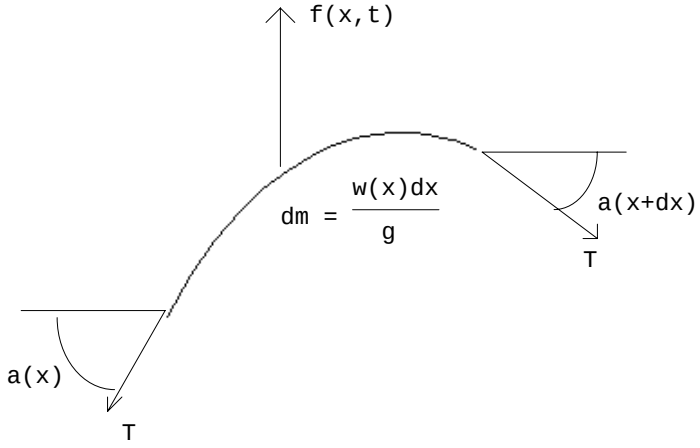


Figure 8.4

La tension en chacune de ses extrémités est dirigée suivant la tangente au segment et les composantes verticales de cette tension sont respectivement :

$$T\cdot\sin(\alpha(x+dx)) \cong T\cdot tg(\alpha(x+dx)) \; ; \; T\cdot\sin(\alpha(x)) \cong T.tg(\alpha(x)).$$

L'accélération subie par dm étant $\frac{\partial^2 y}{\partial t^2}$, avec la loi de Newton on déduit que :

$$T\cdot tg(\alpha(x+dx)) - T\cdot tg(\alpha(x)) + f(x,t)\cdot dx = \frac{\partial^2 y}{\partial t^2}\cdot\frac{w(x)\cdot dx}{g}$$

En écrivant que $tg(\alpha(x) = \frac{\partial y}{\partial x}(x,t)$, et en faisant tendre dx vers 0 on déduit alors que le déplacement vertical y(x,t) est solution de l'équation aux dérivées partielles :

$$\frac{\partial^2}{\partial t^2}(x,t) = \frac{T\cdot g}{w(x)}\cdot\frac{\partial^2 y}{\partial x^2}(x,t) + \frac{g}{w(x)}\cdot f(x,t)$$

avec les conditions :

$$\begin{cases} y(0,t) = y(L,t) = 0 \text{ pour } t \geq 0 \text{ (conditions aux limites)} \\ y(x,0) = y_0(x) \text{ et } \dfrac{\partial y}{\partial t}(x,0) = y_1(x) \text{ sur } [0,L] \text{ (conditions initiales)} \end{cases}$$

L'équation obtenue est *« l'équation des ondes à une dimension »* encore appelée *« équation des cordes vibrantes »*.

Un tel problème est appelé *problème aux limites* (à cause des conditions aux limites de la corde) *d'évolution* (pour préciser que le temps intervient) *avec conditions initiales* (car on connaît le comportement initial de la corde).

Remarque 1 — Si on note $c(x)^2 = \dfrac{T \cdot g}{W(x)}$, alors les dimensions de c(x) sont celles d'une vitesse.

Dans le cas particulier d'une corde homogène, la densité linéaire de masse w(x) est une constante σ et $c = \sqrt{\dfrac{T \cdot g}{\sigma}}$ est la vitesse de propagation de l'onde (voir Barret et Wylie p. 465).

Remarque 2 — Dans certains cas, les forces extérieures f(x,y) sont négligeables devant la tension T. De plus si la corde est homogène alors l'équation des ondes va s'écrire :

$$\frac{\partial^2 y}{\partial t^2} - c^2 \cdot \frac{\partial^2 y}{\partial x^2} = 0$$

Cette dernière est connue comme *l'équation de d'Alembert* et peut être résolue de façon explicite.

Il est facile de voir que si f et g sont des fonctions d'une variable réelle deux fois dérivables, alors $y(x,t) = f(x - c \cdot t) + g(x + c \cdot t)$ est solution de l'équation des ondes. Inversement, en posant $u = x - c \cdot t$ et $v = x + c \cdot t$, on transforme l'équation en

$\dfrac{\partial^2 y}{\partial u \partial v} = 0$ et les solutions de cette dernière s'écrivent f(u) + g(v) où f et g sont deux fonctions d'une variable deux fois dérivables. Les conditions initiales permettent alors de déterminer f et g de façon explicite. En effet, on a :

$$\begin{cases} y(x,0) = f(x) + g(x) = y_0(x) \\ \dfrac{\partial y}{\partial t}(x,0) = -c \cdot f'(x) + c \cdot g'(x) = y_1(x) \end{cases}$$

donc :

$$\begin{cases} f(x) = \dfrac{1}{2} \cdot \left(y_0(x) - \displaystyle\int_0^x y_1(t) \frac{dt}{c} \right) + k \\ g(x) = \dfrac{1}{2} \cdot \left(y_0(x) + \displaystyle\int_0^x y_1(t) \frac{dt}{c} \right) - k \end{cases}$$

et :

$$y(x,t) = \frac{1}{2} \cdot \left(y_0(x - c \cdot t) + y_0(x + c \cdot t) \right) + \int_{x-c \cdot t}^{x+c \cdot t} y_1(t) \frac{dt}{2 \cdot c}$$

Remarque 3 — Si on sait résoudre de façon explicite l'équation des ondes pour une corde homogène, dans le cas général il y a peu d'espoir d'obtenir une solution explicite. On cherchera alors une solution approchée à l'aide de la méthode des différences finies.

5.1.2 L'équation des ondes en dimension deux

De la même façon, on peut étudier les petits déplacements verticaux d'une membrane Ω tendue le long d'une courbe fermée dans le plan des x,y.

Les hypothèses sont les suivantes :

- le poids par unité de surface dxdy est w(x,y) ;
- la membrane est tendue avec une même tension T en tout point et toute direction ;
- la membrane est soumise à une force verticale f(x,y,t) par unité de surface dxdy;
- la déformation de la membrane influe de manière négligeable sur ses dimensions;
- la position de la membrane et la distribution des vitesses à t = 0 sont supposées connues, c'est-à-dire qu'on connaît $z(x,y,0) = z_0(x,y)$ et $\frac{\partial z}{\partial t}(x,y,0) = z_1(x,y)$.

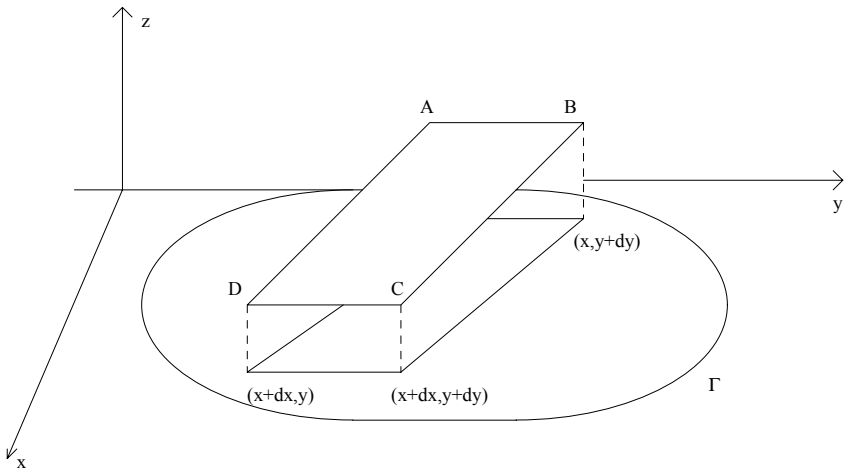


Figure 8.5

En travaillant sur un élément de surface, on déduit de la loi de Newton que le déplacement z(x,y,t) est solution de l'équation :

$$\frac{\partial^2 z}{\partial t^2} = \frac{T \cdot g}{w(x,y)} \cdot \left(\frac{\partial^2 z}{\partial x^2} + \frac{\partial^2 z}{\partial y^2} \right) + \frac{g}{w(x,y)} \cdot f(x,y,t)$$

avec les conditions initiales :

$$\begin{cases} z(x,y,0) = z_0(x,y) \text{ sur } \Omega \\ \dfrac{\partial z}{\partial t}(x,y,0) = z_1(x,y) \text{ sur } \Omega \\ z(x,y,t) = 0 \text{ sur } \Gamma \text{ pour tout } t \geq 0 \end{cases}$$

L'équation obtenue est *l'équation des ondes en dimension deux*.

On note $\Delta = \dfrac{\partial^2}{\partial x^2} + \dfrac{\partial^2}{\partial y^2}$ (opérateur de Laplace ou *laplacien*) .

Remarque 1 — Si on s'intéresse aux solutions stationnaires, c'est-à-dire qui ne dépendent pas du temps, on obtient *l'équation de Poisson* :

$$\Delta z = h(x,y) \text{ sur } \Omega$$

qui pour h = 0 est connue comme *l'équation de Laplace*.

Remarque 2 — Pour w(x,y) = w constant et sans forces extérieures, l'équation devient :

$$\frac{\partial^2}{\partial t^2} = c^2 \cdot \Delta z$$

avec $c = \sqrt{\dfrac{T \cdot g}{w}}$ qui a les dimensions d'une vitesse (c'est la vitesse de propagation de l'onde).

5.1.3 Equation de la chaleur en dimension un

On considère ici le problème de l'écoulement de la chaleur dans un milieu conducteur. On se donne donc un solide dans lequel la chaleur se transmet par conduction.

Comme toujours, on travaille sur un élément de volume dxdydz. Si le poids volumique est ρ, alors la masse d'un tel élément est $dm = \rho \cdot \dfrac{dxdydz}{g}$. De plus, si u(x,y,z,t) mesure la température en un point M(x,y,z) à l'instant t et si du est la variation de température durant dt alors la quantité de chaleur stockée dans cet élément de volume durant dt est : dH = c·dm·du (la quantité de chaleur gagnée ou perdue par un corps quand sa température change est proportionnelle à la masse du corps et au changement de température. La constante de proportionnalité est la *chaleur spécifique* du corps notée c).

La vitesse de stockage de la chaleur est alors :

$$\frac{dH}{dt} = \frac{c \cdot \rho}{g} \cdot dxdydz \cdot \frac{du}{dt}$$

Les changements de température sont dus à deux causes :

(i) la chaleur peut être générée à l'intérieur du corps par effet électrique, chimique, ..., à une vitesse f(x,y,z,t) par unité de volume ;

(ii) l'élément de volume reçoit de la chaleur par chacune de ses faces.

En particulier, la vitesse à laquelle la chaleur s'écoule à travers la face EFGH est approximativement (en prenant comme valeur moyenne du gradient de température sa valeur au centre de EFGH) :

$$-k \cdot dydz \cdot \frac{\partial u}{\partial x}\left(x, y + \frac{dy}{2}, z + \frac{dz}{2} \right)$$

(la vitesse à laquelle la chaleur s'écoule à travers une surface est proportionnelle à son aire et au gradient de température, en degrés par unité de longueur, dans une direction perpendiculaire à la surface. La constante de proportionnalité, notée k, est la *conductivité thermique*)

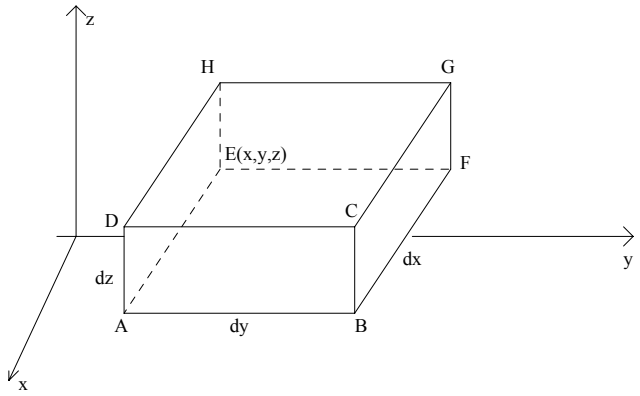


Figure 8.6

De même pour ABCD, on a approximativement :

$$k \cdot dydz \cdot \frac{\partial u}{\partial x}\left(x + dx, y + \frac{dy}{2}, z + \frac{dz}{2} \right)$$

La somme de ces deux quantités donne la vitesse de conduction de la chaleur suivant l'axe des x. On procède de même pour les deux autres axes.

Ensuite, on écrit que la vitesse à laquelle la chaleur est stockée par l'élément de volume est égale à la vitesse à laquelle elle est produite par cet élément (c'est le bilan thermique) plus la vitesse d'écoulement de la chaleur à travers ce dernier. Ce qui donne, au voisinage de l'élément de volume :

$$\frac{c\cdot\rho}{g}\cdot dxdydz\cdot\frac{du}{dt}=f(x,y,z,t)\cdot dxdydz$$
$$+\ k\cdot dydz\cdot\left(\frac{\partial u}{\partial x}\left(x+dx,y+\frac{dy}{2},z+\frac{dz}{2}\right)-\frac{\partial u}{\partial x}\left(x,y+\frac{dy}{2},z+\frac{dz}{2}\right)\right)$$
$$+\ k\cdot dxdz\cdot\left(\frac{\partial u}{\partial y}\left(x+\frac{dx}{2},y+dy,z+\frac{dz}{2}\right)-\frac{\partial u}{\partial y}\left(x+\frac{dx}{2},y,z+\frac{dz}{2}\right)\right)$$
$$+\ k\cdot dxdy\cdot\left(\frac{\partial u}{\partial z}\left(x+\frac{dx}{2},y+\frac{dy}{2},z+dz\right)-\frac{\partial u}{\partial z}\left(x+\frac{dx}{2},y+\frac{dy}{2},z\right)\right)$$

En divisant par k·dxdydz, puis en faisant tendre dx, dy et dz vers 0, on obtient *l'équation de conduction de la chaleur* (ou *équation de Fourier*) :

$$a^2\cdot\frac{\partial u}{\partial t}=\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2}+\frac{\partial^2 u}{\partial z^2}+\frac{1}{k}\cdot f(x,y,z,t)$$

avec $a^2=\dfrac{C\cdot\rho}{k\cdot g}$

On a, de plus les conditions :

$$\begin{cases}u(x,y,z,0)=u_0(x,y,z)\ \text{(conditions initiales)}\\ u(x,y,z,t)=g(x,y,z,t)\ \text{sur}\ \partial\Omega\ \text{(conditions aux limites)}\end{cases}$$

Là encore, c'est un problème d'évolution avec conditions aux limites.

Remarque — Dans des cas particuliers importants, la chaleur n'est ni générée ni perdue dans le volume Ω. On dit alors qu'on s'intéresse à la distribution de température en régime stationnaire. Dans ces conditions, on a f = 0 et $\dfrac{\partial u}{\partial t}=0$, de sorte que l'équation de la chaleur se réduit à l'équation de Laplace $\Delta u = 0$.

5.2 Classification des équations aux dérivées partielles d'ordre 2

5.2.1 Introduction

On peut se poser la question de savoir quelles sont les équations aux dérivées partielles qui se résolvent aussi simplement que celle des ondes à une dimension par une méthode analogue à celle de d'Alembert.

On s'intéresse donc ici aux *équations du second ordre quasi-linéaires* du type :

$$a\cdot\frac{\partial^2 u}{\partial x^2}+2\cdot b\cdot\frac{\partial^2 u}{\partial xy}+c\cdot\frac{\partial^2 u}{\partial y^2}=d\qquad(1)$$

où a, b, c sont des fonctions continues de x et y et d est une fonction continue de x, y, u, $\dfrac{\partial u}{\partial x}$ et $\dfrac{\partial u}{\partial y}$.

Remarque — Une telle équation peut se ramener à un système de deux équations du premier ordre en posant $p = \frac{\partial u}{\partial x}$ et $q = \frac{\partial u}{\partial y}$, ce qui donne :

$$\begin{cases} a \cdot \frac{\partial p}{\partial x} + b \cdot \frac{\partial p}{\partial y} + b \cdot \frac{\partial q}{\partial x} + c \cdot \frac{\partial q}{\partial y} = d \\ \frac{\partial q}{\partial x} - \frac{\partial q}{\partial y} = 0 \end{cases}$$

5.2.2 Cas particulier des équations à coefficients constants et sans second membre

On suppose ici que a, b, c sont des constantes et d = 0.

En utilisant la méthode de d'Alembert, on cherche des solutions de la forme $u(x,y) = f(x + \lambda \cdot y)$ et on voit que nécessairement l est solutions de *l'équation caractéristique* :

$$c \cdot \lambda^2 + 2 \cdot b \cdot \lambda + a = 0$$

Par analogie avec les coniques, on dira alors que l'équation est :

- *hyperbolique* si $b^2 - a \cdot c > 0$;
- *parabolique* si $b^2 - a \cdot c = 0$;
- *elliptique* si $b^2 - a \cdot c < 0$.

Si λ_1 et λ_2 sont les racines de l'équation caractéristique, les droites $x + \lambda_j \cdot y = c_j$, pour j = 1 et j = 2, sont appelées les *caractéristiques* de l'équation.

5.2.3 Cas général

De manière générale on dira que l'équation (1) est :

- *hyperbolique* si $b^2(x,y) - a(x,y) \cdot c(x,y) > 0$ sur Ω ;
- *parabolique* si $b^2(x,y) - a(x,y) \cdot c(x,y) = 0$ sur Ω ;
- *elliptique* si $b^2(x,y) - a(x,y) \cdot c(x,y) < 0$ sur Ω ;

où Ω est un ouvert sur lequel est définie l'équation.

Les courbes intégrales de l'équation différentielle :
$a(x,y) \cdot (y')^2 - 2 \cdot b \cdot (x,y) \cdot y' + c(x,y) = 0$ sont alors appelées les *caractéristiques* de (1). On écrira ces courbes intégrales sous la forme $\varphi(x,y) = c_1$ et $\psi(x,y) = c_2$.

Les équations elliptiques, dont les exemples typiques sont l'équation de Laplace $\Delta u = 0$ ou l'équation de Poisson $\Delta u = g$, décrivent en général des *phénomènes stationnaires* (c.a.d. indépendants du temps) comme le champ électrique dans un conducteur, l'écoulement d'un fluide non compressible, le transfert de chaleur par conduction, ...

De manière générale, le changement de variables :

$$r = \frac{\varphi(x,y) + \psi(x,y)}{2}, \quad s = \frac{\varphi(x,y) - \psi(x,y)}{2 \cdot i}$$

va transformer une équation elliptique sous la forme standard :

$$\Delta u(r,s) = g\left(u, \frac{\partial u}{\partial r}, \frac{\partial u}{\partial s}, r, s\right)$$

Les équations hyperboliques ont généralement pour origines des problèmes de vibrations ou de propagation d'ondes . L'exemple typique étant l'équation des

ondes à une dimension $\frac{\partial^2 u}{\partial t^2} - v^2 \cdot \frac{\partial^2 u}{\partial x^2} = 0$ (ici t joue le rôle de la variable y). Ce type d'équation apparaît souvent dans le cadre de *problèmes d'évolution* (i.e. le temps intervient) avec valeurs initiales.

Le changement de variable :

$$r = \varphi(x,y),\ s = \psi(x,y)$$

va alors transformer une équation hyperbolique sous la forme standard :

$$\frac{\partial^2 u}{\partial r \partial s} = g\left(u, \frac{\partial u}{\partial r}, \frac{\partial u}{\partial s}, r, s\right)$$

Les équations paraboliques interviennent aussi dans les *problèmes d'évolution*. L'exemple typique étant l'équation de diffusion $\frac{\partial u}{\partial t} + d \cdot \frac{\partial^2 u}{\partial x^2} = 0$.

Le changement de variable :

$$r = x,\ s = \varphi(x,y)$$

va transformer une équation parabolique en :

$$\frac{\partial^2 u}{\partial r^2} = g\left(u, \frac{\partial u}{\partial r}, \frac{\partial u}{\partial s}, r, s\right)$$

5.2.4 Exemple

L'équation du potentiel des vitesses des écoulements irrotationnels d'un fluide compressible peut s'écrire, dans le cas stationnaire :

$$\left(u^2 - a^2\right) \cdot \frac{\partial^2 \varphi}{\partial x^2} + 2 \cdot u \cdot v \cdot \frac{\partial^2 \varphi}{\partial x \partial y} + \left(v^2 - a^2\right) \cdot \frac{\partial^2 \varphi}{\partial y^2} = 0$$

où x et y sont des variables d'espace, φ le potentiel des vitesses, u et v les composantes de la vitesse ($u = \frac{\partial \varphi}{\partial x}$ et $v = \frac{\partial \varphi}{\partial y}$) et a est la vitesse du son qui est reliée à u et v par :

$$\frac{1}{\gamma - 1} \cdot a^2 + \frac{1}{2} \cdot \left(u^2 + v^2\right) = \text{Const.}$$

en supposant le gaz parfait à chaleur spécifique constante de rapport γ.

Le type de cette équation dépend de $\delta = a^4 \cdot (M^2 - 1)$ où $M = \frac{\sqrt{u^2 + v^2}}{a}$ est le *nombre de Mach*.

En un point où $M < 1$ (*écoulement subsonique*), $\delta < 0$ et l'équation est elliptique, pour $M = 1$ (*écoulement sonique*), l'équation est parabolique et pour $M > 1$ (*écoulement supersonique*), l'équation est hyperbolique.

On a donc ainsi un exemple où le type de l'équation peut changer d'une région à l'autre de l'espace des (x,y).

5.2.5 Remarque

Du point de vue numérique, il sera plutôt important de savoir si on a affaire à un problème d'évolution avec valeurs initiales (le temps intervient) ou si on affaire à un problème avec conditions aux bords (de type « statique »).

5.3 Principe de la méthode des différences finies

Comme dans le cas des équations différentielles, l'idée est de trouver des valeurs approchées de la solution sur un ensemble fini de points d'un maillage donné du domaine Ω sur lequel on étudie l'équation aux dérivées partielles.

En dimension deux, on pourra prendre un maillage rectangulaire qui recouvre le domaine considéré .

Par exemple, pour l'équation de Laplace sur un domaine borné $\Omega \subset R$ (problème avec conditions aux bords) on prendra un maillage du type suivant :

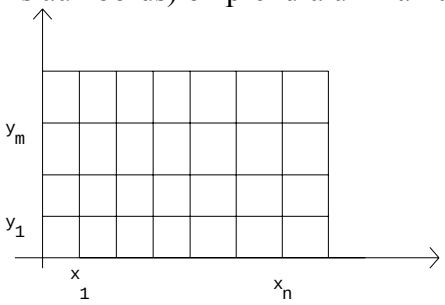


Figure 8.7

Pour l'équation des ondes à une dimension ou l'équation de la chaleur (problème d'évolution) on prendra un maillage infini du type suivant

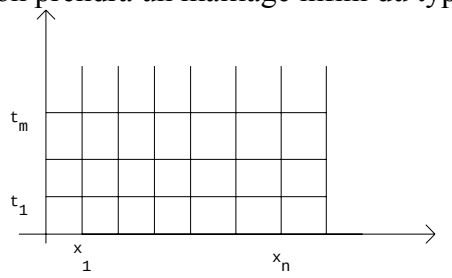


Figure 8.8

L'idée sera de remplacer chaque dérivée partielle par des différences finies et on aboutira en général à un système linéaire qui pourra être résolu soit directement soit par des méthodes itératives.

5.4 Cas d'un problème avec conditions au bord : l'équation de Poisson (équation de la chaleur en régime stationnaire)

On cherche donc une fonction de classe C^2 sur un domaine Ω de R^2 telle que :

$$\begin{cases} \Delta u(x,y) = f(x,y) \text{ à l'intérieur de } \Omega \\ u(x,y) = g(x,y) \text{ sur le bord de } \Omega \end{cases}$$

où les fonctions f et g sont données.

Remarque — Avec de bonnes hypothèses de régularité sur f, g et $\Gamma = \partial\Omega$, on peut montrer que ce problème admet une unique solution.

Notations — On fait un maillage de R^2 de même pas $h > 0$ dans les directions x et y dont les noeuds sont notés :

$$x_i = i \cdot h \text{ et } y_j = j \cdot h \text{ pour } i, j \text{ dans } Z$$

On note Ω_h l'ensemble des noeuds qui sont dans Ω et Γ_h l'ensemble des points d'intersection du bord $\Gamma = \partial\Omega$ avec les lignes du maillage (a priori les points de Γ_h ne sont pas des noeuds du maillage).

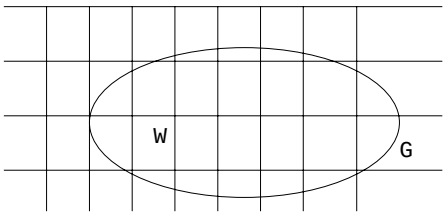


Figure 8.9

On posera aussi :

$$\begin{cases} f_{i,j} = f(x_i, y_j) \\ g_{i,j} = g(x_i, y_j) \end{cases}$$

et $u_{i,j}$ est la valeur approchée cherchée de $u(x_i,y_j)$ pour un point intérieur à Ω.

Le calcul approché du laplacien d'une fonction φ de classe C^2 dans un voisinage de (x_0,y_0) se fera à l'aide des différences finies, c'est-à-dire en posant :

$$\begin{cases} \dfrac{\partial^2 \varphi}{\partial x^2}(x_0, y_0) \cong \dfrac{\varphi(x_0 + h, y_0) - 2 \cdot \varphi(x_0, y_0) + \varphi(x_0 - h, y_0)}{h^2} \\ \dfrac{\partial^2 \varphi}{\partial y^2}(x_0, y_0) \cong \dfrac{\varphi(x_0, y_0 + h) - 2 \cdot \varphi(x_0, y_0) + \varphi(x_0, y_0 - h)}{h^2} \end{cases}$$

Le système discrétisé associé au problème de Poisson s'écrit alors :

$$\frac{u_{i+1,j} - 2 \cdot u_{i,j} + u_{i-1,j}}{h^2} + \frac{u_{i,j+1} - 2 \cdot u_{i,j} + u_{i,j-1}}{h^2} = f_{i,j}$$

pour tous les (i,j) tels que $(x_i,y_j) \in \Omega_h$.

En choisissant une façon de numéroter les points intérieurs, on aboutit ainsi à un système linéaire.

5.5 *Equations elliptiques avec conditions aux bords sur un rectangle*

5.5.1 Position du problème et notations

On considère une équation elliptique du type :

$$(1) \quad \begin{cases} \alpha \cdot \dfrac{\partial^2 u}{\partial x^2} + \beta \cdot \dfrac{\partial u}{\partial x} + \gamma \cdot \dfrac{\partial^2 u}{\partial y^2} + \delta \cdot \dfrac{\partial u}{\partial y} + \varepsilon \cdot u = f \text{ sur } \Omega =]0,a[\times]0,b[\\ u(x,y) = g(x,y) \text{ sur } \partial\Omega \end{cases}$$

où α, β, γ, δ, ε, f et g sont des fonctions données assez régulières sur Ω pour assurer l'existence et l'unicité d'une solution.

Remarque 1 — Par un changement de variable toute équation elliptique d'ordre deux se ramènera à la forme ci-dessus.

Remarque 2 — Les conditions aux bords peuvent aussi porter sur le gradient de u.

Les exemples les plus classiques de tels problèmes sont :

- *le problème de Dirichlet* — $\begin{cases} \Delta u = f \text{ sur } \Omega \\ u = g \text{ sur } \partial\Omega \end{cases}$
- *le problème de Neumann* — $\begin{cases} \Delta u = f \text{ sur } \Omega \\ \dfrac{\partial u}{\partial n} = g \text{ sur } \partial\Omega \end{cases}$
- *le problème mixte de Fourier* — $\begin{cases} \Delta u = f \text{ sur } \Omega \\ \dfrac{\partial u}{\partial n} + k \cdot u = 0 \text{ sur } \partial\Omega \end{cases}$

On suppose que les dimensions du rectangle Ω vérifient :

$$\begin{cases} a = (n+1) \cdot h \\ b = (m+1) \cdot h \end{cases}$$

et on définit un maillage de Ω en posant :

$$\begin{cases} x_i = i \cdot h \ (i = 0, 1, \ldots, n+1) \\ y_j = j \cdot h \ (j = 0, 1, \ldots, m+1) \end{cases}$$

Pour toute fonction φ définie sur Ω, on posera :

$$\varphi_{i,j} = \varphi(x_i, y_j) \qquad (0 \le i \le n+1 \ , \ 0 \le j \le m+1)$$

La fonction u solution de (1) est donc supposée connue aux points du bord $(x_i,0)$, $(x_i,m+1)$, $(0,y_j)$ et $(n+1,y_j)$ pour $i = 0, \ldots, n+1$ et $j = 0, \ldots, m+1$.

Le problème est alors de trouver des valeurs approchées, que nous noterons $u_{i,j}$ des $u(x_i,y_j)$ pour tous les points intérieurs de Ω, c'est-à-dire pour $i = 1, 2, \ldots, n$ et $j = 1, 2, \ldots, m$.

Ces valeurs inconnues seront consignées dans le vecteur :

$$u = {}^t(u_{11},u_{21},\ldots,u_{n1},u_{12},u_{22},\ldots,u_{n2},\ldots\ldots,u_{1m},u_{2m},\ldots,u_{nm})$$

ce qui revient à numéroter les points de l'intérieur de Ω de gauche à droite en démarrant par le bas.

Les dérivées partielles sont alors approximées par :

$$\frac{\partial u}{\partial x}\left(x_i, y_j\right) \cong \frac{u_{i+1,j} - u_{i-1,j}}{2 \cdot h}$$
$$\frac{\partial u}{\partial y}\left(x_i, y_j\right) \cong \frac{u_{i,j+1} - u_{i,j-1}}{2 \cdot h}$$
$$\frac{\partial^2 u}{\partial x^2}\left(x_i, y_j\right) \cong \frac{u_{i+1,j} - 2 \cdot u_{i,j} + u_{i-1,j}}{h^2}$$
$$\frac{\partial^2 u}{\partial y^2}\left(x_i, y_j\right) \cong \frac{u_{i,j+1} - 2 \cdot u_{i,j} + u_{i,j-1}}{h^2}$$

On dit qu'on a des approximations à 5 points des dérivées partielles.

5.5.2 Discrétisation du problème (1)

Le problème se discrétise donc de la façon suivante :

$$a_{i,j} \cdot u_{i+1,j} + b_{i,j} \cdot u_{i-1,j} + c_{i,j} \cdot u_{i,j+1} + d_{i,j} \cdot u_{i,j-1} + e_{i,j} \cdot u_{i,j} = f_{i,j}$$

pour i = 1, ..., n, j = 1, ..., m, avec :

$$a_{i,j} = \frac{\alpha_{i,j}}{h^2} + \frac{\beta_{i,j}}{2 \cdot h}$$
$$b_{i,j} = \frac{\alpha_{i,j}}{h^2} - \frac{\beta_{i,j}}{2 \cdot h}$$
$$c_{i,j} = \frac{\gamma_{i,j}}{h^2} + \frac{\delta_{i,j}}{2 \cdot h}$$
$$d_{i,j} = \frac{\gamma_{i,j}}{h^2} - \frac{\delta_{i,j}}{2 \cdot h}$$
$$e_{i,j} = -\frac{2 \cdot \alpha_{i,j}}{h^2} - \frac{2 \cdot \gamma_{i,j}}{h^2} + \varepsilon_{i,j}$$

Ce qui donne un système de n·m équations à n·m inconnues :

$$(2) \quad A \cdot u = v$$

où la matrice A est tridiagonale par blocs de la forme :

$$A = \begin{pmatrix} A_1 & C_1 & 0 & .... & .... & 0 \\ B_2 & A_2 & C_2 & 0 & .... & 0 \\ . & . & . & . & . & . \\ 0 & ... & 0 & B_{m-1} & A_{m-1} & C_{m-1} \\ 0 & ... & ... & 0 & B_m & A_m \end{pmatrix}$$

où :

$$A_j = \begin{pmatrix} e_{1,j} & a_{1,j} & 0 & .... & .... & 0 \\ b_{2,j} & e_{2,j} & a_{2,j} & 0 & .... & 0 \\ . & . & . & . & . & . \\ 0 & ... & 0 & b_{n-1,j} & e_{n-1,j} & a_{n-1,j} \\ 0 & ... & ... & 0 & b_{n,j} & e_{n,j} \end{pmatrix}$$

$$B_j = \begin{pmatrix} d_{1,j} & 0 & 0 & .... & .... & 0 \\ 0 & d_{2,j} & 0 & 0 & .... & 0 \\ . & . & . & . & . & . \\ 0 & ... & 0 & 0 & d_{n-1,j} & 0 \\ 0 & ... & ... & 0 & 0 & d_{n,j} \end{pmatrix}$$

$$c_j = \begin{pmatrix} c_{1,j} & 0 & 0 & .... & .... & 0 \\ 0 & c_{2,j} & 0 & 0 & .... & 0 \\ . & . & . & . & . & . \\ 0 & ... & 0 & 0 & c_{n-1,j} & 0 \\ 0 & ... & ... & 0 & 0 & c_{n,j} \end{pmatrix}$$

Le vecteur v contient les $f_{i,j}$ et les $u_{i,j}$ qui sont connus sur le bord.

De manière plus précise, la méthode des différences finies sur la ligne j va se traduire par :

$$\left(B_j, A_j, C_j\right) \cdot \begin{pmatrix} u_{1,j-1} \\ . \\ . \\ u_{n,j-1} \\ u_{1,j} \\ . \\ . \\ u_{n,j} \\ u_{1,j+1} \\ . \\ . \\ u_{n,j+1} \end{pmatrix} = \begin{pmatrix} f_{1,j} - b_{1,j} \cdot u_{0,j} \\ f_{2,j} \\ . \\ . \\ . \\ f_{n-1,j} \\ f_{n,j} - a_{n,j} \cdot u_{n+1,j} \end{pmatrix}$$

pour j = 2, ..., m – 1.

Et pour j = 1, j = m, on a :

$$(A_1, C_1)\cdot\begin{pmatrix} u_{1,1} \\ . \\ . \\ u_{n,1} \\ u_{1,2} \\ . \\ . \\ u_{n,2} \end{pmatrix} = \begin{pmatrix} f_{1,1} - b_{1,1}\cdot u_{0,1} - d_{1,1}\cdot u_{1,0} \\ f_{2,1} - d_{2,1}\cdot u_{2,0} \\ . \\ . \\ . \\ f_{n-1,1} - d_{n-1,1}\cdot u_{n-1,0} \\ f_{n,1} - a_{n,1}\cdot u_{n+1,1} - d_{n,1}\cdot u_{n,0} \end{pmatrix}$$

$$(B_m, A_m)\cdot\begin{pmatrix} u_{1,m-1} \\ . \\ . \\ u_{n,m-1} \\ u_{1,m} \\ . \\ . \\ u_{n,m} \end{pmatrix} = \begin{pmatrix} f_{1,m} - b_{1,m}\cdot u_{0,m} - d_{1,m}\cdot u_{1,m-1} \\ f_{2,m} - d_{2,m}\cdot u_{2,m+1} \\ . \\ . \\ . \\ f_{n-1,m} - d_{n-1,m}\cdot u_{n-1,m+1} \\ f_{n,m} - a_{n,m}\cdot u_{n+1,m} - d_{n,m}\cdot u_{n,m+1} \end{pmatrix}$$

Remarque — Le système obtenu est tridiagonal par blocs avec une matrice très creuse (sur chaque ligne, il y a au plus cinq termes non nuls). Si pour résoudre ce système, on emploie une méthode directe, par exemple la méthode L·R de Crout (ou de Cholesky quand la matrice est symétrique définie positive), le calcul des coefficients des matrices L et R va demander environ $(n\cdot m)^2$ opérations élémentaires, ce qui est prohibitif pour n et m grands. Pour cette raison on préférera utiliser une méthode itérative (Gauss-Seidel ou relaxation) qui va conserver la structure creuse de la matrice de départ et demander un nombre de calculs beaucoup moins important.

5.5.3 Méthode SOR (Simultaneous Over-Relaxation) pour résoudre un système tridiagonal par blocs

Dans les méthodes itératives de Gauss-Seidel et de relaxation, on avait utilisé les notations suivantes :

$$A = D + E + F$$

où D est la diagonale de A, E est le triangle inférieur strict et F est le triangle supérieur strict.

L'algorithme de Gauss-Seidel pour résoudre A·u = v est alors le suivant :

$$\begin{cases} u^{(0)} \text{ donné} \\ (D+E)\cdot u^{(k)} = -F\cdot u^{(k-1)} + v \quad (k \geq 1) \end{cases}$$

La méthode de relaxation consiste à "corriger" l'algorithme de Gauss-Seidel à l'aide d'un facteur ω de la façon suivante :

$$\begin{cases} u^{(0)} \text{ donné} \\ (D+\omega\cdot E)\cdot u^{(k)} = (1-\omega)\cdot D\cdot u^{(k-1)} + \omega\cdot\left(-F\cdot u^{(k-1)} + v\right) \quad (k \geq 1) \end{cases}$$

Ce qui donne :

$$u_i^{(k)} = \frac{1}{a_{ii}} \cdot \left\{ (1-\omega) \cdot a_{ii} \cdot u_i^{(k-1)} - \omega \cdot \sum_{j=1}^{i-1} a_{i,j} \cdot u_j^{(k)} - \omega \cdot \sum_{j=i+1}^{n} a_{i,j} \cdot u_j^{(k-1)} + \omega \cdot v_i \right\}$$

Ce qui peut aussi s'écrire, pour $k \geq 1$:

$$u_i^{(k)} = u_i^{(k-1)} - \omega \cdot \xi_i^{(k)} \quad (1 \leq i \leq n)$$

où $\xi_i^{(k)}$ est le terme résiduel défini par :

$$\xi_i^{(k)} = \sum_{j=1}^{i-1} a_{i,j} \cdot u_j^{(k)} + \sum_{j=i}^{n} a_{i,j} \cdot u_j^{(k-1)} - v_i$$

Pour le type de matrice qui nous intéresse, on a alors les résultats suivants :

Théorème **:** Si A est une matrice à diagonale strictement dominante telle que toutes les valeurs propres de la matrice de Jacobi correspondante soient réelles, alors la méthode de relaxation converge si, et seulement si $\omega \in]0,2[$ et le paramètre de relaxation optimal est donné par :

$$\omega_0 = \frac{2}{1 + \sqrt{1 - \rho(J)^2}}$$

Dans ces conditions, le rayon spectral de la matrice L_{ω_0} qui intervient dans la méthode de relaxation est donné par :

$$\rho(L_{\omega_0}) = \omega_0 - 1 = \left(\frac{\rho(J)}{1 + \sqrt{1 - \rho(J)^2}} \right)^2$$

Démonstration — Voir Ciarlet p. 106 .

Si on n'a pas d'idée sur ρ(J) (le rayon spectral de la matrice d'itérations dans la méthode de Jacobi) on pourra essayer plusieurs valeurs de ω dans]1,2[(on dit alors qu'on utilise une méthode de sur-relaxation).

Pour le problème qui nous concerne, on utilise plutôt la matrice A d'ordre $((n+2)\cdot(m+2))^2$ qui fait intervenir tous les coefficients $u_{i,j}$ pour i = 0, ..., n + 1 et j = 0, ..., m + 1. Ce qui donne, pour le terme résiduel à l'ordre k – 1 :

$$\xi_{i,j}^{(k-1)} = a_{i,j} \cdot u_{i+1,j}^{(k-1)} + b_{i,j} \cdot u_{i-1,j}^{(k)} + c_{i,j} \cdot u_{i,j+1}^{(k-1)} + d_{i,j} \cdot u_{i,j-1}^{(k)} + e_{i,j} \cdot u_{i,j}^{(k-1)} - f_{i,j}$$

Et alors :

$$u_{i,j}^{(k)} = u_{i,j}^{(k-1)} - \omega \cdot \frac{\xi_{i,j}^{(k-1)}}{e_{i,j}}$$

Comme test d'arrêt, on pourra prendre :

$$\left\| \xi^{(k-1)} \right\| < \varepsilon \cdot \|f\|$$

où f désigne le vecteur de coordonnées $f_{i,j}$ et $\varepsilon > 0$ est une précision donnée.

Remarque 1 — Si on sépare les coordonnées du vecteur $u^{(k)}$ en deux groupes « pair » et « impair » (la « parité » de $u_{i,j}^{(k)}$ étant celle de i + j), alors les

coordonnées paires de $u^{(k)}$ ne dépendent que des coordonnées impaires de $u^{(k-1)}$ et vice versa. Il nous suffit donc, à chaque étape, de calculer seulement la moitié des coordonnées.

Remarque 2 : Procédé d'accélération de la convergence de Tchébycheff — Quand on a la possibilité de le connaître, le choix de ω optimal dès le début des itérations n'est pas le plus judicieux, un minimum de n itération sera nécessaire à la convergence. L'idée de Tchébycheff est de changer le paramètre ω à chaque itération de la manière suivante : on part de $\omega^{(0)} = 1$, puis on prend $\omega^{(1)} = \frac{1}{1-\frac{\rho(J)^2}{2}}$ et à l'étape $n \geq 1$, on prendra :

$$\omega^{(n+1)} = \frac{1}{1-\frac{\rho(J)^2 \cdot \omega^{(n)}}{4}}$$

La suite $(\omega^{(n)})$ converge en fait vers ω optimal et le procédé de Tchébycheff diminuera, en général, le nombre d'itérations.

5.5.4 Programmation structurée de la méthode S.O.R. avec accélération de Tchébycheff

Au préalable on a initialisé les $u_{i,0}$,$u_{i,m+1}$,$u_{0,j}$ et $u_{n+1,j}$.

```
PROCEDURE S.O.R.(Entrée : n, m : Entier ; ρ : Réel ; A, B, C, D, E, F : Matrice ;
                 Entrée_Sortie U : Matrice) ;
Début
   ω = 1 ;
   k := 1 ;
   ||f|| = 0 ;
   Pour i allant de 1 à n faire
   Début
      Si |f_{i,j}| > ||f||
      Alors faire ||f|| = |f_{i,j}| ;
   Fin ;
   Répéter
      NormeRésidu = 0 ;
      Pour i allant de 1 à n faire
      Début
         Pour j allant de 1 à m faire
         Début
            Si (i + j et k de même parité)
            Alors faire
            Début
               Résidu = a_{i,j}·u_{i+1,j} + b_{i,j}·u_{i–1,j} + c_{i,j}·u_{i,j+1} + d_{i,j}·u_{i,j–1} + e_{i,j}·u_{i,j} – f_{i,j} ;
               Si |Résidu| > NormeRésidu
               Alors faire NormeRésidu = |Résidu| ;
```

$$u_{i,j} = u_{i,j} - \omega \cdot \frac{Résidu}{e_{i,j}} \; ;$$

Fin ;

Fin ;

Fin ;

Si k = 1

Alors $\omega = \dfrac{1}{1 - \dfrac{\rho(J)^2}{2}}$;

Sinon faire $\omega = \dfrac{1}{1 - \dfrac{\rho(J)^2 \cdot \omega}{4}}$;

k = k + 1 ;

J'usqu'à (k = MaxItération) ou (NormeRésidu < $\varepsilon \cdot \|f\|$*) ;*

Si k = MaxItération

Alors Afficher('Méthode inadaptée : convergence trop lente ou non convergence. ') ;

Fin ;

5.6 Cas particulier de l'équation de Poisson

Dans ce cas, on a :

$$a_{i,j} = b_{i,j} = c_{i,j} = d_{i,j} = 1,\ e_{i,j} = -4 \text{ et } f_{i,j} = h_2 \cdot f(x_i, y_j)$$

De plus, on connaît explicitement ρ(J), il est donné par :

$$\rho(J) = \frac{\mathrm{Cos}\left(\dfrac{\pi}{n+1}\right) + \mathrm{Cos}\left(\dfrac{\pi}{m+1}\right)}{2}$$

De manière plus générale, pour des pas δx et δy différents, on a :

$$\rho(J) = \frac{\mathrm{Cos}\left(\dfrac{\pi}{n+1}\right) + \left(\dfrac{\delta x}{\delta y}\right)^2 \cdot \mathrm{Cos}\left(\dfrac{\pi}{m+1}\right)}{1 + \left(\dfrac{\delta x}{\delta y}\right)^2}$$

(Cf. Stoer et Burlisch p. 558) .

Remarque — Pour f = 0, on a l'équation de Laplace Δu = 0 et la méthode des différences finies s'exprime en disant que $u_{i,j}$ est la moyenne des quatre points qui l'entourent $u_{i-1,j}$, $u_{i+1,j}$, $u_{i,j-1}$ et $u_{i,j+1}$. C'est simplement la version discrète de la propriété de la moyenne pour les fonctions harmoniques (une fonction est harmonique si, et seulement si elle vérifie Δu = 0, ce qui équivaut à dire qu'elle vérifie la propriété de la moyenne, et de cette propriété on déduit que le maximum de u ne peut être atteint qu'en un point du bord. — cf. Cartan ch. 7 S 4 —).

5.7 Exemple d'application : distribution de la température en régime stationnaire dans une cheminée

Le problème de la distribution de la température, en régime stationnaire, dans une cheminée nous conduit à résoudre le problème de Dirichlet :

$$\begin{cases} \Delta u = 0 \text{ sur } \Omega = R - R' \\ u = g \text{ sur } \partial R \\ u = h \text{ sur } \partial R' \end{cases}$$

où $R' \subset R$ sont deux rectangles.

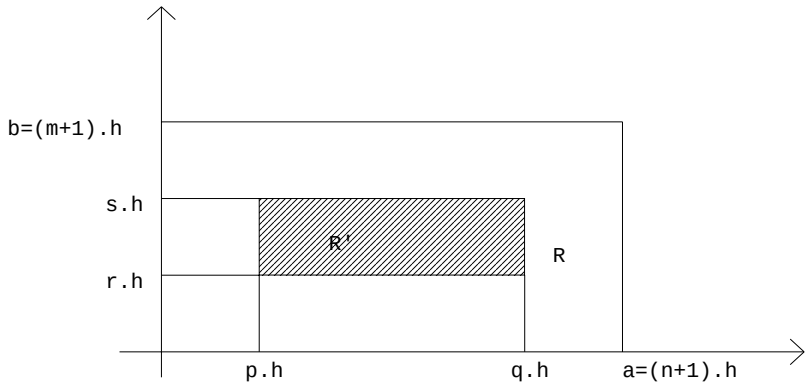


Figure 8.10

En modifiant la procédure ci-dessus, on écrit facilement un programme qui permet de calculer la température en tout point intérieur de R–R'.

5.8 Premier exemple d'équation parabolique : l'équation de la chaleur à une dimension

5.8.1 Introduction

Dans ce paragraphe, on s'intéresse au problème de la propagation de la chaleur par conduction dans une barre isolée de sorte que les échanges de chaleur avec l'extérieur soient limités aux extrémités maintenues à température constante, la distribution de température à un instant initial t = 0 étant donnée.

Avec les notations du paragraphe (5.1.3), on est donc ramené à trouver une fonction u des deux variables $t \geq 0$ et $x \in [0,L]$ solution du problème d'évolution avec conditions initiales :

$$(1) \quad \begin{cases} \dfrac{\partial u}{\partial t}(x,t) = k \cdot \dfrac{\partial^2 u}{\partial x^2}(x,t), \text{ pour } t > 0 \text{ et } 0 < x < L \\ u(x,0) = f(x), \text{ pour } 0 \leq x \leq L \\ u(0,t) = u(L,t) = 0, \text{ pour } t \geq 0 \end{cases}$$

où la constante k est le coefficient de diffusion et L > 0 est la longueur de la barre.

Remarque — Par continuité, on a nécessairement f(0) = f(L) = 0.

5.8.2 Existence et unicité de solutions de (1)

Pour simplifier, on fait k = 1.

Théorème **:** Une solution de (1) est donnée par :

$$u(x,t) = \frac{1}{2\cdot\sqrt{\pi\cdot t}}\cdot\int_R e^{-\frac{(x-y)^2}{4\cdot t}}\cdot f(y)dy$$

Démonstration — On suppose que les fonctions f, $x \mapsto u(x,t)$, $x \mapsto \frac{\partial u}{\partial t}(x,t)$ et $x \mapsto \frac{\partial^2 u}{\partial x^2}(x,t)$ sont dans $L^1(R)$ (i.e. absolument intégrables) avec $\left|\frac{\partial u}{\partial t}(x,t)\right| \le g(x)$ où $g \in L^1(R)$.

En notant :

$$\hat{u}(\nu,t) = \int_R e^{-2i\pi\nu x}\cdot u(x,t)dx$$

la transformée de Fourier en x de u, l'équation (1) devient :

$$(1') \qquad \begin{cases} \frac{\partial \hat{u}}{\partial t} + 4\cdot\pi^2\cdot\nu^2\cdot\hat{u} = 0 \\ \hat{u}(\nu,0) = f(\nu) \\ \hat{u}(0,t) = \hat{u}(L,t) = 0 \end{cases}$$

En intégrant par rapport à t, on déduit que :

$$\hat{u}(\nu,t) = f(\nu)\cdot e^{-4\cdot\pi^2\cdot\nu^2\cdot t}$$

Mais $e^{-4\cdot\pi^2\cdot\nu^2\cdot t}$ est la transformée de Fourier de $\frac{1}{\sqrt{4\cdot\pi\cdot t}}\cdot e^{-\frac{x^2}{4\cdot t}}$ et la transformation de Fourier transformant le produit de convolution en produit ordinaire, avec l'injectivité de cette transformation, on déduit le résultat.

Remarque — Du théorème précédent, on déduit que :

$$\lim_{t\to+\infty} u(x,t) = 0, \text{ pour tout } x \in [0,L],$$

si u est solution de (1) (ce qui est évident d'un point de vue physique si u représente une température).

Pour montrer l'unicité de u, on utilise la quantité :

$$E(t) = \int_0^L u^2(x,t)\frac{dx}{2}$$

qui correspond à une *énergie cinétique* de la barre à l'instant t.

Théorème **:** L'énergie cinétique à l'instant t, vérifie les propriétés suivantes :
(i) $\forall\ t \ge 0$, $E(t) \ge 0$;
(ii) E est décroissante ;

(iii) E est continue à droite en 0 ;
(iv) $E = 0 \Leftrightarrow u = 0$;
cette dernière propriété assurant l'unicité de la solution de (1).

Démonstration — On a, $E'(t) = \int_0^L u(x,t)\cdot\frac{\partial u}{\partial t}(x,t)dx = \int_0^L u(x,t)\cdot\frac{\partial^2 u}{\partial x^2}(x,t)dx$ et une intégration par parties donne $E'(t) = -\int_0^L \left(\frac{\partial u}{\partial x}(x,t)\right)^2 dx \le 0$. Donc E est décroissante.

Avec un théorème de convergence dominée de Lebesgue, on déduit que $\lim_{t\to 0} E(t) = E(0)$, c'est-à-dire que E est continue à droite en 0.

Si E = 0, avec la continuité de u on déduit que u = 0. D'où l'unicité.

Conclusion — Le problème (1) admet une unique solution nulle à l'infini.

Remarque — La solution ainsi obtenue n'est pas satisfaisante d'un point de vue numérique. Et la situation sera encore moins satisfaisante si on ajoute un second membre non nul à l'équation (1), ou si le coefficient de diffusion $\sqrt{k}$ est non constant ou encore si on s'intéresse à l'équation de la chaleur en dimension deux.

On utilisera donc une méthode de différences finies pour obtenir des valeurs approchées de u.

5.8.3 Résolution de l'équation de la chaleur à une dimension par des méthodes de différences finies

On se donne un maillage de [0,+∞[x[0,L] en faisant : une discrétisation en temps de pas $\Delta t > 0$: c'est-à-dire en posant, $t_j = j\cdot\Delta t$, pour tout $j \in N$; et une discrétisation en espace de pas $\Delta x = \frac{L}{n+1}$, où $n \ge 1$: c'est-à-dire en posant, $x_i = i\cdot\Delta x$, pour tout i = 0 , 1 , ... , n + 1.

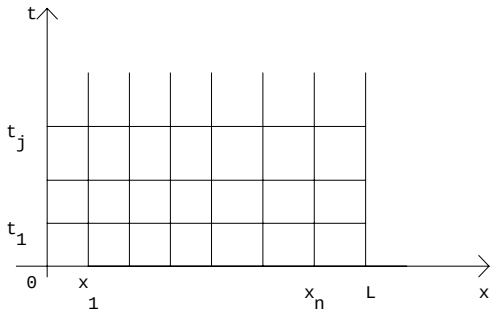


Figure 8.11

On notera $u_{i,j}$ une valeur approchée (inconnue) de $u(x_i,t_j)$ pour $1 \le i \le n$ et $j \ge 1$.

Les valeurs connues étant les :

$$\begin{cases} u_{i,0} = u(x_i, 0) = f(x_i) = f_i, \text{ pour } 0 \le i \le n + 1 \\ u_{0,j} = u(0, t_j) = 0, \text{ pour } j \ge 0 \\ u_{n+1,j} = u(L, t_j) = 0, \text{ pour } j \ge 0 \end{cases}$$

On consignera les $u_{i,j}$, à j fixé, dans un vecteur $u_j = \begin{pmatrix} u_{1,j} \\ . \\ . \\ u_{n,j} \end{pmatrix}$.

Les dérivées partielles d'ordre 2 par rapport à x, seront approximées par différences finies centrées, soit :

$$\frac{\partial^2 u}{\partial x^2}(x_i, t_j) \cong \frac{u_{i+1,j} - 2 \cdot u_{i,j} + u_{i-1,j}}{\Delta x^2}$$

Pour calculer les u_j par une récurrence d'ordre 1 (u_0 étant connu), on a le choix pour approximer $\frac{\partial u}{\partial t}(x_i, t_j)$ entre les différences finies progressives ou régressives.

En utilisant les *différences finies progressives*, on a le schéma :

$$\frac{u_{i,j+1} - u_{i,j}}{\Delta t} = k \cdot \frac{u_{i+1,j} - 2 \cdot u_{i,j} + u_{i-1,j}}{\Delta x^2} \quad (1 \le i \le n \text{ et } j \ge 0)$$

soit, pour tout $j \ge 0$:

$$u_{i,j+1} = u_{i,j} + k \cdot \frac{\Delta t}{\Delta x^2} \cdot (u_{i+1,j} - 2 \cdot u_{i,j} + u_{i-1,j}) \quad (1 \le i \le n)$$

En posant $r = \frac{\Delta t}{\Delta x^2}$ et en tenant compte du fait que $u_{0,j} = u_{n+1,j} = 0$, on peut écrire les choses sous la forme matricielle suivante :

$$u_{j+1} = u_j + k \cdot r \cdot A \cdot u_j \text{ , pour } j \ge 0$$

où on a posé :

$$A = \begin{pmatrix} -2 & 1 & 0 & . & . & 0 \\ 1 & -2 & 1 & 0 & \dots & 0 \\ . & . & . & . & . & . \\ 0 & . & . & . & 1 & -2 \end{pmatrix}$$

Donc les u_j sont déterminés par le *schéma explicite* suivant :

$$(2) \quad \begin{cases} u_0 \text{ donné} \\ u_j = B \cdot u_{j-1} \quad (j \ge 1) \end{cases}$$

où $B = I_d + k \cdot r \cdot A$.

En utilisant les *différences finies régressives*, on a le schéma suivant :

$$\frac{u_{i,j} - u_{i,j-1}}{\Delta t} = k \cdot \frac{u_{i+1,j} - 2 \cdot u_{i,j} + u_{i-1,j}}{\Delta x^2} \quad (1 \le i \le n \text{ et } j \ge 0)$$

soit, pour $j \ge 1$:

$$u_{i,j} - k \cdot r \cdot (u_{i+1,j} - 2 \cdot u_{i,j} + u_{i-1,j}) = u_{i,j-1} \quad (1 \le i \le n)$$

Toujours avec $u_{0,j} = u_{n+1,j} = 0$ et en notant $C = I_d - k \cdot r \cdot A$, les u_j sont définis par le *schéma implicite* :

$$(3) \quad \begin{cases} u_0 \text{ donné} \\ C \cdot u_j = u_{j-1} \quad (j \geq 1) \end{cases}$$

Dans le schéma implicite, u_j est solution d'un système linéaire tridiagonal de matrice constante C. Il semble donc, a priori, que ce schéma soit moins intéressant car plus coûteux en temps de calcul. En fait, on verra par la suite que ce coût est compensé par une meilleure « stabilité » de la méthode.

Remarque — Pour chacune des deux méthodes, l'erreur par rapport à la solution exacte est un O(h). Une idée pour améliorer la précision est alors de faire la moyenne entre les méthodes implicites et explicites, ce qui donne le *schéma de Crank-Nicholson* :

$$\frac{u_{i,j+1} - u_{i,j}}{\Delta t} = \frac{k}{2} \cdot \frac{u_{i+1,j+1} - 2 \cdot u_{i,j+1} + u_{i-1,j+1} + u_{i+1,j} - 2 \cdot u_{i,j} + u_{i-1,j}}{\Delta x^2} \quad (1 \leq i \leq n \text{ et } j \geq 0)$$

Ce qui donne un schéma implicite, écrit sous forme matricielle :

$$(4) \quad \begin{cases} u_0 \text{ donné} \\ \left(I_d - \frac{k}{2} \cdot r \cdot A \right) \cdot u_j = \left(I_d + \frac{k}{2} \cdot r \cdot A \right) \cdot u_{j-1} \quad (j \geq 1) \end{cases}$$

Dans ce cas, non seulement la méthode est stable, mais elle est aussi d'ordre 2, c'est-à-dire que l'erreur est un $0(h^2)$.

De manière plus générale, en prenant les moyennes des schémas implicites et explicites pondérées par ω et $1 - \omega$, avec $0 \leq \omega \leq 1$, on a le schéma :

$$\frac{u_{i,j+1} - u_{i,j}}{\Delta t} = \omega \cdot \frac{k}{\Delta x^2} \cdot \left\{ u_{i+1,j+1} - 2 \cdot u_{i,j+1} + u_{i-1,j+1} \right\} + (1 - \omega) \cdot \frac{k}{\Delta x^2} \cdot \left\{ u_{i+1,j} - 2 \cdot u_{i,j} + u_{i-1,j} \right\}$$

soit, sous forme matricielle :

$$(5) \quad \begin{cases} u_0 \text{ donné} \\ (I_d - \omega \cdot k \cdot r \cdot A) \cdot u_j = (I_d + (1 - \omega) \cdot k \cdot r \cdot A) \cdot u_{j-1} \quad (j \geq 1) \end{cases}$$

Pour $\omega = 0$, on a le schéma explicite donné par les différences finies progressives, pour $\omega = 1$, on a le schéma implicite donné par les différences finies régressives et pour $\omega = \frac{1}{2}$, on a le schéma implicite de Crank-Nicholson.

La matrice $(I_d - \omega \cdot k \cdot r \cdot A)$ est toujours symétrique définie positive car symétrique, à diagonale strictement dominante et de coefficients diagonaux tous strictement positifs. On aura donc intérêt, pour résoudre le système linéaire donnant u_j, dans les méthodes implicites, à utiliser la méthode de décomposition de Cholesky, qui va se simplifier à cause du caractère tridiagonal des matrices.

5.8.4 Problèmes de stabilité

Dans ce paragraphe, on va se contenter d'une présentation succincte de la notion de stabilité d'une méthode numérique pour résoudre l'équation de la chaleur. Pour plus de détails sur ce sujet, on pourra se reporter à Euvrard (p. 71 à 84) ou Numerical Recipes (p. 625 à 640)

On a vu que si u est solution de (1), alors $\lim_{t\to+\infty} u(x,t) = 0$, pour tout $x \in [0,L]$.

Une condition nécessaire de stabilité, pour une méthode itérative donnant les valeurs approchées $u_{i,j}$ des $u(x_i,t_j)$, sera alors : $\lim_{j\to+\infty} u_j = 0$, pour tout $n \geq 1$.

Si on considère le schéma (5), avec $0 \leq \omega \leq 1$, en notant :

$$B_\omega = I_d - \omega\cdot k\cdot r\cdot A \text{ et } C_\omega = I_d + (1-\omega)\cdot k\cdot r\cdot A$$

on a :

$$u_j = B_\omega^{-1}\cdot C_\omega \cdot u_{j-1} = \left(B_\omega^{-1}\cdot C_\omega\right)^j \cdot u_0, \text{ pour } j \geq 0$$

et la condition de stabilité va se traduire par :

$$\lim_{j\to+\infty}\left(B_\omega^{-1}\cdot C_\omega\right)^j = 0, \text{ pour tout } n \geq 1$$

ce qui équivaut à :

$$\sigma(B_\omega^{-1}\cdot C_\omega) < 1, \text{ pour tout } n \geq 1$$

où σ désigne le rayon spectral.

Lemme **:** Les valeurs propres de la matrice $A = \begin{pmatrix} -2 & 1 & 0 & . & . & 0 \\ 1 & -2 & 1 & 0 & \dots & 0 \\ . & . & . & . & . & . \\ 0 & . & . & . & 1 & -2 \end{pmatrix}$ sont les :

$$\lambda_k = -4\cdot\mathrm{Sin}^2(\theta_k) \text{ , où } \theta_k = \frac{k\cdot\pi}{2\cdot(n+1)} \quad (k = 1 , \dots , n)$$

Démonstration — Dire que λ est valeur propre de A, équivaut à dire qu'il existe un vecteur non nul x tel que $A\cdot x = \lambda\cdot x$, ce qui équivaut au système :

$$x_{i-1} - 2\cdot x_i + x_{i+1} = \lambda\cdot x_i \qquad (i = 1 , \dots , n)$$

en posant $x_0 = x_{n+1} = 0$.

Si on pose $x_i = \mathrm{Sin}(i\cdot\theta)$, pour $i = 0, 1, \dots, n+1$, on a :

$$x_{i-1} + x_{i+1} = 2\cdot\cos(\theta)\cdot x_i$$

et il vient : $2\cdot(\cos(\theta) - 1)\cdot x_i = \lambda\cdot x_i$. Ce qui donne :

$$\lambda = 2\cdot(\cos(\theta) - 1) = -4\cdot\sin^2(\theta/2)$$

Enfin, en tenant compte de $x_{n+1} = \sin((n+1)\cdot\theta) = 0$, on déduit que les seules valeurs possibles de θ sont les $\frac{k\cdot\pi}{n+1}$, pour $k = 1, \dots , n$. D'où le résultat.

On déduit donc que les valeurs propres de $B_\omega^{-1}\cdot C_\omega$ sont les :

$$\mu_k = \frac{1+(1-\omega)\cdot k\cdot r\cdot\lambda_k}{1-\omega\cdot k\cdot r\cdot\lambda_k} = \frac{1-4\cdot(1-\omega)\cdot k\cdot r\cdot\mathrm{Sin}^2(\theta_k)}{1+4\cdot\omega\cdot k\cdot r\cdot\mathrm{Sin}^2(\theta_k)}$$

Et la condition de stabilité va se traduire par :

$$2\cdot k\cdot r\cdot\sin^2(\theta_k)\cdot(1-2\cdot\omega) < 1 \qquad (1 \leq k \leq n \text{ et pour tout } n \geq 1)$$

Pour $\omega \in [\frac{1}{2},1]$, cette condition est toujours vérifiée, on dit alors que la méthode est *inconditionnellement stable*.

Si $\omega \in [0,\frac{1}{2}[$, on a une condition de stabilité sur $r = \frac{\Delta t}{\Delta x^2}$ donnée par :

$$\mathrm{Sin}^2(\theta_k)\cdot r < \frac{1}{2\cdot k\cdot(1-2\cdot\omega)}, \text{ pour } k = 1, \ldots, n \text{ et } n \geq 1$$

ce qui équivaut a :

$$\mathrm{Sin}^2(\theta_n)\cdot r < \frac{1}{2\cdot k\cdot(1-2\cdot\omega)}, \text{ pour tout } n \geq 1$$

et en faisant $n \to +\infty$, on a une condition nécessaire de stabilité :

$$r < \frac{1}{2\cdot k\cdot(1-2\cdot\omega)}$$

En résumé **:**

- Pour $\frac{1}{2} < \omega < 1$, la méthode est *inconditionnellement stable* ;
- pour $0 < \omega < \frac{1}{2}$, la méthode est *conditionnellement stable***.**

En particulier, les méthodes implicites pour $\omega = 1$ et $\omega = \frac{1}{2}$ sont inconditionnellement stables et la méthode explicite pour $\omega = 0$ est conditionnellement stable, la condition de stabilité étant $r < \frac{1}{2\cdot k}$.

5.9 Equations paraboliques linéaires à une variable d'espace avec conditions initiales

On considère ici le cas d'une équation parabolique, avec conditions initiales du type :

$$\begin{cases} \frac{\partial u}{\partial t}(x,t) = a(x)\cdot\frac{\partial^2 u}{\partial x^2}(x,t) + b(x)\cdot\frac{\partial u}{\partial x}(x,t) + c(x)\cdot u(x,t) + d(x) \quad (0 \leq x \leq L,\ t \geq 0) \\ u(x,0) = h(x), \text{ pour} 0 \leq x \leq L \\ u(0,t) = f(t);\ u(L,t) = g(t) \quad (t \geq 0) \end{cases}$$

On considère le même maillage qu'en (5.8) et on garde les mêmes notations.

Le schéma (5) du paragraphe (5.8.3), avec paramètre $\omega \in [0,1]$ s'écrit alors :

$$\frac{u_{i,j+1} - u_{i,j}}{\Delta t} = \omega\cdot\left\{a_i\cdot\frac{u_{i+1,j+1} - 2\cdot u_{i,j+1} + u_{i-1,j+1}}{\Delta x^2} + b_i\cdot\frac{u_{i+1,j+1} - u_{i-1;j+1}}{2\cdot\Delta x} + c_i\cdot u_{i,j+1} + d_i\right\}$$
$$+ (1-\omega)\cdot\left\{a_i\cdot\frac{u_{i+1,j} - 2\cdot u_{i,j} + u_{i-1,j}}{\Delta x^2} + b_i\cdot\frac{u_{i+1,j} - u_{i-1;j}}{2\cdot\Delta x} + c_i\cdot u_{i,j} + d_i\right\}$$

pour $j \geq 0$ et $i = 1, \ldots, n$.

Ce qui peut s'écrire :

$$\beta_i\cdot u_{i-1,j+1} + \alpha_i\cdot u_{i,j+1} + \gamma_i\cdot u_{i+1,j+1} = \beta'_i\cdot u_{i-1,j} + \alpha'_i\cdot u_{i,j} + \gamma'_i\cdot u_{i+1,j} + d_i$$

pour $j \geq 0$ et $i = 1, \ldots, n$,

avec :

$$\begin{cases} \beta_i = -\omega \cdot \Delta t \cdot \left\{ \dfrac{a_i}{\Delta x^2} - \dfrac{b_i}{2 \cdot \Delta x} \right\} \\ \alpha_i = 1 - \omega \cdot \Delta t \cdot \left\{ c_i - \dfrac{2 \cdot a_i}{\Delta x^2} \right\} \\ \gamma_i = -\omega \cdot \Delta t \cdot \left\{ \dfrac{a_i}{\Delta x^2} + \dfrac{b_i}{2 \cdot \Delta x} \right\} \end{cases}$$

et :

$$\begin{cases} \beta'_i = (1-\omega) \cdot \Delta t \cdot \left\{ \dfrac{a_i}{\Delta x^2} - \dfrac{b_i}{2 \cdot \Delta x} \right\} \\ \alpha'_i = 1 + (1-\omega) \cdot \Delta t \cdot \left\{ c_i - 2 \cdot \dfrac{a_i}{\Delta x^2} \right\} \\ \gamma'_i = (1-\omega) \cdot \Delta t \cdot \left\{ \dfrac{a_i}{\Delta x^2} + \dfrac{b_i}{2 \cdot \Delta x} \right\} \end{cases}$$

ce qui donne encore u_{j+1} comme solution d'un système tridiagonal.

Remarque — Pour $\omega = 0$, on a une méthode explicite, pour $\omega = \dfrac{1}{2}$, c'est la méthode de Crank-Nicholson et pour $\omega \neq 0$ c'est une méthode implicite.

Pour $0 \leq \omega < \frac{1}{2}$, la méthode est conditionnellement stable et pour $\frac{1}{2} \leq \omega \leq 1$, la méthode est inconditionnellement stable.

6. Exemple d'application : Phénomène de sustentation par utilisation d'une source de pression. Le patin hydrostatique

6.1 Présentation du problème

Les dispositifs de lubrification (ex. glissières de machines) utilisent l'effet de sustentation hydrostatique : laminage par utilisation d'un fluide sous pression.

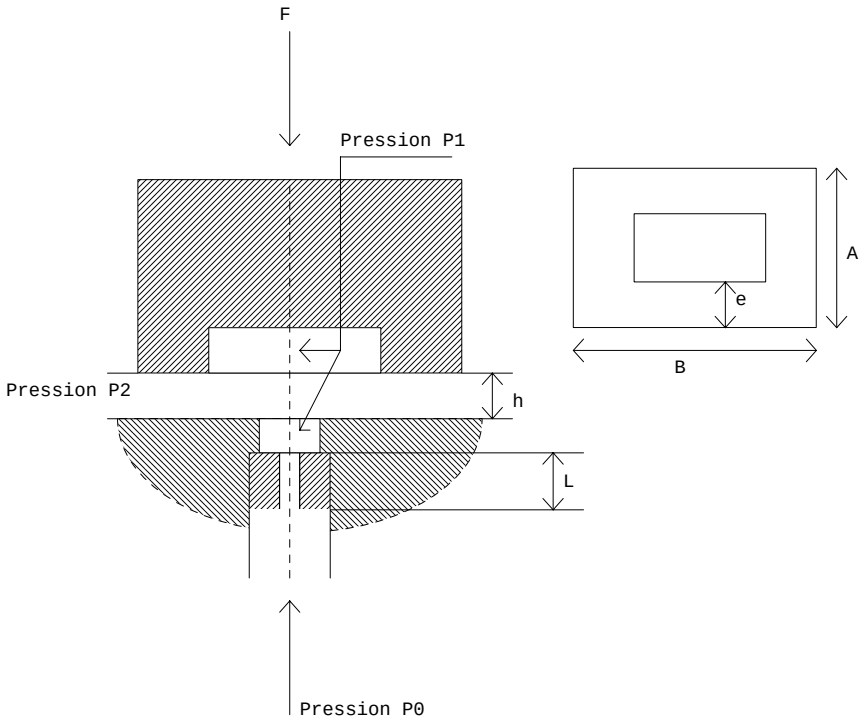


Figure 8.12

Les données mécaniques sont les suivantes : un patin rectangulaire est alimenté par un capillaire de diamètre d_0 et on désigne par :

- P0 la pression d'alimentation ;
- P1 la pression dans la chambre ;
- P2 la pression à la sortie : $P2 = P_{at}$ (pression atmosphérique) ;
- la pression utile est la pression effective notée $P_{eff} = P - P_{at}$;

On utilise le rapport de pression : $r = \frac{P_{eff}}{P0_{eff}}$, pour $0 < r < 1$

On note :

$$\begin{cases} \text{Kq le coefficient de débit} \\ \text{Ks le coefficient de surface} \\ \text{Kf le coefficient d'effort} \end{cases}$$

Pour une application numérique, on pourra prendre les valeurs suivantes : A = 50·mm, B = 70·mm et C = 10·mm.

Les lois gouvernant l'équilibre du patin sont :

$$(1) \quad P0 - P1 = \frac{128 \cdot \mu}{\pi} \cdot \frac{L}{d_0^{\,4}} \cdot Q_{v_0} \quad \text{(Loi des écoulements laminaires)}$$

$$(2) \quad P1 - P2 = \frac{1}{Kq} \cdot \frac{\mu \cdot Q_{v_0}}{h^3}$$

6.2 Détermination du coefficient Ks

Pour chaque hauteur de stabilisation h, on mesure : $P0_{eff}$, $P1_{eff}$ et F = f – P, où :

- f désigne la charge due à la pression ;
- F désigne la charge mesurée (compensation du poids du coulisseau).

On écrit alors : $f = P1_{eff} \cdot S_{eff}$, où S_{eff} est une surface effective inférieure à la surface réelle, pour tenir compte de la chute de pression.

On pose $S_{eff} = Ks \cdot A \cdot B$, avec $0 < Ks < 1$ et on écrit $F = Ks \cdot A \cdot B \cdot P1_{eff} + b_1$, soit en posant $a_1 = Ks \cdot A \cdot B$:

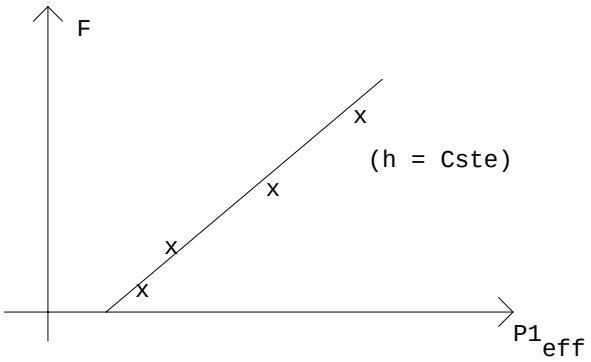


Figure 8.13

$$F = a_1 \cdot P1_{eff} + b_1$$

Le coefficient Ks peut alors s'obtenir à l'aide d'une régression affine.

Remarques — On aura intérêt à calculer Ks pour chaque groupe de valeurs mesurées à h constante. Dans le cas de 4 groupes de valeurs, on obtiendra des valeurs différentes de Ks. La comparaison de ces valeurs permet de juger, si la loi de comportement à adopter dans le laminage, est indépendante ou non de la hauteur.

Si les différentes valeurs des Ks varient peu, on pourra prendre la valeur moyenne Ks_{moy}, pour modéliser le comportement du patin dans son ensemble. L'extrapolation pour des valeurs de h inférieures à la valeur minimale des essais, devra cependant être envisagée avec prudence.

6.3 Détermination du coefficient Kq

Kq peut être déduit des tableaux de mesures, en effet, $\frac{(1)}{(2)}$ donne :

$$\frac{P0-P1}{P2-P1}=\frac{\frac{128\cdot\mu}{\pi}\cdot\frac{L}{{d_0}^4}\cdot Q_{v_0}}{\frac{1}{Kq}\cdot\frac{\mu}{h^3}\cdot Q_{v_0}}$$

$$\frac{1-r}{r}=\frac{128\cdot Kq\cdot L\cdot h^3}{\pi\cdot {d_0}^4}$$

$$h=\left(\frac{\pi\cdot {d_0}^4}{128\cdot Kq\cdot L}\right)^{\frac{1}{3}}\cdot\left(\frac{1-r}{r}\right)^{\frac{1}{3}}$$

Soit en posant $X=\left(\frac{1-r}{r}\right)^{\frac{1}{3}}$: $h = a\cdot X$.

Si Kq est constant, alors a est constant et une régression affine donnera Kq.

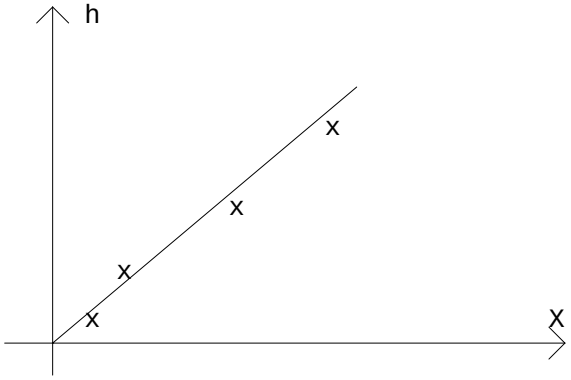


Figure 8.14

6.4 Raideur du patin

Il s'agit d'un paramètre mécanique d'une grande importance, notamment dans l'application aux glissières ; elle traduit l'aptitude à la variation de h sous l'action d'une variation de f.

La *force portante* est $f = Ks\cdot A\cdot B\cdot P1_{eff}$ et $r=\frac{P1_{eff}}{P0_{eff}}$, donc :

$$f=(Ks\cdot A\cdot B\cdot P0_{eff})\cdot r = Kf\cdot r$$

où Kf est le *coefficient d'effort*.

La *raideur* est définie par :

$$C=-\frac{df}{dh}=-\frac{df}{dr}\cdot\frac{dr}{dh}=-Kf\cdot\frac{dr}{dh}$$

or $h=a\cdot\left(\frac{1-r}{r}\right)^{\frac{1}{3}}$, donc $\frac{dh}{dr}=-\frac{3}{a}\cdot r^{\frac{4}{3}}\cdot(1-r)^{\frac{2}{3}}$, d'où :

$$C = \frac{3 \cdot Kf}{a} \cdot r^{\frac{4}{3}} \cdot (1 - r)^{\frac{2}{3}}$$

La raideur maximale du patin est obtenue pour $\frac{dC}{dr} = 0$, soit $r = \frac{2}{3}$.

6.5 Modélisation du champ de pression

6.5.1 Introduction

Le patin rectangulaire étant chargé uniformément (charge P), une alimentation d'huile à la partie inférieure assurant le maintien d'un film d'huile de hauteur h.

Pour les calculs, on admettra que la pression P_1 est uniforme dans tout le volume intérieur. La pression est P_2 à la sortie du film.

6.5.2 Les équations de laminage

- Bilan de masse : $\frac{\partial \rho}{\partial t} + \mathrm{Div}(\rho \cdot \vec{V}) = 0$, où ρ est la masse volumique, supposée constante et $\vec{V}$ est le champ des vitesses.

Dans la phase de mouvement stationnaire, on a $\frac{\partial \rho}{\partial t} = 0$. L'équation se réduit donc à :

$$\mathrm{Div}(\vec{V}) = 0 \qquad (1)$$

- Bilan de quantité de mouvement (forme locale) : $\rho \cdot \gamma_i = \rho \cdot g_i + \frac{\partial \sigma_{ij}}{\partial x_j}$ avec :

  $\sigma_{ij} = -p \cdot \delta_{ij} + 2 \cdot \mu \cdot D_{ij} + \lambda \cdot \theta \cdot \delta_{ij}$;

  σ_{ij} : Tenseur des contraintes ;

  D_{ij} : Tenseur des taux de déformation : $D_{ij} = \frac{1}{2} \cdot \left(\frac{\partial V_i}{\partial x_j} + \frac{\partial V_j}{\partial x_i} \right)$

  $q = \mathrm{Div}(\vec{V}) = 0$ (d'après (1))

  p est la pression en un point

  μ est le coefficient de viscosité dynamique supposé constant est un autre coefficient de viscosité.

Finalement, cette équation se réduit à :

$$\rho \cdot \gamma_i = \rho \cdot g_i + \mu \cdot \sum_{j=1}^{3} \frac{\partial^2 V_i}{\partial x_j^2} - \frac{\partial p}{\partial x_i}$$

soit encore :

$$\rho \cdot \vec{\gamma} = \rho \cdot \vec{g} - \overrightarrow{\mathrm{Grad}}(p) + \mu \cdot \overrightarrow{\Delta V} \qquad (2)$$

où $\overrightarrow{\Delta V}$ est le laplacien de la vitesse.

6.5.3 Résolution dans le cas où les termes d'accélération sont très petits

Dans ce cas le système se réduit à :

$$\begin{cases} \mathrm{Div}\left(\vec{V}\right) = \vec{0} \\ -\rho\cdot\vec{g} + \overrightarrow{\mathrm{Grad}}(p) + \mu\cdot\overrightarrow{\Delta V} \end{cases}$$

Notations et propriétés

En posant : $\hat{P} = P + \rho\cdot g\cdot z$, le premier membre s'écrit $\overrightarrow{\mathrm{Grad}}\left(\hat{P}\right)$ et le système :

$$\begin{cases} \mathrm{Div}\left(\vec{V}\right) = 0 \\ \overrightarrow{\mathrm{Grad}}\left(\hat{P}\right) = \mu\cdot\overrightarrow{\Delta V} \end{cases}$$

Mais, $\overrightarrow{\Delta V} = -\overrightarrow{\mathrm{Rot}}\left(\overrightarrow{\mathrm{Rot}}\left(\vec{V}\right)\right)$ et $\mathrm{Div}(\overrightarrow{\mathrm{Rot}}\left(\vec{V}\right)) = 0$, d'où avec (2) : $\Delta\hat{P} = 0$

Remarque — Dans le problème posé, le terme $\rho\cdot g\cdot z$ est négligeable devant P et l'équation se réduit à : $\Delta P = 0$.

7. Exercices

7.1 Un problème de thermique

Dans un problème d'échange thermique, on est amené à rechercher deux fonctions f et g définies sur [0,+∞[x[0,+∞[et telles que :

$$(1)\quad \begin{cases} \dfrac{\partial f}{\partial y}(x,y) = g(x,y) - f(x,y), \text{ pour tous } x \geq 0 \text{ et } y \geq 0 \\ \dfrac{\partial g}{\partial x}(x,y) = f(x,y) - g(x,y), \text{ pour tous } x \geq 0 \text{ et } y \geq 0 \\ f(x,0) = \varphi(x), \text{ pour tout } x \geq 0 \\ g(0,y) = \psi(y), \text{ pour tout } y \geq 0 \end{cases}$$

où φ et ψ sont des fonctions données.

On choisit une valeur limite, pour x et y, qui sera notée $a > 0$.

En vue de résoudre (1), par discrétisation, on définit un maillage du carré Ω = [0,a]x[0,a] en posant $h = \dfrac{a}{n}$, $x_i = i\cdot h$ pour $0 \leq i \leq n$ et $y_j = j\cdot h$ pour $0 \leq j \leq n$

On notera f_{ij} et g_{ij} des valeurs approchées de $f(x_i,y_j)$ et $g(x_i,y_j)$.

1°) En approchant les dérivées partielles par des différences finies progressives, traduire le problème (1).

2°) Décrire un algorithme de résolution du problème discrétisé obtenu en 1°). Ecrire ensuite la programmation structurée correspondante.

3°) Application numérique : On prend $\varphi(x) = 1$ et $\psi(y) = 0$. Pour $n = 3$ et $a = 1$, donner les valeurs approchées f_{ij} et g_{ij}.

4°) Quels problèmes pose l'utilisation de différences finies régressives ?

7.2 Méthode des éléments finis

On considère le problème aux limites sur [0,1] :

$$(1) \quad \begin{cases} -\left(p(x)\cdot u'(x)\right)' + q(x)\cdot u(x) = f(x) \ (0 < x < 1) \\ u(0) = u(1) = 0 \end{cases}$$

où les fonctions p, q, et sont données telles que :

$$\begin{cases} p \in C^1([0,1]); \forall x \in [0,1],\ p(x) \geq p_0 > 0 \\ q \in C^0([0,1]); \forall x \in [0,1],\ q(x) \geq 0 \\ f \in C^0([0,1]) \end{cases}$$

et la fonction inconnue u est cherchée dans $C^2([0,1])$.

On admettra, qu'avec nos hypothèses, une telle fonction u existe.

La méthode étudiée pour résoudre ce problème est la *méthode variationnelle de Rayleigh, Ritz et Galerkin*, ou *méthode des éléments finis*.

On note E = { v : [0,1] → R ; continue et C^1 par morceaux ; v(0) = v(1) = 0}.

Pour tout n ≥ 1 dans N, on pose : $h = \dfrac{1}{n+1}$, $x_i = i\cdot h$, pour i = 0, 1, ..., n + 1.

On notera u la solution exacte de (1).

1°) En utilisant une intégration par parties, montrer que u est aussi solution du problème :

$$(2) \quad \forall v \in E,\ \int_0^1 (p\cdot u'\cdot v' + q\cdot u\cdot v)(x)dx = \int_0^1 (f\cdot v)(x)dx$$

Dans la suite du problème, on pose, pour tous v, w dans E :

$$a(v,w) = \int_0^1 (p\cdot w'\cdot v' + q\cdot w\cdot v)(x)dx \text{ et } b(v) = \int_0^1 (f\cdot v)(x)dx$$

2°) Montrer que a est une forme bilinéaire symétrique définie positive.

3°) En déduire que (1) admet une unique solution.

Pour résoudre (2) de façon approchée, l'idée de la méthode des éléments finis est de remplacer l'espace vectoriel de dimension infinie E par un s. e. v. E_h de dimension finie, le paramètre h > 0 étant lié à la dimension de E_h et destiné à tendre vers 0.

Le problème (2) sera alors remplacé par le problème discret :

$$(2)_h \quad \begin{cases} \text{Trouver } u_h \text{ dans } E_h \text{ tel que} \\ \forall v \in E_h,\ a(u_h, v) = b(v) \end{cases}$$

On espère que $\lim_{h\to 0}(u_h) = 0$. Et ce résultat sera atteint avec une précision qui dépendra du choix du s. e. v. E_h. On admet donc que la méthode converge.

Si $\{v_1, ..., v_n\}$ est une base de E_h, on écrira : $u_h = \sum_{i=1}^{n} \alpha_i \cdot v_i$.

3°) Montrer que le problème $(2)_h$ admet une unique solution.

4°) On désigne par E_h le s. e. v. de E engendré par les fonctions $v_1, ..., v_n$, où v_i est la fonction affine par morceaux dont le graphe est représenté sur la figure 8.15 :

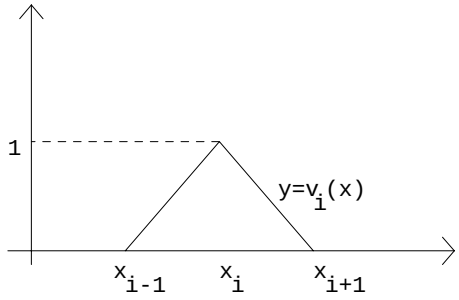


Figure 8.15

(a) Expliciter le système linéaire dont est solution $\alpha = (\alpha_1,...,\alpha_n)$.
(b) Décrire une procédure de résolution du système obtenu en (a), en supposant que l'on dispose d'une procédure de calcul des intégrales définies.
(c) En prenant u_h comme valeur approchée de u, expliquer pourquoi les coefficients α_i donnent des valeurs approchées des $u(x_i)$, pour i = 1, ..., n.
(d) Comparer cette méthode à celle utilisant les différences finies centrées.

7.3

Ecrire les procédures de résolution d'une équation aux dérivées partielles de type parabolique.

8. Programmation Ada

8.1 Spécification du paquetage DIRICHLET

Dans ce paquetage on trouve des procédures de résolution d'équations différentielles d'ordre 2 avec conditions aux limites. On pourra également en écrire une version générique.

```
with CURVE, COMMON_MATH2 ;
use CURVE, COMMON_MATH2 ;
package DIRICHLET is
procedure RESOLUTION_DIRICHLET(p, q, f : in FONCTION ; -- § 3.1
             a, b, ya, yb : in FLOAT  ; C : in out COURBE) ; -- § 3.2
procedure RESOLUTION_POISSON(p, f : in FONCTION ; y0, y1 : in FLOAT ;
             C : in out COURBE) ;
end DIRICHLET ;
```

8.2 Démonstration du paquetage DIRICHLET

```
with TEXT_IO, IIO, FIO, CRT, COMMON_MATH0, MATH0, CURVE,
     MATH2, MATH3, DIRICHLET ;
use TEXT_IO, IIO, FIO, CRT, COMMON_MATH0, MATH0, CURVE,
    MATH2, MATH3, DIRICHLET ;
procedure DEM_DIRI is
a, b, ya, yb, y0, y1 : FLOAT ;
p, q, f : FONCTION ;
```

```
C : COURBE(300) ;
CHOIX : INTEGER range 1..3 ;

procedure DEMO_POISSON is
begin
   MODE_AFFICHAGE ;
   PUT_LINE("Resolution de y'' + p(x).y(x) = f(x) sur [0,1] ") ;
   PUT_LINE("avec p negative et y(0) et y(1) donnes.") ;
   GET_LINE(p,"Entrer p(x) : ") ;
   GET_LINE(f,"Entrer f(x) : ") ;
   ENTRER_REEL(y0,"y(0) = ") ;
   ENTRER_REEL(y1,"y(1) = ") ;
   RESOLUTION_POISSON(p,f,y0,y1,C) ;
   CLOSE(IMP) ;
end DEMO_POISSON ;

procedure DEMO_DIRICHLET is
begin
   MODE_AFFICHAGE ;
   PUT_LINE("Resolution d'une equation differentielle lineaire ") ;
   PUT_LINE("du second ordre avec conditions initiales.") ;
   PUT_LINE(" y''(x) + p(x).y'(x) + q(x).y(x) = f(x) (a < x < b )") ;
   PUT_LINE(" avec y(a) et y(b) donnes.") ;
   GET_LINE(p,"Entrer p(x) : ") ;
   GET_LINE(q,"Entrer q(x) : ") ;
   GET_LINE(f,"Entrer f(x) : ") ;
   ENTRER_REEL(a,"a = ") ;
   ENTRER_REEL(ya,"ya = ") ;
   ENTRER_REEL_BORNE(a,1.0E12,b,"b = ") ;
   ENTRER_REEL(yb,"yb = ") ;
   RESOLUTION_DIRICHLET(p,q,f,a,b,ya,yb,C) ;
   CLOSE(IMP) ;
end DEMO_DIRICHLET ;

begin
    loop
       PUT_LINE(" -1- Equation du type de Poisson") ;
       PUT_LINE(" -2- Equation du type de Dirichlet") ;
       PUT_LINE(" -3- FIN") ;
       ENTRER_ENTIER_BORNE(1,3,CHOIX," Votre choix : ") ;
       case CHOIX is
         when 1 => DEMO_POISSON ;
         when 2 => DEMO_DIRICHLET ;
         when 3 => exit ;
       end case ;
       if LIRE_REPONSE("Voulez vous l'affichage des resultats? ") then
          for k in 0..C.n
          loop
             PUT("x = ") ; PUT(C.u(k),4,6,0) ;
             PUT("     y = ") ; PUT(C.v(k),4,6,0) ;
             NEW_LINE ;
             if ((k + 1) mod 20 = 0) then
                PAUSE ;
```

```
            end if ;
          end loop ;
       end if ;
       NEW_LINE ;
       if LIRE_REPONSE("Voulez vous le trace graphique? ") then
          TRACE_UNE_COURBE(C) ;
          SORTIE_GRAPHIQUE ;
       else
          CLRSCR ;
       end if ;
    end loop ;
end DEM_DIRI ;
```

8.3 Spécification du paquetage NIVEAUX

Ce paquetage est utilisé pour tracer les courbes de niveaux d'une fonction u(x,y) solution d'une équation elliptique. Sa conception est inspirée du logiciel Modulog. Pour plus de détails, on se reportera à l'exercice 10 du chapitre 2.

```
with MATH2, COMMON_MATRIX ;
use MATH2, COMMON_MATRIX ;
package NIVEAUX is
procedure DISCRETISE_FONCTION(f : in FONCTION ;
             xMin, xMax, yMin, yMax : in FLOAT ; M : in out MATRICE) ;
procedure TRACE_NIVEAU(XMin,XMax,YMin,YMax : in FLOAT ;
             M : in MATRICE ; f : in FLOAT) ;
procedure TRACE_EQUIPOTENTIELLES(M : in MATRICE ;
             xMin, xMax, yMin, yMax : in FLOAT ;
             u0 : in VECTEUR ; AXES : in BOOLEAN := FALSE) ;
end NIVEAUX ;
```

8.4 Spécification du paquetage EDP_ELLIPTIQUES

Dans ce paquetage on décrit des procédures de résolution de l'équation de Poisson $\Delta u = f$ et d'une équation plus générale de type elliptique. On pourra également en écrire une version générique.

```
with MATH2, COMMON_MATRIX ;
use MATH2, COMMON_MATRIX ;
package EDP_ELLIPTIQUES is
procedure RESOLUTION_POISSON(f, g1, g2, g3, g4 : in FONCTION ;
             RHO_JACOBI, h : in FLOAT ;  U : in out MATRICE) ; -- § 5.6
PROCEDURE RESOLUTION_ELLIPTIQUE(f, g1, g2, g3, g4,
             Alpha, Beta, Gama, Delt, Epsi : in FONCTION ;
             Omega, h : in FLOAT ; U : in out MATRICE) ;
procedure AFFICHER_SOLUTION(U : in MATRICE) ; -- § 5.5.4
procedure RESOLUTION_POISSON_2(f, g1, g2, g3, g4,
             h1, h2, h3, h4 : in FONCTION ;
             p , q , r , s : in INTEGER ; RHO_JACOBI, h : in FLOAT ;
             U : in out MATRICE) ;
procedure AFFICHER_SOLUTION_2(p, q, r, s : in INTEGER ;
             U : in MATRICE) ;
```

```
ERREUR_CONVERGENCE exception ;
end EDP_ELLIPTIQUES ;
```

8.5 Démonstration du paquetage EDP_ELLIPTIQUES

```
with TEXT_IO, IIO, CRT, COMMON_MATH0, MATH0, COMMON_MATH1, MATH1,
     MATH2, COMMON_MATRIX, NIVEAUX, EDP_ELLIPTIQUES ;
use TEXT_IO, IIO, CRT, COMMON_MATH0, MATH0, COMMON_MATH1, MATH1,
    MATH2, COMMON_MATRIX, NIVEAUX, EDP_ELLIPTIQUES ;
procedure DEM_ELLI is
CHOIX : INTEGER range 1..4 ;

procedure DEMO_EDP_POISSON is
f, g1, g2, g3, g4 : FONCTION ;
n, m, NB_COURBES : INTEGER ;
h, RHO_JACOBI : FLOAT ;
begin
   MODE_AFFICHAGE ;
   PUT_LINE("    Resolution de l'equation de Poisson : ") ;
   PUT_LINE("    Delta(u) = f sur un rectangle [0,a]x[o,b] ;") ;
   PUT_LINE("    avec les conditions au bord : u(x,y) = g(x,y) ;") ;
   PUT_LINE("    ou g(x,0) = g1(x), g(x,b) = g2(x),
                  g(0,y) = g3(y), g(a,y) = g4(y).") ;
   PUT_LINE("On utilise les notations suivantes : ") ;
   PUT_LINE(" a = (n + 1).h ; b = (m + 1).h ,
             ( h > 0 pas de discretisation donne ) ;") ;
   PUT_LINE("un maillage du rectangle etant defini par : ") ;
   PUT_LINE(" (x(i),y(j)) = (i.h,j.h) pour i = 0, ..., n + 1
             et j = 0, ..., m + 1 . ") ;
   PAUSE ; CLRSCR ;
   PUT_LINE("ENTREE DES DONNEES POUR L'EQUATION DE POISSON") ;
   ENTRER_ENTIER_BORNE(2,DIM_MAX,n,
                       "Nombre de points sur l'axe des x : ") ;
   ENTRER_ENTIER_BORNE(2,DIM_MAX,m,
                       "Nombre de points sur l'axe des y : ") ;
   ENTRER_REEL(h,"h = ") ;
   GET_LINE(f,"f(x,y) = ") ;
   GET_LINE(g1,"Condition a y = 0 , g1(x) = ") ;
   GET_LINE(g2,"Condition a y = b , g2(x) = ") ;
   GET_LINE(g3,"Condition a x = 0 , g3(y) = ") ;
   GET_LINE(g4,"Condition a x = a , g4(y) = ") ;
   RHO_JACOBI :=0.5*(COS(PI/FLOAT(n + 1)) + COS(PI/FLOAT(m + 1))) ;
   declare
      U : MATRICE(0..n + 1,0..m + 1) ;
   begin
      RESOLUTION_POISSON(f,g1,g2,g3,g4,RHO_JACOBI,h,U) ;
      if LIRE_REPONSE("Voulez vous l'affichage des resultats? ") then
         CLRSCR ;
         AFFICHER_SOLUTION(U) ;
         PAUSE ; CLRSCR ;
      end if ;
      if LIRE_REPONSE("Voulez vous tracer les equipotentielles? ") then
         ENTRER_ENTIER(NB_COURBES,"Nombre de courbes a tracer : ") ;
```

```
         declare
            xMin, xMax, yMin, yMax : FLOAT ;
            U0 : VECTEUR(1..NB_COURBES) ;
         begin
           for i in 1..NB_COURBES
           loop
              PUT("   Courbe numero ") ; PUT(i) ; NEW_LINE ;
              ENTRER_REEL(U0(i),"Entrer u0 : ") ;
           end loop ;
           xMin := 0.0 ; xMax := FLOAT(n + 1)*h ;
           yMin := 0.0 ; yMax := FLOAT(m + 1)*h ;
           TRACE_EQUIPOTENTIELLES(U,xMin,xMax,yMin,yMax,U0) ;
         end ;
       end if ;
   end ;
   CLOSE(IMP) ;
end DEMO_EDP_POISSON ;

procedure DEMO_EDP_POISSON_2 is
f, g1, g2, g3, g4, h1, h2, h3, h4 : FONCTION ;
n, m, p, q, q, r, s, NB_COURBES : INTEGER ;
h, RHO_JACOBI : FLOAT ;
begin
   MODE_AFFICHAGE ;
   PUT_LINE("  Resolution de l'equation de Poisson : ") ;
   PUT_LINE(" Delta(u) = f sur un rectangle troue
             [0,a]x[o,b] - [c,d]x[e,f]") ;
   PUT_LINE("avec les conditions au bord : u(x,y) = g(x,y) sur le bord"
          & " exterieur et u(x,y) = h(x,y) sur le bord interieur.") ;
   NEW_LINE ;
   PUT_LINE("Avec g(x,0) = g1(x), h(x,e) = h1(x), g(x,b) = g2(x), "
          & "h(x,f) = h2(x), g(0,y) = g3(y), h(c,y) = h3(y),
             g(a,y) = g4(y), h(d,y) = h4(y).") ;
   PUT_LINE("On utilise les notations suivantes : ") ;
   PUT_LINE(" a = (n + 1).h , b = (m + 1).h, c = p.h, d = q.h, "
          & "e = r.h, f = s.h (h > 0 pas de discretisation donne) ;") ;
   PUT_LINE("un maillage du rectangle etant defini par :") ;
   PUT_LINE(" (x(i),y(j)) = (i.h,j.h) pour i = 0, ..., n + 1
             et j = 0, ..., m + 1 . ") ;
   PAUSE ; CLRSCR ;
   PUT_LINE("ENTREE DES DONNEES") ;
   ENTRER_ENTIER_BORNE(2,DIM_MAX,n,
                       "Nombre de points sur l'axe des x, n = ") ;
   ENTRER_ENTIER_BORNE(2,DIM_MAX,m,
                       "Nombre de points sur l'axe des y, m = ") ;
   ENTRER_ENTIER_BORNE(2,n - 2,p,"Entrer p (entre 2 et n - 2) : ") ;
   ENTRER_ENTIER_BORNE(p + 1,n - 1,q,
                       "Entrer q (entre p + 1 et n - 1) : ") ;
   ENTRER_ENTIER_BORNE(2,m - 2,r,"Entrer r (entre 2 et m - 2) : ") ;
   ENTRER_ENTIER_BORNE(r + 1,m - 1,s,
                       "Entrer s (entre r + 1 et m - 1) : ") ;
   ENTRER_REEL(h,"h = ") ;
   GET_LINE(f,"f(x,y) = ") ;
```

```
   GET_LINE(g1,"Condition a y = 0 , g1(x) = ") ;
   GET_LINE(h1,"Condition a y = r.h , h1(x) = ") ;
   GET_LINE(h2,"Condition a y = s.h , h2(x) = ") ;
   GET_LINE(g2,"Condition a y = b , g2(x) = ") ;
   GET_LINE(g3,"Condition a x = 0 , g3(y) = ") ;
   GET_LINE(h3,"Condition a x = p.h , h3(y) = ") ;
   GET_LINE(h4,"Condition a x = q.h , h4(y) = ") ;
   GET_LINE(g4,"Condition a x = a , g4(y) = ") ;
   RHO_JACOBI := 0.5*(COS(PI/FLOAT(n + 1)) + COS(Pi/FLOAT(m + 1))) ;
   CLRSCR ;
   declare
      U : MATRICE(0..n + 1,0..m + 1) ;
   begin
      RESOLUTION_POISSON_2(f,g1,g2,g3,g4,h1,h2,h3,h4,p,q,r,s,
                           RHO_JACOBI,h,U) ;
      if LIRE_REPONSE("Voulez vous l'affichage des resultats? ") then
         CLRSCR ;
         AFFICHER_SOLUTION_2(p,q,r,s,U) ;
         PAUSE ; CLRSCR ;
      end if ;
     if LIRE_REPONSE("Voulez vous tracer les equipotentielles? " then
        ENTRER_ENTIER(NB_COURBES,"Nombre de courbes a tracer : ") ;
        declare
           xMin, xMax, yMin, yMax : FLOAT ;
           U0 : VECTEUR(1..NB_COURBES) ;
        begin
           for i in 1..NB_COURBES
           loop
              PUT("   Courbe numero ") ; PUT(i) ; NEW_LINE ;
              ENTRER_REEL(U0(i),"Entrer U0 : ") ;
           end loop ;
           xMin := 0.0 ; xMax := FLOAT(n + 1)*h ;
           yMin := 0.0 ; yMax := FLOAT(m + 1)*h ;
           TRACE_EQUIPOTENTIELLES(U,xMin,xMax,yMin,yMax,U0) ;
        end ;
      end if ;
   end ;
   CLOSE(IMP) ;
end DEMO_EDP_POISSON_2 ;

procedure DEMO_EDP_ELLIPTIQUE is
Alpha, Beta, Gama, Delt, Epsi, f, g1, g2, g3, g4 : FONCTION ;
n, m, NB_COURBES : INTEGER ;
h, Omega : FLOAT ;
begin
   MODE_AFFICHAGE ;
   PUT_LINE("    Resolution d'une equation aux derivees partielles
                 de type elliptique : ") ;
   PUT_LINE("sur un rectangle [0,a]x[o,b] ;") ;
   PUT_LINE("    avec les conditions au bord : u(x,y) = g(x,y) ;") ;
   PUT_LINE("    ou g(x,0) = g1(x), g(x,b) = g2(x),
                    g(0,y) = g3(y), g(a,y) = g4(y) .") ;
   PUT_LINE("On utilise les notations suivantes : ") ;
```

```
   PUT_LINE(" a = (n + 1).h ; b = (m + 1).h ,
             ( h > 0 pas de discretisation donne ) ;") ;
   PUT_LINE("un maillage du rectangle etant defini par : ") ;
   PUT_LINE(" (x(i),y(j)) = (i.h,j.h) pour i = 0, ..., n + 1
             et j = 0, ..., m + 1 . ") ;
   PAUSE ; CLRSCR ;
   PUT_LINE("ENTREE DES DONNEES POUR UNE EQUATION ELLIPTIQUE") ;
   ENTRER_ENTIER_BORNE(2,DIM_MAX,n,
                       "Nombre de points sur l'axe des x : ") ;
   ENTRER_ENTIER_BORNE(2,DIM_MAX,m,
                       "Nombre de points sur l'axe des y : ") ;
   ENTRER_REEL(h,"h = ") ;
   GET_LINE(Alpha,"Alpha(x,y) = ") ;
   GET_LINE(BETA,"Beta(x,y) = ") ;
   GET_LINE(GAMA,"Gamma(x,y) = ") ;
   GET_LINE(DELT,"Delta(x,y) = ") ;
   GET_LINE(EPSI,"Epsilon(x,y) = ") ;
   GET_LINE(f,"f(x,y) = ") ;
   GET_LINE(g1,"Condition a y = 0 , g1(x) = ") ;
   GET_LINE(g2,"Condition a y = b , g2(x) = ") ;
   GET_LINE(g3,"Condition a x = 0 , g3(y) = ") ;
   GET_LINE(g4,"Condition a x = a , g4(y) = ") ;
   ENTRER_REEL(OMEGA,"Coeeficient de relaxation omega = ") ;
   declare
      U : MATRICE(0..n + 1,0..m + 1) ;
   begin
      RESOLUTION_ELLIPTIQUE(f,g1,g2,g3,g4,
                        Alpha,Beta,Gama,Delt,Epsi,Omega,h,U) ;
      if LIRE_REPONSE("Voulez vous l'affichage des resultats? ") then
         CLRSCR ;
         AFFICHER_SOLUTION(U) ;
         PAUSE ; CLRSCR ;
      end if ;
      if LIRE_REPONSE("Voulez vous tracer les equipotentielles? ") then
         ENTRER_ENTIER(NB_COURBES,"Nombre de courbes a tracer : ") ;
         declare
            xMin, xMax, yMin, yMax : FLOAT ;
            U0 : VECTEUR(1..NB_COURBES) ;
         begin
           for i in 1..NB_COURBES
           loop
              PUT("   Courbe numero ") ; PUT(i) ; NEW_LINE ;
              ENTRER_REEL(U0(i),"Entrer u0 : ") ;
           end loop ;
           xMin := 0.0 ; xMax := FLOAT(n + 1)*h ;
           yMin := 0.0 ; yMax := FLOAT(m + 1)*h ;
           TRACE_EQUIPOTENTIELLES(U,xMin,xMax,yMin,yMax,U0) ;
         end ;
       end if ;
   end ;
   CLOSE(IMP) ;
end DEMO_EDP_ELLIPTIQUE ;
```

```
begin
    loop
       CLRSCR ;
  PUT_LINE("-1- Resolution d'une equation de Poisson : Delta(u) = f") ;
  PUT_LINE("-2- Resolution de Delta(u) = f sur un rectangle troue") ;
       PUT_LINE("-3- Resolution d'une e.d.p. de type elliptique") ;
       PUT_LINE("-4- FIN") ;
       NEW_LINE ;
       ENTRER_ENTIER_BORNE(1,4,CHOIX,"Votre choix : ") ;
       case CHOIX is
          when 1 => DEMO_EDP_POISSON ;
          when 2 => DEMO_EDP_POISSON_2 ;
          when 3 => DEMO_EDP_ELLIPTIQUE ;
          when 4 => exit ;
       end case ;
    end loop ;
end DEM_ELLI ;
```

8.6 Spécification du paquetage EDP_PARABOLIQUES

Dans ce paquetage on décrit des procédures de résolution de l'équation de diffusion de la chaleur dans une barre. On pourra le compléter avec des procédures de résolution d'une équation de type parabolique plus générale.

```
with MATH2, COMMON_MATRIX ;
use MATH2, COMMON_MATRIX ;
package EDP_PARABOLIQUES is
NX_MAX : constant := 200 ;
NT_MAX : constant := 30 ;
procedure EDP_CHALEUR_EXPLICITE(k, dx, dt : in FLOAT ;
             f : in FONCTION ; U : in out MATRICE) ; § 5.8.3
procedure EDP_CHALEUR_IMPLICITE(w, k, dx, dt : in FLOAT ;
             f : in FONCTION ; U : in out MATRICE) ; § 5.8.3
procedure AFFICHER_TEMPERATURES(U : in MATRICE ; dt : in FLOAT ;
             p : in INTEGER) ;
procedure COURBES_CHALEUR(U : in MATRICE ; dx : in FLOAT ;
             p : in INTEGER) ;
end EDP_PARABOLIQUES ;
```

8.7 Démonstration du paquetage EDP_PARABOLIQUES

```
with TEXT_IO, CRT, COMMON_MATH0, MATH0, MATH2, COMMON_MATRIX,
     EDP_PARABOLIQUES ;
use TEXT_IO, CRT, COMMON_MATH0, MATH0, MATH2, COMMON_MATRIX,
     EDP_PARABOLIQUES ;

procedure DEM_CHAL is
Eps : constant FLOAT := 1.0E-12 ;
nx, nt, p : INTEGER ;
k, w, L, dx, dt, dtb : FLOAT ;
f : FONCTION ;
CORRECT_F : BOOLEAN ;
```

```
procedure PRESENTATION_CHALEUR is
begin
   PUT_LINE("la barre est a temperature nulle aux bords") ;
   PUT_LINE("le reste n'a pas d'echange de chaleur avec l'exterieur") ;
   PUT_LINE("la longueur de la barre est notee L = (nx + 1)*dx") ;
   PUT_LINE("tf = nt*dt est la valeur finale du temps.") ;
   PUT_LINE("le coefficient de conduction est note k") ;
   PUT_LINE("w est le coefficient de la methode de relaxation") ;
   PUT_LINE("la temperature a t = 0 en x est f(x)") ;
   -- Pour des raisons de compatibilite, f doit etre nulle en 0 et L
end PRESENTATION_CHALEUR ;

begin
   MODE_AFFICHAGE ;
   PRESENTATION_CHALEUR ;
   PAUSE ; CLRSCR ;
   ENTRER_REEL(L," Entrez L : ") ;
   ENTRER_ENTIER_BORNE(2,NX_MAX,nx,
                       " Nombre de points sur l'axe des x : ") ;
   dx := L/FLOAT(nx + 1) ;
   ENTRER_REEL(k," Entrez la valeur de k = ") ;
   loop
      GET_LINE(f," Entrez f(x) : ") ;
      CORRECT_F := (abs(EVALUE(f,0.0)) <= Eps)
                    and (ABS(EVALUE(f,L)) <= Eps) ;
      if not Correct_f then
         BIP ;
         PUT_LINE("* Erreur * f doit etre nulle en 0 et L") ;
      end if ;
      exit when CORRECT_F ;
   end loop ;
   ENTRER_REEL_BORNE(0.0,1.0,w,"Entrer w : ") ;
-- Choix du pas de discretisation en t :
-- Si 0 <= w < 1/2, une condition de stabilite est donnee par :
--               r = dt/(dx^2) < 1/(2*k*(1 - 2*w))
-- La valeur maximale de dt est alors dtb = (dx**2)/(2*k*(1 - w))
   if (w < 0.5) then
      dtb := (dx*dx)/(2.0*k*(1.0 - 2.0*w)) ;
   else
      dtb := 10.0 ;
   end if ;
   ENTRER_REEL_BORNE(0.0,dtb,dt,"Entrez la valeur de dt : ") ;
   ENTRER_ENTIER_BORNE(2,NT_MAX,nt,
                     "Entrez le nombre de points sur l'axe des t : ") ;
   ENTRER_ENTIER_BORNE(1,nt,p,
                       "Entrer la periode des traces de courbes : ") ;
   declare
      U : MATRICE(0..nx + 1,0..nt) ;
   begin
      if (w = 0.0) then
         EDP_CHALEUR_EXPLICITE(k,dx,dt,f,U) ;
      else
```

```
         EDP_CHALEUR_IMPLICITE(w,k,dx,dt,f,U) ;
      end if ;
      if LIRE_REPONSE("Voulez vous l'affichage des resultats? ") then
         AFFICHER_TEMPERATURES(U,dt,p) ;
      end if ;
      if LIRE_REPONSE("Voulez vous le trace des courbes? ") then
         COURBES_CHALEUR(U,dx,p) ;
      end if ;
   end ;
   CLOSE(IMP) ;
end DEM_CHAL ;
```

INDEX

Bibliographie

Chapitre 1

A un premier niveau on pourra consulter :
[1] *Manuel de référence du langage de programmation Ada*. Alsys.
[2] J. BARNES. *Programmer en Ada.* InterEditions — Un ouvrage de base.
[3] B. LEGUY. *Ada : Guide d'utilisation.* Eyrolles — Un excellent ouvrage d'initiation. Très pédagogique.
[4] R. OGOR, R. RANNOU. *Langage Ada et algorithmique.* Hermes — On trouvera de nombreux exemples. C'est aussi une bonne initiation au langage.

En deuxième lecture :
[5] G. BOOCH. *Ingénierie du logiciel avec Ada.* Addison Whesley.
[6] CH. CARREZ. *Des structures aux bases de données.* Dunod.
[7] N. H. COHEN. *Ada as a second language.* Mc Graw Hill.
[8] F. CULWIN. *Ada a developmental approach.* Prentice Hall.
[9] D. J. DAVID. *Le langage Ada.* Editions du PSI.
[10] B. FORD, J. KOK, M. W. ROGERS. *Scientific Ada.* Cambridge University Press.
[11] M. GAUTHIER. *Ada, un apprentissage.* Dunod.
[12] N. GEHANI. *Ada. Introduction avancée.* Eyrolles.
[13] P. LIGNELET. *Structures de données avec Ada. Conception orientée objet.* 2 Vol. Masson.
[14] R. MITCHELL. *Abstract data types and Ada.* Prentice Hall.

Chapitre 2

[1] Logiciel Modulog : Edil Belin : Atelier/Courrier, 20 , Rue Gossin 92 120 Montrouge.
[2] R. DONY. *Graphisme dans le plan et dans l'espace avec Turbo-Pascal 4.0.* Masson.
[3] M. DUCAMP ET A. REVERCHON. *Mathématiques en Turbo-Pascal. Vol. 1 : Analyse.* Eyrolles.
[4] A. LEVINE. *Calcul formel.* Ellipses.

Chapitre 3

[1] J. BARANGER. *Analyse numérique.* Hermann.
[2] R. BURLISCH ET J. STOER. *Introduction to Numerical Analysis.* Springer Verlag.
[3] CEA-EDF-INRIA. *Résolution numérique des grands systèmes linéaires.* Eyrolles.
[4] P.G. CIARLET. *Introduction à l'analyse numérique matricielle et à l'optimisation.* Masson.

[5] P.G CIARLET B. MIARA J.M. THOMAS. *Exercices d'analyse numérique matricielle et d'optimisation.* Masson.
[6] A.S. HOUSEHOLDER. *The theory of matrices in numerical analysis.* Dover.
[7] P. LASCAUX R. THEODOR. *Analyse numérique matricielle appliquée à l'art de l'ingénieur. Vol. 1 et 2.* Masson.
[8] J.P. NOUGIER. *Méthodes de calcul numérique.* Masson.
[9] C. NOWAKOWSKI**.** *Méthodes de calcul numérique. Programmes en Basic et en Pascal. Vol. 1 et 2.* Editions du PSI..
[10] PRESS, FLANNERY, TEUKOLSKY, VETTERLING. *Numerical Recipes*. Cambridge Univ. Press.
[11] C. REINSCH J.H. WILKINSON. *Linear Algebra.* Springer Verlag.
[12] J. VIGNES. *Algorithmes numériques. Analyse et mise en oeuvre.* Vol. 1 et 2. Technip.

Chapitre 4

[1] J. BARANGER. *Analyse numérique.* Hermann.
[2] R. BURLISCH ET J. STOER. *Introduction to Numerical Analysis.* Springer Verlag.
[3] D. MONASSE. *La méthode de Newton et les racines des polynômes.* RMS, Janvier 1987.
[4] J.P. NOUGIER. *Méthodes de calcul numérique.* Masson.
[5] J. M. ORTEGA, W. C. RHEINBOLDT. *Iterative solution of nonlinear equations in several variables.* Academic Press.
[6] PRESS, FLANNERY, TEUKOLSKY, VETTERLING. *Numerical Recipes.* Cambridge Univ. Press.
[7] R. THEODOR. *Initiation à l'analyse numérique.* Masson.

Chapitre 5

[1] AHLBERG, NILSON, WALSH. *The theory of Splines and their applications. Mathematics in science and engineering.* Vol. 38. Academic Press.
[2] R. H. BARTELS, C. BEATY, A. BARSKY. *Mathématiques et CAO.* Vol 6. Hermes.
[3] R. BURLISCH, J. STOER. *Introduction to Numerical Analysis.* Springer Verlag.
[4] CHUEN YEN CHOW. *An introduction to computational fluid mechanics.* Seminole Publishing Company.
[5] PH. DAVIS. *Interpolation and approximation.* Dover Publication.
[6] R. DONY. *Calcul des parties cachées, approximation des courbes par la méthode de Bézier et des B_Splines.* Masson.
[7] ENCYCLOPEADIA UNIVERSALIS. Article : *Représentation et approximation des fonctions.*
[8] S. A. MORSI, A. J. ALEXANDER**.** *An investigation of particule trajectories in two phase flow systems.* Coughborough university of technology.
[9] PICHAT, DI CRESCENZO, WOLF. *Mathématiques pour l'informatique. Exercices et problèmes.* Armand Colin.

[10] PRESS, FLANNERY, TEUKOLSKY, VETTERLING. *Numerical Recipes.* Cambridge.
[11] R. THEODOR. *Initiation à l'analyse numérique.* Masson.

Chapitre 6

[1] AYANT ET BORG. *Fonctions spéciales.* Dunod.
[2] M. CROUZIEX ET A. MIGNOT. *Analyse numérique des équations différentielles.* Masson.
[3] PH. DAVIS. *Interpolation and approximation*. Dover Publication.
[4] P. DEHEUVELS. *L'intégrale.* PUF.
[5] N. N. LEBEDEV. *Special functions and their applications.* Prentice-Hall.
[6] P. PELLETIER. *Techniques numériques appliquées au calcul scientifique.* Masson.
[7] PRESS, FLANNERY, TEUKOLSKY, VETTERLING**.** *Numerical recipes.* Cambridge.
[8] B. RANDE. *Primitives élémentaires.* RMS, Mai 89. Vuibert.
[9] J. STOER ET R. BURLISCH. *Introduction to numerical analysis.* Springer Verlag.
[10] A. WARUSFEL. *A propos de la méthode d'intégration de Romberg.* RMS, Janvier 86. Vuibert.

Chapitre 7

[1] R. BURLISCH ET J. STOER. *Introduction to numerical Analysis.* Springer-Verlag.
[2] CROUZEIX ET MIGNOT. *Analyse numérique des équations différentielles.* Masson.
[3] J. P. DEMAILLY. *Analyse numérique des équations différentielles.* Presses Universitaires de Grenoble.
[4] M. DUCAMP ET A. REVERCHON. *Mathématiques et Turbo-Pascal.* Eyrolles.
[5] G. HACQUES. *Mathématiques pour l'informatique. Vol. 3 : Algorithmique numérique.* Armand Colin.
[6] P. HENRICE. *Discrete variable methods in ordinary differential equations.* Wiley.
[7] P. HENRICE. *Error propagation for difference methods.* Wiley.
[8] A. HILLION. *Les théories mathématiques des populations.* PUF..
[9] J. P. NOUGIER. *Méthodes de calcul numérique.* Masson.
[10] PRESS, FLANNERY, TEUKOLSKY, VETTERLING. *Numerical recipes.* Cambridge.

Chapitre 8

[1] L. C. BARRET, C. R. WYLIE. *Advanced engineering mathematics.* Mac Graw Hill.
[2] R. BURLISCH, J. STOER. *Introduction to numerical Analysis.* Springer Verlag.
[3] H. CARTAN. *Théorie élémentaire des fonctions analytiques d'une ou plusieurs variables complexes*. Masson.
[4] P. G. CIARLET. *Introduction à l'analyse numérique matricielle et à l'optimisation.* Masson.

[5] D. EUVRARD. *Résolution numérique des équations aux dérivées partielles.* Masson.
[6] FLANNERY, PRESS, TEUKOLSKY, VETTERLING. *Numerical Recipes.* Cambridge.
[7] J. P. NOUGIER. *Méthodes de calcul numérique.* Masson.
[8] H. REINHARD. *Equations aux dérivées partielles.* Dunod.