Calcul Scientifique et Symbolique, Logiciels
Licence Mathématiques UE N1MA3003

Alain Yger

Institut de Mathématiques, Université Bordeaux 1, Talence 33405, France

E-mail address: Alain.Yger@math.u-bordeaux1.fr

Version du 9 mai 2014.

Résumé. Ce cours correspond à l'enseignement dispensé en 2013-2014 dans l'UE N1MA3003 ≪ Calcul Scientifique et Symbolique, Logiciels ≫ de la Licence de Mathématiques. Ce cours prend la suite du cours d'≪ Initiation au Calcul Scientifique et Symbolique ≫ (ex MHT304, [**Y1**]). Plusieurs chapitres de l'ouvrage [**MathAp**] (en particulier les chapitres 1,2,3,9,10) ont servi de référence pour la rédaction du cours et peuvent être utilisés pour des approfondissements. Les feuilles de TP (sous l'environnement `Maple` pour ce qui concerne le calcul symbolique, sous l'environnement `MATLAB` en ce qui concerne le calcul scientifique) traitées pendant l'année 2012-2013, ainsi que la liste des projets proposés aux étudiants (travail en binôme ou trinôme, soutenance d'une demi-heure avec illustration machine), sont proposées en annexe de ce polycopié. Les documents accompagnant les TP (routines, corrigés des TP mis en ligne après remise des travaux) sont disponibles sur les sites :

`http://www.math.u-bordeaux1.fr/~yger/initiationMAPLE`
`http://www.math.u-bordeaux1.fr/~yger/initiationMATLAB`

On trouvera aussi sur ces pages des liens vers des textes aidant à la prise en main de ces logiciels. Ce cours vise à illustrer (en s'appuyant souvant dessus) les acquis de L1, notamment le cours de MISMI (*cf.* [**Y0**] pour le polycopié en ligne) et d'Analyse 1 (*cf.* [**Yan**] pour le polycopié en ligne) ; ces cours sont bien sûr à consulter pour tout support théorique, de même que les cours d'Algèbre 1 (semestre 1) et d'Algèbre 2 (semestre 3), auquels ce cours renvoie fréquemment.

Table des matières

CHAPITRE 1

Calcul symbolique, Calcul numérique, erreurs

1.1. Les deux objectifs du cours ; deux logiciels, deux missions

Ce cours a un double objectif :
- d'une part, une familiarisation avec la prise en main de deux logiciels, l'un dévolu au calcul symbolique (`Maple 13`), l'autre au calcul scientifique (en l'occurrence `MATLAB 7`), et à la programmation sous ces environnements ;
- d'autre part, une première initiation aux bases du calcul *symbolique* (ou encore *formel*) et au calcul *scientifique*.

Le logiciel `Maple 13` est un logiciel de calcul *symbolique* (ou calcul formel). Il propose des outils de calcul symbolique et fournit en parallèle de puissants outils de graphisme et de calcul. Cependant, il n'est pas conçu pour le calcul scientifique. L'ossature de son noyau s'appuie pour une bonne part sur la théorie algébrique de l'élimination (voir plus loin dans le cours), ainsi que sur des outils tels que la dérivation et l'intégration formelle des fonctions appartenant à ce que l'on appelle communément la *classe de Liouville* : fractions rationnelles, logarithmes et fonctions afférentes telles que arctan, exponentielle, fonctions trigonométriques ou hyperboliques, solutions d'équations différentielles de nature algébrique, c'est-à-dire à coefficients polynomiaux. Ce logiciel (payant) `Maple` est, avec `Mathematica`, l'un des deux logiciels de calcul formel les plus utilisés dans les milieux universitaires. D'autres logiciels (libres) le complètent pour des tâches plus spécifiques : `PARI` en Théorie des Nombres, `Macaulay` ou `Cocoa`, `Reduce`, pour l'algèbre polynomiale (utile en robotique par exemple) ou la géométrie algébrique [1]. Il s'agit d'un logiciel conçu sur la base d'un langage *interprété*, et non *compilé*, ce qui explique fait que les instructions (`do`, `while`,...) correspondant à des boucles de calcul soient exécutées lentement. On tire profit à utiliser au maximum les fonctions toutes intégrées. La programmation sous cet environnement n'est pas toujours chose aisée.

Aux antipodes de `Maple` et du calcul symbolique (donc essentiellement « sans pertes ») le logiciel `MATLAB 7` qui, lui aussi , utilise un langage de programmation ressemblant plutôt à un langage de *script* qu'à un langage compilé tel que le `C` ou `FORTRAN 90`, est un logiciel de calcul scientifique (donc cette fois de calcul « avec pertes »). L'ossature de son noyau repose cette fois sur le calcul matriciel [2]. Si ce type de langage *script* exécute les instructions bien plus lentement qu'un langage compilé (même remarque qu'à propos de `Maple`, mis à part que les choses s'avèrent plus cruciales ici car les tableaux en jeu dans les calculs peuvent être

1. Il convient aussi ici de mentionner le logiciel libre `Sage` (téléchargeable sur le site `http://www.sagemath.org/fr`) offrant toutes les potentialités du calcul symbolique implémenté sous `Maple`, ce sous le langage informatique `Python`.

2. L'acronyme `MATLAB` vient d'ailleurs de là : `MATLAB` pour `Matrix Laboratory`.

de taille énorme), ce type de langage dit « interprété » se justifie par un temps de conception et de « debogage » excessivement réduits par rapport à ceux que nécessite un langage compilé. Dans le milieu industriel, où il est communément répandu, `MATLAB` (souvent couplé avec `SIMULINK`, ces deux logiciels étant gérés par la société `Mathworks`), ce logiciel professionnel sert essentiellement à concevoir des maquettes de programmes et à les tester, ce avant une phase ultérieure de programmation, cette fois sous un logiciel compilé tel `C++` ou `FORTRAN 90`. Comme `Maple`, `MATLAB` offre une bibliothèque très importante de fonctions et de documentation : la version disponible sur le site du CREMI [3] est une version accompagnée des `toolboxes` suivants : `Image Processing`, `Signal Processing`, `Statistics`, `Partial Differential Equations`, `Wavelets`. Le noyau du logiciel suffira aux besoins de ce cours à ce niveau de L2.

Les deux logiciels `Maple` et `MATLAB` sont des logiciels professionnels et payants. Il faut signaler cependant que le logiciel libre `Scilab` réalise un clone de `MATLAB` conçu et géré par le Consortium `Scilab` (INRIA, ENPC). Il est couramment utilisé, mais essentiellement dans le milieu universitaire [4]. À l'heure actuelle, seul `MATLAB` est réellement connu et exploité dans le milieu industriel. Outre le logiciel libre `Sage` mentionné plus haut, d'autres logiciels libres de calcul symbolique (inspirés de `Maple`) ont été développés en vue d'utilisation plus spécifique dans le domaine de l'enseignement ; on peut par exemple citer `Xcas`, téléchargeable librement sur

`http://www-fourier.ujf-grenoble.fr/~parisse/giac_fr.html`

La programmation élémentaire (réalisation de programmes à partir de maquettes réalisées sous un environnement tel qu'`Algobox`, rencontré peut-être dans le secondaire) est plus aisée sous un environnement tel que `MATLAB` (elle consiste en la rédaction de routines `.m`) ou `Scilab` (réalisation de fichiers `.sce`) qu'elle ne l'est sous `Maple` (environnement sous lequel peut par contre tirer avantageusement profit des bibliothèques et commandes pré-installées [5]). Cependant, comme on l'a vu, les objectifs ne sont pas les mêmes : faire du calcul numérique (ou scientifique) d'un côté (`MATLAB` ou `Scilab`), faire du calcul formel en maniant des expressions symboliques (polynômes, fonctions, ...) de l'autre (`Maple`).

Si le second objectif puise ses sources dans les acquis de L1 (en particulier en arithmétique, en analyse, en algèbre des polynômes ou des fractions rationnelles, ainsi qu'en algèbre linéaire) et dans les bases mathématiques que vous allez engrangées tout au long du semestre (cours d'Algèbre 1, d'Algèbre 2, d'Analyse 1, d'Analyse 2), il arrivera aussi que ce cours anticipe certaines notions intervenant

3. Licence pour 30 utilisateurs en simultané.

4. On peut télécharger le logiciel libre `Scilab` depuis le site `http://www.scilab.org`. À signaler également qu'il existe un site où l'on peut télécharger une version conçue pour l'apprentissage de `Scilab` dans les lycées : `http://www.scilab.org/fr/community/education/maths`

5. La prise en main de `Maple` consiste à apprendre dans un premier temps à le manier comme une grosse « calculette ». Sur le site `http://www.math.u-bordeaux1.fr/~yger/initiationMaple`, sont disponibles pour ce cours des fichiers de « prise en main » interactifs pour une première initiation avec les commandes du logiciel (en particulier le fichier `premierspasMaple.mw`, à ouvrir sous l'environnement `Maple`, fichier sur lequel je me suis appuyé pour les illustrations du cours). Faire `Enregistrer la cible du lien sous ...` avant d'ouvrir `Maple` sous un répertoire où vous aurez enregistré ce fichier. En ce qui concerne la prise en main de `MATLAB` ou `Scilab`, on trouvera sous `http://www.math.u-bordeaux1.fr/~yger/initMS.pdf` une aide à la prise en main de ces logiciels. Une familiarité avec le calcul matriciel est cependant requise.

ultérieurement dans le cursus de Licence : résolution numérique des équations ou systèmes différentiels, orthogonalité et conséquences, mise en route d'algorithmes basés sur les méthodes de « point fixe » (les algorithmes de recherche sur la toile tels `pagerank` sur lequel se fonde le principe de `Google` en sont un bon exemple), ou bien encore certains aspects du calcul matriciel, en particulier du point de vue spectral, avec l'approfondissement de la théorie de la décomposition des matrices.

Pour résumer cette introduction, disons que ce cours d'initiation au calcul scientifique et symbolique sera constamment illustré *via* l'utilisation d'un logiciel de calcul. Le mieux adapté au calcul *symbolique* (axé sur la manipulation des expressions formelles et par voie de conséquence le calcul « sans pertes » relevant plutôt de l'arithmétique) sera `Maple`. Le mieux adapté au calcul *scientifique* (axé cette fois sur le calcul dans le champ des réels au service de la modélisation en mathématiques appliquées) sera `MATLAB`. Ces deux logiciels seront alternativement utilisés tout au long de ce cours dans les scéances de TP depuis leur prise en main jusqu'à l'illustration des acquis de cours et à une initiation à la programmation sous leur environnement.

Terminons cette introduction par un exemple tout bête soulignant la différence fondamentale entre calcul formel et calcul scientifique. Pour se convaincre que `Maple` est un logiciel de calcul symbolique, on fait (comme on l'a fait en cours) le test d'évaluer $\sqrt{x}$ par exemple en $x = 346$. Le logiciel retourne :

$$\sqrt{346}$$

Mais l'on a pris la précaution de déclarer 346 comme un nombre décimal (et non plus comme un entier traité de fait comme un symbole), par exemple si l'on demande l'évaluation de $\sqrt{x}$ en $x = 346.0$, `Maple` répond cette fois :

```
18.60107524
```

On peut ici s'entrainer avec d'autres fonctions `Maple` (par exemple l'évaluation `eval`) en utilisant l'aide interactive disponible dans `initiationMaple` mentionnée plus haut. Le logiciel `Maple`, conçu pour faire du calcul symbolique, c'est-à-dire de la manipulation d'expresions mathématiques formelles, aura ici été détourné de sa fonction pour traiter ce calcul comme un calcul scientifique (la valeur fournie ici pour $\sqrt{346}$ n'étant bien sûr qu'une valeur tronquée à certaines décimales après la virgule).

Si l'on demande par contre à `MATLAB` de retourner le nombre `pi` en spécifiant `format long`, il retournera ce nombre avec ses quinze premières décimales. Le nombre π est stocké ainsi dans la machine et tout calcul numérique l'impliquant sera automatiquement un « calcul avec pertes ». On reviendra dans la section suivante sur la représentation des réels en *virgule flottante* et le codage machine des nombres ou des symboles.

1.2. Représentation des nombres en machine, types d'erreur

La rédaction de cette section (plus particulièrement de la sous-section 1.2.3) s'appuie sur la présentation enrichie de nombreux exemples faite par P. Zimmermann dans [**Zim**].

1.2.1. Un premier exemple troublant. Commençons par soumettre la machine à un test très simple, celui consistant à effectuer le calcul itératif du nombre

$$x = (10^n + 1.5) - 10^n\,,\ n = 0, 1, 2, ...$$

Le code de calcul que nous soumettons à la machine (ici sous MATLAB) est donc :

```
function test1
temp=1;
for i=1:30
    temp=10*temp;
    x=temp+1.5;
    y=temp;
    z=x-y;
    [i z]
end
```

Notons ici encore la présence d'une boucle de calculs :

```
for i=1:30
    ...
    ...
    ...
end
```

et la réinitialisation de la variable temp à chaque étape.

Voici la réponse obtenue au niveau de l'affichage des sorties (le numéro i de l'itération, puis la valeur correspondante z du résultat) :

```
1.0000    1.5000
2.0000    1.5000
3.0000    1.5000
4.0000    1.5000
5.0000    1.5000
6.0000    1.5000
7.0000    1.5000
8.0000    1.5000
9.0000    1.5000
10.0000    1.5000
11.0000    1.5000
12.0000    1.5000
13.0000    1.5000
14.0000    1.5000
15.0000    1.5000
16     2
17     0
18     0
19     0
20     0
21     0
22     0
23     0
24     0
```

```
25      0
26      0
27      0
28      0
29      0
30      0
```

Le constat est clair : au delà de la 16-ème itération, la machine ne rend plus le résultat escompté (qui bien sûr devrait être 1.5000). En fait, ce qui se passe ici est que la taille des nombres impliqués a nécessité (au niveau de leur codage) un espace de travail dont le volume dépasse le seuil fixé par la machine : il faut savoir en effet que pour coder un nombre réel en double précision (ce que fait `MATLAB` par défaut), la machine dispose de 64 bits (seulement 32 bits pour le codage en *simple précision*). Nous verrons au paragraphe suivant comment ces 64 bits sont « organisés » pour le codage des réels en *virgule flottante*. Mais d'ores et déjà, nous nous rendons compte que la défaillance du calcul ici est manifestement liée à des erreurs d'arrondi, la capacité de discernement de la machine ne s'avérant plus suffisante lorsque les entrées sont des expressions faisant intervenir trop de *digits*, comme des entiers positifs de trop grande taille (comme c'est le cas ici lorsque l'exposant de 10 se met à dépasser 16). Nous reviendrons sur cette question dans la sous-section 1.2.3.

1.2.2. Un second exemple. Le second exemple proposé, outre qu'il permet une première prise en main de `MATLAB`, met en évidence un autre type de faille du calcul scientifique. À la fois la machine et les mathématiques y sont cette fois pour quelque chose (nous le verrons beeucoup plus loin dans ce cours lorsque nous dégagerons la notion de *conditionnement*).

Rappelons tout d'abord quelques bases concernant la résolution des systèmes linéaires de N équations à N inconnues

$$M \cdot X = B\,,$$

où M est une matrice $N \times N$ à coefficients réels et B un vecteur colonne à entrées réelles ; si le rang de la matrice M est égal à N (ce qui signifie $\det M \neq 0$), alors l'application

$$X \in \mathbb{R}^N \longmapsto M \cdot X$$

est bijective et l'unique vecteur colonne

$$X = \begin{pmatrix} x_1 \\ \vdots \\ x_N \end{pmatrix}$$

de l'équation $M \cdot X = B$ est donné par

$$X = M^{-1} \cdot B\,,$$

où

$$M^{-1} = \frac{1}{\det M}[\text{cofacteurs}\,(M)]^t \tag{1.1}$$

(A^t désignant la transposée d'une matrice A).

On verra au chapitre suivant pourquoi la méthode (algorithmique) du pivot s'avère plus judicieuse pour résoudre un tel système linéaire que la résolution *via* le calcul de

la matrice M^{-1} (que nous utiliserons ici). Nous allons dans cette section justement utiliser `MATLAB` qui, comme son nom l'indique, un logiciel s'articulant essentiellement autour du *calcul matriciel*. On part d'une matrice M très simple que nous déclarons sous `MATLAB` (ce calcul est une première initiation à cet environnement, initiation que vous allez approfondir en TP) :

```
>> M= [10  7  8  7 ; 7 5 6 5 ; 8 6 10 9 ; 7 5 9 10] ;
M =

    10     7     8     7
     7     5     6     5
     8     6    10     9
     7     5     9    10
```

Le calcul du déterminant de M montre que $\det M = 1$. On déclare un vecteur colonne B par

```
>> B = [32 ; 23 ; 33 ; 31];

B =

    32
    23
    33
    31
```

La résolution immédiate du système donne :

```
>> M^(-1)*B

ans =

    1.0000
    1.0000
    1.0000
    1.0000
```

L'opération $*$ introduite ici correspond à la multiplication usuelle entre deux matrices (ou tableaux) rectangulaires M et N « multipliaples », c'est-à-dire telles que le nombre de lignes de M (matrice de gauche) soit égal au nombre de colonnes de N (matrice de droite) [6].

Si ce problème numérique correspond à un calcul numérique sollicité par le résultat d'une expérience, il est vraisemblable que les entrées de l'appareil (représenté par l'action de la matrice M) sont connues non pas exactement, mais avec une marge d'erreur. Il en est de même pour les coordonnées de la « sortie » B. Perturbons donc légèrement les entrées de M et tentons à nouveau de résoudre le système. Les nouvelles matrices M et B perturbées sont les suivantes :

6. En revanche, l'opération `M.*N` (lorsque M et N sont deux matrices rectangulaires de mêmes dimensions) réalise le « produit au sens naif » (cette fois commutatif, alors que le produit $*$ ne l'était pas) `P=M.*N` des deux matrices M et N entrée par entrée ($p_{ij} = m_{ij} \times n_{ij}$ pour chaque paire d'indices i, j indexant les deux tableaux en jeu). Le point-virgule suivant une instruction `MATLAB` signifie que l'on ne demande pas l'affichage du résultat. Attention ! sous `Maple`, ce point-virgule en fin de ligne n'a pas la même fonction : il est nécessaire pour clôre une ligne de commande.

```
>> Mperturb=[10 7 8.05 7.05 ; 7.04 5.02 6 5 ;8 5.99 9.98 9 ; 7 5 9 10]

Mperturb =

   10.0000    7.0000    8.0500    7.0500
    7.0400    5.0200    6.0000    5.0000
    8.0000    5.9900    9.9800    9.0000
    7.0000    5.0000    9.0000   10.0000
```

On peut aussi envisager de perturber M en introduisant une perturbation par aléatoire obtenue commme un échantillon de loi uniforme par

```
>> perturb = (1/100)*(rand (4,4) - ones (4,4)/2);
>> Mperturb = M + perturb ;
```

On peut ainsi varier les tests numériques à l'infini. Le résultat du calcul de $X = \widetilde{M}^{-1} \cdot B$ avec ces données (légèrement) perturbées est troublant :

```
>> Mperturb^(-1)*B

ans =

    1.1693
    0.6593
    1.1329
    0.9322
```

On voit que l'on est très loin de la solution

$$X = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}$$

obtenue lorsque l'entrée M n'est pas perturbée !

On peut envisager de perturber aussi B, par exemple en

```
>> Bperturb=[32.01 ; 22.99 ; 33.01 ; 30.997]

Bperturb =

   32.0100
   22.9900
   33.0100
   30.9970
```

Le résultat du calcul de $\widetilde{M}^{-1} \cdot \widetilde{B}$ est encore pire (on y voit apparaitre une entrée négative) !

```
>> Mperturb^(-1)*Bperturb

ans =

    1.9353
   -0.6112
```

```
    1.4540
    0.7420
```

Ce n'est pas ici la machine qui est en jeu, mais les mathématiques elles mêmes ! On peut deviner le problème en réalisant que

```
>> M^(-1)

ans =

   25.0000  -41.0000   10.0000   -6.0000
  -41.0000   68.0000  -17.0000   10.0000
   10.0000  -17.0000    5.0000   -3.0000
   -6.0000   10.0000   -3.0000    2.0000
```

L'inverse de M a des coefficients de taille significative, ce qui est une des raisons pour cette instabilité. La notion sous-jacente est en fait celle de *conditionnement* d'une matrice, mais l'on y viendra plus tard (la notion de valeur propre jouant un rôle important). On comprend en tout cas l'intérêt de développer des méthodes algorithmiques, telles le *pivot* de Gauss que vous avez rencontré en Algèbre 1 ou les méthodes itératives basées sur le théorème du point fixe (Jacobi, Gauss-Seidel) dont nous parlerons dans ce cours (en les implémentant sous `Maple` ou `MATLAB`), plutôt que d'attaquer la résolution en inversant la matrice M. Notons d'ailleurs au passage que l'expression développée d'un déterminant d'ordre N implique $N!$ termes, ce qui fait du problème du calcul d'un gros déterminant un problème très vite d'une excessive complexité algorithmique ! Le calcul de M^{-1} directement suivant (1.1) est donc à proscrire du point de vue de la complexité et du risque de mauvais conditionnement de la matrice M.

1.2.3. Le codage en virgule flottante. Étant donné un entier strictement positif β (dit « base »), tout nombre entier naturel se décompose « en base β » de manière unique sous la forme

$$n = \sum_{j=0}^{N} b_j \beta^j, \quad b_j \in \{0, ..., \beta - 1\}.$$

Si le développement en base β le plus familier est celui qui correspond à $\beta = 10$ (développement *décimal*), le plus en phase avec le calcul machine est le développement en base $\beta = 2$ (développement « binaire ») qui n'utilise que deux symboles :

- 0= l'interrupteur est fermé, *i.e* le courant ne passe pas ;
- 1= l'interrupteur est ouvert, *i.e* le courant passe.

Par exemple, en base 2,

$$19 = 1 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 1 \times 2 + 1 = [1\ 0\ 0\ 1\ 1].$$

L'encodage d'un réel en machine, dit *en virgule flottante*, se fait suivant le standard `IEEE754` (1985, révisé en 2008), soit dans l'un des trois formats binaires `binary32`, `binary64`, `binary128`, soit dans l'un des deux formats décimaux que sont `decimal64`, `decimal128`.

Dans un des systèmes binaires (on prendra comme exemple `binary64`, dit *système en double précision* binaire), un nombre réel x (ou plutôt une valeur approchée $\overline{x}$ de ce nombre réel) est encodé sur $1+11+52=64$ bits[7] :

- le premier bit est réservé pour le *signe* du nombre[8] ; on note $s \in \{0,1\}$ la valeur de ce bit ;
- les 11 bits suivants servent à encoder l'*exposant*, *i.e* l'entier $e \in \mathbb{Z}$ défini (lorsque x est non nul) par le fait que $2^{-e}|x| \in [1/2, 1[$, ce qui signifie que le nombre $2^{-e}|x|$ admet un unique développement binaire propre (c'est-à-dire dont les coefficients ne sont pas tous égaux à 1)

$$2^{-e-1}|x| = \frac{b_0}{2} + \frac{b_1}{2^2} + \cdots + \frac{b_k}{2^{k+1}} + \cdots,\ b_0 = 1,\ b_j \in \{0,1\}\ \forall\, j \in \mathbb{N}^* \ ; \tag{1.2}$$

  la valeur codée avec ces 11 bits est donc un entier $0 \leq v \leq 2^{11}-1 = 2047$ et l'exposant $e = v - 1023$ peut prendre toute valeur entière entre -1023 ($v=0$) et 1024 ($v = 2047$) ;
- les 52 bits restants sont utilisés pour coder le mot $[b_1 \cdots b_{52}]$, mot de 52 lettres, chacune valant 0 ou 1 ; le mot $[1\, b_1 \cdots b_{52}]$ est appelé *mantisse*[9] de x.

Les nombres -0 et $+0$ sont donc encodés respectivement avec $v=0$ (tous les bits $b_1, ..., b_{52}$ étant mis à 0) tandis que $\pm\infty$ et `NaN` (« Not A Number ») sont encodés avec $v = 2047$ (respectivement lorsque tous les bits $b_1, ..., b_{52}$ sont mis à zéro ou non).

Exemple 1.1. Par exemple, pour π :

$$\pi \simeq \frac{7074237752028440}{2^{51}} = \frac{2^{52}+2570638124657944}{2^{51}} = 2\Big(1+\frac{N}{2^{52}}\Big)$$

le numérateur $N = 2570638124657944 \leq 2^{51}$ s'exprimant ici comme un mot de 52 caractères (0 ou 1) en base 2. On a donc $s=0$, $e=1$, c'est-à-dire $v=1024$, le mot $[b_1 \cdots b_{52}]$ correspondant à l'écriture en base 2 du nombre

$$N = 2570638124657944 \leq 2^{51}.$$

L'encodage de π sera ainsi (si l'on respecte l'ordre des bits qui a été indiqué) l'écriture en base 2 du nombre

$$\begin{aligned} & s \times 2^{63} + v \times 2^{52} + N = 4614256656552045848 \\ = \ & [0100000000001001001000011111101101010100010001000010110100011000] \end{aligned}$$

Si N_b est le nombre de bits impliqués dans le codage de la mantisse ($N_b = 52$ en double précision, $N_b = 23$ en simple précision, $N_b = 112$ en quadruple précision), l'erreur d'arrondi entre une vraie mantisse b et la mantisse « codée » $\overline{b}$ est donc majorée en module par

$$|b - \overline{b}| \leq 2^{-N_b-1}$$

---

7. En *simple précision*, soit dans `binary32`, le découpage est $1+8+23=32$ bits, tandis qu'en *quadruple précision*, soit dans `binary128`, le découpage est $1+15+112=128$ bits.

8. On note à ce propos qu'une ambiguité existe pour $x=0$ et que la machine nous force à distinguer -0 et $+0$.

9. Le premier bit matérialisé en $b_0 = 1$ dans (1.2) est dit *bit caché* ; mettre ce bit à 0 conduit à la représentation des nombres *dénormalisés* ou *sous-normaux* (*subnormal*), bien utiles dans les opérations pour éviter par exemple une division par zéro conduisant brutalement à $\pm\infty$.

(on « arrondit »au nombre codable avec $N_b + 1$ chiffres le plus proche, qu'il soit plus petit ou plus grand que m, exactement comme on le fait en décimal). L'erreur relative commise lorsque l'on arrondit $x = \pm b \times 2^{e+1}$ en $\overline{x} = \pm\overline{b} \times 2^{e+1} = \pm[1\, b_1 \cdots b_{N_b}] \times 2^{e+1}$ est donc majorée par

$$\frac{|x - \overline{x}|}{|x|} = \frac{|b - \overline{b}|}{b} \leq \frac{|b - \overline{b}|}{\frac{1}{2}} \leq 2 \times 2^{-N_b - 1} = 2^{-N_b}\,.$$

Cette erreur relative 2^{-N_b} (où N_b est le nombre de bits utilisés pour coder la mantisse), est appelée *erreur machine*; c'est elle qui sera responsable des *erreurs d'arrondi* dans les calculs (voir la section suivante).

1.2.4. Arrondis dans les calculs entre flottants. On fixe ici une précision (simple, double ou quadruple), donc un entier N_b (23, 52 ou 115 suivant que l'on est en simple, double ou quadruple) et un entier M_v ($2^8 - 1$, $2^{11} - 1$, $2^{15} - 1$ suivant que l'on est en simple, double ou quadruple précision). Si y est un nombre flottant, *i.e* un réel du type $y = (-1)^s[1\, b_1\, \ldots b_{N_b}]\, 2^e$ avec $s \in \{0, 1\}$ et e tel que $0 \leq v \leq M_v$, on pose

$$\mathrm{ulp}(y) := 2^{e - N_b}.$$

Cette notation (anglosaxonne) vaut pour « *Unit in the Last Place* ».

Définition 1.1 (les cinq modes d'arrondi correct du standard `IEEE 754`). On dit que y est *arrondi au plus proche* du nombre réel x

- avec arrondi pair (`roundTiesToEven`) si y est un nombre flottant tel que
$$|y - x| \leq \frac{\mathrm{ulp}(y)}{2}$$
avec la convention que si x est exactement la valeur médiane entre deux flottants y_1 et y_2, alors on privilégie celui dont la mantisse est paire;
- avec arrondi *away* (`roundTiesToAway`) si y est un nombre flottant tel que
$$|y - x| \leq \frac{\mathrm{ulp}(y)}{2}$$
avec la convention que si x est exactement la valeur médiane entre deux flottants y_1 et y_2, alors on privilégie y_2 si $x > 0$, y_1 si $x < 0$.

On dit que y est un *arrondi dirigé* de x

- vers 0 (`roundTowardZero`) si $|y - x| \leq \mathrm{ulp}(y)$ et $|y| \leq |x|$;
- vers $-\infty$ (`roundTowardNegative`) si $|y - x| \leq \mathrm{ulp}(y)$ et $y \leq x$;
- vers $+\infty$ (`roundTowardPositive`) si $|y - x| \leq \mathrm{ulp}(y)$ et $y \geq x$.

Les opérations arithmétiques classiques que l'on introduit entre nombres flottants sont l'addition, la soustraction, la multiplication et la division, que l'on peut implémenter en flottant comme :

$$\begin{aligned}
(a, b) \mapsto a \oplus b &:= \text{arrondi au plus près de } a + b \\
(a, b) \mapsto a \ominus b &:= \text{arrondi au plus près de } a - b \\
(a, b) \mapsto a \otimes b &:= \text{arrondi au plus près de } a \times b \\
(a, b) \mapsto a \oslash b &:= \text{arrondi au plus près de } a/b.
\end{aligned}$$

À la place du choix qui est fait ici, on peut choisir l'un des cinq modes d'arrondi proposés dans la Définition 1.1.

Aux quatre opérations algébriques mentionnées, il faut ajouter la prise de racine carrée et les conversions binaire/décimal.

Le standard IEEE 754 impose que toutes ces opérations algébriques soient conduites (comme indiqué) avec l'un des cinq modes d'arrondi proposés dans la Définition 1.1. On dit alors que les six opérations mentionnées ont des implémentations *correctement arrondies*. Cette exigence est dite aussi d'*arrondi correct*. Elle est également recommandée dans l'usage des fonctions transcendantes log, exp, sin, cos, tan, atan, acos, asin, $\sqrt{x^2+y^2}$, x^n, $x^{1/n}$, $\sin(\pi(\cdot))$, $\cos(\pi(\cdot))$, atan/π, sinh, cosh, asinh, acosh, tanh, atanh.

La situation est autrement plus délicate lorsqu'il s'agit d'implémenter au niveau des flottants une fonction

$$f\ : (x_1, ..., x_n) \in \mathbb{R}^n \longrightarrow \mathbb{R}$$

et que l'on cherche à arrondir « correctement » $y = f(x_1, ..., x_n)$ pour $(x_1, ..., x_n)$ des flottants donnés. Une bonne approximation $\widehat{y} = y(1+\epsilon)$ de la fonction f permet certes de trouver une grille de flottants avoisinant y, mais ne permet en général pas de décider quel flottant dans cette grille réalise un des cinq arrondis corrects (au sens de la norme IEEE 754) de y. Un test fait sur l'approximation $\widehat{y}$ ne saurait en effet induire de conclusion relativement à ce qui aurait du se passer si le test avait été conduit, comme il aurait du l'être, avec y en place de $\widehat{y}$.

Signalons les deux lemmes suivants (faciles à vérifier et souvent utiles comme « garde-fous », par exemple pour valider la preuve d'un postulat mathématique à l'aide d'une machine[10]).

LEMME 1.1 (lemme de Sterbenz, 1974). *Si x et y sont deux flottants tels que l'on ait $x/2 < y < 2x$, alors $x \ominus y = x - y$.*

LEMME 1.2 (erreur d'arrondi sur l'addition). *Si x et y sont des flottants, l'erreur d'arrondi $(x+y) - (x \oplus y)$ est un flottant et l'on a*

$$(x+y) - (x \oplus y) = y \ominus \big((x \oplus y) \ominus x\big). \tag{1.3}$$

Il en est de même pour $(x \times y) - (x \otimes y)$, mais la formule remplaçant (1.3) *dans ce cas est plus complexe.*

1.3. Les boucles logiques (if/else, for, while) sur quelques exemples

Cette section visant à rappeler quelques rudiments de programmation pourra aussi être mise à profit pour une famirialisation avec la rédaction de routines .m sous MATLAB. On y retrouvera sous une forme algorithmique cette fois le bagage arithmétique introduit en MISMI.

1.3.1. La recherche du PGCD de deux entiers a, b, $b > 0$. Rappelons ici la définition du plus grand diviseur commun (PGCD) de deux entiers positifs ou nuls non tous les deux nuls ainsi que celle du petit multiple commun (PPCM) de deux entiers a et b strictement positifs.

10. La preuve en 1998 par T. Hales de la conjecture de Kepler (1611) stipulant que densité de l'empilement cubique à faces centrées ($\pi/\sqrt{18} \simeq .74$) maximise la densité d'un empilement de sphères égales en est un exemple instructif.

DÉFINITION 1.2. Si a et b sont deux entiers positifs non tous les deux nuls, on appelle *plus grand diviseur commun* (en abrégé PGCD) de a et b le plus grand de tous les nombres entiers positifs divisant à la fois a et b. Si $b = 0$, le PGCD de a et b vaut donc a. On dit que a et b sont *premiers entre eux* si leur PGCD est égal à 1. Le PPCM de deux entiers strictement positifs a et b est le plus petit entier strictement positif multiple à la fois de a et de b et l'on a la relation

$$\mathrm{PPCM}(a,b) \times \mathrm{PGCD}(a,b) = a \times b\,,\ \forall\, a,b \in \mathbb{N}^*\,.$$

Le calcul du PGCD de deux nombres entiers positifs a et b avec b non nul fait apparaitre une démarche mathématique constructive (donc implémentable sur une machine) que l'on appelle un *algorithme*. Ce terme vient du surnom Al-Khwarizmi du mathématicien ouzbek Al Khuwarismi, 780-850, (dont le début du titre d'un des ouvrages fournit d'ailleurs aussi le mot *algèbre*).

Notons (comme sous la syntaxe du logiciel de calcul `MATLAB` que nous utilisons ici)

```
[q,r] = div(a,b)
```

l'instruction qui calcule, étant donnés deux nombres entiers tels que $b \neq 0$, l'unique couple (q,r) avec $r \in \{0,...,b-1\}$ tel que $a = bq+r$, donné par la division euclidienne de a par b. On pourrait tout aussi bien utiliser les commandes de `Maple`. La fonction `div` s'exprime ainsi :

```
function [q,r] = div (x,y) ;
q = floor (x./y) ;
r = x- q*y ;
```

Considérons la suite d'instructions qui conduit au calcul du PGCD de deux entiers a et b ; le nombre à calculer est noté PGCD dans la suite d'instructions ci-dessous :

```
function PGCD=PGCD(a,b);

x=a ;
y=b ;
while y>0
    [q,r] = div(x,y);
    if r==0
        PGCD = y;
        y = 0 ;
    else
        [q1,r1] = div(y,r);
        x = r;
        PGCD = x ;
        y=r1 ;
    end
end
```

Si l'on traduit ceci en français, voici la lecture :

```
fonction PGCD=PGCD(a,b);

x=a;
y=b;
tant que y est non nul, faire
   [q,r] = div(x,y);
```

```
   si r=0
     PGCD = y;
     y=0;
   sinon
     [q1,r1]= div(y,r);
     x=r;
     PGCD = x;
     y=r1;
   fin
fin
```

Si maintenant, on lit ceci en langage « mathématique » et de manière exhaustive (toutes les instructions sont listées et l'on n'effectue pas de ré-assignement des variables), la démarche que traduit cette routine est :

$$\begin{aligned} a &= bq + r \\ b &= rq_1 + r_1 \\ r &= r_1 q_2 + r_2 \\ \vdots &= \vdots \\ r_{N-2} &= q_N r_{N-1} + r_N \\ r_{N-1} &= r_N q_{N+1} + 0 \end{aligned}$$

(comme les restes successifs $r, r_1, ...$ décroissent strictement et qu'il y a un nombre fini d'entiers entre 0 et b, il vient forcément un moment où r_N divise r_{N-1}, r_N étant le dernier reste obtenu non nul). Le PGCD de a et b est aussi celui de b et r, de r et r_1, et ainsi, en cascade, de r_N et 0 ; il vaut donc r_N.

CONCLUSION : *Le PGCD de a et b est donc égal au dernier reste non nul r_N dans ce très célèbre algorithme de division dit algorithme d'Euclide.*

Par exemple, pour $a = 13$ et $b = 4$,

$$\begin{aligned} 13 &= 4 \times 3 + 1 \\ 4 &= 4 \times 1 + 0 \end{aligned}$$

le dernier reste non nul étant ici $r_0 = 1$. On a donc $\mathrm{PGCD}\,(a, b) = 1$.

La routine MATLAB que l'on a mis en place pour implémenter la fonction PGCD inclut une boucle de calcul

```
while y > 0
  ...
  ...
  ...
end
```

Tant que la variable y garde une valeur strictement positive, une certaine opération est répétée en boucle. Cette variable y, ré-initialisée après chaque retour de boucle, matérialise le dernier reste atteint dans la division euclidienne (tant que se reste reste non nul). A l'intérieur même de cette boucle, figure un nœud de décision :

```
if  r = 0
    ...
    ...
```

```
    ...
else
    ...
    ...
    ...
end
```

La première alternative conduit à l'arrêt du processus précisément lors de la boucle en cours (puisque on assigne à y la valeur 0 sous cette alternative) tandis que la seconde appelle la boucle suivante avec des variables x et y qui entre temps ont été ré-initialisées.

Avec cet exemple de routine `PGCD`, on a un exemple concret tant de la commande « `while` » que de l'alternative « `if/else` » .

1.3.2. La résolution de l'identité de Bézout. Si a est un entier relatif, on pose $|a| = a$ si $a \in \mathbb{N}$, $|a| = a'$ si $a = -a'$ avec $a' \in \mathbb{N}$.

Si a et b sont deux entiers relatifs non tous les deux nuls, on appelle *plus grand diviseur commun* de a et b (ou encore $\mathrm{PGCD}(a, b)$) le PGCD des entiers positifs $|a|$ et $|b|$. Les nombres a et b sont premiers entre eux si ce PGCD vaut 1. On attribue au mathématicien français Etienne Bézout (1730-1783) le résultat suivant, dont l'intérêt pratique tant en mathématiques pures qu'appliquées est aujourd'hui devenu capital (il conditionne ce que l'on appelle le *lemme des restes chinois*[11]).

THEORÈME 1.1. *Soient a et b deux nombres entiers relatifs non tous les deux nuls et d leur PGCD ; il existe au moins un couple $(u_0, v_0) \in \mathbb{Z}^2$ tel que*

$$au_0 + bv_0 = d$$

(une telle relation est appelée identité de Bézout lorsque $d = 1$).

Preuve. La construction de u_0 et v_0 est algorithmique et se fait en remontant depuis l'avant-dernière ligne les calculs faits dans l'algorithme de division euclidienne de $|a|$ par $|b|$ (on suppose ici $b \neq 0$).

$$\begin{aligned}
|a| &= |b|q_0 + r_0 \\
|b| &= r_0 q_1 + r_1 \\
r_0 &= r_1 q_2 + r_2 \\
\vdots &= \vdots \\
r_{N-3} &= q_{N-1} r_{N-2} + r_{N-1} \\
r_{N-2} &= q_N r_{N-1} + d \\
r_{N-1} &= d q_{N+1} + 0
\end{aligned}$$

11. En voici un exemple d'application parlant : c'est grâce à ce type de résultat que l'on peut combiner deux prises de vue à des temps d'exposition premiers entre eux (disons par exemple a et b secondes) d'un mobile se déplacant à vitesse constante pour réaliser un cliché instantané net de ce mobile en mouvement (imaginez deux photos d'une avenue la nuit avec les trainées lumineuses des phares de voitures, lorsque l'on veut précisément voir se fixer sur la pellicule le flux de véhicules).

On écrit

$$\begin{aligned} d &= r_{N-2} - q_N r_{N-1} \\ &= r_{N-2} - q_N(r_{N-3} - q_{N-1} r_{N-2}) \\ &= -q_N r_{N-3} + (1 + q_N q_{N-1}) r_{N-2} \\ \vdots &= \vdots \\ &= u_0 a + v_0 b \end{aligned}$$

Plus que l'énoncé du théorème 2.1 lui même, ce qui est très important (parce que très utile) est qu'il s'agisse d'une assertion dont la démonstration est constructive, c'est-à-dire s'articule sur un algorithme, dit *algorithme d'Euclide étendu*. Voici, en quelques lignes de code, la fonction qui calcule, étant donnés deux entiers relatifs a et b avec $b \neq 0$ à la fois le PGCD d de a et b, mais aussi une paire d'entiers relatifs (u, v) telle que $d = au + bv$ (il s'agit, on le remarquera, d'un algorithme inductif qui s'auto-appelle) :

```
fonction [PGCD,u,v]=bezout(a,b);
x=a ;
y= abs(b) ;
[q,r]=div(x,y);
if r==0
    PGCD = y;
    u=0 ;
    v=1;
else
 [d,u1,v1]=bezout(y,r);
 PGCD=d;
 u=v1;
 v=sign(b)*(u1- q*v1);
end
```

On retrouve ici une alternative

```
if
  ...
  ...
else
  ...
  ...
end
```

mais cette fois non plus une boucle

```
while  ...
   ...
   ...
   ...
end
```

mais (dans un des volets de l'alternative) un ré-appel au programme (ici appelé bezout) comme sous-programme de lui-même. Il faut prendre garde ici que la présence précisément de l'alternative interdit que la procédure ne boucle sur elle même. Cette

alternative devient à une certaine étape un *test d'arrêt* . La notion dégagée ici est celle de *récursivité* en programmation.

1.4. La force et les limites du calcul symbolique

On poursuit dans cette sous-section la familiarisation avec l'outil `Maple`. Exploitant certains résultats mathématiques (parfois en aval de votre cursus), on mettra en évidence ce que peut apporter le calcul symbolique par rapport au calcul formel (le calcul des 100 premières décimales de π en est un exemple, voir la sous-section 1.4.1), en même temps que l'on mettra en lumière les failles justifiant le recours inéluctable aux outils de calcul scientifique (avec les risques liés aux « pertes » qui en découlent). La résolution sous `Maple` des équations algébriques jusqu'au degré 4 seulement (sous-section 1.4.2) constituera une transition avec les méthodes numériques (Newton, sécante, fausse position, dichotomie) qui feront l'objet du chapitre à venir.

1.4.1. La division euclidienne au service de la formule de John Machin et du calcul des décimales de π. Le nombre π (demi-périmètre du cercle unité) joue un rôle central en mathématiques. Ce nombre ne saurait être racine d'une équation algébrique à coefficients entiers et, par là même, échappe à la classe des nombres modélisables « sans pertes », par exemple par le biais du calcul symbolique (n'impliquant que des entiers naturels et des signes). Nous avons vu dans l'exemple 1.1 comment encoder π dans un système binaire. Sous un logiciel en double précision tel `MATLAB`, le nombre π ne peut être encodé qu'au travers d'une approximation décimale ou binaire. Par exemple, si l'on demande à `MATLAB` de renvoyer la valeur de ce nombre (en `format long`, c'est-à-dire avec affichage du nombre maximal de décimales possibles, ici 15), on obtient :

```
>> format long
>> pi

ans =

   3.141592653589793
```

Ce que nous venons de remarquer dans la section précédente à propos des limites de la représentation des nombres en virgule flottante lorsqu'il s'agit de calcul scientifique (simple ou double précision pour le codage des nombres réels) ne vaut plus lorsqu'il s'agit de représenter sous forme décimale un nombre rationnel du type N/D où N et D sont deux entiers strictement positifs sous un logiciel de calcul symbolique tel que `MATLAB` ou `Mathematica`. En effet, ce que peut faire le logiciel dans ce cas, c'est effectuer l'algorithme de division euclidienne de N par D (celui que vous connaissez depuis l'école primaire), ce autant de fois qu'on le veut, et afficher donc un nombre arbitrairement grand de décimales du nombre N/D. On obtient ici une valeur numérique de N/D avec une précision arbitraire (bien sûr, la seule limite est la capacité de stockage mémoire de la machine).

Illustrons cela avec une formule d'analyse (que vous prouverez ultérieurement dans le cours d'Analyse 3 au semestre 4) permettant d'obtenir le nombre π comme limite

commune des deux suites adjacentes

$$u_n = 4\sum_{k=0}^{2n-1} \frac{(-1)^k}{2k+1}\,, \quad n \geq 1$$

(cette suite croit) et

$$v_n = 4\sum_{k=0}^{2n} \frac{(-1)^k}{2k+1}\,, \quad n \geq 1$$

(cette suite décroit), avec $\lim_{n\to+\infty}(v_n - u_n) = 0$; ces deux suites sont des suites de nombres rationnels. La formule que l'on admet pour justifier ceci est

$$\begin{aligned} \pi = 4\,\mathrm{atan}(1) = 4\int_0^1 \frac{dt}{1+t^2} = 4\int_0^1 \Big(\sum_{k=0}^{\infty}(-t^2)^k\Big)\,dt = 4\sum_{k=0}^{\infty}\frac{(-1)^k}{2k+1} \\ = 4 + 4\sum_{k=1}^{\infty}\frac{(-1)^k}{2k+1}. \end{aligned} \tag{1.4}$$

Sous `Maple13`, on peut par exemple calculer en utilisant la commande `sum` les nombres u_{100} et v_{100} :

$$u = u_{100} = 4 + 4\sum_{k=1}^{199}\frac{(-1)^k}{2k+1} \qquad v = v_{100} = 4 + 4\sum_{k=1}^{200}\frac{(-1)^k}{2k+1}. \tag{1.5}$$

```
> u:=4:
for i from 1 to 199 do
  u:=u + 4*(-1)^(i)/(2*i+1);
end do:
 > u;
731070388038968202720510198283781775454721070454247644121830326235077353957 42\

  648084627718263852275691373137426993671519326356091089553028236126486890826\

  620494680531149718 4/233077884665326398150824103062920077329319208418023678 3\

  989076833198484591101069034917139195411721198713640327975684786306003115020\

  030669668404746294722675526611061681111 25

> v:=4:
for i from 1 to 200 do
  v:=v + 4*(-1)^(i)/(2*i+1);
end do:

> v;
294091537142287554883527885924048172266660426085825400006449591553545412773 36\

  844643332400591621231400189173721343489670702108917127711991109422911095010\

  381839431335458281528 4/934642317507958856584804653282309510090570025756274 9\
```

```
503796198101125923210315286830017728173601002006841697715182495993087072491\

230322985370303032641837928861710357341256112 5
```

Les résultats sont ici explicités sous forme de fraction puisque les expressions (1.5) donnant $u = u_{100}$ et $v = v_{100}$ sont des expressions rationnelles et que `Maple` se comporte comme un logiciel de calcul symbolique. C'est l'algorithme de division euclidienne que `Maple` met en jeu lorsque l'on invoque la commande `evalf` en imposant le nombre de décimales (ici par exemple 50). On obtient ici :

```
> evalf(u,50);
              3.1365926848388167504149697050776129667152913517315

> evalf(v,50);
              3.1465677471829564012877876601898324180868624240507
```

Le nombre π est ici encadré par ces deux valeurs. Malheureusement, ce résultat est très mauvais : il nous permet simplement d'affirmer que la première décimale de π vaut 1. La raison pour laquelle nous ne pouvons avoir de résultat meilleur ici tient au fait que la convergence de la suite

$$\Big(4\sum_{k=0}^{n}\frac{(-1)^k}{2k+1}\Big)_{n\geq 0}$$

vers sa limite, donc des deux suites adjacentes $(u_n)_{n\geq 0}$ et $(v_n)_{n\geq 0}$ vers π, est excessivement lente. L'erreur entre la somme de tous les termes d'une série alternée (c'est de cela qu'il s'agit ici car on ajoute, avec les $(-1)^k/(2k+1)$, des termes dont le signe alterne avec l'indice k) et la somme de ses n premiers termes est en effet seulement majorée en module par le premier terme négligé au cran n, c'est-à-dire le $(n+1)$-ième. Ce qui donne ici

$$\Big|\pi - 4\sum_{k=0}^{n}(-1)^k\frac{1}{2k+1}\Big| \leq \frac{4}{2(n+1)+1} = \frac{4}{2n+3} \qquad \forall\, n\in\mathbb{N}.$$

On peut aisément montrer qu'en fait ici cette erreur est équivalente à $2/n$ lorsque n tend vers l'infini. Il s'agit donc là d'une convergence très lente et on est bien loin de la convergence exponentielle ! Il faudrait donc travailler avec 2000 termes pour espérer par exemple une erreur en 10^{-3} ! On est donc loin du compte avec seulement 199 ou 200 termes comme ce que l'on prend ici.

Pourtant une formule algébrique aussi simple que

$$(5+i)^4 = 2(239+i)(1+i) \tag{1.6}$$

(à vérifier) nous assure que

$$\operatorname{atan}(1) = \frac{\pi}{4} = 4\operatorname{atan}(1/5) - \operatorname{atan}(1/239).$$

Comme $1/5 < 1$ et $1/239 < 1$, la formule [12]

$$\begin{aligned}\pi &= 16\sum_{k=0}^{\infty}\frac{(-1)^k}{2k+1}\,(1/5)^{2k+1} - 4\sum_{k=0}^{\infty}\frac{(-1)^k}{2k+1}(1/239)^{2k+1}\\ &= 4(4/5-1/239) + 4\sum_{k=1}^{\infty}\frac{(-1)^k}{2k+1}\Big(4(1/5)^{2k+1} - (1/239)^{2k+1}\Big)\end{aligned} \tag{1.7}$$

fournit deux suites adjacentes $(f_n)_n$ et $(g_n)_n$ définies respectivement par

$$\begin{aligned}f_n &:= 4(4/5-1/239) + \sum_{k=1}^{2n-1}\frac{(-1)^k}{2k+1}\Big(4(1/5)^{2k+1} - (1/239)^{2k+1}\Big)\\ g_n &:= 4(4/5-1/239) + \sum_{k=1}^{2n}\frac{(-1)^k}{2k+1}\Big(4(1/5)^{2k+1} - (1/239)^{2k+1}\Big)\end{aligned} \tag{1.8}$$

la première croissante, la seconde décroissante, telles que $f_n \le \pi \le g_n$ et que l'erreur $g_n - f_n$ soit cette fois exponentiellement petite comme fonction de n lorsque n tend vers l'infini. On obtient cette fois

```
> f:=4*(4/5-1/239):
for i from 1 to 199 do
  f:=f + 4*((-1)^(i)/(2*i+1))*(4*(1/5)^(2*i+1)- (1/239)^(2*i+1));
end do:

> evalf(f,500);
3.14159265358979323846264338327950288419716939937510582097494459230781640628\

  62089986280348253421170679821480865132823066470938446095505822317253594081\

  28111745028410270193852110555964462294895493038196442881097566593344612847\

  56482337867831652712019091456485669234603486104543266482114118363373510285\

  35076359081339898934439794046156228875441930324313039229345995246595044357\

  37206792623722068844007898313193728922903729918543724114310106870448116924\

  28854999047785085414874766831399145283928764215324170\
```

```
> g:=4*(4/5-1/239):
for i from 1 to 200 do
  g:=g + 4*((-1)^(i)/(2*i+1))*(4*(1/5)^(2*i+1)- (1/239)^(2*i+1));
end do:

> evalf(g,500);
3.14159265358979323846264338327950288419716939937510582097494459230781640628\
```

12. Cette formule (et d'autres similaires), basée ici sur l'identité algébrique (1.6), est très ancienne. On la doit au mathématicien anglais John Machin (1680-1751), à qui l'on doit le calcul des cent premières décimales de π. Évidemment aujourd'hui, les outils informatiques, ne serait-ce que `Maple` comme ici, permettent d'aller beaucoup plus loin.

```
20899862803482534211706798214808651328230664709384460955058223172535940812 8\

48111745028410270193852110555964462294895493038196442881097566593344612847 5\

64823378678316527120190914564856692346034861045432664821347248461911614262 1\

58482325849940965709691587584538479590529938688911713220232737198528503829 3\

87367700901374338092486313273529421656847125526267755207862481420685502842 1\

65864878259164928873191868094079515070935248127 75
```

Le fait que f_{100} et g_{100} aient ici leurs 277 premières décimales égales justifie, puisque π se trouve entre ces deux nombres, que ces 277 premières décimales soient en fait les 277 premières décimales de π. En travaillant avec f_n et g_n avec $n >> 100$, on déterminerait les décimales de π bien au delà de ce seuil de 277. Toute la force du calcul symbolique, mettant en jeu ici l'algorithme d'Euclide pour traiter la division de deux nombres entiers afin de déterminer le développement décimal de fractions telles f_{100} ou g_{100}, a été exploité ici pour obtenir π avec une telle précision. Ceci n'a rien a voir avec l'encodage de π dans un logiciel de calcul scientifique tel `MATLAB` car nous avons mis en œuvre ici (avec la division euclidienne des entiers) une démarche algorithmique. Il convient de noter tout de fois que la démarche consistant à remplacer l'approximation (1.4) (trop lente) par l'approximation (1.7) est, elle, une démarche relevant du calcul scientifique : c'est ce que l'on appelle l'*accélération de convergence*. On la retrouvera dans ce cours avec l'interpolation, le calcul approché d'intégrales et la démarche proposée par le mathématicien néo-zélandais Alexander Craig Aitken, 1895-1967, dont on reparlera plus loin.

1.4.2. Les commandes `solve`, `fsolve` sous `Maple` ; au delà du degré 5 ? Les équations algébriques à coefficients complexes sont résolubles par radicaux tant qu'elles sont de degré inférieur où égal à 4. On connait bien les cas des degrés 1 et 2. Le cas du degré 3 consiste à ramener la résolution de l'équation à celle de

$$X^3 + PX + Q = 0$$

(ce sont les formules de Cardan [13]). La résolution par radicaux des équations de degré 4 a été proposée par le mathématicien italien Ludovico Ferrari (1522-1565) ; elle consiste à se ramener au cas des équations du type

$$x^4 + px^2 + qx + r = 0$$

et à ramener la résolution de cette équation à celle (proposée par Cardan pour le degré 3) de l'équation de degré 3 cette fois :

$$X^3 + 2pX^2 + (p^2 - 4r)X - q^2.$$

La commande `solve` sous un logiciel de calcul symbolique tel que `Maple` reprend, lorsque le degré de l'équation à résoudre est inférieur ou égal à 4, donc que la résolution par radicaux est possible, la démarche explicite proposée par Cardan

13. Le mathématicien italien de la Renaissance Gerolamo Cardano (plus connu sous le nom de Cardan, 1501-1576), est aussi connu, avec son traité de la théorie des jeux, comme l'un des précurseurs de la théorie de probabilités.

ou Ferrari. Par exemple, on trouve, sous `Maple13`, pour un exemple d'équation de degré 3 tel que $x^3 + x^2 + 1 = 0$:

```
> solve(x^3+x^2+1 = 0);
```

```
                      (1/3)
  1 /           (1/2)\                    2              1  1
- - \116 + 12 93     /      - ------------------------- - -, --
  6                                              (1/3)   3  12
                                /           (1/2)\
                              3 \116 + 12 93     /
```

```
                   (1/3)
  /           (1/2)\                    1              1
  \116 + 12 93     /      + ------------------------- - -
                                                (1/3)   3
                              /           (1/2)\
                            3 \116 + 12 93     /
```

```
                /                      (1/3)                          \
    1    (1/2) |  1 /           (1/2)\                    2            |  1
  + - I 3      |- - \116 + 12 93     /      + ------------------------ , --
    2          |  6                                                (1/3)|  12
               |                                 /           (1/2)\     |
               \                               3 \116 + 12 93     /     /
```

```
                   (1/3)
  /           (1/2)\                    1              1
  \116 + 12 93     /      + ------------------------- - -
                                                (1/3)   3
                              /           (1/2)\
                            3 \116 + 12 93     /
```

```
                /                      (1/3)                          \
    1    (1/2) |  1 /           (1/2)\                    2            |
  - - I 3      |- - \116 + 12 93     /      + ------------------------|
    2          |  6                                                (1/3)|
               |                                 /           (1/2)\     |
               \                               3 \116 + 12 93     /     /
```

La racine réelle et les deux racines complexes conjuguées sont ici séparées par des virgules. Pour une équation de degré 4, telle que $x^4 + 3x^2 + 5x + 1 = 0$, `Maple` propose la résolution par radicaux de Ricatti :

```
> solve(x^4+3*x^2+5*x+1 = 0);
```

```
                                (1/3)
      1 /             (1/2)\                    22                1  1
-1, - - \244 + 12 1005     /      + ------------------------- + -, --
      6                                                (1/3)   3  12
                                     /             (1/2)\
                                   3 \244 + 12 1005     /
```

```
                      (1/3)
  /             (1/2)\                    11                1
  \244 + 12 1005     /      - ------------------------- + -
                                                  (1/3)   3
                               /             (1/2)\
                             3 \244 + 12 1005     /
```

```
                /                         (1/3)                              \
    1    (1/2) |  1 /             (1/2)\                   22                |
  + - I 3      |- - \244 + 12 1005     /      - -------------------------|,
    2          |  6                                                  (1/3)|
               |                                  /             (1/2)\     |
               \                                3 \244 + 12 1005     /     /
```

```
                       (1/3)
  1  /             (1/2)\                    11                1
  -- \244 + 12 1005     /      - ------------------------- + -
  12                                              (1/3)   3
                                  /             (1/2)\
                                3 \244 + 12 1005     /
```

```
                /                         (1/3)                              \
    1    (1/2) |  1 /             (1/2)\                   22                |
  - - I 3      |- - \244 + 12 1005     /      - -------------------------|
    2          |  6                                                  (1/3)|
               |                                  /             (1/2)\     |
               \                                3 \244 + 12 1005     /     /
```

Cependant, même avec des équations de degré 4, il peut arriver que le logiciel soit incapable d'effectuer cette résolution par radicaux, pourtant en principe possible, comme sur l'exemple de l'équation x^4+3x^3+5x+1 où la non-réponse est éloquente :

```
> solve(x^4+3*x^3+5*x+1 = 0);
```

```
      /  4       3                      \
RootOf\_Z  + 3 _Z  + 5 _Z + 1, index = 1/,

       /  4       3                      \
RootOf\_Z  + 3 _Z  + 5 _Z + 1, index = 2/,

       /  4       3                      \
RootOf\_Z  + 3 _Z  + 5 _Z + 1, index = 3/,

       /  4       3                      \
RootOf\_Z  + 3 _Z  + 5 _Z + 1, index = 4/
```

La résolution par radicaux devient de toutes façon impossible au delà du degré (du fait du résultat de Galois assurant la non simplicité du groupe A_n pour $n \geq 5$, comme vous le verrez plus tard) et ceci se reflète dans la réponse de `Maple`, logiciel de calcul symbolique, donc formel, à la commande `solve`. Ainsi sur l'exemple de l'équation $x^5 + x^3/2 + 1 = 0$:

```
> solve(x^5+(1/2)*x^3+1 = 0, x);
       /    5     3              \        /    5     3              \
 RootOf\2 _Z  + _Z  + 2, index = 1/, RootOf\2 _Z  + _Z  + 2, index = 2/,

         /    5     3              \        /    5     3              \
   RootOf\2 _Z  + _Z  + 2, index = 3/, RootOf\2 _Z  + _Z  + 2, index = 4/,


         /    5     3              \
   RootOf\2 _Z  + _Z  + 2, index = 5/
```

En revanche, la commande `fsolve` fournit pour l'exemple de $x^5 + x^3/2 + 1 = 0$ la solution réelle (unique) de cette équation de manière approchée, soit :

```
> fsolve(x^5+(1/2)*x^3+1 = 0, x);
```

```
                          -0.9098248906
```

Les méthodes conduisant à ces résolutions numériques relèvent, elles, du calcul scientifique (donc en général « avec pertes », à moins que les démarches d'approximation en jeu ne fassent intervenir que des fractions rationnelles). Ces méthodes s'appuient sur le théorème du point fixe et nous en détaillerons certaines au chapitre suivant. On a, pour les mettre en œuvre, plutôt intérêt à utiliser un logiciel de calcul scientifique (tel `MATLAB`), certainement plus adapté qu'un logiciel de calcul symbolique auquel on a forcé la main (avec une commande telle que `fsolve` au lieu de `solve`) pour faire du calcul numérique.

CHAPITRE 2

La résolution numérique des équations non linéaires

Comme nous l'avons vu dans au chapitre 1, l'approche numérique est la seule envisageable pour envisager de résoudre des équations algébriques de degré supérieur ou égal à 5. Ceci est *a fortiori* encore plus vrai lorsqu'il s'agit d'équations du type $\{f(x) = 0\}$, où f est une fonction non algébrique (en général tabulée dans le noyau du logiciel interprété), avec tous les problèmes d'évaluation et d'arrondi que cela peut poser (cf. la sous-section 1.2.4 au chapitre 1). L'objectif de ce chapitre est de présenter un certain nombre de méthodes itératives conduisant au calcul approché des racines d'une équation $\{f(x) = 0\}$ (*Newton*, « *fausse position* », *sécante*, *dichotomie*), de dégager ensuite les outils mathématiques impliqués dans leur validation (formule de Taylor, pour l'essentiel), enfin de dégager la notion d'*ordre*.

2.1. Présentation des méthodes

2.1.1. Présentation sur des exemples de la méthode de Newton sur $\mathbb{R}$. Considérons la fonction

$$x \longmapsto f(x) = x^5 + x^3/2 + 1$$

sur $[-1, -1/2]$. Nous savons qu'elle est croissante (strictement), que sa valeur en -1 vaut $-1/2$ et que sa valeur en $-1/2$ vaut $-1/32 - 1/16 + 1 > 0$. Cette fonction doit donc (d'après le théorème des valeurs intermédiaires) s'annuler en un unique point de $[-1, -1/2]$.

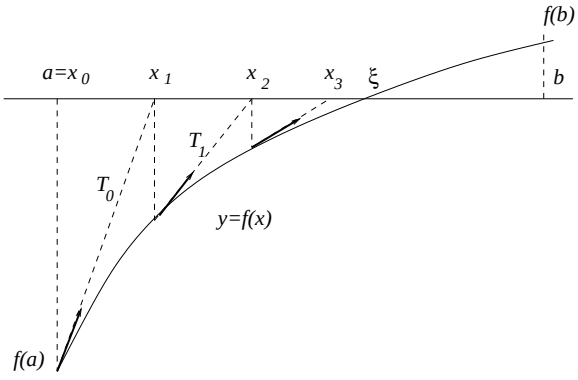

FIGURE 2.1. Méthode de Newton

Lançons l'algorithme itératif qui consiste, partant de $x_0 = a = -1$, à calculer de manière itérative les x_k suivant la règle

$$x_{k+1} = x_k - \frac{x_k^5 + x_k^3/2 + 1}{5x_k^4 + 3x_k^2/2} = x_k - \frac{f(x_k)}{f'(x_k)}\,.$$

En fait x_{k+1} est l'abcisse du point où la tangente en $(x_k, f(x_k))$ au graphe de f rencontre l'axe des x (voir la figure 2.1). Voici ici la routine itérative sous `MATLAB` :

```
function x=Newton(init,N);
x=init;
for i=1:N
    x = x - (x^5+x^3/2+1)/(5*x^4+3*x^2/2);
end
```

Si l'on prend comme valeur initiale `init=-1`, on constate la convergence de la suite $(x_k)_k$:

```
>> format long
>> Newton(-1,3)
  -0.909825093948150
>> Newton(-1,5)
  -0.909824890637916
>> Newton(-1,10)
  -0.909824890637916
```

On vérifie que l'on récupère bien (ici en partant de $x_0 = -1$, et au bout de 5 itérations), les 14 premières décimales de l'unique racine réelle ξ de l'équation $f(x) = 0$ (en double précision, `MATLAB` ne pouvant pas faire mieux) appartenant à $]-1, -1/2[$. Le résultat est concordant avec celui que nous donnait la commande `fsolve` sous `Maple` (voir la sous-section 1.4.2). C'est ce qu'il nous reste maintenant à clarifier et à comprendre.

Supposons que f soit une fonction de classe C^2 sur un intervalle ouvert I de $\mathbb{R}$, à valeurs réelles, et que a et b soient deux points de I tels que :

- $f(a)f(b) < 0$;
- $f' \neq 0$ sur $[a, b]$.

La seconde condition implique que f' reste soit strictement positive, soit strictement négative sur $[a, b]$. Nous avons ici représenté la figure 2.1 en supposant, comme dans l'exemple, $f' > 0$ sur $[a, b]$.

Pour faciliter la compréhension de la figure 2.1 sur laquelle nous nous appuyons, nous supposerons de plus (mais, on le verra, cette hypothèse n'a rien d'essentiel) que f'' ne s'annule pas sur $[a, b]$, c'est-à-dire que le graphe de f ne présente aucun changement de sens de concavité sur $[a, b]$: soit ce graphe « regarde toujours vers le bas » ($f'' < 0$ sur $[a, b]$, *i.e.* f *concave*, ce que nous supposerons, puisque c'est ici le cas de notre exemple), soit il regarde toujours vers le haut ($f'' > 0$ sur $[a, b]$, *i.e.* f *convexe*).

Comme $f(a)f(b) < 0$ et que $f' > 0$ sur $[a, b]$, la fonction f est strictement croissante sur $[a, b]$ et, d'après le théorème des valeurs intermédiaires [1], s'annule en un unique point $\xi \in]a, b[$. Notre but ici est de calculer ξ *via* une méthode itérative.

1. Revoir le cours de MIS 101 (pour l'énoncé, on pourra par exemple se reporter à [**Y0**], section 3.2) et celui d'Analyse 1 [**Yan**] (pour la preuve et une étude plus poussée).

On part de $x_0 = a$ et l'on construit la tangente T_0 au point $(x_0, f(x_0))$, tangente dont l'équation est

$$y - f(x_0) = f'(x_0)(x - x_0)\,.$$

Cette tangente coupe l'axe des abscisses au point $(x_1, 0)$ tel que

$$y(x_1) = 0 = f(x_0) + f'(x_0)(x_1 - x_0)\,,$$

soit

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)} \geq x_0\,.$$

Il est clair que nous n'avions pas ici le choix entre a et b comme point initial (`init`) ; en effet, si l'on partait de b, la tangente issue de b intersecterait l'axe réel en un point x_1 qui sortirait du champ sur lequel est définie la fonction f, à savoir le segment $[a, b]$. Du fait de la propriété de concavité ($f'' < 0$ sur $[a, b]$), la tangente T_0 au point $(x_0, f(x_0))$ reste au dessus du graphe de f et le nombre x_1 (voir la figure 2.1) se trouve entre x_0 et l'abscisse inconnue ξ. On peut donc recommencer (car $x_1 \in [a, b]$). On construit donc la tangente T_1 au point $(x_1, f(x_1))$; cette tangente coupe l'axe des abscisses au point $(x_2, 0)$ avec

$$x_2 = x_1 - \frac{f(x_1)}{f'(x_1)} \in [x_1, \xi]\,.$$

Le procédé itératif consistant, partant de $x_0 = a$, à calculer de proche en proche les x_k, $k \geq 0$, *via* la relation de récurrence à un pas

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}\,,\ k \in \mathbb{N}\,, \tag{2.1}$$

génère une suite

$$x_0 \leq x_1 \leq x_2 \leq \cdots \leq \xi\,.$$

Cette suite est une suite croissante majorée, donc convergente (vers un point x_∞ de $[x_0, \xi] \subset [a, b]$). Comme

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}\,,\ k \in \mathbb{N}\,,$$

on a, en passant à la limite lorsque k tend vers $+\infty$ et en utilisant le fait que f et f' sont continues (avec $f' \neq 0$ sur $[x_0, \xi] \subset [a, b]$) :

$$x_\infty = x_\infty - \frac{f(x_\infty)}{f'(x_\infty)}\,,$$

d'où $f(x_\infty) = 0$, ce qui implique nécessairement $x_\infty = \xi$. La suite $(x_k)_{k\geq 0}$ approche donc le point ξ (ici en croissant).

L'un des défauts (au niveau de l'implémentation) concernant la méthode de Newton est le fait qu'elle nécessite la connaissance non seulement des valeurs de f, mais de celles de f'. Si la fonction f est tabulée, il faut effectuer un calcul numérique de dérivée discrète (avec les erreurs d'arrondi que cela suppose) avant d'exprimer x_{k+1} en fonction de x_k. Outre (on le verra dans la sous-section 2.2.2) son efficacité en ce qui concerne la rapidité de convergence (à condition qu'elle converge, ce qui, on le verra, n'est pas toujours le cas), la méthode itérative proposée ici présente l'avantage d'être une *méthode à un pas* : la valeur de x_{k+1} « écrase » celle de x_k car x_{k+1} est fonction seulement de x_k. Il n'est nul besoin de conserver des valeurs calculées en mémoire.

Comme autre exemple d'application, prenons une fonction tabulée sous `MATLAB` dont la dérivée est aussi tabulée ($f(x) = \cos x$, $f'(x) = -\sin(x)$) et cherchons la valeur approchée de π en remarquant que la fonction cos est strictement décroissante, strictement concave sur $]0, 2]$ et que $\cos(0) = 1$, $\cos(2) < 0$. On peut calculer le zéro $\xi = \pi/2$ de cos sur $[0, 2]$ en partant de `init = 2` et en utilisant la méthode de Newton. On constate qu'au terme de 5 itérations, on obtient les 14 premières décimales de π.

Il faut noter que, si l'algorithme itératif ainsi décrit converge sous les hypothèses de stricte monotonie et de stricte concavité (ou convexité) pour la fonction f, sous lesquelles nous nous sommes placés [2], il n'en est pas de même en général, même si f' garde un signe fixe sur $[a, b]$; des changements de convavité du graphe peuvent nous faire sortir du cadre $[a, b]$. Même si d'ailleurs la fonction f est de classe C^2 et strictement monotone (avec $f' > 0$ ou $f' < 0$) sur $\mathbb{R}$ tout entier, de tels changements de concavité peuvent aisément induire la non convergence de l'algorithme lorsqu'il est initié en un point arbitraire de $\mathbb{R}$; l'exemple de la fonction

$$x \in [-1, 1] \mapsto \frac{5x - x^3}{4},$$

pour lequel la méthode, initiée à -1 ou 1, ne fait que « rebondir » entre ces deux valeurs, en est un exemple (notons qu'il y a ici changement de concavité en $x = 0$). Voir la figure 2.2.

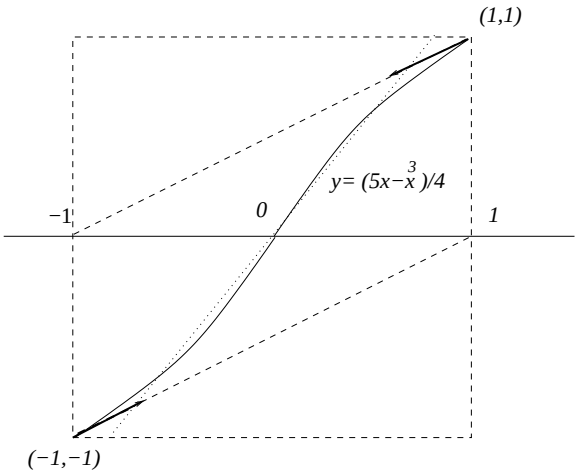

FIGURE 2.2. La méthode de Newton prise en défaut

Cette démarche algorithmique pour résoudre numériquement l'équation $f = 0$, est une démarche attribuée à Isaac Newton ; c'est l'*algorithme de Newton* [3]. Nous verrons dans la sous-section 2.2.2) que le théorème du point fixe est impliqué dans

2. Même si, comme nous le verrons dans la section 2.2.2, ces hypothèses de stricte concavité ou convexité peuvent être allégées.

3. Philosophe, mathématicien, physicien, esprit « universel », l'anglais Isaac Newton (1642-1727) fut, avec le mathématicien prussien Leibniz, l'un des pères incontestés du calcul différentiel moderne.

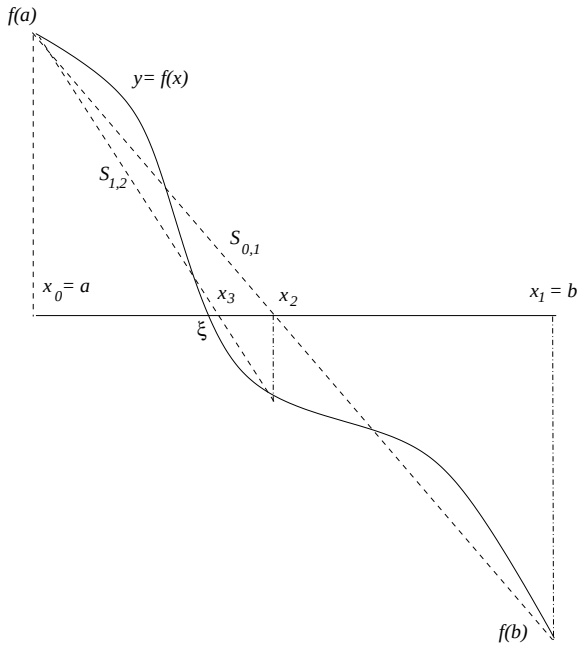

FIGURE 2.3. Méthode de « *fausse position* »

la justification (si elle s'avère licite) de la convergence et que la méthode de Newton est une méthode dite d'ordre 2.

2.1.2. La méthode de « fausse position ». Décrivons maintenant une méthode à pas multiples (deux en fait), la *méthode de fausse position*. On suppose simplement ici (pour l'instant) que la fonction f (toujours de classe au moins C^2) est strictement monotone sur $[a,b]$, et que $f(a)f(b) < 0$. On suppose pour fixer les idées $f' < 0$ sur $[a,b]$ (comme sur la figure 2.3). La fonction f s'annule (comme dans la section précédente) en un unique point ξ de $]a,b[$, que l'on se propose d'approcher. On ne fait pas pour l'instant aucune hypothèse portant sur le sens de concavité de f (il peut varier sur $[a,b]$, comme sur la figure 2.3).

On part cette fois du couple de points $x_0 = a$, $x_1 = b$ (notre méthode sera cette fois une méthode à deux pas, tout au moins en ce qui concerne l'étape « calcul ») : à chaque cran, x_{k+1} sera fonction des deux valeurs obtenues aux crans précédents, à savoir x_k et x_{k-1}.
On construit la droite (dite *sécante*) $S_{0,1}$ joignant les points $(x_0, f(x_0))$ et $(x_1, f(x_1))$; cette droite a pour équation

$$y - f(x_1) = \frac{f(x_1) - f(x_0)}{x_1 - x_0}\,(x - x_0)\,;$$

elle coupe l'axe des abscisses en un point x_2 tel que

$$0 - f(x_1) = \frac{f(x_1) - f(x_0)}{x_1 - x_0}\,(x_2 - x_1)\,,$$

soit

$$x_2 = x_1 - f(x_1)\frac{x_1 - x_0}{f(x_1) - f(x_0)}\,.$$

Une fois ce calcul de x_2 effectué, intervient un processus de choix (`[if ... else ...]`) : on calcule $f(x_2)$ et l'on opère comme suit :

- soit $f(x_2)$ est du signe de $f(x_1)$ (c'est le cas sur notre figure 2.3), auquel cas le calcul de x_3 se fait comme précédemment, mais cette fois à partir des valeurs de x_0 et x_1 réactualisées, c'est-à-dire :

$$\begin{aligned} x_0 &\longleftarrow x_0 \\ x_1 &\longleftarrow x_2 \end{aligned}$$

  (de manière à ce que les x_0 et x_1 réactualisés continuent, comme les précédents, à encadrer le point ξ) ;
- soit $f(x_2)$ est du signe de $f(x_0)$, auquel cas, on pose

$$\begin{aligned} x_0 &\longleftarrow x_2 \\ x_1 &\longleftarrow x_1 \end{aligned}$$

  (de manière à ce que les x_0 et x_1 réactualisés continuent, comme les précédents, à encadrer le point ξ) .

On poursuit suivant ce processus pour construire un processus itératif à deux pas convergent vers le point ξ. Pour déterminer x_3, il faut, en effet disposer des valeurs de x_0, x_1, x_2 en mémoire, autant pour le processus de décision, que, une fois ce processus de décision effectué, pour le calcul de x_3 (à partir des valeurs de x_0 et x_1 réactualisées comme indiqué ci-dessus). Ce calcul de x_3 n'implique, une fois le choix décidé, que deux de ces trois valeurs (x_0 et x_2, ou bien x_1 et x_2).

Cette méthode est dite méthode de « *fausse position*[4] », ou aussi des « *regula falsi* ».

Une fois le processus de décision implémenté et les valeurs de x_k et x_{k-1} réactualisées comme indiqué plus haut, le calcul à effectuer est donc celui de

$$x_{k+1} = x_k - f(x_k)\frac{x_k - x_{k-1}}{f(x_k) - f(x_{k-1})}. \tag{2.2}$$

On peut préférer à cette forme la forme plus « symétrique »

$$x_{k+1} = \frac{x_{k-1}f(x_k) - x_k f(x_{k-1})}{f(x_k) - f(x_{k-1})}. \tag{2.3}$$

Ce sera d'ailleurs cette seconde formulation (2.3) que nous exploiterons dans la sous-section 2.2.3 pour préciser la vitesse de convergence de la méthode et comparer les performances de cet algorithme avec celui de Newton.

On note qu'un des avantages de la méthode de fausse position (par rapport à la méthode de Newton), indépendamment des questions de rapidité de convergence dont nous reparlerons, est que la mise en place du calcul itératif (2.2) ou (2.3) ne nécessite que la connaissance des valeurs (éventuellement tabulées) de la fonction f, pas celles de sa dérivée f' (comme c'était le cas avec l'itération (2.1)).

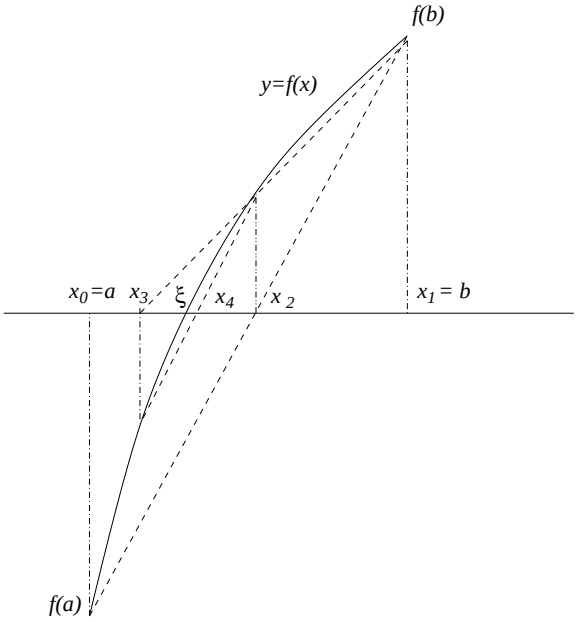

FIGURE 2.4. Méthode de la « *sécante* »

2.1.3. La méthode de la sécante. La *méthode de la sécante*, plus simple à implémenter (car elle n'implique pas de processus de décision), se base simplement sur l'itération à deux pas (2.2) ou (2.3). La routine (écrite sous `MATLAB`) correspondant à son implémentation est donc :

```
function xi=secante(init1,init2,N);
x1=init1 ;
x2=init2 ;
for i=1:N
   y=x1 ;
   x1=x2 ;
% il faut garder en effet cette valeur de x2 en memoire pour la suite
  delta = f(x2) - f(y);
  if delta == 0
     x=(x1+x2)./2 ;
  else
     x2=x2 - (f(x2).*(x2-y))./delta ;
  end
end
xi=x2
```

(les valeurs initiales x_0 et x_1 sont ici `init1` et `init2`, et N est le nombre d'itérations). Notons qu'ici, il a été nécessaire d'introduire un test d'arrêt : la division par $f(x_k) - f(x_{k-1})$ dans (2.2) ou (2.3) au cran k n'est en effet possible que si, pour la machine, $f(x_k) \neq f(x_{k-1})$. Si ce test d'arrêt (`[if delta == 0 ... else ...]`)

4. La valeur x_k est la « fausse position » du zéro ξ, fausse position que l'algorithme itératif conduit ici s'emploie à corriger.

n'était pas incorporé dans la routine, le résultat fourni par le logiciel (ici, par exemple, `MATLAB`) serait `NaN` (« `Not a Number` »). Ici, dans le cas où $f(x_k) = f(x_{k-1})$ aux erreurs machine près, on peut décider du compromis $\xi = (x_{k-1}+x_k)/2$.

Nous pouvons ici comparer cette routine avec celle régissant la méthode de fausse position (que nous donnerons plus loin). Il faut noter toutefois que le fait de ne plus chercher à « piéger » ξ entre x_{k-1} et x_k (en préalable au calcul de x_{k+1}) peut fort bien nous faire sortir du cadre. Dans les bons cas (voir la figure 2.4), sous des hypothèses de stricte concavité, la convergence de la suite $(x_k)_{k\geq 0}$ vers ξ se présente comme une convergence alternée (x_k et x_{k+1} étant de part et d'autre de ξ). La convention consistant à poser $\xi = (x_{k-1}+x_k)/2$ est bien dans ce cas justifiée.

La routine correspondant à l'algorithme de *fausse position* est similaire, mais introduit en plus un processus de décision. Il convient donc de modifier ainsi la routine précédente :

```
function xi=fausseposition(init1,init2,N);
x1=init1;
x2=init2;
for i=1:N
   delta = f(x2) - f(x1);
   if delta == 0
      xi= (x1 + x2)./2;
   else
      y = x2 - (f(x2) .* (x2-x1))./delta ;
     fy = f(y).* f(x1);
     if fy < 0
       x1 = x1;
       x2 = y;
     else
       x1= x2;
       x2= y;
     end
   end
end
xi=x2;
```

2.2. Comparaison des méthodes en terme d'« *ordre* »

2.2.1. Les notions d'« ordre » et de « nombre de pas ». Une notion importante, celle d'*ordre*, rend compte de la rapidité de convergence, donc de l'efficacité, d'une méthode itérative (lorsque celle ci génère une suite $(x_k)_k$ convergente vers un zéro réel d'une fonction f donnée), tandis qu'une autre notion (celle de *nombre de pas*) rend compte de la capacité de stockage mémoire nécessaire pour implémenter la méthode récursive :

DÉFINITION 2.1 (ordre et nombre de pas). Soit p un entier positif non nul. Une méthode itérative (de calcul de zéro réel approché d'une fonction donnée f satisfaissant certaines hypothèses au voisinage de ce zéro), basée sur l'utilisation d'une relation inductive

$$x_{k+1} = F_f(x_k, x_{k-1}, ..., x_{k-(p-1)}), \ k \geq p-1, \tag{2.4}$$

et initiée donc avec des données initiales $x_0, ..., x_{p-1}$, est dite méthode itérative *à p pas*. On dit que cette méthode est d'ordre $q \in]0, +\infty[$ si et seulement si q est la borne supérieure de l'ensemble des nombres réels strictement positifs κ tels que, dès que la suite $(x_k)_{k\geq 0}$ converge vers une limite finie x_∞ (zéro d'une fonction f remplissant, au voisinage de x_∞, les conditions sous lesquelles la méthode est implémentée), on a

$$\limsup_{k\to+\infty} \frac{|x_{k+1} - x_\infty|}{|x_k - x_\infty|^\kappa} < \infty\,. \tag{2.5}$$

REMARQUE 2.1. Souvent, on se contente d'une minoration κ de q, ce qui nous permet de dire que l'ordre de la méthode est ≪ au moins ≫ égal à κ. On peut par exemple choisir κ de manière à ce que la clause (2.5) soit remplie pour toute suite $(x_k)_{k\geq 0}$ convergent vers une limite x_∞ (zéro de f sous les conditions dans lesquelles on travaille). Pour prouver que l'ordre est ≪ exactement ≫ égal à κ, et donc être sûr que le $q = \kappa$ ainsi choisi réalise la borne supérieure de l'ensemble de tous les κ possibles (telle que (2.5) soit satisfaite pour toute suite $(x_k)_{k\geq 0}$ convergent vers une limite finie x_∞, zéro réel d'une fonction f sous les conditions sous lesquelles est envisagé le problème), il convient en plus d'exhiber une suite $(x_k)_{k\geq 0}$ générée par l'algorithme itératif (2.4) à partir des p données initiales $x_0, ..., x_{p-1}$, convergent vers une limite finie x_∞ (qu'il convient donc de connaitre, ce qui complique ici le problème, et qui soit zéro de la fonction f sous les conditions imposées), et telle que, pour cette suite,

$$\liminf_{k\to+\infty} \frac{|x_{k+1} - x_\infty|}{|x_k - x_\infty|^\kappa} = l \in]0, \infty[.$$

Ceci n'est pas toujours si facile ! En revanche, un minorant de l'ordre q est donné par tout réel strictement positif κ tel que, pour toute suite $(x_k)_{k\geq 0}$ générée par l'algorithme et convergent vers une limite finie x_∞ (qui soit un zéro réel de la fonction f en jeu), il existe une constante C (dépendant *a priori* de la suite $(x_k)_{k\geq 0}$), telle que

$$|x_{k+1} - x_\infty| \leq C|x_k - x_\infty|^\kappa\,,\ \forall\, k \geq 0\,.$$

2.2.2. La méthode de Newton est au moins d'ordre 2. Que la méthode de Newton (pour la recherche des zéros réels d'une fonction f de classe C^2 dont la dérivée ne s'annule pas au voisinage de tels zéros) soit d'ordre au moins égal à 2 repose sur la Proposition 2.1 suivante.

PROPOSITION 2.1. *Soit f une fonction de classe C^2 sur l'intervalle $]\xi - r, \xi + r[$, s'annulant en $x = \xi$, et telle que f' ne s'annule pas dans $]\xi - r, \xi + r[$. Soit*

$$\gamma = \sup_{x,u\in]\xi-r,\xi+r[} \frac{|f''(u)|}{|f'(x)|}.$$

Si $\gamma r < 2$, la suite de Newton $(x_k)_{k\in\mathbb{N}}$ régie par (2.1), initiée en un point x_0 de $]\xi - r, \xi + r[$, converge vers ξ lorsque k tend vers l'infini, et l'on a même les estimations

$$\begin{aligned} &|x_{k+1} - \xi| \leq \frac{\gamma}{2}\,|x_k - \xi|^2 \\ &|x_k - \xi| \leq \Big(\frac{\gamma r}{2}\Big)^{2^k-1} |x_0 - \xi| \quad \forall\, k \in \mathbb{N}. \end{aligned} \tag{2.6}$$

DÉMONSTRATION. Prenons $x \in]\xi - r, \xi + r[$ et supposons que

$$N_f(x) = x - \frac{f(x)}{f'(x)}$$

appartienne encore à $]\xi - r, \xi + r[$. En utilisant la formule de Taylor avec reste intégral (deux fois, une fois avec f, une fois avec f'), il vient

$$\begin{aligned} N_f(x) - \xi &= x - \xi - \frac{f(x) - f(\xi)}{f'(x)} \\ &= \frac{1}{f'(x)}\Big(f'(x)(x-\xi) - (f(x) - f(\xi))\Big) \\ &= \frac{1}{f'(x)}\Big((x-\xi)\Big(f'(\xi) + (x-\xi)\int_0^1 f''(tx + (1-t)\xi)\, dt\Big) \\ &\qquad -\Big(f'(\xi)(x-\xi) + (x-\xi)^2 \int_0^1 (1-t) f''(tx + (1-t)\xi)\, dt\Big)\Big) \\ &= (x-\xi)^2 \int_0^1 t \frac{f''(tx + (1-t)\xi)}{f'(x)}\, dt. \end{aligned}$$

On en déduit l'estimation

$$|N_f(x) - \xi| \leq \frac{\gamma}{2}|x - \xi|^2. \tag{2.7}$$

Si nous supposons que $x_0, x_1, ..., x_k$ restent dans $]\xi - r, \xi + r[$ et que

$$|x_k - \xi| \leq \frac{\gamma}{2}|x_{k-1} - \xi|^2, \quad |x_k - \xi| \leq \Big(\frac{\gamma r}{2}\Big)^{2^k - 1}|x_0 - \xi|,$$

on constate grâce à (2.7) que la première inégalité dans (2.6) est satisfaite. D'autre part :

$$|x_{k+1} - \xi| \leq \frac{\gamma}{2}|x_k - \xi|^2 \leq \frac{\gamma}{2}\Big(\frac{\gamma r}{2}\Big)^{2^{k+1}-2}|x_0 - \xi|^2 \leq \Big(\frac{\gamma r}{2}\Big)^{2^{k+1}-1}|x_0 - \xi|,$$

ce qui prouve la seconde inégalité dans (2.6) par récurrence. Comme $\gamma r/2 < 1$, il y a bien convergence vers 0 de la suite géométrique de raison $\gamma r/2$, donc convergence de la suite de Newton $(x_k)_k$ vers la racine ξ de f. □

Cette proposition prouve que la méthode de Newton (sous les hypothèses faites sur f, à savoir que f est de classe C^2 et de dérivée non nulle au point ξ, donc au voisinage de ξ par continuité) est d'ordre au moins égal à 2. Pour voir qu'elle est d'ordre 2, prendre par exemple la fonction

$$f \ : x \in \mathbb{R} \mapsto x^2 - \alpha^2$$

au voisinage du réel $\alpha > 0$. On a dans ce cas, si $x = \alpha + h$,

$$N_f(x) = x - \frac{x^2 - \alpha^2}{2x} = \frac{x^2 + \alpha^2}{2x} = \frac{2\alpha^2 + 2\alpha h + h^2}{2(\alpha + h)} = \alpha + \frac{h^2}{2\alpha} + o(|h|^2).$$

On constate que, pour cet exemple (c'est l'*algorithme de Babylone* pour le calcul approché des racines carrées)

$$|x_{k+1} - \alpha| \sim \frac{1}{2\alpha}|x_k - \alpha|^2.$$

L'ordre de la méthode de Newton est donc bien égal à 2. En conclusion, on peut énoncer :

PROPOSITION 2.2 (ordre de la méthode de Newton). *L'ordre de la méthode de Newton (dans le cadre f de classe C^2 et de dérivée non nulle aux zéros réels) est égal à* 2.

2.2.3. Ordre des méthodes de la sécante et de fausse position. Nous nous contenterons ici d'énoncer le résultat suivant :

PROPOSITION 2.3 (ordre de la méthode de la sécante ou de la méthode de fausse position). *Dans le contexte des fonctions f de classe C^2 et de dérivée non nulle aux zéros réels, l'ordre de la méthode de la sécante (ou de celle de fausse position) est au moins égal au nombre d'or* $(1+\sqrt{5})/2 \simeq 1.618....$

REMARQUE 2.2. En fait, on pourrait démontrer que cet ordre est exactement égal au nombre d'or en examinant, une fois encore, l'approximation du zéro $\alpha > 0$ de $x \mapsto x^2 - \alpha^2$.

DÉMONSTRATION. On suppose pour simplifier que le zéro de f que l'on cherche à approcher (limite de la suite $(x_k)_{k\geq 0}$) est $\xi = 0$. On suppose aussi que la fonction f est de classe C^2 dans un voisinage ouvert du segment $[-\epsilon, \epsilon]$ dans lequel se trouvent tous les points x_k, $k \geq 0$.

On peut écrire, on l'a vu, la relation entre x_{k-1}, x_k et x_{k+1} sous la forme plus ≪ symétrique ≫ (2.3) :

$$x_{k+1} = \frac{x_{k-1}f(x_k) - x_k f(x_{k-1})}{f(x_k) - f(x_{k-1})}\,.$$

Ainsi a t'on, si $e_k := x_k - \xi = x_k$,

$$|e_{k+1}| = |e_{k-1}|\,|e_k|\,|\varphi(x_{k-1}, x_k)|\,,$$

avec

$$\varphi(u,v) := \frac{\frac{f(u)}{u} - \frac{f(v)}{v}}{f(u) - f(v)}\,.$$

La fonction de deux variables

$$(u,v) \longmapsto \varphi(u,v)$$

est en fait bornée sur $[-\epsilon, \epsilon]$: pour justifier ceci, on utilise en effet au numérateur et au dénominateur la formule des accroissements finis [5], applicable tant au numérateur qu'au dénominateur. En effet, les hypothèses faites sur la fonction f (de classe C^2 avec ici $f(0) = 0$) impliquent que

$$x \in [-\epsilon, \epsilon] \longmapsto \frac{f(x)}{x}$$

(prolongée par $f'(0)$ en 0) est de classe C^1 sur $[-\epsilon, \epsilon]$. On sait que $|f'| \geq \eta > 0$ sur $[-\epsilon, \epsilon]$ (pourvu que ϵ soit choisi assez petit), et que la dérivée de $\check{f} : x \mapsto f(x)/x$ est majorée en module par une constante $K > 0$ sur $[-\epsilon, \epsilon]$ (comme fonction continue sur ce segment). En écrivant

$$\varphi(u,v) = \frac{\check{f}'(\check{\theta}_{u,v})}{f'(\theta_{u,v})}\,, \quad \theta_{u,v},\ \check{\theta}_{u,v} \in [-\epsilon, \epsilon],$$

5. Voir le cours d'Analyse 1 [**Yan**].

on trouve que $|\varphi| \leq M = K/\eta$ sur $[-\epsilon, \epsilon]$. Si l'on pose $\rho_k = M|e_k|$, $k \in \mathbb{N}$, on constate que

$$\rho_{k+1} \leq \rho_k \rho_{k-1}\,,\ k \geq 1\,.$$

En prenant le logarithme, il vient

$$\log \rho_{k+1} \leq \log \rho_k + \log \rho_{k-1}\,.$$

Si l'on introduit maintenant la suite récurrente (à deux pas) définie par les conditions initiales

$$\alpha_0 = \log \rho_0 < 0, \quad \alpha_1 = \log \rho_1 < 0$$

(quitte à supposer que x_0 et x_1 sont déjà dans $]-1/M, 1/M[$, on peut en effet supposer $\rho_0 < 1$ et $\rho_1 < 1$) et par la relation inductive

$$\alpha_{k+1} = \alpha_k + \alpha_{k-1} \quad \forall\, k \geq 1,$$

(celle qui donne les *nombres de Fibonacci*), on constate (par récurrence sur k) que

$$\forall\, k \in \mathbb{N}\,,\ \log \rho_k \leq \alpha_k\,.$$

Mais l'on sait [6] que

$$\alpha_k = \lambda \xi^k + \mu \eta^k = \lambda\Big(\frac{1+\sqrt{5}}{2}\Big)^k + \mu\Big(\frac{1-\sqrt{5}}{2}\Big)^k\,,$$

où ξ et η sont les racines de $X^2 - X - 1 = 0$; la constante μ (calculée à partir des conditions initiales $\alpha_0 < 0$ et $\alpha_1 < 0$) étant strictement négative (car tous les α_k le sont par récurrence), on a

$$\alpha_k \leq -C\Big(\frac{1+\sqrt{5}}{2}\Big)^k$$

pour k assez grand pour un certain $C > 0$, donc, en passant aux exponentielles [7],

$$M|e_k| \leq (e^{-C})\Big(\frac{1+\sqrt{5}}{2}\Big)^k\,,$$

(toujours pouvu que k soit assez grand). Ce raisonnement montre bien que que l'ordre de la méthode de la sécante (ou de la méthode de fausse position basée sur la même formule inductive à deux pas, écrite sous forme symétrique comme (2.3)) est au moins égal au nombre d'or. □

2.3. La méthode de la *dichotomie*

La méthode de dichotomie est aussi une méthode à deux pas, mais cette fois (on le verra plus loin, cf. la Proposition 2.4) d'ordre $q = 1$; on verra aussi (en le justifiant) que c'est une méthode *linéaire*. C'est la plus ancienne des méthodes itératives visant au calcul approché des zéros réels d'une fonction d'une variable réele sur un segment de $\mathbb{R}$. Si cette méthode est la plus lente (au niveau vitesse de convergence) des méthodes présentées dans ce chapitre, il n'en reste pas moins qu'il s'agit souvent de la plus robuste, en même temps que de la plus infaillible (sous la simple hypothèse que f est continue et strictement monotone sur un segment $[a, b]$, avec $f(a) \times f(b) < 0$).

---

6. Voir les cours de MISMI [**Y0**] et d'Analyse 1 [**Yan**] pour l'étude des suites récurrentes à deux pas : $u_{k+1} = au_{k-1} + bu_k$.

7. On rappelle que $\log|e_k| = \log \rho_k - \log M \leq \alpha_k - \log M$ pour tout $k \in \mathbb{N}$.

Son principe est très simple : on part encore d'un segment $[a, b]$ sur laquelle f vérifie les propriétés précédentes (f continue, strictement monotone, $f(a)$ et $f(b)$ de signes opposés). On part de $x_0 = a$, $x_1 = b$ et on calcule

$$c = \frac{x_0 + x_1}{2}\,.$$

Si $f(x_0)f(c) \leq 0$, on pose $x_2 = x_0$ et $x_3 = c$ et on continue ; si $f(x_0)f(c) > 0$, on pose $x_2 = c$ et $x_3 = x_1$ et on poursuit ainsi, suivant le processus itératif décrit dans la routine suivante :

```
function x=dichot(init1,init2,N);
x1=init1;
x2=init2;
for i=1:N
   y=(x1+x2)/2;
   u=f(x1)*f(y);
   if u <= 0
   x1=x1;
   x2=y;
   else
       x1=y;
       x2=x2;
   end
end
x=x2
```

Notons que l'algorithme sur lequel se fonde la méthode de fausse position inclut dans son synopsis une procédure de dichotomie.

Cette méthode est seulement linéaire, comme l'est, ce qui justifie l'épithète, le calcul de $y = (x_k + x_{k-1})/2$ en fonction de x_k et x_{k-1}. Elle peut être néanmoins exploitée pour déterminer les nombres a et b à partir lesquels on s'autorisera à lancer une méthode plus rapide (par exemple, celle de Newton ou celle de fausse position ou de la sécante), offrant ainsi un premier « dégrossissage » de la situation avec une première localisation grossière du zéro éventuel ξ entre $x_0 = a$ et $x_1 = b$. Sur notre exemple

$$x \longmapsto f(x) = x^5 + \frac{x^3}{2} + 1\,,$$

la méthode de dichotomie initiée avec $x_0 = -1$ et $x_1 = -1/2$ fournit par exemple :

```
>> dichot(-1,-.5,3)
  -0.875000000000000
>> dichot(-1,-.5,5)
  -0.906250000000000
>> dichot(-1,-.5,10)
  -0.909667968750000
>> dichot(-1,-.5,20)
  -0.909824848175049
>> dichot(-1,-.5,50)
  -0.909824890637916
>> dichot(-1,-.5,100)
  -0.909824890637916
```

On constate que la vitesse de convergence ici est loin d'être comparable à celle donnée par l'algorithme de Newton (ou par les algorithmes de « fausse position » ou de la sécante). Nous avons en fait la Proposition suivante.

PROPOSITION 2.4 (ordre de la méthode de dichotomie). *La méthode de dichotomie est une méthode linéaire à deux pas, d'ordre $q = 1$.*

DÉMONSTRATION. Comme la méthode repose sur la relation inductive

$$x_{k+1} = \frac{x_k + x_{k-1}}{2}$$

(x_{k+1} est une fonction linéaire des deux entrées x_k et x_{k-1}), la méthode de dichotomie (qui est aussi une méthode à deux pas) mérite son qualificatif de *linéaire*. On vérifie de plus par récurrence que

$$x_{k+1} - \xi = \frac{1}{2}(x_k - \xi) + \frac{1}{2}(x_{k-1} - \xi)$$

et donc, si $e_k := x_k - \xi$ pour $k \in \mathbb{N}$, que

$$e_{k+1} = \frac{e_k + e_{k-1}}{2}\,.$$

Si $e_{k+1} = O(e_k^q)$ avec $q > 1$, on aurait

$$e_k = -e_{k-1} + O(e_k^q)\,,$$

donc $e_k \simeq -e_{k-1}$, ce qui imposerait $q \leq 1$. L'ordre de la méthode de dichotomie ne peut donc être strictement supérieur à 1. Il est de fait égal à 1 du fait de la linéarité de la méthode. □

CHAPITRE 3

Polynômes, interpolation, élimination

Au travers du maniement des polynômes (sous l'angle du calcul symbolique, mais aussi sous l'angle du calcul scientifique), on esquissera dans ce chapitre une introduction à deux champs de calcul importants (tant sous l'aspect symbolique que sous l'aspect scientifique) : l'*interpolation* et l'*élimination*. On envisagera également l'interpolation *via* les polynômes trigonométriques, ce qui constituera une première approche aux outils présidant à l'étude digitale des signaux (1D) et des images (2D).

3.1. Quelques généralités en prise avec l'algorithmique

3.1.1. Le ≪ codage ≫ d'un polynôme en machine ; l'algorithme de Hörner. Un polynôme de degré N,

$$P(X) = a_0 X^N + a_1 X^{N-1} + \cdots + a_N,$$

avec coefficients dans un corps commutatif $\mathbb{K}$, ou même un anneau commutatif unitaire $\mathbb{A}$, se code par son degré N et par la liste des coefficients

$$[a_0, a_1, \cdots, a_N],$$

présentée comme une matrice ligne à $N+1$ entrées. Les coefficients sont rangés dans l'ordre ci dessus, en commençant par le coefficient (non nul) du monôme de degré maximal (ce degré vaut ici N) ; si un monôme tel X^k ne figure pas, on code l'entrée a_{N-k} correspondante comme un zéro. Par exemple, sous `MATLAB`, on déclare par

```
>> P = [-3 4 1 0 2 -6];
```

le polynôme de degré 5 :

$$P(X) = -3X^5 + 4X^4 + X^3 + 2X - 6\,.$$

Les $N+1$ entrées a_k, $k = 0, ..., N$, qui sont les coefficients du polynôme, sont soit des éléments de $\mathbb{Z}$ ou $\mathbb{Q}$ (que l'on code alors ≪ sans pertes ≫ en travaillant sous un logiciel de calcul symbolique, tel `Maple13`), soit des entrées réelles ou complexes (dont on code des approximations en virgule flottante sous un logiciel de calcul scientifique tel `MATLAB`). Les entrées $a_0, ..., a_N$ peuvent aussi être des expressions formelles ou des mots fabriqués à partir d'un nombre fini de symboles, ce qui se passe lorsque l'on envisage de travailler avec le polynôme P sous l'angle du *calcul symbolique*.

L'évaluation (on dit aussi la ≪ spécification ≫) du polynôme en un élément ou un symbole x (qui peut être un nombre ou une expression formelle pourvu que l'on sache donner un sens aux expressions :

$$x \cdot a + \tilde{a}$$

lorsque $a, \tilde{a}$ sont deux éléments de $\mathbb{A}$ et à l'itération de ce processus), se fait par l'algorithme itératif décrit par la formule

$$a_N + \left[x \cdot \left[a_{N-1} + \cdots + \left[x \cdot \left[x \cdot [x \cdot a_0 + a_1] + a_2 \right] + a_3 \right] \cdots \right] \right]$$

et par le code suivant (sous `MATLAB`) ; ce code évalue le polynôme dont les coefficients sont donnés par le vecteur ligne `A` sur un vecteur ligne d'entrées réelles ou complexes `XX` :

```
function P=Horner(A,XX);
N=length(A);
NN=length(XX);
P=A(1)*ones(1,NN);
for i=2:N
      P = XX.* P + A(i);
end
```

Applioqué à un scalaire `X=x` (`NN=1`), cet algorithme « consommme » N multiplications et N additions (en fait N opérations du type $x * (\cdot) + (\cdot)$) ; la méthode naive consistant à calculer toutes les puissances x^k, $k = 1, ..., N$ nécessite aussi N multiplications mais en plus de l'espace mémoire (ce que ne nécessite pas la démarche algorithmique présentée ci dessus) ; il reste ensuite (toujours si l'on suit cette méthode naive) N multiplications à effectuer pour évaluer

$$\sum_{k=0}^{N} a_k \cdot x^{N-k} = P(x) \ ;$$

cette méthode naive s'avère donc plus coûteuse (en temps et en espace mémoire). L'algorithme décrit dans le synopsis présenté plus haut (synopsis synthétisant une boucle, un test d'arrêt et un branchement) est dit *algorithme de Hörner*[1] . C'est lui que, par exemple, les commandes telles que

```
>> P = [-3 4 1 0 2 -6];
>> y = polyval (P,x);
```

(sous `MATLAB`) exploitent pour le calcul du vecteur ligne de scalaires

$$(P(x_0), ..., P(x_M))$$

si $x_0, x_1, ..., x_M$ sont $M+1$ valeurs (ici nombres réels ou complexes car on travaille avec un logiciel de calcul scientifique) spécifiées.

Sous `Maple13`, ce sont les commandes du type

```
> eval (P(x),x)
> eval (f(x,y),x)
```

qui réalisent l'évaluation d'un polynôme $P(X)$ (dont les coefficients peuvent tout aussi bien être des expression symboliques, par exemple des polynômes en une nouvelle variable Y indépendante de X) sur un scalaire x ou une expression symbolique (par exemple $f(X, Y)$, où f est une expression rationnelle en X et Y).

1. C'est au mathématicien anglais William George Hörner (1786-1837) que l'on attribue cet algorithme, même si probablement l'origine remonte bien au delà dans l'histoire des mathématiques et du calcul.

3.1.2. La division euclidienne (suivant les puissances « décroissantes ». On utilisera ici essentiellement la structure d'anneau euclidien de $\mathbb{K}[X]$, liée au fait qu'existe dans $\mathbb{K}[X]$ un *algorithme de division euclidienne* : étant donnés $H \in \mathbb{K}[X]$ et $P \in \mathbb{K}[X]$, avec $\deg P = N \geq 0$, il existe un unique couple (Q, R) de $\mathbb{K}[X] \times \mathbb{K}[X]$ avec

$$H(X) = P(X)Q(X) + R(X)$$

et $\deg R < N$ (le polynôme identiquement nul étant considéré comme de degré $-\infty$).

Si P_1 et P_2 sont deux polynômes non tous les deux nuls de $\mathbb{K}[X]$, l'ensemble des polynômes H de $\mathbb{K}[X]$ de la forme

$$H(X) = P_1(X)A_1(X) + P_2(X)A_2(X)$$

coincide avec l'ensemble des polynômes multiples du dernier reste non nul Δ dans l'algorithme de division euclidienne de P_1 par P_2 (si $\deg P_2 \geq 0$) ou de P_2 par P_1 (si $\deg P_1 \geq 0$). Le polynôme non nul Δ de degré inférieur à $\max(\deg P_1, \deg P_2)$ ainsi défini est appelé PGCD (*plus grand diviseur commun*) de P_1 et P_2 ; il est défini à la multiplication par un élément non nul de $\mathbb{K}$ près.

C'est en remontant les calculs conduits dans l'algorithme de division euclidienne (de P_1 par P_2 ou de P_2 par P_1) que l'on réalise une identité polynômiale (dite *identité de Bézout*[2]), de la forme

$$\Delta(X) = U(X)P_1(X) + V(X)P_2(X)$$

avec $\deg U \leq \deg P_2 - 1$ et $\deg V \leq \deg P_1 - 1$. En particulier, si $P_1 \equiv 0$, on a $\Delta = P_2$ et l'identité de Bézout devient simplement $\Delta = P_2 = 1 \times P_2$.

Lorsque $\mathbb{K} = \mathbb{C}$, le fait que $\deg(\mathrm{PGCD}(P_1, P_2)) > 0$ équivaut à ce que P_1 et P_2 aient une racine commune dans $\mathbb{C}$. Ceci résulte du théorème fondamental de l'algèbre, à savoir le théorème de Jean Lerond d'Alembert, suivant lequel tout polynôme à coefficients complexes et de degré N se factorise dans $\mathbb{C}[X]$ sous la forme

$$P(X) = a_0 \prod_{j=1}^{N} (X - \lambda_j)\,,$$

où $\lambda_1, ..., \lambda_N$ sont des nombres complexes ; si λ_j est répété exactement m_j fois dans cette liste, on dit que l'entier strictement positif m_j est la *multiplicité* de λ_j comme zéro du polynôme P ; dire que $\lambda_j \in \mathbb{C}$ est zéro de multiplicité $m_j \in \mathbb{N}^*$ équivaut à dire

$$P(\lambda_j) = P'(\lambda_j) = \cdots P^{(m_j-1)}(\lambda_j) = 0\,,$$

où $P^{(k)}$ désigne, pour $k \in \mathbb{N}$, le k-ème polynôme dérivé de P.

---

2. Si les travaux du mathématicien et professeur français Etienne Bézout (1730-1783) traitent tant de la résolution des équations algébriques que de l'élimination (où il fit réellement œuvre de précurseur, ses travaux devant être repris au siècle suivant par exemple par Cayley et Kronecker), on n'y trouve nulle trace de cette fameuse identité ! On peut néanmoins penser que l'influence des travaux de Bézout dans l'enseignement (écoles d'artillerie ou de marine) à travers ses divers traités didactiques explique cette terminologie.

3.1.3. La division suivant les puissances croissantes. Tout aussi important, sinon plus, que l'algorithme de division euclidienne (qui consiste à traiter la division en présentant les polynômes de manière à ce que les monômes soient rangés dans l'ordre décroissant, ce qui correspond à la présentation de Hörner, voir la sous section 3.1.1), l'algorithme de division suivant les puissances croissantes est initié avec deux entrées polynomiales A et B dont les monômes sont rangés cette fois dans l'ordre croissant :

$$A(X) = a_0 + a_1X + \cdots + a_NX^N\,,\ B(X) = b_0 + b_1X + \cdots + b_0X^M.$$

Pour démarrer, il faut (ceci est essentiel) que $b_0 \neq 0$. On écrit

$$A(X) = \frac{a_0}{b_0}B(X) + A_1(X)$$

avec

$$A_1(X) = \alpha_{11}X + \alpha_{12}X^2 + \cdots$$

(les termes constants sont asbsents de A_1). On peut donc recommencer et diviser le « reste » A_1 par B :

$$A_1(X) = \frac{\alpha_{11}}{b_0}XB(X) + A_2(X),$$

avec

$$A_2(X) = \alpha_{22}X^2 + \alpha_{23}X^3 + \ldots$$

(les termes de degré 0 et 1 sont absents de A_2). On continue de la sorte et l'on trouve après k opérations (ce processus est sans fin, contrairement à celui de la division euclidienne)

$$A(X) = B(X)(\gamma_0 + \gamma_1X + \cdots + \gamma_kX^k) + B_k(X),$$

avec

$$B_k(X) = \alpha_{k+1,k+1}X^{k+1} + \alpha_{k+1,k+2}X^{k+2} + \cdots$$

Le polynôme B_k est appelé *reste à l'ordre k* dans le processus de division suivant les puissances croissantes, tandis que

$$Q_k(X) = \sum_{j=0}^{k}\gamma_jX^j$$

est appelé *quotient à l'ordre k* dans ce processus.

Vous avez rencontré cet algorithme très important en analyse, par exemple dans la recherche de développements limités pour des quotients. Il est tout aussi aisément implémentable que celui de la division euclidienne.

3.1.4. Une opération coûteuse : la multiplication des polynômes. Multiplier deux polynômes (de degrés respectifs N_1-1 et N_2-1) conduit (pour le calcul des coefficients) aux formules dites de Cauchy :

$$\begin{aligned}&\Big(u(0) + u(1)X + \cdots + u(N_1-1)X^{N_1-1}\Big)\times\\ &\times\Big(v(0) + v(1)X + \cdots + v(N_2-1)X^{N_2-1}\Big) =\\ &= \sum_{k=0}^{N_1+N_2-2} w(k)X^k\,,\end{aligned} \tag{3.1}$$

avec

$$w(k) = \sum_{k_1+k_2=k} u(k_1)v(k_2),\ k = 0, ..., N_1 + N_2 - 2.$$

Multiplier deux polynômes de degrés $N-1$ de manière naive consomme N^2 multiplications. Il suffit de penser à la multiplication de deux polynômes de degré 10000 pour entrevoir le coût du calcul !

Le problème de la *multiplication rapide des polynômes*[3] (c'est-à-dire de leur multiplication « à moindre coût ») a été résolu par les ingénieurs informaticiens américains J.W. Cooley et J.W. Tukey vers 1965. Le résultat majeur de Cooley-Tukey est à la genèse de ce que l'on appelle communément aujourd'hui la *révolution numérique* des années 1965-1970.. Nous nous contentons ici d'en esquisser une présentation.

Si l'on choisit $N \geq N_1 + N_2 - 1$, la relation (3.1) est équivalente à

$$\Big(\sum_{k=0}^{N_1-1} u(k)\, X^k\Big) \times \Big(\sum_{k=0}^{N_2-1} v(k)\, X^k\Big) \equiv \sum_{k=0}^{N_1+N_2-2} w(k)\, X^k \mod X^N - 1, \tag{3.2}$$

ou encore, puisque les N racines complexes du polynôme $X^N - 1$ sont les N racines complexes N-ième de l'unité $\exp(-2i\pi j/N)$, $j = 0, ..., N-1$, au système de N relations :

$$\Big(\sum_{k=0}^{N_1-1} u(k)\, e^{-\frac{2i\pi kj}{N}}\Big) \times \Big(\sum_{k=0}^{N_2-1} v(k)\, e^{-\frac{2i\pi kj}{N}}\Big) = \sum_{k=0}^{N_1+N_2-2} w(k)\, e^{-\frac{2i\pi kj}{N}}, \tag{3.3}$$
$$j = 0, ..., N-1.$$

Posons $W_N := \exp(-2i\pi/N)$ et remarquons maintenant la propriété algébrique suivante :

LEMME 3.1 (matrice de transformation de Fourier discrète[4] `dft`). *Soit $N \in \mathbb{N}^*$ et $\mathtt{dft}\,(N)$ la matrice carrée symétrique à coefficients complexes de terme général W_N^{kj}, k indice de ligne, j indice de colonne. La matrice $\mathtt{dft}\,(N)$ est inversible, d'inverse*

$$[\mathtt{dft}\,(N)]^{-1} = \frac{1}{N}\,\overline{\mathtt{dft}\,(N)}. \tag{3.4}$$

DÉMONSTRATION. Il suffit de remarquer que

$$\sum_{k=0}^{N-1} W_N^{jk} = \begin{cases} N & \text{si} \quad j = 0 \\ 0 & \text{si} \quad j \neq 0 \end{cases}$$

car $X^N - 1 = (X-1)(1 + X + \cdots + X^{N-1})$. □

Si les vecteurs-ligne $[u(0), ..., u(N_1-1)]$, $[v(0), ..., v(N_2-1)]$, $[w(0), ..., w(N_1+N_2-2)]$ sont complétés par des zéros en des vecteurs ligne U, V, W de longueur N, puis

3. Nous envisageons ici les choses du point de vue numérique (car nous ferons appel à des nombres complexes tels les racines N-ièmes de l'unité dont on ne connait que des approximations). Cependant, des idées similaires peuvent être introduites dans un cadre plus arithmétique, avec la notion de *transformée de Fourier arithmétique*. Signalons aussi qu'un autre algorithme de multiplication rapide (dans un contexte algébrique) a été proposé en 1960 par le mathématicien russe Anatolii Karatsuba (1937 –).

4. « Pour *Discrete Fourier Transform* ».

transposés en des vecteurs colonne ${}^tU$, ${}^tV$ et ${}^tW$ de longueur N, on peut donc exprimer le système de relations (3.3) sous la forme

$$ {}^tW = \frac{1}{N}\,\overline{\mathtt{dft}\,(N)}\cdot\Big[\Big(\mathtt{dft}(N)\cdot{}^tU\Big)\,.*\,\Big(\mathtt{dft}(N)\cdot{}^tV\Big)\Big] \tag{3.5}$$

où $.*$ désigne (comme par exemple dans les environnements `MATLAB` ou `Scilab`) la multiplication des vecteurs colonne de longueur N entrée par entrée.

Lorsque $N = 2^p$, avec $p \in \mathbb{N}^*$, on doit aux deux ingénieurs informaticiens américains James William Cooley and John Wilder Tukey (autour de 1965) la construction d'un algorithme permettant d'implémenter les opérations de multiplication matricielle

$$\mathtt{dft}(2^p)\cdot{}^tU,\quad \Big(\text{resp.}\quad \frac{1}{2^p}\,\overline{\mathtt{dft}(2^p)}\cdot{}^t\widehat{W}\Big),$$

où ${}^tU$ et ${}^t\widehat{W}$ sont deux vecteurs colonne de longueur 2^p donnés, avec $p2^{p-1}$ (au lieu de $(2^p)^2 = 2^{2p}$) multiplications (en fait seulement $(p-1)2^{p-1}$ si l'on prend en compte que 2^{p-1} de ces multiplications sont des multiplications par -1, que l'on peut donc considérer comme des additions au niveau de la complexité). Nous énonçons ici ce résultat majeur, moteur de ce qui allait être la *révolution numérique* des années 1968-1970.

THEORÈME 3.1 (algorithme de Cooley-Tukey (environ 1965)). *Lorsque $N = 2^p$, $p \in \mathbb{N}^*$, il existe un algorithme consommant seulement $p \times 2^{p-1}$ multiplications (parmi lesquelles 2^{p-1} sont des multiplications par -1) permettant la multiplication matricielle*

$$\mathtt{dft}(2^p)\cdot{}^tU,\quad \Big(\text{resp.}\quad \frac{1}{2^p}\,\overline{\mathtt{dft}(2^p)}\cdot{}^t\widehat{W}\Big),$$

lorsque ${}^tU$ et ${}^t\widehat{W}$ sont deux vecteurs colonne donnés de longueur 2^p. Cet algorithme est appelé « Algorithme de Transformation de Fourier Rapide », ou plus communément $\mathtt{fft}(2^p)$ *pour « Fast Fourier Transform » (respectivement « Algorithme de Transformation de Fourier Rapide Inverse », ou plus communément* $\mathtt{ifft}(2^p)$ *pour « Inverse Fast Fourier Transform »).*

REMARQUE 3.1. Sous l'environnement `MATLAB` ou `Scilab`, ces algorithmes s'implémentent *via* les routines

```
>> (hatU)' = fft(U',N);
>> W' = ifft ((hatW)',N);
```

DÉMONSTRATION. La clef de l'algorithme de Cooley-Tukey consiste à profiter du fait que l'on a

$$\mathtt{dft}(2) = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

(l'action de cette matrice, dite « action papillon », est donc très simple) et à enchainer, pour calculer une prise de $\mathtt{fft}(2^k)$, deux prises de $\mathtt{fft}(2^{k-1})$ avec 2^{k-1} prises de $\mathtt{dft}(2)$. Il est commode de visualiser cette idée (sous-tendant la récursivité) grâce au diagramme suivant (où $W_N := \exp(-2i\pi/N)$).

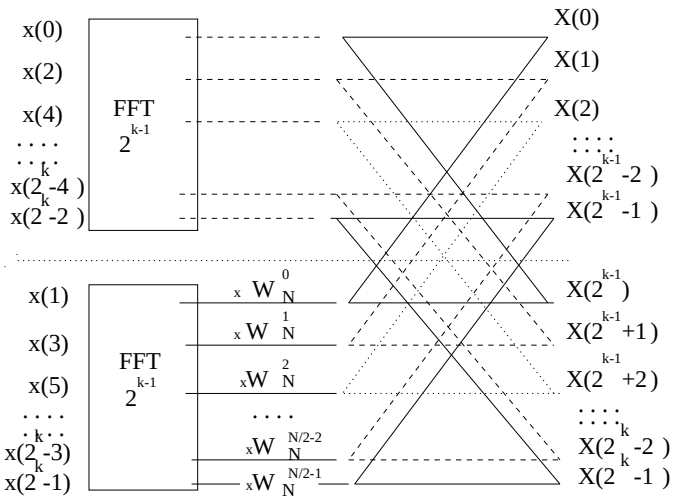

FIGURE 3.1. *Algorithme de Cooley Tukey* $N/2 = 2^{k-1} \to N = 2^k$

L'algorithme récursif à implémenter pour calculer par exemple $\texttt{dft}(2^p) \cdot X$ est présenté comme l'algorithme 1 ci-dessous. On remarque que cet algorithme implique à chaque étape (lignes 5 et 6) une réorganisation des entrées (réalisée par un renversement de l'un des bits des indices des entrées, $0 \to 1$, $1 \to 0$). Il est aisé de constater que cet algorithme consomme

$$\frac{2^p}{2} + 2\frac{2^p}{4} + \cdots + 2^{p-1}\frac{2^p}{2^p} = p\frac{2^p}{2} = p \times 2^{p-1}$$

multiplications, dont 2^{p-1} sont en fait des multiplications par $e^{-2i\pi/2} = -1$. L'algorithme $\texttt{ifft}(2^p)$ est en tous points similaire : il suffit de remplacer w par $\overline{w}$ et de conclure avec une division par N. □

Algorithme 1 `fft` $(X, Y, 2^p, w)$

1: $w \Leftarrow e^{-2i\pi/2^p}$
2: **si** $p = 0$ **alors**
3: $\quad Y(0) \Leftarrow X(0)$
4: **sinon**
5: $\quad$ `fft` $([X(0), ..., X(2^p - 2)], [U(0), ..., U(2^{p-1})], 2^{p-1}, w^2)$
6: $\quad$ `fft` $([X(1), ..., X(2^p - 1)], [V(0), ..., V(2^{p-1})], 2^{p-1}, w^2)$
7: $\quad$ **pour** $k = 0$ **jusqu'à** $2^p - 1$ **faire**
8: $\quad\quad Y(k) \Leftarrow U(k \bmod 2^{p-1}) + w^k\, V(k \bmod 2^{p-1})$
9: $\quad$ **fin pour**
10: **fin si**

On peut donc conclure :

PROPOSITION 3.1 (multiplication rapide de deux polynômes de degrés au plus $2^{p-1}-1$). *Suivant* (3.5) *avec* $N = 2^p > 2\times 2^{p-1}-1$, *la multiplication rapide de deux*

polynômes de degrés $2^{p-1}-1$ *nécessite trois prises de* `fft`(2^p) *ou* `ifft`(2^p)*, chacune consommant* $p \times 2^{p-1}$ *multiplications (cf. le théorème de Cooley-Tukey 3.1), et* 2^p *multiplications supplémentaires (correspondant aux multiplications terme à terme* `.*` *au second membre de* (3.5)*). Le bilan final du nombre de multiplications est donc :*

$$N_{2^{p-1}-1}(\times) = 3 \times p \times 2^{p-1} + 2^p = 2^{p-1}(3p+2) << 2^{2(p-1)}.$$

Le gain est évidemment d'autant plus appréciable que p *est très grand.*

REMARQUE 3.2. C'est l'algorithme de Cooley-Tukey (valable pour les puissances de 2) qui explique que les résolutions d'écran d'ordinateur sont du type 1024×1024, 2048×2048... On retrouve ces puissances de 2 dans la longueur des signaux digitaux (par exemple en téléphonie mobile).

3.2. Interpolation de Lagrange et différences divisées

L'interpolation des fonctions par des fonctions polynomiales s'impose de manière naturelle du fait que les fonctions polynomiales sont les seules fonctions qu'il est possible de coder en machine sous forme de listes de données. Le problème de l'interpolation polynomiale s'avère donc un problème majeur, non exempt de réelles difficultés, tant de nature théorique que numérique. Nous l'abordons ici avec le modèle le plus simple (mais non le plus performant, on le verra), celui de l'*interpolation de Lagrange.*

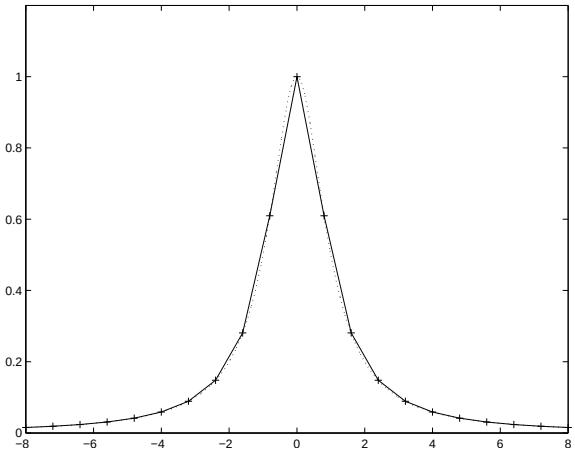

FIGURE 3.2. Le graphe de $x \longmapsto 1/(1+x^2)$ sur $[-8,8]$ et les points (x_k, y_k) « marqués »

3.2.1. Définition ; une formule inexploitable. Soient $x_0 = a, ..., x_N = b$, $N+1$ nombres réels ou complexes distincts. Lorsque l'on pense à des nombres réels, on pensera à $x_0 = a < x_1 < \cdots < x_{N-1} < x_N = b$ comme des points intermédiaires d'un intervalle $[a,b]$ fermé borné donné. Dans notre exemple, le vecteur

$$X = (x_0, ..., x_N)$$

est le vecteur (ici à $21 = N+1$ entrées)

```
>> x=-8:16/20:8;
```

des 21 points régulièrement espacés entre -8 et 8

$$x_k := -8 + \frac{16k}{20} \ , ..., \ k = 0, ..., 20 \, .$$

Associons à ces valeurs x_k, $k = 0, ..., N$, des valeurs numériques $(y_0, ..., y_N)$, réelles ou complexes. Dans l'exemple que nous traitons ici, nous prendrons par exemple

```
>> y = 1./ (1+ x.^2);
```

ce qui signifie

$$y = (y_0, ..., y_N)$$

et

$$y_k = \frac{1}{1 + x_k^2} \, , \ k = 0, ..., N \, .$$

Les points (x_k, y_k) affichés ici avec des croix sont des points du graphe de la fonction

$$f \ : x \in [-8, 8] \longmapsto \frac{1}{1 + x^2} \, .$$

Comme on le voit sur la figure 3.2, le graphe de la fonction affine par morceaux interpolant ces points (en plein sur la figure) ne rend pas compte (par exemple sur $[-2, 2]$) de la forme du graphe de f, en pointillés sur la figure, qui présente par exemple une tangente horizontale au point $(0, 1)$.

Le $\mathbb{R}$ (*resp.* $\mathbb{C}$)- espace vectoriel des polynômes à coefficients réels (*resp.* complexes) de degré au plus N est un $\mathbb{R}$ (*resp.* $\mathbb{C}$)-espace vectoriel de dimension $N+1$ dont une base est constituée de l'ensemble des monômes $\{1, X, X^2, ..., X^N\}$. Ecrire qu'un tel polynôme

$$P(X) = \sum_{k=0}^{N} a_k X^{N-k}$$

prend des valeurs spécifiéees $y_0, ..., y_N$ (réelles ou complexes) aux points distincts $x_0, ..., x_N$ revient à écrire les $N + 1$ contraintes

$$P(x_k) = \sum_{k=0}^{N} a_k x_j^k = y_j \, , \ j = 0, ..., N \, .$$

Ces contraintes correspondent à un système linéaire

$$M(x_0, ..., x_N) \cdot \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_N \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_N \end{pmatrix} \tag{3.6}$$

où $M(x_0, ..., x_N)$ est une matrice carrée dont on vérifie que le déterminant (dit de Vandermonde [5]) est égal à

$$\begin{vmatrix} x_0^N & x_0^{N-1} & \dots & x_0 & 1 \\ x_1^N & x_1^{N-1} & \dots & x_1 & 1 \\ \vdots & \vdots & \vdots & \vdots & \\ x_{N-1}^N & x_{N-1}^{N-1} & \dots & x_{N-1} & 1 \\ x_N^N & x_N^{N-1} & \dots & x_N & 1 \end{vmatrix} = \prod_{0 \le k < l \le N} (x_k - x_l) \neq 0 \,.$$

Ceci prouve que le système (3.6) est un système de Cramer qui admet une unique solution $(a_0, ..., a_N)$. Le polynôme

$$P_N(X\,; Y) = a_0 X^N + a_1 X^{N-1} + \cdots + a_N$$

correspondant est l'unique polynôme de degré N prenant les valeurs prescrites des coordonnées du vecteur ligne Y

$$Y = (y_0, ..., y_N)\,,$$

(dans cet ordre) respectivement aux points $x_0, ..., x_N$. Ce polynôme est dit *polynôme d'interpolation de Lagrange* [6] attaché aux données

$$\{x_0, y_0\}\,,\ \{x_1, y_1\}\,,\ \dots\,,\ \{x_N, y_N\}\,.$$

La matrice $M(x_0, ..., x_N)$ est en général mal conditionnée et l'inverser n'est pas la bonne stratégie !

Certes, on vérifie immédiatement, si l'on pose

$$P(X) = (X - x_0) \cdots (X - x_N)\,,$$

que notre polynôme cherché s'écrit

$$\begin{aligned} P_N(X; Y) &= \sum_{k=0}^{N} \frac{P_k(X) - P_k(x_k)}{P_k'(x_k)(X - x_k)}\, y_k \\ &= \sum_{k=0}^{N} \frac{P_k(X)}{P_k'(x_k)(X - x_k)}\, y_k \\ &= \sum_{k=0}^{N} \frac{\prod\limits_{\substack{l=0 \\ l \neq k}}^{N} (X - x_l)}{\prod\limits_{\substack{l=0 \\ l \neq k}}^{N} (x_k - x_l)}\, y_k \,. \end{aligned} \tag{3.7}$$

Si l'on pose en effet $X = x_k$, on trouve bien $P_N(x_k; Y) = y_k$ pour $k = 0, ..., N$; le polynôme $P_N(\cdot\,; Y)$ étant l'unique polynôme ayant cette propriété, c'est bien lui qui est donné explicitement par la formule (3.7) ci-dessus.

---

5. Mathématicien franais contemporain d'Etienne Bézout, Alexandre-Théophile Vandermonde (1735-1796) s'est intéressé à la résolution des équations algébriques en même temps qu'à la combinatoire ; le dernier des quatre articles qu'il a rédigé est sans doute l'article fondateur de la théorie des déterminants.

6. Mathématicien franco-italien (1736-1813), Joseph-Louis Lagrange marqua profondément tant l'algèbre que l'analyse (en particulier le calcul différentiel) et la mécanique ; ce fut aussi un astronome.

Tout ce que nous avons dit dans cette section s'adapte au cas où $x_0, ..., x_N$ sont $N+1$ nombres complexes distincts et $y_0, ..., y_N$ nombres complexes. Dans ce nouveau contexte, avant de clôre cette section, il n'est pas inintéressant de remarquer que la construction de polynômes d'interpolation de Lagrange fournit, lorsque $\mathbb{K} = \mathbb{C}$, une résolution directe de l'identité de Bézout envisagée dans la sous-section 3.1.2. On a en effet la

PROPOSITION 3.2 (interpolation de Lagrange et identité de Bézout). *Soient P et Q deux polynômes à coefficients complexes, de degrés respectifs $p > 0$ et $q > 0$, n'ayant tous les deux que des racines simples ($\lambda_1, ..., \lambda_p$ pour P, $\mu_1, ..., \mu_q$ pour Q)*[7]. *Si P et Q n'ont aucune racine commune et si A désigne le polynôme d'interpolation de Lagrange (de degré exactement $q-1$) interpolant les valeurs $1/P(\mu_j)$, avec $j = 1, ..., q$ aux points $\mu_1, ..., \mu_q$ et B le polynôme d'interpolation de Lagrange (de degré exactement $p-1$) interpolant les valeurs $1/Q(\lambda_j)$, avec $j = 1, ..., p$, aux points $\lambda_1, ..., \lambda_p$, on a l'identité de Bézout*

$$1 = A(X)P(X) + B(X)Q(X)\,.$$

DÉMONSTRATION. Il suffit de remarquer que le polynôme

$$1 - A(X)P(X) - B(X)Q(X)$$

s'annule en les $p + q$ points distincts $\lambda_1, ..., \lambda_p, \mu_1, ..., \mu_q$ et est de degré au plus $p + q - 1$. C'est donc le polynôme identiquement nul et la proposition est ainsi démontrée. □

REMARQUE 3.3 (identité de Bézout en plusieurs variables : un puissant outil en robotique). Si cette proposition est d'un intérêt mineur lorsque l'on travaille avec des polynômes en une variable (l'algorithme de division euclidienne étant dans ce cas un outil algorithmique de complexité optimale), il n'en est plus de même lorsque l'on recherche (comme c'est le cas dans nombre de problèmes pratiques surgis par exemple de la robotique) une identité de Bézout

$$1 = A_1(X_1, ..., X_n)P_1(X_1, ..., X_n) + \cdots + A_m(X_1, ..., X_n)P_m(X_1, ..., X_n)\,,$$

---

7. Cette condition de simplicité des zéros peut être levée, mais il faut disposer de la notion de polynôme d'interpolation de Lagrange à des points éventuellement multiples, notion que nous n'avons pas développé en cours, mais qui pourra être introduite par exemple en TP. En fait, on a l'identité de Bézout dès que l'on pose (racines multiples ou non pour P ou Q)

$$\begin{aligned} A(X) &= \sum_{j=1}^{q} \mathrm{Res}_{Y=\mu_j}\Big[\frac{Q(X)}{(X-Y)P(Y)Q(Y)}\Big] \\ B(X) &= \sum_{j=1}^{p} \mathrm{Res}_{Y=\lambda_j}\Big[\frac{P(X)}{(X-Y)P(Y)Q(Y)}\Big] \end{aligned}$$

si le résidu en $u = 0$ d'une fraction rationnelle $R(u) = N(u)/u^m D(u)$ (avec $D(0) \neq 0$) est le coefficient de u^{-1} dans le développement de R obtenu après division suivant les puissances croissantes de N par D. En effet, on constate que $A(X)P(X) + B(X)Q(X)$ s'écrit aussi

$$\sum_{j=1}^{q} \mathrm{Res}_{Y=\mu_j}\Big[\frac{P(X)Q(X) - P(Y)Q(Y))}{(X-Y)P(Y)Q(Y)}\Big] + \sum_{j=1}^{p} \mathrm{Res}_{Y=\lambda_j}\Big[\frac{P(X)Q(X) - P(Y)Q(Y))}{(X-Y)P(Y)Q(Y)}\Big] \equiv 1\,.$$

où $P_1, ..., P_m$ sont m polynômes en n variables sans zéros communs dans $\mathbb{C}^n$ [8]. Dans ce cas, le recours à des identités obtenues directement suivant le mécanisme suggéré par l'interpolation de Lagrange peut s'avèrer dans bien des cas algorithmiquement moins complexe que la (trop) coûteuse méthode d'*élimination*.

3.2.2. Différences divisées et méthode récursive d'Aitken. Utiliser la formule (3.7) n'est pas non plus la meilleure stratégie pour calculer le polynôme d'interpolation de Lagrange (comme d'ailleurs le fait de tenter de résoudre le système (3.6), ce à cause des probèmes de conditionnement mentionnés auparavant). On préfère utiliser la méthode (due à Newton) des *différences divisées* consistant à chercher à exprimer $P(\cdot\,; Y)$ non dans la base usuelle

$$\{1, X, ..., X^N\}$$

des $\mathbb{R}$ (*resp.* $\mathbb{C}$)-polynômes de degré au plus N à coefficients réels (*resp.* complexes), mais à l'exprimer dans la base

$$\{1\,,\,(X-x_0)\,,\,(X-x_0)(X-x_1)\,,\,...\,,\,(X-x_0)\cdots(X-x_{N-1})\}$$

(on peut prendre N points parmi les $N+1$ points x_k proposés et choisir un ordre arbitraire). Afin de spécifier la suite $x_0, ..., x_N$ choisie et les valeurs interpolées $y_0, ..., y_N$, nous noterons le polynôme d'interpolation de Lagrange $P_N(X;Y)$ plus précisément :

$$P_N(X;Y) = \text{Lagrange}[x_0, ..., x_N\,;\, y_0, ..., y_N](X),$$

Ce polynôme s'exprime alors comme

$$\begin{aligned}
&\text{Lagrange}[x_0, ..., x_N\,;\, y_0, ..., y_N](X) = y[x_0] + y[x_0, x_1]\,(X - x_0)\\
&+ y[x_0, x_1, x_2]\,(X - x_0)(X - x_1) + \cdots\\
&\cdots + y[x_0, x_1, ..., x_{N-1}] \prod_{j=0}^{N-2} (X - x_j) + y[x_0, x_1, ..., x_N] \prod_{j=0}^{N-1} (X - x_j),
\end{aligned} \tag{3.8}$$

où les nombres complexes $y[x_0, ..., x_n]$, $n = 0, ..., N$, sont appelés *différences divisées* des y_n par les x_n (dans l'ordre imposé $n = 0, ..., N$) et sont à extraire d'un tableau de nombres organisé suivant les règles

$$\begin{aligned}
&y[x_n] = x_n\\
&y[x_0, ..., x_n] = \frac{y[x_0, ..., x_{n-1}] - y[x_1, ..., x_n]}{x_0 - x_n}\ , \qquad n = 0, ..., N.
\end{aligned} \tag{3.9}$$

Le calcul des différences divisées s'organise suivant un algorithme triangulaire aisé à décrire sous forme d'un tableau à $N+2$ colonnes, numérotées de -1 à N. La colonne d'indice k ($k = 0, ..., N$) de ce tableau représente la suite des nombres $u_n^{(k)} := y[x_n, x_{n+1}, ..., x_{n+k}]$, $n = 0, ..., N-k$. Cette colonne d'indice N est donc réduite à un élément ($y[x_0, ..., x_N]$) tandis que la colonne d'indice $k = 0$ est la colonne des entrées $y_n = y[x_n]$, $n = 0, ..., N$. Il est commode de placer (pour mémoire) en position $k = -1$ la colonne des nœuds $x_0, ..., x_N$, colonne qui sera rappelée à chaque étape de la progression vers la droite dans la construction du

8. C'est à David Hilbert que l'on doit dans ce cas le résultat impliquant l'existence de tels polynômes $A_1, ..., A_m$.

tableau. On passe en effet de la colonne d'indice $k \geq 0$ à la colonne suivante (d'indice $k+1$) par la règle

$$u_n^{(k+1)} = \frac{u_n^{(k)} - u_{n+1}^{(k)}}{x_n - x_{n+k+1}}\,,\ n = 0, ..., N-k-1.$$

La présentation est donc la suivante :

$$\begin{array}{cccccc}
\underline{x_0} & y_0 = y[x_0] & & & & \\
\underline{x_1} & y_1 = y[x_1] & y[x_0, x_1] & & & \\
\underline{x_2} & y_2 = y[x_2] & y[x_1, x_2] & y[x_0, x_1, x_2] & & \\
\vdots & \vdots & \vdots & \vdots\searrow & & \\
\vdots & \vdots & \vdots & \vdots & & y[x_0, ..., x_N] \\
\vdots & \vdots & \vdots & \vdots\nearrow & & \\
\underline{x_{N-2}} & y_{N-2} = y[x_{N-2}] & y[x_{N-2}, x_{N-1}] & y[x_{N-2}, x_{N-1}, x_N] & & \\
\underline{x_{N-1}} & y_{N-1} = y[x_{N-1}] & y[x_{N-1}, x_N] & & & \\
\underline{x_N} & y_N = y[x_N] & & & &
\end{array}$$

On remarque que le calcul de la différence divisée finale $y[x_0, ..., x_N]$ est indépendant de l'ordre dans lequel on organise les nœuds x_n, $n = 0, ..., N$, pourvu que les y_n, $n = 0, ..., N$ soient organisés suivant la même permutation de $\{0, ..., N\}$.

Algorithmiquement, la démarche décrite ci-dessus se traduit par le code suivant sous `MATLAB` :

```
function DV=DiffDiv(X,Y)
N=size(X,2)-1;
U=zeros(N+1,N+2);
U(:,1)= conj(X');
U(:,2)= conj(Y');
DV = [Y(1)];
for j=2:N+1
    for i=1:N+2-j
        U(i,j+1)= (U(i,j)-U(i+1,j))/(U(i,1) - U(i+j-1,1));
    end
    DV = [U(1,j+1) DV];
end
```

Ici `X` et `Y` sont deux vecteurs ligne de longueur `N+1`, le premier désignant les abscisses (distinctes) ou les affixes complexes des points d'interpolation $x_0, ..., x_N$, le second les valeurs prescrites $y_0, ..., y_N$ précisément en ces points d'interpolation. La suite des différences divisées est fournie par cette routine dans l'ordre :

$$y[x_0, ..., x_N],\ y[x_0, ..., x_{N-1}], ..., y[x_0, x_1],\ y[x_0].$$

Cet ordre est en concordance avec l'utilisation de l'algorithme de Hörner (voir la section 3.1.1). La suite `DiffDiv (X,Y)` entre en jeu dans l'évaluation du polynôme d'interpolation de Lagrange en un point x *via* la formule (3.8) Cette méthode de calcul du polynôme d'interpolation de Lagrange *via* le calcul préalable des différences divisées est dite *méthode d'interpolation de Newton*. Sa complexité algorithmique est en $O(N^2)$ (il suffit de compter les multiplications). Sous `Maple`, voici comment

se visualise en calcul numérique (sous la forme d'un tableau triangulaire, comme sur la présentation algorithmique décrite plus haut) la table des différences divisées :

```
>> with(Student[NumericalAnalysis]);
>> P:=[[x0,y0],...,[xN,yN]];
>> PI:= PolynomialInterpolation(P,independentvar='x',method=newton):
>> DividedDifferenceTable(PI);
```

En calcul formel, on a par exemple (sous `Maple`) la syntaxe :

```
> with (CurveFitting);
> PolynomialInterpolation([x0, ... , xN], [y0, ...,yN], z, form = Newton);
```

Le lemme qui soutend pareille démarche est dû à A. Aitken [9] :

LEMME 3.2 (lemme d'Aitken). *Soient $x_0, ..., x_N$ $N+1$ nombres réels distincts et $y_0, ..., y_N$ $N+1$ nombres réels. Si Q désigne le polynôme d'interpolation de Lagrange interpolant les valeurs $y_0, ..., y_{N-1}$ respectivement aux points $x_0, ..., x_{N-1}$, R le polynôme d'interpolation de Lagrange interpolant $y_1, ..., y_N$ respectivement aux points $x_1, ..., x_N$, on a, si P désigne le polynôme d'interpolation de f aux points $x_0, ..., x_N$, la formule d'Aitken :*

$$P(X) = \frac{(X - x_0)R(X) - (X - x_N)Q(X)}{x_N - x_0}.$$

DÉMONSTRATION. On note

$$\widetilde{P}(X) = \frac{(X - x_0)R(X) - (X - x_N)Q(X)}{x_N - x_0}.$$

Il suffit de remarquer que, pour $k = 1, ..., N-1$,

$$\begin{aligned} \widetilde{P}(x_k) &= \frac{(x_k - x_0)R(x_k) - (x_k - x_N)Q(x_k)}{x_N - x_0} \\ &= \frac{(x_k - x_0)y_k - (x_k - x_N)y_k}{x_N - x_0} = y_k \end{aligned}$$

(puisqu'à la fois Q et R interpolent les valeurs y_k aux points $x_1, ..., x_N$) et que d'autre part

$$\begin{aligned} \widetilde{P}(x_0) &= \frac{-(x_0 - x_N)Q(x_0)}{x_N - x_0} = Q(x_0) = y_0 \\ \widetilde{P}(x_N) &= \frac{(x_N - x_0)R(x_N)}{x_N - x_0} = R(x_N) = y_N \end{aligned}$$

puisque Q interpole y_0 au point x_0 et que R interpole y_N au point x_N. Comme le degré de $\widetilde{P}$ est exactement égal à N (puisque Q et R sont deux polynômes unitaires de degré exactement $N-1$), $\widetilde{P}$ coincide bien avec le polynôme d'interpolation de Lagrange aux points $x_0, ..., x_N$. □

Ce lemme établi, voici un procédé algorithmique permettant de conduire de manière récursive le calcul du polynôme d'interpolation d'un nombre fini de valeurs en un nombre fini de points. Cette procédure est dite *méthode d'interpolation de Neville-Aitken*. Soient $x_0, ..., x_N$ $N+1$ nombres réels distincts et $y_0, ..., y_N$ $N+1$ nombres

9. Mathématicien néo-zélandais, Alexander Craig Aitken, 1895-1967, est aussi connu comme un calculateur mental ≪ prodige ≫ ; le lemme cité ici lui est certainement bien antérieur, mais non sans relation avec les méthodes d'accélération de convergence qu'il développa.

réels. On introduit la collection de polynômes $P_{k,j}$, $j = 0, ..., N$, $k = j, ..., N$ définis comme suit :

- $P_{k,0} \equiv y_k$, $k = 0, ..., N$;
- $P_{k,j}$, $k = j, ..., N$, est le polynôme d'interpolation de Lagrange interpolant les valeurs $y_{k-j}, ..., y_k$ respectivement aux points $x_{k-j}, ..., x_k$.

La relation de récurrence permettant de calculer $P_{k,j}$ à partir de $P_{k,j-1}$ et $P_{k-1,j-1}$ lorsque $j \geq 1$ et $k \geq j$ s'obtient aisément à partir du lemme 3.2 : le polynôme $P_{k,j-1}$ interpole les valeurs $y_{k-j+1}, y_{k-j}, ..., y_{k-1}, y_k$ aux points

$$x_{k-j+1}, ..., x_{k-1}, x_k$$

(c'est le polynôme constant y_k si $j = 1$), tandis que le polynôme $P_{k-1,j-1}$ interpole les valeurs $y_{k-j}, ..., y_{k-1}$ aux points $x_{k-j}, ..., x_{k-1}$ (c'est le polynôme constant y_{k-1} si $j = 1$) ; le rôle de Q dans le lemme d'Aitken est tenu par $P_{k,j-1}$, les points $x_0, ..., x_{N-1}$ étant maintenant les points $x_{k-j+1}, ..., x_{k-1}$; le rôle de R est tenu par $P_{k-1,j-1}$; le polynôme $P_{k,j}$ qui interpole les valeurs $y_{k-j}, ..., y_k$ aux points $x_{k-j}, ..., x_k$ est donc donné, si l'on utilise le lemme, par

$$P_{k,j}(X) = \frac{(x_{k-j} - X)P_{k,j-1}(X) - (x_k - X)P_{k-1,j-1}(X)}{x_{k-j} - x_k}\,,\ 1 \leq j \leq k\,.$$

Le calcul des $P_{k,j}$ peut ainsi être présenté sous la forme du tableau triangulaire suivant :

$$\begin{array}{llllllllll}
x_0 & P_{0,0} = y_0 & & & & & & & & \\
x_1 & P_{1,0} = y_1 & P_{1,1} & & & & & & & \\
x_2 & P_{2,0} = y_2 & P_{2,1} & P_{2,2} & & & & & & \\
\vdots & \vdots & \vdots & \vdots & & & & & & \\
x_k & P_{k,0} = y_k & P_{k,1} & P_{k,2} & \dots & P_{k,j} & \dots & P_{k,k} & & \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & & \\
x_N & P_{N,0} = y_N & P_{N,1} & P_{N,2} & \dots & P_{N,j} & \dots & P_{N,k} & \dots & P_{N,N}
\end{array}$$

La première colonne de cette matrice figure les polynômes $P_{k,0} \equiv y_k$. La démarche algorithmique consiste à calculer les colonnes de gauche à droite en utilisant la formule inductive (3.10) permettant de calculer l'entrée $P_{k,j}$ à partir des deux entrées $P_{k-1,j-1}$ et $P_{k,j-1}$ de la colonne précédente. Le polynôme cherché $P_N = P_{N,N}$ est l'entrée figurant dans le coin inférieur droit de ce tableau triangulaire (dit *tableau d'Aitken*).

La procédure récursive ainsi décrite se schématise comme ci-dessous :

```
function F=proc(N,f,a,b)
if N>1
   F = [(X-a)*proc(N-1,f, a+(b-a)/N,b)
                        -(X-b)*proc(N-1,f,a,b-(b-a)/N)]/(b-a)
if N=1
   F = [(X-a)*f(b) - (X-b)*f(a)]/(b-a)
```

lorsqu'il s'agit de retourner, N, f, a, b étant donnés, le polynôme d'interpolation $P_N[f]$ de f aux $N + 1$ points régulièrement espacés

$$x_k := a + k\,\frac{b-a}{N}\,,\quad k = 0, ..., N$$

d'un segment $[a, b]$ sur lequel est définie la fonction f.

Plus généralement, voici la procédure d'Aitken, telle que l'on peut, par exemple, l'implémenter de manière récursive sous MATLAB :

```
function P = lagrange (x,y);
N = length(x);
  if N == 1
    P= y(1);
  else
    P1 = lagrange (x(2:N),y(2:N));
    P2 = lagrange (x(1:N-1),y(1:N-1));
    P11 = [P1 0] - x(1)*[0 P1];
    P22 = [P2 0] - x(N)*[0 P2];
    P=(P11 - P22)./(x(N)-x(1));
  end
end
```

On notera cependant le coût important (en temps et en mémoire) de cet algorithme récursif. Sous Maple, l'interpolation de Lagrange est proposée suivant une routine toute intégrée certainement conçue *via* l'algorithme de Aitken :

```
> with(CurveFitting);
[ArrayInterpolation, BSpline, BSplineCurve, Interactive, LeastSquares,
PolynomialInterpolation, RationalInterpolation, Spline, ThieleInterpolation]
> PolynomialInterpolation([[0, 6], [1, 4], [2, 3], [3, 7]], z);
                            2  3   3  2   7
                            - z  - - z  - - z + 6
                            3      2      6
> PolynomialInterpolation([0, 3, a, 8], [2, 5, b, 7], z, form = Lagrange);
      (z - 3) (z - a) (z - 8)   5 z (z - a) (z - 8)   b z (z - 3) (z - 8)
    - ----------------------- + ------------------- + -------------------
               12 a                  -45 + 15 a          a (a - 3) (a - 8)

         7 z (z - 3) (z - a)
       + -------------------
              320 - 40 a
```

Il convient de signaler toutefois que, sous le logiciel MATLAB, l'interpolation de Lagrange est réalisée sous la commande polyfit :

```
>> P = polyfit(x,y,N);
```

Cependant, il apparait que cette commande polyfit n'implémente pas sous ce logiciel de calcul scientifique l'interpolation de Lagrange suivant le cheminement de l'algorithme de Aitken. En effet, il arrive très fréquemment que l'on se heurte à un problème de mauvais conditionnement (en fait de la matrice de Vandermonde qui se trouve ici inversée). Pour y parer, il s'avère nécéssaire de « normaliser » les données comme suit :

```
>> sigma = std (x) ;
>> xx= x./sigma ;
>> PP = polyfit (xx,y,N);
```

Ici par exemple :

$$\texttt{std}(x) := \sqrt{\frac{1}{N}\sum_{n=0}^{N}(x_n - \overline{x})^2},$$

où

$$\overline{x} := \frac{1}{N+1}\sum_{n=0}^{N} x_k$$

il s'agit du calcul de l'écart-type (« non biaisé » ici), « *standard deviation* » en anglosaxon. L'évaluation du polynôme de Lagrange $P(\cdot\,;\,Y)$ sur un vecteur de points `xxx` de l'intervalle d'étude est alors donnée par

```
>> yy = polyval(PP, xxx/sigma);
```

3.2.3. Interpolation de Lagrange et « effets de bord ». Soit une fonction numérique $f : [a,b] \to \mathbb{R}$, que l'on suppose de classe C^∞ sur $[a,b]$ (c'est-à-dire pouvant se prolonger en une fonction de classe C^∞ dans un voisinage ouvert de $[a,b]$). Si $x_0, ..., x_N$ sont $N+1$ points distincts de $[a,b]$, on a la proposition suivante, que l'on peut interpréter comme une « version décentrée » de la formule de Taylor-Lagrange (vue dans le cours d'Analyse 2).

Proposition 3.3 (version décentrée de la formule de Taylor-Lagrange). *Pour tout $x \in [a,b]$, il existe $\xi_x \in [a,b]$, tel que :*

$$f(x) = \text{Lagrange}[x_0, ..., x_N\,;\, f(x_0), ..., f(x_N)](t) + \frac{f^{N+1}(\xi_x)}{(N+1)!}\prod_{j=0}^{N}(x - x_j). \tag{3.10}$$

En particulier, l'erreur uniforme entre f et la fonction polynomiale de degré N $P_N[f]$ interpolant les valeurs de f aux points $x_0, ..., x_N$ est estimée en :

$$\max_{[a,b]}|f - P_N[f]| \leq \frac{1}{(N+1)!}\sup_{t\in[a,b]}\Big(\prod_{j=0}^{N}|t - x_j|\Big) \times \sup_{[a,b]}|f^{(N+1)}|. \tag{3.11}$$

Démonstration. La preuve de la Proposition 3.3 repose sur une application répétée du théorème de Rolle à la fonction

$$\begin{aligned} t \mapsto\ & \text{Lagrange}\left[x_0, ..., x_m\,;\, f(x_0), ..., f(x_m)\right](t) - f(t) + \\ & + \Big(f(x) - \text{Lagrange}\left[x_0, ..., x_m\,;\, f(x_0), ..., f(x_m)\right](x)\Big)\prod_{j=0}^{n}\frac{t - x_j}{x - x_j}. \end{aligned} \tag{3.12}$$

On suppose ici que x est distinct de tous les x_j (dans le cas contraire, la formule (3.10) est immédiate). Cette fonction admet $m+1$ zéros distincts dans $[a,b]$. Le théorème de Rolle appliqué m fois assure que que sa dérivée d'ordre $m+1$ s'annule en un point ξ_x de $]a,b[$, ce qui donne le résultat voulu. □

Pour rendre compte des problèmes pratiques que peut soulever l'interpolation de Lagrange, reprenons l'exemple de la fonction

$$t \mapsto \frac{1}{1+t^2} = \frac{1}{2}\Big(\frac{1}{t+i} + \frac{1}{t-i}\Big).$$

Notons que la dérivée d'ordre $N+1$ de cette fonction est

$$t \mapsto \frac{(-1)^{N+1}(N+1)!}{2} \times \frac{(t+i)^{N+2} + (t-i)^{N+2}}{(1+t^2)^{N+2}}.$$

L'interpolation de Lagrange à partir d'une subdivision de $[-8,8]$ de pas constant introduit (en ce qui concerne l'approximation de f en convergence uniforme) des artefacts de plus en plus prononcés au fur et à mesure que l'on s'approche des bords de l'intervalle $[-8,8]$. Ce phénomène (qui ne fait que s'aggraver lorsque l'on augmente le nombre de points) s'appelle *phénomène de Runge*[10] . En témoignent les figures :

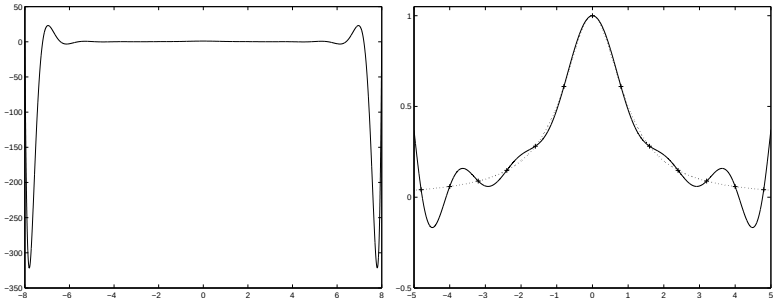

FIGURE 3.3. Interpolation de Lagrange à pas constant et approximation uniforme

Sur la figure de gauche, on a représenté le graphe de P_N sur $[-8,8]$ lorsque $N=20$, les 21 points x_j étant uniformément répartis sur $[-8,8]$. Les oscillations très amplifiées lorsque l'on s'approche des extrémités font que le graphe de P_{20} « décroche » complètement de celui de la fonction $t \mapsto 1/(1+t^2)$ que P_{20} est censée interpoler. En revanche, sur un intervalle tel $[-3,3]$, il semble que P_{20} réalise une approximation en norme uniforme convenable pour la fonction f. Resserer le maillage semble ainsi s'avérer nécessaire lorsque la dérivée $f^{(21)}$ est en valeur absolue trop grande. Les $N+1$ zéros du polynôme T^{N+1} de Tchebychev (de degré $N+1$ et donné par $T^{N+1}(\cos(\theta)) = \cos((N+1)\theta)$), tous réels et situés dans $]-1,1[$, fournissent une liste de points s'avérant en un certain sens « optimale » pour interpoler les fonctions numériques par des fonctions polynomiales sur $[a,b]=[-1,1]$. Transposée à un segment $[a,b]$ quelconque, cette suite devient la suite

$$x_k = \frac{a+b}{2} + \frac{b-a}{2}\cos\big(\frac{2k+1}{2(N+1)}\,\pi\big), \quad k=0,...,N.$$

Voir par exemple la section 4.2.3 de [**Y2**] pour plus de détails en même temps qu'une justification de ce fait que l'on observera empiriquement en TP.

3.3. Interpolation polynomiale approchée au sens des moindres carrés

Pour pallier aux difficultés que soulève (en terme de rendu au niveau de l'approximation uniforme) le procédé d'interpolation exact de Lagrange (*cf.* la sous-section 3.2.3), nous allons proposer une approximation polynomiale en norme non plus uniforme, mais *quadratique*[11].

10. Carl Runge, mathématicien appliqué (mais aussi théorien) et physicien allemand (1856-1927) le souligna.

11. Pour une suite discrète $\{z_0,...,z_N\}$ de nombres complexes, la norme quadratique (ou euclidienne) $\|z\| = (\sum_j |z_j|^2)^{1/2}$ s'interprète en physique comme la racine carrée de l'énergie. Cette norme dérive d'un produit scalaire sur $\mathbb{C}^N$.

Si $x_0, ..., x_N$ sont $N+1$ nombres réels distincts, l'ensemble des fonctions définies sur l'ensemble $\{x_0, ..., x_N\}$ et à valeurs réelles a une structure de $\mathbb{R}$-espace vectoriel $\mathcal{V}$ de dimension $N+1$. La somme $f+g$ de deux fonctions est la fonction définie par

$$(f+g)(x_k) := f(x_k) + g(x_k)\,,\ k = 0, ..., M\,;$$

la multiplication d'une fonction f par un scalaire $\lambda \in \mathbb{R}$ est la fonction définie par

$$(\lambda \cdot f)(x_k) = \lambda f(x_k)\,,\ k = 0, ..., N\,.$$

L'ensemble des restrictions à $\{x_0, ..., x_N\}$ des fonctions polynômiales de degré M $(0 \leq M \leq N)$ est donc un $\mathbb{R}$-sous-espace vectoriel $\mathcal{W}_M$ de $\mathcal{V}$ de dimension $M+1$: ceci résulte du fait que, si $x_0, ..., x_N$ sont $N+1$ nombres réels distincts, les vecteurs

$$\begin{pmatrix} 1 \\ x_j \\ x_j^2 \\ \vdots \\ x_j^N \end{pmatrix}\,,\ j = 0, ..., N\,,$$

sont linéairement indépendants, ce qui implique que la matrice constituée avec leurs colonnes est de rang $N+1$ (son déterminant est un déterminant de Vandermonde, voir les cours d'Algèbre 1 et Algèbre 2). Une base de $\mathcal{W}_M$ est constituée des restrictions à $\{x_0, ..., x_N\}$ des fonctions

$$Y_k\ : x \mapsto x^k\,,\ k = 0, ..., M\,.$$

Sur l'espace $\mathcal{V}$, on peut définir un produit scalaire par

$$\langle f\,,\, g\rangle := \sum_{j=0}^{N} f(x_j)g(x_j)\,.$$

On peut donc définir une opération de projection orthogonale sur le sous-espace vectoriel $\mathcal{W}_M$ (voir le cours d'Algèbre 2). La fonction polynomiale p de degré M telle que

$$\sum_{j=0}^{N} (y_j - p(x_j))^2$$

soit minimale est l'élément de $\mathcal{W}_M$ le plus proche de la fonction

$$Y\ : x_k \mapsto y_k, \quad k = 0, ..., N\,,$$

relativement à la distance associée à ce produit scalaire. On a donc, grâce au théorème de Pythagore,

$$p = \mathrm{Proj}_{\mathcal{V}_M}[Y]\,.$$

Cette projection se calcule aisément si l'on dispose d'une base orthonormée de $\mathcal{W}_M$; malheureusement, la construction d'une telle base *via* le procédé d'orthonormalisation de Gram-Schmidt [12] s'avère souvent délicate numériquement car impliquant

12. Si on l'attribue au mathématicien allemand Erhard Schmidt (1876-1959), spécialiste d'analyse fonctionnelle, et au mathématicien danois Jørgen Pedersen Gram (1853-1916), qui s'est penché sur la méthode des moindres carrés, ce procédé algorithmique (rencontré dans le cours d'Algèbre 2) est certainement bien plus ancien et connu de Cauchy et Laplace au début du XIX-ème siècle (Cauchy le manipula expressément vers 1830).

des problèmes de sous-conditionnement ; on peut aussi calculer cette projection en écrivant

$$\langle Y - p\,,\, Y_k\rangle = 0\,, \quad k = 0, ..., M\,,$$

soit encore

$$\Big\langle Y - \sum_{j=0}^{M} a_j x^j\,,\, x^k \Big\rangle = 0\,,\ k = 0, ..., M\,.$$

Ceci s'écrit

$$\sum_{j=0}^{N} \Big(y_j - \sum_{l=0}^{M} a_l x_j^l \Big) x_j^k = 0\,,\ k = 0, ..., M\,,$$

c'est-à-dire comme un système linéaire en $a_0, ..., a_M$. La matrice d'un tel système (dite *matrice de Gram*) s'avère être souvent mal conditionnée. Sur l'exemple ci-dessous, on a représenté un nuage de points (20 points de coordonnées x, y avec $-1 \leq x \leq 2$ et $4 \leq y \leq 10$) sur lequel on a fait agir les routines `MATLAB` :

```
>> P= polyfit(x,y,1);
>> PP= polyfit(x,y,10);
>> PPP=polyfit(x,y,14);
>> t=-0.5:.01:1.5 ;
>> z=polyval (P,t);
>> zz=polyval (PP,t);
>> zzz=polyval (PPP,t);
>> plot(x,y,'+');
>> hold
>> plot(t,z,'r');
>> plot(t,zz,'-.');
>> plot(t,zzz,'k');
```

On a ainsi affiché les graphe des meilleures aproximations polynomiales quadratiques respectivement de degrés 1, 10 et 14.

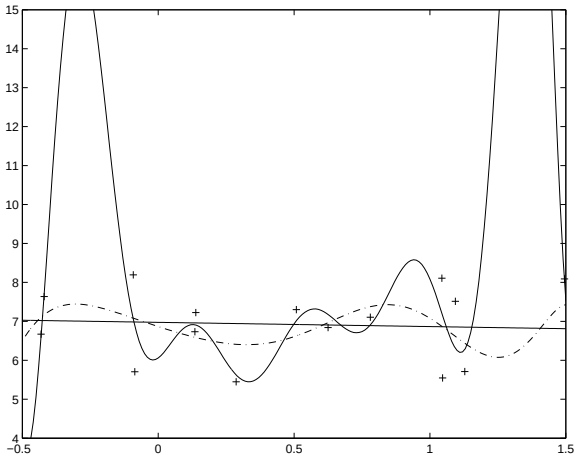

FIGURE 3.4. Approximation polynomiale au sens des moindres carrés

3.4. Une sensibilisation au principe de l'≪ élimination ≫ algébrique

Si P et Q sont deux polynômes non nuls de degrés respectifs $p > 0$ et $q > 0$ à coefficients dans un corps $\mathbb{K}$, dire que P et Q sont premiers entre eux dans $\mathbb{K}[X]$, *i.e* ont un PGCD égal à 1 (ce que l'on teste en conduisant dans $\mathbb{K}[X]$ l'algorithme d'Euclide) équivaut à affirmer qu'il existe des polynômes A, B, de degrés respectifs $q-1$ et $p-1$, tels que

$$A(X)P(X) + B(X)Q(X) \equiv 1\,.$$

On constate d'ailleurs puisque tout autre solution $(\widetilde{A}, \widetilde{B})$ de $\widetilde{A}P + \widetilde{B}Q \equiv 1$ s'écrit alors (à cause du lemme de Gauss [13])

$$(\widetilde{A}, \widetilde{B}) = (A, B) + H \times (Q, -P)\,,$$

où H est un polynôme arbitraire dans $\mathbb{K}[X]$, que la solution (A, B) avec ces degrés $(q-1, p-1)$ précisés est unique.

On conclut donc de ce qui précède que le fait que P et Q soient premiers entre eux dans $\mathbb{K}[X]$ équivaut à ce que le système de $p+q$ équations en $p+q$ indéderminées (les coefficients des polynômes A et B que l'on cherche) obtenu en écrivant l'égalité polynomiale

$$\Big(\sum_{j=0}^{q-1} u_j X^j\Big)P(X) + \Big(\sum_{j=0}^{p-1} v_j X^j\Big)Q(X) \equiv 1 \tag{3.13}$$

(la différence est un polynôme de degré $p+q-1$, il y a donc $p+q$ équations affines en les inconnues à écrire) est un système de Cramer.

En résumé, si P et Q sont de degrés respectifs exactement $p > 0$ et $q > 0$, dire que P et Q sont premiers entre eux dans $\mathbb{K}[X]$ équivaut à dire que le déterminant de ce système (que l'on appelle *résultant de Sylvester* [14]) est non nul. On peut écrire explicitement ce déterminant $R_{p,q}(P, Q)$ si

$$\begin{aligned} P(X) &= a_0X^p + \cdots + a_p,\ a_0 \neq 0 \\ Q(X) &= b_0X^q + \cdots + b_q,\ b_0 \neq 0 \end{aligned}$$

et l'on obtient :

$$R_{p,q}[P, Q] = \begin{vmatrix} a_0 & a_1 & \dots & a_p & 0 & \dots & 0 & 0 \\ 0 & a_0 & \dots & a_{p-1} & a_p & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \dots & \dots & \dots & \dots & a_p & 0 \\ 0 & 0 & \dots & \dots & \dots & \dots & a_{p-1} & a_p \\ b_0 & b_1 & \cdots & \cdots & \cdots & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & \cdots & \cdots & \cdots & \cdots & \cdots & b_{q-1} & b_q \end{vmatrix} \tag{3.14}$$

Voici un code pour générer sous `Maple` la matrice de Sylvester :

---

13. Voir le cours d'Algèbre 1 : si P et Q sont premiers entre eux dans $\mathbb{K}[X]$ et si P divise un produit du type $B(X) \times Q(X)$, alors P divise B.

14. James Sylvester (1814-1897), avocat, musicologue et mathématicien anglais, ami de Cayley (vous connaissez depuis le cours d'Algèbre 2 le célèbre théorème de Cayley-Hamilton) ; Sylvester fut l'un des pionniers de la théorie des déterminants ; c'est à lui d'ailleurs que l'on doit le qualificatif ≪ matrice ≫.

```
> with (PolynomialTools);
> with (LinearAlgebra)
> Sylvester:=proc(P,Q,X)
local V1,V2,M,D,i,ii;
V1:=CoefficientList(P,X,'termorder=reverse');
V2:=CoefficientList(Q,X,'termorder=reverse');
D:=nops(V1)+nops(V2)-2;
M:=Matrix(D);
for i from 1 to nops(V2)-1 do
ii:=i+nops(V1)-1;
M[i,i..ii]:=Matrix(V1);
end do;
for i from 1 to nops(V1)-1 do
ii:=i+nops(V2)-1;
M[nops(V2)-1+i,i..ii]:=Matrix(V2);
end do;
M;
end proc;
```

Les formules de Cramer (utilisées pour chercher les inconnues $u_0, ..., u_{q-1}, v_0, ..., v_{p-1}$ du système de Cramer correspondant à la formule (3.13) traduite sur les coefficients des puissances de X : $1, X, ..., X^{p+q-1}$) nous montrent d'ailleurs que $R_{p,q}[P,Q]$ s'exprime sous la forme

$$A_{p,q}(a_0, ..., a_p, b_0, ..., b_q; X)\, P(X) + B_{p,q}(a_0, ..., a_p, b_0, ..., b_q; X)\, Q(X),$$

où les expressions $A_{p,q}$ et $B_{p,q}$ sont à coefficients des expressions polynomiales (ne dépendant que de p et q) à coefficients entiers en les « entrées » $a_0, ..., a_p, b_0, ..., b_q$ correspondant aux coefficients des polynômes P et Q.

Cette dernière remarque a son importance : si P et Q sont par exemple des polynômes en deux variables (X, Y) à coefficients dans $\mathbb{K}$:

$$\begin{aligned} P(X,Y) &:= a_0(Y)X^p + \cdots + a_p(Y) \\ Q(X,Y) &:= b_0(Y)X^q + \cdots + b_q(Y) \end{aligned}$$

et que l'on forme le résultant de Sylvester $R_{p,q}[P,Q](Y)$ de ces deux polynômes considérés comme polynômes en la variable X, on obtient un polynôme $R(Y)$ tel que

$$\Big(P(X,Y) = 0 \text{ et } Q(X,Y) = 0\Big) \Longrightarrow R(Y) = 0\,,$$

ce qui permet (à condition toutefois que le polynôme $R_{p,q}[P,Q](Y)$ que l'on construit ne soit pas identiquement nul) d'« éliminer » la variable X dans le système de deux équations à deux inconnues (mais non linéaire !)

$$P(X,Y) = Q(X,Y) = 0\,.$$

Sous `Maple`, le calcul de résultant (par exemple ici, de deux expressions algébriques P et Q en deux variables x et y) se fait sous une routine du type :

```
> P := 3*x^2*y+y^4*x+5*y^2*x^3+8*x*y+2;
                          2      4        2  3
                P = 3 x  y + y  x + 5 y  x  + 8 x y + 2

> Q := 2*y^2*x^3+y^2*x^4-y*x^3+x*y+1;
```

```
                          2  3    2  4      3
                  Q = 2 y  x  + y  x  - y x  + x y + 1
> evala(Resultant(3*x^2*y+y^4*x+5*y^2*x^3+8*x*y+2,
2*y^2*x^3+y^2*x^4-y*x^3+x*y+1, x));

 20       18      17      16        15       14        13         12        11
y   + 20 y   + 4 y   + 8 y   + 340 y   - 71 y   + 142 y   + 1925 y   - 156 y
          10         9         8         7        6       5
   + 847 y   + 3494 y  + 1178 y  + 1280 y  - 328 y  + 64 y

> evala(Resultant(3*x^2*y+y^4*x+5*y^2*x^3+8*x*y+2,
2*y^2*x^3+y^2*x^4-y*x^3+x*y+1, y));

     17        16        15        14        13        12        11       10
 16 x   + 120 x   + 396 x   + 774 x   + 972 x   + 739 x   + 256 x   - 29 x
          9      8       7       6      5    2
    - 34 x  - 2 x  + 46 x  + 34 x  - 6 x  + x
```

La routine `Resultant` correspond à une commande inerte (il faut en effet, on le voit ci-dessus, la combiner avec l'évaluation de l'expression par `evala`). On peut aussi préférer la routine (non inerte cette fois) `resultant` :

```
>> Rx:=resultant(P,Q,x);
>> Ry:=resultant(P,Q,y);
```

qui fournit des sorties qui sont respectivement un polynôme en `y` (en ce qui concerne `Rx`) ou un polynôme en `x` (en ce qui concerne `Ry`). Les racines y_j (ou x_k) de ces polynômes dans le plan complexe se calculent numériquement *via* :

```
>> fsolve(Rx,y,complex);
>> fsolve(Ry,x,complex);
```

Parmi les points de coordonnées complexes (x_j, y_k) obtenus ainsi se trouvent les couples (x, y) solutions du système `P(x,y)=Q(x,y)=0` (mais on trouve en général beaucoup trop de couples possibles, il reste à essayer ceux qui marchent). Il faut savoir qu'un théorème profond d'Etienne Bézout (admis ici bien sûr) permet d'affirmer que le nombre total de solutions (si toutefois il n'y en a qu'un nombre fini) ne devrait pas excéder le produit des deux degrés totaux des polynômes `P` et `Q`.

Ce procédé se généralise : si par exemple $F_1, ..., F_M$ sont une liste d'expressions polynomiales en n symboles $X_1, ..., X_n$, alors, en formant le résultant de Sylvester de F_1 et $F_1 + u_2 F_2 + ... + u_M F_M$ (où $u_2, ..., u_M$ sont des paramètres), tous les deux considérés comme des polynômes en X_1 (on suppose que X_1 figure explicitement dans F_1), on obtient une expression polynomiale en $u_2, ..., u_M$ et $X_2, ..., X_n$; la liste des coefficients $G_1(X_2, ...X_n), ..., G_{M'}(X_2, ..., X_n)$ de cette expression (considérée comme polynôme en $u_2, ..., u_M$) fournit une liste d'expressions polynomiales en $X_2, ..., X_n$, toutes s'exprimant sous la forme

$$G_j(X_2, ..., X_n) = \sum_{k=1}^{M} A_{j,k}(X_1, ..., X_n) F_k(X_1, ..., X_n)\,,\ j = 1, ..., M'\,.$$

Si l'on s'intéresse à résoudre le système d'équations

$$F_1(X_1, ..., X_n) = \cdots = F_M(X_1, ..., X_n) = 0\,,$$

le système

$$G_1(X_2, ..., X_n) = \cdots = G_{M'}(X_2, ..., X_n) = 0$$

fournit un nouveau système où la variable X_1 a été « éliminée ». On peut ainsi continuer et c'est là la pierre d'angle de la *théorie de l'élimination* que l'on a fait qu'esquisser ici [15].

Si $\mathbb{K} = \mathbb{C}$, dire que $R_{p,q}[P, Q] = 0$ équivaut à dire que P et Q (de degrés exactement $p > 0$ et $q > 0$) ont une racine commune dans $\mathbb{C}$.

Si P est un polynôme de $\mathbb{C}[X]$ de degré exactement $p > 1$, le résultant $R_{p,p-1}[P, P']$ de P et P' est un déterminant de taille $2\deg P - 1$ tel que $R_{p,p-1}[P, P'] = 0$ équivaut (toujours si $a_0 \neq 0$) au fait que le polynôme

$$a_0 X^p + \cdots + a_p = 0$$

ait une racine double. On pourra par exemple calculer le résultant $R[P, P']$ si

$$P(X) = aX^2 + bX + c$$

(en fonction des paramètres a, b, c), ce qui donne

$$R_{2,1}(P, P') = \begin{vmatrix} a & b & c \\ 2a & b & 0 \\ 0 & 2a & b \end{vmatrix} = -ab^2 + 4a^2c = -a(b^2 - 4ac)\ ;$$

si

$$P(X) = X^3 + \beta X + \gamma\,,$$

(autre cas important), on trouve

$$R_{3,2}(P, P') = \begin{vmatrix} 1 & 0 & \beta & \gamma & 0 \\ 0 & 1 & 0 & \beta & \gamma \\ 3 & 0 & \beta & 0 & 0 \\ 0 & 3 & 0 & \beta & 0 \\ 0 & 0 & 3 & 0 & \beta \end{vmatrix} = 4\beta^3 + 27\gamma^2\,.$$

Le résultant $R_{d,d-1}(P, P')$ de P et P' (le degré de P étant précisé) est un déterminant $(2d-1) \times (2d-1)$ que l'on appelle le *discriminant* du polynôme P ; il s'exprime polynomialement en les coefficients de P et est nul si et seulement P a une racine multiple dans $\mathbb{C}$ (attention, seulement si le coefficient dominant a_0 de P est non nul, comme en témoigne l'exemple du polynôme $P(X) = aX^2 + bX + c$ dont le discriminant n'est pas $b^2 - 4ac$ mais $-a(b^2 - 4ac)$!).

Le principe de la théorie de l'élimination, outre le fait qu'il permette la résolution des systèmes d'équations algébriques à coefficients entiers (au moins numériquement, au travers de la recherche de l'approximation des zéros complexes du résultant par des méthodes itératives du type Newton, *cf.* le chapitre 2), est aussi la clef de voûte du calcul symbolique de primitives, par exemples de primitives de fractions

15. Il faut toutefois souligner que ce procédé devient très coûteux en temps et espace de calcul dès que le nombre de variables $X_1, ..., X_n$ augmente ! D'autres méthodes, inspirées de l'algorithme d'Euclide (la plus importante est celle qui passe par la construction de bases de Gröbner), ont été introduites depuis les années 1970 (à l'aube de la révolution numérique) pour gérer à moindre coût les systèmes d'équations en beaucoup de paramètres que fait par exemple surgir la robotique.

rationnelles de $\mathbb{Q}(X)$ dans des logiciels de calcul symbolique. En témoignent des méthodes telles celle initiée dans les années 1970 par Michael Rothstein et Barry Trager (voir les séances de TP).

CHAPITRE 4

Méthodes itératives en algèbre linéaire

4.1. Le théorème du point fixe dans $\mathbb{K}^N$ ($\mathbb{K} = \mathbb{R}$ ou $\mathbb{C}$)

4.1.1. Un théorème mathématique fondée sur une démarche algorithmique. On sait que, sur le $\mathbb{K}$-espace vectoriel $\mathbb{K}^N$, deux normes quelconques $|\ |$ et $\|\ \|$ sont toujours équivalentes, ce qui signifie quel existe deux constantes c et C, toutes les deux strictement positives (dépendant bien sûr du choix des deux normes) et telles que

$$c|X| \leq \|X\| \leq C\,|X| \quad \forall\, X \in \mathbb{K}^N. \tag{4.1}$$

Pour formuler le théorème sous-tendant tout ce chapitre, à savoir le *théorème du point fixe*, nous allons convenir pour l'instant du choix d'une norme sur $\mathbb{K}^N$ (nous la noterons $\|\ \|$). Ce choix n'affecte en rien l'énoncé que nous allons donner.

THEORÈME 4.1 (théorème du point fixe). *Soit T une application de $\mathbb{K}^N$ dans lui-même, contractant strictement les distances, c'est-à-dire telle qu'il existe une constante $\kappa \in [0, 1[$ telle que*

$$\forall X, Y \in \mathbb{K}^N\,, \|T(X) - T(Y)\| \leq \kappa \|X - Y\|. \tag{4.2}$$

L'application T a un unique point fixe (sous l'action de T) dans $\mathbb{K}^N$, et ce point s'atteint en partant de n'importe quel point X_0 de $\mathbb{K}^N$ comme la limite de la suite $(X_k)_{k\geq 0}$ définie récursivement par

$$X_k = T(X_{k-1})\,,\ \forall\, k \geq 1\,.$$

DÉMONSTRATION. C'est le fait que toute suite de Cauchy d'éléments de $\mathbb{K}^N$ soit convergente [1] qui soutend la preuve de ce résultat.

Que le point fixe soit unique est une évidence car s'il y avait deux ($T(Y_1) = Y_1$ et $T(Y_2) = Y_2$ avec $Y_1 \neq Y_2$), on aurait

$$\|T(Y_1) - T(Y_2)\| = \|Y_1 - Y_2\| \leq \kappa \|Y_1 - Y_2\|\,,$$

ce qui est impossible si $\kappa < 1$. Si k est un entier strictement positif,

$$X_{k+1} - X_k = T(X_k) - T(X_{k-1})\,,$$

donc

$$\|X_{k+1} - X_k\| \leq \kappa \|X_k - X_{k-1}\| \leq \cdots \leq \kappa^k \|X_1 - X_0\|\,.$$

1. C'est un résultat connu depuis le cours d'Analyse 1 (lorsque $N = 1$) ou d'Analyse 2 (lorsque $N > 1$) : toute suite de Cauchy de scalaires dans $\mathbb{K}$ est convergente dans $\mathbb{K}$; on rappelle qu'une suite $(X_k)_k$ de $\mathbb{K}^N$ est de Cauchy si et seulement si la distance entre deux points de la suite X_p et X_q peut être rendue arbitrairement petite pourvu que p et q soient assez grands. Pour passer du cas $N = 1$ au cas N quelconque, il convient de raisonner coordonnée par coordonnée.

Ainsi, si $p \geq 1$,

$$\|X_{k+p} - X_k\| \leq \sum_{l=k}^{k+p-1} \|X_{l+1} - X_l\|$$
$$\leq \kappa^k(1 + \kappa + \cdots + \kappa^{p-1})\|X_1 - X_0\| \leq \frac{\kappa^k}{1-\kappa}\|X_1 - X_0\|$$

puisque la série géométrique $\sum_{k\geq 0} \kappa^k$ est convergente (de somme $1/(1-\kappa)$). La suite $(X_k)_{k\geq 0}$ est donc bien de Cauchy, donc converge vers un point Y, l'erreur d'approximation de Y par X_k au cran k de l'itération étant majorée par

$$\|X_k - Y\| \leq \kappa^k \frac{\|X_1 - X_0\|}{1-\kappa}\,.$$

C'est une convergence très rapide (exponentielle puisque κ peut s'écrire $\kappa = e^{-u}$ avec $u > 0$). Comme T est continue (car contractant, même strictement, les distances), on déduit de $T(X_{k-1}) = X_k$ que $T(Y) = Y$, donc que Y est le point fixe de T. □

4.1.2. Un exemple frappant : `Pagerank`. Dans cette sous-section, on prend $\mathbb{K} = \mathbb{R}$. L'algorithme `Pagerank`, brique de base de la version primitive du moteur de recherche `Google` sur la toile, constitue aussi une illustration du champ applicatif qu'entr'ouvre le théorème du point fixe [2].

L'ensemble des sites `web` peut être considéré comme l'ensemble E des sommets d'un *graphe orienté*. Il s'agit d'un ensemble de cardinal gigantesque (on évoque une trentaine de milliards), mais fini, et dont on peut indéxer les éléments de 1 à N. Pour compléter la définition de graphe orienté, il faut ajouter la donnée d'un sous-ensemble fini V de $E \times E$, un couple de sommets (i, j) de $E \times E$ étant appelé une *arête* du graphe orienté (E, V) lorsque $(i, j) \in V$. Les arêtes du graphe orienté correspondant à la toile `internet` sont ici les liens $i \to j$ pointant d'une page i sur une page j ; $(i, j) \in V$ si et seulement si il existe un tel lien.

On associe à cette configuration une matrice $\mathbb{G}$ de taille $N \times N$ ($\mathbb{G}$ pour `Google`) définie comme suit : l'entrée g_{ij} de $\mathbb{G}$ (i indice de ligne, j indice de colonne) est nulle si et seulement si il n'existe aucun lien pointant de i vers j (*i.e.* $(i, j) \notin V$). S'il existe un lien de i vers j, on définit g_{ij} comme l'inverse $1/L_i$ du nombre de liens de la page i vers une autre page `web` (dont, bien sûr, la page j, puisque $(i, j) \in V$ dans ce cas). Cette matrice $\mathbb{G}$ a, pourvu que l'on fasse l'hypothèse que de toute page part toujours au moins un lien vers une autre page, l'intéressante propriété suivante : la somme des coefficients de chaque ligne vaut 1 ; de plus toutes les entrées de cette matrice sont positives ou nulles ; une telle matrice est dite *matrice stochastique* (on peut considérer chaque ligne comme une distribution de probabilité sur l'ensemble fini $\{1, ..., N\}$). On voit que 1 est valeur propre de cette matrice : en effet le vecteur colonne $\texttt{ones}(N, 1)$ dont toutes les coordonnées valent 1 est vecteur propre associé

2. On pourra aussi consulter avec profit (pour plus de détails) le texte rédigé par Michael Eisermann [**Eiser**] dont je me suis inspiré pour cette section (consulter aussi [**Eiser0**] pour une version plus ≪ piétonne ≫). On pourra aussi, pour des exemples plus élémentaires (au niveau de l'enseignement secondaire) se référer au document ressources [**TermS**] mis en ligne par le Ministère de l'Education Nationale à l'appui des nouveaux programmes de la spécialité en Terminale S, pages 15 à 26.

à la valeur propre 1. Si l'on choisit comme norme sur $\mathbb{R}^N$ la norme $\| \ \|_\infty$ (on la notera pour simplifier $\| \ \|$), on voit immédiatement que

$$\forall\, X \in \mathbb{R}^N,\ \|\mathbb{G} \cdot X\| \leq \|X\|,$$

donc, si l'on définit une norme (notée encore $\| \ \|$) sur les matrices $N \times N$ à entrées réelles par

$$\|A\| := \sup_{X \neq 0} \frac{\|A \cdot X\|_\infty}{\|X\|_\infty} = \sup_{X \neq 0} \frac{\|A \cdot X\|}{\|X\|},$$

on a, pour cette norme, $\|\mathbb{G}\| \leq 1$. De fait, on a même $\|\mathbb{G}\| = 1$ puisque 1 est valeur propre de $\mathbb{G}$.

Supposons un instant qu'aucune page du réseau ne soit une impasse, i.e. ne pointe sur aucune autre page. Si un robot dépourvu de la moindre capacité de discernement se déplace sur la toile (à partir d'un instant initial noté $k = 0$) et que l'on note $\mu_{k,j}$ la probabilité que notre robot se trouve sur la page j au clic k, on a

$$\mu_{k+1,i} = \sum_{j=1}^{N} \mathbb{P}\Big(\text{clic sur i} \,\Big|\, \text{le robot est en } j\Big)\, \mu_{k,j} \quad \forall\, i = 1, ..., N. \tag{4.3}$$

Comme le robot est idiot, il clique au hasard et la probabilité qu'il fasse son $(k+1)$-clic vers la page i depuis la page j (où il est arrivé après k clics) vaut $1/L_j = g_{ji}$. Les relations (4.3) se lisent matriciellement

$$[\mu_{k+1,1}, ..., \mu_{k+1,N}] = [\mu_{k,1}, ..., \mu_{k,N}] \cdot \mathbb{G}.$$

On peut certes corriger le fait que certaines pages puissent se révéler être des impasses en supposant que toute page pointe sur elle-même, ce que nous ferons. Mais afin de gommer les effets qu'entraine ce « palliatif », on suppose qu'avec une probabilité $1 - \kappa$ (on prend couramment .15), le robot, au moment de décider où aller au bout de k clics, décide d'aller avec la probabilité $1/N$ vers une page arbitraire du réseau. Ceci revient à modifier la matrice stochastique $\mathbb{G}$ en posant

$$\mathbb{G}_\kappa = \frac{1-\kappa}{N}\, \texttt{ones}(N, N) + \kappa\, \mathbb{G}\,,$$

où $\texttt{ones}(N, N)$ est la matrice $N \times N$ dont les entrées sont toutes des 1. Les relations matricielles deviennent

$$[\mu_{k+1,1}, ..., \mu_{k+1,N}] = [\mu_{k,1}, ..., \mu_{k,N}] \cdot \mathbb{G}_\kappa.$$

Une « mesure d'équilibre » $[\mu_1, ..., \mu_N]$ (au seuil de tolérance $1-\kappa$) est par définition une distribution de probabilité sur $\{1, ..., N\}$ telle que

$$[\mu_1, ..., \mu_N] = [\mu_1, ..., \mu_N] \cdot \mathbb{G}_\kappa.$$

En fait, on voit que ceci est équivalent à dire que $[\mu_1, ..., \mu_N]$ (traité comme vecteur ligne) est un point fixe de l'application affine T_κ de $\mathbb{R}^N$ dans $\mathbb{R}^N$ (les vecteurs étant ici traités en ligne) définie par

$$T_\kappa \ : [x_1, ..., x_N] \mapsto \frac{1-\kappa}{N}\, \texttt{ones}(1, N) + \kappa\, [x_1, ..., x_N] \cdot \mathbb{G}.$$

Comme le rayon spectral de $\mathbb{G}$ vaut 1, cette application T_κ est $\kappa \simeq .85 < 1$- contractante et le théorème du point fixe assure l'existence et l'unicité de la « mesure

d'équililibre » (au seuil de tolérance $1-\kappa$), en même temps que la possibilité d'approcher asymptotiquement (avec une erreur décroissant exponentiellement) cette « mesure d'équilibre » en partant d'une distribution de probabilité arbitraire

$$[\mu_{\text{init},1}, ..., \mu_{\text{init},N}] = \mu_{\text{init}}$$

(correspondant à $k = 0$) et en considérant la démarche itérative

$$[\mu_{k+1,1}, ..., \mu_{k+1,N}] = T_\kappa([\mu_{k,1}, ..., \mu_{k,N}]), \quad k = 0, 1, 2, ...$$

Cette mesure d'équilibre (accessible *via* l'algorithme du point fixe) traduit les « poids » relatifs des diverses pages `web` sur la toile.

Évidemment, la matrice $\mathbb{G}$ est de taille colossale et demande à être réactualisée en permanence. D'où la difficulté de la maintenir stable le temps d'un calcul lourd (de par la complexité inhérente à la taille des matrices en jeu), donc forcément consommateur en temps ! Ce qui nous sauve cependant est l'aspect « creux » de la matrice (une page `web` pointe en moyenne seulement vers une dizaine de pages sur la toile, mais il faut encore beaucoup travailler pour passer du théorique à l'opérationnel [3]). Ce que nous venons de présenter correspond à une version primitive de l'algorithme `Pagerank`, pièce maitresse du dispositif du moteur de recherche `Google` élaboré vers 1997 par Serguey Brin et Larry Page à l'université de Stanford. On dispose ainsi d'une formidable application (combien actuelle !) du théorème du point fixe « en situation ».

4.2. Quelques notions préalables à l'algorithmique matricielle

4.2.1. Normes sur $\mathbb{K}^N$ et normes d'applications $\mathbb{K}$-linéaires et de matrices. Le choix d'une norme sur $\mathbb{K}^N$ ($\mathbb{K} = \mathbb{R}$ ou $\mathbb{C}$) a pu jusque là paraitre indifférent. Ceci n'est toutefois pas complètement vrai, comme nous allons le voir (dans la présentation de `Pagerank` à la section 4.1.2, nous avons, notons le, fait le choix de la norme $\| \ \|_\infty$ sur $\mathbb{R}^N$). Changer de norme sur $\mathbb{K}^N$ affecte en effet la manière de quantifier la « taille » des vecteurs, et par voie de conséquence, des matrices. Les normes les plus importantes sont :

- la norme euclidienne $\| \ \|_2$

$$\|(x_1, ..., x_N)\|_2 := (x_1^2 + \cdots + x_N^2)^{1/2},$$

  importante car liée au produit scalaire dans $\mathbb{K}^N$, outil nous permettant d'y faire de la géométrie « la Pythagore », nous y reviendrons ; c'est aussi la norme par défaut `norm` de la plupart des logiciels de calcul scientifique (`MATLAB`, `Scilab`) ;
- la norme $\| \ \|_\infty$ dfinie par

$$\|(x_1, ..., x_N)\|_\infty := \sup_{j=1,...,N} |x_j| ;$$

- la norme $\| \ \|_1$ définie par

$$\|(x_1, ..., x_N)\|_1 := \sum_{j=1}^{N} |x_j| ;$$

3. Voir par exemple [**Eiser**].

– plus généralement, si p désigne un nombre réel supérieur [4] ou égal 1, la norme $\| \ \|_p$ définie par [5]

$$\|X\|_p := \Big(\sum_{j=1}^{N} |x_j|^p \Big)^{1/p} .$$

Le choix d'une norme $\| \ \|$ sur $\mathbb{K}^N$ ($\mathbb{K} = \mathbb{R}$ ou $\mathbb{C}$) induit le choix d'une norme sur les applications linaires $L \ : \mathbb{K}^N \longmapsto \mathbb{K}^N$ par

$$\|L\| := \sup_{X \in (\mathbb{K}^N)^*} \frac{\|L(X)\|}{\|X\|} .$$

On peut aussi envisager le point de vue matriciel et définir la norme d'une matrice $N \times N$ A comme la norme de l'application linéaire que cette matrice représente. Si A est une matrice $N \times N$ (à coefficients dans $\mathbb{K}$), on a donc

$$\|A\| := \sup_{X \in (\mathbb{K}^N)^*} \frac{\|A \cdot X\|}{\|X\|} .$$

La norme de toute matrice représentant la même application linéaire que celle que représente A est donc égale la norme de A. Bien sûr, cette définition de la *norme d'une matrice* (ou d'une application linéaire), dite aussi *norme matricielle* dépend du choix de la norme qui a été fait dans $\mathbb{K}^N$. On peut faire le choix de la norme euclidienne dans $\mathbb{K}^N$, mais on peut tout aussi bien faire un autre choix : par exemple, si l'on choisit la norme $\| \ \| = \| \ \|_\infty$ sur $\mathbb{K}^N$, la norme $\|A\|_\infty$ correspondante d'une matrice $A = [a_{i,j}]$ de taille (N, N) à coefficients dans $\mathbb{K}$ est

$$\|A\|_\infty = \|[a_{i,j}]\|_\infty = \sup_i \sum_{j=1}^{N} |a_{i,j}|$$

(i indice de ligne, j indice de colonne) ; si par contre, on prend comme norme sur $\mathbb{K}^N$ la norme $\| \ \| = \| \ \|_1$, la norme $\|A\|_1$ correspondante d'une matric A de taille (N, N) à coefficients dans $\mathbb{K}$ est

$$\|A\|_1 = \|[a_{i,j}]\|_1 = \sup_j \sum_{i=1}^{N} |a_{i,j}| . \tag{4.4}$$

C'est donc très différent ! Si l'on fait le choix de la norme euclidienne sur $\mathbb{K}^N$, la norme d'une matrice réelle A de taille (N, N) n'est d'ailleurs pas si simple à exprimer. Il faut introduire les valeurs propres (réelles positives) de ${}^t\overline{A} \cdot A$ (symétrique si $\mathbb{K} = \mathbb{R}$, hermitienne si $\mathbb{K} = \mathbb{C}$) et prendre le maximum des racines carrées positives de ces valeurs propres [6] :

$$\|A\|_2 = \sup\{|\lambda| \ ; \ \lambda^2 \text{ valeur propre de } {}^t\overline{A} \cdot A\}.$$

On définit de même la norme d'une application linéaire de $\mathbb{K}^p$ dans $\mathbb{K}^q$ en équipant en général ces deux espaces de la même norme $\| \ \|$ l'arrivée et au départ (ceci n'est

4. Pour $p \in]0, 1[$, l'inégalité triangulaire est en défaut, et ce n'est donc plus une norme.

5. Qu'il s'agisse d'une norme est ici moins immédiat ; l'inégalité triangulaire est une célèbre inégalité due au mathématicien russe, géomètre tant des espaces que des nombres, Hermann Minkowski (1864-1909).

6. Ces racines carrées positives sont appelées *valeurs singulières* de la matrice A, on y reviendra.

cependant en rien une obligation). Ce que nous avons dit pour les matrices carrées vaut donc aussi pour les matrices rectangulaires.

Étant données deux matrices carrées et un choix de norme dans $\mathbb{K}^N$, on a toujours :

$$\|A \cdot B\| \leq \|A\| \times \|B\|$$

(ce pour le choix de la norme dérivé du choix de norme qui a été fait dans $\mathbb{K}^N$, disons par exemple ici la norme euclidienne, mais on peut très bien avoir fait un autre choix).

4.2.2. Rayon spectral d'une matrice carrée. La notion de *rayon spectral* d'une matrice carrée (réelle ou complexe) s'avère une première notion très importante du pont de vue de l'analyse numérique et du calcul matriciel appliquée. Cette notion conditionne en général la justification de l'utilisation du théorème du point fixe.

Soit A une matrice $N \times N$ à coefficients réels ou complexes.

DÉFINITION 4.1 (rayon spectral). On appelle rayon spectral de la matrice A le maximum des modules des racines du polynôme caractéristique $\det[A - XI_N]$ dans le corps algébriquement clos $\mathbb{C}$.

La proposition suivante sera pour nous capitale :

PROPOSITION 4.1. *Si A est une matrice (N, N) coefficients complexes de rayon spectral $\rho(A)$ et si $\| \ \|$ est une norme arbitraire sur $\mathbb{K}^N$, ($\mathbb{K} = \mathbb{R}$ ou $\mathbb{C}$), induisant donc une norme matricielle sur le $\mathbb{K}$-espace des matrices (N, N), on a toujours l'inégalité :*

$$\rho(A) \leq \|A\|. \tag{4.5}$$

D'autre part, pour tout $\epsilon > 0$, il existe toujours une norme $\| \ \|^{(\epsilon)}$ convenable sur $\mathbb{K}^n$ telle que l'on ait, pour la norme matricielle induite,

$$\|A\|^{(\epsilon)} \leq \rho(A) + \epsilon.$$

DÉMONSTRATION. Prouvons d'abord la première assertion. Prenons une valeur propre (par exemple λ_1) de module maximum et V un vecteur propre associé. On a

$$\|A \cdot V\| = \|\lambda_1 V\| = |\lambda_1| \times \|V\| \leq \|A\| \, \|V\|$$

par définition de la norme de A :

$$\|A\| = \sup_{X \in \mathbb{K}^N \setminus \{0\}} \frac{\|A \cdot X\|}{\|X\|}.$$

En divisant par $\|V\| \neq 0$, on trouve bien $\rho(A) \leq \|A\|$.

Pour la seconde assertion, on peut supposer (qui peut le plus peut le moins) que $\mathbb{K} = \mathbb{C}$. On sait que la matrice A peut s'exprimer sous la forme $A = {}^t\overline{U} \cdot T \cdot U$, où T est une matrice triangulaire supérieure et U une matrice unitaire [7] . Il est en effet possible de coupler l'algorithme de trigonalisation des matrices carrées à coefficients complexes avec l'algorithme de Gram-Schmidt (et donc faire en sorte que la matrice de passage P dans $A = P \cdot T \cdot P^{-1}$ soit unitaire). On remarque que A et T ont même rayon spectral (car si V est vecteur propre de A, ${}^t\overline{U} \cdot V$ est vecteur propre de T

7. C'est-à-dire, telle que les vecteurs colonnes forment une base orthonormée de $\mathbb{K}^N$ pour le produit scalaire usuel, soit ${}^t\overline{U} \cdot U = U \cdot {}^t\overline{U} = I_N$.

avec la même valeur propre. Il suffit donc de prouver la seconde assertion pour une matrice triangulaire supérieure T. Fixons C assez grand et introduisons la matrice diagonale $D_C = \mathrm{diag}[1, C, C^2, ..., C^{N-1}]$. On remarque que l'on a

$$(D_C \cdot T \cdot D_C^{-1})_{i,j} = \begin{cases} t_{i,j} C^{j-i} & \text{si} \quad j \leq i \\ 0 \quad \text{si} & j > i. \end{cases}$$

Il suffit donc de choisir la norme $\| \ \|^{(\epsilon)}$ sur $\mathbb{K}^N$ de manière à ce que la norme matricielle associée soit la norme définie sur le $\mathbb{K}$-espace des matrices (N, N) par

$$\|M\|^{(\epsilon)} = \|D_C^{-1} \cdot M \cdot D_C\|_\infty,$$

la constante strictement positive C étant choisie suffisamment grande. □

Remarque 4.1 (le cas des matrices diagonalisables sur $\mathbb{C}$). Dans le cas où A est diagonalisable sur $\mathbb{C}$, il existe une base $(V_1, ..., V_N)$ de vecteurs propres dans $\mathbb{C}^N$. En prenant comme norme d'un vecteur $V \in \mathbb{K}^N \subset \mathbb{C}^N$ le maximum des modules des coordonnées de $V \in \mathbb{C}^N$ dans cette base $(V_1, ..., V_N)$, un induit sur le $\mathbb{K}$-espace des matrices (N, N) une norme matricielle telle que l'on ait exactement $\|A\| = \rho(A)$.

Le point important concernant le rayon spectral d'une matrice carrée est qu'il peut être calculé algorithmiquement, par une méthode itérative que nous décrivons ici.

Proposition 4.2 (algorithme itératif pour le calcul approché du rayon spectral). *Soit A une matrice à coefficients dans $\mathbb{K}$ ($\mathbb{K} = \mathbb{R}$ ou $\mathbb{K} = \mathbb{C}$), diagonalisable*[8] *sur $\mathbb{C}$, telle que les valeurs propres (distinctes ou confondues) aient des modules s'organisant comme suit*[9]

$$|\lambda_1| > |\lambda_{\mu+1}| \geq |\lambda_{\mu+2}| \geq ... \geq |\lambda_N| \geq 0$$

(donc $\lambda_1 = \lambda_2 = \cdots = \lambda_\mu$). Soit $(V_1, ..., V_N)$ une base de $\mathbb{C}^N$ constituée de vecteurs propres pour A telle que $V_1, ..., V_\mu$ soit une base du sous espace propre associé λ_1. Soit X_0 un vecteur

$$X_0 = x_1 V_1 + \cdots + x_N V_N \,,$$

où l'un au moins des x_k, $k = 1, ..., \mu$, est non nul[10]*. Sous ces hypothèses, l'algorithme itératif initié à X_0 et régi ensuite par*

$$X_{k+1} = \frac{A \cdot X_k}{\|A \cdot X_k\|} \,, \ k \geq 0$$

(une norme sur $\mathbb{K}^N$ ayant été arbitrairement choisie) est tel que

$$\lim_{k \to +\infty} \|A \cdot X_k\| = |\lambda_1|,$$

et fournit donc un moyen numérique d'approcher le rayon spectral de A. La vitesse de convergence est de plus exponentielle.

---

8. C'est le cas, avec une probabilité 1, pour une matrice dont les coefficients sont pris au hasard (suivant une loi uniforme) dans $\mathbb{K}$.

9. Il n'y a donc qu'une seule valeur propre de plus grand module, c'est le cas par exemple pour une matrice dont toutes les entrées sont positives, ou une matrice dont toutes les valeurs propres sont réelles (par exemple une matrice symétrique réelle ou une matrice hermitienne).

10. C'est encore le cas d'un X_0 pris au hasard (de manière uniforme) dans $\mathbb{K}^N$, ce avec une probabilité égale à 1.

REMARQUE 4.2. Le choix d'une norme doit être fait au préalable ; sous MATLAB, la norme que l'on choisit en priorité est la norme euclidienne (ou $\| \ \|_2$), mais l'on pourrait prendre aussi n'importe laquelle des normes $\| \ \|_p$, $p \in [1, \infty]$. Voici (sous MATLAB) la routine algorithmique exprimée dans l'énoncé (ici avec la norme euclidienne) :

```
function r=rayonspectral1(A,X,k)

x=X;
for i=1:k
    y=A*x;
    x=y/norm(y);
end
   r=norm(y);
```

DÉMONSTRATION. Montrons d'abord par récurrence sur k que, pour tout entier $k \geq 1$,

$$X_k = \frac{A^k \cdot X_0}{\|A^k \cdot X_0\|} .$$

Ceci est vrai pour $k = 1$ par définition de X_1 et on a, pour $k \geq 1$,

$$X_{k+1} = \frac{A \cdot X_k}{\|A \cdot X_k\|} = A \cdot \Big(\frac{A^k \cdot X_0}{\|A^k \cdot X_0\|}\Big) \times \Big(\frac{\|A^{k+1} \cdot X_0\|}{\|A^k \cdot X_0\|}\Big)^{-1} = \frac{A^{k+1} \cdot X_0}{\|A^{k+1} \cdot X_0\|} ,$$

ce qui prouve le résultat au cran $k+1$. Or

$$\begin{aligned} A^k \cdot X_0 &= \sum_{j=1}^{N} \lambda_j^k x_j V_j = \lambda_1^k \Big[\Big(\sum_{j=1}^{\mu} x_j V_j \Big) + \sum_{j=\mu+1}^{N} \Big(\frac{\lambda_j}{\lambda_1}\Big)^k x_j V_j \Big] \\ &= \lambda_1^k \Big(\sum_{j=1}^{\mu} x_j V_j + \vec{\epsilon}_k \Big) \end{aligned}$$

avec

$$\|\vec{\epsilon}_k\| < \|X_0\| (|\lambda_{\mu+1}|/|\lambda_1|)^k .$$

On a donc

$$|\lambda_1|^k \Big\| \sum_{j=1}^{\mu} x_j V_j \Big\| (1 - \eta_k) \leq \|A^k \cdot X_0\| \leq |\lambda_1|^k \Big\| \sum_{j=1}^{\mu} x_j V_j \Big\| (1 + \eta_k) \tag{4.6}$$

avec

$$|\eta_k| < (|\lambda_{\mu+1}|/|\lambda_1|)^k \frac{\|X_0\|}{\Big\| \sum\limits_{j=1}^{\mu} x_j V_j \Big\|} .$$

On a d'autre part

$$A \cdot X_k = \frac{\lambda_1^k (\lambda_1 \sum\limits_{j=1}^{\mu} x_j V_j + A \cdot \vec{\epsilon}_k)}{\|A^k \cdot X_0\|}$$

et, en prenant les normes

$$\frac{|\lambda_1|^{k+1} \Big\| \sum\limits_{j=1}^{\mu} x_j e_j \Big\| (1 - \tilde{\eta}_k)}{\|A^k \cdot X_0\|} \leq \|A \cdot X_k\| \leq \frac{|\lambda_1|^{k+1} \Big\| \sum\limits_{j=1}^{\mu} x_j \cdot V_j \Big\| (1 + \tilde{\eta}_k)}{\|A^k \cdot X_0\|} \tag{4.7}$$

avec

$$\tilde{\eta}_k \leq (|\lambda_{\mu+1}|/|\lambda_1|)^k \|A\| \frac{\|X_0\|}{|\lambda_1| \left\| \sum_{j=1}^{\mu} x_j V_j \right\|}.$$

On achève donc la preuve en combinant les encadrements (4.6) et (4.7). Comme $(|\lambda_{\mu+1}|/|\lambda_1|)^k$ (qui gouverne la décroissance vers 0 de η_k et $\tilde{\eta}_k$ est en $e^{-\rho k}$ avec $\rho > 0$ (puisque $|\lambda_{\mu+1}| < |\lambda_1|$), on a bien une vitesse exponentielle de convergence de $\|A \cdot X_k\|$ vers $|\lambda_1|$. □

4.3. Les algorithmes itératifs pour la résolution de $M \cdot X = B$

4.3.1. Comment le théorème du point fixe entre en action. Pour résoudre un système du type $M \cdot X = B$ (que l'on suppose dans un premier temps ici carré, de taille (N, N), avec M de plus inversible [11]), il n'est pas question d'utiliser la formule $X = M^{-1} \cdot X$, car le calcul de M^{-1} (par exemple comme la matrice des cofacteurs divisée par le déterminant) induit des calculs de déterminants en général impossibles (calculs trop coûteux en temps !) lorsque N est très grand (sauf si l'on a de la chance et que la matrice M à le bon goût d'être « creuse », c'est-à-dire de contenir beaucoup d'entrées nulles).

Deux pistes s'offrent à nous ici pour parer à ce problème :

- celle consistant à attaquer le problème par une méthode dite *méthode directe*, telle par exemple que la méthode algorithmique du *pivot de Gauss*, vue dans le cours d'Algèbre 1 en L1 ;
- celle consistant à utiliser une démarche itérative (*méthode itérative*) inspirée par la démarche algorithmique fondant le théorème du point fixe (Théorème 4.1).

On choisit ici de s'intéresser au second angle d'attaque.

Le théorème du point fixe implique (Théorème 4.1) implique le résultat suivant :

PROPOSITION 4.3 (résolution itérative de $M \cdot X = B$). *Soit M une matrice (N, N) à coefficients dans $\mathbb{K}$ ($\mathbb{K} = \mathbb{R}$ ou $\mathbb{C}$) s'écrivant sous la forme $M = M_1 - M_2$, où M_1 et M_2 sont deux matrices (N, N) à coeficients dans $\mathbb{K}$ telles que M_1 soit inversible et que $\rho(M_1^{-1} M_2) < 1$. Alors, la suite $(X_k)_{k \geq 0}$ initiée en $X_0 \in \mathbb{K}^N$ (arbitraire) et régie ensuite par l'algorithme itératif*

$$X_{k+1} = M_1^{-1} \cdot M_2 \cdot X_k + M_1^{-1} \cdot B \quad \forall\, k \geq 0 \tag{4.8}$$

converge, lorsque k tend vers l'infini, vers la solution $\mathbf{X}$ du système de Cramer $M \cdot X = B$. De plus, si l'on choisit une norme $\| \ \|$ quelconque sur $\mathbb{K}^N$, la convergence de $\|X_k - \mathbf{X}\|$ vers 0 est exponentielle.

DÉMONSTRATION. On remarque que

$$M \cdot X = B \Longleftrightarrow M_1 \cdot X = M_2 \cdot X + B \Longleftrightarrow X = M_1^{-1} \cdot M_2 \cdot X + M_1^{-1} \cdot B.$$

Choisissons (ce qui est possible grâce à la Proposition 4.1) une norme sur $\mathbb{K}^N$ telle que, pour la norme matricielle induite :

$$\|M^{-1} \cdot M_2\| \leq \rho(M_1^{-1} \cdot M_2) + \epsilon < 1$$

(on prend $\epsilon > 0$ assez petit et $\| \ \| = \| \ \|^{(\epsilon)}$). L'application

$$X \in \mathbb{K}^N \longmapsto M_1^{-1} \cdot M_2 \cdot X + M_1^{-1} \cdot B$$

11. Il s'agit donc d'un système de Cramer.

est alors strictement contractante (on prend $\kappa = \rho(M_1^{-1} \cdot M_2) + \epsilon$) avec ce choix de norme. Le Théorème 4.1 s'applique et on obtient les conclusions voulues (toutes les normes sur $\mathbb{K}^N$ étant équivalentes, avoir la dernière assertion pour la norme sur $\mathbb{K}^N$ que l'on vient d'utiliser équivaut en effet à l'avoir pour n'importe quelle norme). □

On peut également envisager comme $\mathbb{K}$-espace vectoriel (à la place de $\mathbb{K}^N$) le $\mathbb{K}$-espace vectoriel $\mathcal{M}_{\mathbb{K}}(N,N)$ des matrices (N,N) à coefficients dans $\mathbb{K}$ et déterminer l'inverse de la matrice M en générant la suite $(U_k)_{k\geq 0}$ de matrices (N,N) initiée en une matrice (N,N) arbitraire M_0 (par exemple la matrice `zeros(M,M)`) et régie par l'algorithme itératif

$$U_{k+1} = M_1^{-1} \cdot M_2 \cdot U_k + M_1^{-1} \qquad \forall\, k \geq 0.$$

En effet, l'application linéaire

$$U \in \mathcal{M}_{\mathbb{K}}(N,N) \longmapsto M_1^{-1} \cdot M_2 \cdot U + M_1^{-1} \in \mathcal{M}_{\mathbb{K}}(N,N)$$

est encore strictement contractante (pour le même choix de norme matricielle que dans la preuve de la Proposition 4.3) et converge donc vers l'unique solution dans $\mathcal{M}_{\mathbb{K}}(N,N)$ de

$$U = M_1^{-1} \cdot M_2 \cdot U + M_1^{-1}.$$

Or, on remarque que

$$U = M_1^{-1} \cdot M_2 \cdot U + M_1^{-1} \Longleftrightarrow M_1 \cdot U = M_2 \cdot U + I_N \Longleftrightarrow M \cdot U = I_N \Longleftrightarrow U = M^{-1}.$$

La suite $(U_k)_{k\geq 0}$ ainsi construite converge donc bien vers l'inverse M^{-1} de M.

4.3.2. Les algorithmes de Jacobi et Gauß-Seidel. Lorsque M est une matrice dont les coefficients diagonaux m_{ii} sont tous non nuls, les deux matrices $D := \mathrm{diag}[m_{11}, ..., m_{NN}]$ ou $T_{\mathrm{inf}}[M]$ (matrice triangulaire inférieure où l'on a gardé les coefficients m_{ij} tels que $i \geq j$ et mis à zéro tous les autres) sont deux matrices aisément inversibles (l'une est diagonale, l'autre est triangulaire inférieure, tous les coefficients diagonaux étant dans les deux cas non nuls). On dispose donc des deux écritures possibles

$$M = D - (D - M) \quad \text{ou} \quad M = T_{\mathrm{inf}}[M] - (T_{\mathrm{inf}}[M] - M)$$

comme écritures candidates pour $M = M_1 - M_2$, avec M_1 aisément inversible. Ces deux décompositions s'écrivent sous `MATLAB` respectivement comme :

```
>> M = diag(diag(M)) - (diag(diag(M))-M);
>> M = tril(M) - (tril(M) -M);
```

(`tril` pour `triangular-lower`). La première

$$M = D - (D - M) = D - E \tag{4.9}$$

soutend l'*algorithme itératif de Jacobi*[12], tandis que la seconde

$$M = T_{\mathrm{inf}}[M] - (T_{\mathrm{inf}}[M] - M) = T_{\mathrm{inf}}[M] - F \tag{4.10}$$

12. Algébriste et géomètre allemand Carl Gustav Jacobi (1804-1851) marqua l'essor des mathématiques au XIX-ème siècle (déterminants, fonctions elliptiques, algèbre linéaire, problèmes géométriques d'intersection,...).

soutend l'*algorithme itératif de Gauß*[13]*-Seidel*[14].

EXEMPLE 4.1. La décomposition

$$\begin{pmatrix} 5 & 2 & 2 \\ 1 & 6 & 3 \\ 3 & 4 & -8 \end{pmatrix} = \begin{pmatrix} 5 & 0 & 0 \\ 0 & 6 & 0 \\ 0 & 0 & -8 \end{pmatrix} - \begin{pmatrix} 0 & -2 & -2 \\ -1 & 0 & -3 \\ -3 & -4 & 0 \end{pmatrix}$$

illustre (sur un exemple $(3,3)$) l'approche préparatoire de Jacobi, tandis que la décomposition

$$\begin{pmatrix} 5 & 2 & 2 \\ 1 & 6 & 3 \\ 3 & 4 & -8 \end{pmatrix} = \begin{pmatrix} 5 & 0 & 0 \\ 1 & 6 & 0 \\ 3 & 4 & -8 \end{pmatrix} - \begin{pmatrix} 0 & -2 & -2 \\ 0 & 0 & -3 \\ 0 & 0 & 0 \end{pmatrix}$$

illustre (sur le même exemple) celle de Gauß-Seidel.

La procédure de Jacobi s'implémente ainsi sous `MATLAB` :

```
function XX=Jacobi(M,B,X,k);
D=diag(diag(M));
E=D-M;
XX=X;
for i=1:k
    XX=D^(-1)*E*XX+D^(-1)*B;
end
```

Ici `M` désigne une matrice carrée de taille `(N,N)` (réelle ou complexe) inversible dont les termes diagonaux sont tous non nuls, `B` un vecteur colonne de l'espace $\mathbb{K}^N$ dans lequel on travaille, `X` le vecteur initial de la procédure (on peut prendre `X=zeros(N,1)` par exemple) et `k` le nombre d'itérations de la boucle. Avec les mêmes `input`, la procédure de Gauß-Seidel s'écrit

```
function XX=GaussSeidel(M,B,X,k);
T=tril(M);
F=tril(M)-M;
XX=X;
for i=1:k
    XX=T^(-1)*F*XX+T^(-1)*B;
end
```

Dans les deux cas, la sortie est (si l'algorithme converge, ce que l'on va discuter ensuite) le vecteur colonne candidat à être une valeur approchée de la solution du système de Cramer `M*XX=B`. On verra en TP comment mettre des tests d'arrêt dans de telles procédures.

DÉFINITION 4.2 (matrice à diagonale dominante). Soit $\mathbb{K} = \mathbb{R}$ ou $\mathbb{C}$. Une matrice $M = [m_{i,j}]_{1 \leq i,j \leq N}$ (i indice de ligne, j indice de colonne), à coefficients

13. On retrouve ici le mathématicien, astronome et philosophe allemand Carl Friedrich Gauß (1777-1855), certainement l'un de ceux qui ont le plus contribué guider l'évolution des mathématiques tous domaines confondus (algèbre, analyse, théorie des nombres et géométrie).

14. Au nom de Gauß, se trouve ajouté celui Philipp Ludwig von Seidel (1821-1896), élève de Jacobi, astronome, géomètre et probabiliste allemand.

dans $\mathbb{K}$, est dite *à diagonale dominante* si et seulement si

$$\forall\, i = 1, ..., N \quad |m_{ii}| > \sum_{\substack{j=1 \\ j\neq i}}^{N} |m_{ij}|, \tag{4.11}$$

autrement dit, compte-tenu de (4.4) et si $M = D - E$ est la décomposition (4.9) de Jacobi, si et seulement si D est inversible et $\|D^{-1} \cdot E\|_\infty < 1$.

Exemple 4.2. Les matrices

$$\begin{pmatrix} 5 & 2 & 2 \\ 1 & 6 & 3 \\ 3 & 4 & -8 \end{pmatrix} \qquad \begin{pmatrix} 5+2i & 3 & 2 \\ 1 & 3(1+i) & 3 \\ 3+i & 4 & -8-i \end{pmatrix}$$

sont à diagonale dominante (la première en tant que matrice à entrées réelles, la seconde en tant que matrice à entrées complexes). Les matrices

$$\begin{pmatrix} 5 & 2 & 2 \\ 1 & 6 & 3 \\ -5 & 4 & -8 \end{pmatrix} \qquad \begin{pmatrix} 5+2i & 3 & 2 \\ 1 & 3(1+i) & 3 \\ 3-7i & 4 & -8-i \end{pmatrix}$$

ne le sont pas (en effet, la dernière ligne pose dans les deux cas problème).

Si M est à diagonale dominante, les deux décompositions (4.9) et (4.10) sont du type $M = M_1 - M_2$, où M_1 et M_2 satisfont aux hypothèses de la Proposition 4.3. De plus, dans les deux cas, la matrice M_1 s'inverse très facilement car elle est soit diagonale, soit triangulaire inférieure, avec dans les deux cas tous ses termes diagonaux non nuls. On a en effet la :

Proposition 4.4 (validation des algorithmes de Jacobi ou de Gauß-Seidel pour les matrices à diagonale dominante). *Soit M une matrice à coefficients réels ou complexes et à diagonale dominante. Alors, on a $\rho(D^{-1}\cdot E) < 1$ et $\rho(T_{\rm inf}[M]^{-1}\cdot F)$ si $M = D - E = T_{\rm inf}[M] - F$ sont les décompositions* (4.9) *et* (4.10) *respectivement de Jacobi et de Gauß-Seidel. Dans les deux cas, la suite $(X_k)_{k\geq 0}$ initiée en un point X_0 quelconque de $\mathbb{K}^N$ et générée ensuite par l'algorithme itératif* (4.8) *converge vers la solution du système de Cramer $M \cdot X = B$.*

Démonstration. On voit immédiatement que $\|D^{-1}E\|_\infty < 1$ car cela résulte immédiatement de la définition (4.2). On utilise ensuite l'inégalité (4.5) qui assure que $\rho(D^{-1}\cdot E) \leq \|D^{-1}\cdot E\|_\infty < 1$. Concernant $T_{\rm inf}^{-1}\cdot F$, on montre directement que toute valeur propre λ (complexe) de cette matrice est de module strictement inférieur à 1 : soit λ une telle valeur propre (complexe) et V (de coordonnées $x_1, ..., x_N$ dans la base canonique de $\mathbb{C}^N$) un vecteur propre associé ; on a

$$\begin{aligned} & T_{\rm inf}[M]^{-1}\cdot F\cdot V = \lambda V \Longleftrightarrow F\cdot V = T_{\rm inf}[M]\cdot V \\ \Longleftrightarrow & \begin{cases} -\sum_{j=i+1}^{N} m_{i,j}\, x_j = \lambda \sum_{j=1}^{i} m_{i,j}\, x_j & \forall\, i = 1, ..., N-1 \\ 0 = \lambda \sum_{j=1}^{N} m_{N,j}\, x_j\,. \end{cases} \end{aligned} \tag{4.12}$$

Supposons $|\lambda| \geq 1$. Soit i_0 tel que $|x_{i_0}| = \sup_{1\leq i\leq N} |x_i| > 0$. De deux choses l'une.

– Soit $i_0 \in \{1, ..., N-1\}$, et l'on a alors, d'après (4.12),

$$\lambda\, m_{i_0,i_0}\, x_{i_0} = -\sum_{j=i_0+1}^{N} m_{i_0,j}\, x_j - \lambda \sum_{j<i_0} m_{i_0,j}\, x_j,$$

ce qui implique d'après l'inégalité triangulaire, le fait que $|\lambda| \geq 1$ et que $|x_{i_0}| \geq |x_j|$ quelque soit j,

$$|\lambda||m_{i_0,i_0}||x_{i_0}| \leq \sum_{j>i_0} |m_{i_0,j}||x_j| + |\lambda| \sum_{j<i_0} |m_{i_0,j}||x_j| \leq |\lambda||x_{i_0}| \sum_{j\neq i_0} |m_{i_0,j}|,$$

soit, en divisant par $|\lambda||x_{i_0}|$,

$$|m_{i_0,i_0}| \leq \sum_{j\neq i_0} |m_{i_0,j}|.$$

– Soit $i_0 = N$, auquel cas la dernière relation dans (4.12), qui s'écrit aussi

$$m_{N,N}\, x_N = -\sum_{j=1}^{N-1} m_{N,j}\, x_j,$$

implique, toujours en utilisant le fait que $|x_N| = |x_{i_0}| = \sup_j |x_j|$,

$$|m_{N,N}||x_N| = |m_{N,N}||x_{i_0}| \leq \sum_{j\neq N} |m_{N,j}||x_j| \leq |x_N| \sum_{j\neq N} |m_{N,j}|$$

et, par conséquent, en divisant par $|x_N| = |x_{i_0}|$,

$$|m_{N,N}| = |m_{i_0,i_0}| \leq \sum_{j\neq N} |m_{N,j}| = \sum_{j\neq i_0} |m_{i_0,j}|.$$

Dans tous les cas, quelque soit donc la valeur de $i_0 \in \{1, ..., N\}$, on a mis en contradiction le fait que

$$|m_{i,i}| > \sum_{j\neq i} |m_{i,j}| \qquad \forall\, i = 1, ..., N.$$

L'hypothèse $|\lambda| \geq 1$ est donc absurde, et l'on a bien prouvé ainsi (par l'absurde) que $\rho(T_{\inf}[M]^{-1} \cdot F) < 1$. □

Exemple 4.3 (une exemple d'implémentation de l'algorithme de Jacobi sous `MATLAB`). Nous prendrons ici comme exemple la matrice (à diagonale dominante et entrées réelles) :

```
>> M=[30 7 8 7 ; 7 20 6 5 ; 8 6 40 9 ; 7 5 9 30]
M =
    30     7     8     7
     7    20     6     5
     8     6    40     9
     7     5     9    30
```

Nous allons envisager la résolution du système de Cramer $M \cdot X = B$ avec ici

```
>> B=[32 ; 23 ; 33 ; 31];
```

La solution immédiate proposée sous `MATLAB` pour la résolution directe de ce système linéaire (de Cramer) (avec les erreurs numériques que cela comporte) est

```
>> M^(-1)*B
ans =
   0.649375600384246
   0.625452420067077
   0.457215765882871
   0.640405560134302
```

Implémentons plutôt ici l'algorithme de Jacobi :

$$M = D - E$$

en « isolant » les termes diagonaux de M dans la matrice D. Ici

```
>> D=[30 0 0 0 ; 0 20 0 0 ; 0 0 40 0 ; 0 0 0 30]
D =
    30     0     0     0
     0    20     0     0
     0     0    40     0
     0     0     0    30
>> E=[0 -7 -8 -7 ; -7 0 -6 -5 ; -8 -6 0 -9 ; -7 -5 -9 0]
E =
     0    -7    -8    -7
    -7     0    -6    -5
    -8    -6     0    -9
    -7    -5    -9     0
```

On choisit donc un vecteur initial `X0` arbitrairement et on implémente donc la routine suivante :

```
>> X0 = [1 ; -7 ; 4 ; 10]
>> f=Jacobi(M,B,X0,iter);
```

On obtient, pour respectivement `iter`=50, `iter`=100, `iter`=150, `iter`=200, les résultats :

```
>> Jacobi(M,B,X0,50)
ans =
   0.649375579812054
   0.625452396020122
   0.457215748919845
   0.640405540658039

>> Jacobi(M,B,X0,100)
ans =
   0.649375600384245
   0.625452420067076
   0.457215765882871
   0.640405560134301

>> Jacobi(M,B,X0,150)
ans =
   0.649375600384246
   0.625452420067077
   0.457215765882871
   0.640405560134302
```

```
>> Jacobi(M,B,X0,200)
ans =
   0.649375600384246
   0.625452420067077
   0.457215765882871
   0.640405560134302
```

Si l'on part d'un autre vecteur $X0$:

```
>> X00= [168 ; -754 ; -432 ; 1218]

>> Jacobi(M,B,X00,50)
ans =
   0.649378833791283
   0.625456199615299
   0.457218432024114
   0.640408621290100

>> Jacobi(M,B,X00,100)
ans =
   0.649375600384397
   0.625452420067254
   0.457215765882996
   0.640405560134445

>> Jacobi(M,B,X00,150)
ans =
   0.649375600384246
   0.625452420067077
   0.457215765882871
   0.640405560134302

>> Jacobi(M,B,X00,200)
ans =
   0.649375600384246
   0.625452420067077
   0.457215765882871
   0.640405560134302
```

On retrouve bien la convergence, ce vers manifestement la solution du système de Cramer $M \cdot X = B$.

La méthode de Gauß-Seidel est aussi opérationnelle lorsque $\mathbb{K} = \mathbb{R}$ et que $M = S$ est une matrice réelle symétrique définie positive, *i.e.* telle que

$$\forall\, X \in (\mathbb{R}^N)^*, \quad \langle S \cdot X, X \rangle > 0. \tag{4.13}$$

Notons que cette condition implique

$$\forall\, i \in \{1, ..., N\}, \quad s_{i,i} = \langle S \cdot e_i, e_i \rangle > 0$$

(ici $(e_1, ..., e_N)$ désigne la base canonique de $\mathbb{R}^N$). Par conséquent la matrice $T_{\inf}[S]$ est (facilement) inversible (comme matrice triangulaire inférieure à termes diagonaux tous non nuls). Le fait que l'algorithme de Gauß-Seidel soient opérationnel pour la résolution de systèmes de Cramer $S \cdot X = B$ lorsque S est une telle matrice est une conséquence du résultat suivant (qu'il convient ensuite de combiner avec la Proposition 4.3).

PROPOSITION 4.5 (décomposition de Gauß-Seidel pour une matrice symétrique réelle définie positive). *Soit S une matrice symétrique réelle définie positive de taille (N, N) et $S = T_{\inf}[S] - F$ sa décomposition de Gauß-Seidel* (4.10). *On a $\rho(T_{\inf}[S]^{-1} \cdot F) < 1$.*

DÉMONSTRATION. Comme S est définie positive, tous les termes diagonaux $\langle S(e_j), e_j\rangle$, $j = 1, ..., n$ ($e_1, ..., e_n$ désignant ici la base canonique de $\mathbb{R}^n$) sont strictement positifs. La matrice symétrique réelle définie positive S s'écrit

$$S = T_{\inf}[S] - F = (\mathrm{diag}[S] + {}^tT) + T = (D + {}^tT) + T,$$

où T désigne la matrice triangulaire supérieure (avec diagonale nulle) $-F$. La matrice $T_{\inf}[M] = D + {}^tT$ est une matrice triangulaire inférieure inversible (en vertu de la remarque ci-dessus concernant la stricte positivité des coefficients de S situés sur la diagonale) qui peut s'écrire

$$\begin{aligned} D + {}^tT &= \sqrt{D} \cdot \big(\sqrt{D})^{-1} \cdot (D + {}^tT) \cdot (\sqrt{D})^{-1}\big) \cdot \sqrt{D} \\ &= \sqrt{D} \cdot \big(I_N + (\sqrt{D})^{-1} \cdot {}^tT \cdot (\sqrt{D})^{-1}\big) \cdot \sqrt{D}. \end{aligned}$$

Les valeurs propres de la matrice

$$\begin{aligned} A &= (T_{\inf}[S])^{-1} \cdot F = -(D + {}^tT)^{-1} \cdot T \\ &= -(\sqrt{D})^{-1} \cdot \Big(I_N + (\sqrt{D})^{-1} \cdot {}^tT \cdot (\sqrt{D})^{-1}\Big)^{-1} \cdot (\sqrt{D})^{-1} \cdot T \end{aligned} \tag{4.14}$$

sont les mêmes (au signe près) que celles de la matrice $\sqrt{D} \cdot A \cdot (\sqrt{D})^{-1}$, donc, compte-tenu de (4.14), les mêmes (au signe près) que celles de la matrice

$$B = (I_N + \widetilde{T})^{-1} \cdot {}^t\widetilde{T},$$

où

$$\widetilde{T} := (\sqrt{D})^{-1} \cdot {}^tT \cdot (\sqrt{D})^{-1}.$$

Si X est un vecteur propre (dans $\mathbb{C}^N$) de norme 1, associé à une valeur propre complexe λ de la matrice B, on a

$$(I_N + \widetilde{T})^{-1} \cdot {}^t\widetilde{T} \cdot X = \lambda\, X,$$

soit

$${}^t\widetilde{T} \cdot X = \lambda(I_N + \widetilde{T}) \cdot X = \lambda\, X + \lambda\, \widetilde{T} \cdot X.$$

On a par conséquent, en prenant le produit scalaire à gauche avec X,

$$\langle X, {}^t\widetilde{T} \cdot X\rangle = \lambda\, \|X\|^2 + \lambda\, \langle X, \widetilde{T} \cdot X\rangle = \lambda\big(1 + \langle X, \widetilde{T} \cdot X\rangle\big),$$

et, par conséquent

$$|\langle X, {}^t\widetilde{T} \cdot X\rangle|^2 = |\lambda|^2 \times \big|1 + \langle X, \widetilde{T} \cdot X\rangle\big|^2. \tag{4.15}$$

Comme la matrice S, la matrice

$$(\sqrt{D})^{-1} \cdot S \cdot \sqrt{D} = I_N + \widetilde{T} + {}^t\widetilde{T}$$

est symétrique réelle, définie positive, ce qui implique en particulier :

$$\langle X, (\sqrt{D})^{-1}\cdot S\cdot\sqrt{D}\cdot X\rangle = \|X\|^2 + 2\,\mathrm{Re}\,\langle X, \widetilde{T}\cdot X\rangle = 1 + 2\,\mathrm{Re}\,\langle X, \widetilde{T}\cdot X\rangle > 0. \tag{4.16}$$

Si

$$\langle X, \widetilde{T}\cdot X\rangle = x + iy,$$

on a donc, d'après (4.15) et (4.16),

$$x^2 + y^2 = |\lambda|^2\,(1 + 2x + x^2 + y^2) > |\lambda|^2(x^2 + y^2 + \eta)$$

avec $\eta = (1 + 2x)/2 > 0$. Il en résulte bien $|\lambda| < 1$. Ceci est vrai pour toutes les valeurs propres de B, ce qui implique $\rho(B) < 1$ et, par conséquent, $\rho(A) < 1$. □

Exemple 4.4 (calcul de projection orthogonale). Soit $\langle\ ,\ \rangle$ un produit scalaire sur $\mathbb{R}^N$ et W un sous-espace de $\mathbb{R}^N$ de dimension $p < N$ rapporté à une base $e_1, ..., e_p$ (non nécessairement orthonormée pour ce produit scalaire). Chercher la projection orthogonale $\sum_{j=1}^{p} y_j e_j$ d'un vecteur $X \in \mathbb{R}^N$ sur le sous-espace vectoriel W revient à résoudre le système de Cramer :

$$\sum_{j=1}^{M} \langle e_i, e_j\rangle\, y_j = \langle e_i, X\rangle, \qquad i = 1, ..., p. \tag{4.17}$$

La matrice $S := G[e_1, ..., e_p]$ de ce système de Cramer est une matrice de Gram symétrique réelle définie positive (car $e_1, ..., e_p$ constituent une base de W, donc en particulier un système libre de $\mathbb{R}^N$) et l'on peut donc utiliser la méthode de Gauß-Seidel (d'après la Proposition 4.5) pour résoudre le système de Cramer (4.17) et déterminer ainsi la projection orthogonale d'un vecteur arbitraire X sur W sans avoir à inverser cette matrice de Gram $G[e_1, ..., e_p]$. Une autre manière de procéder aurait consisté à construire (à partir de l'algorithme de Gram-Schmidt initié avec le système libre $(e_1, ..., e_p)$, base de E) une base orthonormée $(\tilde{e}_1, ..., \tilde{e}_p)$ de W (voir le cours d'Algèbre 2). Auquel cas, le calcul de la projection orthogonale d'un vecteur $X \in \mathbb{R}^N$ sur W est donné par

$$\mathrm{Proj}_W^{\perp}[X] = \sum_{j=1}^{p} \langle X, \tilde{e}_j\rangle\, \tilde{e}_j.$$

Il faut noter cependant que l'algorithme d'orthonormalisation de Schmidt peut s'avérer très instable, par exemple lorsque deux des vecteurs e_{j_1} et e_{j_2} de la base $(e_1, ..., e_p)$ de E sont « presque » colinéaires, auquel cas le procédé d'orthonormalisation de Schmidt implique une « presque » division par 0, source évidemment d'une grande instabilité numérique ! La méthode de Gauß-Seidel doit dans ce cas être privilégiée.

4.4. Algorithmes itératifs et méthode des moindres carrés

4.4.1. La décomposition en valeurs singulières d'une matrice rectangulaire réelle ou complexe. La *décomposition en valeurs singulières* (des matrices cette fois *a priori* rectangulaires, de taille $p \times N$ et à coefficients dans $\mathbb{K} = \mathbb{R}$ ou $\mathbb{C}$) est intimement liée à la méthode dite *des moindres carrés*, basée sur l'itération de projections orthogonales, que nous introduirons plus loin (dans la sous-section 4.4.3).

PROPOSITION 4.6 (décomposition en valeurs singulières). *Soit M une matrice rectangulaire à coefficients dans $\mathbb{K} = \mathbb{R}$ (resp. dans $\mathbb{C}$), à p lignes et N colonnes, de rang $r \leq \min(p, N)$. Il existe une matrice carrée orthogonale (resp. unitaire) U de taille (r, r), une unique matrice diagonale réelle positive $D = \mathrm{diag}[\sigma_1, ..., \sigma_r]$, avec $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r > 0$, une matrice carrée orthogonale (resp. unitaire) V de taille (N, N), telles que*

$$M = U \cdot \widetilde{D} \cdot {}^tV \quad (\textit{resp.} \quad M = U \cdot \widetilde{D} \cdot {}^t\overline{V}), \tag{4.18}$$

$\widetilde{D}$ désignant la matrice à p lignes et N colonnes obtenue en complétant D inférieurement et latéralement (à droite) si nécessaire par des zéros. Les nombres $\sigma_1^2 \geq \cdots \geq \sigma_r^2 > 0$ sont les valeurs propres strictement positives de la matrice symétrique réelle positive (resp. hermitienne positive) $M \cdot {}^tM$ (resp. $M \cdot {}^t\overline{M}$). Les racines strictement positives $\sigma_1 \geq \sigma_2 \cdots \geq \sigma_r > 0$ de ces nombres sont appelées valeurs singulières de la matrice M.

REMARQUE 4.3. L'implémentation sous `MATLAB` de la Proposition 4.6 est réalisée *via* la routine :

```
>>[U,D,V]=svd(M);
```

Notons, sous cette implémentation sous `MATLAB`, la matrice D est une matrice à p lignes et N colonnes, mais dont les seuls coefficients non nuls sont les d_{ii}, $i = 1, ..., r \leq \min(p, N)$. La matrice U est, elle, une matrice (p, p) unitaire. Le résultat fourni par `MATLAB` correspond donc à celui donné dans la proposition seulement dans le cas où $N \geq p$ et $r = \mathrm{rang}(M)$ (*i.e.* dans le cas où l'application $\mathbb{K}$-linéaire $L : \mathbb{K}^N \to \mathbb{K}^p$ représentée par M dans les bases canoniques à la source $\mathbb{K}^N$ et au but $\mathbb{K}^p$ est surjective).

DÉMONSTRATION. On fait la démonstration en supposant $r = p$, c'est-à-dire l'application M surjective (on remplace l'espace d'arrivée par l'image de l'application linéaire dont M figure la matrice dans les bases canoniques respectivement de $\mathbb{K}^N$ et de $\mathbb{K}^p$). Le cas général s'y ramène aisément. Soit $L : \mathbb{K}^N \to \mathbb{K}^p = \mathbb{K}^r$ l'application linéaire surjective que représente M dans les bases canoniques de $\mathbb{K}^N$ et $\mathbb{K}^r$. D'après la formule du rang, le noyau de L est un sous-espace vectoriel $\mathrm{Ker}\, L$ de $\mathbb{R}^N$, de dimension $N - r$, que l'on peut donc rapporter à une base orthonormée $(e_{r+1}, ..., e_N)$ (pour le produit scalaire canonique). Si l'on complète (suivant le procédé de Gram-Schmidt, *c.f.* le cours d'Algèbre 2) cette base en une base orthonormée $(e_1, ..., e_N)$ de $\mathbb{K}^N$, on constate que $(e_1, ..., e_r)$ constitue une base du sous-espace du sous-espace $(\mathrm{Ker}L)^\perp$ constitué des vecteurs de $\mathbb{K}^N$ orthogonaux au noyau de L. La restriction de L à $(\mathrm{Ker}\, L)^\perp$ est une application linéaire bijective entre $(\mathrm{Ker}\, L)^\perp$ et $\mathbb{K}^r$. Si $\mathbb{K}^r$ est rapporté à la base $(e_1, ..., e_r)$ et $\mathbb{K}^N$ à sa base canonique, la matrice de L dans ces bases s'écrit sous la forme :

$$\begin{pmatrix} & 0 & \cdots & \cdots & 0 \\ A & \vdots & \vdots & \vdots & \vdots \\ & 0 & \cdots & \cdots & 0 \end{pmatrix}$$

où $A \in \mathcal{M}_{\mathbb{K}}(r, r)$ est une matrice inversible ; il existe donc une matrice unitaire [15] V_1 de taille (N, N) ($V_1 \cdot {}^t\overline{V_1} = {}^t\overline{V_1} \cdot V_1 = I_N$), une matrice unitaire U_1 de taille (r, r),

15. Si $\mathbb{K} = \mathbb{R}$, il faut remplacer partout « matrice unitaire » par « matrice orthogonale ».

telles que

$$M = U_1 \cdot \begin{pmatrix} & 0 & \cdots & \cdots & 0 \\ A & \vdots & \vdots & \vdots & \vdots \\ & 0 & \cdots & \cdots & 0 \end{pmatrix} \cdot {}^t\overline{V_1}.$$

La matrice symétrique réelle strictement positive $A \cdot {}^tA$ si $\mathbb{K} = \mathbb{R}$ (*resp.* hermitienne strictement positive $A \cdot {}^t\overline{A}$ si $\mathbb{K} = \mathbb{C}$) se diagonalise dans une base orthonormée [16] de $\mathbb{K}^N$:

$$A \cdot {}^t\overline{A} = U \cdot \mathrm{diag}\,(\sigma_1^2, ..., \sigma_r^2) \cdot {}^tU\,,$$

où U est une matrice unitaire (r, r). Comme

$$A \cdot {}^t\overline{A} = \Big(U \cdot \mathrm{diag}\,(\sigma_1, ..., \sigma_r)\Big) \cdot {}^t\overline{\big[U \cdot \mathrm{diag}\,(\sigma_1, ..., \sigma_r)\big]}$$

la matrice

$${}^t\overline{A} \cdot \Big({}^t\overline{\big[U \cdot \mathrm{diag}\,(\sigma_1, ..., \sigma_r)\big]}\Big)^{-1}$$

est une matrice unitaire W de taille (r, r) et l'on peut donc écrire

$${}^t\overline{A} = W \cdot \mathrm{diag}\,(\sigma_1, ..., \sigma_r) \cdot {}^t\overline{U}\,,$$

soit

$$A = U \cdot \mathrm{diag}\,(\sigma_1, ..., \sigma_r) \cdot {}^t\overline{W}.$$

On peut donc ainsi écrire

$$A = U \cdot \begin{pmatrix} \sigma_1 & 0 & \cdots & 0 & 0 & \cdots & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & \cdots & \cdots & \sigma_r & 0 & \cdots & \cdots & 0 \end{pmatrix} \cdot {}^t\overline{V} \tag{4.19}$$

où U et V sont respectivement des matrices unitaires de tailles respectivement (r, r) et (N, N). □

Supposons que $u_1, ..., u_r$ soient les r vecteurs colonnes de la matrice (r, r) U (orthogonale ou unitaire suivant que $\mathbb{K} = \mathbb{R}$ ou $\mathbb{K} = \mathbb{C}$) donnée dans (4.18), complétés par des zéros en des vecteurs de $\mathbb{K}^p$ (ou directement les r premiers vecteurs colonnes de la matrice (p, p) U orthogonale ou unitaire donnée, *c.f.* la Remarque 4.3, par la commande `svd` sous `MATLAB`) et $v_1, ..., v_N$ les N vecteurs colonnes de la matrice V. Si $Y \in \mathbb{K}^p$, le vecteur

$$X = L_{\mathrm{app}}^{-1}[Y] = \sum_{j=1}^{r} \frac{\langle Y, u_j \rangle}{\sigma_j}\, v_j$$

représente le vecteur X de $\mathbb{K}^N$ de norme minimale parmi tous les vecteurs $x \in \mathbb{K}^N$ qui minimisent l'application

$$x \in \mathbb{K}^N \longmapsto \|L(x) - Y\|_2^2$$

On l'appelle *pseudo-inverse* de Y *via* l'application L. Lorsque L est surjective (c'est-à-dire $r = p \leq N$), ce vecteur $X = L_{\mathrm{app}}^{-1}[Y]$ figure donc la projection orthogonale du vecteur nul sur le sous-espace affine de $\mathbb{K}^N$ défini comme

$$E_L(Y) := \{x \in \mathbb{K}^N\,;\, L(x) = Y\}.$$

16. Voir le cours d'Algèbre 2. Si $\mathbb{K} = \mathbb{R}$, il s'agit d'une matrice symétrique réelle et il convient de remplacer partout ≪ matrice unitaire ≫ par ≪ matrice orthogonale ≫.

Nous verrons dans la sous-section 4.4.3 un moyen itératif de calculer, lorsque $Y \in \operatorname{Im} L$, l'inverse approché $L^{-1}_{\text{app}}[Y]$ sans avoir à passer par la décomposition en valeurs singulières de la matrice de l'application linéaire L exprimée dans les bases canoniques.

4.4.2. Valeurs singulières et conditionnement. La notion de *conditionnement* (pour les matrices carrées inversibles) est une notion capitale en algorithmique numérique. Elle « conditionne » précisément la stabilité (sous petites perturbations portant sur les données) de la résolution de système de Cramer du type $M \cdot X = B$, M désignant une matrice $N \times N$ inversible à coefficients dans le corps $\mathbb{K} = \mathbb{R}$ ou $\mathbb{C}$, B un vecteur colonne donné. Cette question avait déjà été soulevée au début de ce cours, avec l'exemple 1.2.2.

Revenons à la résolution d'un système de Cramer

$$M \cdot X = B \tag{4.20}$$

où M est une matrice (N, N) inversible dont nous envisageons de perturber les entrées. Notons $M + \Delta M$ la matrice M perturbée (`Mperturb` dans l'exemple 1.2.2) et $X + \Delta X$ la solution du système (que l'on suppose toujours de Cramer, la perturbation étant assez petite pour que $\det(M + \Delta M) \neq 0$)

$$(M + \Delta M) \cdot (X + \Delta X) = B \tag{4.21}$$

(ici, on ne perturbe pas B). En mettant ensemble les deux relations (4.20) et (4.21), on trouve immédiatement (par différence)

$$\Delta M \cdot X + M \cdot \Delta X + \Delta M \cdot \Delta X = 0\,,$$

ce que l'on réécrit

$$M \cdot \Delta X = -\Delta M \cdot (X + \Delta X)\,. \tag{4.22}$$

On peut transformer (4.22) en

$$\Delta X = -M^{-1} \cdot \Delta M \cdot (X + \Delta X)$$

et en déduire

$$\|\Delta X\| \leq \|M^{-1}\| \times \|\Delta M\| \times \|X + \Delta X\|\,,$$

ce que l'on écrit (un peut artificiellement !)

$$\frac{\|\Delta X\|}{\|X + \Delta X\|} \leq \Big(\|M\| \times \|M^{-1}\|\Big) \times \frac{\|\Delta M\|}{\|M\|}\,,$$

ou encore

$$\frac{\|(X + \Delta X) - X\|}{\|X + \Delta X\|} \leq \Big(\|M\| \times \|M^{-1}\|\Big) \times \frac{\|\Delta M\|}{\|M\|}\,.$$

Le membre de gauche

$$\frac{\|(X + \Delta X) - X\|}{\|X + \Delta X\|}$$

peut s'interpréter comme une « erreur relative » sur X tandis qu'au membre de droite, on voit apparaitre

$$\frac{\|\Delta M\|}{\|M\|}\,,$$

qui correspond (en norme) à l'erreur relative commise sur M. La quantité

$$\|M\| \times \|M^{-1}\|$$

qui « contrôle » en un certain sens la stabilité de la résolution du système $M \cdot X = B$ est appelée à jouer un rôle majeur.

DÉFINITION 4.3 (la notion de conditionnement). Si M est une matrice carrée inversible (N, N) de nombres réels ou complexes, on appelle *conditionnement* de M (relativement au choix d'une norme $\| \ \|$ sur $\mathbb{R}^N$ ou $\mathbb{C}^N$, suivant le contexte on l'on se place) la quantité $\|M\| \times \|M^{-1}\|$.

Si l'on choisit comme norme sur $\mathbb{K}^N$ ($\mathbb{K} = \mathbb{R}$ ou $\mathbb{K} = \mathbb{C}$) la norme euclidienne $\| \ \|_2$ (choix « par défaut » dans `MATLAB`), et que $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_N > 0$ désignent les valeurs singulières de M (*c.f.* la Proposition 4.6), on a $\|M\|_2 = \sigma_1$ et $\|M^{-1}\|_2 = 1/\sigma_N$, ce qui donne pour le conditionnement :

$$\mathrm{cond}_2(M) = \sigma_1/\sigma_N. \tag{4.23}$$

Plus le conditionnement de M est grand, plus la résolution numérique de $M \cdot X = B$ s'avère instable sous l'effet d'une perturbation des entrées (dans M ou dans B). C'est souvent le cas des matrices symétriques de Gram qui s'avèrent fréquemment mal conditionnées. Ce problème de mauvais conditionnement nous est déjà apparu à propos de l'interpolation de Lagrange (*cf.* les remarques en fin de la sous-section 3.2.2).

4.4.3. Itération de projections orthogonales. Soient $W_1, ..., W_p$ p-sous-espaces vectoriels de $\mathbb{K}^N$, W_j, $j = 1, ..., p$, étant défini comme le noyau (de dimension $N - p_j$) d'une certaine application linéaire L_j surjective $L_j \ : \mathbb{K}^N \to \mathbb{K}^{p_j}$, $j = 1, ..., M$. Soit x un vecteur (*a priori* inconnu) de $\mathbb{K}^N$, tel que les vecteurs $L_j(x) = Y_1, ..., L_p(x) = Y_p$ soient connus (par exemple mesurés lors d'une expérience physique). L'élément X de $\mathbb{K}^N$ de norme minimale parmi tous les $u \in \mathbb{K}^N$ tels que $L_j(u) = Y_j$, $j = 1, ..., p$, est par définition même la projection orthogonale du vecteur nul de $\mathbb{K}^N$ sur le sous-espace affine

$$\bigcap_{j=1}^{M} (x + W_j) = x + \bigcap_{j=1}^{M} W_j.$$

Cette projection orthogonale s'obtient *via* un processus itératif aisé à implémenter : on part de $X_0 = 0$, puis on définit

$$X_{k+1} = \big(\mathrm{Proj}^{\perp}_{x+W_p} \circ \cdots \circ \mathrm{Proj}^{\perp}_{x+W_1}\big)[X_k]. \tag{4.24}$$

Pour calculer $\mathrm{Proj}_{x+W_j}[v]$ pour un vecteur $v \in \mathbb{K}^N$ quelconque, on remarque que

$$v - \mathrm{Proj}_{x+W_j}[v] \in F_j^{\perp} = \mathrm{Im}\, L_j^*$$

et donc qu'il existe $w_j(v) \in \mathbb{K}^N$ tel que :

$$L_j(v) - L_j(\mathrm{Proj}_{x+W_j}[v]) = L_j(v) - L_j(x) = L_j(v) - Y_j = (L_j \circ L_j^*)(w_j(v)).$$

On trouve $w_j(v)$ en composant avec l'inverse de $L_j \circ L_j^*$ (dans le cas $\mathbb{K} = \mathbb{R}$, on peut calculer cet inverse en utilisant la méthode de Gauß-Seidel, *c.f.* la Proposition 4.5) et on en déduit

$$\mathrm{Proj}_{x+W_j}[v] = v - L_j^*[w_j(v)].$$

L'algorithme itératif (4.24) est ainsi implémentable à partir du vecteur initial $X_0 = 0$. Il conduit à une approximation du vecteur X (de norme minimale) solution de $L_j(X) = Y_j$, $j = 1, ..., p$.

Cette démarche itérative, impliquant cette fois le concept d'orthogonalité, a été proposée par le mathématicien polonais Stefan Kaczmarz (1895-1940). Pour prouver la convergence de la suite $(X_k)_{k\geq 0}$ initiée au vecteur nul vers la projection de ce vecteur sur le sous-espace affine des vecteurs $u \in \mathbb{K}^N$ tels que $L_j(u) = Y$ pour $j = 1, ..., p$, on pourra par exemple s'appuyer sur la figure 4.1 suivante.

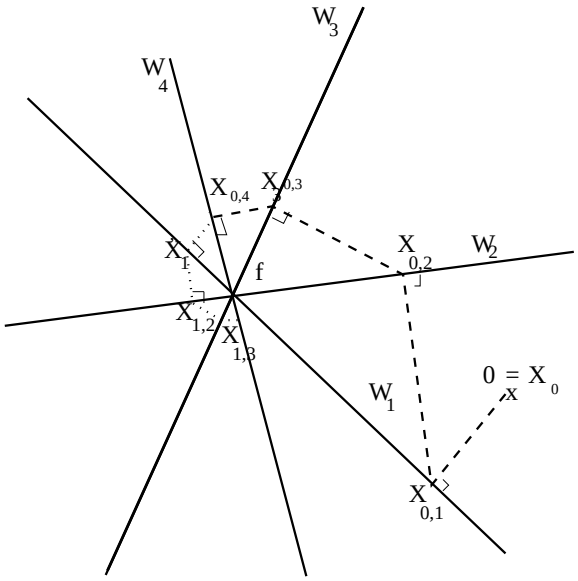

FIGURE 4.1. La démarche itérative de S. Kaczmarz

CHAPITRE 5

Schémas numériques simples pour la résolution des EDO

5.1. Les bases théoriques (admises) : Cauchy-Lipschitz

On se propose de modéliser un phénomène physique $t \longmapsto Y(t)$ (et, si possible, d'en anticiper le passé ou d'en prévoir l'évolution à partir de sa valeur $Y_0 = Y(t_0)$ à l'instant $t = t_0$). Ici $(t, Y(t))$ prend ses valeurs dans un ouvert U de $\mathbb{R}^{\mathbf{N}+1}$ dit *espace des phases* ou encore *espace des états* du phénomène. On fait l'hypothèse que ce phénomène est régi (on dit aussi « contraint ») par une équation différentielle [1]

$$Y'(t) = F(t, Y(t)), \tag{5.1}$$

F désignant une fonction continue dans l'ouvert U et à valeurs dans $\mathbb{R}^{\mathbf{N}}$. Ce qui signifie que $t \longmapsto Y(t)$ est de classe C^1 sur son intervalle ouvert I de vie (à déterminer) autour de t_0, et se plie sur cet intervalle I à la relation (5.1).

Le théorème majeur que nous admettrons ici (et qui soutend de fait la résolution numérique du problème) est le Théorème de Cauchy-Lipschitz [2] dont voici l'énoncé. Il s'agit en fait d'un avatar du Théorème du point fixe (Théorème 4.1), mais ici dans un cadre où le $\mathbb{R}$-espace de dimension finie $\mathbb{R}^{\mathbf{N}}$ se trouve remplacé par un $\mathbb{R}$-espace vectoriel de dimension infinie, en l'occurrence le $\mathbb{R}$-espace vectoriel des fonctions continues de $[t_0, t_0 + T]$, ($t_0 \in \mathbb{R}$, $T > 0$) dans $\mathbb{R}^N$, $[t_0, t_0 + T]$ désignant un segment non vide de l'axe réel $\mathbb{R}$. Le fait de passer ici du cadre de la dimension finie au cadre de la dimension infinie nous prive (à ce niveau L2) de la possibilité de justifier ce thórème autrement qu'heuristiquement ou (de manière empirique) numériquement.

THEORÈME 5.1 (théorème de Cauchy-Lipschitz). *Soit U un ouvert de $\mathbb{R}^{\mathbf{N}+1}$ et $F : U \to \mathbb{R}^N$ une fonction continue satisfaisant au voisinage de tout point la condition suivante (dite de Lipschitz) : pour tout $(t_0, Y_0) \in U$, il existe $\epsilon > 0$, $\eta > 0$, $K \geq 0$ (dépendants (t_0, Y_0)) tels que $[t_0 - \epsilon, t_0 + \epsilon] \times \overline{B_{\mathbb{R}^{\mathbf{N}}}(Y_0, \eta)} \subset U$ et*

$$\begin{aligned} &\forall\, t \in [t_0 - \epsilon, t_0 + \epsilon]\,,\ \forall\, Y_1, Y_2 \in \overline{B_{\mathbb{R}^n}(Y_0, \eta)}, \\ &\|F(t, Y_1) - F(t, Y_2)\| \leq K \|Y_1 - Y_2\|. \end{aligned} \tag{5.2}$$

Alors, pour tout $(t_0, Y_0) \in U$, il existe un <u>unique</u> couple (I, Y), où I est un intervalle ouvert de $\mathbb{R}$, $Y : I \to \mathbb{R}$ une fonction de classe C^1, tel que :

---

1. EDO : « **E**quation **D**ifférentielle **O**rdinaire, c'est à dire en une variable (le temps en général), par rapport à l'acronyme EDP pour « **E**quation aux **D**érivées **P**artielles lorsque plusieurs variables (en général de temps et d'espace) sont impliquées.

2. Au nom du mathématicien français Augustin Cauchy (1789-1857) est ici associé celui de l'analyste allemand Rudolph Lipschitz (1832-1903), à qui l'on doit la mise en évidence de l'importance de la condition (5.2) ; une fonction satisfaisant cette condition est d'ailleurs appelée fonction *localement lipschitzienne*.

(5.3)
(1) *pour tout* $t \in I$, $(t, Y(t)) \in U$ *et*
$$Y(t_0) = Y_0 \text{ (\textit{condition initiale})}$$
$$\forall\, t \in I, \quad (t, Y(t)) \in U \quad \& \quad Y'(t) = F(t, Y(t))$$
(on dit que (I, Y) *est solution du problème de Cauchy* (5.3)*);*

(2) *si* $(\widetilde{I}, \widetilde{Y})$ *est un autre couple solution du même problème de Cauchy* (5.3)*, alors* $\widetilde{I} \subset I$ *et* $\widetilde{Y}$ *est la restriction de* Y *à* $\widetilde{I}$ *(on dit que* (I, Y) *est une solution maximale du problème de Cauchy* (5.3)*).*

Dans le contexte de l'algorithmique numérique, nous retiendrons un résultat plus fort (avec des hypothèses plus contraignantes). Si $U = I_0 \times \mathbb{R}^{\mathbf{N}}$ et $F : I_0 \times \mathbb{R}^{\mathbf{N}} \to \mathbb{R}^{\mathbf{N}}$ est une fonction continue telle que, pour tout segment $[t_0, t_0 + T]$ de I_0, pour tout $R > 0$, il existe une constante $K_{[t_0, t_0+T], R}$ telle que

(5.4)
$$\forall\, t \in [t_0, t_0 + T], \quad \forall\, Y_1, Y_2 \in B_{\mathbb{R}^{\mathbf{N}}}(0, R),$$
$$\|F(t, Y_1) - F(t, Y_2)\| \leq K_{[t_0, t_0+T], R} \|Y_1 - Y_2\|,$$

alors il existe, pour chaque segment $[t_0, t_0 + T] \subset I_0$, pour chaque $Y_0 \in \mathbb{R}^{\mathbf{N}}$, une unique fonction $t \in [t_0, t_0 + T] \to \mathbb{R}^{\mathbf{N}}$ de classe C^1 sur le segment $[t_0, t_0 + T]$ (c'est-à-dire se prolongeant en une fonction de classe C^1 au voisinage de ce segment [3]) telle que

$$Y(t_0) = Y_0 \quad \& \quad \forall\, t \in [t_0, t_0 + T],\ Y'(t) = F(t, Y(t)), \tag{5.5}$$

ce que l'on peut résumer en

$$\forall\, t \in [t_0, t_0 + T], \quad Y(t) = Y_0 + \int_0^t F(\tau, Y(\tau))\, d\tau = T_{Y_0}[Y](t) \tag{5.6}$$

(voilà le « point fixe » en action : la solution $t \mapsto Y(t)$ du problème apparait ici comme un point fixe de l'application $\mathbb{R}$-linéaire T_{Y_0}). Le segment $[t_0, t_0 + T] \subset I_0$ étant ici donné, nous nous intéresserons dans ce chapitre (Section 5.2), sous l'angle de l'algorithmique numérique, à la manière de calculer numériquement une solution approchée $t \in [t_0, t_0 + T] \longmapsto Y_{\text{app}}(t)$ à l'équation intégrale figurant en (5.6).

Exemple 5.1 (équations différentielles linéaires). Lorsque $U = I_0 \times \mathbb{R}^{\mathbf{N}}$, où I_0 est un intervalle de $\mathbb{R}$, et que la fonction F est de la forme $F(t, Y) = A(t) \cdot Y + B(t)$, où A et B sont respectivement des applications continues de I_0 dans l'espace $\mathcal{M}_{\mathbf{N},\mathbf{N}}$ (des matrices réelles de taille $(\mathbf{N}, \mathbf{N})$) et de $\mathbb{R}$ dans $\mathbb{R}^{\mathbf{N}}$, les conditions (5.6) sont vérifiées pour tout segment $[t_0, t_0 + T]$ inclus dans I_0. Mais il faut avoir conscience que, à moins que $t \mapsto A(t)$ ne soit une fonction constante, on ne sait pas (en général) résoudre $Y' = A(t) \cdot Y + B(t)$ avec données initiales arbitraires $Y(t_0) = Y_0$ autrement que numériquement ! Autrement dit, même dans le cas linéaire (pourtant relativement simple !), l'approche numérique est en général incontournable.

Il faut noter que la résolution [4] des équations différentielles d'ordre supérieur
$$y^{(\mathbf{N})} = F(t, y, y', ..., y^{(\mathbf{N}-1)}),$$

3. De fait, la fonction se prolonge en une fonction de classe C^1 à I_0 tout entier, cette fonction vérifiant d'ailleurs $Y'(t) = F(t, Y(t))$ pour tout $t \in I_0$.

4. On cherche, pour $(t_0, y_{0,0}, ..., y_{0,\mathbf{N}-1}) = (t_0, Y_0) \in U$, les couples (I, y) tels que I soit un intervalle de $\mathbb{R}$ avec $(t, y(t), y'(t), ..., y^{(\mathbf{N}-1)}(t)) \in U$ pour tout $t \in I$,
$$y^{(\mathbf{N})}(t) = F(t, y(t), y'(t), ..., y^{(\mathbf{N}-1)}(t)) \quad \forall\, t \in I \tag{5.7}$$

où F est une fonction continue dans un ouvert U de $\mathbb{R}^{\mathbf{N}+1}$ vérifiant la condition (5.2) dans cet ouvert, se ramène à celle des équations différentielles $Y' = F(t, Y)$. Il suffit en effet de remarquer que dire que (I, y) vérifie l'équation d'ordre $\mathbf{N}$ (5.7) avec les conditions initiales (5.8) équivaut à dire que (I, Y), où $Y(t) = (y(t), y'(t), ..., y^{(\mathbf{N}-1)}(t))$ vérifie le système

$$\begin{aligned} &Y_0'(t) = Y_1(t) \\ &Y_1'(t) = Y_2(t) \\ &\vdots \\ &Y_k'(t) = Y_{k+1}(t) \\ &\vdots \\ &Y_{\mathbf{N}-2}'(t) = Y_{\mathbf{N}-1}(t) \\ &Y_{\mathbf{N}-1}'(t) = F(t, Y_0(t), ..., Y_{\mathbf{N}-1}(t)). \end{aligned} \tag{5.9}$$

avec les conditions initiales $Y(t_0) = (y_{0,0}, ..., y_{0,\mathbf{N}-1})$.

5.2. Résolution numérique des EDO

On se place dans le contexte présenté dans la section 5.1, où la condition de Lipschitz forte (5.4) est supposée remplie par la fonction F. On se limitera aussi ici au cas $\mathbf{N} = 1$ (le cas général se traitant coordonnée-fonction par coordonnée-fonction ou matriciellement). On notera donc $F(t, y) = f(t, y)$ pour $(t, y) \in I_0 \times \mathbb{R}$.

5.2.1. Schémas numériques explicites ou implicites : l'exemple d'Euler. Nous allons introduire ici un principe [5] basé sur la démarche suivante (lorsque $[t_0, t_0 + T] \subset I_0$) :

(1) on choisit un « pas maximal » $h_0 > 0$ et une fonction continue

$$\Phi[f] \ : [t_0, t_0 + T] \times \mathbb{R} \times [0, h_0] \longrightarrow \mathbb{R}.$$

(2) pour $h = T/N \leq h_0$, on construit la suite récurrente $(y_{h,k})_{k \geq 0}$ solution de

$$\frac{y_{h,k+1} - y_{h,k}}{h} = \Phi[f](t_k, y_{h,k}, h), \quad k = 0, ..., N-1, \ (\text{ici } t_k = t_0 + k\,h) \tag{5.10}$$

et initiée à $y_{h,0} = y_0$, y_0 étant donné dans $\mathbb{R}$ (condition initiale).

L'objectif visé est que, si le pas h est fixé assez petit (en tout cas inférieur à h_0), $y_{h,k}$ approche la valeur de la solution f de l'équation différentielle $y'(t) = f(t, y(t))$ (avec condition initiale $y(t_0) = y_0$) au point $t_k = t_0 + k\,h$ (pour simplifier, on omet dans la notation t_k la dépendance implicite en h) du maillage

$$t_0 < t + h < t + 2h < \cdots < t_0 + (N-1)\,h < t_0 + Nh = t_0 + T.$$

On se repose pour cela sur le fait que

$$\frac{y_{h,k+1} - y_{h,k}}{h}$$

---

et que soient remplies les conditions initiales

$$y(t_0) = y_{0,0},\ y'(t_0) = y_{0,1}, ..., y^{(\mathbf{N}-1)}(t_0) = y_{0,\mathbf{N}-1} \quad \text{(conditions initiales)}, \tag{5.8}$$

5. C'est le principe des méthodes dites « à un pas ».

peut être interprété comme la valeur approchée de $t \mapsto y'(t)$ soit au point médian de $[t_k, t_{k+1}]$ (calcul numérique de dérivée dit « centré »), soit au point t_k ou t_{k+1} (calcul numérique de dérivée dit « décentré »). En conséquence, il n'est plus question ici, comme nous l'avions fait jusque là dans les méthodes itératives décrites aux chapitres 2 et 3 (Newton, sécante, dichotomie, Jacobi, Gauß-Seidel, etc.) d'« écraser » la valeur $y_{h,k}$ au fur et à mesure du déroulement de l'algorithme. Ces valeurs $y_{h,0} = y_0$, $y_{h,1}$,..., $y_{h,k}$, ..., doivent au contraire être ici stockées en mémoire, ce en vu de l'affichage final du « graphe approché » de la solution f sur le segment temporel $[t_0, t_0 + T]$, en l'occurence de celui de l'application discrétisée

$$k \in \{0, ..., N\} \longmapsto y_{h,k} \simeq f(t_0 + kh).$$

Un tel schéma numérique est dit explicite car le calcul de $y_{h,k+1}$ se fait à partir de la connaissance de $y_{h,k}$ tant que $k = 0, ..., N-1$.

On peut aussi envisager les *schémas implicites* où, dans l'étape 2 du processus décrit, on remplace (5.10) par

$$\frac{y_{h,k} - y_{h,k-1}}{h} = \Psi[f](t_k, y_{h,k-1}, y_{h,k}, h), \quad k = 1, ..., N, \text{ (ici } t_k = t_0 + k\,h), \tag{5.11}$$

où

$$\Psi[f] \ : (t, y, \xi, h) \in [t_0, t_0 + T] \times \mathbb{R} \times \mathbb{R} \times [0, h_0] \longmapsto \Psi(t, y, \xi, h) \in \mathbb{R}$$

est une fonction continue (toujours en maintenant la condition initiale $y_{h,0} = 0$) ; cette fois la détermination de $y_{h,k}$ à partir de $y_{h,k-1}$ passe par la résolution de l'équation implicite (d'inconnue ξ)

$$\frac{\xi - y_{h,k-1}}{h} = \Phi[f](t_k, y_{h,k-1}, \xi, h).$$

Qu'il s'agisse de la méthode explicite (basée sur (5.10)) ou implicite (basée sur (5.11)), la récurrence permettant de calculer de manière inductive les $y_{h,k}$ lorsque le pas $h \leq h_0$ est fixé est une récurrence à un terme ($y_{h,k+1}$ fonction de $y_{h,k}$). C'est la raison pour laquelle on appelle ces méthodes *méthodes à un pas*.

Exemple 5.2 (les méthodes d'Euler explicite, implicite et modifiée). Si l'on prend

$$\Phi[f](t, y, h) = f(t, y)$$

dans (5.10) (cette fonction est indépendante de h dans ce cas), on obtient le schéma numérique dit *schéma d'Euler explicite*, que vous avez certainement rencontré dès la Terminale [6]. Voici le code `MATLAB` pour ce schéma, la fonction `f` (de deux variables `t` et `y`) étant déclarée en ligne par

```
>> f = inline ('expression MATLAB en t et y', 't','y');
```

l'instant initial étant `t0`, la condition initiale `y(t0)=y0`, l'intervalle d'étude `[t0,t0+T]`, et le pas choisi étant ici `T/N`).

```
function [t,y] = Euler(t0,y0,T,N,f)
h = T/N;
t = [t0];
y = [y0];
for k=1:N
```

6. On doit ce schéma numérique au mathématicien suisse Leonhard Euler (1707-1783), pionnier des mathématiques actuelles au siècle des lumières, qui l'introduisit dès 1768 sans doute.

```
  yfinal = y(k) + h*f(t(k),y(k));
  t = [t,t(k)+h];
  y = [y,yfinal];
end
plot(t,y)
```

En prenant

$$\Psi[f]\,(t,y,\xi,h) = f(t,\xi)$$

dans (5.11), on obtient avec (5.11) le *schéma d'Euler implicite* (ou *rétrograde*). Si l'on souhaite respecter le fait que l'erreur dans le calcul numérique de dérivée est meilleure dans la situation centrée (fait que l'on admettra ici), on choisit comme fonction $\Phi[f]$ la fonction

$$(t,y,h) \longmapsto f(t+h/2, y+h/2\, f(t,y)).$$

Cette fois la fonction Φ fait intervenir la variable h et le schéma numérique ainsi construit est le *schéma d'Euler explicite modifié*.

5.2.2. Intégration numérique : les méthodes de Newton-Cotes. Une alternative pour remplacer les relations (5.10) ou (5.11), si l'on a en mémoire l'équivalence entre (5.5) et (5.6) pour traduire que $t \mapsto y(t)$ est solution du problème de Cauchy, est de penser à la résolution de l'équation sous la forme de la recherche d'une solution à l'équation intégrale

$$y(t) = y_0 + \int_0^t f(\tau, y(\tau))\, d\tau,$$

soit au jeu d'équations intégrales

$$y_{h,k+1} - y_{h,k} \simeq y(t_{k+1}) - y(t_k) = \int_{t_k}^{t_{k+1}} f(\tau, y(\tau))\, d\tau, \tag{5.12}$$

le membre de droite étant exprimé à partir d'une méthode numérique. Nous allons donc dans cette sous-section présenter donc succintement de telles méthodes d'intégration numérique sur un segment $[a,b]$ de $\mathbb{R}$, pour lequel on dispose d'un maillage

$$a \leq x_0 < x_1 < \cdots < x_M \leq b. \tag{5.13}$$

L'objectif que nous avons ici est de présenter un calcul approché de l'intégrale

$$I[f;[a,b]] := \int_a^b f(t)\, dt$$

(f étant une fonction continue sur un segment borné $[a,b]$) avec ces deux exigences :

- l'intégrale approchée $I_{\rm app}[f;[a,b]]$ sur $[a,b]$ dépend de manière linéaire des entrées évaluations de f aux $M+1$ nœuds du maillage, *i.e.* $f(x_0), ..., f(x_M)$;
- le calcul approché devient exact, c'est-à-dire

  $$I_{\rm app}[f;[a,b]] = I[f;[a,b]],$$

  lorsque f est une fonction polynomiale de degré inférieur où égal à M.

On souhaite donc déterminer des scalaires $\lambda_0, ..., \lambda_M$ (« universelles », c'est-à-dire ne dépendant que des nœuds $x_0, ..., x_M$ du maillage) tels que

$$I_{\rm app}[f;[a,b]] = \sum_{j=0}^{M} \lambda_j\, f(x_j) \tag{5.14}$$

et

(5.15)
$$\int_a^b t^k\,dt = \frac{b^{k+1}-a^{k+1}}{k+1} = I[t^k;[a,b]] = I_{\rm app}[t^k;[a,b]] = \sum_{j=0}^{M} \lambda_j\, x_j^k \quad \forall\, k = 0,...,M.$$

Comme les x_j, $j = 0,...,M$, sont $M+1$ points distincts, le système linéaire de $M+1$ équations en les $M+1$ inconnues $\lambda_0,...,\lambda_M$ est de Cramer (le déterminant de ce système est ce que l'on appelle un déterminant de Vandermonde, valant $\prod_{0\le j_1<j_2\le M}(x_{j_2}-x_{j_1})$, donc non nul puisque les x_j sont distincts). Il existe donc un unique vecteur de coefficients $(\lambda_0,...,\lambda_M)$ solution de notre problème. Voici plusieurs cas particuliers importants.

- Le cas où $M = 0$ et où, par souci de compromis, on prend $x_0 = (a+b)/2$. On trouve dans ce cas $\lambda_0 = b-a$ et la formule approchée est dans ce cas

(5.16)
$$I_{\rm app}[f;[a,b]] = (b-a) f\Big(\frac{a+b}{2}\Big).$$

  Ce calcul approché est dit *méthode des rectangles.*. Il se trouve que l'on a de la chance ici car la formule

(5.17)
$$\int_a^b f(t)\,dt = \mathrm{I}_{\rm app}[f;[a,b]]$$

  se révèle exacte pour les fonctions polynomiales de degré 1 (elle devient fausse par contre pour les fonctions polynomiales de degré 2).
- Le cas où $M = 1$ et où, toujours par souci de compromis, on prend $x_0 = a$ et $x_1 = b$. Dans ce cas, on trouve $\lambda_0 = \lambda_1 = (b-a)/2$ et la formule approchée devient

(5.18)
$$I_{\rm app}[f;[a,b]] = \frac{b-a}{2}\,(f(a)+f(b)).$$

  Ce calcul approché correspond à la *méthode des trapèzes*. Le calcul approché cesse aussi d'être exact pour les fonctions polynomiales de degré 2.
- Le cas $M = 2$ où, toujours par souci de compromis, on prend $x_0 = a$, $x_1 = (a+b)/2$ et $x_2 = b$. On trouve dans ce cas

$$\lambda_0 = \lambda_2 = \frac{b-a}{6} \quad \& \quad \lambda_1 = \frac{2(b-a)}{3}$$

  (il est facile de résoudre le système de Cramer (5.15) dans ce cas) et la formule approchée devient

(5.19)
$$I_{\rm app}[f;[a,b]] = \frac{b-a}{6}\,\Big(f(a) + 4\,f\Big(\frac{a+b}{2}\Big) + f(b)\Big).$$

  Cette méthode est dite *méthode de Simpson*[7]. Un miracle se produit encore ici : la formule (5.17) est encore valide (comme on le vérifie aisément) pour les fonctions polynomiales de degré 3 (elle est fausse par contre pour les fonctions polynomiales de degré 4).

7. Ainsi dénommée en référence au mathématicien et astrologue britannique Thomas Simpson (1710-1761) ; de fait, elle avait été déjà introduite par Johannes Kepler deux siècles auparavant.

– Si $M = 3$, on introduit, toujours par souci de symétrie, les 4 points $x_0 = a$, $x_1 = a + (b-a)/3$, $x_2 = a + 2(b-a)/3$, $x_3 = b$. Le calcul des coefficients (facile à mener en résolvant un système de Cramer à 4 inconnues) conduit à

$$\lambda_0 = \lambda_3 = \frac{b-a}{8} \quad \& \quad \lambda_1 = \lambda_2 = 3\,\frac{b-a}{8}.$$

La formule approchée est donc dans ce cas

$$I_{\rm app}[f;[a,b]] = \frac{b-a}{8}\Big(f(a) + 3\Big(f\Big(\frac{2a+b}{3}\Big) + f\Big(\frac{a+2b}{3}\Big)\Big) + f(b)\Big). \tag{5.20}$$

C'est la *méthode à quatre points*. La formule (5.17) cesse d'être vraie pour les fonctions polynomiales de degré 4 et au delà.

Toutes les méthodes présentées ici, dans lesquelles le maillage est un maillage régulier (ou encore à pas constant) entrent dans la catégorie des méthodes dites de *méthodes de Newton-Cotes*[8].

Pour contrôler l'erreur dans une telle méthode, on utilise la formule de Taylor avec reste intégral (en a), qui assure que, si f est de classe C^∞, on peut écrire

$$f(x) = \mathrm{Taylor}_p[f;a](x) + \frac{1}{p!}\int_a^x (x-t)^p f^{(p+1)}(t)\,dt,$$

où $\mathrm{Taylor}_p[f;a]$ est le polynôme de Taylor de f à l'ordre p en a, soit

$$\mathrm{Taylor}_p[f;a] = \sum_{k=0}^{p} \frac{f^{(k)}(a)}{k!}\,(x-a)^k.$$

Si on note $(x-t)_+ := \sup(x-t,0)$, l'erreur $E[f;[a,b]]$ commise entre $I[f;[a,b]]$ et $I_{\rm app}[f;[a,b]]$ dans une formule de Newton-Cotes est celle que l'on commet avec la fonction

$$x \in [a,b] \mapsto \frac{1}{p!}\int_a^b (x-t)_+^p f^{(p+1)}(t)\,dt.$$

Cette erreur s'exprime aussi (si l'on utilise le théorème de Fubini) comme

$$E[f;[a,b]] = \frac{1}{p!}\int_a^b f^{(p+1)}(t)\,E\Big[x \mapsto (x-t)_+^p;[a,b]\Big]\,dt$$

En particulier, en utilisant la formule de la moyenne[9], on trouve, si la fonction

$$t \in [a,b] \longmapsto E\Big[x \mapsto (x-t)_+^p\Big]$$

garde un signe constant[10], que

$$E[f;[a,b]] = \frac{f^{(p+1)}(\xi)}{p!}\int_a^b E\Big[x \mapsto (x-t)_+^p;[a,b]\Big]\,dt.$$

8. Ces formules sont apparues à l'occasion du travail de relecture par le mathématicien anglais Roger Cotes (1682-1716) des *Principia* d'Isaac Newton.

9. Qui assure $\int_a^b u(t)\,v(t)\,dt = u(\xi)\int_a^b v(t)\,dt$ pour un certain $\xi \in [a,b]$ si u et v sont continues sur $[a,b]$ et v y garde un signe constant.

10. Ce sera le cas dans nos exemples, on le verra.

Le calcul de $E[f:[a,b]]$ (dans tous les cas de figure pour une méthode de Newton-Cotes exacte à l'ordre p) conduit (pour une fonction C^∞ sur $[a,b]$) à une estimation d'erreur du type

$$|E[f;[a,b]]| = \Big|I[f;[a,b]] - I_{\rm app}[f;[a,b]]\Big| \leq C[f] \times (b-a)^{p+2}. \tag{5.21}$$

Ceci résulte du double processus d'intégration impliqué dans le calcul de l'expression

$$\int_a^b E\Big[x \mapsto (x-t)_+^p; [a,b]\Big]\, dt$$

dans laquelle figure déjà sous le premier intégrant la fonction $x \mapsto (x-t)_+^p$. Les calculs peuvent être menés explicitement dans les trois premiers exemples mentionnés (rectangles et trapèzes avec $p=1$, Simpson avec $p=3$) :

– pour la méthode des rectangles, on trouve, pour $t \in [a,b]$,

$$E_{\rm rect}\Big[x \mapsto (x-t)_+^p; [a,b]\Big] = \begin{cases} \frac{(t-a)^2}{2} \text{ si } t \leq \frac{a+b}{2} \\ \frac{(t-b)^2}{2} \text{ si } t \geq \frac{a+b}{2} \end{cases}$$

et, en intégrant sur $[a,b]$, puis en appliquant la formule de la moyenne

$$E_{\rm rect}[f;[a,b]] = f''(\xi) \times \frac{(b-a)^3}{24} \quad (\xi \in [a,b]). \tag{5.22}$$

– pour la méthode des trapèzes, on trouve, pour $t \in [a,b]$,

$$E_{\rm trap}\Big[x \mapsto (x-t)_+^p; [a,b]\Big] = \frac{(t-a)(t-b)}{2}$$

et donc, en intégrant sur $[a,b]$, puis en appliquant la formule de la moyenne

$$E_{\rm trap}[f;[a,b]] = -f''(\xi) \times \frac{(b-a)^3}{12} \quad (\xi \in [a,b]). \tag{5.23}$$

– pour enfin la méthode de Simpson, les calculs sont plus laborieux mais l'on trouve, pour $t \in [a,b]$,

$$E_{\rm Simp}\Big[x \mapsto (x-t)_+^p; [a,b]\Big] = \begin{cases} \frac{(b-t)^3(2b+a-3t)}{12} \text{ si } t \leq \frac{a+b}{2} \\ \frac{((a-t)^3(b+2a-3t)}{2} \text{ si } t \geq \frac{a+b}{2} \end{cases}$$

et donc, en intégrant sur $[a,b]$, puis en appliquant la formule de la moyenne

$$E_{\rm Simp}[f;[a,b]] = -f^{(4)}(\xi) \times \frac{(b-a)^5}{2880} \quad (\xi \in [a,b]). \tag{5.24}$$

On montrerait, si $h := (b-a)/M$, que l'erreur dans la formule de Newton-Cotes à $M+1$ points uniformément répartis dans $[a,b]$ est (en module) en h^{M+3} si M est pair (exemple $M=2$ avec la méthode de Simpson) et seulement en h^{M+2} si M est impair. Ce « gain » lorsque M est pair tient au fait que l'on peut exploiter le point médian $(a+b)/2$ (qui appartient alors au maillage) comme « pivot » pour évaluer l'erreur avec la formule de Taylor. Ceci n'est plus possible lorsque M est impair. La formule à 4 points n'est donc pas plus intéressante (au niveau du contrôle d'erreur) que la formule de Simpson à 3 points.

Le plus grand entier $p = p(M)$ tel que l'erreur dans une méthode de Newton-Cotes à $M+1$ points (appliquée à une fonction C^∞ sur un segment $[a,b]$) soit contrôlée en h^p (où $h := (b-a)/M$) est appelé *ordre de la méthode de Newton-Cotes*. Plus

M augmente, certes plus p augmente, mais aussi plus la méthode devient instable car les coefficients $u_{M,0}, ..., u_{M,M}$ dans

$$\int_a^b f(t)\,dt \simeq \sum_{k=0}^{M} u_{M,k}\, f(x_{M,k})$$

« explosent » avec M, pour la même raison en fait que celle impliquant les effets de bord dans l'interpolation de Lagrange [11] sous-jacente aux méthodes de Newton-Cotes : on ne peut approcher sans risque une fonction dont les dérivées successives s'amplifient près des extrémités de $[a, b]$ par des fonctions polynomiales dont les dérivées d'ordre assez grand sont automatiquement identiquement nulles ! L'idée pratique pour pallier à pareille difficulté consiste à utiliser des *méthodes composites*. On découpe $[a, b]$ en M segments $[a_k, b_k]$ de même longueur (donc de longueur $h = (b-a)/M$) et, sur chacun de ces segments, on utilise une méthode de Newton-Cotes d'ordre p. L'ordre de la méthode composite est alors $p-1$ car il faut multiplier h^p (contrôlant l'erreur sur chaque segment) par $M = (b-a)/h$, ce qui donne une erreur en h^{p-1}.

Le calcul approché $I(h)$ de l'intégrale par le procédé composite associé se présente donc sous la forme

$$I(h) = \int_a^b f(t)\,dt + \alpha h^{p-1} + o(h^{p-1})\,. \tag{5.25}$$

Connaitre explicitement α n'est pas évident, mais si l'on refait le calcul approché en divisant le pas par 2, on trouve (ceci en effet revient à remplacer h par $h/2$ dans (5.25))

$$I(h/2) = \int_a^b f(t)\,dt + \alpha(h/2)^{p-1} + o(h^{p-1}) \tag{5.26}$$

(tous les $o(h^{p-1})$ ont ici été notés de manière identique pour alléger les notations). En soustrayant (5.26) à (5.25), il vient

$$\alpha(h/2)^{p-1} \sim \frac{I(h) - I(h/2)}{2^{p-1} - 1}$$

au voisinage de $h = 0$, ce qui implique donc que l'étude du comportement de

$$h \longmapsto \frac{I(h) - I(h/2)}{(2^{p-1} - 1)(h/2)^{p-1}}$$

lorsque h tend vers 0 nous permet de déterminer explicitement α (au moins de manière approchée) et d'être par là même capable de contrôler l'ordre de grandeur (et le signe si $\alpha \neq 0$) de l'erreur numérique

$$\int_a^b f(t)dt - I(h)\,.$$

Il y a aussi un autre moyen d'exploiter pareille idée, aux fins cette fois d'en tirer une approximation plus efficace (c'est-à-dire convergeant plus rapidement) de l'intégrale inconnue. Ce procédé à été initié assez récemment par L.F. Richardson [12] et est

11. Voir la sous-section 3.2.3.

12. Les travaux du physicien et mathématicien anglais Lewis Fry Richardson (1881-1953) ont été pour une grande part tournés vers les prévisions météorologistes ; c'est dans cette optique qu'a surgi la technique d'extrapolation (et d'accélération de convergence) que nous mentionnons ici.

aujourd'hui très utilisé dans le calcul approché des limites de suites ou des sommes de séries numériques.

On multiplie cette fois la relation (5.26) par 2^{p-1} et l'on soustrait la relation obtenue à la formule (5.25), ce qui donne

$$(1-2^{p-1})\int_a^b f(t)\,dt = I(h) - 2^{p-1}I(h/2) + o(h^{p-1})\,,$$

ou encore

$$\int_a^b f(t)\,dt = \frac{I(h) - 2^{p-1}I(h/2)}{1-2^{p-1}} + o(h^{p-1})\,.$$

Si l'on pose

$$\widetilde{I}(h) := \frac{I(h) - 2^{p-1}I(h/2)}{1-2^{p-1}}\,, \tag{5.27}$$

on constate que l'erreur commise en remplaçant l'intégrale

$$\int_a^b f(t)\,dt$$

par $\widetilde{I}(h)$ est cette fois en $o(h^{p-1})$, alors que l'erreur commise en la remplaçant par $I(h)$ était en $O(h^{p-1})$, d'où un gain significatif dans la vitesse de convergence de l'erreur vers 0.

Le procédé d'extrapolation (et d'accélération de convergence) de Richardson (appliqué ici aux fins de calcul numérique d'intégrales) peut même être itéré si l'on sait *a priori* que

$$\int_a^b f(t)\,dt = I(h) + \alpha_0 h^{p-1} + \alpha_1 h^p + \cdots + \alpha_k h^{p-1+k} + o(h^{p-1+k})\,. \tag{5.28}$$

En écrivant aussi

$$\int_a^b f(t)\,dt = I(h/2) + \alpha_0 (h/2)^{p-1} + \alpha_1 (h/2)^p + \cdots + \alpha_k (h/2)^{p-1+k} + o(h^{p-1+k})$$

et en combinant avec la relation précédente, on trouve

$$\begin{aligned}\int_a^b f(t)\,dt \;=\; & \frac{I(h) - 2^{p-1}I(h/2)}{1-2^{p-1}} \\ & + \alpha_1 \frac{1-1/2}{1-2^{p-1}}\,h^p + \alpha_2 \frac{1-1/4}{1-2^{p-1}}\,h^{p+1} + \cdots + \alpha_k \frac{1-1/2^k}{1-2^{p-1}}\,h^{p-1+k} \\ & + o(h^{p-1+k})\,,\end{aligned}$$

relation que l'on peut encore écrire

$$\int_a^b f(t)\,dt = \widetilde{I}(h) + \widetilde{\alpha}_0 h^{\tilde{p}-1} + \widetilde{\alpha}_1 h^{\tilde{p}} + \cdots + \widetilde{\alpha}_{\tilde{k}} h^{\tilde{p}-1+\tilde{k}} + o(h^{\tilde{p}-1+\tilde{k}}) \tag{5.29}$$

avec $\tilde{p} := p+1$, $\tilde{k} := k-1$ et

$$\widetilde{I}(h) := \frac{I(h) - 2^{p-1}I(h/2)}{1-2^{p-1}}\,.$$

On remarque que la nouvelle relation obtenue (5.29) est exactement du type de la relation (5.28), ce qui nous permet de réitérer le processus en écrivant

$$\int_a^b f(t)\,dt = \check{I}(h) + \check{\alpha}_0 h^{\check{p}-1} + \check{\alpha}_1 h^{\check{p}} + \cdots + \check{\alpha}_{\check{k}} h^{\check{p}-1+\check{k}} + o(h^{\check{p}-1+\check{k}}) \tag{5.30}$$

avec $\check{p} := \tilde{p} + 1 = p + 2$, $\check{k} := \tilde{k} - 1 = k - 2$ et

$$\check{I}(h) := \frac{\widetilde{I}(h) - 2^{\tilde{p}-1}\widetilde{I}(h/2)}{1 - 2^{\tilde{p}-1}}\,.$$

On est ainsi en position de recommencer, et ainsi de suite jusqu'à ce que l'on ait épuisé le développement limité. L'opération peut ainsi être itérée k fois et conduit à une approximation de l'intégrale

$$\int_a^b f(t)\,dt$$

avec une erreur en $o(h^{p-1+k})$, tous les termes de la partie principale développement jusqu'à cet ordre ayant été « aspirés ». Ceci est d'autant plus important que l'on sait, d'après une formule sommatoire dite formule d'Euler-MacLaurin[13], que le terme d'erreur dans la méthode des trapèzes composite se présente précisément sous la forme

$$\int_a^b f(t)\,dt - I(h) = \alpha_0 h^2 + \alpha_2 h^4 + \cdots + \alpha_{2(p-1)} h^{2(K-1)} + O(h^{2K})\,, \tag{5.31}$$

ce qui permet de lui appliquer le processus d'extrapolation de Richardson de manière itérative comme ci-dessus. Le procédé décrit précédemment et transposé à cet exemple particulier est ce que l'on appelle aujourd'hui la *méthode de Romberg*[14]. C'est une méthode très utilisée du fait de son efficacité.

REMARQUE 5.1 (pourquoi parler d'« extrapolation » ?). On peut naturellement se poser la question suivante : pourquoi parler d'« extrapolation » à propos de la méthode de L. F. Richardson (ou de celle de W. Romberg qui s'en déduit) ? La raison en est que la quantité

$$\frac{I(h) - 2^{p-1} I(h/2)}{1 - 2^{p-1}}$$

13. Cette formule, établie par Leonhard Euler et le mathématicien écossais Colin MacLaurin autour de 1735, stipule en effet que, si f est de classe C^{2K} sur le segment $[0, M]$,

$$\begin{aligned}\int_0^M f(t)\,dt \quad &= \quad \Big[\frac{f(0)}{2} + \sum_{j=1}^{M-1} f(j) + \frac{f(M)}{2}\Big] - \sum_{k=1}^{K} \frac{b_{2j}}{(2j)!}\Big(f^{(2j-1)}(M) - f^{(2j-1)}(0)\Big) \\ &\quad + \int_0^M f^{(2K)}(t)\frac{B_{2K}(t-[t])}{(2K)!}\,dt\,,\end{aligned}$$

où les $b_2, ..., b_{2K}$ sont des nombres rationnels indépendants de f (les *nombres de Bernoulli* d'indices $2, 4, ..., 2K$) et B_{2K} un certain polynôme (lui aussi indépendant de f), dit polynôme de Bernoulli d'indice $2K$. Il suffit d'appliquer cette formule à la fonction

$$t \longmapsto f(a + ht)\,, \quad h = \frac{b-a}{M}$$

(définie sur $[0, M]$) pour en déduire le résultat (5.31) voulu. Vous démontrerez plus tard (voir [**Y2**], section 5.2.4) la formule importante d'Euler-MacLaurin ; il s'agit d'une formule dans le même esprit que la *formule de Taylor avec reste intégral*.

14. Du nom du mathématicien allemand Werner Romberg (1909-2003) qui l'introduisit dans ses travaux en intégration numérique.

introduite dans (5.27) s'interprète comme une « extrapolation [15] » de la fonction I à partir de ses valeurs en h et $h/2$. Cette idée n'est pas sans rapport avec l'identité de Bézout rappelée dans la sous-section 3.1.2 : si N_1 et N_2 (par exemple $N_1 = 2$ et $N_2 = 3$) sont deux nombres premiers entre eux, les polynômes

$$1 + X + \cdots + X^{N_1-1} \quad \text{et} \quad 1 + X + \cdots + X^{N_2-1}$$

sont premiers entre eux et l'on peut trouver deux polynômes U_1 et U_2 tels que

$$U_1(X)(1 + \cdots + X^{N_1-1}) + U_2(X)(1 + \cdots + X^{N_2-1}) = 1\,,$$

soit aussi, en multipliant par $X - 1$,

$$U_1(X)(X^{N_1} - 1) + U_2(X)(X^{N_2} - 1) = X - 1\,.$$

Si $(u_k)_{k\in\mathbb{Z}}$ est une suite de nombres complexes indexée par $\mathbb{Z}$, cette relation permet de déduire (essayez d'expliciter comment) un procédé de calcul de la suite de décalages $(u_{k+1} - u_k)_{k\in\mathbb{Z}}$ à partir des deux suites de décalages $(u_{k+N_1} - u_k)_{k\in\mathbb{Z}}$ et $(u_{k+N_2} - u_k)_{k\in\mathbb{Z}}$, permettant ainsi d'extrapoler les différences successives entre les u_k depuis les différences prises entre les mêmes u_n, mais avec des sauts de $N_1 - 1$ indices, et celles prises avec des sauts de $N_2 - 1$ indices.

5.2.3. Formules de Newton-Cotes et schémas numériques. Nous donnons ici deux exemples de construction de schémas numériques implicites ou explicites à partir des formules à un point (rectangle) ou deux points (trapèzes).

Exemple 5.3 (avec la méthode des rectangles : schéma d'*Euler modifié*). La formule à un point (rectangles) conduit par exemple à

$$y_{h,k+1} - y_{h,k} \simeq h\, f\Big(t_k + \frac{h}{2}\,,\, y\big(t_k + \frac{h}{2}\big)\Big).$$

En combinant avec

$$y\big(t_k + \frac{h}{2}\big) = y(t_k) + \frac{h}{2}\, y'(t_k) + o(h) = y(t_k) + \frac{h}{2}\, f\big(t_k, y(t_k)\big) + o(h),$$

on voit que le choix de $\Phi[f]$ est alors

$$\Phi[f] \ : (t, y, h) \longmapsto f\Big(t + \frac{h}{2}\,,\, y + \frac{h}{2} f(t, y)\Big). \tag{5.32}$$

On retrouve le *schéma d'Euler modifié*. Voici le code `MATLAB` pour ce schéma, la fonction `f` (de deux variables `t` et `y`) étant déclarée en ligne par

```
>> f = inline ('expression MATLAB en t et y', 't','y');
```

l'instant initial étant `t0`, la condition initiale `y(t0)=y0`, l'intervalle d'étude `[t0,t0+T]`, et le pas choisi étant ici `T/N`) :

```
function [t,y] = Eulermodif(t0,y0,T,N,f)
h = T/N;
t = [t0];
y = [y0];
for k=1:N
```

15. « Extrapoler » une fonction à partir de ses valeurs en des points donnés x_k, c'est pouvoir reconstituer ses valeurs en d'autres points que les points x_k où la fonction est *a priori* donnée ; ce n'est pas la même chose qu'« interpoler » une fonction f aux points x_k par une fonction plus simple $\widetilde{f}$ appartenant à une classe connue (par exemple une fonction polynômiale), même si la connaissance des $\widetilde{f}(x)$ pour x différent des x_k fournit une manière d'extrapoler f depuis ses valeurs aux points x_k.

```
  taux = t(k) + h/2 ;
  A = f(t(k),y(k));
  yfinal = y(k) + h*f(taux,y(k) + h*A/2);
  t = [t,t(k)+h];
  y = [y,yfinal];
end
plot(t,y)
```

Exemple 5.4 (avec la méthode des trapèzes : schéma explicite d'*Heun* et implicite de *Craig-Nicholson*). La formule à 2 points (trapèzes) conduit, elle, à

$$y_{h,k+1} - y_{h,k} \simeq \frac{h}{2}\Big(f\big(t_k, y(t_k)\big) + f\big(t_{k+1}, y(t_{k+1})\big)\Big). \tag{5.33}$$

En combinant avec

$$\begin{aligned} y(t_{k+1}) &= y(t_k + h) = y(t_k) + h\, y'(t_k) + o(h) \\ &= y(t_k) + f(t_k, y(t_k))\, h + o(h), \end{aligned} \tag{5.34}$$

on voit que le choix de $\Phi[f]$ dans (5.10) qui est adapté à cette méthode des trapèzes est

$$\Phi[f] \ : (t, y, h) \longmapsto \frac{1}{2}\Big(f(t, y) + f\big(t + h, y + hf(t, y)\big)\Big). \tag{5.35}$$

C'est le *schéma de Heun*[16] *explicite*. Voici le code `MATLAB` pour ce schéma, la fonction `f` (de deux variables `t` et `y`) étant déclarée en ligne par

```
>> f = inline ('expression MATLAB en t et y', 't','y');
```

l'instant initial étant `t0`, la condition initiale `y(t0)=y0`, l'intervalle d'étude `[t0,t0+T]`, et le pas choisi étant ici `T/N`) :

```
function [t,y] = Heun(t0,y0,T,N,f)
h = T/N;
t = [t0];
y = [y0];
for k=1:N
  A1 = f(t(k),y(k));
  A2 = f(t(k)+h,y(k)+h*A1);
  yfinal = y(k) + (h/2)*(A1+A2);
  t = [t,t(k)+h];
  y = [y,yfinal];
end
plot(t,y)
```

Mais on peut également ne pas utiliser l'approximation (5.34) et utiliser directement (5.33) pour générer le schéma (implicite cette fois)

$$\frac{y_{h,k} - y_{h,k-1}}{h} = \frac{1}{2}\Big(f(t_k, y_{h,k-1}) + f(t_k + h, y_{h,k})\Big) \tag{5.36}$$

On pose alors

$$\Psi \ : (t, y, \xi, h) \longmapsto \frac{1}{2}\Big(f(t, y) + f(t + h, \xi)\Big)$$

16. Du nom du mathématicien allemand Karl Heun (1859-1929) qui l'introduisit.

pour obtenir le schéma implicite du type (5.11) dit *schéma de Craig-Nicholson implicite*[17].

Pour une étude plus approfondie de la résolution des équations différentielles ordinaires (EDO) par des schémas numériques (et l'introduction des notions d'ordre, de convergence, de consistance et de stabilité), on renvoie au chapitre 6 de [**Y2**]. On y trouvera également (Exemple 6.19, sous-section 6.3.3 de [**Y2**]) une présentation des *méthodes de Runge-Kutta* introduites au niveau de cette UE dans le cadre des projets. Comme la formule de Simpson (dont la construction de ce type de schéma numérique dérive, sur la base de la formule (5.12)) permet un calcul approché d'intégrale avec une erreur d'ordre $5 = 2+3$, le schéma numérique de Runge-Kutta conduit à une erreur en $h^5/h = h^4$, si h désigne le pas. Pour être complet dans ce cours, nous esquissons ici le principe du schéma numérique de Runge-Kutta le plus classique[18]. Étant donnée l'équation différentielle

$$y'(t) = f(t, y(t))$$

(sur $[t_0, t_0+T]\times\mathbb{R}$), le principe de schéma est, une fois le pas $h > 0$ fixé, de calculer les nombres $y_{h,k}$, $k = 1, ..N$, tels que $y_{h,0} = y_0$ et

$$y_{h,k+1} - y_{h,k} = h\Phi[f](t_k, y_{h,k}, h) \simeq \int_{t_0+hk}^{t_0+h(k+1)} f(t, y(t))\, dt,\ \ k = 0, 1, 2, ...$$

($t_k = t_0 + kh$, $k = 0, 1, 2, ...$) de proche en proche suivant le principe décrit sur les quatre étapes suivantes :

$$\begin{aligned}
&t_{k,1} := t_0 + kh = t_k\\
&t_{k,2} := t_{k,3} = t_{k,1} + h/2\\
&t_{k,4} := t_{k,2} + h/2 = t_{k,3} + h/2 = t_{k,1} + h = t_0 + (k+1)h\\
&A_{k,1} := f(t_{k,1}, y_{h,k})\\
&A_{k,2} := f(t_{k,2}, y_{h,k} + h\, A_{k,1}/2) = f(t_{k,1} + h/2, y_{h,k} + h\, A_{k,1}/2)\\
&A_{k,3} := f(t_{k,3}, y_{h,k} + h\, A_{k,2}/2) = f(t_{k,1} + h/2, y_{h,k} + h\, A_{k,2}/2)\\
&A_{k,4} := f(t_{k,3}, y_{h,k} + h\, A_{k,3})\\
&y_{h,k+1} := y_{h,k} + \frac{h}{6}\big(A_{k,1} + 2A_{k,2} + 2A_{k,3} + A_{k,4}\big).
\end{aligned} \tag{5.37}$$

Du point de vue algorithmique, ceci peut être codé ainsi sous `MATLAB` une fois que la fonction de deux variables `f` : $(t, y) \mapsto f(t, y)$ a été déclarée `'inline'` au préalable par

```
>> f = inline ('expression MATLAB en t et y', 't','y');
```

Le but est ici de construire les valeurs approchées de la solution de l'EDO `y'=f(t,y)` aux points `t0 + (k-1)*T/N`, `k=1,...,N+1`, lorsque la condition initiale est `y(t0) =y0` (`t0` désigne ici l'instant initial et `T` la longueur du segment `[t0,t0+T]` sur lequel se trouve échantillonnée la solution `y`) :

17. On doit cette méthode (initialement introduite pour la résolution de l'équation de la chaleur) à la mathématicienne britannique Phillis Nicholson (1917-1968) et à son compatriote le physicien John Craig (1916-2006).

18. Cette démarche a été introduite par Carl Runge, mathématicien et physicien allemand (1856-1927) et développée numériquement par le mathématicien allemand Martin Kutta (1867-1944), connu aussi pour ses travaux en aérodynamique.

```
function [t,y] = RungeKutta(t0,y0,T,N,f)
% on declare d'abord le pas :
h = T/N ;
% on initialise le vecteur des temps :
t = [t0];
% on initialise le vecteur de y
% (y(t0)=y0 figure ici la condition initiale) :
y = [y0];
% On redige maintenant la boucle de calcul suivant
% les instructions decrites ci-dessus en (5.37) :
for k=1:N
  % on declare les points intermediaires entre t0+(k-1)*h et t0+k*h :
  t1 = t(k);
  t2 = t1 + h/2 ;
  t3 = t1 + h/2 ;
  t4 = t1 + h ;
  % on calcule les quatre valeurs auxiliaires A1, A2, A3, A4 :
  A1 = f(t1, y(k));
  A2 = f(t2, y(k) + h*A1/2);
  A3 = f(t3, y(k) + h*A2/2);
  A4 = f(t3, y(k) + h*A3) ;
  % on calcule y(t0+(k+1)*h) comme indique :
  yfinal  = y(k) + (h/6) *(A1 + 2*A2 + 2*A3 + A4) ;
  % on complete les vecteurs des temps et des y :
  t  = [t,t4] ;
  y  = [y,yfinal] ;
end
plot(t,y)
```

Pareil code pourra être mis en œuvre dans le cadre d'un projet et par exemple implémenté sur le système de Lotka-Volterra (proie-prédateur) introduit dans la section 5.3 suivante. On pourra comparer l'efficacité de ce code avec ceux des codes introduits précédemement pour les schémas d'Euler modifiée (5.32) et de Heun (5.34) (tous deux d'ordre $2 = 3 - 1$ puisque méthode des rectangles et méthodes des trapèzes sont toutes deux des méthodes d'intégration numérique pour lesquelles l'erreur est en h^3 , voir respectivement (5.22) et (5.23) dans la section précédente).

La méthode ainsi construite, directement issue de la méthode de Newton-Cotes à trois points (Simpson) [19] dont on sait qu'elle induit une erreur en $O(h^5)$, est d'ordre $4 = 5-1$ (il y a une division par h une fois approchée l'intégrale). Ce solveur explicite d'ordre 4 est très utilisé dans la pratique : c'est la méthode de Runge-Kutta dite « classique ».

Les schémas numériques d'Euler explicite ou implicite conduisent à une erreur contrôlée en $h^{2-1} = h$. En revanche, les schémas numériques de Heun et Craig-Nicholson (fondées sur l'utilisation de la formule des trapèzes, qui est d'ordre 3,

19. On retrouve bien, avec l'approximation de l'intégrale

$$\int_{t_0+kh}^{t_0+(k+1)h} f(t, y(t))\, dt \simeq \frac{h}{6}\big(A_{k,1} + 2A_{k,2} + 2A_{k,3} + A_{k,4}\big), \quad k = 0, 1, ...$$

les coefficients $1/6, 2/6 + 2/6 = 2/3, 1/6$ de la formule de Simpson (5.19).

voir (5.23)) conduisent à une erreur contrôlée en $h^{3-2} = h^2$. Il en est de même pour le schéma numérique correspondant à Euler modifié (reposant sur l'utilisation de la formule des rectangles, qui est aussi une méthode d'ordre 3, voir (5.22)). Le schéma de Runge-Kutta évoqué ci-dessus conduit, on l'a vu, à une erreur en $h^{5-1} = h^4$.

5.3. Un modèle de système autonome : le modèle *proie-prédateur*

Les systèmes différentiels du type

$$\begin{aligned} \frac{dx}{dt} &= F(x(t), y(t)) \\ \frac{dy}{dt} &= G(x(t), y(t)) \end{aligned} \tag{5.38}$$

(on les appelle *autonomes* car F et G sont fonctions de x et y, mais non du temps t) induisent des notions (pouvant sembler *a priori* différentes, bien que ce ne soit pas en fait le cas) de *trajectoire* et de *plan de phase*. Le plan de phase est cette fois la représentation dans le plan (plus précisément dans l'ouvert U de $\mathbb{R}^2$ où sont définies et continues les deux fonctions F et G) des *trajectoires*, c'est-à dire ici des courbes paramétrées (ce ne sont plus des graphes comme dans le cas $\mathbf{N} = 1$)

$$t \in I \mapsto (x(t), y(t))$$

solutions du système différentiel autonome (5.38). Le théorème de Cauchy-Lipschitz impose que par chaque point (x_0, y_0) de U ne passe qu'une et une seule trajectoire. Une trajectoire ne saurait d'autre part admettre de point double. Cependant, certaines trajectoires peuvent être fermées (être des lacets sans points doubles), on les appelle des *orbites*. Les trajectoires réduites à un singleton sont dites *stationnaires* : si le point initial est ce singleton, alors on n'en bouge pas !

Le modèle *proie-prédateur*[20] (ou de *Lotka-Volterra*[21]) est aujourd'hui un modèle de système autonome très classique en dynamique des populations et dans nombre de questions relevant de questions appliquées. Il permet d'introduire le concept important de *stabilité*, ce que nous nous contenterons de faire ici de manière heuristique. Le modèle (continu) de ce système d'évolution est le suivant : a, b, c, d désignant quatre paramètres réels strictement positifs, il s'agit du système différentiel

$$\begin{aligned} x'(t) &= x(t)\,(a - by(t)) \\ y'(t) &= y(t)\,(-c + dx(t)). \end{aligned} \tag{5.39}$$

(ici $U = \mathbb{R} \times \mathbb{R}^2$). L'interprétation correspondant à ce modèle est la suivante : deux types de population cohabitent. La première (dont l'évolution est matérialisée en termes de proportion par x est l'effectif des *proies*) se développe exponentiellement en $\exp(at)$; la seconde (effectif des *prédateurs*, matérialisée en termes de proportion par y) s'éteint exponentiellement en $\exp(-ct)$; le facteur b s'interprète comme la *pression de prédation*, le facteur d comme l'*accessibilité des proies*. Ces modèles se retrouvent couramment en épidémiologie et, bien sûr, les questions de propagation

20. On se reportera à [**Vial**] pour une présentation plus détaillée (enrichie de l'aspect numérique) dont je me suis ici beaucoup inspiré. On pourra aussi consulter avec profit le document ressources mis en ligne par le Ministère de l'Education Nationale à l'appui des nouveaux programmes de Spécialité ≪ Mathématiques ≫ en Terminale S [**TermS**], pages 54 à 59.

21. Le mathématicien et statisticien autrichien Alfred James Lotka (1880-1949) et le mathématicien et physicien italien Vito Volterra (1860-1940) l'introduisirent vers 1925, ouvrant la voie à la dynamique des populations.

de virus en sécurité informatique font qu'on les croise également en informatique et sécurité réseaux. Les modèles plus réalistes sont les modèles perturbés où l'on suppose que le taux de croissance x des proies diminue lorsque la population augmente (du fait de contraintes environnementales ou de subsistance par exemple). Le modèle du système d'évolution est alors

$$\begin{aligned} x'(t) &= x(t)\,(a - \epsilon x(t) - by(t)) \\ y'(t) &= y(t)\,(-c + dx(t)), \end{aligned} \tag{5.40}$$

où ϵ est un cinquième paramètre. L'attaque de ce type de problème se fait numériquement par discrétisation. Nous utiliserons ici le schéma d'Euler explicite. Il faut noter cependant que le système (5.39) présente deux points stationnaires (trajectoires réduites à un point), à savoir $(0,0)$ et $(c/d, a/b)$, de nature différente :

- si l'on perturbe l'origine en initiant la trajectoire en un point voisin (x_0, y_0), on constate que la nouvelle trajectoire initiée en (x_0, y_0) s'éloigne de l'origine (on dit que l'origine est un point d'équilibre *instable*) ;
- au contraire, si l'on perturbe le point $(c/d, a/b)$ en initiant la trajectoire en un point voisin, la nouvelle trajectoire reste une orbite autour du point $(c/d, a/b)$; on dit que $(c/d, a/b)$ est un point d'équilibre *stable*.

Les notions d'*instabilité* et de *stabilité* pour les équilibres sont fondamentales dans les questions relevant de l'analyse qualitative des systèmes différentiels autonomes (plus généralement des équations différentielles $Y'(t) = F(t, Y(t))$, autonomes ou non). Au voisinage d'un point d'équilibre stable, l'étude d'un système autonome de type (5.39) peut être approchée par celle du système linéaire autonome obtenu en remplaçant les seconds membres $(x, y) \mapsto x(a - by)$ et $(x, y) \mapsto y(-c + dx)$ par leurs polynômes de Taylor à l'ordre 1 (donc des fonctions affines) au voisinage du point d'équilibre $(0, 0)$ ou $(c/d, a/b)$. Ceci résulte d'un théorème majeur dans l'étude des systèmes dynamiques, le théorème de Lyapunov. C'est aussi par ce biais que l'on peut constater qu'un point d'équilibre (tel $(0, 0)$ pour le système (5.39)) est instable : les valeurs propres de la matrice $(2, 2)$ du système linéarisé sont ici deux nombres réels non nuls de signe opposés, ce qui correspond à une configuration de *point-selle* et donc à une situation d'équilibre instable (certaines trajectoires sont attirées, d'autres sont repoussées). Sans chercher à linéariser le problème au voisinage d'un des deux points d'équilibre, on pourra également envisager l'approche numérique à la résolution des systèmes autonomes (5.39) ou (5.40) en utilisant les schémas numériques décrits dans la Section 5.2 dans le cadre $\mathbf{N} = 1$, mais qu'il est aisé de transporter au cas $\mathbf{N} = 2$ (par exemple Euler explicite).

Voici le synopsis de la méthode d'Euler explicite conduisant au tracé des trajectoires autour du point $x = c/d$, $y = a/b$ correspondant au point d'équilibre stable $(c/d, a/b)$. On pose

$$u(t) = x(t) - c/d \quad \& \quad v(t) = y(t) - a/b,$$

d'où le système autonome à étudier numériquement (déduit de (5.39) :

$$u'(t) = -(bc/d)\,u(t) - b\,u(t)v(t) \qquad v'(t) = (ad/b)\,u(t) + d\,u(t)v(t).$$

```
function [U,V] = proiestable(uinit,vinit,pas,a,b,c,d,N);
% [U,V]=proiestable(uinit,vinit,pas,a,b,c,d,N);
U=uinit;
V=vinit;
```

```
u=uinit;
v=vinit;
for i=1:N;
    u=u+pas*(-(b*c/d)*v -b*u*v);
    v=v+pas*((a*d/b)*u +d*u*v);
    U=[U,u];
    V=[V,v];
end;
```

Voici maintenant le synopsis de la méthode d'Euler explicite conduisant au tracé des trajectoires autour du point $x = c/d$, $y = a/b$ correspondant au point d'équilibre instable $(0, 0)$:

```
function [U,V] = proieinstable(uinit,vinit,pas,a,b,c,d,N);
% [U,V]=proieinstable(uinit,vinit,pas,a,b,c,d,N);
U=uinit;
V=vinit;
u=uinit;
v=vinit;
for i=1:N;
    u=u+pas*(a*u -b*u*v);
    v=v+pas*(-c*v +d*u*v);
    U=[U,u];
    V=[V,v];
end;
```

Ce synopsis correspond à la résolution du système approché :

$$\begin{aligned} &\frac{u_{k+1}-u_k}{h} = u_k(a - bv_k) \\ &\frac{v_{k+1}-v_k}{h} = v_k(-c + du_k), \\ &k = 0, ..., N, \end{aligned} \tag{5.41}$$

initiée en u_0 =`uinit` et v_0 =`vinit`. Pour ce même système, la méthode d'Euler implicite conduirait à

$$\begin{aligned} &\frac{u_k-u_{k-1}}{h} = u_k(a - bv_k) \\ &\frac{v_k-v_{k-1}}{h} = v_k(-c + du_k), \\ &k = 1, ..., N, \end{aligned} \tag{5.42}$$

avec les mêmes conditions initiales u_0 et v_0, l'expression de (u_k, v_k) en termes de (u_{k-1}, v_{k-1}) à partir des relations (5.42) étant cette fois *implicite* et non plus *explicite*. Que ce soit pour le modèle discret ou pour le modèle continu, on peut dresser un tableau de variations pour prédire l'évolution du processus depuis une position donnée. Le plan est ainsi partitionné en neuf regions dans lesquelles on précise les sens de variation de x et y. Il y a attraction vers $(0, 0)$ le long de l'axe des y, répulsion le long de l'axe des x. Le calcul numérique (*via* Euler explicite) [22] fait apparaitre des cycles lorsque l'on initie le processus discret à (x_0, y_0) avec $x_0 > 0$

22. Avec les routines explicitées ci-dessus sous `MATLAB`. On suggère les valeurs numériques $a = 3$, $b = 1$, $c = 2$, $d = 1$, le pas `pas` h étant pris égal à .05.

et $y_0 > 0$. Les cycles deviennent cependant des cycles « épais » si (x_0, y_0) s'écarte de plus en plus du point d'équilibre stable $(c/d, a/b)$.

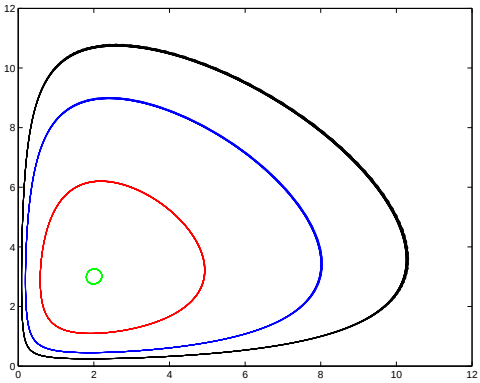

FIGURE 5.1. Quelques trajectoires obtenues avec Euler explicite pour (5.39) ($a = 3$, $c = 2$, $b = d = 1$, `pas`$= .05$)

Ce que l'on observe ici numériquement peut, même au niveau L2, être justifié théoriquement. Si $t \mapsto (x(t), y(t))$ est solution de (5.39) et que $x(t)y(t)$ reste non nul, on a

$$\frac{x'(t)}{x(t)}(-c + dx(t)) = \frac{y'(t)}{y(t)}(a - by(t)).$$

Ceci s'écrit encore

$$c\frac{x'(t)}{x(t)} + a\frac{y'(t)}{y(t)} = dx'(t) + by'(t).$$

En intégrant, on trouve

$$\log |x(t)|^c |y(t)|^a = d\, x(t) + b\, y(t) + \text{const}.$$

On obtient ainsi une intégrale première du système différentiel (5.39). Si on fixe $x(t) = x \neq 0$ et la constante const, il apparait qu'il existe un unique $y \in]0, a/b[$ tel que

$$\log y^a + c \log x = dx + by + \text{const} \Longrightarrow a \log y - by = \text{const} + dx - c \log x.$$

Cette remarque peut être utilisée comme point de départ pour montrer (en exploitant le tableau de variation) qu'une trajectoire initiée en un point (x_0, y_0) avec $x_0 > 0$ et $y_0 > 0$ est automatiquement un cycle.

Au voisinage du point $(c/d, a/b)$ ($x_n = c/d + u_n$, $y_n = a/b + v_n$, u_n, v_n voisins de $(0, 0)$), l'étude du système (5.39) linéarisé se ramène à celle de

$$\begin{aligned} u_{n+1} &= u_n - \tau \frac{bc}{d}\, v_n \\ v_{n+1} &= v_n + \tau \frac{ad}{b}\, u_n. \end{aligned} \tag{5.43}$$

Ici, on se ramène à la résolution d'une équation aux différences à deux pas pour trouver x_n et y_n explicitement. On retrouve ici l'algèbre linéaire et le calcul matriciel (équation caractéristique). On fait une étude similaire au voisinage de l'autre point d'équilibre (instable cette fois), $(0, 0)$.

ANNEXE A

TP1 : prise en main des logiciels `Maple` et `MATLAB`

Cette première séance de TP a essentiellement pour objectif de vous familiariser avec le logiciel de calcul symbolique `Maple12`. En fin de séance, vous aurez aussi un premier contact avec le logiciel (de calcul scientifique cette fois) `MATLAB`, ce qui vous permettra au moins juste d'appréhender la signification de l'acronyme : `MATrixLABoratory`.

EXERCICE A.1. Pour commencer le travail :

(1) Connectez vous sur une des machines, *via* votre `login` et votre mot de passe, exactement comme vous vous connecteriez à votre ENT (par exemple pour aller sur le serveur `Ulysse`).

(2) Une fois que vous avez accès à votre répertoire, mettez vous dans `Documents` et créez là un dossier `N1MA3003`. Placez vous ensuite dans ce nouveau dossier pour y créer deux sous-dossiers `TPMaple12` et `TPMATLAB`.

(3) Utilisez le navigateur web pour vous placer sur le site
`http://www.math.u-bordeaux1.fr/~yger/initiationMAPLE`
et téléchargez (dans le répertoire `TPMaple12` que vous venez de créer) le fichier `premierspasMAPLE.mw` ; n'essayez pas pour l'instant d'ouvrir ce fichier (vous l'ouvrirez ultérieurement sous l'environnement `Maple12`).

(4) Toujours avec le navigateur web, placez vous sur le site
`http://www.math.u-bordeaux1.fr/~yger/initiationMATLAB`
et téléchargez tous les fichiers (`.m`) y figurant dans votre dossier `TPMATLAB` ; n'essayez pas pour l'instant d'ouvrir ces fichiers (vous les ouvrirez ultérieurement sous l'environnement `MATLAB`).

EXERCICE A.2 (préparation de l'environnement `Maple12`). En utilisant votre fenêtre de terminal, lancez `Maple12` en tapant la commande `xmaple12`. Une fois le logiciel ouvert, deux opérations préliminaires sont à effectuer pour préparer votre environnement une fois pour toutes.

- Vous allez être amenés à travailler sous `Maple` toujours en mode *Worksheet*, donnant accès, contrairement au mode *Document*, à toute la puissance du logiciel. Pour que `Maple` s'ouvre tout le temps dans votre environnement sous ce mode *Worksheet*, suivez l'arbre :

  $$Tools \to Options \to Interface \to Default\ format... \to Worksheet$$

  Cliquez sur `Apply globally` pour valider ces instructions.
- Il est de loin préférable (pour limiter les erreurs) de travailler en mode `Text [C-Maple Input]` plutôt qu'en mode `Math`. Pour faire en sorte que ceci soit automatique dès que vous ouvrez le logiciel, suivez l'arbre :

  $$Tools \to Options \to Display \to Input\ display \to Maple\ notation$$

Gardez *2-D Math notation* en revanche pour l'affichage en sortie (*Output display*). Validez encore par `Apply globally`.

Les instructions que vous allez effectuer après le prompt seront maintenant (comme le prompt) en rouge. Vérifiez le. Ouvrez (en vous mettant sous l'onglet `file`) le fichier `premierspasMAPLE.mw` que vous avez mis dans votre répertoire `TPMaple12`. Ce fichier a été rédigé sous `Maple10`, mais vous devez pouvoir l'ouvrir en mode *Worksheet* sous `Maple12` (ce mode reprenant la syntaxe des versions antérieures).

Dans une des instructions (1) à (4), remplacer le [;] final par [:] . Que constatez vous ? Ligne (2), remplacez 2 par un entier `N` que vous choisirez au hasard. Constat ? Refaites l'opération sur la même ligne en remplaçant `N` par `N.0`. Comment expliquez vous ce qui se passe ? Fermez le fichier `premierspasMAPLE.mw` sans enregistrer les modifications que vous avez été amenés à faire dans cet exercice.

EXERCICE A.3 (le rôle de quelques commandes clef).

(1) Ouvrez sous `Maple12` une feuille de travail vierge (onglet *File* → *New*). Tapez les instructions suivantes et dégager le rôle des commandes `;`, `:`, `%`, `:=` (comparez avec `=`), `?`, `restart` (derrière `#`, on a fait figurer des commentaires) :

```
> P:= 1*2*3*4*5 ;
> S:= 1+2+3+4+5 ;
> P+S ;
> S ;
> A := % ; b := %%% /15 ;
> (P+S)/15 ; #verifier que b=(P+S)/15
> c = 2+3 ;
> c; #Pourquoi n'obtient-t-on pas 5 comme resultat ici ?
> ?%
> p; P;
> restart ;
> p; P #Quel est le role de la commande restart?
```

(2) Tapez les commandes suivantes et expliquez ce ce qui se passe :

```
> u := < 1, 2, 3 > ;
> v := < 4 | 5 | 6 > ;
> whattype (u) ;
> whattype (v) ;
> A := << 1,0,-1> | < 2,3,0 > | < -1,5,3 >> ;
> whattype (A) ;
> A.u ; u.A; A.v; v.A;
```

Déclarez sous `Maple` la matrice

$$A = \begin{pmatrix} 2 & -4 & -1 & -3 \\ 1 & 0 & -5 & 7 \\ -6 & -8 & 0 & 7 \end{pmatrix}$$

(3) Tapez l'instruction

```
> plot (sin(x)/x, x=-12..12);
```

et expliquez ce qui se passe.

EXERCICE A.4 (le changement de base dans les systèmes de numération). Ouvrez une nouvelle feuille de travail vierge sous `Maple`. Apprenez à utiliser les onglets `T` et `[>` pour donner un titre (en texte « interte ») à votre feuille de travail (par exemple : `TP1-exo4`). Utilisez (après avoir étudié dans le *Help* le sens de leur fonction) les routines

```
> Digits := m ;
> convert (N,binary) ;
> convert (evalf(r),binary) ;
> convert (f,binary) ;
```

lorsque `N`, `r`, `f` désignent respectivement un entier naturel, un rationnel strictement positif, un nombre réel flottant strictement positif, pour calculer l'exposant et la mantisse (voir le cours, section 1.2.3) des nombres suivants : $N = 78679$, $r = 89765432/456843$, $f = \pi$, lorsque le nombre `Digits` est fixé d'abord égal à 10, puis à 20. Sauvez votre session (en lui donnant comme nom `TP1-exo4`) dans votre répertoire `TPMaple12`. Ce sera un fichier `.mw`.

EXERCICE A.5 (la génération de fonctions). Ouvrez une nouvelle feuille de travail vierge sous `Maple`. Donnez lui un titre (par exemple `TP1-exo5`). Une fonction d'une variable réelle (par exemple `g`) se déclare sur le modèle par exemple de la fonction $x \mapsto x^2$:

```
> g := x -> x^2 ;
```

Vérifiez le avec cette fonction, puis déclarez les fonctions $h : x \mapsto \sin x/(1+x^2)$, $h \circ g$ et $h \circ h$ (exploiter pour cela les commandes `@` et `subs`). Tracez les graphes de ces fonctions sur $[-1,1]$, $[-10,10]$, puis sur $\mathbb{R}$. Sauvez votre session (en lui donnant comme nom `TP1-exo5`) dans votre répertoire `TPMaple12`.

EXERCICE A.6 (l'utilisation de la palette de gauche « expressions »). Ouvrez une nouvelle feuille de travail vierge sous `MAPLE`. Donnez lui un titre (par exemple `TP1-exo6`).

(1) Calculez la limite en $\pi/4$ de la fonction cosinus, puis une approximation du résultat (avec `Digits:=10`, puis `Digits:=20`).

(2) Calculez la limite en 0 de $x \mapsto (1-\cos x)/x^2$.

(3) Tester ce que répond le logiciel lorsqu'on lui demande de calculer une primitive (intégrale sans bornes précisées) pour

$$x \mapsto \arctan(x),\ x \mapsto \frac{\log x}{1+x},\ x \mapsto e^{-x^2},\ x \mapsto (x^2+1)\log x.$$

Dans quel cas (parmi ces quatre exemples) le résultat est-il réellement concluant et exploitable du point de vue pratique ? Quel est dans ce cas l'outil que le logiciel est parvenu à mettre en œuvre ?

(4) Calculez la somme des entiers de 1 à n et exploiter la commande `factor` pour factoriser cette somme. Evaluer cette somme lorsque $n = 1234$ en utilisant `eval` et `subs`.

(5) Calculez, pour $n \in \mathbb{N}$,

$$\Pi(n) = \frac{\prod_{0<k \text{ pair } \leq 2n} k}{\prod_{0<k \text{ impair } <2n} k},$$

(faites une recherche avec l'aide pour comprendre ce qu'est la fonction Gamma, Γ, que le logiciel introduit ici dans ses réponses) puis évaluez $\Pi(n)/\sqrt{n}$ pour $n = 1000$, $n = 5000$, $n = 10000$. Que semble-t-il se dessiner ? En utilisant toujours la palette de gauche « Expressions » , calculez

$$\lim_{n\to+\infty} \Pi(n)/\sqrt{n}.$$

Sauvez votre session (en lui donnant comme nom `TP1-exo6`) dans votre répertoire `TPMaple12`.

EXERCICE A.7 (manipulation de nombres complexes, formule de Machin). Cet exercice est la mise en pratique d'un exemple qui sera exploité en cours (section 1.4.1 du polycopié). Mais il s'agit ici que vous conduisiez vous même les calculs sous `Maple`. Ouvrez une nouvelle feuille de travail vierge sous `Maple`. Donnez lui un titre (par exemple `TP1-exo7`).

(1) Calculez (avec `Maple`) module et argument des nombres complexes $1 + i$ et $2 + 3i$.

(2) Vérifiez (toujours sous `Maple`) la formule de John Machin (mathématicien anglais, 1680-1751) :

$$(5 + i)^4 = 2(239 + i)(1 + i). \qquad (*)$$

Peut-on se fier par contre à `Maple` pour valider la formule

$$\arctan(1) = 4\arctan(1/5) - \arctan(1/239) \qquad (**)$$

ou faut-il raisonner mathématiquement à partir de la formule (*) validée, elle, grâce au logiciel ? Faites le si nécessaire dans ce cas.

(3) On admet ici que la formule (**) se lit aussi :

$$\frac{\pi}{4} = \arctan(1) = \lim_{n\to+\infty} \sum_{k=0}^{n} \frac{(-1)^k}{2k+1}\Big(4\,(1/5)^{2k+1} - (1/239)^{2k+1}\Big) \qquad (***)$$

car

$$\arctan(x) = \int_0^x \frac{dt}{1+t^2} \quad \forall\, x \in \mathbb{R}$$

(en particulier pour $x = 1/5$ et $x = 1/239$) et que

$$\frac{1}{1+t^2} = \lim_{n\to+\infty} \Big(\sum_{k=0}^{n} (-1)^k t^{2k}\Big) \quad \forall\, t \in]-1, 1[$$

(en particulier pour tout $t \in [0, 1/5] \supset [0, 1/239]$). On pose, pour tout $n \in \mathbb{N}$,

$$u_n := 4\sum_{k=0}^{n} \frac{(-1)^k}{2k+1}\Big(4\,(1/5)^{2k+1} - (1/239)^{2k+1}\Big).$$

Montrez que les deux suites $(u_{2n})_{n\geq 1}$ et $(u_{2n-1})_{n\geq 1}$ sont adjacentes.

(4) Calculez, en utilisant la palette de gauche « Expressions » sous `Maple`, les deux fractions u_{199} et u_{200}, puis leurs écritures avec `Digits=500`. Combien de décimales exactes de π peut-on déduire des résultats affichés ? Justifiez clairement votre réponse en vous appuyant sur un raisonnement mathématique (reposant sur la formule $(***)$ partiellement admise).

Sauvez votre session (en lui donnant comme nom `TP1-exo7`) dans votre répertoire `TPMaple12`.

EXERCICE A.8 (familiarisation avec les routines `solve` et `fsolve`). Ouvrez une nouvelle feuille de travail vierge sous `Maple`. Donnez lui un titre (par exemple `TP1-exo8`).

(1) Résolvez les équations :

$$x^2 + x + 1 = 0,\ x^3 + x + 1 = 0,\ x^2 + 3x + 1 > 0,\ \sin x = 0.$$

Expliquez pourquoi dans les deux premiers cas, on obtient bien toutes les racines complexes, alors que ce n'est manifestement pas le cas dans le dernier cas.

(2) Résolvez les inéquations (cette fois dans $\mathbb{R}$) :

$$x^2 + 3x + 1 > 0,\ x^3 + x + 1 > 0.$$

(3) Dites que répond le logiciel lorsqu'on lui demande de résoudre

$$x^5 - 2x^4 + 3x^2 - 3/2 = 0, \quad x \in \mathbb{C},$$

soit l'instruction :

```
> solve (x^5 - 2*x^4 + 3*x^2 - 3/2 =0,x) ;
```

Est-ce vraiment une surprise ? Dites ce qui se passe par contre avec la commande :

```
> solve(x^5 - 2*x^4 + 3*x^2 - 1.5 = 0,x);
```

Expliquez pourquoi la réaction du logiciel est dans ce cas différente. Que se passe t-il sur cet exemple si l'on remplace la routine `solve` par `fsolve` ? Préciser ce qui se passe si l'on utilise la commande avec option :

```
> fsolve (x^5 - 2*x^4 +3*x^2 - 3/2 = 0,x,complex);
```

Sauvez votre session (en lui donnant comme nom `TP1-exo8`) dans votre répertoire `TPMaple12`.

EXERCICE A.9 (un (tout) premier contact avec `MATLAB`). Fermez l'environnement `Maple` et lancez `MATLAB` avec la commande `matlab` depuis la fenêtre de votre Terminal. Une fois `MATLAB` ouvert, placez vous, en utilisant l'onglet dans le bandeau supérieur, dans votre répertoire `TPMATLAB`.

(1) Ouvrez (en utilisant l'onglet `file`) le fichier `test1.m` et analysez-en la syntaxe. Que devrait être le résultat final ?

(2) Lancez `test1` derrière le prompt de `MATLAB`. Qu'observez vous ? Sachant que le logiciel code les réels en double précision (`binary64`), peut-on prédire à partir de quel seuil d'itérations N le résultat de `test1` n'est plus fiable ? Calculez pour cela $\log_{10}(2^{52})$; qu'observez vous ?

(3) Ouvrez (toujours en utilisant l'onglet `file`) le fichier `PGCD.m`. Essayez d'analyser comment l'algorithme présenté ici traduit la démarche mathématique conduisant au calcul du PGCD (souvenez vous de l'algorithme d'Euclide présenté en MISMI). Quel sous-programme est appelé dans cette procédure ? Testez la commande `PGCD` en le lançant derrière le prompt :

```
>> d=PGCD (a,b);
```

(avec des valeurs des entiers naturels a et $b > 0$ fixées). Testez aussi le sous-programme que cette procédure appelle.

(4) Vérifiez que la matrice

$$A = \begin{pmatrix} 2 & -4 & -1 & -3 \\ 1 & 0 & -5 & 7 \\ -6 & -8 & 0 & 7 \end{pmatrix}$$

se déclare sous MATLAB ainsi :

```
>> A = [ 2 - 4  - 1  -3 ; 1  0 - 5  7 ; -6 - 8  0  7] ;
>> A
```

Déclarez, si vous avez compris, une matrice B cette fois à 4 lignes et 3 colonnes (mettez des entrées réelles arbitraires). Que se passe -il lorsque vous tentez les commandes :

```
>> C = A*B;
>> C = B*A;
>> BB = B';
>> C = BB*A;
>> C = A*BB;
>> C = A.*BB;
>> C = BB.*A;
```

Dans chaque cas où vous n'avez pas reçu un message d'erreur, affichez BB ou C en faisant

```
>> BB
>> C
```

(sans point virgule cette fois) et tentez d'analyser ce qui se passe. Les bases de calcul matriciel acquises en S2 vous seront ici utiles.

Sauvez votre session (elle le sera automatiquement dans le répertoire TPMATLAB sous lequel vous travaillez sous forme d'un fichier .mat) ainsi (vérifiez bien après que ce fichier a été créé) :

```
>> save TPMATLAB-1
```

Exercice A.10 (préparation au TP suivant : Euclide étendu, boucles et récursivité).

(1) Révisez dans votre cours de MISMI la démarche conduisant à la recherche d'une solution de l'identité de Bézout : étant donnés deux entiers relatifs a et b avec $b \neq 0$, trouver deux nombres entiers u et v tels que

$$au + bv = \mathrm{PGCD}\,(|a|, |b|).$$

(2) Ouvrez sous MATLAB la routine bezout.m (en vous mettant dans le répertoire TPMATLAB).

(3) Réfléchissez sur le synopsis de la procédure (ici récursive, on notera qu'elle s'auto-appelle) écrite dans ce programme. Essayez de la relier à la démarche mathématique vue en MISMI pour trouver une solution (u, v) à l'identité de Bézout (et en même temps d'ailleurs le PGCD de $|a|$ et $|b|$).

(4) Faites des tests pour valider cette routine sur des exemples.

ANNEXE B

TP2 : assignation/réassignation de variables, boucles (Maple)

Cette seconde séance de TP a essentiellement pour objectif de vous familiariser avec l'assignation, la réassignation (et *a fortiori* la libération) de variables sous le logiciel de calcul symbolique Maple12. Ensuite, on s'entrainera à la construction de codes (sous Maple12) impliquant les commmandes

```
> for i from d to f by p while [...] do
instruction1 ;
instruction2 ;
...
instructionN ;
end do:

> for s in S while [...] do
instruction1 ;
instruction2 ;
...
instructionN ;
end do:

> if [condition1] then
instruction1 ;
instruction2 ;
...
instructionN ;
elif [condition2] then
instruction1bis ;
instruction2bis ;
...
instructionNbis ;
else
instruction1ter;
instruction2ter ;
...
instructionNter ;
end if:
```

Ces commandes correspondent pour les deux premières à la réalisation de boucles `do`, tandis que la dernière correspond à la mise en œuvre d'un processus de décision (alternative ici à trois volets, le second volet suivant l'instruction `elif` pouvant

ête éliminé). Dans les boucles `do` les éléments de syntaxe `by ...` et `while [...]` peuvent, le cas échéant, être omis si leur présence n'est pas justifiée par l'algorithme. On notera que l'instruction `:` au lieu de `;` lors de l'instruction `end` évite les affichages intermédiaires au fur et à mesure de l'exécution du code.

La réalisation de codes sous `Maple12` sera toujours faite ici en se positionnant derrière le prompt et en suivant sous les onglets du bandeau le chemin

$$Insert \rightarrow Code\ Edit\ Region$$

L'exécution du code, une fois celui ci rédigé dans la fenêtre adéquate, se fait avec le clic droit.

EXERCICE B.1 (réalisation d'un processus de décision). Ouvrez une feuille de travail vierge et titrez la `Boucles if`. En utilisant les instructions commandant une alternative :

```
if [...] then
instruction ;
else
instructionbis ;
end if ;
```

réalisez (puis validez) un code qui, étant données deux variables `a` et `b` auxquelles on assigne des valeurs rationnelles (par exemple `a:= 2/5; b:= 22/7;`), renvoie le message `a est plus grand que b` lorsque $a \geq b$ et le message `b est strictement plus grand que a` sinon. L'affichage d'un message (par exemple le mot `message`) sous `Maple12` se fait via l'instruction

```
> "message" ;
```

EXERCICE B.2 (réalisation d'un processus de décision). Continuez avec la feuille de travail précédemment ouverte. Comme dans l'exercice B.1, réalisez (puis validez) un code qui, étant donnés trois nombres `a`, `b`, `c` auxquels on assigne soit des valeurs rationnelles, soit des valeurs flottantes (faites un code dans les deux cas), pourvu que l'on assigne à `a` une valeur non nulle, renvoie :

- dans le cas où les valeurs assignées aux variables `a,b,c` sont rationnelles, l'expression algébrique des deux racines du trinôme aX^2+bX+c dans $\mathbb{C}$ (ou de l'unique racine double si $b^2-4ac=0$) ;
- dans le cas où les valeurs assignées aux variables `a,b,c` sont des réels flottants, des valeurs approchées pour les deux racines (éventuellement complexes) du trinôme aX^2+bX+c dans $\mathbb{C}$. Est-il raisonnable dans ce cas d'isoler le sous cas où il y a une racine double $(b^2-4ac=0)$?

Sauvez votre feuille de travail en l'enregistrant sous votre répertoire `TPMaple12` comme `TP2-boucleIF`.

EXERCICE B.3 (réalisation de boucles `do`). Ouvrez une feuille de travail vierge sous `Maple12`.

(1) Que se passe t-il lorsque l'on excécute le code

```
for i from 0 to 7 do
[i,i*i];
ifactor (i!);
‘-----‘;
```

```
end do:
```

Comparez avec ce qui se passe lorsque l'on exécute le code :

```
for i from 0 to 7 do
[i,i*i];
ifactor (i!);
‘-----‘;
end do;
```

(2) Réalisez (sous `Maple12`) un code permettant de calculer la somme

$$\sum_{k=1}^{N} k^2$$

lorsque N désigne un entier auquel on assigne ensuite une valeur. Testez ce code.

(3) Réalisez (toujours sous `Maple12`), en faisant intervenir dans la syntaxe les instructions `for ... to ... by ... do ... end do`, un code permettant de calculer la somme des entiers impairs compris entre 1 et un entier naturel $N \in \mathbb{N}^*$ auquel on assigne ensuite une valeur. Toujours sur le même principe, $N \in \mathbb{N}^*$ et $p \in \{1, ..., N\}$ désignant deux entiers naturels auxquels on assignera ensuite des valeurs, réalisez (et testez pour des valeurs de N et de p) un code permettant de calculer la somme

$$\sum_{\substack{1 \leq p \leq N \\ p \text{ divise k}}} k^3.$$

(4) Réalisez (toujours sous `Maple12`) un code permettant de calculer le plus petit entier N tel que la somme

$$\sum_{k=1}^{N} k^3$$

soit au moins égale à un seuil $M \in \mathbb{N}^*$ auquel on assigne une valeur donnée. Testez ce code avec $M = 10^{20}$ en demandant au terme de l'exécution l'affichage à la fois de cet entier $N = N(M)$ et de la valeur de la somme correspondante $\sum_1^{N(M)} k^3$.

Sauvez votre feuille de travail en l'enregistrant sous votre répertoire `TPMaple12` comme `TP2-boucleDO`.

EXERCICE B.4 (boucles `do ... end do` et `if ... else ... end if` enchainées et conjecture de Syracuse). Cet exercice est centré sur une célèbre conjecture, dite *Conjecture de Syracuse*, soulevée autour de 1930 par le mathématicien allemand Lothar Collatz. Voir par exemple, pour une histoire de cette conjecture et l'« état de l'art » aujourd'hui, le site

`http://fr.wikipedia.org/wiki/Conjecture_de_Syracuse`

Cette conjecture stipule que, pour tout entier $N \in \mathbb{N}^*$, la suite générée par récurrence suivant

$$u(1) = N$$
$$u(k+1) = \begin{cases} \frac{u(k)}{2} & \text{si } u(k) \text{ est pair (soit } \texttt{type(u(k),even) = true}\text{)} \\ 3u(k)+1 & \text{si } u(k) \text{ est impair (soit } \texttt{type(u(k),odd) = true}\text{)} \end{cases}$$
$$\forall\, k \geq 1$$

finit par atteindre en un temps fini la valeur 1.

(1) Expliquez pourquoi, une fois cette valeur 1 atteinte pour $k = k(N)$, la suite va nécessairement répéter indéfiniment le motif

```
1  4  2  1  4  2  1  4  2  1  4  2  1  4  2  1 ...
```

(2) Ouvrez une nouvelle feuille de travail, que vous intitulerez (en mode Texte) `Conjecture de Syracuse`. En utilisant la syntaxe

```
u:= N :
for i from 1 to M while u > 1 do
if ...  then
...
else
...
end if :
...
end do ;
```

réalisez un code affichant les valeurs prises par la suite $(u(k))_{k\geq 1}$ (initiée à la valeur N) tant que $k \leq M$ et que $u(k)$ reste différent de 1, ainsi que la valeur du premier cran $1 \leq k \leq M$ (s'il existe) tel que $u(k) = 1$. Testez ce code avec les valeurs de $N = 127$, $N = 537$, $N = 1235$ en prenant (ici arbitrairement) $M = 500$ comme « marge de sécurité ». La conjecture de Syracuse est-elle bien validée pour ces valeurs de N ? Que vaut dans chacun de ces trois cas le cran $k \geq 1$ tel que $u(k)$ prenne la valeur 1 pour la première fois ? Peut on affirmer ce cran est de plus en plus grand au fur est à mesure que N augmente ?

(3) Modifiez le code que vous venez d'élaborer pour que soient affichées uniquement au terme de son exécution :
   - le cran k (s'il existe entre 1 et M) où $u(k)$ prend la valeur 1 pour la première fois ;
   - la valeur maximale prise par l'entier $u(k)$ avant d'atteindre cette valeur 1.

Sauvez votre feuille de travail en l'enregistrant sous votre répertoire `TPMaple12` comme `TP2-Syracuse`.

Exercice B.5 (Algorithmes d'Euclide et d'Euclide étendu). Ouvrez une nouvelle feuille de travail sous `Maple12`.

(1) Apprenez à vous familiariser avec les commandes `iquo` et `irem` (fournissant quotient et reste dans la division euclidienne de deux entiers) et `quo` et `rem` (fournissant quotient et reste dans la division euclidienne de deux polynômes à coefficients entiers ou rationnels).

(2) Dans l'exercice 9 de la feuille 1 (question 3), vous avez observé la syntaxe du code `PGCD.m` (et du code `div.m` que ce code appelle) fournissant le calcul du PGCD de deux entiers sous `MATLAB`. On rappelle ces deux codes ici, tels qu'ils sont donnés dans le polycopié de cours :

```
function [q,r] = div (x,y) ;
q = floor (x./y) ;
r = x- q*y ;
```

```
function PGCD=PGCD(a,b);
x=a ;
y=b ;
while y>0
    [q,r] = div(x,y);
    if r==0
        PGCD = y;
        y = 0 ;
    else
        [q1,r1] = div(y,r);
        x = r;
        PGCD = x ;
        y=r1 ;
    end
end
```

En vous inspirant de ces modèles, élaborez et testez :

- un code sous `Maple12` (utilisant les instructions `iquo` et `irem`) permettant, au terme de son exécution, de calculer le PGCD de deux entiers naturels a et b, avec $b > 0$;
- un code sous `Maple12` (utilisant les instructions `quo` et `rem`) permettant, au terme de son exécution, de calculer le PGCD (dans $\mathbb{Q}[X]$) de deux polynômes A et B à coefficients rationnels, avec $B \neq 0$.

(3) Dans l'exercice 10 du TP1, vous avez observé la syntaxe du code `bezout.m` permettant de manière récursive (ce code s'auto-appelle) de calculer, étant donnés deux entiers relatifs a et b avec $b \neq 0$, une solution (u, v) de l'identité de Bézout :

$$\mathrm{PGCD}(|a|, |b|) = au + bv. \qquad (*)$$

On rappelle ici ce code, tel qu'il figure dans le polycopié de cours :

```
fonction [PGCD,u,v]=bezout(a,b);
x=a ;
y= abs(b) ;
[q,r]=div(x,y);
if r==0
    PGCD = y;
    u=0 ;
    v=1;
```

```
else
 [d,u1,v1]=bezout(y,r);
 PGCD=d;
 u=v1;
 v=sign(b)*(u1- q*v1);
end
```

En vous inspirant de ce modèle, élaborez et testez :

- un code sous `Maple12` (utilisant cette fois les instructions `iquo` et `irem`) permettant, au terme de son exécution, de calculer, étant donné un couple d'entiers relatifs (a, b) avec $b \neq 0$), le PGCD de $|a|$ et $|b|$ ainsi qu'un couple d'entiers (u, v) solution de l'identité de Bézout $(*)$;
- un code sous `Maple12` (utilisant les instructions `quo` et `rem`) permettant, au terme de son exécution, de calculer le PGCD (dans $\mathbb{Q}[X]$) de deux polynômes A et B à coefficients rationnels, avec $B \neq 0$, ainsi qu'un couple de polynômes U et V de $\mathbb{Q}[X]$ tels que
$$A(X)U(X) + B(X)V(X) = \mathrm{PGCD}(A, B)$$
(le PGCD étant ici considéré à un facteur multiplicatif $\lambda \in \mathbb{Q}^*$ près).

Sauvez votre travail dans votre dossier `TPMaple12` en l'enregistrant sous le nom `TP2-Euclide`.

EXERCICE B.6 (familiarisation avec les commandes `eval` et `subs`). Soit l'expression algébrique en trois variables :
$$P(X, Y, Z) = 3XY + 5XYZ^2 - 2XY^3Z.$$

(1) En utilisant la commande `eval` (on en examinera avec `?eval` la syntaxe), déclarez sous `Maple12` la fonction de la variable t :
$$t \longmapsto P(t^\alpha, t^\beta, t^\gamma),$$
où α, β, γ sont trois nombres rationnels strictement positifs auxquels des valeurs seront ensuite assignées.

(2) Affichez le graphe de la fonction :
$$t \in [0, 1] \mapsto \Big[P(x, y, z)\, \cos(xy)\Big]_{x=t^{1/5}, y=t^{1/8}, z=t^{1/4}}.$$

(3) En utilisant la commande `subs` (on en examinera avec `?subs` la syntaxe), déclarez sous `Maple12` la fonction des variables (u, v, w) :
$$(u, v, w) \longmapsto P(u + v + v, uv + vw + wu, uvw).$$

ANNEXE C

TP3 : familiarisation avec MATLAB, travail sur les tableaux

Le langage interprété MATLAB se fonde, on le verra, comme son nom l'indique (MATrixLABoratory), sur le travail à partir de tableaux (matérialisés en termes mathématiques par des matrices). L'objectif de cette troisième séance de TP est de poursuivre la familiarisation avec ce logiciel au travers précisément de ce travail sur les tableaux. On s'initiera aussi à la représentation visuelle avec les instructions `plot` (représentation graphique des fonctions d'une variable réelle), `mesh` ou `surf` (représentation graphique en 3D des fonctions de deux variables réelles), `image`, `imagesc`, `imshow` (visualisation des tableaux sous forme d'images, la brilliance des pixels traduisant la taille des entrées, ce de manière inversement proportionnelle : blanc = grande entrée numérique, noir=petite entrée numérique). Vous serez aussi amenés avec les commandes du graphisme telles `colormap`.

Une initiation à MATLAB (et Scilab) en `pdf` (malheureusement non interactive) est disponible sur le lien

`http://www.math.u-bordeaux1.fr/~ayger/initMS.pdf`

Téléchargez la depuis ce lien et sauvez la copie dans votre répertoire `TPMATLAB`. Vous imprimerez ultérieurement ce fichier pour le consulter à tête reposée.

Ouvrez pour l'instant MATLAB en tapant sous votre terminal la commande `matlab` : vous devez disposer de quatre espaces (que vous pouvez visualiser ou non en utilisant les onglets `Desktop` et `Window` sur le bandeau) :

- *Command Window* (l'espace pour vous le plus important) où vous allez taper le plus souvent vos instructions (derrière de prompt `>>`) et voir s'afficher les résultats au terme de leur exécution [1]. Il est utile d'exploiter le fait que la machine garde en mémoire les instructions déjà effectuées (ceci est bien utile lorsque vous voulez taper une instruction similaire à une instruction déjà effectuée : tapez les premières lettres et vous verrez apparaitre l'ancien modèle que vous pourrez alors corriger).
- *Command History* : c'est ici que s'affice l'historique des commandes. C'est une fenêtre que vous pouvez fermer pour ne pas encombrer votre écran de console.
- *Current Directory* : vous voyez dans cet espace le contenu du répertoire dans lequel vous vous êtes placé (qui devrait être toujours le répertoire `TPMATLAB` crée lors du TP1). C'est en particulier dans ce répertoire que se trouveront

1. Attention : mettre `;` derrière une instruction vous évite de voir l'affichage (c'est exactement le contraire de ce qui se passe avec `Maple12!`) ; ne rien mettre derrière vous fait au contraire courir le risque de voir s'afficher un résultat, ce qui n'est en règle générale pas souhaitable, surtout lorsque les variables en jeu sont de grands tableaux.

tous les fichiers `.m` correspondant à des `script` (par exemple commençant par `[.,., ...,.] = function (.,...,.)`) que vous aurez à écrire ou à utiliser, les fichiers `.mat` correspondant aux sessions ou aux variables que vous sauverez éventuellement, ainsi que les fichiers de données numériques (tableaux de nombres, images `jpeg`, *etc.*) préalablement téléchargés.
- *Workspace* : ici se trouvent recensées les variables actuellement affectées dans votre présente session de travail.

Les commandes `Unix` restent valables sous l'environnement `MATLAB` ; reportez vous au fascicule de TP 2011-2012 (F. Pazuki), page 12, pour un rappel de ces commandes avec les touches `Ctrl + [...]`.

Quelques tests préliminaires. Au contraire de `Maple12` (qui est un logiciel interprété de calcul symbolique ou formel), donc entrainé aux calculs *sans pertes* lorsque les variables d'entrée sont des entiers, des rationnels, ou des polynômes à coefficients entiers, rationnels, voire algébriques, `MATLAB` est un logiciel interprété de calcul scientifique, où les calculs numériques impliquent en général des nombres réels ou complexes déclarés en flottants (`floating`, soit -c'est le cas par défaut- en double précision `double`, ou bien en simple précision `single`) et sont donc tributaires des arrondis inévitables liés à l'erreur machine (en accord toutefois avec la norme du standard `IEEE754` (*cf.* les sections 1.2.3 et 1.2.4 du polycopié de cours). Testez les commandes suivantes :

```
>> eps
>> realmax
>> realmin
```

A l'aide de `help` (exemple : tapez `help eps`), analysez le sens de ces variables modifiables ; comment interpréter le résultat de la commande

```
>> eps(x)
```

lorsque `x` est un nombre flottant (faites le test avec `eps(pi)`, avec `eps(1)`, `eps(1/2)`, `eps(`$10^{(20)}$ `+ pi)`, `eps(`10^7`+pi)`). Conclusion ? En quoi `eps(x)` peut il être considéré comme un indicateur de la précision au niveau du flottant `x` ? Le logiciel n'affiche par défaut que quatre chiffres représentatifs pour les nombres réels flottants (en double précision) ; pour en voir 16 (comme cela est théoriquement possible, aux arrondis près, *cf.* la valeur de `eps(x)`), il faut préalablement taper l'instruction :

```
>> format long
```

Testez ceci sur des exemples (par exemple sur les variables non effacables `i`, `j`, `sqrt(-1)`). Attention : `j` est le `i` des physiciens, ce n'est pas ici le `exp(2*i*pi/3)` des mathématiciens !

EXERCICE C.1 (Manipulation de tableaux numériques sous `MATLAB`). Les commandes `rand (M,N)` et `randn (M,N)` génèrent des matrices à `M` lignes et `N` colonnes dont les entrées sont des réalisations de variables aléatoires réelles (mutuellement indépendantes) suivant respectivement :
- en ce qui concerne la fonction `rand`, la loi uniforme sur $[0, 1]$ (chaque entrée est un nombre flottant aléatoire entre 0 et 1, tous les choix étant équiprobables) ;
- en ce qui concerne la fonction `randn`, la loi normale (moyenne 0 et écart type égal à 1) sur $\mathbb{R}$.

(1) Générez de telles matrices suivant les instructions :

```
>> A = rand(10,8);
>> A
>> Adouble = [A A];
>> Adouble
>> B = randn(12,9);
>> B
>> Bdouble = [B;B];
>> Bdouble
>> AA = A(1:2:10,1:2:8);
>> AA
>> BB = B(1:3:8,2:2:9);
>> ONES = ones (7,5);
>> ONES
>> ZEROS = zeros (10,7);
>> ZEROS
>> EYE = eye (10,5);
>> EYE
>> AAA = flipud (A);
>> AAA
>> BBB = fliplr (B);
>> BBB
```

Analysez l'opération qu'exécute chacune de ces instructions. Qu'exécute en particulier l'instruction `Mat1=Mat(1:pas1:M,1:pas2:N)` si `Mat` est un tableau numérique à `M` lignes et `N` colonnes, `pas1` et `pas2` étant des entiers respectivement entre 1 et `M` et 1 et `N` ?

(2) Construisez une matrice `Apad` à 16 lignes et 16 colonnes en bordant la matrice `A` générée à la question (1) de manière symétrique (des deux côtés) par des blocs de zéros [2]. Faites ensuite la même chose avec cette fois des blocs de 1.

(3) Créez une matrice à 19 lignes et 15 colonnes dont les lignes et les colonnes sont celles de la matrice `A` générée à la question (1), mais séparées cette fois entre elles par des lignes et des colonnes de zéros.

(4) Quelle est la méthode la plus judicieuse (au niveau du temps d'exécution) pour déclarer sous `MATLAB` les matrices :

$$U = \begin{pmatrix} 0 & 0 & 3 & 0 & 1 & 0 \\ 5 & 0 & 0 & 0 & 0 & 8 \\ 0 & 2 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 8 & 0 \\ 1 & 0 & 0 & 0 & 0 & 5 \end{pmatrix} \qquad V = \begin{pmatrix} 1 & 1 & 3 & 1 & 1 & 1 \\ 5 & 1 & 1 & 1 & 1 & 8 \\ 1 & 2 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 8 & 1 \\ 1 & 1 & 1 & 1 & 1 & 5 \end{pmatrix} \quad ?$$

Déclarez ces deux matrices de la manière que vous jugerez la plus efficace possible (au niveau du temps d'exécution).

Sauvez votre session dans votre répertoire `TPMATLAB` (celui dans lequel vous êtes précisément en train de travailler) avec l'instruction `save TP3exo1`. Vérifiez que

2. Pareille opération, fort utile dans la pratique, soit pour compléter une matrice rectangulaire en une matrice carrée pour des calculs ultérieurs, soit pour s'accorder une ≪ marge de sécurité ≫, telle un ≪ cadre ≫, autour d'un tableau ou d'une image, s'appelle le *zeropadding*.

vous avez bien ainsi créé un fichier `TP3exo1.mat` dans votre répertoire `TPMATLAB`. Faites ensuite l'instruction `clear all` pour rendre à nouveau vierge tout votre *Workspace*. Vérifiez que c'est bien le cas. Que se passe t'il si vous donnez les instructions :

```
>> load TP3exo1
>> who
```

Faites `clear all` à nouveau.

EXERCICE C.2 (décomposition de Haar d'une image). Voici encore un exercice prétexte à la manipulation de tableaux sous `MATLAB`. Vous y verrez aussi l'étroite relation entre tableaux de nombres (donc matrices) et images. Téléchargez depuis le site `http://www.math.u-bordeaux1.fr/~yger/initiationMATLAB` le fichier (de données) `imageTP3exo2`. Une fois ceci fait, « chargez » ce fichier dans votre *Workspace* en tapant sous la *Command Window* les instructions :

```
>> load imageTP3exo2
>> I=imageTP3exo2;
```

Grâce à la commande `size`, vérifiez que la variable ainsi déclarée `I` est un tableau à 256 lignes et 256 colonnes [3]. Testez les valeurs de quelques entrées `I(i,j)` pour voir qu'il s'agit en fait d'une matrice dont les entrées sont des nombres entiers positifs (mais considérés ici comme des nombres flottants en double précision, comme la réponse `1` à l'instruction `isfloat(I)` vous le prouvera, vérifiez le).

(1) Visualisez la matrice `I` de deux manières :
   - en utilisant la visualisation en 3D suivant `mesh(I)` ;
   - en utilisant la visualisation en 2D suivant `image(I)`, `imagesc(I)` (voire aussi `imshow(I,[])` si vous êtes au CREMI, *cf.* plus loin).

   Quelle est, pour ce type de matrice `I`, de ces deux visualisations, celle qui est la plus adaptée ? Quel est l'effet de la commande

   ```
   >> II = imrotate(I,angle);
   ```

   (où `angle` désigne la valeur en flottant d'un angle entre 0 et 360 degrés exprimé en flottant) ? [4]. On utilisera l'une des routines du type `image` mentionnées plus haut pour visualiser l'image `II` ainsi qu'une instruction convenable (faire `help axis`) pour que l'affichage respecte la taille (ici carrée) de l'image. Quelle est la difficulté liée à la réalisation mathématique d'une telle transformation de matrice (pensez au passage entre le repérage cartésien dans une grille de pixels et le repérage polaire) ? Que se passe t-il en particulier lorsqu'un pixel (i, j) tourne d'un certain angle ? Tombe t'on encore sur un pixel de la grille cartésienne ?

(2) Ouvrez un fichier `.m` vierge en utilisant l'onglet *File* du bandeau. Vous allez essayer dans ce fichier de rédiger un code, dont la première instruction (déclaration des entrées et sorties de la fonction) sera

   ```
   function [RR,DH,DV,DO] = Haar (ipt)
   ```

3. Notez que 256 est une puissance de 2, ce qui n'est, on le verra plus tard, nullement un hasard en ce qui concerne les formats d'images `jpeg`.

4. Cette instruction `imrotate` bien utile n'est malheureusement pas présente dans la bibliothèque de `Scilab`.

qui est censé calculer, étant donnée une matrice de flottants `ipt` de taille $(2^N, 2^N)$, les quatre matrices, de taille $(2^{N-1}, 2^{N-1})$, dont les entrées sont respectivement les nombres

$$\begin{aligned}
\texttt{RR}(i,j) &= \frac{\texttt{ipt}(2i-1,2j-1)+\texttt{ipt}(2i-1,2j)+\texttt{ipt}(2i,2j-1)+\texttt{ipt}(2i,2j)}{2}\\
\texttt{DH}(i,j) &= \frac{\texttt{ipt}(2i-1,2j-1)+\texttt{ipt}(2i-1,2j)-\texttt{ipt}(2i,2j-1)-\texttt{ipt}(2i,2j)}{2}\\
\texttt{DV}(i,j) &= \frac{\texttt{ipt}(2i-1,2j-1)-\texttt{ipt}(2i-1,2j)+\texttt{ipt}(2i,2j-1)-\texttt{ipt}(2i,2j)}{2}\\
\texttt{DO}(i,j) &= \frac{\texttt{ipt}(2i-1,2j-1)-\texttt{ipt}(2i-1,2j)-\texttt{ipt}(2i,2j-1)+\texttt{ipt}(2i,2j)}{2}
\end{aligned}$$

Testez votre code en l'exécutant avec comme matrice `ipt` la matrice `I` :

```
>> [RR, DH, DV, DO] = Haar(I);
```

Affichez avec `image` (ou mieux `imagesc`) les quatre matrices `RR, DH, DV, DO`. Pour le rendu des couleurs et du graphisme, vous pouvez jouer avec la commande `colormap` (exemples les instructions `colormap hot`,`colormap jet`, `colormap pink`, `colormap jet`, *etc.*, faire `help colormap` pour décider). Pour un meilleur rendu graphique, vous pouvez aussi faire appel à l'instruction

```
>> imshow(MATRICE,[ ])
```

Cette commande est dans la bibliothèque du *Toolbox* `Image Processing` (disponible sous `MATLAB` au CREMI). Justifiez la terminologie : `RR` = *Résumé*, `DH` = *Détails Horizontaux*, `DV` = *Détails Verticaux*, `DO` = *Détails Obliques*. Recommencez le traitement de la matrice `I` cette fois sur la matrice `RR`. Qu'observez vous sur les quatre images de taille $(64, 64)$ ainsi obtenues ? Poursuivez si vous voulez pour obtenir quatre images de taille (32,32) en décomposant le nouveau résumé. La décomposition que vous êtes en train d'effectuer ici s'appelle *décomposition de Haar* : elle permet d'« isoler » les « structures » cohérentes des détails (ou « accidents ») d'une image digitalisée. Sauvez votre fichier `.m` sous le nom `Haar` si vous ne l'avez pas encore fait et fermez le.

(3) Ouvrez un nouveau fichier vierge `.m` depuis l'onglet *File* du bandeau. Rédigez dans ce fichier un code (correspondant encore à une fonction) dont la première instruction sera

```
function II = HaarInverseAux (I1,I2,I3,I4)
```

qui, étant donnée quatre matrices `I1,I2,I3,I4` de taille $(2^{N-1}, 2^{N-1})$, fabrique une matrice `II` de taille $(2^N, 2^N)$ telle que

$$\begin{aligned}
\texttt{II}(2i-1,2j-1)=\texttt{I1}(i,j),\ \texttt{II}(2i-1,2j)=\texttt{I2}(i,j)\\
\texttt{II}(2i,2j-1)=\texttt{I3}(i,j),\ \texttt{II}(2i,2j)=\texttt{I4}(i,j)
\end{aligned}$$

pour toute paire d'entiers (i, j) entre 1 et 2^{N-1}. Sauvez ce nouveau fichier `.m` sous le nom `HaarInverseAux`. Vous pourrez être amenés à utiliser ce code comme code auxiliaire par la suite.

(4) Ouvrez un nouveau fichier `.m` où vous réaliserez un code (toujours une fonction) dont la première instruction sera cette fois

`function I=HaarInverse (RR,DH,DV,DO)`

qui, étant données quatre matrices `RR, DH, DV, DO`, toutes de même taille $(2^{N-1}, 2^{N-1})$, reconstitue une matrice I de taille $(2^N, 2^N)$ de manière à ce que ce code exécute l'opération inverse de celle exécutée par la fonction `Haar`. Validez votre résultat en utilisant la matrice `I` de la question 1. Sauvez votre fichier `.m` comme `HaarInverse`.

EXERCICE C.3 (Réalisation d'une fonction `in line`). Une fonction `g` d'une variable (réelle ou complexe) peut être déclarée sous MATLAB par la commande

```
g = inline('x^2-1','x');
```

(1) Déclarez sur ce modèle la fonction

$$x \in \mathbb{R} \mapsto \frac{\sin(\pi x)}{\pi x},$$

dite aussi *sinus cardinal*. Représentez ensuite le graphe de cette fonction sur $[-10, 10]$, le pas d'échantillonnage étant choisi pour les abscisses égal à $1/100$. Utilisez pour cela la commande `plot`. Quel est l'effet de l'instruction `plot(x,y,'r')` par rapport à l'instruction `plot(x,y,'k')`, ou à la simple instruction `plot(x,y)` ? Déclarez la nouvelle fonction

$$x \in \mathbb{R} \mapsto \frac{\sin(\pi x)}{\pi(\sqrt{x^2+1})}.$$

En utilisant `hold` (faire `help hold`), affichez le graphe de cette seconde fonction (toujours au dessus de $[-10, 10]$ avec le même pas) sur la même figure que précédemment, mais avec une autre couleur que le graphe précédent.

(2) On considère maintenant les deux fonctions, cette fois de deux variables,

$$F : (r, \theta) \longmapsto r^5 \cos(5\theta) + r^3(\cos(3\theta))/2 + 1$$
$$G : (r, \theta) \longmapsto r^5 \sin(5\theta) + r^3(\sin(3\theta))/2$$

(on notera qu'il s'agit de la partie réelle et de la partie imaginaire de la fonction polynomiale $z \mapsto z^5 + z^3/2 + 1$, exprimées toutes deux en coordonnées polaires). Ouvrez un fichier `.m` (que vous dénommerez, en le sauvant, `TP3exo3`) et dans lequel, sur les deux premières lignes, vous déclarerez ces deux fonctions `F` et `G`, fonctions des variables `r` et `theta`, sur le modèle suivant lequel vous avez déclaré $x \mapsto \sin(x)/x$ à la première question. Sur les lignes qui suivent, rédigez les instructions qui conduisent :

- à l'évaluation des fonctions F, G et $\sqrt{F^2+G^2}$ sur le maillage `r=0:2/100:2`, `theta=0:pi/100:pi` de $[0, 2] \times [0, \pi]$ (de pas $2/100$ en abscisse et $\pi/100$ en ordonnée). Attention ! pensez à déclarer l'une des deux variables `r` ou `theta` (à vous de décider laquelle) en ligne et l'autre en colonne, car il faut que `F(r,theta)` soit une matrice carrée de taille ici $(101, 101)$, et non un scalaire !
- à l'affichage en trois dimensions (suivant la commande `mesh`), dans cet ordre, sur trois figures successives, des graphes des trois fonctions $F, G, \sqrt{F^2+G^2}$ au dessus de la grille carrée définie plus haut ;
- avec l'instruction finale `axis([0 pi 0 2 0 1])` (faire `help axis` pour en comprendre la syntaxe), à un zoom sur la dernière figure obtenue, à savoir celle du graphe de $\sqrt{F^2+G^2}$, permettant ainsi

de localiser (au moins grossièrement) les trois zéros du polynôme $P(X) = X^5 + X^3/2 + 1$ situés dans le demi plan $\operatorname{Im} z \geq 0$ du plan complexe.

En utilisant l'onglet *Desktop* dans le bandeau du fichier `TP3exo3` que vous venez de remplir, lancez l'exécution des instructions rédigées dans ce code. Localisez les valeurs approchées des modules et des arguments des trois racines de $P(X)$ de partie imaginaire positive ou nulle. Que peut-on dire des deux racines manquantes (dans tout le plan complexe cette fois) ?

Nettoyez votre *Workspace* avec `clear all` (après avoir vérifié que votre fichier `TP3exo3` a bien été sauvé comme un fichier `.m` dans votre répertoire `TPMATLAB`).

EXERCICE C.4 (Maillages et évaluation de fonctions sur un maillage). Faites `help meshgrid` et analysez l'exemple qui est proposé sous ce `help` pour l'évaluation de la fonction

$$(x, y) \mapsto x \exp(-x^2 - y^2)$$

au dessus d'un maillage de $[-2, 2]^2$ avec un pas de .2 suivant les deux axes de coordonnées.

(1) Sur le même modèle que dans l'exemple ainsi défini, construire 3 variables `X,Y,Z` de manière à ce que la commande

```
>> surf (X,Y,Z)
```

corresponde à l'affichage en 3D du graphe de la fonction

$$(x, y) \mapsto \operatorname{sinc}(x^2 + y^2)$$

(où sinc représente le sinus cardinal, voir la question 1 de l'exercice 3) au dessus du maillage régulier de $[-2, 2]^2$ de pas .1 (suivant les deux axes). Quelle est la taille des trois variables `X,Y,Z` ? Réalisez cette même représentation graphique avec cette fois l'instruction `mesh` au lieu de `surf`.

(2) Ouvrez un nouveau fichier `.m` que vous sauverez dans votre répertoire de travail `TPMATLAB` comme `MaillagePolaire`. Rédigez sur ce fichier le code d'une fonction

```
function Z=MaillagePolaire(r1,r2,K)
```

qui retourne, étant donnés deux nombres flottants positifs `0<r1<r2` et un entier `K>0`, la matrice `Z` de taille `(K+1,K+1)` dont l'entrée (i, j) correspond au nombre complexe de module et argument respectivement :

$$r(i, j) = \texttt{r1} + \frac{i-1}{\texttt{K}}\,\texttt{(r2-r1)}, \quad \theta(i, j) = -\pi + 2\pi\,\frac{j-1}{\texttt{K}}$$

(vous utiliserez ici encore l'instruction `meshgrid` sur le modèle de ce que vous avez fait à la question 1).

Après vous être assuré que votre fichier `MaillagePolaire.m` avait bien été sauvé, faites l'instruction `clear all` pour libérer votre *Workspace*.

EXERCICE C.5 (méthode de Newton dans le champ complexe).

(1) Ouvrez le fichier `Newton.m` et sauvez le (toujours comme fichier `.m`) sous le nom `Newton1`. Modifiez ce fichier de manière à ce que la boucle `DO` s'arrête automatiquement dès que `abs(x(k+1)-x(k)) <= eps` ou que le nombre d'itérations dépasse strictement un seuil `N` fixé, et que la fonction

`function [x,err,Nb] = Newton1(init,N)`

ainsi construite valide (lorsque `Nb <N` et `err=0`) la valeur approchée `x` de l'unique zéro réel de $X^5+X^3/2+1$ obtenue, l'erreur finalement commise [5] `err=abs(x(Nb+1)-x(Nb))` au terme de l'exécution du code, ainsi que le nombre d'itérations `Nb` (nécessairement inférieur ou égal à `N`) effectuées réellement lors de l'exécution de la boucle pour parvenir (si toutefois l'algorithme y parvient dans les limites du nombre maximal `N` d'itérations imparti) au premier cran tel que `abs(x(Nb+1)-x(Nb))<eps`. Testez votre nouveau code avec `init=-1` et le seuil garde fou `N=20`. Quelle valeur de `Nb` trouvez vous ?

(2) Testez votre nouvelle fonction `Newton1` en prenant cette fois `init=1+i` (`i=sqrt(-1)`) avec `N=100`, puis `init=-1+i` avec `N=100`. Qu'observez vous ? Déduisez en les valeurs approchées des cinq racines (complexes) du polynôme $X^5 + X^3/2 + 1$.

(3) Modifiez si nécessaire votre fonction `Newton1` pour qu'elle puisse admettre comme variable d'entrée une variable `init` qui soit cette fois une matrice de nombres complexes et non plus un nombre complexe. Vérifiez (mathématiquement) que les cinq racines du polynôme $X^5 + X^3/2 + 1$ sont dans la couronne du plan complexe $\{0.8 \leq |z| \leq 2\}$ et que le polynôme dérivé $5X^4 + 3X^2/2$ ne s'annule pas dans cette couronne. Lancez

```
>> [Z,err,Nb] = Newton1(MaillagePolaire(.8,2,100),100);
```

(la fonction `MaillagePolaire` a été construite à l'exercice 4). Retrouvez à partir de l'affichage (avec `imagesc(X)`, `imshow(X,[])` ou `mesh`(X), appliqué à une matrice `X` qui sera `abs(Z)` ou `angle(Z)`) la position (en polaire) des cinq racines complexes du polynôme $X^5 + X^3/2 + 1$ (*cf.* la question 2 de l'exercice 3).

EXERCICE C.6 (À propos de trigonalisation des matrices (en relation avec le cours d'algèbre)). Toute matrice à coefficients complexes se trigonalise sur $\mathbb{C}$ (c'est le théorème de Cayley-Hamilton). On suppose dans cet exercice que l'on dispose d'une procédure `MATLAB` du type

```
>> [v,V] = ValeurVecteurPropre (M);
```

qui, étant donnée une matrice carrée `M`, fournisse (de manière approchée) une valeur propre de `M` en même temps qu'un vecteur propre associé. Écrivez une procédure récursive (impliquant la routine `ValeurVecteurPropre` comme subroutine) qui, implémentée sous la forme

```
>> [P,T] = trigonalisation (M);
```

fournisse :

(1) en colonne dans la matrice `P` les vecteurs d'une base $\mathcal{B}$ dans laquelle l'endomorphisme de matrice `M` (dans la base canonique) se représente par une matrice triangulaire supérieure dans la base $\mathcal{B}$;

(2) la matrice triangulaire supérieure `T` représentant l'endomorphisme (de matrice `M` dans la base canonique) dans la base $\mathcal{B}$.

5. Ce devrait en principe être 0 pour la machine lorsque `Nb<N` ; si ce n'est pas le cas, ceci signifie que l'algorithme n'a pu aboutir au terme du nombre garde-fou `N` d'itérations imparti *a priori*.

ANNEXE D

TP4 : procédures sous Maple, représentation graphique

Cette séance de TP poursuit la familiarisation avec Maple12. Le chapitre 2 du cours (méthodes approchées pour le calcul des zéros d'équations non linéaires) fournira le support théorique à ce TP. Ouvrez Maple12 pour commencer. L'initiation à la rédaction de *procédures*, en particulier de *procédures récursives*, constituera la trame de cette séance. Le tracé de courbes fractales (flocon de Von Koch, courbe de Lévy) mettra en œuvre les techniques acquises ici.

Avant de commencer, voici quelles sont les principales structures de données sous Maple12. Contrairement à l'environnement MATLAB, nous disposons ici d'un champ de structures de données plus vaste que celui des structures à base de tableaux, ceci s'avérant nécessaire en vue de l'objectif assigné au logiciel, à savoir le calcul symbolique.

- Les *suites* (finies bien sûr) se déclarent ainsi sous Maple12 :
  `> S := 5,6,7,n,8,n,9,n+1;`
  On accède au terme d'indice k dans la suite (k=1,...,N si celle ci est de longueur N) avec l'instruction
  `> S[k];`
  (faites par exemple le test avec la suite déclarée ci-dessus, puis refaites le une fois spécifiée une valeur de la variable non déclarée n). La suite vide se déclare comme `S:= Null;`.
- Les *listes* (ordonnées) sont assimilables aux tableaux à une ligne (et N colonnes) tels qu'ils sont déclarés sous MATLAB (excepté le fait que l'on doive ici séparer les entrées successives par des virgules) :
  `> L := [5,6,7,n,8,n,9,n+1];`
  `> L[k];`
  (faites encore le test). La liste vide se déclare comme `L:=[ ];`.
- Les *ensembles* correspondent à des listes, mais cette fois non ordonnées. Un ensemble (fini) E se déclare *via* :
  `> E := {5,6,7,n,8,n,9,n+1};`
  L'ensemble vide emptyset se déclare comme `emptyset :={ };`.

Familiarisez vous avec l'instruction op qui renvoie les opérandes d'une expression e (lorsque l'on précise leur position), ou le nombre d'opérantes (nops), voire l'expression e simplifiée (op(e)), par exemple :

```
> op(0,5,6,7,n,8,n,9,n+1);
> op(0,[5,6,7,n,8,n,9,n+1]);
> op(0,{5,6,7,n,8,n,9,n+1});
> op(4,5,6,7,n,8,n,9,n+1);
> op(7,[5,6,7,n,8,n,9,n+1]);
```

```
> op(8,{5,6,7,n,8,n,9,n+1});
> op(9,{5,6,7,[n,8,n,9,n+1]});
> op(3..4,{5,6,7,[n,8,n,9,n+1]});
> op(4..6,{5,6,7,n,8,n,9,n+1});
> op(4..6,{5,6,7,[n,8,n,9,n+1]});
> op([5,6,7,n,{8,n,9,n+1}]);
> nops([5,6,7,n,{8,n,9,n+1}]);
```

Pour appliquer une fonction (disons `g`, préalablement déclarée) à une suite, une liste, ou un ensemble (et donc calculer la suite, la liste ou l'ensemble constituées des images par `g` des éléments de la suite, de la liste ou bien de l'ensemble initial), on utilise l'instruction `map`. Exemple :

```
> g:= x-> x^3 - sin(x);
> map(g,S);
> map(g,L);
> map(g,E);
```

Recommencez après avoir assigné une valeur numérique à `n`. Entrainez vous.

Dans ce TP, vous allez apprendre à rédiger des programmes autonomes. Supposons que l'on souhaite rédiger une `procedure` sous `Maple12`, c'est-à-dire créer une fonction `Maple` qui, étant donnée un certain nombres d'`arguments` (ou encore « données » en `input`, par exemple `argument1,argument2,...,argumentk`), renvoie, une fois la procédure lancée, une sortie (`NomProcedure`) correspondant au dernier résultat fourni par l'exécution du code sous les données initiales que sont `argument1`,...,`argumentk`. Le schéma de la rédaction d'une telle procédure dans une zone de code est donc le suivant :

```
NomProcedure := proc(argument1,...,argumentk)
instruction1;
instruction2;
...
instructionk;
end proc;
```

Exercice D.1 (Génération de suites récurrentes à un ou plusieurs pas). Le but de cet exercice est de réaliser des procédures récursives (un seul `argument` entier noté `n`, plus éventuellement des arguments supplémentaires figurant des valeurs de paramètres dont dépend la suite à générer) permettant de générer une suite régie par une récurrence à un ou plusieurs pas. Le terme *récursive* signifie que la procédure s'auto-appelle lors de son exécution. Pour éviter que l'exécution du code ne boucle indéfiniment, il faut bien sûr être soigneux de déclarer les « cas initiaux » !

(1) Vérifiez que la procédure suivante permet de générer une suite arithmétique de raison `4` et de premier terme `3` :

```
fonction:=proc(n)
if n=0 then
3;
else
fonction(n-1) + 4;
end if;
end proc;
```

Pourquoi est-il essentiel de faire figurer la boucle `if n=0 then else ... end if;` dans le synopsis ? Déduisez de cela la construction d'une procédure récursive

`fonction1TP4 := proc(raison,init,n)`

qui, étant données les trois arguments `raison`, `init`, `n`, génère la suite arithmétique de premier terme `init` et de raison `raison`.

(2) Sur le modèle établi à la question **1**, rédigez une procédure récursive

`FibonacciRec:=proc(n)`

permettant de calculer le `n`-ième nombre de Fibonacci. On rappelle que la suite des nombres de Fibonacci est la suite $(F_n)_{n\geq 0}$ telle que $F_0 = F_1 = 1$ et

$$F_n = F_{n-2} + F_{n-1} \qquad \forall\ n \geq 2.$$

Adaptez ce code pour l'insérer dans une procédure récursive

`fonction2TP4 :=proc(a,b,init0,init1,n)`

qui, étant donnés les cinq arguments `a,b,init1,init2,n`, génère la suite $(u_n)_{n\geq 0}$ initiée à $u_0 =$ `init0` et $u_1 =$ `init1` et régie ensuite par la récurrence linéaire à deux pas :

$$u_n = a\,u_{n-1} + b\,u_{n-2} \qquad \forall\, n \geq 2.$$

(3) En utilisant la commande `time` (combinée avec la commande `seq` permettant de générer des suites ou des listes), générez la liste `L` des temps de calcul CPU pris par l'exécution de `FibonacciRec (i)`, `i=20..35`. En utilisant `listplot` (sous `with (plots);`), affichez (en fonction de l'indice `i`) la liste obtenue en transformant `L` par l'application `log`. Que constatez vous ?

(4) Pour chaque entier n supérieur ou égal à 2, soit c_n le nombre d'appels au programme `FibonacciRec` impliqués dans `FibonacciRec(n)`. Montrez que la suite $(c_n)_{n\geq 2}$ vérifie la relation $c_n = c_{n-1} + c_{n-2} + 1$, puis déduisez en que $c_n = 2F_n - 1$. Lorsque n tend vers $+\infty$, prouvez en utilisant cette fois la relation matricielle

$$\begin{pmatrix} F_n \\ F_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} F_{n-1} \\ F_{n-2} \end{pmatrix}$$

et votre cours d'Algèbre 3 que $F_n \sim ((1+\sqrt{5})/2)^n$. Comparez la pente visible sur le graphe obtenu précédemment avec le nombre d'or $(1+\sqrt{5})/2$. Pouviez vous prévoir le résultat observé ainsi ?

(5) Modifiez la syntaxe de la procédure `FibonacciRec` en utilisant l'instruction additionnelle `option remember`, sous la forme

`FibonacciRec := proc(n) option remember;`

Testez là maintenant avec des valeurs de `n` supérieures à `35`. Que constatez vous ? Modifiez la procédure obtenue en une procédure `FibonacciRecbis` de manière à ce que se trouvent affichées en sortie toutes les valeurs intermédiaires $F_0, ..., F_n$ (au lieu de simplement la valeur finale F_n). Testez là pour $n = 30$ et comparez avec les temps de calculs impliqés dans l'utilisation de `FibonacciRec` aux questions **3** et **4**.

Sauvez votre feuille de travail comme `TP4exo1` dans votre répertoire `TPMaple12` puis fermer cette feuille de travail.

EXERCICE D.2 (approximation de zéros par des méthodes itératives). Ouvrez une nouvelle feuille de travail.

(1) Sur le modèle de ce qui a été fait à l'exercice 1, construisez une procédure récursive [1]

`fonction3exactTP4 := proc(a,b,c,init0,init1,n)`

qui, étant donnés 6 arguments `a,b,c,init0,init1,n` (cinq rationnels, un entier naturel)

- soit calcule exactement, lorsque cela est possible (c'est-à-dire si l'exécution du code ne fait jamais apparaitre de division par 0), le terme d'ordre `n` dans la suite initiée avec $u_0 =$ `init0`, $u_1 =$ `init1`, et régie par la relation inductive :

$$u_n = a + \frac{b}{u_{n-1}} + \frac{c}{u_{n-1}u_{n-2}} \; ; \qquad (\dagger)$$

- soit retourne le message d'erreur
`Error, division by zero`
dès que surgit dans l'exécution du code une division par 0. Testez cette procédure en déclarant les arguments `a:=108, b:=-815, c:=1500, init0:=11/3, init1:=43/11, n:=30`.

Modifiez la en une procédure `fonction3exactTP4bis` qui affiche en sortie toutes les valeurs u_k intermédiaires.

(2) Modifiez la procédure `fonction3exactTP4` pour construire une procédure `fonction3approcheeTP4` qui cette fois, toujours avec les mêmes arguments, calcule (toujours de manière récursive) de manière approchée les u_n de proche en proche. Modifiez la en une procédure `fonction3exactTP4bis` qui affiche en sortie toutes les valeurs u_k intermédiaires. Testez ces procédures `fonction3approcheeTP4` et `fonction3approcheeTP4bis` avec les mêmes arguments déclarés qu'à la question **1** et comparez avec ce que donnent les procédures `fonction3exactTP4` et `fonction3exactTP4bis` réalisées à la question **2**. Que constatez vous de manière évidente ?

(3) En utilisant l'instruction `solve`, trouvez avec `Maple12` les trois racines (exactes) du polynôme $X^3 - 108X^2 + 815X - 1500$.

(4) On fixe maintenant, comme dans les questions **1** et **2**, `a:=108, b:=-815, c:=1500, init0:=11/3, init1:=43/11`. On pose $a_0 = 1$ et, pour tout $n \in \mathbb{N}$, $a_{n+1} = u_0 \cdots u_n$. En utilisant la relation inductive (†), vérifiez (mathématiquement) que la suite $(a_n)_{n\geq 0}$ obéit à la relation récurrente (à trois pas) :

$$a_{n+3} - 108\, a_{n+2} + 815\, a_{n+1} - 1500\, a_n = 0 \quad \forall\, n \in \mathbb{N}. \qquad (\dagger\dagger)$$

Vérifiez que les suites $(3^n)_{n\geq 0}$, $(5^n)_{n\geq 0}$ et $(100^n)_{n\geq 0}$ vérifient aussi la relation récurrente (††). Chargez le package `LinearAlgebra` et consultez

1. Pensez à y intégrer l'instruction `option remember` comme à la question **5** de l'exercice 1.

l'aide de `LinearSolve` pour trouver trois nombres rationnels α, β, γ de manière à ce que :

$$\begin{aligned} a_0 &= \alpha + \beta + \gamma \\ a_1 &= 3\,\alpha + 5\,\beta + 100\,\gamma \\ a_2 &= 9\,\alpha + 25\,\beta + 10000\,\gamma. \end{aligned}$$

Déduisez en

$$\forall\, n \in \mathbb{N},\ a_n = \frac{2}{3}\,3^n + \frac{1}{3}\,5^n.$$

Définissez (sous `Maple12`) une fonction `f` qui à n associe a_n. Comment s'exprime la fonction $n \mapsto u_n$ en fonction de `f` ? Déclarez cette nouvelle fonction $n \mapsto u_n$ sous `Maple12`. En utilisant l'instruction `simplify`, vérifiez avec `Maple12` que la suite $(u_n)_{n\geq 0}$ est bien une suite croissante, majorée par 5. Ceci vous permet-il de justifier ce que vous aviez observé à la question **1** concernant le comportement asymptotique de la suite $(u_n)_{n\geq 0}$?

Sauvez votre travail comme le fichier `TP4exo2` dans votre répertoire `TPMaple12`, puis fermez cette feuille de travail.

EXERCICE D.3 (extraction d'une racine carrée par l'algorithme « de Babylone »). L'algorithme dit « de Babylone » constitue depuis l'Antiquité l'illustration la plus célèbre de la méthode de Newton. Cet exercice est centré autour de cette méthode classique. Ouvrez une nouvelle feuille de travail.

(1) Montrez que la suite itérative conduisant à l'approximation de $\sqrt{2}$ par la méthode de Newton est régie par la relation inductive :

$$x_n = \frac{x_{n-1}}{2} + \frac{1}{x_{n-1}} \quad \forall\, n \in \mathbb{N}^*.$$

Vérifiez que si cette suite $(x_n)_{n\geq 0}$ est initiée à $x_0 = 7/5$, c'est une suite de rationnels et que l'on a

$$|x_{n+1} - \sqrt{2}| \leq |\sqrt{2} - x_n|^2 \quad \forall\, n \in \mathbb{N}. \qquad (*)$$

Quel terme de cette suite $(x_n)_{n\geq 0}$ convient-il alors de calculer pour déterminer $\sqrt{2}$ avec 1000 décimales exactes ?

(2) Rédigez deux procédures pour générer la suite de Babylone $(x_n)_{n\geq 0}$ (générée avec $x_0 =$ `init` et régie par la relation inductive $(*)$:
   - l'une récursive `procNewtonRec:=proc(init,n)`
   - l'autre directe `procNewton:=proc(init,n)`

   retournant toutes deux le terme d'ordre `n` de la suite $(x_n)_{n\geq 0}$ lorsque celle ci est initiée au nombre `init` (que l'on déclarera comme un nombre flottant). Testez ces deux procédures avec `init`=1.4 et `n`=10 en prenant `Digits:=1001`. Comparez le résultat avec `evalf(sqrt(2))`.

(3) Transformez la procédure `procNewton` en une autre, `procNewton2`, qui commence la calcul avec une précision aussi faible que possible (`Digits:=2`) et double la précision à chaque itération. Quelle valeur de `n` faut-il cette fois prendre pour retrouver $\sqrt{2}$ avec 1000 décimales exactes ? Comparez les temps d'exécution de la procédure `procNewton` (avec `init`=1.4 et `n`=10 sous `Digits:=1001`) et de la nouvelle procédure `procNewton2`, avec toujours `init`=1.4, mais cette fois `n`=12.

Sauvez votre travail cette fois comme le fichier `TP4exo3` (toujours dans le répertoire `TPMaple12`) et fermez ensuite cette feuille de travail.

EXERCICE D.4 (une approximation de π par des suites adjacentes). Ouvrez une nouvelle feuille de travail.

(1) Réalisez une procédure `approxPi1TP4:=proc(n)` (basée sur une boucle `DO`) générant simultanément les deux suites $(u_n)_{n\geq 0}$ et $(v_n)_{n\geq 0}$ de nombres strictement positifs initiées respectivement par $u_0 = 1.$, $v_0 = 2.$ (attention de déclarer ces entrées comme des nombres flottants !) et obéissant au couple de relations inductives :

$$\begin{aligned} u_{n+1} &= \frac{u_n + v_n}{2} \\ v_{n+1} &= \sqrt{u_{n+1} v_n}. \end{aligned}$$

Lancez cette procédure sous par exemple `Digits:=20` avec `n=50`. Que constatez vous ? On admet que les deux suites $(u_n)_{n\geq 0}$ et $(v_n)_{n\geq 0}$ sont adjacentes et ont pour limite commune le nombre $\sqrt{27}/\pi$. Modifiez votre procédure en une procédure `approxPi2TP4:=proc(n)` qui fournisse l'approximation de π déduite de l'égalité approchée $(u_n + v_n)/2 \simeq \sqrt{27}/\pi$.

(2) Modifiez la procédure `approxPi2TP4` en une procédure

`approxPi3TP4:=proc(epsilon,n)`

qui, étant donné un seuil `epsilon` fixé, retourne l'approximation de π déduite de l'égalité approchée $(u_n + v_n)/2 \simeq \sqrt{27}/\pi$ dès que la condition

$$\frac{|u_n - v_n|}{u_n + v_n} \leq \epsilon$$

est satisfaite (et affiche le nombre `i`$\leq$`n` d'itérations de la boucle `DO` nécessaires).

(3) Réalisez une procédure inductive `approxPiRecTP4:=proc(n)` générant simultanément les deux suites $(u_n)_{n\geq 0}$ et $(v_n)_{n\geq 0}$ de la question **1**, ce de manière exacte cette fois (on prend $u_0 = 1$ et $v_0 = 2$, déclarés cette fois comme rationnels et non plus comme flottants). Comparez, pour `n`=12 (attention surtout à ne pas prendre de valeur plus grande, car le temps de calcul risque d'exploser !) le temps d'exécution de cette nouvelle procédure `approxPiRecTP4(12)` avec celui de la procédure `approxPi1TP4(12)` (dans laquelle u_0 et u_1 sont déclarés comme les entiers 1 et 2 et non plus comme les flottants 1.0 et 2.0). Que constatez vous ?

Sauvez votre feuille de travail comme `TP4exo4` (dans `TPMaple12`) et fermez cette feuille.

EXERCICE D.5 (un modèle de courbe fractale : le flocon de Von Koch). Les courbes planes qui ont la propriété d'*autosimilarité*, c'est-à-dire qui, regardées à la loupe avec n'importe quel grossissement, reproduisent à toutes les échelles le même motif, se prêtent (en ce qui concerne leur tracé) à la réalisation de procédures récursives. Le « flocon de Von Koch », introduit vers 1900 par le mathématicien suédois Helge von Koch (1870-1924), en est une illustration ; regardez par curiosité avant de commencer le site dédié :

`http://fr.wikipedia.org/wiki/Flocon_de_Koch`

(1) Soient a et b deux nombres complexes, définissant un segment (noté $[a,b]$) du plan complexe. On introduit les trois nombres complexes :

$$\begin{aligned} u(a,b) &:= a + \frac{b-a}{3} = \frac{2a+b}{3} \\ v(a,b) &:= b - \frac{b-a}{3} = \frac{2b+a}{3} \\ w(a,b) &:= u(a,b) + e^{i\pi/3}(v(a,b) - u(a,b)) \end{aligned}$$

et l'on note Φ l'application qui au segment $[a,b]$ (d'origine a et d'extrémité b) associe la ligne brisée $\Phi([a,b])$ joignant (dans cet ordre) les quatre points $a, u(a,b), w(a,b), v(a,b), b$. Si $\mathcal{L}$ est une ligne brisée de sommets d'affixes $a_1, a_2, ..., a_N$, la ligne brisée $\Phi(\mathcal{L})$ désigne par extension la ligne brisée obtenue en mettant bout à bout les diverses lignes brisées $\Phi([a_j, a_{j+1}])$ pour $j = 1, ..., N-1$. Exprimez en fonction de N (nombre de sommets de la ligne brisée $\mathcal{L}$) le nombre de sommets de la ligne brisée $\Phi(\mathcal{L})$. Réalisez une procédure

`VonKoch1TP4:=proc(L)`

qui, étant donnée une liste `L` de nombres complexes, considérée comme la liste `[a1,...,aN]` des affixes des sommets successifs d'une ligne brisée $\mathcal{L}$, renvoie (dans l'ordre, et de manière approchée) la liste des affixes des sommets de la ligne brisée $\Phi(\mathcal{L})$.

(2) En utilisant la procédure `VonKoch1TP4`, réalisez une nouvelle procédure

`VonKoch2TP4:=proc(n)`

qui renvoie (dans l'ordre) la liste des affixes des sommets de la ligne brisée $\Phi^{[n]}([0,1])$, où $\Phi^{[n]}$ désigne l'application Φ itérée `n` fois. Vu que le nombre de sommets de la ligne brisée $\Phi^{[n]}([0,1])$ croit vers l'infini comme $2 \times 4^{\mathtt{n}}$ (dites pourquoi), vous ne testerez cette procédure `VonKoch2TP4` que sur des valeurs de `n` entre 0 et 5. En utilisant les instructions

```
> with (plots);
> L:=VonKoch2TP4(n):
> complexplot(L,style = line, scaling = constrained);
```

affichez la ligne brisée obtenue en itérant `n` fois Φ à partir du segment $[0,1]$. Veillez à ne prendre comme valeurs de `n` que des valeurs entre 1 et 7 (notez que $2 \times 4^7 = 32768$ est déjà très grand !). Que se passe-t-il si vous remplacez la dernière instruction par :

```
> complexplot(L,style=line);
```

La courbe ainsi obtenue est une approximation du flocon de Von Koch ; ce flocon de Von Koch est un exemple de courbe fractale, continue mais ne présentant de tangente en aucun point [2] ; c'est en fait la limite uniforme des courbes ainsi tracées – en fonction de `n` – lorsque `n` tend vers l'infini).

2. Vous verrez plus tard dans votre cursus mathématique que pareille courbe fractale n'est pas ≪ rectifiable ≫, et que l'on ne peut donc pas parler de ≪ longueur ≫ du flocon de Von Koch. La nature fourmille de tels modèles fractaux (en relation avec ce que l'on appelle le ≪ chaos dynamique ≫) : pensez par exemple à la découpe d'une côte volcanique telle celle du Groendland, à celle de la ligne d'arête des aiguilles de Chamonix ... Vous trouverez aussi beaucoup de modèles (mathématiques cette fois) d'images fractales sur le `web` (courbes de Mandelbrojt, frontières de domaines de Julia, *etc.*).

(3) Réalisez une procédure cette fois récursive

```
VonKoch3TP4 :=proc(a,b,n)
```

retournant la liste (dans l'ordre) des affixes des sommets de la ligne brisée $\Phi^{[n]}([a,b])$. En prenant $a = 0$ et $b = 1$, retrouvez (en visualisant les résultats avec `complexplot` plutôt qu'en les faisant afficher numériquement !) le résultat `L` de l'instruction

```
>L:= VonKoch2TP4(n):
```

Comparez les temps CPU nécessaires par les deux instructions

```
> L:= VonKoch3TP4 (0,1,n);
> L:= VonKoch2TP4 (n);
```

pour les valeurs de `n` entre 3 et 8 (si toutefois vous parvenez jusqu'à `n` = 8 avec la seconde instruction [3]).

Sauvez votre feuille de travail comme `TP4exo5` dans votre répertoire `TPMaple12`, puis fermez cette feuille.

EXERCICE D.6 (un autre modèle de courbe fractale : la courbe de Lévy). Si elle porte le nom du probabiliste français Paul Lévy, la *courbe de Lévy* (qui est un autre exemple célèbre de courbe fractale) a été introduite par les mathématiciens Ernesto Cesàro (italien) en 1906 et Georg Faber (allemand) en 1910. Regardez par curiosité le site dédié sur `wikipedia`. Ouvrez une nouvelle feuille de travail sous `Maple12`.

(1) Soient a et b deux nombres complexes, définissant un segment du plan complexe. On introduit le nombre complexe :

$$r(a,b) = \frac{a+b}{2} + i\frac{b-a}{2}.$$

et l'on note Ψ l'application qui à ce segment (d'origine a et d'extrémité b) associe la ligne brisée $\Psi([a,b])$ joignant (dans cet ordre) les trois points $a, r(a,b), b$. Si $\mathcal{L}$ est une ligne brisée de sommets d'affixes $a_1, a_2, ..., a_N$, la ligne brisée $\Psi(\mathcal{L})$ désigne par extension la ligne brisée obtenue en mettant bout à bout les diverses lignes brisées $\Psi([a_j, a_{j+1}])$ pour $j = 1, ..., N-1$. Exprimez en fonction de N (nombre de sommets de la ligne brisée $\mathcal{L}$) le nombre de sommets de la ligne brisée $\Psi(\mathcal{L})$. Comme dans la question **1** de l'exercice 5, réalisez une procédure

```
Levy1TP4 := proc(L)
```

qui, étant donnée une liste `L` de nombres complexes, considérée comme la liste `[a1,...,aN]` des affixes des sommets successifs d'une ligne brisée $\mathcal{L}$, renvoie (dans l'ordre, et de manière approchée) la liste des affixes des sommets de la ligne brisée $\Psi(\mathcal{L})$.

(2) Réalisez une procédure récursive

```
Levy2TP4 := proc(epsilon,L)
```

3. Notéz que $2 \times 4^8 = 131072$, ce qui est vraiment très grand pour la taille d'une liste de flottants sous `Maple12` !

qui, étant donnée une liste d'affixes `L` (figurant la liste des sommets ordonnés d'une ligne brisée $\mathcal{L}$) renvoie la liste (ordonnée) des affixes des sommets de la ligne brisée $\Psi^{[n]}(\mathcal{L})$ dès l'instant où le pas de cette ligne brisée devient strictement inférieur au seuil `epsilon`.

(3) En utilisant `complexplot` (après avoir déclaré l'environnement `plots` par l'instruction `with(plots);`), représentez la liste `L` obtenue *via*

```
> L:=Levy2TP4(.03,[I,1,I]):
```

Sauvez votre feuille de travail comme `TP4exo6` dans votre répertoire `TPMaple12`, puis fermez cette feuille.

ANNEXE E

TP5 : les fonctions sous MATLAB et l'interpolation

Cette séance de TP5 poursuit la familiarisation avec MATLAB. Le chapitre 3 du cours, en particulier ce qui concerne l'interpolation de Lagrange (section 3.2 du cours) ou l'interpolation par les polynômes trigonométriques (et ses conséquences : algorithmes de transformation de Fourier rapide, multiplication rapide des polynômes[1], *etc.*, *cf.* la section 3.1.4 du cours), fournira la trame théorique de ce TP5. Ouvrez MATLAB pour commencer.

EXERCICE E.1 (Différences divisées et polynôme de Lagrange).

(1) Depuis le site web

`http://www.math.u-bordeaux1/~yger/initiationMATLAB`

téléchargez le fichier `DiffDiv.m` correspondant à la routine fournissant (dans cet ordre), la liste des `N+1` différences divisées :

`y[x(1),...,x(N+1)], y[x(1),...,x(N)],..., y[x(1)]`

attachée à une liste de nombres complexes `Y=[y(1),...,y(N+1)]` « divisée » par une liste de nombres complexes distincts

`X=[x(1),...,x(N+1)]`.

Après avoir enregistré ce fichier `DiffDiv.m` dans votre répertoire `TPMATLAB`, ouvrez le et vérifiez que la syntaxe de la fonction proposée correspond bien aux formules inductives sous tendant le tableau proposé page 51 du polycopié de cours. Montrez que le nombre de multiplications impliquées dans ce calcul est en $O(N^2)$.

(2) Ouvrez un nouveau fichier `.m`, vierge cette fois. En vous inspirant du schéma de Hörner (*cf.* la syntaxe page 40 du polycopié) et de la formule de Newton

$$\texttt{Lagrange[X;Y](z)} = \\ = \texttt{y[x(1)]} + \sum_{k=2}^{N+1} \texttt{y[x(1),...,x(k)]} * (z - \texttt{x(1)}) * \cdots * (z - \texttt{x(k-1)})$$

(formule (3.8) du polycopié, page 50), rédigez dans ce fichier une fonction :

`function LXX = LagrangeNewton (X,Y,XX)`

qui, étant donnée une liste de nombres complexes distincts `X` (de longueur `N+1`) et une liste de nombres complexes `Y` de même longueur, évalue le polynôme de Lagrange `Z` $\mapsto$ `Lagrange[X;Y](Z)` aux points de la liste

1. Ceci sera repris dans un TP ultérieur sous `Maple12`, cette fois dans le cadre du calcul arithmétique sans pertes, sous l'angle du calcul symbolique et de la cryptograhie.

de nombres complexes `XX` (de longueur arbitraire) et renvoie en sortie la liste `LXX` (de même longueur que `XX`) correspondant aux valeurs prises par ce polynôme de Lagrange aux entrées de `XX` (en respectant l'ordre de ces entrées). La fonction `DiffDiv` sera utilisée comme fonction auxiliaire. Sauvez ce fichier `.m` dans votre répertoire de travail (*Workspace*, ici `TPMATLAB`) comme `LagrangeNewton.m`. Testez ce fichier en déclarant la fonction

```
f = inline('exp(-x.^2/2).*sin(pi*x)','x');
```

puis en déclarant :

```
>> X= -3:.1:3 ;
>> XX=-3:.05:3 ;
>> LXX=LagrangeNewton(X,f(X),XX);
>> plot(XX,f(XX),'r');
>> hold
>> plot(XX,LXX,'k');
```

Qu'observez vous ? Recommencez avec cette fois les instructions

```
>> X= -6:.2:6 ;
>> XX=-6:.05:6 ;
>> LXX=LagrangeNewton(X,f(X),XX);
>> figure
>> plot(XX,f(XX),'r');
>> hold
>> plot(XX,LXX,'k');
```

Qu'observez vous cette fois ? Hors de quel segment de $[-6, 6]$ l'approximation de `f` par son polynôme de Lagrange se met-elle à dérailler ? Recommencez en prenant cette fois plus de points d'interpolation :

```
>> X=-6:.15:6;
>> XX=-6:.05:6 ;
>> LXX=LagrangeNewton(X,f(X),XX);
>> figure
>> plot(XX,f(XX),'r');
>> hold
>> plot(XX,LXX,'k');
```

Les choses s'arrangent-elles ou empirent-elles par rapport à la figure 2 ? Continuez avec encore plus de points :

```
>> X=-6:.1:6 ;
>> XX=-6:.05:6 ;
>> LXX=LagrangeNewton(X,f(X),XX);
>> figure
>> plot(XX,f(XX),'r');
>> hold
>> plot(XX,LXX,'k');
```

Vous observez ici le *phénomène de Runge*. Voir pour plus de détails le site sur `wikipedia` :

`http://fr.wikipedia.org/wiki/Ph%C3%A9nom%C3%A8ne_de_Runge`

(3) Ouvrez un nouveau fichier `.m` vierge et rédigez dans ce fichier un code

```
function LXXT= LagrangeTchebychev(a,b,N,f,XX)
```

qui, étant donné un segment fermé borné [a,b] de $\mathbb{R}$ (donné par deux bornes distinctes `a` et `b` déclarées en flottants), un entier strictement positif `N`, une fonction `f` de [a,b] dans $\mathbb{R}$ ou $\mathbb{C}$ déclarée `inline('f(x)','x')` au préalable (voir l'exercice 3 du TP 3), et une liste `XX` de points du segment [a,b], renvoie la liste des valeurs aux points de `XX` prises par le polynôme d'interpolation de Lagrange interpolant les `N+1` valeurs de la fonction `f` aux `N+1` points du segment [a,b] donnés par

$$\texttt{X(k)} = \frac{\texttt{a}+\texttt{b}}{2} + \Big(\frac{\texttt{b}-\texttt{a}}{2}\Big)\cos\Big(\frac{\texttt{(2*k-1)*pi}}{\texttt{2*(N+1)}}\Big), \quad \texttt{k=1,...,N+1}.$$

Ces points sont déduits par homothétie et translation des `N+1` zéros dans [-1,1] du polynôme de Tchebychev T_{N+1} donné par

$$T_{N+1}(\cos(\theta) = \cos((N+1)\theta) \qquad \forall\, \theta \in \mathbb{R}.$$

Vous serez amenés à utiliser la fonction `LagrangeNewton` construite à la question 2 comme fonction auxiliaire appelée lors de l'exécution du code correspondant à la fonction `LagrangeTchebychev`. Sauvez le fichier dans votre répertoire de travail (*Workspace*, ici `TPMATLAB`) comme le fichier `LagrangeTchebychev.m`. Testez le en prenant la fonction `f` de la question 2 :

```
>> f = inline('exp(-x.^2/2).*sin(pi*x)','x');
>> XX=-6:.05:6 ;
>> LXXT=LagrangeTchebychev(-6,6,30,f,XX);
>> figure
>> plot(XX,f(XX),'r');
>> hold
>> plot(XX,LXXT,'k');
```

Qu'observez vous maintenant en comparaison avec le phénomène de Runge observé à la question 2 ? Prenez des valeurs de `N` plus grandes que `N=30`, par exemple `N=35,40,45,50` et recommencez. La situation empire-t'elle encore cette fois ? Que se passe-t'il malgré tout si l'on persiste à augmenter `N` (prenez `N=60` pour voir). Pourquoi cette liste de points déduite des zéros du polynôme de Tchebychev T_{N+1} se comporte-t-elle mieux du point de vu de la qualité de l'interpolation au bord (si on la compare à l'interpolation de Lagrange avec des points équidistribués sur [a,b] (comme à la question 2) ? Expliquez pourquoi les choses se gâtent malgré tout lorsque `N` est pris trop grand (exemple `N=60`).

Nota. Vous pouvez observer la répartition des points d'interpolatioon sur le segment $[a, b]$ (ramené à $[-1, 1]$) en affichant le graphe de la fonction $t \mapsto \cos((N+1)\arccos(t))$ sur $[-1, 1]$; les points où l'on a effectué l'interpolation sont les $N+1$ zéros de cette fonction polynomiale (on observe qu'ils s'accumulent lorsque N augmente vers les extrémités du segment $[-1, 1]$) :

```
>> XX = -1 : 1/M : 1;
>> f=inline('cos((N+1)*acos(x))','x');
```

```
>> plot(XX,f(XX));
```

EXERCICE E.2 (la méthode récursive de Neville-Aitken).

(1) Téléchargez depuis le site

`http://www.math.u-bordeaux1/~yger/initiationMATLAB`

le fichier `lagrange.m` (il a été réactualisé, aussi devez vous le re-télécharger même si vous l'avez déjà). Ouvrez ce fichier et vérifiez que la syntaxe du code récursif écrit ici est bien celle conduisant à la construction du polynôme d'interpolation de Lagrange suivant la démarche récursive de Neville-Aitken, telle qu'elle est présentée pages 52 et 53 du polycopié.

(2) Ouvrez un fichier `.m` vierge et, en vous inspirant du code `lagrange`, rédigez un code

`function LXXN=NevilleAitken(a,b,N,f)`

qui, étant donné un segment fermé borné [`a`,`b`] de $\mathbb{R}$ (donné par deux bornes distinctes `a` et `b` déclarées en flottants), un entier strictement positif `N`, une fonction `f` de [`a`,`b`] dans $\mathbb{R}$ ou $\mathbb{C}$ déclarée `inline('f(x)','x')` au préalable (voir l'exercice 3 du TP 3), renvoie (sous la forme de la liste de ses coefficients) le polynôme d'interpolation de Lagrange interpolant les `N+1` valeurs de la fonction `f` aux `N+1` points du segment [`a`,`b`] équirépartis entre `X(1)=a` et `X(N+1)=b`. Testez ensuite ce code pour représenter (avec des valeurs de `N` égales à `N=5,7,10,12`) les graphes des interpolées sur $[-1,1]$ avec pas régulier des fonctions

```
f1= inline('x.^7+3*x.^5-2*x+1','x');
f2= inline ('x.*cos(x) -log(x+2)+1','x');
f3= inline('1./(1+x.^2)','x');
f4= inline('(1+x.^2-x.^3).*exp(x/5)','x');
```

Pensez pour cela à évaluer au préalable votre polynôme `LXXN` (dont `LXXN` figure juste la liste des coefficients) sur le vecteur `XX` des `M+1` points de $[-1,1]$ régulièrement espacés de `1/M` (prenez par exemple 10^2 ou 10^3 comme valeur de `MM`) :

```
>> XX=-1:1/M:1;
>> LXXN=NevilleAitken(-1,1,N,f);
>> LXXNevalue=polyval(LXXN,XX);
```

Vous pouvez aussi utiliser, à la place de `polyval`, la routine `Horner` disponible ici :

`http://www.math.u-bordeaux1/~yger/initiationMATLAB`

Justifiez la qualité des résultats obtenus en vous référant aux formules de Taylor explicites (Taylor-Lagrange ou Taylor avec reste intégral, *cf.* votre cours d'Analyse 1 de S2 [2]).

2. En ligne : `http://www.math.u-bordeaux1.fr/~yger/analyse1.pdf`, sections 2.5.3 et 2.5.5.

EXERCICE E.3 (multiplication rapide des polynômes et `fft`).

(1) Sous `MATLAB`, la routine effectuant la multiplication à gauche d'un vecteur colonne de nombres complexes, de longueur $\mathtt{N} = 2^p$ ($p \in \mathbb{N}^*$), par la matrice

$$\left[W_{\mathtt{N}}^{kj}\right]_{0\leq j,k\leq \mathtt{N}-1} \qquad (\text{où } W_{\mathtt{N}} = \exp(-2i\pi/\mathtt{N}))$$

suivant l'algorithme de Cooley-Tukey (impliquant seulement $(p-1)2^{p-1}$ « vraies » multiplications au lieu des 2^{2p} attendues, *cf.* le cours pages 44-45) est la commande

```
>> Y=fft(X,N);
```

En utilisant la relation

$$\left(\left[W_{\mathtt{N}}^{kj}\right]_{0\leq j,k\leq \mathtt{N}-1}\right)^{-1} = \frac{1}{\mathtt{N}}\left[\overline{W_{\mathtt{N}}{}^{kj}}\right]_{0\leq j,k\leq \mathtt{N}-1},$$

écrire sur un fichier `.m` vierge une routine `InverseFFT` :

```
function X=InverseFFT(Y,N)
```

qui, étant donné un vecteur colonne `Y` de nombres complexes, de longueur `N`, calcule la multiplication de `Y` à gauche par la matrice

$$\left(\left[W_N^{kj}\right]_{0\leq j,k\leq N-1}\right)^{-1}$$

(utilisez la routine `conj` qui conjugue les entrées des vecteurs ou des matrices). Validez ce code sur un exemple, par exemple :

```
>> X=rand(16,1) + i*rand(16,1);
>> Y=fft(X,16);
>> XX=InverseFFT(Y,16);
>> max(abs(X-XX))
```

Que constatez vous ? Pourquoi ne trouvez vous pas `0` comme vous vous y attendriez ? Ne perdez pas de vue que vous êtes ici en train de faire du calcul scientifique, et non du calcul symbolique. On reviendra dans un TP ultérieur sous `Maple12` cette fois sur la `fft` discrète, dans le cadre (arithmétique) du calcul symbolique cette fois. Sauvez votre fichier (appelé `InverseFFT`) comme un fichier `.m` dans votre répertoire `TPMATLAB` pour le conserver dans vos archives, tout en sachant que la routine

```
>> X=ifft(X,N);
```

(déjà implémentée dans le noyau du logiciel) correspond de fait à la même fonction.

(2) Ouvrez un fichier `.m` vierge. Rédigez dans ce fichier une procédure

```
function  P1P2 = ProduitPolynomes (P1,P2)
```

qui, étant donnés deux polynômes `P1` et `P2` déclarés sous forme de listes comme

```
>> P1 = [P1(1) ... P1(N1)];
>> P2 = [P2(1) ... P2(N2)];
```

lorsque

$$\begin{aligned} \texttt{P1}(X) &= \texttt{P1(1)}\, X^{\texttt{N1}-1} + \cdots + \texttt{P1(N1-1)}\, X + \texttt{P1(N1)} \\ \texttt{P2}(X) &= \texttt{P2(1)}\, X^{\texttt{N2}-1} + \cdots + \texttt{P2(N1-1)}\, X + \texttt{P2(N2)}, \end{aligned}$$

retourne (sous forme d'une liste de longueur `N1+N2-1`) les coefficients du polynôme `P1*P2`, les monômes de ce polynôme étant rangés dans l'ordre décroissant. La première étape de la procédure est de déterminer le premier entier `p` tel que $2^p >$ `N1+N2-1` (l'entier le plus proche d'un nombre réel flottant `x` lorsque l'on va à droite de ce nombre est `ceil(x)` ; c'est `floor(x)` si l'on va à gauche). Utilisez ensuite les routines `fft(.,N)` et `ifft(.,N)` comme indiqué dans le cours, avec précisément `N` $= 2^p$, ce après avoir si nécessaire complété les listes `P1` et `P2` par des zéros. Sauvez le fichier `ProduitPolynomes.m` dans votre répertoire `TPMATLAB` après l'avoir testé (et validé) sur des polynômes de bas degré (inférieur ou égal à `10`).

(3) La commande `MATLAB` permettant de déterminer l'écriture binaire d'un entier `M` (bien sûr inférieur à 2^{52}) est

```
>> P = dec2bin(M);
```

mais, attention, la sortie `P` est ici au format `char`, pas au format numérique double précision `double`. Ce format `double` est pourtant nécessaire ici car les calculs de `fft` impliquent la matrice de nombres fottants

$$[W_{\texttt{N}}^{kj}]_{0\leq j,k\leq \texttt{N}-1}.$$

Il faut donc convertir `P` au format `double` pour travailler numériquement ; en faisant un petit test, vous verrez que le caractère `0` correspond dans cette conversion à un certain entier positif `P=double('0')`, tandis que `1` correspond (heureusement) à `P+1=double('1')`, ce qu'il convient donc de prendre en compte pour les conversions. À vous de trouver ces deux nombres `double('0')` et `double('1')`, car vous voulez, vous, `0` et `1` comme nombres. Sur un nouveau fichier vierge `.m`, rédigez une procédure

```
function M1M2=ProduitEntiers(M1,M2)
```

qui, étant donnés deux entiers naturels tous deux inférieurs à 2^{26} (cela vaut mieux, dites pourquoi !), utilise les routines `dec2bin`, `fft(.,64)`, `ifft(.,64)` pour calculer leur produit `M1*M2`. Pensez à associer à tout entier naturel `M` (inférieur à 2^{52}) le polynôme

```
>> PolM = double(dec2bin(M))-double('0') ;
```

(pour avoir des coefficients numériquement égaux à `0` et `1` et non plus cette fois à `double('0')` et `double('1')` comme le donne la conversion numérique du format `char` au format `double`), puis ensuite à évaluer le polynôme `PolM1*PolM2` (calculé comme à la question 2) en `x=2`.

Exercice E.4 (L'algorithme de Cooley-Tukey). La routine sous `MATLAB`

```
>> Y=fft(X,N);
```

(lorsque `N` est une puissance de 2) correspond à l'implémentation de l'algorithme de Cooley-Tukey (*cf.* le polycopié de cours pages 44-45). Vous pouvez aussi consulter pour plus de détails les sites `wikipedia`

`http://en.wikipedia.org/wiki/Cooley_FFT_algorithm`
`http://fr.wikipedia.org/wiki/Transform%C3%A9e_de_Fourier_rapide`
(le site en anglais est plus riche). Le but de cet exercice (prétexte, comme dans le TP4, à votre familiarisation avec le principe de récursivité) est de réaliser une routine réalisant la même opération, ce afin de comprendre le schéma récursif sur lequel se fonde la procédure. Ouvrez un fichier `.m` vierge sur lequel vous allez rédiger une procédure récursive :

`function Y=CooleyTukey(X,p)`

transformant, si `p` désigne un entier supérieur ou égal à `1` et `X` un vecteur colonne de longueur $2^{\mathtt{p}}$, le vecteur `X` en son image par la multiplication à gauche par la matrice

$$\left[W_{2^{\mathtt{p}}}^{jk}\right]_{0\leq j,k\leq 2^{\mathtt{p}-1}}, \qquad \text{où } W_{2^{\mathtt{p}}} := \exp(-2i\pi/2^{\mathtt{p}}). \tag{†}$$

(1) Puisqu'il s'agit de construire un algorithme récursif, vous commencerez par mettre sur pied une boucle

```
if p == 1
Y= ... ;
else
... ;
... ;
Y= ...;
end
```

en rédigeant d'abord les instructions (très simples) correspondant à l'alternative `p==1`. Ce cas correspond à l'initialisation de la procédure récursive.

(2) Il vous faut maintenant remplir les instructions sous l'alternative `else`. Regardez pour cela l'architecture du programme telle qu'elle est présentée sur le diagramme page 45 du polycopié.
- Commencez par sérier en deux vecteurs colonne `Xpair` et `Ximpair` (tous deux de longueur $2^{\mathtt{p}-1}$) le vecteur colonne d'entrée `X`.
- Faites agir sur ces deux vecteurs colonne la procédure au cran $\mathtt{p}-1$ comme suggéré dans le diagramme.
- Opérez les multiplications terme-à-terme nécessaires concernant le traitement de `Ximpair` après passage à travers la procédure au cran $\mathtt{p}-1$ (regardez bien pour cela le diagramme).
- Complétez enfin la liste des intructions sous l'alternative `else` par une boucle `do` rendant compte des calculs impliquant l'≪ action papillon ≫ de la matrice correspondant à $\mathtt{p}=1$ (suivez soigneusement la syntaxe du synopsis présenté sur le diagramme).

(3) Validez enfin votre code en comparant les résultats `Y` et `Yref` de

```
X=rand(2^p,1);
Y=CooleyTukey(X,p);
Yref=fft(X,2^p);
```

pour de petites valeurs de `p` (`p=3,4,5`). Sauvez votre fichier `CooleyTukey.m` dans votre répertoire `TPMATLAB`.

(4) Justifiez sur le papier par un raisonnement mathématique pourquoi ce code récursif correspond bien à la multiplication à gauche de `X` par la matrice (†).

EXERCICE E.5 (interpolation par des polynômes trigonométriques). Soit `f` une fonction d'une variable réelle (définie sur $[0,1]$, à valeurs réelles) que vous déclarerez ultérieurement (avec une expression explicite pour `f(x)`) *via*

```
>> f= inline('f(x)','x');
```

sous MATLAB (en veillant à ce que la syntaxe permette de calculer `f(x)` lorsque `x` est un vecteur ligne ou colonne de nombres de $[0,1]$, et non seulement un nombre de $[0,1]$). Il est souvent plus judicieux (en particulier si `f` est une fonction oscillante comme on en rencontre dans le monde des télécommunications) de chercher à interpoler `f` sur $[0,1]$ non par des fonctions polynomiales (commme dans l'interpolation de Lagrange), mais par des fonctions polynomiales trigonométriques, du type

$$\theta \in [0,1] \mapsto \sum_{l=-M}^{M-1} u_k\, e^{2i\pi l\theta}, \quad \text{où } M \in \mathbb{N}^*, \tag{$*$}$$

qui sont, elles aussi, des fonctions oscillantes, donc de même nature que la fonction à interpoler `f`. Vous verrez plus tard que c'est le principe de ce que l'on appelle faire l'*analyse de Fourier* de l'information fournie par `f`. Soit $\mathtt{N} = 2^{\mathtt{p}}$.

(1) Vérifiez, si `p` est un entier naturel non nul donné, que le système de $2^{\mathtt{p}}$ équations à $2^{\mathtt{p}}$ inconnues u_k, $k = -2^{\mathtt{p}-1}, ..., 2^{\mathtt{p}-1}-1$,

$$\Big[\sum_{k=-2^{\mathtt{p}-1}}^{2^{\mathtt{p}-1}-1} u_k\, e^{2i\pi k\theta}\Big]_{\theta=j/2^{\mathtt{p}}} = f(j/2^{\mathtt{p}}), \qquad j = 0, ..., 2^{\mathtt{p}}-1 \tag{$**$}$$

(qui traduit le fait que la fonction polynomiale trigonométrique $(*)$, avec $M = 2^{\mathtt{p}-1}$, interpole la fonction `f` aux $2^{\mathtt{p}}$ points $j/2^{\mathtt{p}}$, $j = 0, ..., 2^{\mathtt{p}}-1$, régulièrement espacés dans $[0,1]$) se lit aussi

$$\sum_{k=0}^{2^{\mathtt{p}}-1} v_k\, \overline{W_{2^{\mathtt{p}}}{}^{kj}} = (-1)^j f(j/2^{\mathtt{p}}), \quad j = 0, ..., 2^{\mathtt{p}}-1 \tag{$***$}$$

si l'on pose $v_k = u_{k-2^{\mathtt{p}-1}}$ pour k entre 0 et $2^{\mathtt{p}}-1$ et $W_{2^{\mathtt{p}}} := \exp(-2i\pi/2^{\mathtt{p}})$. Ouvrez un fichier `.m` sur lequel vous rédigerez (en utilisant soit la procédure `fft`$(.,2^{\mathtt{p}})$, soit la procédure `CooleyTukey(.,p)` construite à l'exercice 4) une fonction

```
function U=InterpolTrigo1(f,p)
```

qui renvoie en sortie, sous forme d'une colonne, la liste des coefficients v_k, $k = 0, ..., 2^{\mathtt{p}}-1$, satisfaisant au système (†) (correspondant à la liste, dans le même ordre, des coefficients u_k solutions du système $(*)$). Pourquoi au fait ces systèmes $(**)$ ou $(***)$ sont ils des systèmes de Cramer ? Sauvez votre fichier `InterpolTrigo1.m` dans votre répertoire `TPMATLAB`.

(2) Sur un nouveau fichier `.m`, rédigez une procédure

```
function PolTrigo = InterpolTrigo2(f,p,XX)
```

qui calcule, étant donné une fonction `f`, un entier naturel non nul `p` et un vecteur ligne `XX` de nombres flottants de $[0,1]$, la liste (en ligne) des valeurs prises aux entrées de `XX` par la fonction polynomiale trigonométrique $(*)$ (avec $M = 2^{\mathtt{p}-1}$) dont les coefficients u_k (dans cet ordre) sont ceux fournis en colonne par la commande

```
>> U= InterpolTrigo1(f,p);
```

Testez votre routine sur un polynôme trigonométrique `f1` (mais non 1-périodique) déclaré en ligne

```
>> f5= inline('4*sin(pi/2*x) + cos(5*x) - 2*sin(3*x)','x');
```

suivant les instructions :

```
>> XX=0:1/2000:1;
>> PXXT=InterpolTrigo2(f5,p,XX);
>> plot(XX,f5(XX),'r')
>> hold
>> plot(XX,PXXT)
```

Prenez pour cela des valeurs de `p` entre `6` ($2^p = 64$) et `10` ($2^p = 1024$). Qu'observez vous lorsque `p` augmente concernant la qualité de l'approximation de `f` par son polynôme trigonométrique interpolant ? Recommencez avec cette fois la fonction :

```
>> f6= inline('4*sin(pi/2*x) + cos(5*x) - 2*sin(3*x)','x');
```

Après avoir évalué `f(1)-f(0)`, essayez de d'expliquer ce qui crée le phénomène au bords de $[0, 1]$ dans le cas du premier exemple, et ne semble plus le créer (en tout cas de manière aussi nette) ici. Ce phénomène s'appelle, lorsqu'il se produit *phénomène de Gibbs*; c'est le pendant du *phénomène de Runge* observé dans le cadre de l'interpolation polynomiale par le polynôme de Lagrange (Exercice 1). Voir par exemple le site (succint) sur `wikipedia` :

`http://fr.wikipedia.org/wiki/Ph%C3%A9nom%C3%A8ne_de_Runge`

Sauvez votre fichier `InterpolTrigo2.m` dans votre répertoire `TPMATLAB`. Testez aussi votre programme avec une fonction polynomiale :

```
>> f7 = inline('x.^7+3*x.^5-2*x+1','x');
```

(c'est la fonction `f1` de l'exercice 2). Est-il préférable dans ce dernier cas d'utiliser l'interpolation par des polynômes trigonométriques plutôt que l'interpolation de Lagrange par des polynômes ?

(3) Pour mieux vous convaincre du phénomène de Gibbs, déclarez la fonction `f8` définie par

```
>> f8=inline('(1/2)*(sign ((x-1/3).*(2/3-x))+1)','x');
```

Affichez le graphe de `f8` :

```
>> XX=0:1/2000:1;
>> plot(XX,f8(XX));
```

Sur le même graphe, affichez (en utilisant la commande `hold`) avec diverses couleurs les graphes successifs des divers `PXXT` :

```
>> PXXT = InterpolTrigo2(f8,p,XX);
>> hold
>> plot(XX,PXXT,'color');
```

avec `p` variant de 6 à 10 et `color= b,g,m,r,k`. Quel constat faites vous ? Pourquoi est-il nécessaire pour observer le phénomène de Gibbs d'avoir pris un pas $1/2000 < 2^{-10}$ pour le vecteur ligne `XX` où sont évaluées la fonction et son polynôme d'interpolation trigonométrique ? Le phénomène de Gibbs apparait dans la pratique dès que l'on tente de couper les hautes fréquences d'une fonction oscillante, mais présentant tout de même des discontinuités (des « *cracks* »). On rehausse (ou affaiblit) artificiellement la fonction avant ou après le passage par une discontinuité. Le repiquage audio de vieux enregistrements (très bruités, donc contenant des composantes haute-fréquence) se heurte constamment à ce problème, que l'on tente de corriger en électronique, et que l'on appelle l'*aliasing*.

TP6 : le principe de l'élimination algébrique sous `Maple`

Cette séance de TP6 s'inscrit dans la poursuite de la familiarisation avec le logiciel de calcul symbolique `Maple12`. Elle vise à illustrer la section 3.4 des notes de cours (élimination au travers de la construction du déterminant de Sylvester). On y verra les fondements de l'intégration symbolique des fonctions rationnelles correspondant aux élémentsd de $\mathbb{Q}(X)$ *via* leur décomposition en éléments simples (méthode de Rothstein-Trager, 1976). Ouvrez pour commencer une feuille de travail vierge sous `Maple12`.

Exercice E.6 (Calculs de discriminants ordinaires). Pourquoi (à la lumière du cours) existe-t'il, pour tout entier $d \geq 2$, un polynôme Δ_d en $d-1$ variables $X_1, ..., X_{d-1}$ tel que l'on ait l'équivalence suivante :

$$\Big(z \in \mathbb{C}^{d-1} \text{ et } \Delta_d(z_1, ..., z_{d-1}) = 0\Big)$$
$$\iff \Big(X^d + z_1 X^{d-1} + \cdots + z_{d-1} X - 1 = 0 \text{ a une racine double dans } \mathbb{C}\Big).$$

Ce polynôme Δ_d (en $d-1$ variables $X_1, ..., X_{d-1}$) s'appelle le *discriminant ordinaire d'ordre d* ; il joue un rôle important dans les branches des mathématiques en relation avec la théorie des nombres, la géométrie algébrique ou différentielle, la théorie des équations différentielles, celle des fonctions spéciales (en particulier des fonctions sommes de séries entières $z \mapsto \sum_{k\geq 0} a_k z^k$ particulières, dites *hypergéométriques*), la combinatoire, *etc.*

(1) Calculez Δ_2 à la main (à un coefficient près).

(2) Étant donnés deux polynômes `P` et `Q` déclarés sous `Maple12` comme des polynômes en les d variables $X_1, ..., X_{d-1}, X$, dites à quoi correspond la routine

```
>> RX:= resultant(P,Q,X);
```

(3) Soient les quatre polynômes respectivement en trois variables `x,y,X`, en quatre variables `x,y,z,X`, en cinq variables `x,y,z,t,X`, en six variables `x,y,z,t,u,X` déclarés ainsi :

```
P3:= X^3 + x*X^2 + y*X -1 ;
P4:= X^4 + x*X^3 + y*X^2 + z*X -1;
P5:= X^5 + x*X^4 + y*X^3 + z*X^2 + t*X -1;
P6:= X^6 + x*X^5 + y*X^4 + z*X^3 + t*X^2 +u*X-1;
```

Existe-t'il un moyen rapide de déclarer un polynôme de cette liste à partir du polynôme qui le précède dans la liste ? En combinant l'utilisation des trois commandes

```
>> Q:=diff(P,X);
>> R:=resultant(P,Q,X);
>> degree(R);
```

calculez les discriminants ordinaires $\Delta_3, \Delta_4, \Delta_5, \Delta_6$, ainsi que le degré total de ces trois polynômes. Essayez de deviner une formule donnant le degré de Δ_d, puis justifiez cette formule en vous souvenant que Δ_d doit se représenter comme un certain déterminant de Sylvester ; lequel ?

Sauvez votre feuille de travail comme `TP6exo1.mw` dans votre répertoire `TPMaple12`, puis fermez là.

EXERCICE E.7 (intersection de deux courbes planes). Une courbe plane définie dans $\mathbb{C} \times \mathbb{C}$ (espace vectoriel de dimension 2 sur $\mathbb{C}$) comme le lieu des zéros d'un polynôme en deux variables de degré total égal à 3 est appelée *cubique* (tandis qu'elle est appelée *conique* si elle est définie comme le lieu des zéros d'un polynôme de degré total égal à 2). Ouvrez une nouvelle feuille de travail sous `Maple12`.

(1) Vérifiez que les sous-ensembles de $\mathbb{C} \times \mathbb{C}$ définis par

$$\Gamma_1 := \Big\{(x,y) \in \mathbb{C} \times \mathbb{C}\,;\, xy(y-x) - y^2 + 6x - y - 6 = 0\Big\}$$

$$\Gamma_2 := \Big\{(x,y) \in \mathbb{C} \times \mathbb{C}\,;\, (y-2x)^2(y-3x) - (x+y)^2 - x - y - 1 = 0\Big\}$$

sont bien des cubiques et que le seul zéro commun des parties homogènes de plus haut degré des polynômes

```
P:= X*Y*(Y-X) - Y^2  + 6*X - Y - 6 ;
Q:= (Y-2*X)^2*(Y-3*X) - (X+Y)^2 - X -Y -1 =0;
```

définissant ces courbes est l'origine $(0,0)$ de $\mathbb{C}^2$. On admettra que dans ce cas (c'est un important théorème de géométrie algébrique attribué à Etienne Bézout) les deux cubiques Γ_1 et Γ_2 se coupent en exactement $9 = 3 \times 3$ (le produit des degrés totaux de leurs équations définissantes `P=0` et `Q=0`) points de $\mathbb{C} \times \mathbb{C}$ (les points d'intersection des deux courbes pouvant être éventuellement multiples).

(2) Utilisez l'affichage graphique [3]

```
>> with(plots);
>> implicitplot([P=0,Q=0],X=-10..10,Y=-10..10,
   color=[red,blue],gridrefine=2);
```

pour tracer les intersections des deux courbes Γ_1 et Γ_2 avec le monde réel $\mathbb{R}^2$. Qu'observez vous ? Les deux courbes réelles se coupent-elles et en combien de points visibles ? Se coupent t'elles transversalement ? Que peut-on dire des autres points d'intersection des deux courbes Γ_1 et Γ_2 (puisque l'on sait *a priori* qu'il doit y en avoir 9 dans $\mathbb{C}^2$) ?

(3) En utilisant le principe de l'élimination (et donc la commande `resultant`)
- d'abord pour éliminer la variable `X` entre les équations `P(X,Y)=0` et `Q(X,Y)=0`,
- ensuite pour éliminer la variable `Y` entre ces deux mêmes équations,

3. Prenez soin de ≪ décortiquer ≫ au préalable le sens des options de la routine `implicitplot` (affichant une courbe dans $\mathbb{R}^2$ donnée par son équation cartésienne) après avoir fait `?implicitplot` dans votre feuille de travail pour analyser toutes les potentialités de cette commande `implicitplot`.

puis la routine

```
>> zeros :=[fsolve(R,T,complex)];
```

(qui donne sous forme de liste les d zéros complexes, calculés de manière approchée, d'un polynôme $R \in \mathbb{C}[T]$ de degré d) déterminer, concernant les points d'intersection (x, y) des deux courbes Γ_1 et Γ_2 dans $\mathbb{C} \times \mathbb{C}$:

- 9 valeurs possibles pour x, parmi lesquelles on observe que se trouve une et une seule valeur réelle x_0 ;
- 9 valeurs possibles pour y, parmi lesquelles on observe que se trouve une et une seule valeur réelle y_0.

Déduisez-en les valeurs approchées des coordonnées de l'unique point de $\mathbb{R}^2$ appartenant aux deux cubiques Γ_1 et Γ_2. Le résultat est il bien conforme à ce que vous avez observé graphiquement à la question 2 ?

(4) Rédigez une procédure

```
testzeros = proc(epsilon,P,Q,LX,LY)
```

qui, étant donnés deux polynômes en deux variables `P` et `Q` et deux listes de nombres complexes `LX`, `LY` (déclarés en flottants), renvoie la liste des couples `(L[i],L[j])` en lequels l'évaluation de la fonction $\sqrt{|P|^2 + |Q|^2}$ est strictement inférieure au seuil ϵ. En prenant ϵ convenable (essayez 10^{-k}, k variant entre 3 et 8), exécutez ce code `testzeros` avec comme autres entrées `P`, `Q`, et les deux listes `LX`, `LY` de 9 nombres complexes construites à la question 3, pour obtenir la liste des valeurs approchées des 9 couples de nombres complexes correspondant aux positions des 9 points d'intersection des cubiques Γ_1 et Γ_2 cette fois dans $\mathbb{C} \times \mathbb{C}$. Retrouvez vous bien le point réel observé à la question 2 et calculé de manière approché à la question 3 ?

(5) On considère, à la place de la cubique Γ_2, la cubique Γ_3 définie par

$$\Gamma_1' := \Big\{(x, y) \in \mathbb{C} \times \mathbb{C}\,;\, xy(x - y) - x^2 + 6y - x - 6 = 0\Big\}.$$

Par quelle transformation simple de $\mathbb{C} \times \mathbb{C}$ dans lui-même la cubique Γ_1' se déduit-elle de la cubique Γ_1 ? Vérifiez que le lieu des zéros communs des parties homogènes de plus haut degré des polynômes définissant Γ_1 et Γ_1' est constitué cette fois de trois droites de $\mathbb{C} \times \mathbb{C}$: on dira alors que les deux cubiques Γ_1 et Γ_1' se coupent en trois points à l'infini de $\mathbb{C} \times \mathbb{C}$ (ces trois « points à l'infini » se situant à l'infini précisément lorsque l'on suit l'une des trois directions données par les trois droites vectorielles que l'on vient de trouver). Reprenez les questions 2,3,4 précédentes avec cette fois Γ_1' à la place de Γ_2 pour vérifier que les deux cubiques Γ_1 et Γ_1' se coupent en $6 = 9 - 3$ points distincts dans $\mathbb{C} \times \mathbb{C}$. On calculera cette fois ces points algébriquement et de manière non plus approchée, mais exacte (utilisez la routine `solve` au lieu de `fsolve`). Combien parmi ces points sont-t'ils des points à coordonnées réelles ?

Sauvez votre feuille de travail comme `TP6exo2.mw` dans votre répertoire `TPMaple12`, puis fermez la.

EXERCICE E.8 (construction de la matrice et du déterminant de Sylvester). Dans cet exercice, on vous propose de revenir aux sources et de rédiger une procédure

qui fournisse, étant donnés en `input` deux polynômes `P` et `Q` en une variable `X` (de degrés respectifs p et q, à coefficients pouvant dépendre de manière polynomiale ou rationnelle d'autres variables `Y,Z...`), fournisse en `output` le déterminant de la matrice de Sylvester (de taille $(p+q, p+q)$, voir (3.14) dans la section 3.4 du polycopié de cours) de `P` et `Q`, considérés comme des polynômes en `X`. Ouvrez une nouvelle feuille de travail sous `Maple12`.

(1) Chargez les bibliothèques suivantes :

```
>> with(PolynomialTools);
>> with(LinearAlgebra);
```

et étudiez ce que font les routines :

```
>> ?CoefficientList;
>> ?Determinant;
```

ainsi que la syntaxe régissant leur utilisation. Consultez aussi

```
>> ?Matrix;
>> ?MVassignment;
```

pour voir comment modifier une matrice `S` de taille $D \times D$ donnée composée initialement de zéros (déclarée par `S:=Matrix(D)`).

(2) Rédigez une procédure

```
Sylvester := proc(P,Q,X)
```

fournissant, lorsque `P` et `Q` sont deux polynômes non nuls de degrés strictement positifs en la variable `X`, la matrice de Sylvester (*i.e* la matrice (3.14) des notes de cours).

(3) Rédigez une procédure

```
resultantTP := proc(P,Q,X)
```

fournissant, lorsque `P` et `Q` sont deux polynômes non nuls de degrés strictement positif en la variable `X`, leur résultant. Recalculez les déterminants ordinaires Δ_3 et Δ_4 de l'exercice 1 (question 3) avec cette fois votre routine `resultantTP` et comparez avec les résultats obtenus à l'exercice 1 (question 3) avec la routine (intégrée au logiciel) `resultant`.

(4) Étant donnés deux polynômes `P` et `Q` de degrés respectifs $p > 0$ et $q > 0$, montrez que l'on ne change pas le déterminant de la matrice de Sylvester (matrice (3.14) dans les notes de cours) en remplacant le dernier vecteur colonne par le vecteur colonne :

$$\begin{bmatrix} \mathtt{X}^{\mathtt{q}-1} * \mathtt{P(X)} \\ \vdots \\ \mathtt{X} * \mathtt{P(X)} \\ \mathtt{P(X)} \\ \mathtt{X}^{\mathtt{p}-1} * \mathtt{Q(X)} \\ \vdots \\ \mathtt{X} * \mathtt{Q(X)} \\ \mathtt{Q(X)} \end{bmatrix} \qquad (*)$$

(pensez pour cela à ajouter au dernier vecteur colonne la somme de tous les vecteurs colonnes précédents multipliés par $\mathtt{X}^{\mathtt{p}+\mathtt{q}-1}$, $\mathtt{X}^{\mathtt{p}+\mathtt{q}-2}$, ...). Réalisez une procédure

```
resultantTPmodif:=proc(P,Q,X,S,T);
```

qui, étant donnés deux polynômes `P` et `Q` en une variable `X` (de degrés strictement positifs `p` et `q`, les coefficients de ces polynômes pouvant dépendre de manière polynomiale ou rationnelle d'autres variables `Y,Z`...), calcule le déterminant de la matrice `Sylvester(P,Q,X)` modifiée par le fait que la dernière colonne y ait été remplacée par

$$\begin{bmatrix} \mathtt{X}^{\mathtt{q}-1} * \mathtt{S} \\ \vdots \\ \mathtt{X} * \mathtt{S} \\ \mathtt{S} \\ \mathtt{X}^{\mathtt{p}-1} * \mathtt{T} \\ \vdots \\ \mathtt{X} * \mathtt{T} \\ \mathtt{T} \end{bmatrix}, \qquad (**)$$

où `S` et `T` sont deux nouvelles variables. La sortie de cette procédure doit donc être un polynôme en les variables `X`, `S`, `T` et les coefficients des polynômes `P` et `Q`. Vous pourrez au préalable consulter l'aide

```
>>?Vector;
```

pour voir comment déclarer le vecteur colonne (**). Calculez, étant donnés deux polynômes `P` et `Q` comme ci-dessus

```
>> R:= resultantTPmodif(P,Q,X,S,T);
>> resultant(P,Q,X) - simplify(P*diff(R,S)+Q*diff(R,T));
```

Expliquez d'où ce résultat provient. Si le résultant de `P` et `Q` par rapport à `X` est non nul, pourquoi obtient-on ainsi une identité de Bézout `1=A*P+B*Q` avec $\deg \mathtt{A} < \mathtt{q}$ et $\deg \mathtt{B} < \mathtt{q}$?

Sauvez votre feuille de travail comme `TP6exo3.mw` dans votre répertoire `TPMaple12`, puis fermez la.

EXERCICE E.9 (Intégration symbolique de fractions rationnelles et élimination). Les méthodes reposant sur l'élimination algébrique ainsi que sur la division euclidienne (algorithme de Bézout étendu) jouent un rôle clef dans le noyau de logiciels de calcul symbolique tels `Maple12` lorsqu'il s'agit d'effectuer l'intégration symbolique des fractions rationnelles. C'est ceci que cet exercice se propose d'illustrer. Ouvrez une nouvelle feuille de travail sous `Maple12`.

(1) Soit Q un polynôme en une variable à coefficients entiers, de degré strictement plus grand que 1, n'ayant aucune racine double dans $\mathbb{C}$. Pourquoi le résultant (par rapport à X) de Q et $Q' := dQ/dX$ est-il non nul ? Pourquoi existe-t-il bien dans ce cas deux polynômes U et V à coefficients rationnels, avec de plus $\deg(U) \leq \deg Q - 2$ et $\deg V \leq \deg Q - 1$ tels que l'on ait l'identité $Q(X)U(X) + Q'(X)V(X) \equiv 1$?

(2) Regardez la syntaxe de la routine `gcdex` :

```
>> gcdex(P1,P2,X,'U','V');
>> U;
>> V;
>> simplify(U*P1+V*P2);
```

Prenez un polynôme `Q` de degré `3` sans racine double et à coefficients entiers, par exemple

```
>> Q:= X^3 - 2*X^2 + 3*X+5;
```

et calculez les polynômes `U` et `V` tels que `U*Q+V*diff(Q,X)=1` donnés par la routine `gcdex`. Calculez d'autre part

```
>> resultant(Q,diff(Q,X),X);
```

Qu'observez vous ? Est-ce une surprise et avez vous une explication (pensez ràregarder ici les notes de cours, section 3.4) ?

(3) Vérifiez, si P et Q sont deux polynômes (avec $\deg Q > 0$ et Q sans racine double dans $\mathbb{C}$) que, si U et V sont tels que l'on ait l'identité de Bézout $1 = UQ + VQ'$, alors, pour tout entier $n > 1$,

$$\int^X P(t)\,\frac{dt}{(Q(t))^n} = \frac{1}{1-n}\,\frac{V(X)P(X)}{((Q(X))^{n-1}} + \\ + \int^X \Big(P(t)U(t) + \frac{1}{n-1}\frac{d}{dt}[PV](t)\Big)\,\frac{dt}{(Q(t))^{n-1}}.$$

(4) En utilisant la procédure `gcdex` (algorithme d'Euclide étendu résolvant l'identité de Bézout, voir la question 2) et la procédure `int`

```
>> F:=int(P,X);
```

appliquée aux polynômes (qui fournit une primitive d'un polynme en X et qu'il vous serait aisé de construire vous-même), rédigez une procédure récursive

```
HermiteItere:=proc(P,Q,X,n)
```

qui, étant donnés deux polynômes `P`, `Q` en la variable `X` (avec `Q` de degré strictement positif et sans racine double) à coefficients entiers retourne une liste `[R,H]`, où `R` est un élément de $\mathbb{Q}$(`X`) de dénominateur $\mathtt{Q}^{n-1}$ et `H` un élément de $\mathbb{Q}$(`X`) de degré strictement inférieur au degré de `Q` tels que l'on ait :

$$\int^{\mathtt{X}} \frac{\mathtt{P}(t)}{(\mathtt{Q}(t))^n}\,dt = \mathtt{R(X)} + \int^{\mathtt{X}} \frac{\mathtt{H}(t)}{\mathtt{Q}(t)}\,dt.$$

(5) Soient `H` et `Q` deux polynômes à coefficients rationnels tels que $0 \leq \deg \mathtt{H} < \deg \mathtt{Q} = d$ et

```
>> RX := resultant(H-Y*diff(Q,X),Q,X);
```

Pourquoi `RX` est-il un polynôme en `Y` à coefficients rationnels ? Quel est son degré ? Soient $\alpha_1, ..., \alpha_d$ les d racines (complexes, mais algébriques, toutes simples par hypothèses) de `Q`. Montrez (en revenant à la définition du résultant) que :

$$\mathtt{RX(z)} = 0 \Longleftrightarrow \Big(\exists\, l \in \{1, ..., d\} \text{ t.q. } \mathtt{z} = \frac{\mathtt{H}(\alpha_l)}{\mathtt{Q}'(\alpha_l)}\Big).$$

Montrez que, si `z` est une racine de `RX`, alors le polynôme

```
>>  Rz:= gcdex(H-z diff(Q,X),Q,X);
```

(PGCD des deux polynômes en `X` que sont `H-zQ'` et `Q`) a exactement pour facteurs le produit des polynômes $\mathtt{X}-\alpha_l$, où les α_l sont les nombres complexes pris dans la liste $\{\alpha_1,...,\alpha_d\}$ et tels que $\mathtt{z}=\mathtt{H}(\alpha_l)/\mathtt{Q}'(\alpha_l)$.

(6) Soient `H` et `Q` comme à la question 4. En se basant sur la formule (décomposition en éléments simples)

$$\frac{\mathtt{H(X)}}{\mathtt{Q(X)}}=\sum_{j=1}^{d}\frac{\mathtt{H}(\alpha_j)}{\mathtt{Q}'(\alpha_j)}\,\frac{1}{\mathtt{X}-\alpha_j}, \qquad (***)$$

vérifiez qu'une primitive de `H/Q` s'obtient comme

$$\sum_{\mathtt{RX(z)}=0}\mathtt{z}\ln\big(\mathtt{Rz(X)}\big),$$

les zéros de `RX` étant considérés ici comme distincts, c'est-à-dire comptés sans leur multiplicité. Ceci est très important, c'est d'ailleurs tout le « sel » de la méthode, outre le fait que tous les calculs faits ici (hormis la recherche des zéros de `RX` sont des calculs impliquant des polynômes à coefficients entiers ou algébriques (mais jamais traités comme des nombres complexes).

(7) Après avoir examiné la syntaxe de la routine `sqrfree`

```
>>?sqrfree;
```

construire une procédure

```
ReduceFormPol:= proc(R,X)
```

qui, étant donné un polynôme `R` à coefficients rationnels en la variable `X`, renvoie un polynôme en la variable `X` à coefficients entiers ayant dans $\mathbb{C}$ les mêmes zéros que `R`, mais cette fois tous simples (donc distincts).

(8) Combinez la procédure `HermiteItere` réalisée à la question 4 avec le résultat établi à la question 6 ainsi que la procédure `ReduceFormPol` réalisée à la question 7 pour réaliser cette fois une procédure

```
IntSymb := proc(P,Q,X,n)
```

pour la fraction rationnelle $\mathtt{P}/\mathtt{Q}^{\mathtt{n}}$. La procédure que vous avez rédigé ici est due à Michael Rothstein et Barry Trager (autour de 1976). Elle est devenue aujourd'hui une brique de base essentielle en calcul symbolique.

(9) Testez la procédure `IntSymb` pour calculer les primitives suivantes :

$$\int^{\mathtt{X}}\frac{3t^2+5}{t^4+1}\,dt \qquad , \qquad \int^{\mathtt{X}}\frac{3t^2+1}{(t^4+1)^5}\,dt$$

$$\int^{\mathtt{X}}\frac{43t^9+48t^8-116t^7-126t^6-11t^3-26t^2-23t-8}{(t^2-2)(t+1)(t^7+t+1)}\,dt$$

$$\int^{\mathtt{X}}\frac{6t^3+13t^2+28-72t}{t^4-48-8t^2+t^3+4t}\,dt \qquad , \qquad \int^{\mathtt{X}}\frac{dt}{t(1+t^2)}.$$

Calculez dans les cinq cas la dérivée de votre résultat $\mathtt{X}\mapsto\mathtt{Primj(X)}$ avec

```
>> Derj:=normal(diff(Primj,X));
>> Derj:=normal(numer(Derj)/factor(expand(denom(Derj))));
```

(la seconde routine est un peu compliquée, mais je n'ai pas trouvé mieux pour regrouper les facteurs avec radicaux figurant au dénominateur de la fraction rationnelle `Derj` initiale). Calculez aussi le résultant `RXj` et sa factorisation avec les commandes

```
>>RXjbis:=Split(RXj,Y);
>>convert(RXjbis,radical);
```

En conclusion de ces exemples, on voit que la procédure de Rothstein-Trager s'avère d'autant plus intéressante que le résultant `RX` a des zéros multiples. Le nombre de zéros distincts de `RX` peut s'avérer alors nettement plus petit que le degré du polynôme `Q` (qui est aussi le degré en `Y` du résultant `RX`, voir la question 5). De plus, si par chance les racines distinctes de `RX` sont rationnelles (comme dans l'exemple 3), le calcul de la liste des PGCD `Rz` (pour `z` balayant la liste des zéros distincts de `RX`) ne fait pas apparaitre de factorisation intempestive impliquant des radicaux. Si les zéros distincts de `RX` sont calculables, mais complexes, on notera cependant que le regroupement des logarithmes conjugués *via* la formule formelle

$$\texttt{ln}\big(\texttt{X-(a+I*b)}\big) - \texttt{ln}\big(\texttt{X-(a-I*b)}\big) = 2 * \texttt{I} * \texttt{arctan}(\texttt{(X-a)/b})$$

si `a` $\in \mathbb{R}$ et `b` $\in \mathbb{R}^*$ n'est pas opéré ici. Il conviendrait de l'effectuer pour avoir une forme simplifiée (et réelle) de l'expression de la primitive obtenue, ce que nous ne ferons pas faute de temps. Vous pouvez ajouter cette opération supplémentaire de concaténation à titre d'exercice.

Sauvez votre feuille de travail comme `TP6exo4.mw` dans votre répertoire `TPMaple12`, puis fermez la.

ANNEXE F

TP7 : le théorème du point fixe en action sous MATLAB

Cette séance de TP7 poursuit la familiarisation avec MATLAB. Elle illustre le chapitre 4 du cours (le théorème du point fixe et ses applications en algèbre linéaire). Ouvrez MATLAB pour commencer.

EXERCICE F.1 (l'algorithme itératif conduisant au calcul approché du rayon spectral d'une matrice carrée réelle).

(1) Déclarez sous MATLAB la matrice symétrique réelle (de taille $(4,4)$) suivante :

```
>> A
A =

    0.5172    0.5473   -1.2240    0.8012
    0.5473    1.3880    1.3530   -1.1120
   -1.2240    1.3530    0.0364    2.8930
    0.8012   -1.1120    2.8930    0.0583
```

En utilisant la commande `eig(A)`, calculez les quatre valeurs propres réelles de cette matrice réelle symétrique `A`. Cette matrice `A` est-elle définie positive ? Est-elle diagonalisable ? Quel est le signe de la valeur propre de valeur absolue égale au rayon spectral `r(A)` de cette matrice ? Quelle est la dimension (dans $\mathbb{R}^4$) du sous-espace propre correspondant ?

(2) Téléchargez depuis le site

`http://www.math.u-bordeaux1/~yger/initiationMATLAB`

la routine `rayonspectral`. Modifiez ensuite cette routine pour réaliser une routine

`[r,Niter] = rayonspectral1(A,X,epsilon,k);`

qui, étant donné un nombre maximal `k` d'itérations autorisées, une matrice carrée (réelle ou complexe) de taille `(N,N)` `A`, un vecteur colonne `X` de longueur `N`, calcule de manière approchée, dans les cas favorables, le rayon spectral `r` de la matrice `A`, l'algorithme étant initié à `X`, ainsi que le nombre `Niter` $\leq$ `k` d'itérations nécessaires avant que l'erreur entre une approximation et la suivante ne vienne à passer en valeur absolue sous le seuil `epsilon` [1]. Testez cet algorithme sur la matrice `A` de la question 1 en prenant `k` $= 100$, `epsilon=eps`, et pour `X` respectivement

1. Cela ne donne pas *a priori* une majoration par `epsilon` de l'erreur commise entre le rayon spectral et sa valeur approchée (au terme de `Niter` itérations), mais seulement un contrôle de

```
>> X1 = [1;1;1;1];
>> X2 = [1;1;0;0];
>> X3 = [1;0;0;0];
>> X4 = rand(4,1);
```

Quelle valeur de `Niter` trouvez vous dans chacun de ces cinq cas ? Vérifiez que la valeur de `r` obtenue alors est bien en accord avec le résultat fourni à la question 1 par la commande `eig(A)` donnant les quatre valeurs propres (dans ce cas réelles) de la matrice `A`. Recommencez avec cette fois `epsilon` $= 10^{-6}$.

(3) On suppose que `A` est une matrice réelle et que le rayon spectral `r(A)` est la valeur absolue d'une valeur propre réelle simple de la matrice `A`. Vérifiez que c'est bien le cas pour la matrice `A` donnée à la question 1. Modifiez la routine `rayonspectral1` en une routine `AppVPdom` (« Approximation du Vecteur Propre associé à la valeur propre dominante ») :

```
[VP,Niter] = AppVPdom(A,X,epsilon,k);
```

qui fournisse, avec les mêmes données que dans `rayonspectral1` en *input*,

- le vecteur $\texttt{VP}_{\texttt{Niter}+1}$ de la suite initiée à $\texttt{X}_1 = \texttt{X}/\texttt{norm(X)}$ et régie ensuite par l'équation récurrente
$$\texttt{VP}_{k+1} = \frac{\texttt{A} \cdot \texttt{VP}_k}{\texttt{norm}\big(\texttt{A} \cdot \texttt{VP}_k\big)}, \quad k \geq 1.$$
- le nombre d'itérations `Niter` effectué dans la boucle sachant que cette boucle s'arrête dès que, pour la première fois :
```
min (norm(VP_(Niter) -VP_(Niter-1)),
     norm (VP_(Niter)+ VP_(Niter-1))) <= epsilon
```
Testez cet algorithme avec la matrice `A` en prenant `epsilon=eps`, `k=200` et respectivement `X = X1,X2,X3,X4` comme à la question 2. Quelle valeur de `Niter` obtenez vous dans chacun de ces quatre cas ? Recommencez avec cette fois `epsilon` $= 10^{-6}$. Calculez les vecteurs propres (normalisés) de la matrice `A` en utilisant :
```
>> [V,D] = eig(A);
>> V
```
Comparez le vecteur propre `Y=V(:,1)` ainsi obtenu (correspondant à la valeur propre de valeur absolue `r(A)`) et le vecteur `VP` obtenu *via*
```
>> [VP,Niter] = AppVPdom(A,X,epsilon,k);
```

(4) Modifiez la routine `AppVPdom` construite à la question 3 en une nouvelle routine

```
function [VP,Niter,errY] = AppVPdom1(A,X,Y,epsilon,k);
```

qui, en plus des sorties `VP` et `Niter` (comme pour la routine `AppVPdom` construite à la question 3), fournit aussi la liste `errY` des erreurs successives [2]

---

cette erreur à un facteur multiplicatif près, ce contrôle étant de l'ordre de `epsilon`$/(1-\rho)$, où ρ désigne le rapport entre `r(A)` et le module de la première valeur propre de module strictement inférieur à `r(A)`. Voir pour cela la preuve de la Proposition 4.2 du cours.

2. Même remarque que précédemment à propos du contrôle d'erreur : cette tolérance `epsilon` contrôle l'erreur entre $\texttt{VP}_{\texttt{Niter}}$ et un vrai vecteur propre normalisé (pour la valeur propre $\pm$ `r(A)`) en `epsilon`$/(1-\rho)$, où ρ a été défini dans la note 1 précédente.

`min(norm(VP_k-Y), norm(VP_k+Y)),  k=1,2,...,Niter`

lorsque `Y` est un vecteur colonne de $\mathbb{R}^4$ donné en *input* en plus des données précédentes `A,X,epsilon,k`. En utilisant cette routine avec `epsilon` $= 10^{-6}$, `Y=V(:,1)` (vecteur propre normalisé de la matrice `A` correspondant à la valeur propre de valeur absolue `r(A)`) et `X=X1,X2,X3,X4`, comparez la rapidité de la convergence de la suite $(\texttt{VP}_k)_k$ vers $\pm$`Y` dans les quatre cas de figure (suivant la valeur du vecteur initial `X` depuis lequel est lancé l'algorithme).

(5) Reprendre la question 4 en remplacant dans la routine l'utilisation de la norme euclidienne `norm` par la norme $\|\ \|_\infty$ (`norm(.,inf)` sous `MATLAB`). On notera la nouvelle routine (obtenue en modifiant à la marge la routine `AppVPdom1` de la question 4) `AppVPdom1bis`.

EXERCICE F.2 (un schéma simpliste pour `Pagerank`). Cet exercice met en œuvre l'approche proposée dans la section 4.1.2 des notes de cours (≪ maquette ≫ simpliste du fonctionnement de `Google`).

(1) Construire une routine

```
function G=AdjacencePonderee(M);
```

qui, étant donnée une matrice `M` de taille `(N,N)` dont les entrées sont constituées de `0` et de `1` (considérée comme la *matrice d'adjacence* d'un certain graphe orienté `(E,V)`), calcule la *matrice d'adjacence pondérée* de ce même graphe orienté, c'est-à-dire la matrice `G` déduite de la matrice `M` en transformant chaque ligne de `M` ainsi :

- une ligne constituée entièrement de `0` reste inchangée ;
- une ligne dans laquelle figure au moins un `1` est divisée par le nombre de `1` présents sur cette ligne.

(2) Rédigez une procédure

```
function [Lequilibre,Niter]
          =Pagerank(M,L0,kappa,epsilon,k);
```

qui, étant donné un graphe orienté `(E,V)` à `N` sommets, matérialisé par sa matrice d'incidence `M`, un nombre `kappa` strictement entre `0` et `1`, calcule de manière approchée le vecteur ligne `Lequilibre` (de longueur `N`), unique point fixe de l'application strictement contractante :

```
L = ((1-kappa)/N)*ones(1,N) + kappa*L*G
```

lorsque :

- `G` désigne la matrice d'adjacence pondérée du graphe orienté `(E,V)` (*cf.* la question 1) ;
- l'algorithme est initié avec le vecteur ligne `L0` (dont les entrées sont positives et de somme 1) ;
- le nombre maximal d'itérations autorisées est `k` ;
- le nombre `Niter` $\leq$ `k` est le nombre d'itérations nécessaires (lorsque cela est possible) jusqu'à ce que, pour la première fois [3], `norm(L_(Niter)-L_(Niter-1)) <=epsilon`

3. D'après l'étude faite en cours, *cf.* la preuve du Théorème 4.1 (du point fixe), l'erreur entre $\texttt{L}_{\texttt{Niter}}$ et sa limite est alors majorée par `epsilon/(1-kappa)`.

(3) Générez un graphe orienté à `10` sommets *via* la donnée de sa matrice d'adjacence `M` et calculez la mesure d'équilibre `Lequilibre` de ce graphe orienté avec le choix de `kappa=0.85` (le choix classiquement privilégié dans l'algorithme `Pagerank`). Prendre `epsilon` $= 10^{-8}$, puis `epsilon=eps`, calculez aussi `Niter` (lorsque `k=100`) et examinez la dépendance en le choix du vecteur ligne initial `L0` : on pourra prendre par exemple pour ce faire comparer les résultats
   - une liste initiale « creuse » telle `L0=[1 0 0 ... 0]` ;
   - une liste initiale « pleine » telle `L0=L00/sum(L00)` (`L00=rand(1,10)`).

   Affichez les résultats par exemple avec `plot(Lequilibre,'d')`.

Exercice F.3 (algorithmes itératifs de Jacobi et de Gauß-Seidel). Téléchargez depuis le site

`http://www.math.u-bordeaux1/~yger/initiationMATLAB`

les deux routines `Jacobi.m` et `GaussSeidel.m`, respectivement basées sur les décompositions `M=D-E=diag(diag(M))-(diag(diag(M))-M)` et `M=T-F=tril(M)-(tril(M)-M)`.

(1) Transformez ces deux routines en des routines :

```
function [XX,Niter] = Jacobi1(M,B,X,epsilon,k);
function [XX,Niter] = GaussSeidel1(M,B,X,epsilon,k);
```

qui, étant donnés une matrice `M` de taille `(N,N)`, sans zéros sur la diagonale, et un vecteur colonne `B` de longueur `N` :
   - renvoient toutes les deux le message

     `condition non remplie`

     si la condition $\|\mathtt{D}^{-1}\cdot\mathtt{E})\|_\infty < 1$ (ou la condition $||\mathtt{T}^{-1}\cdot\mathtt{F}||_2 < 1$) se trouve en défaut ;
   - génèrent, si la condition se trouve remplie, l'algorithme itératif, soit de Jacobi, soit de Gauß-Seidel, initié sur le vecteur colonne `X` (aussi de longueur `N`), et visant à calculer de manière approchée la solution `XX` du système de Cramer `M*XX=B` ; le nombre maximal d'itérations autorisées est `k` (le même que celui autorisé pour calculer en préambule le rayon spectral des matrices $\mathtt{D}^{-1}*\mathtt{E}$ ou $\mathtt{T}^{-1}*\mathtt{F}$), et l'on décide que l'algorithme itératif s'arrête dès que, pour la première fois [4],

     `norm(XX_(Niter)-XX_(Niter-1)) <=epsilon`

(2) Construisez une routine

```
[XX,Niter] = ExempleJacobi(N,B,X,epsilon,k);
```

qui, étant donné un entier strictement positif `N`, un vecteur `B` de taille `(N,1)` :

4. D'après l'étude faite en cours, *cf.* la preuve du Théorème 4.1 (du point fixe), l'erreur entre $\mathtt{XX}_{\mathtt{Niter}}$ et sa limite (à savoir la solution du système de Cramer que l'on tente d'approcher) est alors majorée par `epsilon/(1-r(A))`, où $\mathtt{A} = \mathtt{D}^{-1}*\mathtt{E}$ ou $\mathtt{A} = \mathtt{T}^{-1}*\mathtt{F}$ suivant le cas (Jacobi ou Gauß-Seidel).

– génère la matrice creuse `M(N)` de taille `(N,N)` dont la diagonale est constituée de `3`, la sur-diagonale et la sous-diagonale de `-1`, toutes les autres entrées de la matrice étant nulles ;
– résout par la méthode de Jacobi (initiée au vecteur `X` de taille `(N,1)`, avec `k` itérations autorisées au plus et un seuil d'erreur `epsilon` comme à la question 1) le système de Cramer $\texttt{M(N)} * \texttt{XX} = \texttt{B}$, où la sortie $\texttt{Niter} \leq \texttt{k}$ désigne toujours le nombre d'itérations réellement effectué lors de l'algorithme de Jacobi.

Faites la même chose en remplaçant l'algorithme de Jacobi par celui de Gauß-Seidel pour construire une fonction similaire :

```
[XX,Niter] = ExempleGaussSeidel(N,B,X,epsilon,k);
```

(l'algorithme de Jacobi ayant cette fois été remplacé par l'algorithme de Gauß-Seidel).

(3) Générez une matrice réelle `A` de taille `(20,20)` en utilisant la routine

```
A = 2*(rand(20)-ones(20)/2);
```

Générez ensuite la matrice `A*A'`. L'algorithme de Gauss-Seidel converge-t-il lorsque la matrice `M` est la matrice `M=A*A'` ? Pourquoi ? Vérifiez le en générant un vecteur `B=rand(20,1)` et en essayant de résoudre le système de Cramer `(A*A')* XX =B` de manière itérative en utilisant la routine

```
[XX,Niter] = GaussSeidel1(A*A',B,zeros(20,1),10^(-8),k);
```

Comparez le résultat que vous obtenez ainsi avec celui donné par la résolution directe

```
>> (A*A')^(-1)*B
```

Comment faut-il choisir `k` pour que les deux résultats soient en cohérence ? Calculez $\texttt{eig}(\texttt{T}^{-1} * \texttt{F})$ pour la décomposition de Gauß-Seidel `A*A'=T-F` et expliquez pourquoi il s'avérait nécessaire de choisir de telles valeurs de `k` pour appliquer l'algorithme de Gauß-Seidel.

(4) Soient les trois matrices :

$$\begin{pmatrix} 1 & .75 & .75 \\ .75 & 1 & .75 \\ .75 & .75 & 1 \end{pmatrix} \qquad \begin{pmatrix} 1 & 2 & -2 \\ 1 & 1 & 1 \\ 2 & 2 & 1 \end{pmatrix} \qquad \begin{pmatrix} 2 & -1 & 1 \\ 2 & 2 & 2 \\ -1 & -1 & 2 \end{pmatrix}.$$

Quels algorithmes (de Jacobi ou de Gauß-Seidel) sont-ils convergeant pour la résolution itérative des systèmes $\texttt{M} * \texttt{XX} = \texttt{B}$ lorsque `M` est l'une de ces trois matrices (étudiez les trois cas séparément) ?

EXERCICE F.4 (le conditionnement des matrices et les facteurs d'amplification d'erreurs relatives dans la résolution des ystèmes de Cramer perturbés).

(1) En utilisant la routine `svd` :

```
>> [U,D,V] =svd(M);
```

donnant la *décomposition en valeurs singulières* d'une matrice réelle ou complexe (*cf.* la section 4.4.1 du cours), écrivez une routine

```
function c=ConditionnementNorme2(M);
```

qui donne la valeur du conditionnement d'une matrice carrée inversible `M` relativement à la norme $\| \ \|_2$. En utilisant cette routine, calculez le conditionnement de la matrice

$$\texttt{A} := \begin{pmatrix} 10 & 7 & 8 & 7 \\ 7 & 5 & 6 & 5 \\ 8 & 6 & 10 & 9 \\ 7 & 5 & 9 & 10 \end{pmatrix}$$

utilisée dans la section 1.2.2 du cours. Comparez avec le résultat donné par `cond(A)`. En utilisant les routines

```
>> help cond
>> cond(A,1)
>> cond(A,inf)
```

calculez le conditionnement de la matrice `A` relativement au choix de la norme matricielle $\| \ \|_1$ et de la norme matricielle $\| \ \|_\infty$.

(2) Perturbez la matrice `A` en lui ajoutant la perturbation :

$$\texttt{DeltaA} := \begin{pmatrix} 0 & 0 & .1 & .2 \\ .08 & .04 & 0 & 0 \\ 0 & -.02 & -.11 & 0 \\ -.01 & -.01 & 0 & -.02. \end{pmatrix}$$

Calculez les solutions `XX` et `XXX` des deux systèmes de Cramer

$$\texttt{A} * \texttt{XX} = \texttt{B}, \qquad (\texttt{A} + \texttt{DeltaA}) * \texttt{XXX} = \texttt{B}$$

si `B=[32;23;33;31]`, d'abord par la méthode directe :

```
>> XX = A^{-1} * B ;
>> XXX= (A+DeltaA)^(-1)*B ;
```

puis par les méthodes itératives :

```
>>  [XX1,Niter] = GausssSeidel(A,B,zeros(4,1),10^(-8),k);
>> [XXX1,Niter] = GaussSeidel(A,B,zeros(4,1),10^(-8),k);
```

Calculez le coefficient d'amplification d'erreur relative :

```
>> (norm(XXX-XX)/norm(XXX))*(norm(A)/norm(DeltaA))
```

Que remarquez vous ? Quel est l'ordre de grandeur de ce coefficient d'amplification des erreurs relatives ?

(3) On perturbe maintenant le vecteur `B` (de la question 2) par le vecteur `DeltaB = [.01;-.01;.01;-.01]`. Résoudre (de deux manières différentes, comme à la question 2 (directement ou alors de manière itérative en utilisant l'algorithme de Gauß-Seidel) le système de Cramer :

$$\texttt{A} * \texttt{XXXX} = \texttt{DeltaB}.$$

Calculez encore le coefficient d'amplification de l'erreur relative :

```
>> (norm(XXXX-XX)/norm(XXXX))*(norm(B)/norm(DeltaB))
```

Que remarquez vous à nouveau ? Quel est l'ordre de grandeur de ce coefficient d'amplification des erreurs relatives ?

Projets (à traiter par binôme ou trinôme)

Présentation : 20 minutes par binôme (10 minutes par orateur) + 5 minutes de questions du jury.

1. **Autour du protocole `RSA`.** Implémentation sous `Maple`. On présentera en particulier en les illustrant par des procédures sous `Maple` les algorithmes d'exponentiation rapide en arithmétique. Le logiciel utilisé sera `Maple`.

2. **Les schémas numériques à l'épreuve de la résolution des équations différentielles ordinaires (EDO).** En utilisant `MATLAB`, on présentera, étayées par des exemples, les méthodes du type Runge-Kutta, et on fera le lien avec les méthodes de Newton-Cotes, exploitées, elles, pour le calcul approché d'intégrales. Comme exemples illustratifs, on utilisera les modèles de systèmes autonomes du type « proie-prédateur ». Le logiciel utilisé sera `MATLAB`. On pourra se servir comme base de l'article de G. Vial [**Vial**] ainsi que du polycopié de cours (chapitre 5).

3. **L'intégration formelle des fractions rationnelles.** Ce au travers de la décomposition en éléments simples, de la méthode d'Hermite et de l'algorithme de Rothstein-Trager (utilisant la théorie de l'élimination et la notion de résultant). Le point de départ pourra être l'exercice 4 de la feuille de TP 6, que l'on approfondira. Le logiciel utilisé sera `Maple`.

4. **Les méthodes de multiplication rapide en analyse et en arithmétique.** On illustrera comment le petit théorème de Fermat fournit un cadre pour envisager en arithmétique (dans $\mathbb{Z}/p\mathbb{Z}$, avec p premier) l'algorithme de Cooley-Tukey (au lieu de $W_N = \exp(-2i\pi/N)$, on prend une racine a de $a^{p-1} = 1$ dont les puissances engendrent $(\mathbb{Z}/p\mathbb{Z})^*$). Le logiciel utilisé sera `Maple`.

5. **Le principe du CAT-Scanner en dimension 2.** On présentera la transformation de Radon digitale, brique de base dans la conception du CAT-Scanner en imagerie médicale. On construira le sinogramme d'une image, c'est-à-dire « l'image vue derrière le rayonnement gamma » qui l'a traversé, puis le principe de la restitution de l'image originelle à partir de son sinogramme. Les images utilisée seront par exemple les « fantômes » dits de Shepp-Logan qui servirent de modèle à Mac Cormack et Hounsfield (prix Nobel de médecine en 1979). Le logiciel utilisé sera `MATLAB`, accompagné du toolbox `ImageProcessing`.

6. **Comment la `fft` intervient-elle dans les algorithmes de compression d'image (`JPEG`) ?** Comment la décomposition de Haar (voir le TP 3) intervient-elle dans les algorithmes de compression plus récents tels `JPEG2000` ? On illustrera par des exemples le principe de ces algorithmes et on les implémentera sur des images de petite taille (256 sur 256 ou 512 sur 512 par exemple). Le logiciel utilisé sera `MATLAB`, accompagné du toolbox `ImageProcessing`. Les données seront téléchargées depuis le web et transformées numériquement sous `MATLAB`. On pourra se reporter

à [**Peyr**] pour une présentation (contenant des routines sous `Scilab`) au traitement numérique des images (voir aussi le document ressources du ministère [**TermS**]).

7. **Le principe de l'interpolation trigonométrique et le phénomène de Gibbs.** Le point de départ de ce projet sera l'exercice 5 de la feuille de TP 5, que l'on approndira avec d'autres exemples de fonctions ayant beaucoup de discontinuités (construites par exemple sur le principe des fonctions fractales). Le logiciel utilisé sera `MATLAB`, accompagné si nécessaire du Toolbox `SignalProcessing`.

8. **Le théorème du point fixe et les moteurs de recherche sur le web (`Google`).** On présentera des modèles de graphes orientés et la recherche de distribution d'équilibre. On pourra s'aider des articles introductifs de M. Eisermann [**Eiser**] ou [**Eiser0**], ou bien du document ressources établi par le Ministère de l'Education Nationale pour les enseignants en Spécialité Mathématiques en Terminale S [**TermS**]. Le logiciel utilisé sera de préférence `MATLAB` (plus adapté au calcul scientifique), mais l'on pourra aussi utiliser `Maple`.

9. **Les méthodes d'accélération de convergence à l'épreuve des approximations du nombre π.** On présentera quelques méthodes conduisant à la recherche de décimales du nombre π et faisant intervenir le principe de l'accélération de convergence (formule de John Machin, exercice 4 du TP4, couplé avec le principe de l'accélération de convergence de Richardson, *etc.*). Le logiciel exploité pour ce projet sera `Maple`.

Annales : examen 2011-2012, session 1 (1h30)

Exercice I. Soient a et ϵ deux nombres réels strictement positifs. Donner la procédure retournant une approximation de $\sqrt{a}$ à ϵ près et fondée sur la méthode de Newton. Quel est l'ordre de cette méthode ?

Exercice II. Soient $x_0 = a < x_1 < \cdots < x_N = b$, $N+1$ nombres equi-espacés de l'intervalle $[a, b]$ et $y_0, ..., y_N$ $N+1$ nombres réels.
a) On note $L[x_0, ..., x_N\,;\, y_0, ..., y_N]$ le polynôme d'interpolation de Lagrange prenant, pour chaque $j = 0, ..., N$, la valeur y_j au point x_j. Quel est le degré de ce polynôme ? Donner une expression algébrique de ce polynôme.
b) Vérifier la formule de Aitken :

$$\begin{aligned} &L[x_0, ..., x_N\,;\, y_0, ..., y_N] \\ &= \frac{(X - x_0)L[x_1, ..., x_N\,;\, y_1, ..., y_N] - (X - x_N)L[x_0, ..., x_{N-1}\,;\, y_0, ..., y_{N-1}]}{x_N - x_0}. \end{aligned}$$

c) En utilisant le résultat établi au **b)**, écrire une procédure récursive aboutissant au calcul du polynôme $L[x_0, ..., x_N\,;\, y_0, ..., y_N]$.

Exercice III. Soit $N \in \mathbb{N}^*$ et $\mathbb{G}$ une matrice (N, N) dont les entrées sont des nombres réels positifs, telle que la somme des termes sur chaque ligne de $\mathbb{G}$ soit égale à 1 (on dit que $\mathbb{G}$ est une matrice *stochastique*).
a) Calculer $\|\mathbb{G}\|_\infty$. Que vaut le rayon spectral de la matrice $\mathbb{G}$?
b) Soit $\kappa \in]0, 1[$. Montrer qu'il existe un unique vecteur ligne $X = (x_1, ..., x_N)$ dans $\mathbb{R}^N$ tel que

$$X = \frac{1 - \kappa}{N}\,(1, ..., 1) + \kappa\, X \cdot \mathbb{G}.$$

Décrire une procédure algorithmique aboutissant au calcul de X. Montrer que les entrées de X sont des nombres réels positifs de somme égale à 1.
c) Soient $\lambda_1, ..., \lambda_N$ des nombres réels positifs tous strictement supérieurs à 1 et $\mathbb{G}_\lambda$ la matrice

$$\mathbb{G}_\lambda := \texttt{diag}\,[\lambda_1, ..., \lambda_N] - \mathbb{G},$$

où $\texttt{diag}\,[\lambda_1, ..., \lambda_N]$ désigne la matrice « diagonale » (N, N) dont toutes les entrées sont nulles, excepté celles sur la diagonale, valant respectivement $\lambda_1, ..., \lambda_N$. Pourquoi les algorithmes itératifs de Jacobi et de Gauß-Seidel (pour la résolution en Y du système linéaire $\mathbb{G}_\lambda \cdot Y = B$, B étant un vecteur colonne donné de $\mathbb{R}^N$) convergent-t'ils tous les deux ? Expliciter la procédure conduisant au calcul de la solution Y de $\mathbb{G}_\lambda \cdot Y = B$ suivant la méthode de Jacobi.

Exercice IV. Soient a, b, c, d quatre nombres réels strictement positifs. On considère le système différentiel « proie-prédateur » :

$$\frac{dx}{dt} = x(t)(a - by(t)), \qquad \frac{dy}{dt} = y(t)(-c + dx(t)),$$

où x et y sont deux fonctions réelles du temps $t \in [0, \infty[$.
a) On suppose $x(0) = x_0 > 0$ et $y(0) = y_0 > 0$. Soit $T > 0$ et $h = T/N$ ($N \in \mathbb{N}^*$). Écrire la procédure conduisant à l'approximation des fonctions $t \mapsto x(t)$ et $t \mapsto y(t)$ sur l'intervalle $[0, T]$, basée sur la méthode d'Euler explicite avec h comme pas.
b) On suppose toujours $x(0) = x_0 > 0$ et $y(0) = y_0 > 0$, et T, h comme à la question précédente. Écrire la procédure conduisant à l'approximation des fonctions $t \mapsto x(t)$ et $t \mapsto y(t)$ sur l'intervalle $[0, T]$, basée sur la méthode d'Euler implicite avec h comme pas.

Annales : examen 2011-2012, session 2 (1h30)

Exercice I.

1. Soit A la matrice 7×7 suivante :

$$A = \begin{pmatrix} -6 & 1 & 2 & -1 & -1/2 & 1/4 & 1/2 \\ 1 & 5 & 1/3 & -1/3 & 0 & -1 & 1/2 \\ 0 & -1 & 4 & 0 & 1/2 & -1/3 & 2 \\ -1/2 & 1 & 1/3 & -3 & 1/2 & 1/3 & 0 \\ 2 & -1 & -1/2 & 1/3 & -5 & 1 & 0 \\ -1/2 & 1 & 1/3 & 2 & -1 & 7 & 1 \\ 2 & -3 & -1 & 1 & 0 & -2 & 10 \end{pmatrix}.$$

Décrire une méthode itérative conduisant à la résolution du système :

$$AX = \begin{pmatrix} 1 \\ 0 \\ -2 \\ -1 \\ 0 \\ 3 \\ 1 \end{pmatrix} \qquad (*)$$

2. Donner une majoration du nombre d'itérations nécessaires pour obtenir une valeur approchée de la solution X de $(*)$ à 10^{-4} près, lorsque la méthode proposée à la question **1** est initiée avec la matrice nulle.

3. La méthode proposée à la question **1** fonctionne-t'elle encore si A est remplacée par la matrice :

$$B = \begin{pmatrix} -5 & 1 & 2 & -1 & -1/2 & 1/4 & 1/2 \\ 1 & 5 & 1/3 & -1/3 & 0 & -1 & 1/2 \\ 0 & -1 & 4 & 0 & 1/2 & -1/3 & 2 \\ -1/2 & 1 & 1/3 & -3 & 1/2 & 1/3 & 0 \\ 2 & -1 & -1/2 & 1/3 & -5 & 1 & 0 \\ -1/2 & 1 & 1/3 & 2 & -1 & 7 & 1 \\ 2 & -3 & -1 & 1 & 0 & -2 & 10 \end{pmatrix} ?$$

En admettant que B est inversible et de transposée ${}^tB$, décrire une méthode itérative permettant de résoudre le système de Cramer :

$$(B \cdot {}^tB)\, X = \begin{pmatrix} 1 \\ 0 \\ -2 \\ -1 \\ 0 \\ 3 \\ 1 \end{pmatrix}.$$

Exercice 2.

1. Soit N un entier strictement positif donné. Donner la définition du polynôme d'interpolation de Lagrange de la fonction $x \mapsto 1/\sqrt{x}$ aux points $x_j = 1 + j/N$, $j = 0, ..., N$.

2. Expliquer le principe de la méthode de Aitken conduisant au calcul récursif de ce polynôme de Lagrange P_N.

3. Donner une majoration de $\sup_{[1,2]} |1/\sqrt{x} - P_N(x)|$.

Exercice 3.

1. Expliciter les méthodes d'Euler explicite, d'Euler implicite et d'Euler modifiée pour la résolution de l'équation différentielle :

$$y'(t) = \sin(y^2(t)) + t^2$$

sur l'intervalle $[0, 1]$, avec la condition initiale $y(0) = 1$. On prendra comme pas $h = 1/N$, où N est un entier strictement positif.

2. Quel est l'ordre de chacune de ces trois méthodes ?

Exercice 4.

1. Soient P et Q les polynômes en deux variables :

$$\begin{aligned} P(X,Y) &= X^3Y^2 + 5X^2Y - X(3Y^2+1) - Y^2 - 1 \\ Q(X,Y) &= X^2(Y^3 + 2Y + 4) - XY + 1. \end{aligned}$$

Soit E l'ensemble des couples (x, y) de points de $\mathbb{C}^2$ tels que $P(x, y) = Q(x, y) = 0$. Déterminer des polynômes A et B d'une variable, à coefficients réels, dont on précisera le degré, tels que :

$$\Big((x,y) \in E\Big) \Longrightarrow \Big(A(x) = B(y) = 0\Big).$$

2. Peut on résoudre par radicaux l'équation $B(y) = 0$ dans $\mathbb{C}$? Que retournerait `MAPLE` si on lui soumettait l'instruction `solve(B(y)=0)` ? Expliciter une méthode approchée d'ordre deux permettant de calculer asymptotiquement les racines réelles de l'équation $B(y) = 0$.

Annales : examen 2012-2013, session 1 (1h30), texte + corrigé

Exercice 1. *Soit* $\mathtt{f} : \mathbb{R} \to \mathbb{R}$ *une fonction numérique de classe* C^∞ *prenant des valeurs rationnelles aux points rationnels,* $\mathtt{a}$ *et* $\mathtt{b}$ *deux nombres rationnels, et* $(x_n)_{n\geq 1}$ *la suite de nombres réels définie par* $x_1 = \mathtt{a}$, $x_2 = \mathtt{b}$ *et, pour tout* $n > 2$,

$$x_n = \begin{cases} x_{n-1} - \mathtt{f}(x_{n-1}) \dfrac{x_{n-1} - x_{n-2}}{\mathtt{f}(x_{n-1}) - \mathtt{f}(x_{n-2})} & \text{si} \quad \mathtt{f}(x_{n-1}) \neq \mathtt{f}(x_{n-2}) \\ \dfrac{x_{n-1} + x_{n-2}}{2} & \text{sinon.} \end{cases}$$

(1) *Dites pourquoi la suite* $(x_n)_{n\geq 1}$ *est une suite de nombres rationnels.*

Ceci se voit par récurrence : en effet l'expression rationnelle de x_n en termes de x_{n-1} et x_{n-2} est une expression rationnelle à coefficients rationnels (dans les deux sous-cas de l'alternative proposée pour la définition de x_n).

(2) *On suppose que* $\mathtt{f}(\mathtt{a})\mathtt{f}(\mathtt{b}) < 0$. *À quelle méthode algorithmique correspond la génération de cette suite* $(x_n)_{n\geq 1}$ *? Si cette suite converge vers un nombre réel* ξ *tel que* $\mathtt{f}'(\xi) \neq 0$ *et n'est pas stationnaire, pourquoi a-t'on* $\mathtt{f}(\xi) = 0$ *? Quelle est l'ordre de convergence de cette méthode algorithmique (on pourra se contenter d'en donner une minoration) ?*

La méthode algorithmique à laquelle correspond la génération de la suite $(x_n)_{n\geq 1}$ est la méthode de la sécante (voir la section 2.1.3 du cours). On reconnait en effet la formule inductive (2.2) sous-tendant cette méthode. Si la suite converge vers un nombre ξ tel que $\mathtt{f}'(\xi) \neq 0$, il résulte de la formule des accroissements finis que

$$\lim_{n\to+\infty} \frac{x_{n-1} - x_{n-2}}{\mathtt{f}(x_{n-1}) - \mathtt{f}(x_{n-2})} = \frac{1}{\mathtt{f}'(\xi)}.$$

Le théorème de Rolle assure d'autre part que, puisque la suite $(x_n)_{n\geq 1}$ n'est pas stationnaire, alors nécessairement $\mathtt{f}(x_n) \neq \mathtt{f}(x_{n+1})$ pour n assez grand : si ce n'était pas le cas, on devrait avoir en effet, pour un tel n, $x_n = x_{n+1}$, car, sinon, il y aurait (par Rolle) un zéro de $\mathtt{f}'$ entre x_n et x_{n+1}, ce qui est impossible du fait que la suite $(x_n)_{n\geq 1}$ est censée converger vers ξ tel que $\mathtt{f}'(\xi) \neq 0$ et que $\mathtt{f}'$ est continue en ce point ξ. Pour n assez grand, on a donc

$$x_n - x_{n-1} = \mathtt{f}(x_{n-1}) \times \Big(\frac{1}{\mathtt{f}'(\xi)} + o(1)\Big) = o(1).$$

On en déduit que

$$\mathtt{f}(\xi) = \lim_{n\to+\infty} \mathtt{f}(x_{n-1}) = 0.$$

L'ordre de convergence de la méthode de la sécante est au moins égal au nombre d'or $(1+\sqrt{5})/2$ d'après le cours (Proposition 2.3) et les TP. En fait, il y a égalité.

(3) *Rédiger, en complétant le code* `Maple` *suivant, une procédure récursive qui calcule* x_N *pour* $N \geq 1$, `a,b` *étant des rationnels fixés (la fonction* `f` *ayant été préalablement déclarée sous* `Maple`*) :*

```
algorithme := proc(f,a,b,N) option remember;
local ...  ;
if N=1 then
... ;
elif N=2 then
... ;
else
    if ... then
    ... ;
    else
    ... ;
    ... ;
    ... ;
    end if;
end if;
end proc;
```

Expliquer pourquoi les deux boucles `if ... end if` *sont essentielles. Que se passerait-il en particulier si l'on omettait la première d'entre elles ? Quels sont à la fois le sens et l'intérêt de l'instruction* `option remember` *?*

Voilà comment compléter le synopsis proposé :

```
algorithme := proc(f,a,b,N) option remember;
local u,v;
if N=1 then
a ;
elif N=2 then
b ;
else
u := algorithme (f,a,b,N-2);
v := algorithme (f,a,b,N-1);
    if f(u)=f(v) then
    (u+v)/2 ;
    else
    v - f(v) *(v-u)/(f(v)-f(u));
    end if;
end if;
end proc;
```

La première boucle `if ... end if` est essentielle car, si on l'omettait, la procédure récursive ne serait pas initialisée et bouclerait donc sur elle-même indéfiniment. La

seconde boucle `if ... end if` est tout aussi essentielle, mais pour une autre raison : afin d'éviter cette fois une intempestive division par zéro lors de l'implémentation de la procédure (la procédure récursive générant la suite $(x_n)_{n\geq 1}$ contient d'ailleurs dans sa définition même cette seconde boucle `if ... end if`). L'instruction `option remember` (dans une procédure récursive) permet de conserver en mémoire les résultats et donc de ne pas refaire un calcul déjà fait lors des appels successifs à la récursivité ; l'intérêt de cette instruction réside dans l'allègement qui en résulte concernant le temps d'exécution `time` du code (évalué en temps CPU).

Exercice 2.

(1) *Rappeler comment déclarer sous l'environnement* `MATLAB` *la fonction*

$$f\ : t \in \mathbb{R} \longmapsto \frac{t^2 \times (\cos(t)) \times (\ln(t^2+1))}{2 + t^4 - (\sin(t))^2},$$

de manière à cette fonction puisse aussi être directement évaluée aussi bien sur un vecteur ligne `t` *que sur un simple scalaire* t.

On utilise pour cela la commande :

```
>> f =
inline ('(t.^2).*cos(t).*(log(t.^2+1)./(2+t.^4-(sin(t)).^2)','t');
```

(2) *Rédiger un code (sous* `MATLAB`*) :*

```
function Im = trapezes (a,b,f,m) ;
```

qui, étant donnés deux réels $\mathtt{a} < \mathtt{b}$ *et un entier* $\mathtt{m} \in \mathbb{N}^*$*, renvoie en sortie, une fois exécuté, l'intégrale approchée* `Im` *de* `f` *sur le segment* $[\mathtt{a}, \mathtt{b}]$*, calculée suivant la méthode des trapèzes composite, avec comme pas* $h_{\mathtt{m}} = (\mathtt{b} - \mathtt{a})/\mathtt{M}$*, où* $\mathtt{M} = 2^{\mathtt{m}}$*. Que vaut (en vous reportant au cours) le plus grand entier* $\mathtt{k} \in \mathbb{N}^*$ *tel que l'on puisse affirmer :*

$$\Big| \int_{\mathtt{a}}^{\mathtt{b}} f(t)\, dt - \mathtt{Im} \Big| = O\Big(\Big(\frac{\mathtt{b} - \mathtt{a}}{2^{\mathtt{m}}} \Big)^{\mathtt{k}} \Big)$$

lorque `m` *croit vers l'infini ?*

Il suffit d'utiliser sur chaque segment de la subdivision de $[\mathtt{a}, \mathtt{b}]$ la formule (5.18), puis d'ajouter : les extrémités `a` et `b` sont prises en compte avec un facteur $h_{\mathtt{m}}/2$, tandis que les $2^{\mathtt{m}}$ autres nœuds de la subdivision sont pris en compte avec un facteur $h_{\mathtt{m}}$. Cela donne donc le code `MATLAB` suivant :

```
function Im = trapezes (a,b,f,m) ;
h = (b-a)/(2^m) ;
t = a:h:b ;
Im = h *((f(t(1)) + f(t(2^m+1)))/2 + sum(f(t(2:2^m))));
```

L'ordre de la méthode des trapèzes vaut $p = 3$ (voir la formule (5.23) dans la sous-section 5.2.2 du cours). Lorsque cette méthode des trapèzes est utilisée de manière composite avec un pas $h = (\mathtt{b} - \mathtt{a})/\mathtt{M}$, l'erreur absolue commise entre l'intégrale exacte $\int_{\mathtt{a}}^{\mathtt{b}} \mathtt{f}(t)\, dt$ et sa version approchée est majorée en $\mathtt{M} \times O\big(((\mathtt{b}-\mathtt{a})/\mathtt{M})^3\big) = O\big(((\mathtt{b}-\mathtt{a})/\mathtt{M})^2\big)$. Le plus grand entier `k` possible ici pour que l'assertion exigée soit valide est $\mathtt{k} = 2$.

(3) *On admet (suivant en cela la formule d'Euler-MacLaurin mentionnée en cours), que, lorsque* $m \in \mathbb{N}^*$ *croit vers l'infini,*

$$Im = \int_a^b f(t)\,dt + \alpha_0[f]\left(\frac{b-a}{2^m}\right)^2 + O\Big(\Big(\frac{b-a}{2^m}\Big)^4\Big),$$

avec $\alpha_0[f] = (f'(a) - f'(b))/12$. *Suivant le principe d'extrapolation de L. Richardson, modifier le code* `trapezes` *en un code :*

```
ImBis = trapezes2 (a,b,f,m) ;
```

de manière à ce que, cette fois, on puisse affirmer :

$$\Big|\int_a^b f(t)\,dt - ImBis\Big| = O\Big(\Big(\frac{b-a}{2^m}\Big)^4\Big)$$

lorsque m *croit vers l'infini.*

(4) L'idée est de combiner les deux formules :

$$\begin{aligned} \mathtt{Im} &= \int_{\mathtt{a}}^{\mathtt{b}} \mathtt{f}(t)\,dt + \alpha_0[\mathtt{f}]\,h_{\mathtt{m}}^2 + O(h_{\mathtt{m}}^4) \\ \mathtt{I[m+1]} &= \int_{\mathtt{a}}^{\mathtt{b}} \mathtt{f}(t)\,dt + \alpha_0[\mathtt{f}]\,h_{\mathtt{m+1}}^2 + O(h_{\mathtt{m+1}}^4) \\ &= \int_{\mathtt{a}}^{\mathtt{b}} \mathtt{f}(t)\,dt + \alpha_0[\mathtt{f}]\,\frac{h_{\mathtt{m}}^2}{4} + O(h_{\mathtt{m}}^4). \end{aligned}$$

En formant

$$\mathtt{Imbis} = \frac{1}{3}\Big(4 \times \mathtt{I[m+1]} - \mathtt{Im}\Big),$$

on obtient donc une approximation de l'intégrale exacte $\int_{\mathtt{a}}^{\mathtt{b}} \mathtt{f}(t)\,dt$ avec une erreur absolue en $O(h_{\mathtt{m}}^4)$ (au lieu de $O(h_{\mathtt{m}}^2)$ comme c'était le cas pour l'approximation par `Im`). En termes de code `MATLAB`, cela s'écrit :

```
function Imbis = trapezes2 (a,b,f,m) ;
Im = trapezes(a,b,f,m);
Imaux = trapezes(a,b,f,m+1);
Imbis = (1/3)*(4*Imaux - Im);
```

(5) *Montrer que* `ImBis` *correspond au calcul approché de l'intégrale de* f *sur le segment* `[a,b]`, *calculée suivant cette fois la méthode de Simpson composite, avec comme pas toujours* $h_m = (b-a)/M$, *où* $M = 2^m$.

On vérifie immédiatement que

$$\begin{aligned} \mathtt{Imbis} &= \frac{h_{\mathtt{m}}}{6}\Bigg(\mathtt{f(a)} + \mathtt{f(b)} + 2\sum_{j=1}^{\mathtt{M-1}} \mathtt{f}(\mathtt{a} + j h_{\mathtt{m}}) \\ &\qquad +4\sum_{j=1}^{\mathtt{M}} \mathtt{f}\Big(\mathtt{a} + (2j-1)\frac{h_{\mathtt{m}}}{2}\Big)\Bigg). \end{aligned}$$

Le membre de droite de cette formule correspond exactement à l'approximation de l'intégrale par la méthode de Simpson composite (méthode à 3 points avec précisément les pondérations $L/6, 4L/6, L/6$, L désignant ici la longueur $L = (\mathtt{b-a})/\mathtt{M}$ de chaque segment de la subdivision).

Exercice 3. *On rappelle (voir le cours d'Algèbre 2) que si* A *désigne une matrice réelle à* m *lignes et* n *colonnes et* A' *sa transposée, alors, pour tout vecteur colonne* $x \in \mathbb{R}^n$*, on a*

$$\langle A' \cdot A \cdot x, x\rangle = \|A \cdot x\|^2, \qquad (*)$$

où $\| \ \|$ *désigne la norme euclidienne sur* $\mathbb{R}^n$ *et* $\langle \ , \ \rangle$ *le produit scalaire usuel dans* $\mathbb{R}^n$*. Soit* A *une telle matrice, avec de plus* $m \geq n$ *et* $\mathrm{rang}(A) = n$*.*

(1) *En utilisant* $(*)$*, vérifier que la matrice* $M := A' \cdot A$ *est une matrice symétrique réelle définie positive (c'est-à-dire dont toutes les valeurs propres sont strictement positives).*

La relation $(*)$ implique que, si $\mathtt{M} \cdot x = 0$, alors $\|\mathtt{A} \cdot x\| = 0$. Comme le rang de A est égal au nombre de colonnes n, l'endomorphisme de matrice A relativement aux bases canoniques respectivement de $\mathbb{R}^n$ et de $\mathbb{R}^m$ est injectif. Si $\mathtt{M} \cdot x = 0$, on a donc $x = 0$. La matrice (n, n) M est donc inversible. Cette matrice est symétrique car

$$\mathtt{M'} = (\mathtt{A}' \cdot \mathtt{A})' = \mathtt{A}' \cdot \mathtt{A}'' = \mathtt{A}' \cdot \mathtt{A} = \mathtt{M}.$$

En tant que matrice symétrique réelle inversible, M est diagonalisable et toutes ses valeurs propres λ sont réelles non nulles. Si λ est une telle valeur propre et v un vecteur propre (colonne) non nul associé à cette valeur propre, alors, il vient du fait de $(*)$:

$$\langle \mathtt{M} \cdot v, v\rangle = \lambda \|v\|^2 = \|\mathtt{A} \cdot v\|^2 \geq 0.$$

On en déduit $\lambda > 0$. La matrice M est donc bien symétrique définie positive.

(2) *Soit* B *un vecteur colonne de* $\mathbb{R}^n$*. Écrire sous* MATLAB[5] *le code d'une procédure itérative (reposant sur le théorème du point fixe dans* $\mathbb{R}^n$*, au travers d'un algorithme vu en cours)*

```
function XX = resolIterative(M,B,X,k) ;
```

qui, initiée au vecteur colonne $X \in \mathbb{R}^n$*, fournit au bout de* k *itérations une approximation* XX *de la solution du système de Cramer* $M \cdot XX = B$*.*

D'après le cours (Proposition 4.5), l'algorithme de Gauß-Seidel converge lorsque la matrice M est symétrique réelle définie positive. C'est donc la procédure itérative de Gauß-Seidel que l'on peut implémenter pour résoudre le système de Cramer $\mathtt{M} \cdot \mathtt{XX} = \mathtt{B}$ de manière approchée. La syntaxe sous MATLAB de la procédure demandée ici est donc :

```
function XX = resolIterative(M,B,X,k);
T=tril(M);
F=tril(M)-M;
XX=X;
for i=1:k
    XX=T^(-1)*F*XX+T^(-1)*B;
end
```

(3) *En y intégrant une boucle* `while (...) && (...) ... end`*, modifier le code* `resolIterative` *rédigé à la question* **2** *en un code*

5. Dans cette question, comme dans la suivante, on pourra, faute d'exprimer le code sous la syntaxe du logiciel MATLAB, se contenter de le rédiger sous forme ≪ pseudo-algorithmique ≫, pourvu d'en sérier clairement les instructions.

`function [XX,Niter] = resolIterative2 (M,B,X,epsilon,k);`

de manière à ce que la procédure correspondante s'arrête automatiquement lors de son exécution dès qu'au terme d'un certain nombre d'itérations $0 \leq Niter \leq k$*, on trouve pour la première fois*

$$\left\| XX(Niter+1) - XX(Niter) \right\| \leq epsilon,$$

où `epsilon` *désigne un seuil strictement positif donné. On note ici* `XX(l)`*,* $l = 0, ..., k$*, l'état de la variable* `XX` *au terme de* `l` *itérations lors de l'exécution du code. Que signifie le fait de prendre* `epsilon=eps` *?*

Le travail demandé ici correspond à un travail fait nombre de fois en TP, soit sous `Maple`, soit (comme ici, *cf.* la feuille de TP 7) sous `MATLAB`. Voici la nouvelle routine :

```
function [XX,Niter] = resolIterative2 (M,B,X,epsilon,k);
T=tril(M);
F=tril(M)-M;
% on initialise XX et l'erreur err
% permettant le test d'erreur a venir :
XX=X;
err=2*epsilon;

% on initialise l'indice de comptage d'iterations Niter :
Niter=0;

while (err > epsilon) && (Niter<k)

  % on met en memoire le resultat de XX obtenu au terme des
  % iterations precedentes :
  XXold=XX;

  % on reactualise XX avec une nouvelle iteration :
  XX=T^(-1)*F*XX +T^(-1)*B;

  % on calcule l'erreur entre ce nouvel XX et l'ancien,
  % ce qui permet de reactualiser l'erreur :
  err=norm(XX-XXold);

  % on incremente l'indice de comptage
  Niter=Niter+1;

end
```

Prendre `epsilon=eps` revient à décider que `epsilon` correspond à l'erreur machine (2^{-52} si l'on travaille en double précision). Ceci signifie que l'exécution du code s'arrête (si toutefois on y parvient en moins de `k` itérations) dès que la machine devient incapable de distinguer du point de vue numérique les états au cran `Niter` et au cran `Niter` $+1$ de la variable `XX`.

Bibliographie

[Eiser0] M. Eisermann, l'algorithme `Pagerank` de `Google`, une promenade sur la toile, disponible en ligne sur :

`http://www.igt.uni-stuttgart.de/eiserm/enseignement/google-promenade.pdf`

[Eiser] M. Eisermann, Comment fonctionne Google ?

`http://www.igt.uni-stuttgart.de/eiserm/enseignement/google.pdf`

[MathL2] J.M. Marco, P. Thieullen, J.P. Marco (ed.), Mathématiques L2, Pearson Education, 2007.

[MathAp] A. Yger & J.A. Weil (ed.), *Mathématiques Appliquées L3*, Pearson Education, Paris, 2009.

[Peyr] G. Peyré, le traitement numérique des images, disponible sur :

`http://www.ceremade.dauphine.fr/~peyre/numerical-tour/tours/introduction_6_elementary_fr/`

[TermS] Document Ressources pour la classe de Terminale Générale et Technologique, Mathématiques série S, enseignement de spécialité :

`http://cache.media.eduscol.education.fr/file/Mathematiques`
`/20/8/LyceeGT_ressources_SpeMath_Matrices_218208.pdf`

[Vial] G. Vial, Le système proie-prédateur de Volterra-Lotka, Mars 2011, disponible en ligne sur :

`http://lama.univ-savoie.fr/~labart/generated/fichiers/MATH705/gvial_volterra.pdf`

[Y0] A. Yger, *Mathématiques de base* (MIS 101, cours 2007-2008)

`http://www.math.u-bordeaux1.fr/~yger/coursmismi.pdf`

[Yan] A. Yger, *Analyse 1* (MIS201, cours 2012-2013) :

`http://www.math.u-bordeaux1.fr/~yger/analyse1.pdf`

[Y1] A. Yger, *Calcul Symbolique et Scientifique*, polycopié de l'UE MHT 304 (2010-2011) :

`http://www.math.u-bordeaux1.fr/~yger/mht304.pdf`

[Y2] A. Yger, *Algorithmique Numérique*, polycopié de l'UE MHT 632 (2010-2011) :

`http://www.math.u-bordeaux1.fr/~yger/mht632.pdf`

[Zim] P. Zimmerman *Peut-on calculer sur ordinateur ?*, Leçon de Mathématiques d'aujourd'hui, Bordeaux, Octobre 2010 :

`http://www.loria.fr/~zimmerma/talks/lma.pdf`

Index