



PROGRAMMATION MATHÉMATIQUE

SIMULATIONS SUR SAGEMATH/PYTHON ET PROJET SUR LES RN

1ÈRE ANNÉE DU CYCLE D'INGÉNIEUR À L'ECOLE SUPÉRIEURE
DE LA STATISTIQUE ET DE L'ANALYSE DE L'INFORMATION

INES ABDELJAOUED TEJ
ANNÉE UNIVERSITAIRE 2017/2018

Attention, malgré une re-lecture soigneuse, il est possible que ce cours contienne des fautes de frappe ou des erreurs. Si vous en trouvez, merci de nous en avvertir de préférence par e-mail : inestej@gmail.com.

Avant-propos

Ce document regroupe les notes du cours enseigné en première année de l'Ecole Supérieure de la Statistique et de l'Analyse de l'Information à l'Université de Carthage. Cet enseignement se compose à la fois de cours magistraux et de séances de travaux pratiques.

Il s'agit de présenter plusieurs méthodes numériques de base utilisées dans l'approximation des fonctions par interpolation polynomiale. Nous introduisons également la résolution des équations non linéaires et celle des systèmes linéaires. Le but de cette matière est de familiariser les élèves-ingénieurs avec les techniques de simulations numériques, en abordant notamment la notion de convergence, de précision et de stabilité. L'analyse des méthodes sont complétées par un travail d'implémentation et d'application réalisé par les étudiants sur le logiciel **SAGEmath/Python**.

Table des matières

1	SAGEmath : une plateforme mathématique complète et libre	7
1.1	SAGEmath par l'exemple	7
1.1.1	Commandes de base	8
1.1.2	Exemples divers	12
1.2	Application au codage/décodage RSA	13
1.2.1	Algorithme d'Al Kindi	13
1.2.2	Algorithme de Miller-Rabin	14
1.2.3	Algorithme RSA	16
1.2.4	Correction	17
2	Interpolation et Approximation	21
2.1	Formule d'Interpolation de Lagrange	21
2.2	Algorithme d'Aitken	22
2.2.1	Formule de Newton	23
2.2.2	La méthode des différences divisées	24
2.2.3	Algorithme de Neville	25
2.3	Complexité d'algorithmes	25
2.4	Estimation de l'erreur	26
2.5	Interpolation Spline	27
2.6	Introduction à l'Approximation	29
2.6.1	Résultats fondamentaux	29
2.6.2	Approximation polynomiale discrète	29
2.7	Exercices d'Applications	31
3	Intégration numérique	33
3.1	Introduction	33
3.2	Formules de Newton-Côtes	33
3.2.1	Méthode du trapèze	34
3.2.2	Méthode de Simpson	34
3.2.3	Remarque sur la Formule Composite	35
3.3	Exemples d'Implémentations	35
4	Racine d'équation non linéaire	36
4.1	Quelques algorithmes classiques	36
4.1.1	Méthode de la bisection	36
4.1.2	Méthode de Newton	37
4.1.3	Méthode de la sécante	40

4.1.4	Méthode du point fixe	40
4.2	Méthodes d'accélération de la convergence	41
4.2.1	Procédé Δ^2 d'Aitken	41
4.2.2	Méthode de Steffensen	41
4.3	Exercices d'Applications	42
5	Résolution de systèmes d'équations linéaires	43
5.1	Rappels et compléments sur les matrices	44
5.2	Normes matricielles	44
5.2.1	Conditionnement d'une matrice	45
5.2.2	La méthode de Gauss	46
5.2.3	La factorisation LU d'une matrice	48
5.2.4	La factorisation de Cholesky	49
5.2.5	Valeurs propres et vecteurs propres	50
5.2.6	La méthode de la puissance	50
6	Equations différentielles ordinaires	54
6.1	Rappel	54
6.2	Problème de Cauchy	54
6.3	Méthode d'Euler	55
6.4	Méthode de Taylor d'ordre 2	56
6.5	Méthode de Runge-Kutta	56
7	Application aux réseaux de neurones	57
7.1	Historique	57
7.2	Définitions	58
7.2.1	Neurone formel	58
7.2.2	Fonctions d'activation	58
7.2.3	Perceptron multi-couches	59
7.3	Fonctionnement du réseau de neurones	59
7.4	Statistique et PMC	63
7.5	Logiciels utilisés	64
7.5.1	nnet de R pour le perceptron multi-couches	64
7.5.2	SageMath, Python	65
7.6	Conclusion	65
A	Travaux Pratiques sur SAGEmath	66
A.1	Interpolation	66
A.2	Résolution d'équations	69
A.3	Méthode d'Euler	70
A.4	Analyse Numérique Matricielle : SVD	70
A.4.1	A faire sur Python :	71
A.4.2	Correction	72
A.4.3	Un code plus interactif	73
A.5	Matrice bien ou mal conditionnée	74
A.6	Factorisation LU	74
A.7	Factorisation de Cholesky	74
A.8	Méthodes itératives	76

B Exerices de Révision et Applications	78
C Tests, DS et Examens corrigés	82
C.1 2008-2009	82
C.2 2009-2010	87
C.3 2010-2011	89

Introduction

Scientifiques et ingénieurs approchent les mathématiques de haut niveau de deux manières - en utilisant le Calcul Numérique ainsi que le Calcul Formel. L'analyse numérique qui fait partie du Calcul Scientifique est basée sur les ordinateurs.

L'avantage de l'analyse numérique est de pouvoir donner une réponse numérique à un problème qui n'a pas de solution analytique. Par conséquent, les résultats ne sont qu'approximatifs contrairement aux résultats du Calcul Formel. Un cours d'analyse numérique va vous apprendre à écrire les problèmes mathématiques en code, ainsi que vous apprendre quelles sortes de modèles mathématiques fonctionnent le mieux avec différentes sortes d'équations.

La démarche du numéricien commence par la modélisation du problème ou sa mise en équation. Une analyse mathématique permet ensuite d'étudier le problème, de déterminer sous quelles conditions il existe une solution unique et sinon, choisir celle qui correspond au mieux à la réalité.

Lorsque l'ingénieur ne peut pas calculer toutes les valeurs, le mieux est d'approcher le problème en calculant des valeurs de certains points. C'est la discrétisation, qui doit être accompagnée d'une mesure de l'erreur commise, c'est-à-dire mesurer la distance entre la solution approchée et la véritable solution du problème. Une étape de simplification, comme la linéarisation pour des problèmes non linéaires, permet d'obtenir des problèmes pour lesquels on dispose de méthodes de résolution numérique performantes : résolution de systèmes linéaires, optimisation par une méthode de gradient, recherche de valeurs propres.

Comme suite aux résultats du calcul par ordinateur, il convient d'être circonspect et d'étudier les sources d'erreurs : les données elles-mêmes, l'arrondi, la méthode utilisée. La démarche qui vient d'être présentée est celle que suit le numéricien en charge de la simulation numérique d'un problème.

Le cours est structuré en trois grands chapitres :

- Interpolation et Approximation
- Racine d'équations non linéaires
- Résolution de systèmes d'équations linéaires.

Etant donné la valeur connue d'une certaine fonction en un certain nombre de points, quelle valeur prend cette fonction en un autre point quelconque situé entre deux points donnés ? Une méthode très simple est d'utiliser l'interpolation linéaire, qui suppose que la fonction inconnue évolue linéairement entre chaque paire de points successifs connus. Nous introduisons dans le

premier chapitre l'interpolation polynomiale ainsi que d'autres méthodes d'interpolation utilisant des fonctions localisées telles que les splines.

Un autre problème fondamental est le calcul des solutions d'une équation donnée. Les algorithmes de recherche de racines d'une fonction sont utilisés pour résoudre les équations non linéaires. Si la fonction est différentiable et que sa dérivée est connue, alors nous choisirons la méthode de Newton sinon, d'autres méthodes seront détaillées dans le deuxième chapitre.

Le troisième chapitre porte sur l'étude des méthodes de calcul du vecteur x solution du système d'équations linéaires $Ax = b$. La première partie de ce chapitre sera consacrée aux méthodes directes et la deuxième aux méthodes itératives. Nous aborderons ensuite en troisième partie les méthodes de résolution de problèmes aux valeurs propres.

Le champ de l'analyse numérique est divisé en différentes disciplines suivant le type de problème à résoudre, et chaque discipline étudie diverses méthodes de résolution des problèmes correspondants. Il s'agit dans un premier temps d'étudier les principales méthodes en Analyse Numérique (Méthode de Lagrange, algorithmes du point fixe, de Newton, du gradient conjugué, la factorisation LU, etc...). Nous nous intéressons ensuite aux principes qui sous-tendent ces méthodes ainsi qu'à leur mise en oeuvre pratique en langage SCILAB et MAXIMA. Il s'agira enfin d'implémenter et de tester plusieurs algorithmes d'Analyse Numérique et plusieurs approches de résolution.

Nous donnons à la fin de chaque chapitre une liste d'exercices de difficultés variables. Ces exercices ne sont profitables que si l'élève-ingénieur les travaille. Qu'il garde à l'esprit ce proverbe chinois :

*J'entends et j'oublie, (cours oral)
je vois et je retiens, (étude du cours)
je fais et je comprends (exercices).*

Pour conclure, ce cours est une introduction à des approches plus complexes des mathématiques de l'ingénieur : d'autres thématiques seront développées dans les cours d'Optimisation Convexe, de Calcul de Variations, de Recherche Opérationnelle, etc...

Enfin, rappelez-vous la devise suivante :

That which is learned without desire is soon forgotten.

Chapitre 1

SAGEmath : une plateforme mathématique complète et libre

Python est un langage de programmation adapté au calcul scientifique. Il est interprété, interactif et orienté objet. Il possède de nombreuses extensions pour le calcul, le traçage, le stockage de données. Combiné avec des fonctionnalités comme Tkinter, une interface graphique pour les utilisateurs, Python permet de développer de bonnes interfaces graphiques pour les codes. Par exemple, les principaux mots clés sous Python sont :

```
and else import assert except in      break
exec is class finally lambda  continue for
not def from or del global pass elif if print
raise return try while yield
```

L'aspect le plus intéressant est qu'il existe un ensemble de modules pour le calcul scientifique sous Python. Des modules pour pratiquement tout (vecteurs, tenseurs, transformations, dérivées, algèbre linéaire, transformées de Fourier, statistiques, groupes, etc) sont disponibles. Il est aussi possible d'utiliser des bibliothèques C et Fortran à partir de Python. Finalement, écrire un programme numérique est chose aisée en Python. Il est également possible de réaliser des algorithmes et des calculs parallèles : des interfaces existent vers netCDF (fichiers binaires portables), MPI et BSPLib.

Comme exemple, existe Scipy, la bibliothèque d'outils scientifiques libres pour Python. Elle inclut des modules pour créer des graphiques et faire du traçage, des modules d'optimisation, d'intégration, de fonctions spéciales, de manipulation de signaux et d'images, d'algorithmes génétiques, de solveurs d'équations différentielles ordinaires, etc.

Le responsable, K. Hinsén, propose un bon didacticiel intitulé Scientific Computing in Python (Calcul scientifique avec Python). Consulter aussi I. Lima dans [?].

1.1 SAGEmath par l'exemple

La plateforme SAGEmath est un ensemble de logiciels libres et payants (sous réserve d'avoir la licence) qui cohabitent ensemble grâce au langage Python.

1.1.1 Commandes de base

Afin de mieux illustrer les utilisations de base de **SAGEmath**, nous nous sommes largement inspirés de [?]. Mais avant tout, télécharger sage-4.4.1.tar depuis sagemath.org (280MB) et taper les commandes suivantes :

```
$ tar xf sage-4.3.1.tar
$ cd sage-4.3.1
$ make
```

Aller prendre un café (ou plusieurs). . .

1. Les opérations arithmétiques sont données par :

```
2+3
15 + 3 * 2 / 8 - 2^3
type(21)
--2
30/3
n(30/3, 4)
2^3
8%5
boite=7
```

7

2. Les chaînes de caractères :

```
a="Bonjour, je suis une chaine de caractere"
print(a)
type(a)
<type 'str'>.
```

3. Opérateurs conditionnels :

```
x = 2
y = 3
print x == y
print x <> y
print x, "!=" , y, ":", x != y
print x < y
print x > y
print x >= y
```

False.

4. Utiliser ce code avec la variable **s** et obtenir la solution :

```
var('a,b,c')
eqn = [a+b*c==1, b-a*c==0, a+b==5]
s = solve(eqn, a,b,c)
```

La solution de $\text{eqn} = [bc + a = 1, -ac + b = 0, a + b = 5]$ est :

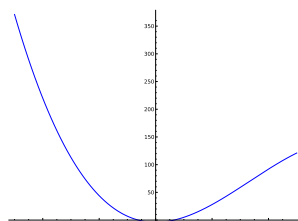
$$\left[a = \frac{25i\sqrt{79} + 25}{6i\sqrt{79} - 34}, b = \frac{5i\sqrt{79} + 5}{i\sqrt{79} + 11}, c = \frac{1}{10}i\sqrt{79} + \frac{1}{10} \right]$$

$$\left[a = \frac{25i\sqrt{79} - 25}{6i\sqrt{79} + 34}, b = \frac{5i\sqrt{79} - 5}{i\sqrt{79} - 11}, c = -\frac{1}{10}i\sqrt{79} + \frac{1}{10} \right]$$

5. `var('x')`
`f=-x^3+3*x^2+7*x^2-4`

On construit l'objet courbe, stocké dans la variable p :

`p = plot(f, x, -5, 5)`



6. La commande if :

```
x = 3
print x > 5
if x > 5:
    print x
    print "plus grand"
    print "Fin du programme"
else:
    print "Introduire une autre valeur"
```

7. L'opérateur booléen and :

```
a = 7
b = 9
print a > 5 and b > 10
if a > 5 and b < 10:
    print "Les deux expressions sont vraies"
```

8. L'opérateur booléen or :

```
print a > 5 or b > 10
if a > 5 or b > 10:
    print "Au moins une des deux expressions est vraie"
```

9. L'opérateur booléen not :

```
a = 7
print a > 5
print not a > 5
```

10. La boucle while :

```
x = 1                #initialisation
while x <= 40:
    print x,
    x = x + 2        #incrementation de x par 2
```

11. Les ensembles :

```

x = Set([2,3,51,21])
print type(x)
print x[0]
print x
y = Set([51])
x.difference(y)
a = Set([1,3,2,5,5])
a.cardinality()
3 in a
a.union(x)
a.intersection(x)

```

12. Les listes :

```

y = [1,1/2,0.75,'Bonjour',x]
type(y)
y[3]
y[4]
add([1,2,3,4])

```

13. Les n-uplets :

```

z = (1,3,5,4)
type(z)
z[0]
a,b,c = (1,2,3)
a; b; c

```

14. Les opérateurs in et not in :

```

54 in [2,54,2]
30 not in [2,54,2]

```

15. La boucle for :

```

for i in [1,2,3,4,5]:
    i,

```

16. Les fonctions :

```

def salut(s):
    """
    Retourne Bonjour ou Au revoir.
    """
    if s>3:
        return "Bonjour"
    else:
        return "Au revoir"

html("<H1>Exemple de fonctions interactive</H1>")
html("<I> Juste pour voir</I>")
@interact
def _(p=1):
    salut(p)

```

17. Quelle est l'utilité des commandes suivantes :

```
denominator(12/23)
divisors(20)
gcd(40,116)
mul([2,3,4])
len([1,2,3,'hello'])
srange(11)
a=srange(4,step=1.3)
b=srange(-5,-1)
zip(a,b)
```

```
[(0.0000000000000000,-5),(1.3000000000000000,-4),(2.6000000000000000,-3),(3.9000000000000000,-2)].
```

18. La boucle for :

```
for t in srange(6):
    t,
```

```
d=zip(a,b)
for (x,y) in d:
    x,y
```

```
[2*t for t in [0,1,2,3]]
```

```
[t^2 for t in range(20) if t%2==0]
```

19. Obtenir des résultats numériques :

```
a = pi
a
n(a)
a.n(digits=30)
a.n(prec=12)
```

20. Calcul symbolique :

```
var('a')
type(2*a)
m = 5 + a
n = 8 + a
y = m * n
z = y.expand()
y(5)
```

130.

21. Equations symboliques : $[a = (-5)]$.

22. Divers :

```
A=matrix(3,3,[1,2,3,2,3,4,1,2,3])
latex(A)
```

23. Courbes en 2 ou 3 dimensions :

24. Calcul de $0 + 1 + \dots + N - 1$:

```
def sum1(N):
    s = 0
    for k in range(N):
        s += k
    return s
SAGEMath: time sum1(10^6)
```

Cette fonction peut être sauvegardée dans un fichier (sum2.pyx par exemple). Les commandes suivantes permettent de lire ou d'exécuter le fichier :

```
SAGEMath: cat sum2.pyx
SAGEMath: load sum2.pyx
SAGEMath: time sum2(10^6)
```

1.1.2 Exemples divers

1. On considère les vecteurs et les matrices suivants

$$l = (1 \ 2 \ 3 \ 4 \ 5), v = \begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{pmatrix}, A = \begin{pmatrix} 1 & 2 & 0 & 0 & 0 \\ 0 & 0 & 3 & 2 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 2 \\ 1 & 1 & 1 & 0 & 0 \end{pmatrix}, B = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 2 & 3 & 4 & 5 & 6 \\ 3 & 4 & 5 & 6 & 7 \\ 4 & 5 & 6 & 7 & 8 \\ 5 & 6 & 7 & 8 & 9 \end{pmatrix}$$

Créer et commenter le résultat des instructions $l * v$, $A + B$, $A * B$, B/A , ${}^t v$.

2. On appelle *méthode de Monte-Carlo* toute méthode visant à calculer une approximation numérique par utilisation de procédés aléatoires. Nous allons calculer une valeur approchée de π par la méthode de Monte-Carlo¹.

On tire au hasard des points de coordonnées (x, y) dans un carré de longueur égale à 2. On vérifie s'ils appartiennent ou non à un disque de rayon 1 et de centre, le centre du carré. Notons que ces points peuvent être tirés avec la même probabilité dans l'ensemble du carré. D'après la loi des grands nombre, lorsque le nombre de tirage tend vers l'infini, le rapport entre le nombre de points tirés dans le disque et le nombre de points tirés au total tend vers le rapport des surfaces du cercle et du carré, soit $\frac{\pi}{4}$.

3. Etant donné $n > 0$, écrire en utilisant le moins de lignes possible, un script **SAGEMath** qui déclare la matrice de Cauchy de coefficients $a_{i,j} = \frac{1}{i+j}$.

4. On peut toujours transformer un problème du type $f(x) = 0$ en un problème de la forme $g(x) = x$ et ce d'une infinité de façons. Le théorème suivant caractérise les fonctions g qui donnent des suites convergentes : Si dans l'intervalle $[a, b]$, g vérifie les conditions suivantes :

(a) $x \in [a, b]$ alors $g(x) \in [a, b]$,

(b) l'application g est strictement contractante, c'est-à-dire $\max_{x \in [a, b]} |g'(x)| < 1$.

Alors pour tout $x_0 \in [a, b]$, la suite définie par $x_{n+1} = g(x_n)$ converge vers l'unique point fixe x^* de g sur $[a, b]$.

1. N. Metropolis et S. Ulam ont inventé en 1947 ce type de méthode. Le nom fait allusion aux jeux de hasard pratiqués dans les casinos de Monte-Carlo

Ecrire une fonction prenant en entrée g , x_0 , ϵ et un nombre d'itérations maximal N et qui rend une approximation du point fixe de g à ϵ près.

1.2 Application au codage/décodage RSA

Ce TP est utilisé pour une initialisation à Python. C'est un bon exemple qui permet la manipulation des grands nombres pseudo-aléatoires, il permet de voir la puissance d'un logiciel comme Python sur des problèmes de décryptage de données. On utilise principalement les résultats d'arithmétique (petit théorème de Fermat) pour répondre à la problématique de chiffrement. Nous encourageons les étudiants à suivre les étapes de ce TP pour se familiariser avec Python. Le système de chiffrement RSA est une méthode qui repose sur certains concepts de la théorie des nombres. C'est une méthode à clé publique, nommée à partir des noms de famille de ses inventeurs Ronald Rivest, Adi Shamir et Leonard Adleman. Ce TP a pour but d'étudier le principe de fonctionnement de l'algorithme RSA. Pour cela, nous allons d'abord étudier un code permettant de décrypter et de crypter un message. A la suite, nous allons voir comment générer des clés publiques et privées pour crypter un message.

Mais avant cela, la première section présentera un petit exemple ludique de décryptage d'un message, permettant d'introduire un certain nombre de vocabulaire lié à la cryptographie. Cet exemple comprend une brève explication du cryptage de Jules César ainsi que de l'algorithme de décryptage d'Al Kindi.

1.2.1 Algorithme d'Al Kindi

Pour décoder un message texte, on va utiliser l'algorithme de Abu Yusuf Yaqub Ibn Ishaq Al Kindi qui a vécu à Bagdad de 801 à 873. Cet algorithme consiste à comparer la fréquence d'apparition des lettres codées avec la fréquence des lettres dans la langue française. En effet, une lettre qui apparaît souvent dans la langue française, va apparaître souvent (via son code) dans un message codé. Ci-dessous les fréquences des lettres dans la langue française :

Lettres	E	S	A	I	T	N	R	U	L	O	D	C	P
Fréquences (%)	14.7	7.9	7.6	7.5	7.2	7.1	6.6	6.3	5.5	5.4	3.7	3.3	3

Soit le message codé suivant² :

```
code = "YR PNEER QR Y ULCBGRAHFR "
code += "RFG RTNY FV WR AR Z NOHFR "
code += "N YN FBZZR QRF PNEERF "
code += "QRF QRHK NHGERF PBGRF"
```

On crée l'objet `message` composé des lettres de `code` et l'ensemble `lettres` des différentes lettres apparaissant dans le message codé :

```
message = [i for i in code]
lettres = set(message)
```

1. Si une lettre apparaît souvent dans un message, on s'attend à ce qu'elle apparaisse autant dans un message codé. On calcule donc le tableau d'effectifs en comptant les lettres du message :

2. TP inspiré des supports du Laboratoire d'informatique et de mathématiques de l'Institut de recherche sur l'enseignement des mathématiques de la Réunion : <http://irem.univ-reunion.fr/IMG/pdf/TP11.pdf>

```
for j in lettres:
    print(j+'-->'+str(message.count(j)))
```

Comparer ce résultat avec le tableau ci-dessus, et donnez le décodage de la lettre E, S et A.

2. Pour continuer le décodage, et vu que les fréquences d'apparition des lettres sont trop proches, on peut utiliser les propriétés linguistiques, comme le fait qu'un mot long commence souvent par une consonne ou qu'il y a peu de mots de deux lettres. On peut également tester si le procédé de chiffrement utilisé est bien celui proposé par Jules César.

Ce procédé date de 120 avant J.C., il utilise la technique suivante : il suffit de procéder à une permutation circulaire de toutes les lettres de l'alphabet, en remplaçant chaque lettre par celle qui est située à s rangs plus loin (ici s correspond à la clé de chiffrement). Dans cet exemple, la lettre A correspond à N, la lettre E correspond à R et la lettre F correspond à S. Vérifiez, grâce au code suivant, que la clé de chiffrement est bien $s = 13$ et en déduire le message déchiffré.

```
s = 13 # s est la cle de chiffrement selon la méthode de Jules César
n = len(message)
MessageDechiffre = ""
for i in range(n):
    caractere = message[i]
    p = ord(caractere) #Varie entre 65 et 91
    q = p + s
    if q > 91:
        q = q - 26
    if q == 91:
        q = 65
    r = chr(q)
    print caractere, p, r, q
    MessageDechiffre = MessageDechiffre + r
print MessageDechiffre
```

3. Pourquoi est-ce que le codage de César est facile à casser ? Indication : trouver le nombre de clés différentes qu'on peut utiliser pour le codage de César.

1.2.2 Algorithme de Miller-Rabin

Le but de cette section est de construire des nombres premiers, le plus grand possible. Pour cela, on va utiliser l'algorithme de Miller-Rabin qui permet de tester si un nombre donné est premier ou pas, en un temps raisonnable.

L'algorithme se base sur le théorème suivant : **Théorème.** Soit p un nombre premier. On pose s la puissance maximale de 2 divisant $p - 1$:

$$p - 1 = 2^s \cdot d \text{ avec } d \text{ pair.}$$

Pour tout $1 \leq n \leq p-1$, on a l'une des possibilités suivantes :

$$n^d = 1 \bmod p \quad \text{ou alors} \quad n^{2^j d} = -1 \bmod p \quad \text{pour un certain } 0 \leq j < s.$$

Il s'agit donc de choisir au hasard un nombre n jusqu'à ce que l'une des deux conditions précédente n'est pas satisfaite. Cet n est appelé "témoin" du fait que p est un nombre composé. Si p n'est pas premier, alors il existe $3/4$ de chance que n soit "témoin". L'idée est donc d'utiliser le petit théorème de Fermat i.e. le fait que $a^{p-1} = 1 \bmod p$.

1. On désigne par t la taille en bits de la clé RSA que l'on souhaite utiliser. Il nous faut donc trouver deux nombres premiers de taille $t/2$. Pour cela, on utilise la fonction `random` pour trouver un nombre de $t/2$ bits au hasard puis `netx_prime` le nombre premier suivant.
2. Déterminez le plus petit nombre premier de 10 chiffres (pensez à la conversion en écriture binaire et à la commande `random_prime(2^1023, 2^1024)`).

```
import random
randint(10^9, 10^10)
random.getrandbits(2048)
```

3. Implémentez l'algorithme de Miller-Rabin. Le pseudo-code obtenu de Wikipédia est donné ci-dessous. La fonction `Témoin_de_Miller(a,n)` renvoie Vrai si a est un témoin de Miller que n est composé, Faux si n est fortement pseudo-premier en base a . Elle prend en entrée un nombre n entier, impair ≥ 3 et a un entier > 1 . On calcule s et d tels que $n-1 = 2^s d$ avec d impair. Ici $s > 0$ car n est impair. On note x le reste de la division de a^d par n , puis dans la boucle Tant que, le reste de la division de x^2 par n .

```
Témoin_de_Miller(a, n):
    x := a^d % n #x entier reste de la division de ad par n
    si x = 1 ou x = n - 1
        renvoyer(Faux)
    Tant que s > 1
        x := x^2 % n
        si x = n - 1
            renvoyer(Faux)
        s := s - 1
    Fin de boucle tant que
    renvoyer(Vrai)
```

Le test de Miller-Rabin peut alors être décrit comme suit, `Miller-Rabin(n, k)` renvoie Vrai si n est fortement pseudo-premier en base a pour k entiers a , Faux s'il est composé. C'est un algorithme qui a une complexité polynomiale en terme de temps de calcul : $O(k \cdot \log(n)^3)$ obtenue en remarquant que la décomposition $n-1 = 2^s d$ se calcule en $O(\log(n))$.

```
Miller-Rabin(n,k):
    répéter k fois :
        choisir a aléatoirement dans l'intervalle [2, n-2]
        si Témoin_de_Miller(a,n)
            renvoyer(Faux)
    Fin de boucle répéter
    renvoyer(Vrai)
```


4. La fonction `is_prime` est un test de primalité qui prouve si un nombre donné est premier ou pas. La fonction `is_pseudoprime` est un test probabiliste de primalité : l'option `> 0` utilise le test de primalité probabiliste de Miller-Rabin. Ce test s'appuie sur le petit théorème de Fermat. Testez ces deux fonctions et expliquez à quoi servent les fonctions `netx_prime` et `random_prime` ou encore `next_probable_prime`³.

1.2.3 Algorithme RSA

Le principe du système de chiffrement RSA est le suivant⁴ :

Trouver deux nombres premiers p et q	p et q doivent rester secrets
Calculer $n = pq$	n est clé publique
Choisir e tel que son PGCD avec $(p-1)(q-1)$ soit 1	e est clé publique
Trouver d tel que $ed \equiv 1 \pmod{(p-1)(q-1)}$	d est la clé privée.

Le cryptage fonctionne de la manière suivante : il y a deux personnes Ali et Besma qui souhaitent communiquer en toute sécurité. Soit M un nombre qui correspond au message à envoyer. B génère n ainsi que les exposants de déchiffrements e et d . B envoie à A les nombres n et e qui constituent la clé publique⁵.

Ici Ali est l'émetteur et il ne connaît que la clé publique, c'est-à-dire les nombres e et n . Il calcule $M^e \equiv N \pmod{n}$ et envoie au destinataire le message N . Le récepteur qui est Besma calcule $N^d \equiv W \pmod{n}$. On constate que W est exactement le message M .

Exercice à faire : Les messages qu'on peut chiffrer sont les entiers M compris entre 0 et $n-1$. Le codage du message M s'obtient par le calcul de N et le déchiffrement par le calcul de W .

1. Calculez $N = M^e \pmod{n}$. Pour cela utilisez la commande `help(mod)` ou encore la commande `%`.
2. Le décodage ou le déchiffrement d'un message codé s'obtient par le calcul de $W = N^d \pmod{n}$. Donnez deux fonctions de cryptage et de décryptage, par clés RSA : on appellera les fonctions `N = encrypt(e, n, M)` et `W = decrypt(d, n, N)`.
3. Les attaques actuelles du RSA se font essentiellement en factorisant l'entier n de la clé publique. La sécurité du RSA repose donc sur la difficulté à factoriser de grands entiers. Pour garantir une certaine fiabilité des transactions, il faut choisir des clés plus grandes : des clés de 2^{1024} voire 2^{2048} bits, i.e. des clés à $1024 \cdot \ln(2)/\ln(10)$ ou $2048 \cdot \ln(2)/\ln(10)$ chiffres.
La commande `factor` factorise les entiers qu'on lui passe en paramètre. Générez des clés RSA de plus en plus grande et constatez l'augmentation du temps de calcul de la factorisation de ces données.

Le RSA est une méthode de chiffrement assez sûre, mais il n'est pas exclu que des services secrets comme la NSA ou le FSB aient réussi le décryptage du RSA.

3. [http : //sametmax.com/les - nombres - en - python/](http://sametmax.com/les-nombres-en-python/), voir aussi [http : //www0.cs.ucl.ac.uk/staff/d.jones/GoodPracticeRNG.pdf](http://www0.cs.ucl.ac.uk/staff/d.jones/GoodPracticeRNG.pdf) pour une bonne utilisation des outils de génération de nombre pseudo-aléatoires

4. Le nombre e est appelé la graine et on pose $\phi = (p-1)(q-1)$.

5. Le code python pour l'algorithme RSA a été inspiré de [https : //gist.github.com/JonCooperWorks/5314103](https://gist.github.com/JonCooperWorks/5314103) alors que la structure du TP a suivi les travaux pratiques du master Informatique de l'Université de Lille 1 : [https : //www.irisa.fr/prive/sgambs/TP7_intro_securite.pdf](https://www.irisa.fr/prive/sgambs/TP7_intro_securite.pdf)

1.2.4 Correction

Algorithme d'Al Kindi

Soit le message chiffré suivant :

YR PNEER QR Y ULCBGRAPHFR RFG RTNY FV WR AR Z NOHFR N YN FBZZR QRF PNEERF QRF
QRHK NHGERF PBGRF

Il y a 20 lettres distinctes : qui apparaissent avec une fréquence détaillée ci-dessous :

Lettres	A	C	B	E	G	F	H	K	L	O	N	Q	P	R	U	T	W	V	Y	Z
Fréquences (%)	2	1	3	5	4	9	4	1	1	1	7	4	3	17	1	1	1	1	4	3

Les trois premières fréquences indiquent les lettres E, S et N. Ainsi, la lettre R correspond très probablement à la lettre E, la lettre F correspond à la lettre S et la lettre N correspond à la lettre A. En tenant compte de la clé de chiffrement $s = 13$, on peut en déduire le message déchiffré :

LE CARRE DE L HYPOTENUSE EST EGAL SI JE NE M ABUSE A LA SOMME DES CARRES DES
DEUX AUTRES COTES

.

Les procédures sur Python sont les suivantes. On peut s'amuser à coder et à décoder, grâce à la méthode de déchiffrement de Cesar, n'importe quel texte contenant les lettres majuscules de l'alphabet.

```
def codage_cesar(cle, message):
    #message est sous format string
    n = len(message)
    MessageChiffre = ""
    for i in range(n):
        caractere = message[i]
        p = ord(caractere) #Varie entre 65 et 91
        q = p + s
        if q > 91:
            q = q - 26
        if q == 91:
            q = 65
        r = chr(q)
        #print caractere, p, r, q
        MessageChiffre=MessageChiffre+r
    return(MessageChiffre)
```

```
message = "LE CARRE DE L HYPOTENUSE"
codage_cesar(cle, message)
```

```
def decodage_cesar(cle,code):
```

```

n = len(code)
MessageDechiffre = ""
for i in range(n):
    caractere = code[i]
    p = ord(caractere) #Varie entre 65 et 91
    q = p + s
    if q > 91:
        q = q - 26
    if q == 91:
        q = 65
    r = chr(q)
    #print caractere, p, r, q
    MessageDechiffre=MessageDechiffre+r
return(MessageDechiffre)

```

```

decodage_cesar(cle,code)

```

Algorithme de Miller-Rabin

```

sage: import random
sage: def decompose(n):
...     s = 0
...     d = n
...     while d % 2 == 0:
...         d = d/2
...         s += 1
...     return s, d
sage: decompose(33)
(0, 33)
sage: def Temoin_de_Miller(a,n):
...     s,d = decompose(n-1)
...     #print n-1, 2^s*d
...     x = a^d % n
...     if x == 1 or x == n - 1:
...         return(False)
...     while s > 1:
...         x = x^2 % n
...         if x == n - 1:
...             return(False)
...         s = s - 1
...     return(True)
sage: def Miller_Rabin(n,k):
...     i = 0
...     while i<k:
...         a = randint(2,n-2)
...         if Temoin_de_Miller(a,n)==True:
...             return(False)

```

```

...         i += 1
...         return(True)
sage: n = 122201
sage: a = 10
sage: k = 10
sage: print Temoin_de_Miller(a, n), Miller_Rabin(n, k)
False True
sage: next_prime(122187)
122201

```

Algorithme RSA

```

sage: import random
sage: """
sage: Ce que fait Besma :
...     D'abord B génère n ainsi que e et d.
...     Ensuite B envoie les clés publiques n et e à Ali.
...
sage: """
sage: p = 100
sage: p = next_prime(p)
sage: q = next_prime(p)
sage: print p,q
sage: n = p*q
sage: #la graine est choisie par hasard ;
sage: #il faut vérifier que e et phi sont premiers entre-eux :
sage: phi = (p-1)*(q-1)
sage: e = random.randrange(1,phi)
sage: print e, n, "les exposants de déchiffrements"
101 103
2522 10403 les exposants de déchiffrements
sage: phi = (p-1)*(q-1)
sage: #On cherche d tel que d*e+a*phi=1
sage: #print phi, e, n, d
sage: """
sage: Ali reçoit les clés de déchiffrement n et e.
sage: Pour envoyer le message M codé, A calcule N :
sage: """
sage: # Soit un message M :
sage: M = randint(2,n-1)
sage: print M
sage: def encrypt(e, n, M):
...     return pow(M,e,n)
sage: N = encrypt(e, n, M)
sage: print N
sage: """
sage: Ce que fait B : En recevant N de Ali, Besma calcule W :

```

```
...      B calcule d qui est la clé privée
...      Ensuite, elle détermine W qui est le M déchiffré.
sage: """
sage: def decrypt(d, n, N):
...     return(pow(N, d, n))
sage: g, a, d = xgcd(phi, e) #phi*a+d*e=g
sage: while g<>1 and d<=0:
...     g, a, d = xgcd(phi, e)
...     e = random.randrange(1,phi)
...
...
sage: W = decrypt(d, n, N)
sage: print p, q, n, e, d, M, N, W, M==W
101 103 10403 9019 19 3805 9308 4727 False
sage: """
sage: Pour casser le codage, il suffit de factoriser n et de récupérer p et q
sage: """
sage: factor(n)
101 * 103
```

Chapitre 2

Interpolation et Approximation

Soient $n + 1$ couples : $(x_0, f_0), (x_1, f_1), \dots, (x_n, f_n) \in \mathbf{R} \times \mathbf{R}$ tels que les x_i soient tous distincts. On cherche un polynôme $P : \mathbf{R} \longrightarrow \mathbf{R}$ tel que $P(x_i) = f_i$ pour $i = 0..n$.

Nous introduisons dans un premier temps la *formule d'interpolation de Lagrange* qui prouve l'existence et l'unicité du polynôme P de degré au plus n . Ensuite nous présenterons l'*algorithme d'Aitken* et la *méthode des différences divisées* qui utilise la *formule d'interpolation de Newton*. Puis, nous introduirons l'*algorithme de Neville* qui présente une méthode très pratique pour le calcul sur machine du polynôme P en un point. Nous introduirons enfin l'*interpolation Spline* qui est un outil fondamental du graphisme sur ordinateur.

2.1 Formule d'Interpolation de Lagrange

Généralement les réels f_i représentent le résultat d'expériences et sont l'expression d'une fonction f continue dans \mathbf{R} en les points x_i . La formule d'interpolation de Lagrange est un moyen qui permet d'approcher la fonction f .

Théorème 2.1.1. Soient (x_i, f_i) pour $i = 0..n$ des points réels ou complexes, avec $x_i \neq x_j$ pour $i \neq j$. Il existe un unique polynôme P de degré inférieur ou égal à n vérifiant $P(x_i) = f_i$ pour $i = 0..n$.

La preuve du théorème 2.1.1 se base sur les *polynômes d'interpolation de Lagrange*

$$L_i(x) = \prod_{j=0, j \neq i}^{j=n} \frac{x - x_j}{x_i - x_j}$$

qui vérifient $L_i(x_i) = 1$ et pour $j \neq i : L_i(x_j) = 0$. Le polynôme P est alors égal à $P(x) = \sum_{i=0}^{i=n} f_i L_i(x)$ et il est appelé *polynôme ou formule de Lagrange*.

Démonstration : Soient P et Q deux polynômes de degré au plus égal à n et vérifiant $P(x_i) = Q(x_i) = f_i$ pour $i = 0..n$. Alors le polynôme $r = P - Q$ (qui est de degré au plus égal à n) s'annule en x_0, x_1, \dots, x_n . C'est donc un polynôme qui a $n + 1$ racines : il est nécessairement nul. Donc $P = Q$, d'où l'unicité.

Pour montrer l'existence d'un tel polynôme, il suffit de remarquer que la famille (L_0, L_1, \dots, L_n) forme une base de l'espace vectoriel des polynômes de degré inférieur ou égal à n (qui est de dimension $n+1$). En effet, pour tout x , $\sum_{i=0}^n \alpha_i L_i(x) = 0$ implique en particulier que $\alpha_k = 0$ pour $k = 0..n$ (on remplace x par x_k). On retrouve la formule de Lagrange en écrivant P dans cette base : $P = \sum_{i=0}^n f_i L_i(x)$ et $P(x_i) = f_i$ pour tout $i = 0..n$.

Exemple 2.1.1. Le polynôme P de degré inférieur ou égal à 3 vérifiant

$$P(0) = 1, \quad P(1) = 2, \quad P(-1) = -1 \quad \text{et} \quad P(2) = 4$$

est $P(x) = 1.L_0(x) + 2.L_1(x) + (-1).L_{-1}(x) + 4.L_2(x) = \frac{1}{3}x^3 - \frac{1}{2}x^2 + \frac{7}{6}x + 1$ où :

$$\begin{aligned} L_0(x) &= \frac{(x-1)(x+1)(x-2)}{(0-1)(0+1)(0-2)}, & L_1(x) &= \frac{(x-0)(x+1)(x-2)}{(1-0)(1+1)(1-2)}, \\ L_{-1}(x) &= \frac{(x-0)(x-1)(x-2)}{(-1-0)(-1-1)(-1-2)}, & L_2(x) &= \frac{(x-0)(x-1)(x+1)}{(2-0)(2-1)(2+1)}. \end{aligned}$$

Exercice 2.1.1. Construire les polynômes d'interpolation de Lagrange de $f(x) = e^{-x^2}$ sur les entiers de $[-2, 2]$.

Exercice 2.1.2. Ecrire un algorithme qui calcule $L_i(x)$ pour n, i et x fixés.

Remarque 2.1.1. Si on utilise la formule de Lagrange, le nombre d'opérations nécessaires pour calculer $P(x)$ est de l'ordre de $4n^2$ opérations pour chaque valeur de x .

Sur ordinateur, on préfère utiliser la *formule barycentrique* :

$$P(x) = \frac{\sum_{k=0}^n \frac{a_k f_k}{x - x_k}}{\sum_{k=0}^n \frac{a_k}{x - x_k}}$$

avec $a_k = \frac{1}{v'(x_k)}$ et $v(x) = (x - x_0)(x - x_1) \dots (x - x_n)$. Cette formule nécessite pour une première valeur de x l'ordre de $2n^2$ opérations afin de déterminer $P(x)$. Pour d'autres valeurs de x , $2n$ opérations sont nécessaires au calcul de P à condition de conserver les valeurs a_k .

2.2 Algorithme d'Aitken

Soit la relation de récurrence suivante :

$$\begin{cases} F_{k,0}(x) &= f_k, & k = 0..n \\ F_{k,j+1}(x) &= \frac{F_{j,j}(x)(x_k - x) - F_{k,j}(x)(x_j - x)}{x_k - x_j}, & k = j + 1..n \end{cases}$$

Théorème 2.2.1. Le polynôme de Lagrange $P(x)$ sur les points x_0, x_1, \dots, x_n est égal à $F_{n,n}(x)$.

La preuve du théorème 2.2.1 se base sur le fait que $F_{k,j}(x)$ avec $k \geq j$ est le polynôme de Lagrange sur les points $x_0, x_1, \dots, x_{j-1}, x_k$.

Algorithme. On calcule $P(x) = F_{n,n}(x)$ à l'aide du tableau triangulaire suivant :

k/j		0	1	2	\dots	$j-1$	j	\dots	$n-1$	$x_k - x$
0	$F_{0,0}$									$x_0 - x$
1	$F_{1,0}$	$F_{1,1}$								$x_1 - x$
2	$F_{2,0}$	$F_{2,1}$	$F_{2,2}$							$x_2 - x$
3	$F_{3,0}$	$F_{3,1}$	$F_{3,2}$	$F_{3,3}$						$x_3 - x$
\vdots					\dots					\vdots
k	$F_{k,0}$	$F_{k,1}$	$F_{k,2}$	$F_{k,3}$		$F_{k,j}$	$F_{k,j+1}$			$x_k - x$
\vdots										\vdots
n	$F_{n,0}$	$F_{n,1}$	$F_{n,2}$						$F_{n,n}$	$x_n - x$

Par exemple, pour calculer $F_{k,2}(x)$, on considère le produit croisé suivant : $F_{1,1}(x)(x_k - x) - F_{k,1}(x)(x_1 - x)$. Nous obtenons :

$$F_{k,2}(x) = \frac{F_{1,1}(x)(x_k - x) - F_{k,1}(x)(x_1 - x)}{x_k - x_1}.$$

Nous calculons les éléments du tableau progressivement afin de déterminer $F_{n,n}(x) = P(x)$.

Démonstration du théorème 2.2.1 : Par récurrence sur j : $F_{k,0}(x) = f_k$ pour $k = 0..n$. Supposons que $F_{k,j}(x)$ est le polynôme de Lagrange sur les points $x_0, x_1, \dots, x_{j-1}, x_k$ pour tout $k = j..n$.

En particulier, et d'après l'hypothèse de récurrence, $F_{j,j}(x)$ est le polynôme de Lagrange sur $x_0, x_1, \dots, x_{j-1}, x_j$.

Montrons que $F_{k,j+1}(x)$ est le polynôme de Lagrange sur $x_0, x_1, \dots, x_j, x_k$ pour $k = j+1..n$. Pour $i = 0..j-1$:

$$F_{k,j+1}(x_i) = \frac{F_{j,j}(x_i)(x_k - x_i) - F_{k,j}(x_i)(x_j - x_i)}{x_k - x_j} = f_i$$

Nous utilisons pour cela l'hypothèse de récurrence : $F_{j,j}(x_i) = f_i$ et $F_{k,j}(x_i) = f_i$ pour $i = 0..j-1$. D'autre part, en utilisant la définition, on trouve que $F_{k,j+1}(x_j) = f_j$ et $F_{k,j+1}(x_k) = f_k$. En prenant $j = n$, on obtient le résultat du théorème.

Remarque 2.2.1. Le polynôme $F_{k,j}(x)$ avec $k \geq j$ est le polynôme de Lagrange sur les points $x_0, x_1, \dots, x_{j-1}, x_k$ de degré inférieur ou égal à j .

Exemple 2.2.1. L'algorithme d'Aitken appliqué à l'exemple 2.1.1 donne :

k/j		0	1	2	$x_k - x$
0	1				$-x$
1	2	$F_{1,1} = 1 + x$			$1 - x$
2	-1	$F_{2,1} = 2x + 1$	$F_{2,2} = -\frac{x^2}{2} + \frac{3x}{2} + 1$		$-1 - x$
3	4	$F_{3,1} = 1 + \frac{3x}{2}$	$F_{3,2} = \frac{x^2}{2} + \frac{x}{2} + 1$	$F_{3,3} = \frac{x^3}{3} - \frac{x^2}{2} + \frac{7x}{6} + 1$	$2 - x$

2.2.1 Formule de Newton

Soient $n \in \mathbb{N}$ et $0 \leq i < j \leq n$ deux entiers distincts. Notons $D_{i,j}(x)$ le polynôme de Lagrange sur x_i, x_{i+1}, \dots, x_j . C'est un polynôme de degré inférieur ou égal à $j - i$ vérifiant $D_{i,j}(x_k) = f_k$ pour $i \leq k \leq j$.

2.2.2 La méthode des différences divisées

Proposition 2.2.2 (relation des différences divisées). Notons $c_{i,j}$ le coefficient de x^{j-i} dans $D_{i,j}(x)$. Alors, on a la relation suivante :

$$\begin{cases} c_{i,i} = f_i & \text{pour } i = 0..n, \\ c_{i,j} = \frac{c_{i+1,j} - c_{i,j-1}}{x_j - x_i} & \text{pour } 0 \leq i < j \leq n. \end{cases}$$

Démonstration : Soit $n \in \mathbb{N}$. Pour tout $0 \leq i < j \leq n$, on a

$$D_{i,j}(x) = \frac{D_{i+1,j}(x)(x - x_i) - D_{i,j-1}(x)(x - x_j)}{x_j - x_i}. \quad (2.2.1)$$

En effet, le degré de $D_{i+1,j}(x)$ est $\leq j - i - 1$ et le degré de $D_{i,j-1}(x)$ est $\leq j - i - 1$ ce qui montre que le degré du polynôme $\frac{(x - x_i)D_{i+1,j}(x) - (x - x_j)D_{i,j-1}(x)}{x_j - x_i}$ est $\leq j - i$. De plus, $\frac{D_{i+1,j}(x_k)(x_k - x_i) - D_{i,j-1}(x_k)(x_k - x_j)}{x_j - x_i} = f_k$ (il faut distinguer les trois cas : $k = i$, $k = j$ et $k = i+1..j-1$ où $D_{i+1,j}(x_k) = D_{i,j-1}(x_k) = f_k$).

Le polynôme $\frac{D_{i+1,j}(x)(x - x_i) - D_{i,j-1}(x)(x - x_j)}{x_j - x_i}$ est alors le polynôme de Lagrange sur les points x_i, \dots, x_j et puisqu'il est unique (voir théorème 2.1.1) on a l'égalité (2.2.1) qui est équivalente à :

$$D_{i,j}(x) = \frac{D_{i+1,j}(x)(x - x_i) - D_{i,j-1}(x)(x - x_i)}{x_j - x_i} + D_{i,j-1}(x).$$

On remarque que $\frac{D_{i+1,j}(x)(x - x_i) - D_{i,j-1}(x)(x - x_i)}{x_j - x_i}$ est un polynôme de degré $\leq j - i$ qui s'annule en $x_i, x_{i+1}, \dots, x_{j-1}$ alors :

$$D_{i,j}(x) = c_{i,j}(x - x_i)(x - x_{i+1}) \dots (x - x_{j-1}) + D_{i,j-1}(x). \quad (2.2.2)$$

En examinant le coefficient de x^{j-i} de part et d'autre des deux égalités (2.2.1) et (2.2.2), nous obtenons la relation $c_{i,j} = \frac{c_{i+1,j} - c_{i,j-1}}{x_j - x_i}$:

$$c_{i,j}(x - x_i)(x - x_{i+1}) \dots (x - x_{j-1}) + D_{i,j-1}(x) = \frac{D_{i+1,j}(x)(x - x_i) - D_{i,j-1}(x)(x - x_j)}{x_j - x_i}.$$

En itérant le résultat de l'équation (2.2.2), on obtient :

$$D_{i,j}(x) = c_{i,j}(x - x_i)(x - x_{i+1}) \dots (x - x_{j-1}) + c_{i,j-1}(x - x_i) \dots (x - x_{j-2}) + \dots + c_{i,i}$$

et pour $i = 0$ et $j = n$, on a :

Définition 2.2.1 (Formule de Newton). Le polynôme de Lagrange $P(x)$ sur les points x_0, x_1, \dots, x_n est donné par la *formule de Newton* :

$$P(x) = D_{0,n}(x) = f_0 + \sum_{k=1}^{k=n} c_{0,k}(x - x_0)(x - x_1) \dots (x - x_{k-1})$$

$c_{0,k}$ est le coefficient de x^k appelée la *différence divisée d'ordre k aux points x_0, x_1, \dots, x_k* .

La *formule de Newton* permet de déterminer le polynôme de Lagrange. Pour cela, nous utilisons un tableau analogue à la méthode d'Aitken : les coefficients de $P(x)$ sont sur la diagonale du tableau triangulaire.

Exemple 2.2.2. En utilisant la méthode des différences divisées et plus précisément la proposition 2.2.2 dans l'exemple 2.1.1 on obtient :

	x_j	$i = j$	$i = j - 1$	$i = j - 2$	$i = j - 3$
$j = 0$	0	$c_{0,0} = 1$			
$j = 1$	1	$c_{1,1} = 2$	$c_{0,1} = 1$		
$j = 2$	-1	$c_{2,2} = -1$	$c_{1,2} = \frac{3}{2}$	$c_{0,2} = -\frac{1}{2}$	
$j = 3$	2	$c_{3,3} = 4$	$c_{2,3} = \frac{5}{3}$	$c_{1,3} = \frac{1}{6}$	$c_{0,3} = \frac{1}{3}$

Nous remplissons le tableau ligne par ligne avec $c_{i,i} = f_i$ pour $i = 0..n$ et $c_{i,j} = \frac{c_{i+1,j} - c_{i,j-1}}{x_j - x_i}$ pour $0 \leq i < j \leq 3$. Ainsi, $D_{0,3}(x) = \frac{1}{3}(x-0)(x-1)(x+1) + (-\frac{1}{2})(x-0)(x-1) + x + 1 = \frac{1}{3}x^3 - \frac{1}{2}x^2 + \frac{7}{6}x + 1$.

2.2.3 Algorithme de Neville

L'algorithme de calcul du polynôme de Lagrange $P(x) = D_{0,n}(x)$ pour x fixé est constitué des n étapes suivantes :

Etape 1 Pour i allant de 0 à $n-1$, calculer

$$D_{i,i+1}(x) = \frac{f_{i+1}(x - x_i) - f_i(x - x_{i+1})}{x_{i+1} - x_i}$$

Etape 2 Pour i allant de 0 à $n-2$, calculer

$$D_{i,i+2}(x) = \frac{D_{i+1,i+2}(x)(x - x_i) - D_{i,i+1}(x)(x - x_{i+2})}{x_{i+2} - x_i}$$

\vdots

Etape $n-1$ Pour i allant de 0 à 1, calculer

$$D_{i,i+n-1}(x) = \frac{D_{i+1,i+n-1}(x)(x - x_i) - D_{i,i+n-2}(x)(x - x_{i+n-1})}{x_{i+n-1} - x_i}$$

Etape n Pour $i = 0$ Calculer $D_{i,i+n}(x) = P(x)$.

2.3 Complexité d'algorithmes

Que devient le temps de calcul si on multiplie la taille des données par 2 ? Pour répondre à cette question, on considère le nombre d'opérations élémentaires (affectations, comparaisons, opérations arithmétiques) effectuées par l'algorithme. Ce nombre est couramment appelé complexité algorithmique et il s'exprime en fonction de la taille n des données. On dit que la complexité de l'algorithme est $O(h(n))$ où h est généralement une combinaison de polynômes, de logarithmes ou d'exponentielles. Ceci reprend la notation mathématique classique, et signifie que le nombre d'opérations effectuées est borné par $ch(n)$, où c est une constante, lorsque n tend vers l'infini.

La croissance de cette complexité est en fonction de la taille des données. Les algorithmes usuels peuvent être classés en un certain nombre de grandes classes de complexité (source wikipédia) :

- Les algorithmes sub-linéaires, dont la complexité est en général en $O(\log n)$. C'est le cas de la recherche d'un élément dans un ensemble ordonné fini de cardinal n .
- Les algorithmes linéaires en complexité $O(n)$ ou en $O(n \log n)$ sont considérés comme rapides, comme l'évaluation de la valeur d'une expression composée de n symboles ou les algorithmes optimaux de tri.
- Plus lents sont les algorithmes de complexité située entre $O(n^2)$ et $O(n^3)$, c'est le cas de la multiplication des matrices et du parcours dans les graphes.
- Au delà, les algorithmes polynomiaux en $O(n^k)$ pour $k > 3$ sont considérés comme lents, sans parler des algorithmes exponentiels (dont la complexité est supérieure à tout polynôme en n) que l'on s'accorde à dire impraticables dès que la taille des données est supérieure à quelques dizaines d'unités.

2.4 Estimation de l'erreur

Pour évaluer l'erreur d'interpolation de Lagrange, on a le théorème suivant :

Théorème 2.4.1. Soient f une fonction de classe C^{n+1} sur un intervalle $[a, b]$ et P le polynôme de Lagrange de f en les points $x_0 < x_1 < \dots < x_n \in [a, b]$. Alors

$$f(x) - P(x) = \frac{(x - x_0)(x - x_1) \dots (x - x_n)}{(n + 1)!} f^{(n+1)}(\zeta) \quad (2.4.1)$$

où $a \leq \min(x, x_0) < \zeta < \max(x, x_n) \leq b$.

Démonstration : La relation est triviale pour $x = x_i \forall i = 0..n$. Soit $x \neq x_i$, pour tout $i = 0..n$. On pose $k(x) = \frac{f(x) - P(x)}{(x - x_0)(x - x_1) \dots (x - x_n)}$ et $w(t)$ un polynôme qui s'annule en x_0, x_1, \dots, x_n , et x définit par : $w(t) = f(t) - P(t) - (t - x_0)(t - x_1) \dots (t - x_n)k(x)$. Il s'annule en $(n + 2)$ points. En appliquant le théorème de Rolle, on montre que $w^{(n+1)}(t)$ s'annule en au moins un point ζ :

$$w^{(n+1)}(\zeta) = f^{(n+1)}(\zeta) - (n + 1)! k(x) = 0 \quad \text{et} \quad k(x) = \frac{f^{(n+1)}(\zeta)}{(n + 1)!} \quad .$$

On remarque que l'erreur dépend de la fonction $f^{(n+1)}(\zeta)$ et des points d'interpolation par $(x - x_0)(x - x_1) \dots (x - x_n)$. Considérons le cas où les points d'interpolation sont équidistants, c'est-à-dire, $x_i = a + hi$ pour $i = 0..n$ avec $h = \frac{b-a}{n}$. Si f est continue et que toutes ses dérivées sont continues jusqu'à l'ordre $n + 1$, alors

$$|f(x) - P(x)| \leq \frac{1}{2(n + 1)} \left(\frac{b - a}{n}\right)^{n+1} \max_{a \leq x \leq b} |f^{(n+1)}(x)| \quad (2.4.2)$$

Cette quantité ne tend pas nécessairement vers 0 car les dérivées $f^{(n+1)}$ de f peuvent grandir très vite lorsque n croît. Dans l'exemple suivant, nous présentons deux cas sur la convergence de l'interpolation de Lagrange.

Exemple 2.4.1. Soit $f(x) = \sin(x)$, $|f^{(k)}(x)| \leq 1$: l'interpolé P converge vers f quelque soit le nombre de points d'interpolation et leur emplacement. Par contre, pour $f(x) = \frac{1}{1+25x^2}$ sur $[-1, 1]$ et bien que f soit indéfiniment continûment dérivable sur $[-1, 1]$, les grandeurs

$$\max_{-1 \leq x \leq 1} |f^{(k)}(x)| \quad , \quad k = 1, 2, 3, \dots$$

explosent très rapidement et l'inégalité (2.4.2) ne nous assure plus la convergence de l'interpolation.

Le choix des points x_i apporte une amélioration sensible de l'interpolation : Lorsque $a \leq x \leq b$, on choisit les abscisses d'interpolation

$$x_i = \frac{a+b}{2} + \frac{b-a}{2} \cos\left(\frac{2i+1}{2(n+1)}\pi\right)$$

pour $i = 0..n$, on obtient :

$$|f(x) - P(x)| \leq \frac{(b-a)^{n+1}}{(n+1)!2^{2n+1}} \max_{x \in [a,b]} |f^{(n+1)}(x)|.$$

Les points $x_i = \frac{a+b}{2} + \frac{b-a}{2} \cos\left(\frac{2i+1}{2(n+1)}\pi\right)$ pour $i = 0..n$ sont calculés à partir des zéros des polynômes de Tchebycheff ($T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x)$).

Nous venons de voir que l'interpolant de Lagrange peut diverger lorsque le nombre de points d'interpolation n devient grand et lorsque les points d'interpolation sont équidistants. Une première solution consiste à découper l'intervalle $[a, b]$ en plusieurs sous-intervalles, c'est l'interpolation par intervalles. On peut également utiliser des points judicieusement placés (zéros des polynômes de Tchebychev par exemple).

Exercice :

Les polynômes de Tchebychev constituent un outil important dans le domaine de l'interpolation. Soit $n \in \mathbf{N}$. Noter que pour minimiser l'erreur d'interpolation de Lagrange, les racines des polynômes de Tchebychev sont les candidats pour être les points d'interpolations. Le polynôme de Tchebychev de degré n est noté par $T_n(x) = \cos(n \arccos(x))$ pour tout $x \in [-1, 1]$ et il vérifie $T_{n+2}(x) = 2xT_{n+1}(x) - T_n(x)$ avec $T_0(x) = 1$ et $T_1(x) = x$.

1. Soit $n \in \mathbf{N}^*$. Montrer par récurrence que le coefficient du terme de degré n de T_n est 2^{n-1} .
2. Vérifier que T_n admet n racines simples définies par

$$x_k = \cos\left(\frac{2k+1}{2n}\pi\right), \quad k = 0, 1, \dots, n-1, \quad n \in \mathbf{N}^*$$

3. Sachant que la dérivée de $T_n(x)$ est définie par $T'_n(x) = \frac{n}{\sqrt{1-x^2}} \sin(n \arccos(x))$. Montrer que T_n admet exactement $(n+1)$ extrema définis par

$$x'_k = \cos\left(\frac{k\pi}{n}\right), \quad k = 0, 1, \dots, n.$$

En déduire que $T_n(x'_k) = (-1)^k$, pour $k = 0, 1, \dots, n$.

4. Vérifier que $\max_{x \in [-1, 1]} |T_n(x)| = 1$ et en déduire que

$$\max_{x \in [-1, 1]} |(x-x_0)(x-x_1)\dots(x-x_n)| = \frac{1}{2^n}.$$

2.5 Interpolation Spline

Nous rencontrons les splines dans la majorité des programmes de CAO et il est certainement bon pour un ingénieur d'en connaître l'existence et les principes.

L'interpolation spline est une approche locale en faisant de l'interpolation par morceaux : sur un sous-intervalle donné, on fait de l'interpolation polynomiale de degré faible (≤ 3). Les splines réalisent une interpolation polynomiale par morceaux et on imposera aux noeuds un degré de régularité suffisant. Avec les splines, on obtient une très bonne approximation, sans effets de bord, contrairement à l'interpolation de Lagrange.

Etant donnée une fonction f définie sur $[a, b]$ et un ensemble de noeuds $a \leq x_0 < x_1 < \dots < x_n \leq b$. Nous appelons *spline d'ordre 3* ou *spline cubique* interpolant f aux noeuds, une fonction S vérifiant les conditions suivantes :

1. S coïncide avec un polynôme P_i de degré 3 sur chacun des intervalles $[x_{i-1}, x_i]$ pour $i = 1..n$.
2. $S(x_i) = P_i(x_i) = f_i$ pour $i = 0..n$.
3. $P_i(x_i) = P_{i+1}(x_i)$ pour $i = 0..n-1$, (continuité de S).
4. $P'_i(x_i) = P'_{i+1}(x_i)$ pour $i = 0..n-1$, (continuité de S').
5. $P''_i(x_i) = P''_{i+1}(x_i)$ pour $i = 0..n-1$, (continuité de S'').
6. S satisfait une des conditions au bord suivantes :
 $S''(x_0) = S''(x_n) = 0$ (spline libre) ou bien $S'(x_0) = f'_0$ et $S'(x_n) = f'_n$ (spline forcée lorsque f est dérivable).

Proposition 2.5.1. Les polynômes $P_i(x)$ sont données par les formules suivantes :

$$P_i(x) = M_{i-1} \frac{(x_i - x)((x_i - x)^2 - h_i^2)}{6h_i} + M_i \frac{(x - x_{i-1})((x - x_{i-1})^2 - h_i^2)}{6h_i} + \frac{f_{i-1}(x_i - x)}{h_i} + \frac{f_i(x - x_{i-1})}{h_i}$$

avec $h_i = x_i - x_{i-1}$ pour $i = 1..n$, $M_0 = M_n = 0$ et M_1, M_2, \dots, M_{n-1} sont solutions du système linéaire :

$$\frac{h_i}{6} M_{i-1} + \frac{h_i + h_{i+1}}{3} M_i + \frac{h_{i+1}}{6} M_{i+1} = \frac{f_{i+1} - f_i}{h_{i+1}} - \frac{f_i - f_{i-1}}{h_i} \quad \text{pour } i = 1..n-1.$$

Indications sur la preuve : Pour montrer ces égalités, on commence par exprimer P''_i en fonction de f''_i et f''_{i+1} par une interpolation linéaire : $P''_i(x) = f''_i \frac{x_{i+1} - x}{x_{i+1} - x_i} + f''_{i+1} \frac{x - x_i}{x_{i+1} - x_i}$. En intégrant deux fois, on obtient : $P_i(x) = f''_i \frac{(x_{i+1} - x)^3}{6h_i} - f''_{i+1} \frac{(x - x_i)^3}{6h_i} + a_i(x_{i+1} - x) - b_i(x - x_i)$ où a_i et b_i sont obtenues par les conditions d'interpolation :

$$a_i = \frac{f_i}{h_i} - f''_i \frac{h_i}{6}, \quad b_i = \frac{f_{i+1}}{h_i} - f''_{i+1} \frac{h_i}{6}$$

En tenant compte des raccords aux noeuds pour P'_i et $i = 1..n-1$, on obtient un système linéaire de $(n-1)$ équations et $(n+1)$ inconnues.

$$\frac{h_i}{6} M_{i-1} + \frac{h_i + h_{i+1}}{3} M_i + \frac{h_{i+1}}{6} M_{i+1} = \frac{f_{i+1} - f_i}{h_{i+1}} - \frac{f_i - f_{i-1}}{h_i} \quad \text{pour } i = 1..n-1.$$

Exemple 2.5.1. On reprend l'exemple 2.1.1 de la section précédente. On cherche la spline cubique $S(x)$ telle que $S(0) = 1$, $S(1) = 2$, $S(-1) = -1$ et $S(2) = 4$. On pose $x_0 = -1$, $x_1 = 0$, $x_2 = 1$ et $x_3 = 2$. Ils sont bien ordonnées. Alors $f_0 = -1$, $f_1 = 1$, $f_2 = 2$, $f_3 = 4$ et $h_i = 1$ pour $i = 1..3$. On prend $M_0 = M_3 = 0$; M_1 et M_2 sont solution du système :

$$\begin{cases} \frac{2}{3} M_1 + \frac{1}{6} M_2 = -1 \\ \frac{1}{6} M_1 + \frac{2}{3} M_2 = 1 \end{cases}$$

Nous obtenons :

$$S(x) = \begin{cases} P_1(x) = -\frac{1}{3}x^3 - x^2 + \frac{4}{3}x + 1 & x \in [-1, 0] \\ P_2(x) = \frac{2}{3}x^3 - x^2 + \frac{4}{3}x + 1 & x \in [0, 1] \\ P_3(x) = -\frac{1}{3}x^3 + 2x^2 - \frac{5}{3}x + 2 & x \in [1, 2] \end{cases}$$

2.6 Introduction à l'Approximation

Soit f une fonction réelle définie soit de façon discrète, soit de façon continue. Le problème général de l'approximation consiste à déterminer une fonction g de \mathbf{R} dans \mathbf{R} de forme donnée, qui, en un certain sens, approche le mieux possible la fonction f . Un cas particulier est l'interpolation polynomiale (la fonction g est alors un polynôme). Mais l'interpolation est un outil de peu de valeur pour le traitement des données expérimentales. On l'utilise seulement lorsque les données sont très précises. Cette section traite particulièrement de la *méthode des moindres carrés discrets*.

2.6.1 Résultats fondamentaux

Il est relativement fréquent d'avoir à ajuster sur des données une loi empirique ou encore d'avoir à déterminer la valeur des paramètres d'une loi théorique de façon à reproduire les résultats d'une expérience. Une façon possible pour aborder de tels problèmes est de chercher les valeurs de paramètres minimisant un critère.

Soient $u_0, u_1, \dots, u_p, p+1$ éléments d'un espace vectoriel normé E . Pour tout $y \in E$, le problème de la meilleure approximation de y admet une solution, c'est-à-dire que la fonction *norme de l'erreur* $\epsilon(a_0, \dots, a_n) = \|y - \sum_{i=0}^n a_i u_i\|^2$ admet un minimum a dans E . Pour cela, on montre que cette fonction est continue sur un fermé borné : elle atteint donc son minimum.

Notons V le sous-espace vectoriel engendré par les $p+1$ points u_0, u_1, \dots, u_p de E deux à deux distincts. La meilleure approximation de y sur V au sens des moindres carrés est $\sum_{i=0}^{i=p} a_i u_i$ où les $a_i, i = 0..p$ sont solutions du système linéaire :

$$\sum_{i=0}^{i=p} a_i \langle u_i, u_j \rangle = \langle y, u_j \rangle \quad \text{pour} \quad j = 0..p. \quad (2.6.1)$$

2.6.2 Approximation polynomiale discrète

Il y a une manière assez simple de trouver l'équation d'une courbe. On positionne les données sur un graphique, on déplace la règle jusqu'à ce que la ligne se trouve équidistante de tous les points. L'analyse numérique change cette supposition à vue d'oeil en un processus plus scientifique c'est l'approximation au sens des moindres carrés.

Notons $E = \mathbf{R}[x]$ l'espace vectoriel des polynômes en la variable x et à coefficient dans \mathbf{R} . Soit V l'espace vectoriel des polynômes de degré $\leq p$. On pose $u_0 = x^p, u_1 = x^{p-1}, \dots, u_{p-1} = x$ et $u_p = 1$.

On considère les abscisses $x_0, x_1, \dots, x_n \in \mathbf{R}$ et f une fonction de \mathbf{R} dans \mathbf{R} telle que $f(x_i) = y_i$ pour $i = 0..n$. On veut approcher l'ensemble discret des points $(x_i, y_i)_{0 \leq i \leq n}$ par un polynôme de degré p à coefficient dans \mathbf{R} :

$$P_p(x) = a_0 x^p + a_1 x^{p-1} + \dots + a_{p-1} x + a_p \quad .$$

Pour tout $g_1, g_2 \in V$, on définit le produit scalaire sur V par

$$\langle g_1(x), g_2(x) \rangle = \sum_{i=0}^{i=n} g_1(x_i) g_2(x_i) \quad .$$

Le système (2.6.1) est alors équivalent au système suivant :

$$\begin{pmatrix} \sum_{i=0}^{i=n} x_i^{2p} & \sum_{i=0}^{i=n} x_i^{2p-1} & \sum_{i=0}^{i=n} x_i^{2p-2} & \dots & \dots & \sum_{i=0}^{i=n} x_i^p \\ \sum_{i=0}^{i=n} x_i^{2p-1} & \sum_{i=0}^{i=n} x_i^{2p-2} & \dots & \dots & \dots & \sum_{i=0}^{i=n} x_i^{p-1} \\ \sum_{i=0}^{i=n} x_i^{2p-2} & \dots & \dots & \dots & \dots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \sum_{i=0}^{i=n} x_i^p & \sum_{i=0}^{i=n} x_i^{p-1} & \dots & \dots & \sum_{i=0}^{i=n} x_i & p+1 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ \vdots \\ \vdots \\ a_p \end{pmatrix} = \begin{pmatrix} \sum y_i x_i^p \\ \sum y_i x_i^{p-1} \\ \vdots \\ \vdots \\ \sum y_i x_i \\ \sum y_i \end{pmatrix}$$

Lorsque $p = 1$, on parle de *régression* et dans ce cas, on cherche une droite $P_1(x) = a_0 x + a_1$ tel que

$$\sum_{i=0}^{i=n} (y_i - a_0 x_i - a_1)^2 \leq \sum_{i=0}^{i=n} (y_i - \alpha_0 x_i - \alpha_1)^2 \quad \forall \alpha_0, \alpha_1 \in \mathbf{R} \quad .$$

Le système (2.6.1) est alors équivalent à :

$$\begin{pmatrix} \sum_{i=0}^{i=n} x_i^2 & \sum_{i=0}^{i=n} x_i \\ \sum_{i=0}^{i=n} x_i & n+1 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \end{pmatrix} = \begin{pmatrix} \sum_{i=0}^{i=n} x_i y_i \\ \sum_{i=0}^{i=n} y_i \end{pmatrix} \quad . \quad (2.6.2)$$

En posant $X = (x_0, x_1, \dots, x_n)$ et $Y = (y_0, y_1, \dots, y_n)$, on obtient à l'aide des formules de la covariance et de la moyenne :

$$\begin{aligned} a_0 &= \frac{\sigma(X,Y)}{\sigma(X^2)} \quad \text{avec} \quad \sigma(X,Y) = \frac{1}{n+1} \sum_{i=0}^{i=n} (x_i - \bar{X})(y_i - \bar{Y}) \\ a_1 &= \bar{Y} - a_0 \bar{X} \quad \text{et} \quad \bar{X} = \frac{1}{n+1} \sum_{i=0}^{i=n} x_i \end{aligned}$$

Le point moyen (\bar{X}, \bar{Y}) appartient à la droite d'équation $P_1(x) = a_0 x + a_1$.

Exemple 2.6.1. On considère la fonction $f(x) = x^2 + 2x$ sur $[-1, 1]$. Calculer les valeurs de f aux points d'abscisses $-1, -\frac{1}{2}, 0, \frac{1}{2}$ et 1 . Donner la droite de meilleure approximation discrète de f pour ces 5 points au sens des moindres carrés. On note $x_0 = -1, x_1 = -\frac{1}{2}, x_2 = 0, x_3 = \frac{1}{2}$ et $x_4 = 1$. Alors $y_0 = -1, y_1 = -\frac{3}{4}, y_2 = 0, y_3 = \frac{5}{4}$ et $y_4 = 3$. On calcule $\sum_{i=0}^{i=4} x_i^2 = \frac{5}{2}, \sum_{i=0}^{i=4} x_i = 0, \sum_{i=0}^{i=4} x_i y_i = 5$ et $\sum_{i=0}^{i=4} y_i = \frac{5}{2}$. On obtient le système suivant à partir du système (2.6.2) :

$$\begin{pmatrix} \sum_{i=0}^{i=4} x_i^2 = \frac{5}{2} & \sum_{i=0}^{i=4} x_i = 0 \\ \sum_{i=0}^{i=4} x_i = 0 & n+1 = 5 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \end{pmatrix} = \begin{pmatrix} \sum_{i=0}^{i=4} x_i y_i = 5 \\ \sum_{i=0}^{i=4} y_i = \frac{5}{2} \end{pmatrix} \quad .$$

et $a_0 = 2, a_1 = \frac{1}{2}$. Donc $P_1(x) = 2x + \frac{1}{2}$.

2.7 Exercices d'Applications

1. Implémenter l'algorithme de Neville dans le logiciel de calcul SAGEmath pour x fixé.
2. Construire le polynôme de Lagrange de $f(x) = e^{-x^2}$ sur les entiers de $[-2, 2]$ par la méthode des différences divisées.
3. Soit la fonction f de \mathbf{R} dans \mathbf{R} définie par $f(x) = \frac{1}{1+x^2}$.
 - (a) Construire le polynôme de Lagrange P de f par la méthode des différences divisées sur 0, 1, 3 et 5.
 - (b) Comparer $P(4)$ et $f(4)$.
 - (c) Que peut-on conclure ?
4. On recueille les données suivantes concernant la météo dans la région de Bizerte le 13 janvier 2003 :

temps (heures)	0h00	5h00	7H00	12h00
pluviométrie (millimètres)	10	3	5	2

- (a) Utiliser la méthode d'Aitken pour calculer le polynôme interpolant la fonction "pluviométrie".
 - (b) Approximer la pluviométrie dans la région de Bizerte au temps $t=10h00$.
5. Soit $f(x) = x + \exp(-x^2)$. Donner la formule du polynôme d'interpolation de Lagrange $L_i(x)$ sur les points $(i, f(i))$ pour i fixé.
 6. Poser $f(x) = \exp(-x^2)$, $a = -1$, $b = 2$ et

$$\mathbf{x}_k = \frac{a+b}{2} + \frac{b-a}{2} \cos\left(\frac{2k+1}{2(n+1)}\pi\right)$$

pour k allant de 1 à 3.

- (a) Tracer l'ensemble des zéros du polynômes de Tchebycheff sur un même graphe que la fonction f .
 - (b) Calculer le polynôme de Lagrange en ces points et comparer avec la fonction f .
7. Déterminer la spline cubique des points $(-1, 0)$, $(0, 1)$, $(1, 3)$ et $(2, 2)$. Comparer cette spline avec le polynôme de Lagrange en ces même points.
 8. L'évolution de la concentration c d'une espèce dans un mélange est donnée par une loi linéaire en fonction du temps : $c(t) = a_0t + a_1$. Un expérimentateur fait une série de mesures pour déterminer les paramètres inconnus :

0	1	2	3	4	5	6	7	8	9	10	11
1,54	3,09	4,97	7,43	10,65	14,92	20,6	28,2	38,42	52,15	70,65	96,6

- (a) On pose $n = 11$. Calculer $\sum_{i=0}^{i=n} t_i^2$, $\sum_{i=0}^{i=n} t_i$, $\sum_{i=0}^{i=n} t_i c_i$ et $\sum_{i=0}^{i=n} c_i$ et écrire le système pour l'approximation au sens des moindres carrés.
- (b) Utiliser la méthode de régression des moindres carrés pour obtenir les paramètres a_0 et a_1 . Rappelons que

$$\mathbf{a}_0 = \frac{\sigma(X, Y)}{\sigma(X^2)} \text{ et } \mathbf{a}_1 = \bar{Y} - a_0 \bar{X}.$$

9. (a) Trouver le polyôme P du second degré passant par $(0, 0)$, $(1, 1)$, $(2, 8)$.

- (b) La fonction x^3 prend les valeurs de P . Utiliser la formule vue en cours pour exprimer l'erreur d'interpolation commise en remplaçant x^3 par le polynôme d'interpolation P .
10. (a) Interpoler e^x aux points $0, \frac{1}{2}$ et 1 par un polynôme du second degré.
- (b) Trouver une expression pour l'erreur d'interpolation et une borne de l'erreur indépendante de ζ .
11. (a) Sachant que la température initiale d'un objet est de 0°C , puis de 5°C après une minute et 3°C après deux minutes, quelle a été la température maximale et quant a-t-elle été atteinte?
- (b) Quelle est la température moyenne entre 1 et 2 minutes?
12. Considérons l'ensemble des points

$$\begin{aligned} x_0 &= -1, & x_1 &= 0, & x_2 &= 1, & x_3 &= 2, \\ f_0 &= 7, & f_1 &= 2, & f_2 &= -1, & f_3 &= -14. \end{aligned}$$

- (a) Calculer les polynômes d'interpolation de Lagrange L_0 et L_3 .
- (b) Vérifier que $L_i(x_i) = 1$ et que $L_i(x_j) \neq 0$ pour tout i et $j \neq i$ de 0 à 3 .
- (c) Citer deux méthodes permettant de calculer le polynôme de Lagrange.
- (d) Vérifier que le polynôme de Lagrange sur les points $(x_0, f_0), (x_1, f_1), (x_2, f_2)$ et (x_3, f_3) est égal à :

$$P(x) = -2x^3 + x^2 - 2x + 2.$$

- (e) Afin de déterminer une approximation du nuage de points (x_i, f_i) pour i de 0 à 3 , calculer la droite de régression linéaire P_0 par la méthode des moindres carrés discrets.
- (f) Quelle est la différence entre P et P_0 .

Méthode d'Hermite. Soit x_0, x_1, \dots, x_n , $(n+1)$ points de a, b et f une fonction de classe C^1 sur a, b . Au lieu de chercher à faire coïncider f et P_n aux points x_i , nous pouvons aussi chercher à faire coïncider f et P_n ainsi que leurs dérivées jusqu'à un ordre donné aux points x_i . De la même manière que dans l'interpolation de Lagrange, nous cherchons un polynôme P_n tel que $P_n(x_i) = f(x_i)$ et $P'_n(x_i) = f'(x_i)$.

Chapitre 3

Intégration numérique

3.1 Introduction

Soit f une fonction continue d'un intervalle $[a, b]$ dans \mathbf{R} . On désire approcher numériquement la quantité

$$\int_a^b f(x) dx \quad .$$

Pour cela, on subdivise l'intervalle $[a, b]$ en N intervalles : $x_i = a + ih$, $i = 0..N - 1$ et $h = \frac{b-a}{N}$. On peut alors approcher $\int_a^b f(x) dx$ par la somme des aires des rectangles de longueur $f(\frac{x_{i+1}+x_i}{2})$ et de largeur $x_{i+1} - x_i$. C'est la *méthode des rectangles*.

Exercice 3.1.1. Calculer l'aire comprise entre l'axe des abscisses, l'axe des ordonnées, la droite $x = 1$ et la courbe $f(x) = e^{x^2}$ avec la méthode du rectangle.

La méthode de Newton-Côtes utilise les polynômes de Lagrange $P(x)$ pour approcher la fonction $f(x)$ sur $[a, b]$; on considère comme valeur approchée de l'intégrale de f entre a et b :

$$\int_a^b P(x) dx \quad .$$

3.2 Formules de Newton-Côtes

Soit $n \in \mathbf{N}^*$. On considère la subdivision uniforme $x_i = a + ih$ avec $i = 0..n$ et $h = \frac{b-a}{n}$. Soit P le polynôme d'interpolation de Lagrange de degré $\leq n$ vérifiant $P(x_i) = f(x_i)$, $i = 0..n$. On obtient alors $\int_a^b f(x) dx \simeq \int_a^b P(x) dx = \int_a^b \sum_{i=0}^{i=n} f(x_i) L_i(x) dx$. En utilisant le changement de variables $x = a + ht$ dans les $L_i(x)$, on obtient :

$$\int_a^b P(x) dx = \sum_{i=0}^{i=n} h f(x_i) \alpha_i \quad \text{et} \quad \alpha_i = \int_0^n \prod_{i=0, k \neq i}^{k=n} \frac{t - k}{i - k} dt \quad .$$

Les nombres rationnels α_i sont indépendants de f et de $[a, b]$.

Remarque 3.2.1. Si $f = 1$, $a = 0$ et $b = 1$, on obtient que le polynôme de Lagrange associé à f soit égal à 1, d'où :

$$1 = \int_0^1 dx = \sum_{i=0}^{i=n} h f(x_i) \alpha_i = h \sum_{i=0}^{i=n} \alpha_i \quad .$$

Ce qui donne $\sum_{i=0}^{i=n} \alpha_i = n$.

Remarque 3.2.2. Pour $n = 1$, on obtient $\alpha_0 = \alpha_1 = \frac{1}{2}$, d'où

$$\int_a^b f(x) dx = \frac{b-a}{2}(f(a) + f(b)) \quad .$$

Pour $n = 2$, on obtient $\alpha_0 = \frac{1}{3}$, $\alpha_1 = \frac{4}{3}$ et $\alpha_2 = \frac{1}{3}$. D'où :

$$\int_a^b f(x) dx = \frac{b-a}{6}(f(a) + 4f(a+h) + f(b)) \quad .$$

3.2.1 Méthode du trapèze

Soit $N \in \mathbf{N}^*$. On considère la subdivision uniforme $x_i = a + ih$, $i = 0..N$ avec $h = \frac{b-a}{N}$. La méthode du trapèze consiste à appliquer la méthode de Newton-Côtes avec $n = 1$ sur chaque intervalle $[x_i, x_{i+1}]$, $i = 0..N-1$. On obtient :

$$\int_a^b f(x) dx = \sum_{i=0}^{N-1} \int_{x_i}^{x_{i+1}} f(x) dx \simeq T(h) \quad \text{et} \quad T(h) = \sum_{i=0}^{N-1} \frac{h}{2}(f(x_i) + f(x_{i+1}))$$

Théorème 3.2.1. *La méthode du trapèze est d'ordre deux. Plus précisément, si f est suffisamment dérivable et $h = \frac{b-a}{N}$, alors il existe $c \in]a, b[$ tel que*

$$\int_a^b f(x) dx - T(h) = -\frac{b-a}{12}h^2 f''(c) \quad .$$

Exercice 3.2.1. Reprendre l'exemple 3.1.1 avec la méthode du trapèze.

3.2.2 Méthode de Simpson

Soit $N \in \mathbf{N}^*$. On considère la subdivision uniforme $x_i = a + ih$, $i = 0..2N$ avec $h = \frac{b-a}{2N}$. La méthode de Simpson consiste à appliquer la méthode de Newton-Côtes avec $n = 2$ sur chaque intervalle $[x_{2i}, x_{2i+2}]$, $i = 0..N-1$. On obtient :

$$\int_a^b f(x) dx = \sum_{i=0}^{N-1} \int_{x_{2i}}^{x_{2i+2}} f(x) dx \simeq S(h)$$

et

$$S(h) = \sum_{i=0}^{N-1} \frac{h}{3}(f(x_{2i}) + 4f(x_{2i+1}) + f(x_{2i+2}))$$

Théorème 3.2.2. *La méthode de Simpson est d'ordre quatre. Plus précisément, si f est suffisamment dérivable et $h = \frac{b-a}{2N}$, alors il existe $c \in]a, b[$ tel que*

$$\int_a^b f(x) dx - S(h) = -\frac{b-a}{180}h^4 f^{(4)}(c) \quad .$$

Exercice 3.2.2. Reprendre l'exemple 3.1.1 avec la méthode de Simpson.

3.2.3 Remarque sur la Formule Composite

Les méthodes du rectangle, du trapèze et de Simpson sont des cas particuliers de la formule composite définie par :

$$\sum_{i=0}^{N-1} \frac{x_{i+1} - x_i}{2} \sum_{j=0}^n \alpha_j f\left(x_i + (x_{i+1} - x_i) \frac{t_j + 1}{2}\right) \quad .$$

Les $n + 1$ points $-1 \leq t_0 \leq \dots \leq t_n \leq 1$ sont appelés *points d'intégration* et les $n + 1$ points $\alpha_0, \dots, \alpha_n$ sont appelés *poids de la formule de quadrature*.

Remarque 3.2.3. Pour la méthode du rectangle, on prends $n = 0$, $t_0 = 0$ et $\alpha_0 = 2$. La formule composite devient

$$\sum_{i=0}^{N-1} \frac{x_{i+1} - x_i}{2} f\left(\frac{x_i + x_{i+1}}{2}\right) \quad .$$

Remarque 3.2.4. Pour la méthode du trapèze, on prends $n = 1$, $t_0 = -1$, $t_1 = 1$, $\alpha_0 = 1$ et $\alpha_1 = 1$. La formule composite devient

$$\sum_{i=0}^{N-1} \frac{x_{i+1} - x_i}{2} (f(x_i) + f(x_{i+1})) \quad .$$

Remarque 3.2.5. Pour la méthode de Simpson, on prends $n = 2$, $t_0 = -1$, $t_1 = 0$, $t_2 = 1$, $\alpha_0 = \frac{1}{3}$, $\alpha_1 = \frac{4}{3}$ et $\alpha_2 = \frac{1}{3}$. La formule composite devient

$$\sum_{i=0}^{N-1} \frac{x_{i+1} - x_i}{6} (f(x_i) + 4f\left(\frac{x_i + x_{i+1}}{2}\right) + f(x_{i+1})) \quad .$$

3.3 Exemples d'Implémentations

Voir en annexe, les simulations numériques sur ces méthodes.

Chapitre 4

Racine d'équation non linéaire

Les méthodes analytiques de résolution d'équations algébriques polynomiales sont limitées à certaines formes et à de faible degré (équations quadratiques, cubiques, quartiques, $ax^{2n}+bx^n+c=0$, etc...). La résolution des équations polynomiales de degré 4 est déjà plus pénible et à partir du degré 5, E. Galois a montré qu'il n'est pas toujours possible de donner une forme exacte des racines au moyen des coefficients (on utilise alors le groupe de l'équation, l'idéal des relations entre les racines ou encore le corps de décomposition du polynôme : ce sont des méthodes formelles [?]).

Pour la résolution des équations polynomiales, nous sommes amenés à l'utilisation des méthodes numériques. Il en est de même pour les équations contenant des fonctions transcendantes (i.e. non-algébriques) telles que $\exp(x)$, $\operatorname{ch}(x)$, etc... En étudiant les méthodes de résolution d'une équation f , on se posera deux questions :

1. La méthode converge-t-elle vers x^* où $f(x^*) = 0$. En d'autres termes la suite $(x_n)_{n \in \mathbf{N}}$ générée par la méthode converge-t-elle vers la solution x^* .
2. Dans l'affirmative, avec quelle rapidité (ceci permet de comparer les méthodes entre-elles).

4.1 Quelques algorithmes classiques

On désigne par (x_n) une suite de nombres réels $x_0, x_1, \dots, x_n, \dots$. Si une telle suite converge vers un point x^* , on écrira $(x_n) \rightarrow x^*$. En analyse numérique, on construit les suites à l'aide d'algorithmes.

4.1.1 Méthode de la bisection

On considère une fonction f continue et on cherche x^* solution de $f(x) = 0$. On suppose qu'on a localisé par tâtonnement un intervalle $[a, b]$ dans lequel la fonction f change de signe : $f(a).f(b) < 0$.

On est donc certain qu'il y a entre a et b au moins un zéro de f . Pour approcher de façon précise l'un de ces zéros, on utilise l'algorithme suivant :

Algorithme 1 (Dichotomie).

Entrée : les extrémités **a** et **b**,
la fonction **f**, la précision ϵ souhaitée et
N le nombre d'itérations maximum.

Sortie : Une approximation c de la racine x^* de **f**
ou bien un message d'erreur.

1. **Si** $f(a).f(b) > 0$ **Alors** Imprimer(*pas de solution*).
Sinon
 $n := 1$;
2. **Tant que** $n \leq N$ **Faire**
3. $c := \frac{a+b}{2}$;
4. **Si** $f(c) = 0$ **ou** $\frac{b-a}{2} \leq \epsilon$ **Alors**
5. Imprimer(c); $n := N + 2$;
- Fin Si**;
6. $n := n + 1$;
7. **Si** $f(a).f(c) > 0$ **Alors** $a := c$; **Sinon** $b := c$;
- Fin Tant que**;
- Si** $n = N + 2$ **Alors** Imprimer(c)
 Sinon Imprimer(*Après N itérations l'approximation de x^*
 obtenue est c et l'erreur maximale est $\frac{b-a}{2}$*);
 Fin Si;
- Fin Si**;

Fin.

A chaque itération, l'algorithme construit un nouvel intervalle, autour de x^* , qui est de longueur égale à la moitié de la longueur de l'intervalle précédent. Au bout de n itérations, la longueur de l'intervalle est alors de $\frac{b-a}{2^n}$.

Le nombre d'itérations nécessaire pour obtenir une précision de calcul égal à ϵ est définie par :

$$n \geq \frac{\log(b-a) - \log \epsilon}{\log 2} .$$

Remarque 4.1.1. Les avantages de cet algorithme sont la convergence assurée et une marge d'erreur sûre. Par contre il est relativement lent. Il est souvent utilisé pour déterminer une approximation initiale à utiliser avec un algorithme plus rapide.

4.1.2 Méthode de Newton

Cet algorithme se base sur la suite suivante (donner une explication géométrique) :

$$\begin{cases} x_0 & \text{donné} \\ x_{n+1} &= x_n - \frac{f(x_n)}{f'(x_n)}. \end{cases}$$

La signification géométrique consiste à mener par le point $(x_n, f(x_n))$ la tangente à la courbe $f(x)$. La pente de cette tangente est

$$T(x) = f(x_n) + (x - x_n)f'(x_n) .$$

Si on cherche le point d'intersection de la tangente avec l'axe des x , i.e. si on résoud $T(x) = 0$, on retrouve x_{n+1} tel que défini par la suite.

Théorème 4.1.1. [Convergence globale de la méthode de Newton] Soit f une fonction de Classe C^2 sur $[a, b]$ et $g(x) = x - \frac{f(x)}{f'(x)}$. Si f vérifie :

1. $f(a).f(b) < 0$
2. $\forall x \in [a, b], f'(x) \neq 0$ (c'est la stricte monotonie),
3. $\forall x \in [a, b], f''(x) \neq 0$ (concavité dans le même sens).

Alors en choisissant $x_0 \in [a, b]$ tel que $f(x_0).f''(x_0) > 0$, la suite (x_n) définie par x_0 et $x_{n+1} = g(x_n)$ converge vers l'unique solution de $f(x) = 0$ dans $[a, b]$.

Preuve : Les deux premières conditions du théorème assurent l'existence et l'unicité d'une racine x^* unique de f dans $[a, b]$. D'autre part, comme $x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$ on obtient :

$$\begin{aligned} x_{n+1} - x^* &= x_n - x^* - \frac{f(x_n)}{f'(x_n)} \\ &= x_n - x^* + \frac{f(x^*) - f(x_n)}{f'(x_n)} \\ &= x_n - x^* + \frac{(x^* - x_n)f'(x_n) + \frac{(x^* - x_n)^2}{2}f''(\zeta_n)}{f'(x_n)} \end{aligned}$$

Ainsi :

$$\begin{aligned} x_{n+1} - x^* &= (x_n - x^*) \left(1 - \frac{f'(x_n) + \frac{x^* - x_n}{2}f''(\zeta_n)}{f'(x_n)} \right) \\ &= \frac{(x_n - x^*)^2}{2} \frac{f''(\zeta_n)}{f'(x_n)} \end{aligned}$$

Si f'' et f' sont de même signe, alors $x_{n+1} - x^* > 0$ et la suite est alors minorée par x^* à partir du rang 1. Sinon, si f'' et f' sont de signes contraires, alors $x_{n+1} - x^* < 0$ et la suite est alors majorée. De plus, et selon les cas, la suite va être soit décroissante ($f'' > 0$ et $f' > 0$ ou bien $f'' < 0$ et $f' < 0$) donc convergente soit croissante ($f'' > 0$ et $f' < 0$ ou bien $f'' < 0$ et $f' > 0$) et donc convergente.

Algorithme 2 (Newton).

Entrée : Une approximation initiale x_0 , la précision ϵ souhaitée et N le nombre maximum d'itérations.

Sortie : Une approximation c de la racine x^* de f ou bien un mesSAGEmath d'erreur.

1. $n := 1$;
2. **Tant que** $n \leq N$ **Faire**
 $c := x_0 - \frac{f(x_0)}{f'(x_0)}$;
3. **Si** $|c - x_0| \leq \epsilon$ **Alors** c ; $n := N + 2$ **Fin Si**;
 $n := n + 1$;
 $x_0 := c$;
Fin Tant que;

Si $n = N + 2$ **Alors** Imprimer(c) **Sinon**
Imprimer(*La méthode a échoué après N itérations*);
Fin Si;

Fin.

Remarque 4.1.2. Il est essentiel de fixer une limite au nombre d'itérations car la convergence n'est pas assurée. On verra dans la suite qu'on peut construire des exemples pour lesquels cet algorithme ne converge pas. Cependant lorsque la convergence aura lieu, elle sera très rapide comme le montre le théorème 4.1.2. D'autre part, le test d'arrêt n'assure pas que la racine x^* soit à une distance ϵ de x .

Définition 4.1.1. La méthode définie par $x_{n+1} = g(x_n)$ est dite *d'ordre p* si la limite de $|\frac{e_{n+1}}{e_n^p}|$ lorsque $n \rightarrow +\infty$ est une constante réelle strictement positive.

Théorème 4.1.2. Soit f une fonction de Classe C^2 . Si x^* est une racine simple de f , alors la méthode de Newton est au moins d'ordre 2.

Preuve : On remarque que $g'(x) = \frac{f(x) \cdot f''(x)}{(f'(x))^2}$ donc $g'(x^*) = 0$. D'autre part :

$$\begin{aligned}
x_{n+1} - x^* &= g(x_n) - g(x^*) \\
&= (x_n - x^*)^2 \frac{g''(x^*)}{2} + (x_n - x^*)^2 \epsilon(e_n) \\
&= e_n^2 \frac{g''(x^*)}{2} + e_n^2 \epsilon(e_n)
\end{aligned}$$

Comme $\epsilon(e_n)$ est négligeable au voisinage de $+\infty$ on obtient le résultat. A chaque itération, on multiplie par 2 le nombre de chiffres exacts de l'approximation.

D'une façon générale, si f est de Classe C^p alors : $e_{n+1} = e_n g'(x^*) + \frac{e_n^2}{2!} g''(x^*) + \dots + \frac{e_n^p}{p!} g^{(p)}(x^*) + e_n^p \epsilon(e_n)$. La méthode est alors d'ordre p si $g'(x^*) = \dots = g^{(p-1)}(x^*) = 0$ et $g^{(p)}(x^*) \neq 0$.

Il est parfois ennuyeux dans l'utilisation de la méthode de Newton-Raphson d'avoir à calculer $f'(x)$. Il se peut par exemple qu'on ne dispose pas du programme d'ordinateur pour en effectuer le calcul alors qu'on peut facilement calculer $f(x)$. L'algorithme suivant peut être considéré comme une approximation de la méthode de Newton.

4.1.3 Méthode de la sécante

Au lieu d'utiliser la tangente au point x_n , nous allons utiliser la sécante passant par les points d'abscisses x_n et x_{n-1} pour en déduire x_{n+1} (donner une explication géométrique). L'équation de la sécante s'écrit sous la forme :

$$S(x) = f(x_n) + (x - x_n)\tau_n \quad \text{avec} \quad \tau_n = \frac{f(x_n) - f(x_{n-1})}{x_n - x_{n-1}}.$$

On définit x_{n+1} comme étant l'intersection de la sécante $S(x)$ avec l'axe des x : $S(x_{n+1}) = 0$ et on obtient

$$x_{n+1} = x_n - f(x_n) \frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})}.$$

Remarque 4.1.3. La méthode de la sécante est une méthode itérative à deux pas en ce sens que x_{n+1} dépend de x_n et de x_{n-1} . Dans la méthode de Newton-Raphson x_{n+1} ne dépend que de x_n . C'est une méthode à un pas.

Une façon commode de procéder est de choisir x_0 arbitrairement puis de choisir $x_1 = x_0 + h$ où h est un accroissement petit. Si x_n et x_{n+1} sont proches (on peut espérer que ce sera le cas

si la méthode converge), on peut considérer que $\frac{f(x_n) - f(x_{n-1})}{x_n - x_{n-1}}$ est une approximation de $f'(x_n)$. Dans ce sens l'algorithme de la sécante est une approximation de l'algorithme de Newton.

Algorithme 3 (Algorithme de la sécante).

Entrée : Deux approximations initiale x_0 et x_1 ,
la précision ϵ souhaitée et **N** le nombre maximum d'itérations

Sortie : Une approximation c de la racine x^* de f ou bien un mesSAGEmath d'erreur.

1. $n := 2$;
 $q_0 := f(x_0)$;
 $q_1 := f(x_1)$;
2. **Tant que** $n \leq N + 1$ **Faire**
 $c := x_1 - q_1 \frac{x_1 - x_0}{q_1 - q_0}$;
3. **Si** $|c - x_1| \leq \epsilon$ **Alors** Imprimer(c) ; **Fin.**
4. $n := n + 1$;
 $x_0 := x_1$; $x_1 := c$;
 $q_0 := q_1$; $q_1 := f(c)$;
Fin Tant que ;
Imprimer(*La méthode a échoué après N itérations*) ;

Fin.

4.1.4 Méthode du point fixe

On peut toujours transformer un problème du type $f(x) = 0$ en un problème de la forme $g(x) = x$ et ce d'une infinité de façons. Par exemple, la méthode de Lagrange est donnée par $x_{n+1} = g(x_n)$ où $g(x) = \frac{af(x) - xf(a)}{f(x) - f(a)}$ (on pose alors $x_0 = b$). Il faut toutefois noter que de telles transformations introduisent parfois des solutions parasites. Par exemple, on peut écrire $f(x) = \frac{1}{x} - 3 = 0$ ou encore $x = 2x - 3x^2 = g(x)$ et 0 est solution de la seconde équation mais

pas de la première.

Le théorème suivant caractérise les fonctions g qui donnent des suites convergentes :

Théorème 4.1.3. *Si dans l'intervalle $[a, b]$, g vérifie les conditions suivantes :*

1. $x \in [a, b]$ alors $g(x) \in [a, b]$,
2. l'application g est strictement contractante, c'est-à-dire $\max_{x \in [a, b]} |g'(x)| = L < 1$.

Alors pour tout $x_0 \in [a, b]$, la suite définie par $x_{n+1} = g(x_n)$ converge vers l'unique point fixe x^* de g sur $[a, b]$.

Il est souvent délicat de déterminer un intervalle $[a, b]$ dans lequel les hypothèses du théorème 4.1.3 sont vérifiées. Si on peut estimer $|g'(x^*)|$, on a le résultat suivant :

Théorème 4.1.4. *Soit x^* une solution de l'équation $x^* = g(x^*)$. Si g' est continue et $|g'(x^*)| < 1$ alors il existe un intervalle $[a, b]$ contenant x^* pour lequel la suite définie par $x_0 \in [a, b]$ et $x_{n+1} = g(x_n)$ converge vers x^* .*

Proposition 4.1.5. *Soit x^* une solution de l'équation $x^* = g(x^*)$. Si g' est continue au voisinage de x^* et si $|g'(x^*)| > 1$ alors pour tout $x_0 \in [a, b]$ différent de x^* , la suite $x_{n+1} = g(x_n)$ ne converge pas vers x^* .*

4.2 Méthodes d'accélération de la convergence

On veut savoir si la suite définie par $x_{n+1} = g(x_n)$ converge rapidement et comment se comporte l'erreur $e_n = x_n - x^*$ d'une itération à la suivante ?

4.2.1 Procédé Δ^2 d'Aitken

Dans les méthodes d'ordre 1 (i.e. $g'(x^*) \neq 0$), on considère la suite

$$y_n = x_n - \frac{(x_{n+1} - x_n)^2}{x_{n+2} - 2x_{n+1} + x_n}.$$

C'est une meilleure approximation de x^* que x_n . Lorsque x_n est proche de la solution x^* , $(x_{n+1} - x_n)^2$ et $x_{n+2} - 2x_{n+1} + x_n$ sont voisins de 0. Le calcul de y_n est alors instable. On utilise alors une autre écriture équivalente à la précédente mais plus stable :

$$y_n = x_{n+1} + \frac{1}{\frac{1}{x_{n+2} - x_{n+1}} - \frac{1}{x_{n+1} - x_n}}.$$

4.2.2 Méthode de Steffensen

On poursuit les itérations en remplaçant x_n par y_n et on calcule ainsi $g(y_n) = z_n$ et $g(g(y_n)) = g(z_n)$. On construit formellement la méthode de Steffensen $z_{n+1} = G(z_n)$ où $G(x) = \frac{xg(g(x)) - g(x)^2}{g(g(x)) - 2g(x) + x}$. La méthode de Steffensen est d'ordre au moins 2 si (x_n) est d'ordre 1 et si (x_n) est d'ordre p alors (z_n) est d'ordre $2p - 1$.

4.3 Exercices d'Applications

1. Soit $f(x) = x^2 - \exp(-1 - x^2)$ une fonction définie sur $[0, 5]$.
 - (a) Tracer f , f' et f'' sur $[0, 5]$.
 - (b) Montrer qu'il existe une racine unique de f sur $[0, 5]$.
 - (c) Appliquer la méthode de Newton avec $x_0 = 5$.
 - (d) Est-ce que la méthode de Newton converge pour $x_0 = 0$? justifier votre réponse.
 - (e) Appliquer la méthode de la sécante à f avec $x_0 = 0$. Que remarquez-vous?
 - (f) Trouver une fonction g vérifiant $g(x) = x$ si et seulement si $f(x) = 0$ qui assure les conditions du théorème du point fixe. Appliquer la méthode des itérations successives.
 - (g) Citer deux procédés pour accélérer la convergence des algorithmes.
2. Refaire l'exercice précédent avec $f(x) = x^3 - 4x + 1$ sur $[0, 0.5]$ et $g(x) = x - f(x)$.
3. (a) Donner le théorème de convergence globale de la méthode de Newton pour une fonction de classe C^2 .
 - (b) Ecrire l'algorithme de Newton qui prend en entrée les points x_0 , $a < b$ et une fonction f et rend la racine de $f(x) = 0$ sur $[a, b]$ ou bien un mesSAGEMath d'erreur.
 - (c) Calculer la racine de $f(x) = -2x^3 + x^2 - 2x + 2$ sur l'intervalle $[0.5, 1]$ avec $x_0 = 1$. La précision des calculs est à 10^{-6} près.
4. Soit f une fonction de \mathbf{R} dans \mathbf{R} telle que

$$f(x) = x^3 + 3x^2 - 1 \quad \text{pour} \quad x \in \mathbf{R}.$$

- (a) Montrer que l'équation $f(x) = 0$ admet une unique solution sur $[0, 1]$.
 - (b) Nous désirons déterminer la solution de l'équation $f(x) = 0$ avec la méthode de Newton sur $[0, 1]$. En quoi consiste cette méthode?
 - (c) Ecrire l'algorithme général de la méthode de Newton qui prend en entrée les points x_0 , $a < b$ et une fonction f et rend la racine de $f(x) = 0$ sur $[a, b]$ ou bien un mesSAGEMath d'erreur.
 - (d) Est-ce que la fonction f vérifie les conditions du théorème de convergence globale de la méthode de Newton pour $x_0 = 1$. Donner le théorème.
 - (e) Calculer la racine de f sur l'intervalle $[0, 1]$ avec une erreur de 10^{-6} .
5. On cherche le zéro réel de $f(x) = x^3 + 4x^2 - 10$.
 - (a) En choisissant $x_0 = 1.5$, appliquer la méthode du point fixe aux transformations suivantes :
 - i. $g_1(x) = x - x^3 - 4x^2 + 10$
 - ii. $g_2(x) = \frac{1}{2}\sqrt{10 - x^3}$
 - iii. $g_3(x) = \sqrt{\frac{10}{4+x}}$

On remarquera que l'algorithme ne converge pas pour g_1 , alors qu'il converge mais à des vitesses différentes pour les autres.
 - (b) Approximer la racine de f avec la méthode de Newton et donner un nombre d'itérations n qui donne une précision de x^* à 10^{-9} près.
6. Implémenter l'ensemble des algorithmes de ce chapitre sur SAGEMath.

Chapitre 5

Résolution de systèmes d'équations linéaires

Un des points essentiels dans l'efficacité des méthodes envisagées concerne la taille des systèmes à résoudre. Entre 1980 et 2000, la taille de la mémoire des ordinateurs a augmenté. La taille des systèmes qu'on peut résoudre sur ordinateur a donc également augmenté, selon l'ordre de grandeur suivant :

1980	Matrice pleine	$n = 10^2$
	Matrice creuse	$n = 10^6$
2000	Matrice pleine	$n = 10^6$
	Matrice creuse	$n = 10^8$

Le développement des méthodes de résolution de systèmes linéaires est lié à l'évolution des machines informatiques. Un grand nombre de recherches sont d'ailleurs en cours pour profiter au mieux de l'architecture des machines (méthodes de décomposition en sous domaines pour profiter des architectures parallèles, par exemple).

De nombreux efforts ont été consacrés au développement de méthodes de résolution de systèmes d'équations linéaires. Les méthodes standards incluent l'élimination de Gauss-Jordan, et la décomposition LU. Les méthodes itératives telles que la méthode du gradient conjugué sont généralement préférées sur les larges systèmes d'équations.

L'analyse numérique matricielle s'intéresse principalement à deux types de problèmes :

- Résoudre un système linéaire $Ax = b$;
- Calculer les valeurs propres et vecteurs propres d'une matrice donnée.

Nous étudierons pour cela des algorithmes destinés à des matrices de grande taille. Comme il s'agit de calculs volumineux sur machine, à chaque opération des erreurs systématiques dites d'arrondi sont générées. Il convient donc de se préoccuper de l'influence de ces petites erreurs sur la solution du problème et donc d'étudier la stabilité. Nous nous intéresserons d'autre part, au coût de ces algorithmes en termes d'opérations élémentaires.

Il s'agira donc de construire et d'analyser des méthodes numériques pour la solution de problèmes d'algèbre linéaire et d'analyse. On étudiera les notions de stabilité, de consistance, de convergence des méthodes numériques.

5.1 Rappels et compléments sur les matrices

Les matrices diagonales et triangulaires jouent un rôle particulier en analyse numérique matricielle pour la raison évidente que les systèmes linéaires avec de telles matrices sont de résolution très simple. Aussi nous nous intéresserons ici aux changements de base permettant d'accéder à des structures de ce type.

Nous considérons un espace vectoriel de dimension n sur \mathbf{R} ou sur \mathbf{C} . Toutes les matrices considérées dans la suite sont des matrices carrées.

Rappelons qu'une matrice est dite *régulière* si elle est inversible, sinon elle est *singulière*. Une matrice A est dite *normale* si $AA^* = A^*A$ où A^* est la matrice adjointe définie par $\langle Au, v \rangle = \langle u, A^*v \rangle$ pour tout $u, v \in \mathbf{C}^n$. Le signe \langle, \rangle dénote le produit scalaire dans \mathbf{C}^n , soit $\langle u, v \rangle = {}^t\bar{u}v$. Une matrice *unitaire* est une matrice vérifiant $A^*A = AA^* = I_n$. Elle est *hermitienne* si $A^* = A$.

Une matrice est dite *symétrique* si elle est réelle et si ${}^tA = A$ (dans ce cas, ${}^tA = A^*$). La matrice A est *orthogonale* si ${}^tAA = A{}^tA = I_n$.

Théorème 5.1.1. *Etant donné une matrice carrée A , il existe une matrice unitaire U telle que $U^{-1}AU$ soit triangulaire. Etant donné une matrice normale A , il existe une matrice unitaire U telle que $U^{-1}AU$ soit diagonale.*

5.2 Normes matricielles

Soit V un espace vectoriel de dimension n sur \mathbf{R} ou \mathbf{C} , on peut définir plusieurs normes équivalentes sur V dont :

$$\begin{aligned} \|x\|_1 &= \sum_{i=1}^n |x_i|, & x &= {}^t(x_1, x_2, \dots, x_n) \in V \\ \|x\|_\infty &= \max_{i=1..n} |x_i| \\ \|x\|_2 &= \sqrt{\sum_{i=1}^n |x_i|^2} = \sqrt{{}^t\bar{x}x}, & \text{norme de schur} \\ \|x\|_p &= (\sum_{i=1}^n |x_i|^p)^{\frac{1}{p}}. \end{aligned}$$

Si V est muni d'une norme $\|\cdot\|$, on définit sur l'ensemble des matrices carrées d'ordre n la norme suivante :

$$\|A\| = \sup_{x \neq 0} \frac{\|Ax\|}{\|x\|} = \max_{\|x\|=1} \|Ax\|$$

Cette quantité définit bien une norme puisqu'elle vérifie

$$\begin{aligned} \|A\| = 0 &\Rightarrow A = 0 \\ \|\lambda A\| &= |\lambda| \|A\| \\ \|A + B\| &\leq \|A\| + \|B\| \end{aligned}$$

Nous considérons un système de n équations linéaires à n inconnues

$$\begin{pmatrix} a_{1,1}x_1 + a_{1,2}x_2 + \dots + a_{1,n}x_n & = & b_1 \\ a_{2,1}x_1 + a_{2,2}x_2 + \dots + a_{2,n}x_n & = & b_2 \\ \vdots & & \vdots \\ a_{n,1}x_1 + a_{n,2}x_2 + \dots + a_{n,n}x_n & = & b_n \end{pmatrix}$$

Les $a_{i,j}$ et les b_i étant fixés.

Ce système est équivalent à $Ax = b$ avec

$$A = \begin{pmatrix} a_{1,1} & a_{1,2} & \dots & a_{1,j} & \dots & a_{1,n} \\ a_{2,1} & a_{2,2} & \dots & a_{2,j} & \dots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ a_{i,1} & a_{i,2} & \dots & a_{i,j} & \dots & a_{i,n} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ a_{n,1} & a_{n,2} & \dots & a_{n,j} & \dots & a_{n,n} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_j \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_i \\ \vdots \\ b_n \end{pmatrix}$$

5.2.1 Conditionnement d'une matrice

Un problème est *bien conditionné* si une petite variation des données entraîne une petite variation sur la solution, *mal conditionné* dans le cas contraire.

Définition 5.2.1. On appelle *conditionnement* d'une matrice le nombre le nombre

$$c(A) = \|A\| \cdot \|A^{-1}\|$$

On se propose d'examiner la sensibilité d'une méthode de résolution :

1. aux erreurs d'arrondis
2. aux "petites perturbations" des données du problème.

On mesurera cette sensibilité par un nombre appelé conditionnement, qui dépend d'une norme sur K^n . Lorsqu'il est très élevé, cela peut mettre cause la pertinence du résultat numérique obtenu.

On part d'un système $Ax = b$, de taille $n \times n$, et on se donne par ailleurs ΔA et Δb , présumés petits par rapport à A et b et tel que $A + \Delta A = \bar{A}$ soit aussi inversible. On se propose de comparer les solutions x et \bar{x} des systèmes : $Ax = b$ et le système perturbé $\bar{A}\bar{x} = \bar{b}$. Soient le système $Ax = b$ et le système perturbé $\bar{A}\bar{x} = \bar{b}$. Les erreurs relatives sur A et b sont : $\epsilon_A = \frac{\|\bar{A}-A\|}{\|A\|}$, $\epsilon_b = \frac{\|\bar{b}-b\|}{\|b\|}$.

Théorème 5.2.1. Si $\epsilon_A c(A) < 1$: $\frac{\|\bar{x}-x\|}{\|x\|} \leq \frac{c(A)}{1-\epsilon_A c(A)} (\epsilon_A + \epsilon_b)$.

Le théorème 5.2.1 donne une borne de l'erreur relative sur x : la solution du système $Ax = b$ risque d'être très sensible aux variations sur les données si le conditionnement est élevé.

Un problème est bien conditionné si une petite variation des données entraîne une petite variation sur la solution, mal conditionné dans le cas contraire.

Exemple 5.2.1. On souhaite résoudre le système linéaire $Ax = b$, où A et b sont donnés par :

$$A = \begin{pmatrix} 10 & 7 & 8 & 7 \\ 7 & 5 & 6 & 5 \\ 8 & 6 & 10 & 9 \\ 7 & 5 & 9 & 10 \end{pmatrix} \text{ et } b = \begin{pmatrix} 32 \\ 23 \\ 33 \\ 31 \end{pmatrix}. \text{ La solution du système est : } x = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}.$$

$$\text{On résout le système perturbé } \bar{A}\bar{x} = \bar{b} \text{ avec } \bar{A} = \begin{pmatrix} 32.1 \\ 22.9 \\ 33.1 \\ 30.9 \end{pmatrix}. \text{ On trouve } \bar{x} = \begin{pmatrix} 9.2 \\ -12.6 \\ 4.5 \\ -11 \end{pmatrix}.$$

Autrement dit, de très petites variations sur b ont conduit à de grandes variations sur x . Le conditionnement de A est $c(A)=4488 \gg 1$. Ce phénomène de mauvais conditionnement explique pour partie la difficulté de prévoir certains phénomènes. Les appareils de mesure ne sont jamais parfaits, et il est impossible de connaître exactement b . Cela peut entraîner une très grande imprécision sur la valeur de x .

Les conditionnements de matrice utilisés dans la pratique correspondent aux trois normes usuelles $\|\cdot\|_p$, $p=1, 2, \infty$. On note $c(A) = \|A\|_p \|A^{-1}\|_p$.

Si le conditionnement de A est de l'ordre 1 alors on dit que A est bien conditionnée sinon, elle est mal conditionnée.

5.2.2 La méthode de Gauss

Exercice 1. On considère le système $Ax = b$ suivant :

$$\begin{pmatrix} 3 & -1 & 0 \\ -2 & 1 & 1 \\ 2 & -1 & 4 \end{pmatrix} x = \begin{pmatrix} 5 \\ 0 \\ 15 \end{pmatrix}$$

$$\begin{array}{ccc|c} 3 & -1 & 0 & 5 \\ -2 & 1 & 1 & 0 \\ 2 & -1 & 4 & 15 \end{array} \quad \begin{array}{l} l_2 \Rightarrow l_2 + 2/3 l_1 \\ l_3 \Rightarrow l_3 - 2/3 l_1 \end{array} \quad \Leftrightarrow \quad \begin{array}{ccc|c} 3 & -1 & 0 & 5 \\ 0 & 1/3 & 1 & 10/3 \\ 0 & -1/3 & 4 & 35/3 \end{array} \quad \begin{array}{l} \\ l_3 \Rightarrow l_3 + l_2 \end{array}$$

Nous effectuons une élimination ascendante (back-substitution) ou une remontée afin de résoudre le nouveau système triangulaire obtenu par la méthode de Gauss :

$$\Leftrightarrow \begin{array}{ccc|c} 3 & -1 & 0 & 5 \\ 0 & 1/3 & 1 & 10/3 \\ 0 & 0 & 5 & 5 \end{array} \quad \begin{array}{l} x_3 = 3 \\ x_2 = 1 \\ x_1 = 2 \end{array}$$

Nous considérons un système de n équations linéaires à n inconnues

$$A = \begin{pmatrix} a_{1,1}x_1 + a_{1,2}x_2 + \cdots + a_{1,j}x_j + \cdots + a_{1,n}x_n & = & b_1 \\ a_{2,1}x_1 + a_{2,2}x_2 + \cdots + a_{2,j}x_j + \cdots + a_{2,n}x_n & = & b_2 \\ \vdots & & \vdots \\ a_{i,1}x_1 + a_{i,2}x_2 + \cdots + a_{i,j}x_j + \cdots + a_{i,n}x_n & = & b_i \\ \vdots & & \vdots \\ a_{n,1}x_1 + a_{n,2}x_2 + \cdots + a_{n,j}x_j + \cdots + a_{n,n}x_n & = & b_n \end{pmatrix}$$

les $a_{i,j}$ et les $b_{i,j}$ étant fixés. Ce système est équivalent à $Ax = b$ et par combinaisons linéaires successives des lignes du système, on triangularise supérieurement. La résolution du système $A^{(1)}x = b^{(1)}$ avec $A^{(1)} = A$ une matrice inversible telle que $a_{1,1}^{(1)} \neq 0$:

$$A^{(1)} = \begin{pmatrix} a_{1,1}^{(1)} & a_{1,2}^{(1)} & \cdots & a_{1,j}^{(1)} & \cdots & a_{1,n}^{(1)} \\ a_{2,1}^{(1)} & a_{2,2}^{(1)} & \cdots & a_{2,j}^{(1)} & \cdots & a_{2,n}^{(1)} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{i,1}^{(1)} & a_{i,2}^{(1)} & \cdots & a_{i,j}^{(1)} & \cdots & a_{i,n}^{(1)} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{n,1}^{(1)} & a_{n,2}^{(1)} & \cdots & a_{n,j}^{(1)} & \cdots & a_{n,n}^{(1)} \end{pmatrix} \quad \text{et} \quad b^{(1)} = \begin{pmatrix} b_1^{(1)} \\ b_2^{(1)} \\ b_3^{(1)} \\ \vdots \\ b_n^{(1)} \end{pmatrix}.$$

Annuler l'élément en position $(i, 1)$ pour $i = 1, \dots, n$ revient à soustraire de la i ème ligne, la première ligne multipliée par $m_{i,1} = a_{i,1}^{(1)}/a_{1,1}^{(1)}$.

Cet ensemble d'éliminations revient à prémultiplier la matrice $A^{(1)}$ par une matrice M_1 vérifiant

$$M_1 = \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 \\ -m_1 & 1 & 0 & \cdots & 0 \\ -m_2 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ -m_n & 0 & 0 & \cdots & 1 \end{pmatrix}$$

$a_{1,1}$ est appelé *élément pivot de la transformation* M_1 et on obtient un nouveau système équivalent au système de départ : $M_1 A^{(1)} x = M_1 b^{(1)}$ où

$$A^{(2)} = \begin{pmatrix} a_{1,1}^{(1)} & a_{1,2}^{(1)} & \cdots & a_{1,j}^{(1)} & \cdots & a_{1,n}^{(1)} \\ 0 & a_{2,2}^{(2)} & \cdots & a_{2,j}^{(2)} & \cdots & a_{2,n}^{(2)} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & a_{i,2}^{(2)} & \cdots & a_{i,j}^{(2)} & \cdots & a_{i,n}^{(2)} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & a_{n,2}^{(2)} & \cdots & a_{n,j}^{(2)} & \cdots & a_{n,n}^{(2)} \end{pmatrix} \quad \text{et} \quad b^{(2)} = M_1 b^{(1)} = \begin{pmatrix} b_1^{(1)} \\ b_2^{(2)} \\ b_3^{(2)} \\ \vdots \\ b_n^{(2)} \end{pmatrix}$$

les seuls éléments à calculer sont les $a_{i,j}^{(2)}$, pour $i = 2..n$ et $b_i^{(2)}$, $i = 2..n$. Ce calcul s'effectue grâce aux formules suivantes :

$$\begin{cases} m_{i,1} &= a_{i,1}^{(1)}/a_{1,1}^{(1)} \\ b_i^{(2)} &= m_{i,1} b_i^{(1)} \\ a_{i,j}^{(2)} &= a_{i,j}^{(1)} - m_{i,1} a_{1,j}^{(1)} \end{cases}$$

De la même manière, on réitère la procédure de l'élimination de Gauss en supposant que $a_{2,2}^{(2)} \neq 0$ selon les formules

$$\begin{cases} m_{i,2} &= a_{i,2}^{(2)}/a_{2,2}^{(2)} \\ b_i^{(3)} &= m_{i,2} b_i^{(2)} \\ a_{i,j}^{(3)} &= a_{i,j}^{(2)} - m_{i,2} a_{2,j}^{(2)} \end{cases}$$

par application de $n - 1$ transformations de même type de M_1 et M_2 on obtient finalement, pour autant que les pivots successifs soient non nuls, le système : $M_{n-1} M_{n-2} \dots M_2 M_1 A^{(1)} x = M_{n-1} M_{n-2} \dots M_2 M_1 b^{(1)}$ équivalent au système $A^{(1)} x = b^{(1)}$, mais dont la matrice est triangulaire supérieure.

Complexité de calcul

Le coût de la méthode de Gauss est donné par : $\sum_{k=1}^{n-1} (2(n-k)(n-k+1) + n-k) = \sum_{i=1}^{n-1} (2i(i+1) + i)$ opérations soit $\frac{2}{3}n^3 + \frac{1}{2}n^2 - \frac{7}{6}n$ auxquelles il faut encore ajouter n^2 opérations pour l'élimination ascendante (algorithme de remontée). La méthode de Gauss exige donc finalement le calcul de $\frac{2}{3}n^3 + \frac{3}{2}n^2 - \frac{7}{6}n$ opérations en tout. *Tout ceci n'est évidemment valable que si les pivots successifs des transformations sont non nuls.*

Remarque 5.2.1. Si, au cours des calculs, on rencontre un pivot nul, il faut permuter la ligne de ce pivot avec une ligne d'indice supérieur, présentant un pivot non nul dans la même colonne. Ceci sera toujours possible si le système donné est compatible ($\det \mathbf{A} \neq 0$). En pratique, on ne permute pas les lignes entre-elles, on change simplement le numéro des lignes à permuter en stockant dans un vecteur de travail les indices des pivots successifs. Pour la "back-substitution" on tient compte de l'ordre utilisé lors de la triangularisation.

5.2.3 La factorisation LU d'une matrice

On dit qu'une matrice $A(n, n)$ possède une factorisation LU si $A = LU$ avec L une matrice (n, n) triangulaire inférieure à diagonale unité et U une matrice triangulaire supérieure.

Proposition 5.2.2 (Unicité). *On suppose que A est régulière et admet une factorisation $A = LU$, alors U est régulière et la factorisation est unique.*

Preuve : $\det(A) = \det(LU) = \det(L)\det(U) = \det(U)$.

Si A est régulière ($\det(A) \neq 0$) alors U l'est aussi. Supposons que $A = L_1 U_1 = L_2 U_2$ alors $L_2^{-1} L_1 = U_2 U_1^{-1} = D$. Comme L_2 est une matrice triangulaire inférieure à diagonale unité donc L_2^{-1} triangulaire inférieure à diagonale unité. De même, comme L_1 est une matrice triangulaire inférieure à diagonale unité donc $D = L_2^{-1} L_1$ est une matrice triangulaire inférieure à diagonale unité.

D'autre part, U_1 est une matrice triangulaire supérieure donc U_1^{-1} est une matrice triangulaire supérieure et U_2 est une matrice triangulaire supérieure donc $D = U_2 U_1^{-1}$ est une matrice triangulaire supérieure. Ainsi D est la matrice unité, $L_1 = L_2$ et $U_1 = U_2$.

Proposition 5.2.3 (Existence). *On suppose que l'élimination de Gauss est faisable sans permutation de lignes sur un système de matrice A supposée régulière. Alors A possède une factorisation LU avec U régulière.*

La méthode de Gauss fournit un système à matrice triangulaire supérieure :

$\mathbf{M}_{n-1} \mathbf{M}_{n-2} \dots \mathbf{M}_2 \mathbf{M}_1 \mathbf{A} \mathbf{x} = \mathbf{M}_{n-1} \mathbf{M}_{n-2} \dots \mathbf{M}_2 \mathbf{M}_1 \mathbf{b} = \mathbf{b}_1$. Posons $\mathbf{M}_{n-1} \mathbf{M}_{n-2} \dots \mathbf{M}_2 \mathbf{M}_1 \mathbf{A} = \mathbf{U}$.

Comme les \mathbf{M}_i sont triangulaires inférieures à diagonale unité le produit

$\mathbf{M}_{n-1} \mathbf{M}_{n-2} \dots \mathbf{M}_2 \mathbf{M}_1$ est aussi triangulaire inférieure à diagonale unité et l'on a :

$\mathbf{A} = (\mathbf{M}_{n-1} \mathbf{M}_{n-2} \dots \mathbf{M}_2 \mathbf{M}_1)^{-1} \mathbf{U} = \mathbf{L} \mathbf{U}$ où $\mathbf{L} = (\mathbf{M}_{n-1} \mathbf{M}_{n-2} \dots \mathbf{M}_2 \mathbf{M}_1)^{-1}$ est une matrice triangulaire inférieure à diagonale unitaire.

Théorème 5.2.4. *La matrice $A(n, n)$ possède une factorisation LU avec U régulière si et seulement si ses sous-matrices principales A^k $k=1 \dots n$ sont régulières.*

Construction de \mathbf{A} : Si on connaît une factorisation (on dit aussi décomposition) LU de la matrice \mathbf{A} du système $\mathbf{A} \mathbf{x} = \mathbf{b}$, sa solution s'obtient de la façon suivante : 1°) on calcule la solution du système $\mathbf{L} \mathbf{y} = \mathbf{b}$ 2°) on résout ensuite $\mathbf{U} \mathbf{x} = \mathbf{y}$. En effet : $\mathbf{A} \mathbf{x} = \mathbf{L} \mathbf{U} \mathbf{x} = \mathbf{L} \mathbf{y} = \mathbf{b}$.

La connaissance de la décomposition LU permet de résoudre le système de départ au prix de $2n^2$ opérations arithmétiques (*supplémentaires*) puisqu'il faut résoudre deux systèmes triangulaires à n inconnues.

Remarque 5.2.2. Le coût de la factorisation LU est du même ordre que celui de la méthode de Gauss soit $2n^2/3$.

La méthode ne fonctionne qu'avec les matrices carrées régulières ($\det \mathbf{A} \neq 0$). La méthode fournit la matrice inverse de \mathbf{A} grâce à la relation $\mathbf{A}^{-1} = \mathbf{U}^{-1}\mathbf{L}^{-1}$. La matrice \mathbf{U} est identique à la matrice triangulaire fournie par la méthode de Gauss. La matrice $\mathbf{L} = (\mathbf{M}_{n-1}\mathbf{M}_{n-2}\dots\mathbf{M}_2\mathbf{M}_1)^{-1}$ s'écrit :

$$\mathbf{L} = \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 \\ m_{21} & 1 & 0 & \cdots & 0 \\ m_{31} & m_{32} & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ m_{n1} & m_{n2} & m_{n3} & \cdots & 1 \end{pmatrix} \quad \text{où les } M_{ik} \text{ sont les facteurs multiplicatifs rencontrés lors de la triangularisation de Gauss.}$$

Exemple 5.2.2. Effectuer la factorisation LU de la matrice : $\mathbf{A} = \begin{pmatrix} 4 & -9 & 2 \\ 2 & -4 & 4 \\ -1 & 2 & 2 \end{pmatrix}$.

D'après les formules : $m_{21} = 0.5$ et $m_{31} = -0.25$ ce qui nous donne $\mathbf{A} = \begin{pmatrix} 4 & -9 & 2 \\ 0 & 0.5 & 3 \\ 0 & -0.25 & 2.5 \end{pmatrix}$.

De la même manière $m_{32} = -0.5$ et nous obtenons que $\mathbf{L} = \begin{pmatrix} 1 & 0 & 0 \\ 0.5 & 1 & 0 \\ -0.25 & -0.5 & 1 \end{pmatrix}$ et $\mathbf{U} = \begin{pmatrix} 4 & -9 & 2 \\ 0 & 0.5 & 3 \\ 0 & 0 & 4 \end{pmatrix}$.

5.2.4 La factorisation de Cholesky

Cette méthode s'applique uniquement aux matrices **symétriques définies positifs** et permet comme déjà mentionné la factorisation $\mathbf{A} = \mathbf{R}\mathbf{R}^T$ où \mathbf{R} est une matrice triangulaire inférieure.

Définition 5.2.2. Soit \mathbf{A} une matrice symétrique, on dit que \mathbf{A} est définie positive si :

$$\begin{cases} \forall \mathbf{x} \in \mathbb{R}^n & \mathbf{x}^T \mathbf{A} \mathbf{x} > 0 \\ \mathbf{x}^T \mathbf{A} \mathbf{x} = \mathbf{x} \mathbf{A} \mathbf{x}^T = 0 \Leftrightarrow \mathbf{x} = 0 \end{cases} \quad \begin{cases} \forall \mathbf{x} \in \mathbb{R}^n & \mathbf{x}^T \mathbf{A} \mathbf{x} > 0 \\ \mathbf{x}^T \mathbf{A} \mathbf{x} = \mathbf{x} \mathbf{A} \mathbf{x}^T = 0 \Leftrightarrow \mathbf{x} = 0 \end{cases}$$

Théorème 5.2.5. Une matrice \mathbf{A} symétrique est définie positive si tous ses déterminants sont positifs

Théorème 5.2.6. Si les valeurs propres d'une matrice \mathbf{A} symétrique sont toutes strictement positives alors \mathbf{A} est définie positive. De même si \mathbf{A} est définie positive alors toutes ses valeurs propres sont strictement positives.

Rappel : Les valeurs propres d'une matrice triangulaire sont les éléments de sa diagonale.

Théorème 5.2.7. Si \mathbf{A} est une matrice symétrique définie positive, alors il existe au moins une matrice réelle triangulaire inférieure \mathbf{R} tel que $\mathbf{A} = \mathbf{R}\mathbf{R}^T$ de plus si $R_{ii} > 0 \forall i=1,2,\dots,n$ alors la factorisation est unique.

Construction de \mathbf{R} par la méthode de Cholesky :

$$r_{11} = \sqrt{a_{11}}$$

$$r_{i1} = \frac{a_{i1}}{r_{11}} \text{ pour } i = 2 \dots n$$

$$r_{ii} = \sqrt{a_{ii} - \sum_{k=1}^{i-1} r_{ik}^2} \text{ pour } i = 2 \dots n$$

$$r_{ji} = \frac{1}{r_{ii}} \left[a_{ji} - \sum_{k=1}^{i-1} r_{jk} * r_{ik} \right] \text{ pour } i = 2 \dots n, j = i + 1 \dots n$$

Exemple 5.2.3. Soit $A = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 5 & 5 & 5 \\ 1 & 5 & 14 & 14 \\ 1 & 5 & 14 & 15 \end{pmatrix}$. La matrice symétrique A décomposée par la méthode de Cholesky tel que $A=RR^T$ avec $R = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 2 & 0 & 0 \\ 1 & 2 & 3 & 0 \\ 1 & 2 & 3 & 1 \end{pmatrix}$

Remarque 5.2.3. Le coût du calcul est de l'ordre de $n^2/2$ opérations arithmétiques et n extractions de racine carré, c'est plus économique que la factorisation **LU** pour une matrice symétrique.

Cette méthode fournit un test efficace pour déterminer si une matrice symétrique est définie positive ou non :

Dans de très nombreux problèmes concrets, les systèmes à résoudre ont des matrices symétriques définies positives en raison même des phénomènes qu'ils modélisent. La méthode de factorisation de Cholesky revêt donc une importance particulière.

Toutefois, comme toutes les méthodes directes, son coût de calcul et les erreurs d'arrondis deviennent un handicap quasiment insurmontable pour des systèmes comptant de l'ordre de 200 équations. En pratique on est surtout amené à résoudre des systèmes de plusieurs dizaines de milliers d'équations dont les matrices, fort heureusement, sont généralement creuses. La résolution de tels systèmes nécessite l'emploi de méthodes itératives.

5.2.5 Valeurs propres et vecteurs propres

Aujourd'hui, le calcul des valeurs et vecteurs propres est indispensable dans toutes les branches de la science, en particulier pour la solution des systèmes des équations linéaires, en théorie de stabilité, pour les questions de convergence de processus itératifs, et en physique et chimie (mécanique, circuits, cinétique chimique, équation de Schrödinger).

Il existe plusieurs méthodes pour la détermination des valeurs et vecteurs propres d'une matrice :

1. Méthode de la puissance
2. Méthode de la puissance inverse de Wielandt
3. Transformation sous forme de Hessenberg (ou tridiagonale)
4. L'itération orthogonale
5. L'algorithme QR

5.2.6 La méthode de la puissance

Définition 5.2.3. Une matrice A admet une valeur propre Dominante lorsqu'elle est la plus grande valeur propre en module :

$$|\lambda_1| \geq |\lambda_2| \geq \dots \geq |\lambda_n|$$

Le vecteur propre associé est dit dominant.

Remarque 5.2.4. La méthode de la puissance permet de calculer la valeur et le vecteur propre dominants de A .

Théorème 5.2.8. Supposons que λ et ν sont respectivement la valeur propre et le vecteur propre d'une matrice A . Pour toute constante α , on a $(\lambda - \alpha)$ est la valeur propre de $(A - \alpha I_n)$ et de vecteur propre A

Preuve : Immédiate.

Théorème 5.2.9. si λ et ν sont respectivement la valeur propre et le vecteur propre d'une matrice A inversible, alors si $\lambda \neq 0$, on a $1/\lambda$ et ν sont respectivement la valeur propre et le vecteur propre de A^{-1} .

Preuve : On a $Av = \lambda v$, $\lambda A^{-1}v = \lambda A^{-1}\lambda v = A^{-1}Av = v$ et donc $A^{-1}v = 1/\lambda v$.

Définition 5.2.4. . Un vecteur propre est dit normalisé si la plus grande de ses coordonnées est égale à 1.

Remarque 5.2.5. Il est facile de transformer un vecteur propre $v = \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{pmatrix}$ en formant un

nouveau vecteur $v' = \frac{1}{c} \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{pmatrix}$ avec $c = v_j$ et

$$|v_j| = \max_{i=1..n} |v_i|.$$

Théorème 5.2.10 (Théorème de la méthode de puissance). Supposons que la matrice A admette une valeur propre dominante et un vecteur propre normalisé v . Ces deux quantités peuvent être déterminées à partir la méthode de la puissance.

On commence par $X_0 = \begin{pmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{pmatrix}$; il est important que X_0 soit différent des autres vecteurs propres.

$X_0 = \sum_{i=1}^n \alpha_i v_i = \alpha v + \sum_{i=2}^n \alpha_i v_i$ où v est le vecteur normalisé

On génère alors une suite (X_k) de manière récursive en utilisant la définition suivante :

$$\begin{cases} Y_k = AX_k \\ X_{k+1} = C_{k+1} Y_k \end{cases}$$

où C_{k+1} est la plus grande des coordonnées en valeur absolue de Y_k

alors la suite (X_k) converge vers v (vecteur normalisé) et la suite C_k converge vers (valeur propre dominante).

$$\begin{cases} \lim_{k \rightarrow +\infty} X_k = v \\ \lim_{k \rightarrow +\infty} C_k = \end{cases}$$

Preuve : Sachant que A admet n valeurs propres associées à n vecteurs propres distincts v_j $1 \leq j \leq n$, linéairement indépendants normalisés et formant une base de dimension n. On peut alors écrire X_0 dans cette base :

$$X_0 = \alpha_1 v_1 + \alpha_2 v_2 + \dots + \alpha_n v_n, \alpha_1 \neq 0$$

on suppose que pour $X_0 = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}$, $\max_{i=1..n} |x_i| = 1$

$$Y_0 = AX_0 = A[\alpha_1 v_1 + \alpha_2 v_2 + \dots + \alpha_n v_n]$$

$$= \alpha_1 A v_1 + \alpha_2 A v_2 + \dots + \alpha_n A v_n$$

$$= \alpha_1 \lambda_1 v_1 + \alpha_2 \lambda_2 v_2 + \dots + \alpha_n \lambda_n v_n$$

$$= \lambda_1 \left[\alpha_1 \frac{\lambda_1}{\lambda_1} v_1 + \alpha_2 \frac{\lambda_2}{\lambda_1} v_2 + \dots + \alpha_n \frac{\lambda_n}{\lambda_1} v_n \right]$$

$$= \lambda_1 \left[\alpha_1 v_1 + \alpha_2 \left(\frac{\lambda_2}{\lambda_1} \right) v_2 + \dots + \alpha_n \left(\frac{\lambda_n}{\lambda_1} \right) v_n \right]$$

en normalisant, on obtient :

$$X_1 = \frac{\lambda_1}{C_1} \left[\alpha_1 v_1 + \alpha_2 \left(\frac{\lambda_2}{\lambda_1} \right) v_2 + \dots + \alpha_n \left(\frac{\lambda_n}{\lambda_1} \right) v_n \right]$$

après k itérations

$$\begin{aligned} Y_{k-1} &= AX_{k-1} \\ &= A \left[\frac{(\lambda_1)^{k-1}}{C_1 C_2 \dots C_{k-1}} \left[\alpha_1 v_1 + \alpha_2 \left(\frac{\lambda_2}{\lambda_1} \right)^{k-1} v_2 + \dots + \alpha_n \left(\frac{\lambda_n}{\lambda_1} \right)^{k-1} v_n \right] \right] \\ &= \left[\frac{(\lambda_1)^{k-1}}{C_1 C_2 \dots C_{k-1}} \left[\alpha_1 A v_1 + \alpha_2 \left(\frac{\lambda_2}{\lambda_1} \right)^{k-1} A v_2 + \dots + \alpha_n \left(\frac{\lambda_n}{\lambda_1} \right)^{k-1} A v_n \right] \right] \\ &= \left[\frac{(\lambda_1)^{k-1}}{C_1 C_2 \dots C_{k-1}} \left[\alpha_1 \lambda_1 v_1 + \alpha_2 \left(\frac{\lambda_2}{\lambda_1} \right)^{k-1} v_2 + \dots + \alpha_n \left(\frac{\lambda_n}{\lambda_1} \right)^{k-1} v_n \right] \right] \\ &= \left[\frac{(\lambda_1)^k}{C_1 C_2 \dots C_{k-1}} \left[\alpha_1 v_1 + \alpha_2 \left(\frac{\lambda_2}{\lambda_1} \right)^k v_2 + \dots + \alpha_n \left(\frac{\lambda_n}{\lambda_1} \right)^k v_n \right] \right] \end{aligned}$$

or $\frac{\lambda_j}{\lambda_1} < 1 \forall j \in [2, n]$ alors $\lim_{k \rightarrow +\infty} \alpha_j \left(\frac{\lambda_j}{\lambda_1} \right)^k = 0 \quad \forall j \in [2, n]$
ainsi

$$\lim_{k \rightarrow +\infty} X_k = \frac{\alpha_1 (\lambda_1)^k}{C_1 C_2 \dots C_k} v_1$$

comme X_k et v_1 sont normalisés, on a :

$$\lim_{k \rightarrow +\infty} \frac{\alpha_1 (\lambda_1)^k}{C_1 C_2 \dots C_k} = 1 \quad (\text{éq 3.3.1})$$

donc

$$\lim_{k \rightarrow +\infty} X_k = v_1$$

on écrit l'équation 3.3.1 pour l'étape (k-1), on aura :

$$\lim_{k \rightarrow +\infty} \frac{\alpha_1(1)^{k-1}}{C_1 C_2 \dots C_{k-1}} = 1 \quad (\text{eq 3.3.2})$$

en effectuant la division de l'eq 3.3.1 par l'eq 3.3.2, on aura :

$$\lim_{k \rightarrow +\infty} \frac{1}{C_k} = 1$$

$$\lim_{k \rightarrow +\infty} C_k = 1$$

Chapitre 6

Equations différentielles ordinaires

6.1 Rappel

On dit qu'une fonction g définie sur un intervalle $[a, b]$ est *lipschitzienne de constante L* si

$$|g(x) - g(x')| \leq L |x - x'| \quad \forall x, x' \in [a, b]$$

En pratique, la fonction g est de classe C^1 sur $[a, b]$ et cette inégalité est une conséquence de la formule des accroissements finis :

$$g(x) - g(x') = g'(c)(x - x') \quad c \in (x, x')$$

et on pose $L = \max_{x \in [a, b]} |g'(x)|$.

Lemme 6.1.1. 1. On montre par récurrence que si $u_{n+1} \leq u_n \alpha + \beta$ pour tout n alors $u_n \leq u_0 \alpha^n + \beta \frac{\alpha^n - 1}{\alpha - 1}$.

2. On montre grâce à une étude de fonction que si $X > 0$ alors $(1 + X)^n \leq e^{nX}$.

6.2 Problème de Cauchy

Soient f une fonction de $[a, b] \times \mathbf{R}$ dans \mathbf{R} continue et $\tau \in \mathbf{R}$. Le problème de Cauchy ou *problème de la condition initiale* est le suivant : existe-t-il une fonction $y(x)$ définie sur $[a, b]$ à valeurs dans \mathbf{R} dérivable sur $[a, b]$ et vérifiant :

$$\begin{cases} y(a) &= \tau \\ y(x) &= f(x, y(x)) \end{cases} \quad (6.2.1)$$

Théorème 6.2.1 (admis). On suppose que f est lipschitzienne par rapport à sa deuxième variable uniformément par rapport à sa première variable. Il existe donc $L > 0$ tel que

$$|f(x, z_1) - f(x, z_2)| \leq L |z_1 - z_2| \quad , \quad \forall x \in [a, b] , z_1, z_2 \in \mathbf{R}. \quad (6.2.2)$$

Alors le problème de Cauchy admet une solution unique.

La condition (6.2.2) sera satisfaite si f admet une dérivée partielle par rapport à sa deuxième variable qui soit bornée dans $[a, b] \times \mathbf{R}$. On obtient grâce au Théorème des Accroissements

Finis :

$$\begin{aligned} f(x, z_1) - f(x, z_2) &= \frac{\partial f}{\partial z}(x, z_c)(z_1 - z_2) \\ \Rightarrow |f(x, z_1) - f(x, z_2)| &\leq \max_{x \in [a, b]} \left| \frac{\partial f}{\partial z}(x, z_c) \right| |z_1 - z_2|. \end{aligned}$$

On présente dans les sections suivantes des méthodes d'approximation de la solution $y(x)$ de (6.2.1). Pour cela, on subdivise l'intervalle $[a, b]$ uniformément et on approche la fonction y en ces points.

Soit n un entier naturel non nul. On pose $x_i = a + ih$ avec $h = \frac{b-a}{n}$ et $i = 0..n$. On va calculer une approximation y_i de $y(x_i)$ pour $i = 0..n$ et estimer l'erreur d'approximation $e_i = y_i - y(x_i)$.

6.3 Méthode d'Euler

Partant de $x_0 = a$, on connaît $y_0 = y(x_0) = \tau$. L'idée de la méthode d'Euler est de considérer que la courbe y sur $[x_0, x_1]$ n'est pas très éloignée de sa tangente en x_0 : une estimation rationnelle de $y(x_1)$ est $y_1 = y_0 + hf(x_0, y_0)$. En effet, pour x_1 , l'équation de la tangente est $y'(x_0)(x - x_0) + y_0$. On estime alors que $y'(x_1)$ n'est pas très différent de $f(x_1, y_1)$. Sur $[x_1, x_2]$, on remplace la courbe par sa tangente approchée en x_1 ce qui donne l'estimation y_2 de $y(x_2)$: $y_2 = y_1 + hf(x_1, y_1)$ et on poursuit de la même manière.

Algorithme d'Euler :

Entrée : n, τ, a et b , et la fonction f .

Sortie : Une approximation de yx .

$x_0 := a$;

$y_0 := \tau$;

$h := \frac{b-a}{n}$;

Pour i allant de 0 à n **Faire**

$x_{i+1} := x_i + h$;

$y_{i+1} := y_i + hf(x_i, y_i)$;

Fin Pour ;

Imprimer(*La méthode d'Euler donne une approximation de y en (x_i, y_i) , $i = 0..n$*).

Fin.

Théorème 6.3.1. Si f est continue sur $[a, b] \times \mathbf{R}$, lipschitzienne par rapport à sa deuxième variable uniformément par rapport à sa première variable et si la solution y de l'EDO est de Classe C^2 sur $[a, b]$, alors :

$$|e_n| \leq \frac{(e^{L(b-a)} - 1)}{L} \frac{\max_{x \in [a, b]} |y''(x)|}{2} h.$$

Preuve : Voir la démonstration du théorème 6.5.1.

Définition 6.3.1. Une méthode de calcul numérique qui fournit une erreur $|e_n| \leq kh^p$ est dite d'ordre p . La méthode d'Euler est d'ordre 1.

Une méthode d'ordre 1 ne converge pas assez vite pour donner des résultats pratiques intéressants. On va étudier dans les deux sections suivantes deux méthodes d'ordre plus élevé.

6.4 Méthode de Taylor d'ordre 2

L'idée consiste à remplacer la courbe y sur un intervalle $[x_i, x_{i+1}]$ par une parabole (et non pas une droite comme dans la méthode d'Euler).

En utilisant la formule de Taylor sur y au voisinage de x_n , on obtient : $y(x_{n+1}) = y(x_n) + hy'(x_n) + \frac{h^2}{2}y''(x_n) + \frac{h^3}{6}y^{(3)}(c_n)$ avec $c_n \in [x_n, x_{n+1}]$. Comme $y'(x) = f(x, y(x))$, on a : $y''(x) = (\frac{\partial f}{\partial x} + f\frac{\partial f}{\partial z})(x, y(x))$. On pose $y_{n+1} = y_n + hf(x_n, y_n) + \frac{h^2}{2}(\frac{\partial f}{\partial x} + f\frac{\partial f}{\partial z})(x_n, y_n)$.

Algorithme de Taylor d'ordre 2 :

Entrée : n, τ, a et b , et la fonction f .

Sortie : Une approximation de yx .

$x_0 := a$;

$y_0 := \tau$;

$h := \frac{b-a}{n}$;

Pour i allant de 0 à n **Faire**

$x_{i+1} := x_i + h$;

$y_{i+1} := y_i + hf(x_i, y_i) + \frac{h^2}{2}(\frac{\partial f}{\partial x} + f\frac{\partial f}{\partial z})(x_i, y_i)$;

Fin Pour ;

Imprimer(*La méthode de Taylor d'ordre 2 donne une approximation de y en (x_i, y_i) , $i = 0..n$*).

Fin.

Théorème 6.4.1. Si f est de Classe C^2 sur $[a, b] \times \mathbf{R}$, lipshitzienne par rapport à sa deuxième variable uniformément par rapport à sa première variable (avec la constante L) et si $\frac{\partial f}{\partial x} + f\frac{\partial f}{\partial z}$ est lipshitzienne par rapport à sa deuxième variable uniformément par rapport à sa première variable (avec la constante L_1), alors pour $h \in [0, h_0]$ on a :

$$|e_n| \leq \frac{(e^{(b-a)(L+\frac{h_0}{2}L_1)} - 1)}{L} \frac{\max_{x \in [a, b]} |y^{(3)}(x)|}{6} h^2.$$

Preuve : Voir la démonstration du théorème 6.5.1.

6.5 Méthode de Runge-Kutta

Définition 6.5.1. Une méthode générale à un pas est définie par $y_{n+1} = y_n + h.\phi(x_n, y_n, h)$ où ϕ est continue sur $[a, b] \times \mathbf{R} \times [0, h_0]$ lipshitzienne par rapport à sa deuxième variable uniformément par rapport à la première et à la troisième de constante L_ϕ .

Théorème 6.5.1. Soit une méthode à un pas définie par $y_{n+1} = y_n + h.\phi(x_n, y_n, h)$ où ϕ est continue sur $[a, b] \times \mathbf{R} \times [0, h_0]$ lipshitzienne par rapport à sa deuxième variable uniformément par rapport à la première et à la troisième de constante L_ϕ et telle que

$$\frac{y(x+h) - y(x)}{h} - \phi(x, y(x), h) \leq kh^p$$

où y est solution de l'EDO $y'(x) = f(x, y(x))$ et f de Classe C^p sur $[a, b] \times \mathbf{R}$. Alors

$$|e_n| \leq K h^p.$$

Chapitre 7

Application aux réseaux de neurones

Un réseau de neurones artificiel est une structure composée d'entités (les neurones) capables de calcul interagissant entre eux. Il permet de traiter, par le biais de l'informatique, des problèmes de différentes natures que les outils classiques ont du mal à résoudre. Les explications et autres définitions sont tirés du polycopé de Marie Cottrel de l'Université de la Sorbonne¹.

7.1 Historique

Les recherches sur le cerveau ont fait d'énormes progrès, tant du point de vue de la compréhension du fonctionnement qu'à l'échelle cellulaire et membranaire. On sait désormais mesurer et enregistrer l'activité électrique d'un neurone ou d'un ensemble de neurones. Le cerveau humain dispose de l'ordre de 10^{11} avec plus de 10^{15} connexions. Il y a des propriétés collectives des neurones qui agissent en relations excitatrices ou inhibitrices. On distingue dans le cerveau des sous-ensembles, des connexions internes et externes, des entrées et des sorties, pour réaliser certaines tâches (parole, vision, émotion,...).

Les années 1940 sont connues pour l'apprentissage du codage et de la recherche opérationnelle : A. Turing avait mis en place la machine de Turing, premier ordinateur conçu pour déchiffrer le code Enigma. Depuis, le besoin de proposer des machines ou des modèles reproduisant des fonctions des cerveaux humains ont été développés. Il s'agit d'outils de calcul, de mémoire pour reproduire des comportements intelligents. Des chercheurs dans différents domaines ont travaillé sur l'intelligence artificielle.

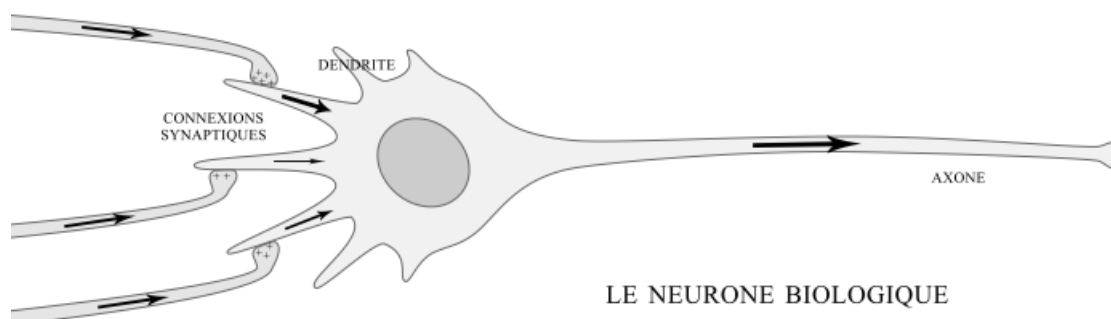
Le tableau suivant donne brièvement un historique des réseaux de neurones artificiels :

1940 - 1960 concepts	Mac Culloch & Pitts 43 Hebb Rosenblatt 58	modèle de neurone loi d'adaptation perceptron
1960 - 1980 transition et déclin	Widrow-Hoff 58 Minsky & Papert 69	adaline limites aux perceptron
1980 - ... renouveau	Hopfield 82 Kohonen 72, 82, 84 Rumelhart, Le Cun 86 Werbos 74	réseaux dynamiques auto-organisation rétro-propagation

1. Marie Cottrel, Les réseaux de neurones historique, méthodes et applications SAMOS-MATISSE, Université Paris 1, Sorbonne

7.2 Définitions

Un neurone biologique est une cellule nerveuse dont la fonction est de transmettre, dans certaines conditions, un signal électrique (0.01s). C'est comme un relai entre une couche de neurones et celle qui la suit. Le corps



Le corps d'un neurone est relié d'une part par un ensemble de *dendrites* (entrées du neurone) et d'autre part à un *axone*, qui est la partie étirée de la cellule. L'axone représente pour nous la sortie du neurone. Les charges électriques, reçues par les *connexions synaptiques* (en amont), s'accumulent dans le noyau du neurone jusqu'à atteindre un certain *seuil*. Au delà de ce seuil, la transmission du signal électrique se déclenche via son axone vers d'autres neurones (en aval). On remarque que les liaisons axone-dendrite entre deux neurones (connexions synaptiques) ne sont pas toutes de la même efficacité. Ainsi l'entrée associée à une certaine dendrite du neurone pourra avoir plus d'influence qu'une autre sur la valeur de sortie. On peut représenter la qualité de la liaison par un *poids*, sorte de coefficient s'appliquant au signal d'entrée. Le poids sera d'autant plus grand que la liaison est bonne. Un poids négatif aura tendance à inhiber une entrée, tandis qu'un poids positif viendra l'accentuer².

7.2.1 Neurone formel

Le neurone formel reproduit le neurone biologique. Il doit être capable de recevoir, en entrée, différentes informations provenant des neurones environnants. Il doit analyser et ajuster ces informations, de manière à envoyer en sortie une réponse. On assimile donc à un neurone un quadruplet (poids w_1, w_n , biais w_0 , fonction d'activation f , seuil θ) :

$$f(x_1, \dots, x_n) = \begin{cases} 1 & \text{si } w_0 + w_1x_1 + \dots + w_nx_n > \theta \\ -1 & \text{sinon.} \end{cases}$$


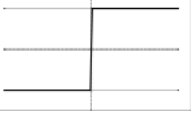
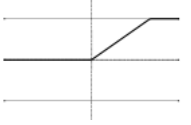
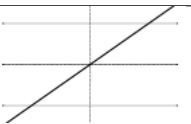
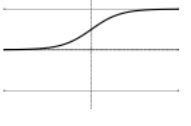
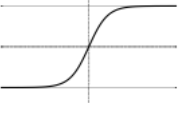
On utilise aussi une notation plus simple :

$$f(\vec{x}) = \begin{cases} 1 & \text{si } \vec{w} \cdot \vec{x} > \theta \\ -1 & \text{sinon.} \end{cases}$$

7.2.2 Fonctions d'activation

Il y a plusieurs exemples de fonctions d'activation utilisées pour les réseaux de neurones. Elle ressemblent le plus souvent à des fonctions *Tout ou rien* :

2. Texte et dessin d'Adrien Vergé, Réseaux de neurones artificiels, Lycée Michelet, TIPE 2009.

Nom	Valeur	Représentation
Seuil	$f(x) = 0$ si $x \leq 0$ et $f(x) = 1$ sinon	
Seuil symétrique	$f(x) = -1$ si $x < 0$ $f(0) = 0$ $f(x) = 1$ si $x > 0$	
Linéaire saturée	$f(x) = 0$ si $x < 0$ $f(x) = x$ pour $0 \leq x \leq 1$ $f(x) = 1$ si $x > 1$	
Linéaire	$f(x) = x$	
Sigmoïde	$f(x) = \frac{1}{1 + e^{-x}}$	
Tangente hyperbolique	$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$	

7.2.3 Perceptron multi-couches

On souhaite classer des images de 4x6 pixels dans deux catégories, selon des critères dépendant de la couleur et de la disposition de ces pixels. La première couche de notre réseau devra comporter 24 entrées, une entrée pour la valeur de chaque pixel. La couche de sortie fournira deux valeurs selon la catégorie. Le classement des images se fera selon les «affinités» : on placera l'image dans la catégorie avec laquelle l'affinité est grande.

Chaque neurone comportant plusieurs entrée et une seule sortie, on peut d'une façon générale, superposer dans une même couche, la couche 1 qui a s_1 neurones formels à e_1 entrées. On obtient alors un système admettant e_1 entrées et s_1 sorties. Puis, lorsqu'on concatène N couches connectées les unes à la suite des autres (le nombre d'entrées d'une couche étant égal au nombre de sorties de la couche précédente : $s_{n-1} = e_n$), on obtient un réseau de N couches de neurones.

7.3 Fonctionnement du réseau de neurones

Les applications des réseaux neuronaux vont de la reconnaissance vocale, en passant par la classification d'image ou de la prédiction financière. Les réseaux de neurones sont bien indiqués lorsqu'il s'agit de modéliser des structures complexes et des données irrégulières. En particulier quand les relations entre les variables sont non linéaires. Ils requièrent un pré-traitement sur

les données aberrantes auxquelles ils sont sensibles. Pour les PMC, les données doivent être normalisées. La particularité du réseau de neurones comme outil mathématique repose sur sa capacité d'apprentissage. Grâce à un processus d'entraînement, on peut améliorer les performances du réseau, c'est-à-dire qu'en réponse à une certaine entrée, il fournit la bonne sortie. Il s'agit de modifier les poids des liaisons en réaction aux entrées qu'on lui soumet, de manière à ce que la sortie corresponde au résultat désiré. Le réseau s'améliore en réduisant l'erreur qu'il commet à chaque simulation. On utilise pour cela un algorithme d'apprentissage. Il y a deux types d'algorithme d'apprentissage :

- Les apprentissages supervisés : on fournit un grand nombre de couples (x, c) où x est l'entrée et c est la sortie désirée. La correction s'effectue selon l'erreur obtenue pour chaque couple. Dans le cas d'un perceptron, la correction de l'erreur se fait comme suit :

$$w_i \leftarrow w_i + \Delta w_i$$

avec

$$\Delta w_i = \eta(t - c)x_i$$

où

$t = f(\vec{x})$ est la valeur cible

c est la sortie désirée du perceptron

η une constante de petite taille (e.g., .1) appelée *taux d'apprentissage*

On prouve la convergence lorsque les données d'apprentissage sont linéairement séparables et η suffisamment petit. On parle en général d'apprentissage par l'exemple.

- Les apprentissages non supervisés : ne connaissant pas la sortie désirée pour une entrée donnée, on n'a pas de signal d'erreur le réseau apprendra par lui-même à classer des entrées similaires en trouvant des points communs aux stimuli appliquées.

La correction de l'erreur en réaction à des stimuli en entrée constitue le processus d'apprentissage

To understand, consider simpler *linear unit*, where

$$o = w_0 + w_1x_1 + \dots + w_nx_n$$

Let's learn w_i 's that minimize the squared error

$$E[\vec{w}] \equiv \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2$$

Where D is set of training examples.

$$\nabla E[\vec{w}] \equiv \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \dots, \frac{\partial E}{\partial w_n} \right]$$

Training rule :

$$\Delta \vec{w} = -\eta \nabla E[\vec{w}]$$

i.e.,

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i}$$

$$\begin{aligned}
\frac{\partial E}{\partial w_i} &= \frac{\partial}{\partial w_i} \frac{1}{2} \sum_d (t_d - o_d)^2 \\
&= \frac{1}{2} \sum_d \frac{\partial}{\partial w_i} (t_d - o_d)^2 \\
&= \frac{1}{2} \sum_d 2(t_d - o_d) \frac{\partial}{\partial w_i} (t_d - o_d) \\
&= \sum_d (t_d - o_d) \frac{\partial}{\partial w_i} (t_d - \vec{w} \cdot \vec{x}_d) \\
\frac{\partial E}{\partial w_i} &= \sum_d (t_d - o_d) (-x_{i,d})
\end{aligned}$$

Gradient-Descent(*training_examples*, η)

Each training example is a pair of the form $\langle \vec{x}, t \rangle$, where \vec{x} is the vector of input values, and t is the target output value. η is the learning rate (e.g., .05).

- Initialize each w_i to some small random value
- Until the termination condition is met, Do
 - Initialize each Δw_i to zero.
 - For each $\langle \vec{x}, t \rangle$ in *training_examples*, Do
 - Input the instance \vec{x} to the unit and compute the output o
 - For each linear unit weight w_i , Do

$$\Delta w_i \leftarrow \Delta w_i + \eta(t - o)x_i$$

- For each linear unit weight w_i , Do

$$w_i \leftarrow w_i + \Delta w_i$$

$\sigma(x)$ is the sigmoid function

$$\frac{1}{1 + e^{-x}}$$

Nice property : $\frac{d\sigma(x)}{dx} = \sigma(x)(1 - \sigma(x))$

L'erreur :

$$\begin{aligned}
\frac{\partial E}{\partial w_i} &= \frac{\partial}{\partial w_i} \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2 \\
&= \frac{1}{2} \sum_d \frac{\partial}{\partial w_i} (t_d - o_d)^2 \\
&= \frac{1}{2} \sum_d 2(t_d - o_d) \frac{\partial}{\partial w_i} (t_d - o_d) \\
&= \sum_d (t_d - o_d) \left(-\frac{\partial o_d}{\partial w_i} \right) \\
&= - \sum_d (t_d - o_d) \frac{\partial o_d}{\partial net_d} \frac{\partial net_d}{\partial w_i}
\end{aligned}$$

But we know :

$$\frac{\partial o_d}{\partial net_d} = \frac{\partial \sigma(net_d)}{\partial net_d} = o_d(1 - o_d)$$

$$\frac{\partial net_d}{\partial w_i} = \frac{\partial (\vec{w} \cdot \vec{x}_d)}{\partial w_i} = x_{i,d}$$

So :

$$\frac{\partial E}{\partial w_i} = - \sum_{d \in D} (t_d - o_d) o_d (1 - o_d) x_{i,d}$$

Backpropagation Algorithm :

Initialize all weights to small random numbers.

Until satisfied, Do

— For each training example, Do

1. Input the training example to the network and compute the network outputs
2. For each output unit k

$$\delta_k \leftarrow o_k(1 - o_k)(t_k - o_k)$$

3. For each hidden unit h

$$\delta_h \leftarrow o_h(1 - o_h) \sum_{k \in \text{outputs}} w_{h,k} \delta_k$$

4. Update each network weight $w_{i,j}$

$$w_{i,j} \leftarrow w_{i,j} + \Delta w_{i,j}$$

where

$$\Delta w_{i,j} = \eta \delta_j x_{i,j}$$

La faiblesse des réseaux de neurones réside dans le caractère non explicite des résultats. La convergence n'est pas forcément vers une solution optimale globale. Le choix peu guidé de la structure du PMC (nombre de couches, nombre de neurones par couches). Sensible à un très grand nombre de variables discriminantes.

Input	Output
10000000 →	10000000
01000000 →	01000000
00100000 →	00100000
00010000 →	00010000
00001000 →	00001000
00000100 →	00000100
00000010 →	00000010
00000001 →	00000001

Learned hidden layer representation :

Input		Hidden Values				Output
10000000	→	.89	.04	.08	→	10000000
01000000	→	.01	.11	.88	→	01000000
00100000	→	.01	.97	.27	→	00100000
00010000	→	.99	.97	.71	→	00010000
00001000	→	.03	.05	.02	→	00001000
00000100	→	.22	.99	.99	→	00000100
00000010	→	.80	.01	.98	→	00000010
00000001	→	.60	.94	.01	→	00000001

7.4 Statistique et PMC

Supposons que X_1, \dots, X_n soient des valeurs observées et que l'on cherche $\phi(X_1, \dots, X_n)$ telle que

$$E(X_{n+1}|X_1, \dots, X_n) = \phi(X_1, \dots, X_n) .$$

Toute fonction $\phi(x_1, \dots, x_n)$ suffisamment régulière peut être approchée par une fonction de la forme :

$$g\left(\sum_{i=1}^n a_i x_i + b_i\right) .$$

Ce qu'il faut chercher est l'estimation statistique $\hat{g}(X_1, \dots, X_n)$ de g s'exprimant sous la forme :

$$\hat{g} = g\left(\sum_{i=1}^n \hat{a}_i x_i + \hat{b}_i\right)$$

où, pour $i \in [1, n]$, les valeurs \hat{a}_i et \hat{b}_i sont respectivement des valeurs approchées des a_i et b_i . (Voir [2] et [3] pour une vitesse d'approximation).

Ce type de problème classique de la statistique est résolu dans des cas très généraux. L'intérêt des réseaux neuronaux intervient dans les cas de grandes dimensions.

Nous avons x_1, \dots, x_n des variables observées et nous cherchons x_{n+1} . Pour $i = 1, \dots, n+1$, notons X_i la variable aléatoire de la i -ième valeur. Posons $X = (X_1, \dots, X_n)$ et $Y = (X_{n+1})$. Pour approcher Y , nous cherchons à construire un réseau qui approxime au mieux l'espérance $E[Y|X]$ par une fonction $f_W(X)$, où W est une matrice de poids synaptiques.

La couche d'entrée comportera n neurones (pour X) et la couche de sortie 1 neurone (pour $f_W(X)$).

Qui dit estimer, sous-entend choisir une fonction d'erreur $\pi(Y, f_W(X))$ entre la sortie obtenue $f_W(x)$ avec $x = (x_1, \dots, x_n)$ et celle désirée $Y = (x_{n+1})$.

--

Une fois cette fonction d'erreur choisie, nous désirons qu'elle soit aussi petite que possible. Donc nous cherchons f_W qui minimise au mieux l'espérance $E[\pi(Y, f_W(X))]$ de la fonction d'erreur π .

La question est de savoir si approcher Y permet de trouver $E[Y|X]$. Cela dépend de la fonction d'erreur choisie.

Exemple 2.1. Choisissons $\pi(Y, f_W(X)) = (Y - f_W(X))^2$. Avec cette fonction d'erreur, nous calculons de manière classique

$$E[(Y - f_W(X))^2|X] = E[(Y - E[Y|X])^2|X] + E[(E[Y|X] - f_W(X))^2|X].$$

(l'erreur totale est la somme de l'erreur de prédiction et de l'erreur statistique de prédiction). Donc minimiser l'erreur entre la sortie du réseau $f_W(X)$ et la valeur désirée Y revient à minimiser celle entre $f_W(X)$ et l'espérance $E[Y|X]$, ce qu'on cherche à estimer.

D'autres fonctions d'erreur entre la sortie du réseau $f_W(X)$ et la valeur désirée Y partagent cette propriété d'*estimation optimale*. Par exemple, la *Cross Entropy* souvent employée dans les PMC :

$$Y \log(f_W(X)) + (1 - Y) \log(1 - f_W(X)).$$

Comme nous l'avons étudié dans le PMC, nous savons que les poids optimaux sont obtenus en minimisant l'espérance d'erreur entre les calculs du réseau et les valeurs désirées obtenues à partir d'un corpus.

7.5 Logiciels utilisés

7.5.1 nnet de R pour le perceptron multi-couches

Le logiciel R est un logiciel de statistiques libre. Il est constitué de packages permettant de réaliser des études statistiques. Le package `nnet` est utilisé pour le Perceptron Multi Couches. Nous allons le comparer avec le modèle polynomial d'interpolation. Pour cela, nous considérons l'exemple introductif donné par Annick Valibouze à l'ISUP lors du cours sur les Réseaux de Neurones Artificiels sous le Logiciel R ³.

Nous considérons une fonction $\sin(x)$ bruitée et nous comparons avec le modèle polynomial (interpolation) ainsi que le PMC.

```
library(nnet)
library(MASS)
# Les donn'ees
x <- sort(10*runif(50))
y <- sin(x)+0.2*rnorm(x)
data <- data.frame(x,y)
# Construction du PMC linout=TRUE pour \^etre entre 0 et 1
nn <- nnet(x,y,size=4,maxit=100,linout=TRUE)
plot(x,y,col="blue",pch=16)
# Le PMC en mode de fonctionnement
lines(x1,predict(nn,data.frame(x=x1)),col="red")
# La courbe sin(x)
x1<-seq(0,10,by=0.01)
lines(x1,sin(x1),col="green")
```

3. Troisième année ISUP, Filière GRIE, Master 2 De Statistique, Major Data Science

```
# Comparaison avec le mod\ele polynomial
modelPoly <- lm(y~x + I(x^2) + I(x^3) + I(x^4))
lines(x,predict(modelPoly), col="orange")
# L'egende
legend(3,1.3,c("donnees simulees avec bruit","fonction sin(x)","PMC",
"modele polynomial"),lty=c(0,1,1,1),pch=c(16,-1,-1-1),
col=c("blue","green","red","orange"))
x #Les 50 valeurs de x
summary(nn) # le réseau de neurones
```

Si vous disposez d'une licence SAS, alors il y a moyen de manipuler des réseaux de neurones artificiels via le package Proc Neural.

7.5.2 SageMath, Python

7.6 Conclusion

Un réseau de neurones est avant tout un outil de traitement de l'information. En le formant via l'apprentissage, il peut être capable d'effectuer des tâches de reconnaissance, de classification, d'approximation, de prévision, etc. Nous utilisons par exemple les réseaux de neurones pour réduire l'écho dans les communications téléphoniques outre-mer (réseau Adaline). Ils peuvent aussi servir à éliminer du bruit, dans les prévisions météorologiques, dans la compression de données, dans la modélisation du marché monétaire, dans le développement de l'Intelligence Artificielle, etc. Il y a aussi moyen de leur apprendre à reconnaître des faces, le son de la voix et autres lésions cancéreuses. Voir par exemple le projet NetTalk qui est un réseau de 309 neurones relié en entrée à un scanner et branché en sortie à un haut parleur. NetTalk a appris à lire un texte à haute voix après avoir appris à reconnaître 50000 mots d'un dictionnaire toute une nuit. Une fois entraîné, le réseau est capable de lire de nouveaux mots qu'il n'a jamais vu auparavant. De nos jours, ce sont de vrais neurones animales qui sont utilisés : en Angleterre ce sont des neurones de rats, mises en culture et connectés par des électrodes au système moteur d'une machine, qui apprend à éviter les murs à force de se cogner.

Annexe A

Travaux Pratiques sur SAGEmath

Les codes suivants représentent des exercices corrigés ou des Projets de Fin d'années que nous avons encadrés en Analyse Numérique. Il s'agit de problèmes liés à l'interpolation polynomiale, à la recherche de solutions d'équations non linéaires et aux problèmes d'Analyse Numérique matricielle.

A.1 Interpolation

TP2 : Estimation de l'erreur

Théorème : Soient f une fonction de classe C^{n+1} sur un intervalle $[a, b]$ et P le polynôme de Lagrange de f en les points $x_0 < x_1 < \dots < x_n \in [a, b]$. Alors

$$f(x) - P(x) = \frac{(x - x_0)(x - x_1) \dots (x - x_n)}{(n + 1)!} f^{(n+1)}(\zeta) \quad (\text{A.1.1})$$

où $a \leq \min(x, x_0) < \zeta < \max(x, x_n) \leq b$.

Démonstration : La relation est triviale pour $x = x_i \forall i = 0..n$. Soit $x \neq x_i$, pour tout $i = 0..n$. On pose $k(x) = \frac{f(x) - P(x)}{(x - x_0)(x - x_1) \dots (x - x_n)}$ et $w(t)$ un polynôme qui s'annule en x_0, x_1, \dots, x_n , et x définit par : $w(t) = f(t) - P(t) - (t - x_0)(t - x_1) \dots (t - x_n)k(x)$. Il s'annule en $(n + 2)$ points. En appliquant le théorème de Rolle, on montre que $w^{(n+1)}(t)$ s'annule en au moins un point ζ :

$$w^{(n+1)}(\zeta) = f^{(n+1)}(\zeta) - (n + 1)! k(x) = 0 \quad \text{et} \quad k(x) = \frac{f^{(n+1)}(\zeta)}{(n + 1)!} \quad .$$

Cette quantité ne tend pas nécessairement vers 0 car les dérivées $f^{(n+1)}$ de f peuvent grandir très vite lorsque n croît. Dans l'exemple suivant, nous présentons deux cas sur la convergence de l'interpolation de Lagrange.

Exemple : Soit $f(x) = \sin(x)$, $|f^k(x)| \leq 1$: l'interpolé P converge vers f quelque soit le nombre de points d'interpolation et leur emplacement. Par contre, pour $f(x) = \frac{1}{1+14x^2}$ sur $[-1, 1]$ et bien que f soit indéfiniment continûment dérivable sur $[-1, 1]$, les grandeurs

$$\max_{-1 \leq x \leq 1} |f^k(x)| \quad , \quad k = 1, 2, 3, \dots$$

explosent très rapidement et ceci ne nous assure plus la convergence de l'interpolation.

Le choix des points x_i apporte une amélioration sensible de l'interpolation : Lorsque $a \leq x \leq b$, on choisit les abscisses d'interpolation

$$x_i = \frac{a+b}{2} + \frac{b-a}{2} \cos\left(\frac{2i+1}{2(n+1)}\pi\right)$$

pour $i = 0..n$, on obtient :

$$|f(x) - P(x)| \leq \frac{(b-a)^{n+1}}{(n+1)!2^{2n+1}} \max_{x \in [a,b]} |f^{(n+1)}(x)|.$$

Les points $x_i = \frac{a+b}{2} + \frac{b-a}{2} \cos\left(\frac{2i+1}{2(n+1)}\pi\right)$ pour $i = 0..n$ sont calculés à partir des zéros des polynômes de Tchebychev ($T_n(x)$ vérifiant $T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x)$).

Exercice à faire sur SAGEmath :

1. Soit $f(x) = \sin(x)$ et une subdivision de l'intervalle $[-1, 1]$ en $n = 5$ sous-intervalles équidistants. Posons $x_0 = -1$, $x_1 = -0.5$, $x_2 = 0$, $x_3 = 0.5$ et $x_4 = 1$. Implémenter l'algorithme de Lagrange se basant sur les polynômes d'interpolation de base. On prendra $f_i = f(x_i)$ pour $i = 0..n$. Dessiner dans un même graphique la fonction f , le polynôme P ainsi que les points (x_i, f_i) . Reprendre les calculs pour $n = 13$.
2. Refaire le même travail avec la fonction $f(x) = \frac{1}{1+14x^2}$, puis avec $x_i = \cos\left(\frac{2i+1}{2n+2}\pi\right)$ pour $i = 0..n$ et $n = 5$ puis $n = 13$.

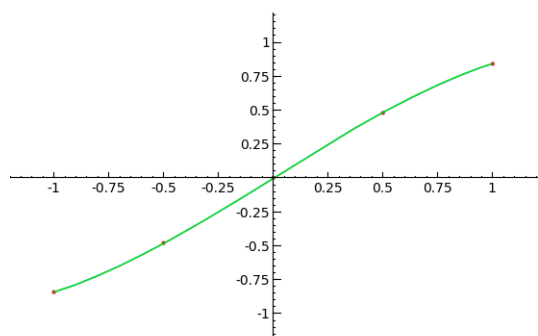
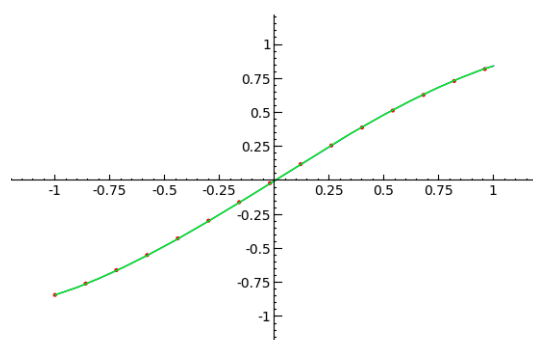
Correction sur SAGEmath :

1. Le code sous SAGEmath est donné ci-dessous :

```
SAGEmath: from pylab import arange
SAGEmath: xdata=arange(-1.0,1.1,0.5)
SAGEmath: f=sin
SAGEmath: P=lagrange(f, xdata)
SAGEmath: R=plot(f, -1,1, rgbcolor=(0,2,1))
SAGEmath: Q=plot(P, -1,1, rgbcolor=(0,1,2))
SAGEmath: n=len(xdata)-1
SAGEmath: s=Graphics()
SAGEmath: for i in range(n+1):
...     xi = xdata[i]
...     fi = f(xdata[i])
...     s = s + point((xi,fi), rgbcolor = (1,0,0))
SAGEmath: R+Q+s
```

Les graphiques suivants donnent la fonction \sin et son polynôme de Lagrange sur $n = 5$ puis sur $n = 13$ points :

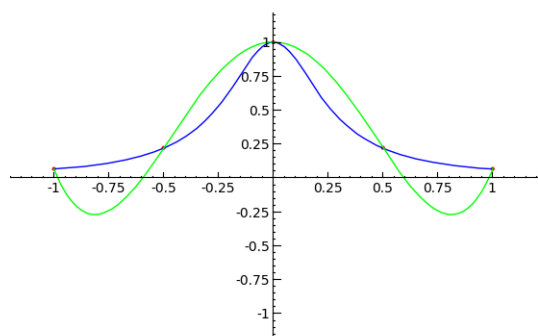
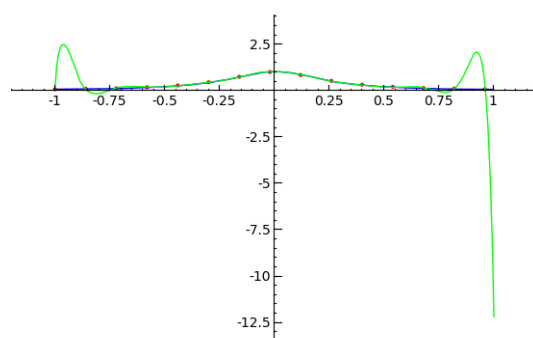
2. Si on calcule le polynôme de Lagrange sur la fonction $f = \frac{1}{1+14x^2}$, nous obtenons :

FIGURE A.1 – *Interpolation pour $n = 5$* FIGURE A.2 – *Interpolation pour $n = 13$*

```

SAGEmath: from pylab import arange
SAGEmath: xdata=arange(-1.0,1.1,0.5)
SAGEmath: f=1/(1+14*x^2)
SAGEmath: P=lagrange(f, xdata)
SAGEmath: R=plot(f, -1,1, rgbcolor=(0,2,1))
SAGEmath: Q=plot(P, -1,1, rgbcolor=(0,1,2))
SAGEmath: n=len(xdata)-1
SAGEmath: s=Graphics()
SAGEmath: for i in range(n+1):
...     xi = xdata[i]
...     fi = f(xdata[i])
...     s = s + point((xi,fi), rgbcolor = (1,0,0))
...
SAGEmath: R+Q+s

```

FIGURE A.3 – *Interpolation pour $n = 5$* FIGURE A.4 – *Interpolation pour $n = 13$*

Enfin, si on considère à la place des abscisses x_i équidistants, les zéros de Tchebychev, nous obtenons la figure 5 pour $n = 5$ points et ensuite la figure 6 pour $n = 13$:

```

SAGEmath: a=-1; b=1; n=5;
SAGEmath: ydata=[simplify((a+b)/2+(b-a)/2*cos((2*k+1)*3.14/(2*n+2)))
for k in range(n+1)]

```

```

SAGEmath: print(ydata)
[0.965960168538399, 0.707388269167200, 0.259459981914882, -0.257921542147459,
-0.706261644820005, -0.965546938710468]
SAGEmath: f=1/(1+14*x^2)
SAGEmath: P=lagrange(f, ydata)
SAGEmath: A=plot(f, -1,1, rgbcolor=(0,2,1))
SAGEmath: B=plot(P, -1,1, rgbcolor=(1,0,0))
SAGEmath: A+B

```

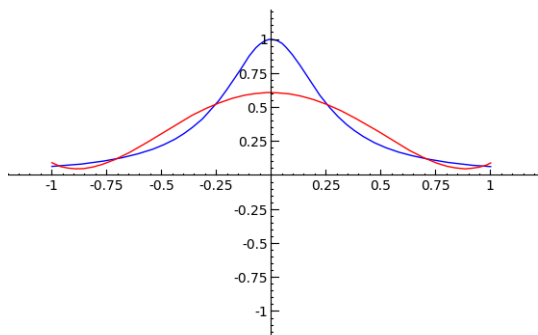


FIGURE A.5 – Interpolation sur les zéros de Tchebychev pour $n = 5$

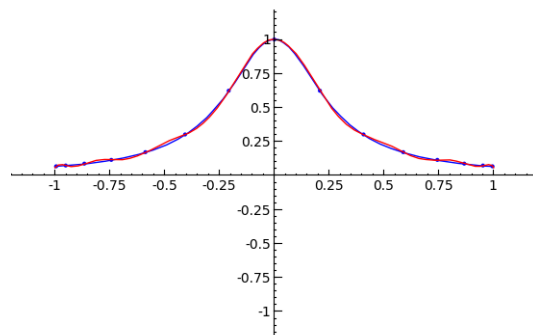


FIGURE A.6 – Interpolation sur les zéros de Tchebychev pour $n = 13$

```

SAGEmath: a=-1; b=1; n=13;
SAGEmath: ydata=[simplify((a+b)/2+(b-a)/2*cos((2*k+1)*3.14/(2*n+2)))
for k in range(n+1)]
SAGEmath: print(ydata)
[0.993718576879459, 0.943939675865892, 0.846875476196380, 0.707388269167200,
0.532465465533183, 0.330869571228815, 0.112699242252689, -0.111116592961247,
-0.329366202474389, -0.531116686408463, -0.706261644820005, -0.846027443250886,
-0.943412715311215, -0.993539086045688]
SAGEmath: f=1/(1+14*x^2)
SAGEmath: P=lagrange(f, ydata)
SAGEmath: A=plot(f, -1,1, rgbcolor=(0,2,1))
SAGEmath: B=plot(P, -1,1, rgbcolor=(1,0,0))
SAGEmath: A+B

```

A.2 Résolution d'équations

Considérons le polynôme $f(x) = x^4 + 6x^2 - 60x + 36$ sur $[0, 4]$. Il s'agira d'appliquer la méthode du point fixe pour les fonctions suivantes :

— $g_1(x) = \frac{x^4 + 6x^2 + 36}{60}$

— $g_2(x) = \frac{-36}{x^3 + 6x - 60}$

— $g_3(x) = (-6x^2 + 60x - 36)^{\frac{1}{4}}$

1. Ecrire une procédure qui prend en entrée g , x_0 et N et qui calcule le N ème terme de la liste $x_{n+1} = g(x_n)$.
2. Vérifier ensuite la convergence de la méthode du point fixe pour chacun des g_i sur $[0, 4]$. Pour cela, il faut chercher à satisfaire toutes les conditions du théorème de convergence de la méthode du point fixe :
3. Ecrire un script qui permet d'accélérer la convergence de g_3 .

A.3 Méthode d'Euler

Partant de $x_0 = a$, on connaît $y_0 = y(x_0) = \tau$. L'idée de la méthode d'Euler est de considérer que la courbe y sur $[x_0, x_1]$ n'est pas très éloignée de sa tangente en x_0 : une estimation rationnelle de $y(x_1)$ est $y_1 = y_0 + hf(x_0, y_0)$. En effet, pour x_1 , l'équation de la tangente est $y'(x_0)(x - x_0) + y_0$. On estime alors que $y'(x_1)$ n'est pas très différent de $f(x_1, y_1)$. Sur $[x_1, x_2]$, on remplace la courbe par sa tangente approchée en x_1 ce qui donne l'estimation y_2 de $y(x_2)$: $y_2 = y_1 + hf(x_1, y_1)$ et on poursuit de la même manière. La méthode d'Euler est d'ordre 1. Une telle méthode ne converge pas assez vite pour donner des résultats pratiques intéressants. On va étudier dans les deux sections suivantes deux méthodes d'ordre plus élevé. La méthode d'Euler est implémentée dans **SAGEmath** comme suit :

```
def euler(f,n,tau,a,b):
    h = (b-a)/n
    x = [x0]
    y = tau
    for i in range(n):
        x += [x[i]+h]
        y += [y[i]+h*f(x[i],y[i])]
    return x,y
```

A.4 Analyse Numérique Matricielle : SVD

«L'imagerie numérique a longtemps été un exemple d'application de la SVD. Maintenant, il existe des moyens plus élaborés et plus efficaces. Reste que la SVD est un outils intéressant pour le transfert d'image.

Une photo en noir et blanc peut être assimilée à une matrice A dont les éléments correspondent à des niveaux de gris pour les différents pixels qui constituent l'image. Chaque $a_{i,j}$ donne le niveau de gris du pixel (i, j) de l'image : $a_{i,j} \in [0, 1]$ avec par exemple $a_{i,j} = 0$ pour un pixel (i, j) blanc et $a_{i,j} = 1$ pour un pixel (i, j) noir».

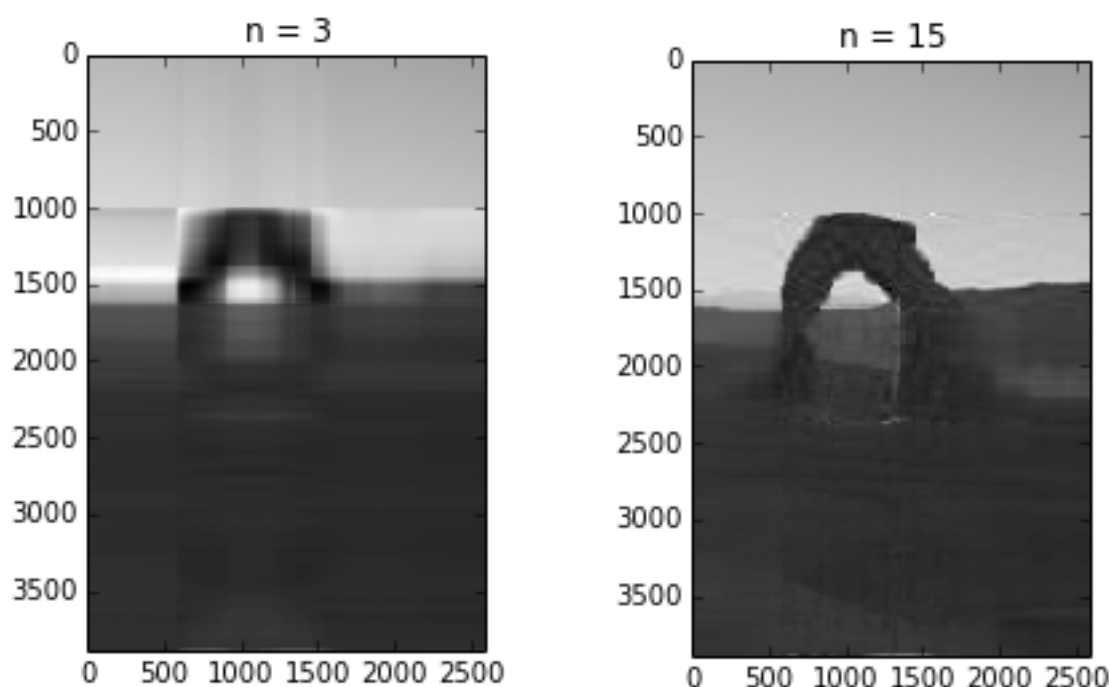
Ci-dessous une image en noir et blanc de taille 2500 X 4000 pixels dont la matrice A représentative est générée par un logiciel de lecture de l'image (via un scanner ou un appareil photo numérique). le résultat renvoyé correspond à une matrice A de rang r . la matrice admet donc r valeurs singulières non nulles. Soit σ_1 la valeur maximale de ces valeurs singulières et soient $\sigma_1, \sigma_2, \dots, \sigma_r$ ces valeurs singulières triées par ordre décroissant. On sait alors qu'il existe une matrice U de taille 2500, une matrice V de taille 4000 telsque $A = U \text{diag}(\sigma_1, \sigma_2, \dots, \sigma_r) V^t$. En désignant par u_i les vecteurs colonnes de U , et par v_i les vecteurs colonnes de V , on obtient

que $A = \sum_{i=1}^r \sigma_i u_i v_i^t$.

La connaissance de A elle-même correspond à la connaissance des 2500×4000 coefficients de cette matrice. Si on regarde l'expression de droite, la connaissance de A est donnée par les r valeurs singulières de A , plus r vecteurs de taille 2500 et r vecteurs de taille 4000.

Mais si on regarde de plus près les valeurs singulières non nulles $\sigma_1, \sigma_2, \dots, \sigma_r$, on constate que nombreuses d'entre-elles sont très petite (voir négligeable) devant les autres. On peut alors décider que $\frac{\sigma_i}{\sigma_1} > \epsilon$, où ϵ est un filtre choisi par l'utilisateur, par exemple $\epsilon = 10^{-3}$. Dans ce cas, σ_i est considérée comme nulle : on réduit ainsi le nombre de valeurs singulières à conserver dans l'expression de A selon la décomposition SVD.

Notons $\sigma_1, \sigma_2, \dots, \sigma_k$ les valeurs singulières conservées. Dans l'exemple proposé ci-dessous, avec $k = 3$, l'image est floue mais on constate facilement que c'est un paysage. Pour $k = 13$ l'image est déjà ressemblante à l'originale. pour $k = 80$ on peut dire que l'image obtenue est identique à l'originale. La connaissance de A est alors donnée par la connaissance de 80 valeurs singulières, plus 80 vecteurs de taille 2500, plus de 80 valeurs de taille 4000. Soit 520 080 au lieu de 10 000 000.



A.4.1 A faire sur Python :

1. Donnez la décomposition en valeurs singulières de la matrice $A = 1, 1, 1, -1$
2. Trouvez la commande qui permet d'effectuer la décomposition en valeurs singulières de A sur SAGEmath
3. Téléchargez une image sous format .png

4. Calculez la SVD de cette image en utilisant les logiciels numpy et pylab.

A.4.2 Correction

```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
import time

from PIL import Image

img = Image.open('input/img_6847.jpg') #ici l'image chargée est input/img_6847.jpg
imggray = img.convert('LA')
plt.figure(figsize=(9, 6))
plt.imshow(imggray);

#On la transforme en une matrice numpy
imgmat = np.array(list(imggray.getdata(band=0)), float)
imgmat.shape = (imggray.size[1], imggray.size[0])
imgmat = np.matrix(imgmat)
plt.figure(figsize=(9,6))
plt.imshow(imgmat, cmap='gray');

#Le calcul de la SVD se fait comme suit :
U, sigma, V = np.linalg.svd(imgmat)

#Ici la reconstruction de l'image
reconstimg = np.matrix(U[:, :1]) * np.diag(sigma[:1]) * np.matrix(V[:1, :])
plt.imshow(reconstimg, cmap='gray');

for i in xrange(2, 4):
    reconstimg = np.matrix(U[:, :i]) * np.diag(sigma[:i]) * np.matrix(V[:i, :])
    plt.imshow(reconstimg, cmap='gray')
    title = "n = %s" % i
    plt.title(title)
    plt.show()

for i in xrange(5, 51, 5):
    reconstimg = np.matrix(U[:, :i]) * np.diag(sigma[:i]) * np.matrix(V[:i, :])
    plt.imshow(reconstimg, cmap='gray')
    title = "n = %s" % i
    plt.title(title)
    plt.show()
```

A.4.3 Un code plus interactif

```

import numpy
import pylab
U, s, V = numpy.linalg.svd(matrix(2,2,[1,1,1,-1]))
U, s, V

import pylab
import numpy

# Read in a png image as a *numpy array* : pylab.imread
# Convert to greyscale : numpy.mean(...,2)
#A_image = numpy.mean(pylab.imread(DATA + 'coffee.png'), 2)
#A_image = numpy.mean(pylab.imread(DATA + 'CapturFiles_11.png'), 2)
A_image = numpy.mean(pylab.imread(DATA + 'chelsea.png'), 2)

# Use numpy to compute the singular value decomposition of the image
u,s,v = numpy.linalg.svd(A_image)
S = numpy.zeros(A_image.shape)
S[:len(s),:len(s)] = numpy.diag(s)
n = A_image.shape[0]

# The SVD is a factorization of A_image as a product: U * S * V
# We verify that this is indeed the case.
# WARNING u * S is *NOT* matrix multiplication in numpy,
# it's componentwise multiplication. Instead use "numpy.dot" to multiply.

@interact
def svd_image(i = ( "Eigenvalues(quality)",(20,(1..A_image.shape[0])))):

    # Thus the following expression involves only the first i eigenvalues,
    # and the first i rows of v and the first i columns of u.
    # Thus we use i*m + i*n + i = i*(m+n+1) storage, compared to the
    # original image that uses m*n storage.
    A_approx = numpy.dot(numpy.dot(u[:,:,:i], S[:i,:i]), v[:i,:])
    p = show(graphics_array([matrix_plot(A_image), matrix_plot(A_approx)]), axes=False)

    # Of course, nothing is ever exactly right with floating point matrices...
    # so we check for closeness.
    print numpy.allclose(A_approx, A_image, 1e-1, 1e-1)

```

Il s'agit d'abord de tester la notion de problème bien ou mal posé par la voie des matrices bien ou mal conditionnées. Ensuite, il faudra implémenter la factorisation LU et la factorisation de Cholesky et de comparer ses deux factorisations.

A.5 Matrice bien ou mal conditionnée

Considérons les deux systèmes suivants : $Ax = b$ avec respectivement

$$A = \begin{pmatrix} 4.218613 & 6.327917 \\ 3.141592 & 4.712390 \end{pmatrix}, b = \begin{pmatrix} 10.546530 \\ 7.853982 \end{pmatrix}$$

et

$$A = \begin{pmatrix} 4.218611 & 6.32791 \\ 3.141594 & 4.712390 \end{pmatrix}, b = \begin{pmatrix} 10.546530 \\ 7.853980 \end{pmatrix}$$

1. Dessinez les deux droites associées au premier système. Vous remarquerez que le premier système est formé de deux droites presque parallèles. Ainsi, résoudre le système revient à chercher l'intersection de ces deux droites presque parallèles.
2. Résoudre ces deux systèmes et comparer les solutions. Que remarquez-vous ? Il est clair que si on perturbe le deuxième système, l'intersection de ces deux nouvelles droites presque parallèles sera fortement modifiée.
3. Calculer le nombre de conditionnement $\|A\| \cdot \|A^{-1}\|$ de chacun de ces systèmes. Le conditionnement étant loin de 1, ceci explique les différences des solutions. On parle de problème mal conditionné.

A.6 Factorisation LU

On considère la matrice

$$A = \begin{pmatrix} 4 & 8 & 12 \\ 3 & 8 & 13 \\ 2 & 9 & 18 \end{pmatrix}$$

1. Ecrire l'algorithme de factorisation LU.
2. Comparer le résultat avec celui de la fonction python qui permet une factorisation LU.

A.7 Factorisation de Cholesky

On considère une matrice A tridiagonale, symétrique et définie-positive d'ordre N . On cherche à compléter un algorithme permettant de résoudre un système linéaire $Ax = b$ par la méthode de Cholesky. Notons a_i les éléments de la diagonale de A et c_j les éléments de la sur-diagonale et sous-diagonale de A pour $i = 1, \dots, N$ et $j = 1, \dots, N-1$:

$$A = \begin{pmatrix} a_1 & c_1 & & & & \\ c_1 & a_2 & c_2 & & & \\ & c_2 & \dots & \dots & & \\ & & & \dots & c_{N-2} & \\ & & & c_{N-2} & a_{N-1} & c_{N-1} \\ & & & & c_{N-1} & a_N \end{pmatrix}$$

Soit R une matrice triangulaire inférieure telle que $A = RR^t$. Alors R aura la forme suivante :

$$R = \begin{pmatrix} l_1 & & & & \\ m_1 & l_2 & & & \\ & & \cdots & \cdots & \\ & & & \cdots & c_{N-2} \\ & & & \cdots & l_{N-1} \\ & & & & m_{N-1} & l_N \end{pmatrix}$$

1. Trouver le l et m en fonction de a et de c .
2. Compléter l'algorithme de décomposition de Cholesky suivant (l est mémorisé dans a et m est mémorisé dans c) :

$$\left\{ \begin{array}{l} a(1) = \sqrt{a(1)} \\ \text{Pour } i \text{ allant de } 1 \text{ à } N-1 \text{ Faire} \\ \quad c(i) = ?/ ? \\ \quad a(i+1) = \sqrt{? - ?*?} \end{array} \right.$$

3. La résolution de $Ax = b$ nécessite la résolution du système linéaire triangulaire inférieur $Ry = b$, puis du système triangulaire supérieur $R^t x = y$. Compléter l'algorithme de sorte à obtenir la solution x du système linéaire $Ax = b$:

- (a) par une méthode de descente de $Ry = b$:

$$\left\{ \begin{array}{l} b(1) = b(1)/a(1) \\ \text{Pour } i \text{ allant de } 1 \text{ à } N-1 \text{ Faire} \\ \quad b(i+1) = (b(i+1) - ?*?) / ? \end{array} \right.$$

- (b) par une méthode de remontée de $R^t x = y$:

$$\left\{ \begin{array}{l} b(N) = b(N)/a(N) \\ \text{Pour } i \text{ allant de } N-1 \text{ à } 1 \text{ (avec un pas de -1) Faire} \\ \quad b(i) = (b(i) - ?*?) / ? \end{array} \right.$$

Les éléments non nuls de la matrice R correspondent-ils aux éléments non nuls de la matrice A ?

Application en Probabilité - Simulation de la loi $\mathcal{N}(m, \Gamma)$. Ici $m \in \mathbf{R}^d$ et Γ est une matrice de $\mathcal{M}_d(\mathbf{R})$, symétrique et définie-positive. Soient C une matrice telle que $CC^t = \Gamma$ et Z un vecteur aléatoire gaussien de loi $\mathcal{N}(0, I_d)$. Alors $X = CZ + m$ a pour loi $\mathcal{N}(m, \Gamma)$.

Correction

```
SAGEMath: N = 3
SAGEMath: a = [3.0 for i in range(N)]
SAGEMath: c = [-1.0 for i in range(N-1)]
SAGEMath: A = matrix(RR, N, N)
SAGEMath: for i in range(N):
...     A[i,i]=a[i]
...
SAGEMath: for i in range(N-1):
...     A[i,i+1]=c[i]
```

```

...
SAGEmath: for i in range(1,N):
...     A[i,i-1]=c[i-1]
...
SAGEmath: show(A)

```

$$\begin{pmatrix} 3.000000000000000 & -1.000000000000000 & 0.000000000000000 \\ -1.000000000000000 & 3.000000000000000 & -1.000000000000000 \\ 0.000000000000000 & -1.000000000000000 & 3.000000000000000 \end{pmatrix}$$

```

show(A.cholesky_decomposition())

```

$$\begin{pmatrix} 1.73205080756888 & 0.000000000000000 & 0.000000000000000 \\ -0.577350269189626 & 1.63299316185545 & 0.000000000000000 \\ 0.000000000000000 & -0.612372435695794 & 1.62018517460197 \end{pmatrix}$$

```

SAGEmath: def cholesky(a,c):
...     a[0]=sqrt(a[0]).n()
...     for i in range(N-1):
...         c[i]=c[i]/a[i].n()
...         a[i+1]=(sqrt(a[i+1]-c[i]^2)).n()
...     A = matrix(CC,N,N)
...     for i in range(N):
...         A[i,i]=a[i]
...     for i in range(1,N):
...         A[i,i-1]=c[i-1]
...     return(A)
SAGEmath: a = [3.0 for i in range(N)]
SAGEmath: c = [-1.0 for i in range(N-1)]
SAGEmath: show(cholesky(a,c))

```

$$\begin{pmatrix} 1.73205080756888 & 0.000000000000000 & 0.000000000000000 \\ -0.577350269189626 & 1.63299316185545 & 0.000000000000000 \\ 0.000000000000000 & -0.612372435695794 & 1.62018517460197 \end{pmatrix}$$

A.8 Méthodes itératives

Il s'agit de résoudre un système d'équation linéaires par les méthodes développées précédemment (LU, Cholesky, Jacobi, Gradient conjugué).

1. Ecrire les algorithmes de descentes puis de remontée appliqués à des matrices triangulaires (respectivement inférieures et supérieures)
2. Résoudre le système d'équations linéaires $Ax = b$ avec

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

3. Vérifiez que A est symétrique définie-positive. Calculez la factorisation de Cholesky de A puis résoudre le système $Ax = b$.

4. La méthode de Jacobi consiste à calculer la suite $x_{k+1} = Jx_k + D^{-1}b$ avec $J = I_n + D^{-1}A$ et D la matrice diagonale contenant les éléments de la diagonale de A . Appliquez la méthode de Jacobi, en partant de $x_0 = \begin{pmatrix} \frac{3}{2} \\ 2 \end{pmatrix}$. Vous remarquerez que $x_2 = x^*$ la solution du système $Ax = b$.
5. Résoudre le système $Ax = b$ grâce à la factorisation de Cholesky puis grâce à la méthode de Jacobi, avec $x_0 = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$:

$$A = \begin{pmatrix} 4 & 2 & 1 \\ -1 & 2 & 0 \\ 2 & 1 & 4 \end{pmatrix}, b = \begin{pmatrix} 4 \\ 2 \\ 9 \end{pmatrix}$$

Annexe B

Exercices de Révision et Applications

Exercice B.0.1. Citer deux méthodes d'interpolation polynomiale. Quelle est la différence entre ces méthodes et la méthode des moindres carrés discrets ?

Exercice B.0.2. Soit f une fonction de \mathbf{R} dans \mathbf{R} telle que

$$f(x) = x^3 + 3x^2 - 1 \quad \text{pour} \quad x \in \mathbf{R}.$$

1. Montrer que l'équation $f(x) = 0$ admet une unique solution sur $[0, 1]$.
2. Nous désirons déterminer la solution de l'équation $f(x) = 0$ avec la méthode de Newton sur $[0, 1]$.
 - (a) Est-ce que la fonction f vérifie les conditions du théorème de convergence globale de la méthode de Newton pour $x_0 = 1$. Donner le théorème.
 - (b) Calculer la racine de f sur l'intervalle $[0, 1]$ avec une précision de calculs égale à 10^{-6} .
3. Ecrire l'algorithme général de la méthode de Newton qui prend en entrée les points x_0 , $a < b$ et une fonction f et rend la racine de $f(x) = 0$ sur $[a, b]$ ou bien un mesSAGEmath d'erreur.

Exercice B.0.3 (10pt). Soit f une fonction de \mathbf{R} dans \mathbf{R} telle que

$$f(x) = x^3 + 3x^2 - 1 \quad \text{pour} \quad x \in \mathbf{R}.$$

1. Montrer que l'équation $f(x) = 0$ admet une unique solution sur $[0, 1]$.
2. Nous désirons déterminer la solution de l'équation $f(x) = 0$ avec la méthode de Newton sur $[0, 1]$. En quoi consiste cette méthode ?
3. Ecrire l'algorithme général de la méthode de Newton qui prend en entrée les points x_0 , $a < b$ et une fonction f et rend la racine de $f(x) = 0$ sur $[a, b]$ ou bien un mesSAGEmath d'erreur.

4. Est-ce que la fonction f vérifie les conditions du théorème de convergence globale de la méthode de Newton pour $x_0 = 1$. Donner le théorème.
5. Calculer la racine de f sur l'intervalle $[0, 1]$ avec une erreur de 10^{-6} .

Exercice B.0.4 (10pt). Soit A une matrice définie par :

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

1. Donner la définition de la décomposition LU d'une matrice carrée.
2. Calculer la factorisation LU de A .
3. En déduire le déterminant de A .
4. Le système linéaire

$$Ax = \begin{pmatrix} 1 \\ 0 \\ -2 \\ 2 \end{pmatrix} \quad \text{où} \quad x \in \mathbf{R}^4$$

admet-il une solution ? est-elle unique ?

5. Résoudre le système en utilisant la factorisation LU de la matrice A .

Exercice B.0.5. 1. Soit f une fonction continue sur un intervalle $[a, b] \subset \mathbf{R}$ avec $f(a) \cdot f(b) < 0$. Expliquer géométriquement la méthode de Newton dans la recherche d'une solution de l'équation $f(x) = 0$ sur $[a, b]$.

2. Donner l'algorithme de la méthode de Newton pour la recherche d'une racine simple d'une fonction de classe C^2 sur $[a, b]$.

Exercice B.0.6. Nous considérons l'ensemble des points

$$\begin{array}{llll} x_0 = -1, & x_1 = 0, & x_2 = 1, & x_3 = 2, \\ f_0 = 1, & f_1 = -1, & f_2 = 3, & f_3 = 19. \end{array}$$

1. Afin de déterminer une approximation du nuage de points (x_i, f_i) pour i de 0 à 3, calculer la droite de régression linéaire P_0 par la méthode des moindres carrés discrets.
2. Citer deux méthodes permettant de calculer le polynôme de Lagrange.
3. Montrer que les polynômes d'interpolation de Lagrange vérifient $L_i(x_i) = 1$ et $L_i(x_j) = 0$ pour tout i et $j \neq i$ de 0 à 3.
4. Calculer le polynôme de Lagrange P sur les points (x_0, f_0) , (x_1, f_1) , (x_2, f_2) et (x_3, f_3) .
5. Quelle est la différence entre P et P_0 .
6. Donner le théorème de convergence globale de la méthode de Newton pour une fonction de classe C^2 .

7. Calculer la racine de $P(x) = 0$ sur l'intervalle $[0, 3]$ par la méthode de Newton avec $x_0 = 1$. La précision des calculs est à 10^{-6} près.

Exercice B.0.7. 1. Soit f une fonction continue sur un intervalle $[a, b] \subset \mathbf{R}$ avec $f(a).f(b) < 0$. Expliquer géométriquement la méthode de la sécante dans la recherche d'une solution de l'équation $f(x) = 0$ sur $[a, b]$.

2. Donner l'ordre de la méthode de Newton pour la recherche d'une racine simple d'une fonction de classe C^2 .

Exercice B.0.8. Nous considérons l'ensemble des points

$$\begin{aligned} x_0 &= -1, & x_1 &= 0, & x_2 &= 1, & x_3 &= 2, \\ f_0 &= -4, & f_1 &= -1, & f_2 &= -2, & f_3 &= 5. \end{aligned}$$

1. Afin de déterminer une approximation du nuage de points (x_i, f_i) pour i de 0 à 3, calculer la droite de régression linéaire P_0 par la méthode des moindres carrés discrets.
2. Citer deux méthodes permettant de calculer le polynôme de Lagrange.
3. Montrer que les polynômes d'interpolation de Lagrange vérifient $L_i(x_i) = 1$ et $L_i(x_j) = 0$ pour tout i et $j \neq i$ de 0 à 3.
4. Calculer le polynôme de Lagrange P sur les points (x_0, f_0) , (x_1, f_1) , (x_2, f_2) et (x_3, f_3) .
5. Quelle est la différence entre P et P_0 .
6. Vérifier que $P(x_i) = f_i$ pour i de 0 à 3 et montrer qu'il existe une unique racine de P sur l'intervalle $[0, 3]$.
7. Donner le théorème de convergence globale de la méthode de Newton pour une fonction de classe C^2 .
8. Calculer la racine de $P(x) = 0$ sur l'intervalle $[0, 3]$ par la méthode de Newton avec $x_0 = 3$. La précision des calculs est à 10^{-6} près.
9. Ecrire l'algorithme de Newton qui prend en entrée les points x_0 , $a < b$ et une fonction f et rend la racine de $f(x) = 0$ sur $[a, b]$ ou bien un mesSAGEmath d'erreur.

Exercice B.0.9. 1. Calculer par la méthode de la puissance, la valeur propre dominante à 10^{-2} près de

$$A = \begin{pmatrix} 4 & -1 & -1 \\ -1 & 2 & -1 \\ -1 & -1 & 2 \end{pmatrix} \quad \text{avec} \quad x_0 = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}.$$

2. Ecrire l'algorithme de la méthode de la puissance inversée.

Exercice B.0.10. Etant donnés une matrice A triangulaire inférieure de taille (n, n) avec des 1 sur la diagonale et un vecteur b de taille n , calculer la complexité en nombre d'opérations élémentaires de la méthode de descente appliquée au système $Ax = b$.

Exercice B.0.11. Soit $n = 3$. Nous considérons l'ensemble de points

$$\begin{aligned} x_0 &= -1, & x_1 &= 0, & x_2 &= 1, & x_3 &= 2, \\ f_0 &= 1, & f_1 &= 0, & f_2 &= 1, & f_3 &= 4. \end{aligned}$$

1. Afin de déterminer une approximation du nuage de points (x_i, f_i) par la droite de régression linéaire, nous appliquerons la méthode des moindres carrés discrets :

(a) Soit A la matrice associée au système des moindres carrés :

$$A = \begin{pmatrix} \sum_{i=0}^{i=n} x_i^2 & \sum_{i=0}^{i=n} x_i \\ \sum_{i=0}^{i=n} x_i & n+1 \end{pmatrix}$$

(b) Donner la décomposition LU de A

(c) Résoudre le système $A \begin{pmatrix} a_0 \\ a_1 \end{pmatrix} = \begin{pmatrix} \sum_{i=0}^{i=n} x_i f_i \\ \sum_{i=0}^{i=n} f_i \end{pmatrix}$ par une descente puis une remontée.

2. Calculer le polynôme de Lagrange P sur les points (x_i, f_i) pour i entre 0 et 3.
3. Quelle est la différence entre P et la droite de régression linéaire ?
4. Donner le théorème de convergence globale de la méthode de Newton pour une fonction de classe C^2 . Peut-on appliquer ce théorème au calcul de la racine de $P(x) = 0$ sur l'intervalle $[0, 3]$ avec $x_0 = 1$?

Exercice B.0.12. Etant donné une matrice A triangulaire supérieure de taille (n, n) inversible et un vecteur b de taille n , calculer la complexité en nombre d'opérations élémentaires de la méthode de remontée appliquée au système $Ax = b$.

Annexe C

Tests, DS et Examens corrigés

C.1 2008-2009

Nom et Prénom :

Groupe :

Ecole ou Institut de l'année dernière :

Test 1 : Interpolation et Approximation

Exercice : Soient (x_i, f_i) pour $i = 0..n$ des points réels ou complexes. Ecrire un algorithme permettant de calculer le polynôme de Lagrange se basant sur les polynômes d'interpolation $L_i(x)$ pour $i = 0..n$.

Déterminer la complexité de cet algorithme en fonction de n .

Correction :

Entrée : (x_i, f_i) pour $i = 0..n$

Sortie : Le polynôme de Lagrange $P(x)$.

```
P := 1 ;
Pour i allant de 0 à n Faire
    L := 1 ;
    Pour j allant de 0 à n Faire
        Si i <> j alors
            L := L + (x - x_j)/(x - x_i) ;
        Fin Si ;
    Fin Pour ;
    P := P + f_i + L ;
Fin Pour ;
Afficher(P) ;
Fin.
```

Il y a $o(4n^2)$ opérations élémentaires. Il s'agit de $o(2n^2)$ additions/soustractions, $o(n^2)$ multi-

plications et $o(n^2)$ divisions.

EXAMEN DU MODULE ANALYSE NUMÉRIQUE

Exercice 1 (6pt).

1. Donner le théorème de Gershgorin appliqué à une matrice A de taille $n \times n$ quelconque.
2. Expliquer la méthode de la puissance appliquée à une matrice A en donnant l'algorithme.
3. Calculer les deux premières itérations de la méthode de la puissance appliquée à :

$$A = \begin{pmatrix} 1 & 0 \\ 0 & 2 \end{pmatrix} \quad \text{avec} \quad y_0 = \begin{pmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{pmatrix}.$$

Exercice 2 (4pt). **Exercice :** Soient (x_i, f_i) pour $i = 0..n$ des points réels ou complexes. Ecrire un algorithme permettant de calculer le polynôme de Lagrange se basant sur les polynômes d'interpolation $L_i(x)$ pour $i = 0..n$. Déterminer la complexité de cet algorithme en fonction de n .

Problème (10pt). Soit $n = 3$. Nous considérons l'ensemble de points

$$\begin{aligned} x_0 &= -1, & x_1 &= 0, & x_2 &= 1, & x_3 &= 2, \\ f_0 &= 1, & f_1 &= 0, & f_2 &= 1, & f_3 &= 4. \end{aligned}$$

1. Donner le théorème de convergence globale de la méthode de Newton pour une fonction de classe C^2 . Peut-on appliquer ce théorème au calcul de la racine de $P(x) = 0$ sur l'intervalle $[0, 3]$ avec $x_0 = 1$?
2. Soit $A = \begin{pmatrix} \sum_{i=0}^{i=n} x_i^2 & \sum_{i=0}^{i=n} x_i \\ \sum_{i=0}^{i=n} x_i & n+1 \end{pmatrix}$. Donner la décomposition LU de A .
3. Résoudre le système $A \begin{pmatrix} a_0 \\ a_1 \end{pmatrix} = \begin{pmatrix} \sum_{i=0}^{i=n} x_i f_i \\ \sum_{i=0}^{i=n} f_i \end{pmatrix}$ par une descente puis une remontée.
4. En déduire une approximation du nuage de points (x_i, f_i) par la droite de régression linéaire.
5. Quelle est la différence entre le polynôme de Lagrange et la droite de régression linéaire ?

CORRECTION DE L'EXAMEN D'ANALYSE NUMÉRIQUE

Exercice 1 (6pt).

1. Le théorème de Gershgorin appliqué à une matrice A de taille $n \times n$ quelconque donne une borne sur le spectre de A : toute valeur propre appartient à l'un au moins des disques de Gerschgorin :

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

2. La méthode de la puissance appliquée à une matrice A consiste à choisir au hasard un vecteur y_0 et à calculer la suite $y_{n+1} = \frac{Ay_n}{\|Ay_n\|}$.
3. Pour $y_0 = \begin{pmatrix} 0.71 \\ 0.71 \end{pmatrix}$, nous avons $y_1 = \begin{pmatrix} 0.45 \\ 0.89 \end{pmatrix}$ et $y_2 = \begin{pmatrix} 0.24 \\ 0.97 \end{pmatrix}$.

Exercice 2 (4pt). Entrée : (x_i, f_i) pour $i = 0..n$

Sortie : Le polynôme de Lagrange $P(x)$.

```

P := 1 ;
Pour i allant de 0 à n Faire
    L := 1 ;
    Pour j allant de 0 à n Faire
        Si i <> j alors
            L := L + (x - x_j)/(x - x_i) ;
        Fin Si ;
    Fin Pour ;
    P := P + f_i + L ;
Fin Pour ;
Afficher(P) ;
Fin.
```

Il y a $o(4n^2)$ opérations élémentaires. Il s'agit de $o(2n^2)$ additions/soustractions, $o(n^2)$ multiplications et $o(n^2)$ divisions.

Problème (10pt). Soit $n = 3$. Nous considérons l'ensemble de points

$$\begin{aligned} x_0 &= -1, & x_1 &= 0, & x_2 &= 1, & x_3 &= 2, \\ f_0 &= 1, & f_1 &= 0, & f_2 &= 1, & f_3 &= 4. \end{aligned}$$

1. Voir le cours pour le théorème de convergence globale de la méthode de Newton pour une fonction de classe C^2 .

On ne peut appliquer ce théorème au calcul de la racine de $P(x) = x^2$ sur l'intervalle $[0, 3]$: il faudra restreindre l'intervalle car $P'(0) = 0$ et $P''(0) = 0$.

2. Soit $A = \begin{pmatrix} 6 & 2 \\ 2 & 4 \end{pmatrix}$. La décomposition LU de A est triviale :

$$L = \begin{pmatrix} 1 & 0 \\ \frac{1}{3} & 1 \end{pmatrix}, \quad U = \begin{pmatrix} 6 & 2 \\ 0 & \frac{10}{3} \end{pmatrix}.$$

3. La résolution du système $A \begin{pmatrix} a_0 \\ a_1 \end{pmatrix} = \begin{pmatrix} 8 \\ 6 \end{pmatrix}$ par une descente donne $\begin{pmatrix} 8 \\ \frac{10}{3} \end{pmatrix}$ puis la remontée donne $\begin{pmatrix} 1 \\ 1 \end{pmatrix}$.
4. Une approximation du nuage de points (x_i, f_i) est donnée par la droite de régression linéaire d'équation $P_0(x) = x + 1$.
5. Conclure sur la variation en terme de degré, de complexité des calculs et en terme de différence entre interpolation et approximation.

EXAMEN DE CONTRÔLE DU MODULE ANALYSE NUMÉRIQUE

Problème 1 (10pt). Notons A une matrice symétrique définie positive d'ordre n .

$$r_{1,1} = \sqrt{a_{1,1}}$$

pour i de 2 à n faire :

$$r_{i,j} = \frac{a_{i,j} - \sum_{k=1}^{j-1} r_{ik} r_{jk}}{r_{j,j}} \quad \text{pour} \quad j = 1..i-1$$

et

$$r_{i,i} = \sqrt{a_{i,i} - \sum_{k=1}^{i-1} r_{ik}^2}.$$

1. Ecrire l'algorithme de la méthode de Cholesky
2. Supposons que la fonction racine suppose au plus 9 opérations élémentaires. Déterminer la complexité de la méthode de factorisation de Cholesky de A en fonction de n .
3. Donner les 2 premières itérations de cette méthode appliquée à la matrice

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

Problème 2 (10pt). Soit $n = 3$. Nous considérons l'ensemble de points

$$\begin{aligned} x_0 &= -1, & x_1 &= 0, & x_2 &= 1, & x_3 &= 2, \\ f_0 &= 1, & f_1 &= 0, & f_2 &= 1, & f_3 &= 4. \end{aligned}$$

1. Donner le théorème de convergence globale de la méthode de Newton pour une fonction de classe C^2 . Peut-on appliquer ce théorème au calcul de la racine de $P(x) = 0$ sur l'intervalle $[0, 3]$ avec $x_0 = 1$?

2. Soit $A = \begin{pmatrix} \sum_{i=0}^{i=n} x_i^2 & \sum_{i=0}^{i=n} x_i \\ \sum_{i=0}^{i=n} x_i & n+1 \end{pmatrix}$. Donner la décomposition LU de A .

3. Résoudre le système $A \begin{pmatrix} a_0 \\ a_1 \end{pmatrix} = \begin{pmatrix} \sum_{i=0}^{i=n} x_i f_i \\ \sum_{i=0}^{i=n} f_i \end{pmatrix}$ par une descente puis une remontée.
4. En déduire une approximation du nuage de points (x_i, f_i) par la droite de régression linéaire.
5. Quelle est la différence entre le polynôme de Lagrange et la droite de régression linéaire ?

CORRECTION DE L'EXAMEN DE CONTRÔLE DU MODULE ANALYSE NUMÉRIQUE

Problème 1 (10pt). Notons A une matrice symétrique définie positive d'ordre n .

1. L'algorithme de la méthode de Cholesky est le suivant :

Entrée : $a_{i,j}$

Sortie : $r_{i,j}$

$$r_{1,1} = \sqrt{a_{1,1}}$$

Pour i allant de 2 à n Faire

$$r_{i,i} = a_{i,i}$$

Pour j allant de 1 à $i - 1$ Faire

$$r_{i,j} = a_{i,j}$$

Pour k allant de 1 à $j - 1$ Faire

$$r_{i,j} = r_{i,j} - r_{i,k}r_{j,k}$$

Fin Pour ;

$$r_{i,j} = \frac{r_{i,j}}{r_{j,j}}$$

Fin Pour

Pour k allant de 1 à $i - 1$ Faire

$$r_{i,i} = r_{i,i} - r_{i,k}r_{i,k}$$

Fin Pour ;

$$r_{i,i} = \sqrt{r_{i,i}}$$

Fin Pour ;

Fin.

2. Sachant que la fonction racine suppose au plus 9 opérations élémentaires, la complexité de la méthode de factorisation de Cholesky de A en fonction de n est calculée comme suit :
 Il y a $\sum_{i=2}^n (\sum_{j=1}^{i-1} (j-1)) + \sum_{i=2}^n (i-1) = \frac{n(n-1)(n-2)}{6} + \frac{n(n-1)}{2}$ additions.
 Il y a $\sum_{i=2}^n (\sum_{j=1}^{i-1} j) + \sum_{i=2}^n (i-1) = \frac{n(n-1)(n-2)}{6} + \frac{n(n-1)}{2}$ multiplications.
 Et il y a $\sum_{i=2}^n 9 = 9(n-1)$. Donc $9(n-1)$ opérations élémentaires permettant le calcul des racines carrés.

Nous obtenons une complexité de l'ordre de $o(\frac{1}{3}n^3 + 9n)$.

3. La matrice R étant triangulaire inférieure, elle est calculée par la méthode de Cholesky appliquée à A . Nous obtenons $\begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}$.

Problème 2 (10pt). Soit $n = 3$. Nous considérons l'ensemble de points

$$\begin{aligned} x_0 &= -1, & x_1 &= 0, & x_2 &= 1, & x_3 &= 2, \\ f_0 &= 1, & f_1 &= 0, & f_2 &= 1, & f_3 &= 4. \end{aligned}$$

1. Voir le cours pour le théorème de convergence globale de la méthode de Newton pour une fonction de classe C^2 .

On ne peut appliquer ce théorème au calcul de la racine de $P(x) = x^2$ sur l'intervalle $[0, 3]$: il faudra restreindre l'intervalle car $P'(0) = 0$ et $P''(0) = 0$.

2. Soit $A = \begin{pmatrix} 6 & 2 \\ 2 & 4 \end{pmatrix}$. La décomposition $PA = LU$ est triviale :

$$L = \begin{pmatrix} 1 & 0 \\ \frac{1}{3} & 1 \end{pmatrix}, \quad U = \begin{pmatrix} 6 & 2 \\ 0 & \frac{10}{3} \end{pmatrix}.$$

Ici, la matrice de permutation P est égale à la matrice identité d'ordre 2.

3. La résolution du système $A \begin{pmatrix} a_0 \\ a_1 \end{pmatrix} = \begin{pmatrix} 8 \\ 6 \end{pmatrix}$ par une descente donne $\begin{pmatrix} 8 \\ \frac{10}{3} \end{pmatrix}$ puis la remontée donne $\begin{pmatrix} 1 \\ 1 \end{pmatrix}$.

4. Une approximation du nuage de points (x_i, f_i) est donnée par la droite de régression linéaire d'équation $P_0(x) = x + 1$.
5. La variation entre ces deux approches se notent en terme de degré : le polynôme d'interpolation ayant un degré égal au nombre de points-1 alors que dans l'approximation, le degré en est indépendant. Dans le cas de l'approximation, le degré est largement plus petit que dans l'interpolation. La différence entre les deux approches est constatée dans la complexité des calculs : le cas étudié en cours, donne une approximation plus rapidement que l'interpolation de Lagrange par exemple.

C.2 2009-2010

EXAMEN DU MODULE ANALYSE NUMÉRIQUE

Exercice 1 (6pt). Notons $A = (a_{ij})$ une matrice symétrique définie positive d'ordre n . La factorisation de la matrice A par la méthode de Cholesky est détaillée comme suit :

$$r_{1,1} = \sqrt{a_{1,1}}$$

pour i de 2 à n faire :

$$r_{i,j} = \frac{a_{i,j} - \sum_{k=1}^{j-1} r_{ik} r_{jk}}{r_{j,j}} \quad \text{pour } j = 1..i-1$$

et

$$r_{i,i} = \sqrt{a_{i,i} - \sum_{k=1}^{i-1} r_{ik}^2}.$$

1. Ecrire l'algorithme de la méthode de Cholesky qui prend en entrée A et qui calcule une matrice R triangulaire vérifiant $A = R^t R$.
2. Supposons que la fonction racine nécessite au plus 9 opérations élémentaires. Déterminer la complexité de la méthode de factorisation de Cholesky de A en fonction de n .
3. Donner la factorisation de Cholesky de la matrice

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

Exercice 2 (4pt). Calculer par la méthode de la puissance la valeur propre et le vecteur propre

dominants de la matrice $A = \begin{pmatrix} 1 & 0 \\ 0 & 2 \end{pmatrix}$ avec $X_0 = \begin{pmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{pmatrix}$.

Problème (10pt). Soit $n = 3$. Nous considérons l'ensemble de points

$$\begin{aligned} x_0 &= -1, & x_1 &= 0, & x_2 &= 1, & x_3 &= 2, \\ f_0 &= 0, & f_1 &= -1, & f_2 &= 0, & f_3 &= 3. \end{aligned}$$

1. Calculer le polynôme de Lagrange P sur les points d'interpolation $x_i, i = 0..3$.
2. Donner le théorème de convergence de la méthode du point fixe pour une fonction de classe C^1 sur un intervalle $[a, b]$. Peut-on appliquer ce théorème au calcul de la racine de $P(x) = 0$ sur l'intervalle $[-2, 0]$ avec $g(x) = \frac{1}{2}P(x) + x$ et $X_0 = 0$? Si oui, calculer la racine de P sur $[-2, 0]$.
3. Soit $A = \begin{pmatrix} \sum_{i=0}^{i=n} x_i^2 & \sum_{i=0}^{i=n} x_i \\ \sum_{i=0}^{i=n} x_i & n+1 \end{pmatrix}$. Résoudre le système $A \begin{pmatrix} a_0 \\ a_1 \end{pmatrix} = \begin{pmatrix} \sum_{i=0}^{i=n} x_i f_i \\ \sum_{i=0}^{i=n} f_i \end{pmatrix}$ par une descente puis une remontée (suite à une factorisation LU).
4. En déduire une approximation du nuage de points (x_i, f_i) par la droite de régression linéaire $p_1(x) = a_0x + a_1$.

EXAMEN DE CONTRÔLE DU MODULE ANALYSE NUMÉRIQUE

Exercice (8pt).

1. Etant donné une matrice $A = (a_{ij})$, λ une de ses valeurs propres associé au vecteur propre v et i l'indice de la coordonnée v_i de v vérifiant $|v_i| = \max_{1 \leq j \leq n} |v_j|$. Montrer que

$$|\lambda - a_{ii}| \leq \sum_{j=1, j \neq i}^{j=n} |a_{ij}|.$$

2. Calculer la valeur propre et le vecteur propre dominants de A par la méthode de la puissance :

$$A = \begin{pmatrix} 1 & 0 \\ 0 & 2 \end{pmatrix} \quad \text{avec} \quad X_0 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}.$$

3. Que se passe-t-il lorsqu'on considère $X_0 = \begin{pmatrix} 0 \\ 2 \end{pmatrix}$?

Problème (12pt). Soit $n = 3$. Nous considérons l'ensemble de points

$$\begin{aligned} x_0 &= -1, & x_1 &= 0, & x_2 &= 1, & x_3 &= 2, \\ f_0 &= 0, & f_1 &= -1, & f_2 &= 0, & f_3 &= 3. \end{aligned}$$

1. Calculer le polynôme de Lagrange P sur les points d'interpolation $x_i, i = 0..3$.
2. Donner le théorème de convergence de la méthode du point fixe pour une fonction de classe C^1 sur un intervalle $[a, b]$. Peut-on appliquer ce théorème au calcul de la racine de $P(x) = 0$ sur l'intervalle $[0, 2]$ avec $g(x) = \frac{1}{2}P(x) + x$ et $X_0 = 0$?
3. Soit $A = \begin{pmatrix} \sum_{i=0}^{i=n} x_i^2 & \sum_{i=0}^{i=n} x_i \\ \sum_{i=0}^{i=n} x_i & n+1 \end{pmatrix}$. Donner la décomposition LU de A .
4. Résoudre le système $A \begin{pmatrix} a_0 \\ a_1 \end{pmatrix} = \begin{pmatrix} \sum_{i=0}^{i=n} x_i f_i \\ \sum_{i=0}^{i=n} f_i \end{pmatrix}$ par une descente puis une remontée.
5. En déduire une approximation du nuage de points (x_i, f_i) par la droite de régression linéaire $p_1(x) = a_0x + a_1$.
6. Quelle est la différence entre le polynôme de Lagrange et la droite de régression linéaire ?

C.3 2010-2011

DEVOIR SURVEILLÉ DU MODULE ANALYSE NUMÉRIQUE

Interpolation.

1. Soient f une fonction de classe C^2 sur $[0, 1]$ et $xdata = [x_0, x_1, \dots, x_n]$ une liste de $n+1$ points deux à deux distincts de $[0, 1]$. Ecrire l'algorithme de la méthode des différences divisées qui prend en entrée f et $xdata$ et qui calcule le polynôme de Lagrange de f aux points x_0, x_1, \dots, x_n .

Indication : la formule des différences divisées est la suivante :

$$\begin{cases} c_{i,i} &= f(x_i) & \text{pour } i = 0, \dots, n \\ c_{i,j} &= \frac{c_{i+1,j} - c_{i,j-1}}{x_j - x_i} & \text{pour } 0 \leq i < j \leq n. \end{cases}$$

2. Répondre par Vrai ou Faux aux affirmations suivantes et justifier votre réponse :
 - (a) Il existe une infinité de polynômes de degré $> n$ interpolant les points $(x_i, f(x_i))_{i=0,\dots,n}$.
 - (b) Le polynôme de Lagrange de $\sin(x)$ sur les points $0, \frac{\pi}{2}$ et π est égal à $P(x) = x^3 + 3x^2$.

Intégration.

3. Calculer par la méthode des trapèzes une approximation $T(h)$ de $\int_0^2 x^2 dx$. Considérer pour cela une subdivision de l'intervalle $[0, 2]$ en 2 sous intervalles de même amplitude.
4. Soient $h = \frac{b-a}{n}$ et $x_i = a + ih$. Sachant que $T(h) = \sum_{i=0}^{n-1} \frac{h}{2}(f(x_i) + f(x_{i+1}))$, vérifier que

$$T\left(\frac{h}{2}\right) = \frac{T(h)}{2} + \frac{h}{2} \sum_{k=0}^{n-1} f\left(a + kh + \frac{h}{2}\right).$$

Connaissant $T(h)$, donner le nombre d'opérations élémentaires nécessaires au calcul de $T(\frac{h}{2})$. Nous supposons qu'une évaluation de f nécessite α multiplications et β additions.

5. En déduire $\int_0^2 x^2 dx$ pour une subdivision de l'intervalle $[0, 2]$ en 4 sous intervalles de même amplitude.

Résolution d'équations non linéaires.

6. Donner le théorème de convergence globale de la méthode de Newton pour une fonction de classe C^2 sur un intervalle $[a, b]$.
7. Peut-on appliquer ce théorème au calcul de la racine x^* de $f(x) = x^3 - 1$ sur l'intervalle $[0.5, 1.5]$? Si oui, déterminer x^* à 10^{-1} près.

ANALYSE NUMÉRIQUE - CORRECTION DU DS

Interpolation.

1. Soient f une fonction de classe C^2 sur $[0, 1]$ et $xdata = [x_0, x_1, \dots, x_n]$ une liste de $n+1$ points deux à deux distincts de $[0, 1]$. L'algorithme suivant reprend la méthode des différences divisées.

Entrée : Une fonction f et $xdata == [x_0, x_1, \dots, x_n]$

Sortie : Le polynôme de Lagrange de f aux points x_0, x_1, \dots, x_n

Méthode des différences divisées (formule de Newton)

```
def newton(f,xdata):
    n = len(xdata)-1
    c = matrix(QQ,n+1,n+1)
    for i in range(n+1):
        c[i,i] = f(xdata[i])
    #print c
    #print("-----")
    for k in range(1,n+1):
        for i in range(n+1-k):
            c[i,i+k] = (c[i+1,i+k]-c[i,i+k-1])/(xdata[i+k]-xdata[i])
        #print c
        #print("-----")
    P = c[0,0]
    P += sum(c[0,k]*prod((x-xdata[j]) for j in range(k)) for k in range(1,n+1))
    return(expand(P))
```

2. Répondre par Vrai ou Faux aux affirmations suivantes et justifier votre réponse :

- (a) Vrai. Il existe une infinité de polynômes de degré supérieur strictement à n interpolant les points $(x_i, f(x_i))_{i=0,\dots,n}$. En particulier, par deux points passe une infinité de courbes, mais une seule est une droite.
- (b) Faux. Le polynôme de Lagrange de $\sin(x)$ sur les points $0, \frac{\pi}{2}$ et π n'est pas égal à $P(x) = x^3 + 3x^2$. Ici, P est un polynôme de degré supérieur au nombre de points + 1 et $P(x_i) \neq \sin(x_i)$.

Intégration.

3. Une approximation $T(h)$ de $\int_0^2 x^2 dx$ par la méthode des trapèzes avec une subdivision de l'intervalle $[0, 2]$ en 2 sous intervalles de même amplitude est égale à $T(1) = 3$.
4. Soient $\frac{h}{2} = \frac{b-a}{2n}$ et $x_i = a + i\frac{h}{2}$ pour $i = 0, \dots, 2n$. Il suffit de regrouper les indices i dans $T(\frac{h}{2})$ en deux groupes : le premier constitué d'indices pairs (donnant $\frac{T(h)}{2}$) et le second groupe d'indices impairs (donnant $\frac{h}{2} \sum_{k=0}^{n-1} f(a + kh + \frac{h}{2})$). Ainsi, nous obtenons que

$$T(\frac{h}{2}) = \frac{T(h)}{2} + \frac{h}{2} \sum_{k=0}^{n-1} f(a + kh + \frac{h}{2}). \quad (\text{C.3.1})$$

Connaissant $T(h)$, Le nombre d'opérations élémentaires nécessaires au calcul de $T(\frac{h}{2})$ est égal à

$$\begin{array}{cc|cc} n(\beta + 2) + 1 & \text{additions} & 0 & \text{soustractions} \\ n(\alpha + 1) & \text{multiplications} & 2 & \text{divisions} \end{array}$$

où α multiplications et β additions sont nécessaires pour une seule évaluation de f .

5. Remplacer $a = 0$, $b = 2$, $f(x) = x^2$, $n = 2$ et $h = 1$ dans (C.3.1). On obtient $\int_0^2 x^2 dx$ pour une subdivision de l'intervalle $[0, 2]$ en 4 sous intervalles de même amplitude, i.e. 2.75 :

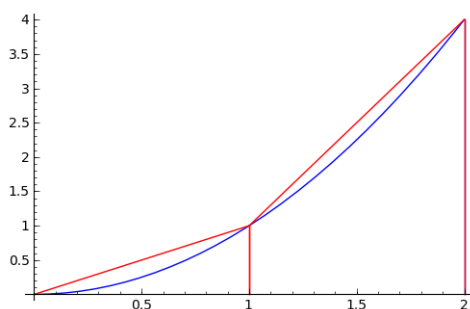


FIGURE C.1 – Intégration pour $h = 1$

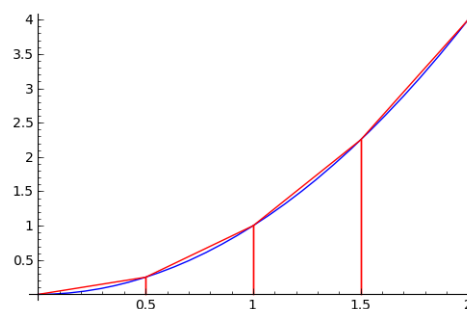
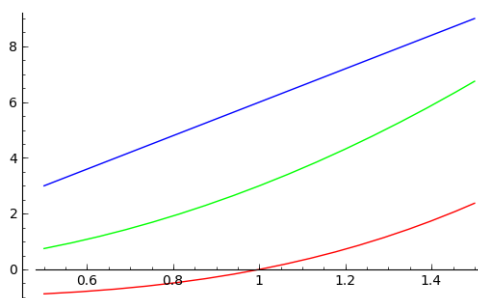


FIGURE C.2 – Intégration pour $h = \frac{1}{2}$

Résolution d'équations non linéaires.

6. Le théorème de convergence globale de la méthode de Newton. Soit f une fonction de classe C^2 sur un intervalle $[a, b]$ vérifiant $f(a).f(b) < 0$. Si pour tout $x \in [a, b]$, $f'(x) \neq 0$ et $f''(x) \neq 0$ alors $\forall x_0 \in [a, b]$ tel que $f(x_0).f''(x_0) > 0$, la suite $x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$ converge vers l'unique racine de f sur $[a, b]$.

FIGURE C.3 – Résolution de $x^3 - 1$

7. La fonction $f(x) = x^3 - 1$ vérifie les conditions du théorème sur l'intervalle $[0.5, 1.5]$. Elle est strictement monotone et elle change de signe. Elle admet donc une unique racine obtenue, par la méthode de Newton, de la manière suivante : x^* à 10^{-1} près. Partant de $x_0 = 1.5$, nous obtenons $x_1 = 1.1481$, $x_2 = 1.0182$ et $x_3 = 1.0003$. La courbe rouge représente f' et la courbe bleue représente f'' .