

Méthodes Numériques.

Licence 3ème année, UFR 27

Denis PENNEQUIN¹

Année Universitaire 2007-2008

¹UFR Mathématiques et Informatique, Université Paris 1, 90 rue de Tolbiac, 75647 Paris CEDEX, France. E-mail : pennequi@univ-paris1.fr

Table des Matières

1	Initiation à SCILAB.	9
1.1	Introduction.	9
1.2	Premières opérations sur les réels et complexes.	11
1.3	Affectations et désaffectations	13
1.4	Fichiers de données, scripts et fonctions	14
1.4.1	Fichiers de données	14
1.4.2	Fichiers scripts	14
1.4.3	Création d'une fonction	15
1.5	Calcul vectoriel et matriciel.	16
1.5.1	Saisie.	17
1.5.2	Premières opérations.	18
1.5.3	Extraction de sous matrices.	19
1.5.4	Opération élément par élément (ou vectorisation).	19
1.5.5	Fonctions complémentaires.	22
1.5.6	Moindres carrés et projections orthogonales.	22
1.5.7	Valeurs propres et exponentielle de Matrice.	24
1.6	Graphiques	25
1.6.1	Ce qu'il faut savoir.	25
1.6.2	Courbes du plan (suite).	26
1.6.3	Courbes de l'espace.	27
1.6.4	Surfaces de l'espace.	28
1.6.5	Subdivision de la fenêtre.	28
1.7	Booléens et opérateurs relationnels.	29
1.8	Boucles et structures conditionnelles.	30
1.8.1	Boucles for.	30
1.8.2	Instruction while.	31
1.8.3	Condition if.	32
1.8.4	Une application aux mathématiques financières.	33
1.9	Polynômes et fractions rationnelles.	34
1.10	Gestion du temps	34
2	Gestion des erreurs.	37
2.1	Erreurs relatives et absolues.	37
2.2	Propagation des erreurs.	37

2.3	Erreurs dans les systèmes linéaires.	38
2.3.1	Normes vectorielles et matricielles.	39
2.3.2	Effets des incertitudes.	40
3	Calcul Scientifique.	43
3.1	Interpolation.	43
3.1.1	Conditions en un seul point.	44
3.1.2	Conditions en plusieurs points.	44
3.2	Calcul de séries.	46
3.2.1	Sommes partielles et calcul de π par la fonction ζ de Riemann.	46
3.2.2	Autres séries convergeant vers π	47
3.2.3	Un exemple d'accélération de convergence.	48
3.2.4	Limites du calcul numérique.	48
3.3	Racines et extrema de fonctions d'une variable.	48
3.3.1	Un exemple.	49
3.3.2	Méthode de dichotomie.	49
3.3.3	Méthode de la sécante et méthode de Newton.	49
3.3.4	Conclusions	51
3.4	Calcul d'intégrales.	51
3.4.1	Méthodes élémentaires.	52
3.4.2	Méthodes composées.	53
3.4.3	Calcul d'intégrales : méthodes de Gauss.	54
3.5	Résolution d'Equations Différentielles Ordinaires.	58
3.5.1	La fonction <code>ode</code> de SCILAB.	58
3.5.2	Méthode d'Euler (explicite).	59
3.6	Résolution d'EDP : Calcul approché de dérivées et différences finies.	60
3.6.1	Calcul approché de dérivées de fonctions d'une variable.	60
3.6.2	La méthode des différences finies en dimension 1.	62
3.6.3	La méthode des différences finies en dimension 2 : calcul des dérivées partielles et résolution d'EDP.	63
4	Probabilités et Statistiques.	65
4.1	Nombres pseudo-aléatoires	65
4.1.1	Un système chaotique.	65
4.1.2	Nombres pseudo-aléatoires.	66
4.2	Les lois usuelles	67
4.2.1	Lois discrètes	67
4.2.2	Lois uniformes	68
4.2.3	Lois normales	68
4.2.4	Simulation d'autres lois usuelles	69
4.3	Théorèmes limites en Calcul des Probabilités.	69
4.3.1	Loi des Grands Nombres	70
4.3.2	Application de la Loi des Grands Nombres : distribution d'une variable aléatoire.	70

4.3.3	Théorème Central Limite	71
4.4	Estimation ponctuelle et ensembliste	71
4.5	La méthode de Monte-Carlo et ses applications.	72
4.5.1	Application aux calculs d'intégrales	73
4.5.2	Application aux EDP issues de la finance	74

I Introduction à MAPLE et à quelque-unes de ses applications. 75

5 Généralités sur MAPLE 77

5.1	Petit tour d'horizon et description de l'environnement de travail	77
5.2	Premières commandes	78
5.3	Expressions et fonctions	79
5.3.1	Fonctions	79
5.3.2	Expressions	80
5.3.3	Passages entre fonctions et expressions	80
5.4	Quelques fonctions et constantes prédéfinies	80
5.5	Types de variables	81
5.5.1	Expressions élémentaires	81
5.5.2	Valeurs	82
5.5.3	Relations et booléens	84
5.5.4	Intervalles	84
5.5.5	Séquences	84
5.5.6	Listes et ensembles	85
5.5.7	Tables, tableaux et matrices	86
5.5.8	Polynômes, développements limités et asymptotiques	88
5.6	Premières règles d'évaluation	88
5.7	Utilisation des bibliothèques (ou packages)	89
5.8	Programmation	90
5.8.1	Instructions de contrôle	90
5.8.2	Fonctions et procédures	91

6 Utilisation de MAPLE en mathématiques 93

6.1	Développement, factorisation et simplification d'expressions	93
6.1.1	Rappels mathématiques	93
6.1.2	Quelques fonctions de manipulation d'expression	94
6.1.3	La fonction <code>convert</code>	95
6.1.4	La fonction <code>simplify</code>	95
6.1.5	Quelques exercices de manipulation d'expressions	96
6.2	Résolution d'équations et de systèmes d'équations	97
6.2.1	Résolution exacte : la commande <code>solve</code>	97
6.2.2	Résolution approchée : la commande <code>fsolve</code>	99
6.2.3	Exercice avec ces commandes	99

6.3	Limites, dérivées et développements	99
6.3.1	Limites	99
6.3.2	Dérivées	100
6.3.3	Développements limités et asymptotiques	101
6.3.4	Petit exercice	101
6.4	Sommes, produits et intégrales	101
6.4.1	Sommes et produits sur des ensembles numériques	101
6.4.2	Sommes	102
6.4.3	Intégration	103
6.5	Graphiques en dimensions 2 et 3	104
6.5.1	L'environnement graphique	104
6.5.2	Généralités sur les graphiques en dimension 2	104
6.5.3	Les options de plot	105
6.5.4	Tracés de surfaces	106
6.6	Algèbre linéaire	106
6.6.1	Premières commandes	107
6.6.2	Opérations matricielles	107
6.6.3	Réduction des matrices	108
6.7	Equations et systèmes différentiels	109
6.7.1	Résolution exacte	109
6.7.2	Résolution numérique	110
6.7.3	Résolution graphique	111
7	Quelques applications de MAPLE	113
7.1	Calculs numériques en grande précision	113
7.2	Systèmes dynamiques linéaires	114
7.3	La méthode de Newton	114
7.4	Une étude d'un modèle dynamique	115
II	Compléments.	117
8	Correction des exercices et sujets précédents.	119
8.1	Correction des exercices de SCILAB.	119
8.1.1	Section 1.5.	119
8.1.2	Exercice 1.5.7	120
8.1.3	Exercice 1.6.1. page 21	120
8.1.4	Exercice 1.7.1. page 23	120
8.1.5	Exercice 1.8.2 page 24	121
8.1.6	Section 1.8.4 : amortissements	121
8.1.7	Exercice 1.11.1 page 26	122
8.1.8	Exercice 2.1.1. page 28	122
8.1.9	Exercice 3.2.1. page 31	122
8.1.10	Exercice sur l'interpolation page 32	122

8.1.11	Exercice application des moindres carrés à Cobb-Douglas page 31	123
8.1.12	Poly page 36	123
8.1.13	Exercice 3.4.1 page 36	124
8.1.14	Système non linéaire de l'exercice 3.5.1 page 37	124
8.1.15	Linéarisé du précédent	124
8.1.16	Exercice sur Fibonacci page 37	124
8.1.17	Système chaotique page 38	124
8.1.18	Exercice 4.1.2 page 40	125
8.1.19	Exercice 4.1.3 page 40	125
8.1.20	Section 4.3. // les corrigés sont à refaire dans toute la 4.3	126
8.1.21	Section 5.1.	127
8.1.22	Section 6.2	128
8.2	Correction des exercices MAPLE	129
8.2.1	Section 7.1.5, manipulation d'expressions	129
8.2.2	Section 7.2.3., résolution d'équations	129
8.2.3	Section 7.3.1.	129
8.2.4	Section 7.4.1.	130
8.2.5	Section 7.4.2., intégrales	130
8.2.6	Section 7.5.1., fonctions de production CES et Cobb-Douglas	131
8.2.7	Section 8.1., évaluation en grande précision	131
8.2.8	Section 8.2., quelques illustrations de systèmes dynamiques linéaires.	131
8.2.9	Section 8.3., méthode de Newton	132
8.2.10	Section 8.4., un système dynamique	132
8.3	Examen du 19 mai 2001.	133
8.4	Examen du 4 mai 2002.	135
9	Sujets posés antérieurement.	139

Introduction.

Ce cours intermédiaire entre l'Analyse Numérique et l'Informatique a pour buts de vous introduire au calcul numérique et d'illustrer les cours de mathématiques.

Ce cours est illustré à l'aide du logiciel qui est SCILAB. Dit de manière simpliste, SCILAB est un environnement de calcul numérique matriciel efficace. SCILAB est très proche d'autres logiciels, au niveau de l'esprit, notamment de MATLAB ou OCTAVE. MATLAB est le plus connu des trois, mais il présente l'inconvénient majeur d'être un logiciel payant (et de plus très cher). SCILAB est un logiciel gratuit que vous pouvez télécharger sur internet. OCTAVE aurait présenté les avantages d'être complètement libre et avoir une syntaxe quasi identique à MATLAB, mais il est beaucoup moins répandu et sa maintenance semble tourner au ralenti. Cependant, une fois l'esprit de SCILAB apprivoisé, vous pourrez tout à fait passer à MATLAB ou à OCTAVE. J'en profite pour vous signaler que

vous pouvez télécharger (gratuitement) un certain nombre de logiciels libres pour windows sur <http://www.framasoft.net>. Certains logiciels libres n'ont rien à envier à leurs concurrents commerciaux, ils les dépassent même parfois.

Venons-en maintenant à l'organisation de ce polycopié. Il est issu au départ d'un enseignement de SCILAB et de MAPLE, la version en ligne sur les espaces EPI du polycopié contient également la partie MAPLE du polycopié. Celle-ci ne figure cependant absolument pas au programme et n'a pas été actualisée à cet enseignement. La partie SCILAB a en revanche été refondue et adaptée à vos besoins. Pour cela, elle a été recentrée sur le calcul scientifique et les probabilités statistiques. Elle a bénéficié notamment des fiches que Jean-Marc Bardet avait produites pour un enseignement analogue, je tiens à le remercier de me les avoir passées. Cet enseignement en douze séances va être construit ainsi :

- deux séances sur le chapitre 1 : apprentissage des premières commandes de scilab.
- une séance sur le chapitre 2 : gestion des erreurs.
- une séance sur l'interpolation.
- une séance sur le calcul de séries.
- une séance sur la recherche de racines des fonctions d'une variable réelle.
- deux séances sur le calcul d'intégrales (une pour chaque section).
- une séance sur la résolution d'équations différentielles.
- une séance sur la génération de nombres (pseudo-)aléatoires et les lois usuelles.
- une séance sur les théorèmes limites en probabilités.
- une séance sur la méthode de Monte-Carlo.

Chapitre 1

Initiation à SCILAB.

1.1 Introduction.

Il est tout d'abord possible d'utiliser SCILAB de manière interactive. Dans ce cas, SCILAB effectue directement les opérations que vous lui demandez. Il est ici possible de passer en ligne de commande des commandes systèmes. La première que nous allons passer concerne un changement de répertoire, afin que SCILAB sauvegarde nos fichiers dans son propre répertoire. Pour se déplacer dans son propre répertoire, on utilise la commande DOS ou UNIX usuelle `cd`.

Ainsi, vous devez toujours au démarrage de SCILAB vous placer dans un répertoire que vous serez préalablement créé sur le lecteur logique D:

Si par exemple vous vous êtes créé un répertoire `D:\L3\dupont`, vous devrez taper au début de chaque séance :

```
> chdir("D:\L3\dupont")
```

Le symbole `>` représente ce que l'on appelle *une invite* (elle est symbolisée par `-->` dans SCILAB). Elle n'est pas à taper, elle signifie simplement que SCILAB attend que vous entriez des commandes.

Vous pouvez vérifier que vous êtes dans le bon répertoire en tapant la commande `pwd`, qui affiche le répertoire courant. Ces commandes sont également accessibles dans le menu **File** (**Change Directory**, ou **Get Current Directory**).

En utilisation interactive, il n'est pas possible d'enregistrer directement ce que l'on fait. On peut cependant enregistrer une session dans un fichier au format texte, qui contiendra nos entrées et les réponses de SCILAB. Pour enregistrer une séance dans un fichier (par exemple avec le nom `essai.txt`), on commence dès le début à taper :

```
> diary('essai.txt') [Entrée]
```

(Désormais, on sous-entendra `[Entrée]`), puis à la fin de la séance, on tape :

```
> diary(0)
```

pour enregistrer sur le disque dans le répertoire par défaut.

Tout ce qui est compris entre ces deux commandes est mis dans un fichier texte, tel une photocopie. Ce fichier contient en particulier les entrées (y compris les invites) et les sorties. Si vous voulez le transformer en un *fichier script* afin de le rendre exécutable (cf. plus loin), vous devrez en particulier le débarrasser de ses invites.

Il y a une liste de commandes à connaître absolument avant tout apprentissage de SCILAB :

- `//` permet d'entrer un commentaire. A partir de ce caractère, plus rien n'est interprété par SCILAB.
- `help` (suivi d'un nom de fonction) : se passe d'explications !
- `apropos` (suivi d'un mot clé) : liste les fichiers d'aide ayant un rapport avec ce mot (essayez par exemple `apropos sin`).

Commençons par étudier notre environnement de travail. Il est possible de passer certaines commandes MSDOS si vous êtes sur un système Microsoft, ou des commandes Unix à l'aide de la commande `unix` et ses dérivées (survolez `apropos unix`). Signalons par exemple la commande `unix_w` qui donne un résultat en sortie ou `unix_s` qui exécute sans afficher. Par exemple :

`unix_s('cd monrep')` : passe dans le répertoire nommé `monrep`.

`unix_s('copy ...')` sous Windows ou `unix_s('cp ...')` sous Unix fait une copie de fichiers (je suppose que vous connaissez les syntaxes de ces commandes).

On dispose aussi de commandes pour les variables qui sont dans le répertoire courant :

`clear var` : supprime la variable nommée `var`.

`clear` : supprime toutes les variables.

`who` : liste les variables. Une version avec renseignements sur les variables est `whos()`. Cette commande est aussi accessible depuis le menu **Applications, Browser Variables**.

On peut utiliser les raccourcis Emacs, où le raccourci `C+` signifie que l'on appuie sur la touche Control et tout en la maintenant enfoncée, on appuie sur l'autre touche :

- `C+p` (ou `↑`) rappelle la ligne précédente (p=previous).
- `C+n` rappelle la ligne suivante (n=next).
- `C+f` (ou `→`) avance d'un caractère (f=forward).
- `C+b` (ou `←`) recule d'un caractère (b=backward).
- `C+d` supprime un caractère au niveau du curseur (d=delete).

- **C+a** va au début de la ligne.
- **C+e** va à la fin de la ligne.

Ces commandes sont également accessibles via le menu **Edit**, **History**.

Attention : évitez l'emploi du copier-coller brutal. Si jamais vous l'utilisez, n'oubliez pas d'enlever les éventuelles invites, cela évitera des désagréments. Mais en général, il est beaucoup plus efficace de rappeler une ligne antérieure à l'aide des raccourcis indiqués ci-dessus.

1.2 Premières opérations sur les réels et complexes.

Entrez ces lignes à l'invite de SCILAB. Déduisez-en les règles de priorité pour les calculs.

```
> 1+2*3
> (1+2)*3
> 2^(1/2)
> 2^1/2
```

Au vu des résultats précédents et d'autres que vous auriez pu essayer, proposez des règles de priorité dans SCILAB.

Passons maintenant aux opérateurs de division. On dispose de / et de \. Essayez les commandes suivantes et commentez :

```
> 2/3
> 2\3
> 3\2
```

Vous pouvez disposer ces commandes à la suite sur une même ligne, à condition de les séparer par une virgule. L'effet produit par ce qui suit est identique :

```
> 2/3, 2\3, 3\2
```

Par défaut, le format des nombres est fixée à 8 chiffres significatifs. Pour changer ce format, on écrit `format("v",n)`, où $n - 2$ est le nombre de chiffres significatifs maximal¹ souhaités (donc par défaut, n vaut 10). On peut aussi écrire les nombres en format mantisse-exposant. Pour tout réel non nul x , on peut trouver un

¹Le comportement de `format` est quelque peu curieux ; attention de grandes valeurs de n aboutissent à des aberrations.

entier relatif n et un réel a tels que $|a| \in [1, 10[$ et $x = a10^n$. a est la mantisse de x et n son exposant. a et x sont de même signe, et a et n se calculent par les formules $n = E(\log_{10}(|x|))$ puis $a = x10^{-n}$, où E désigne la fonction partie entière et \log_{10} le logarithme de base 10 (on verra plus loin comment les calculer). Par exemple, avec 125.23, la mantisse est 1.2523 et l'exposant 2 puisque $125.23 = 1.2523 \times 10^2$. On peut écrire directement dans SCILAB les nombres sous cette forme, en utilisant `format('e',n)` au lieu de `format('v',n)`. Testez ces commandes avec 1234.56789 ; n'oubliez pas de revenir au format par défaut à l'issue en tapant `format('v',10)`.

On dispose de plusieurs fonctions d'arrondis :

`floor(x)` : plus grand entier inférieur ou égal à x . C'est la partie entière usuelle en mathématique.

`ceil(x)` : plus petit entier supérieur ou égal à x .

`fix(x)` : c'est `floor(x)` si $x \geq 0$, `ceil(x)` sinon.

`round(x)` : entier le plus proche de x .

Exercice 1.2.1 *Essayez ces commandes avec 1.23, -1.23, 1.72, -1.72 (par exemple).*

SCILAB dispose de beaucoup de fonctions usuelles (trigonométriques, logarithmiques, exponentielle...) et de fonctions dites spéciales intervenant en ingénierie (fonctions eulériennes, de Bessel, elliptiques...). A titre d'information, signalons les fonctions trigonométriques `sin`, `cos`, `tan`, ... s'appliquant à un angle en radians, et les fonctions logarithmiques `log` (logarithme népérien), `log10` (logarithme décimal), `log2` (logarithme en base 2), `exp` (fonction exponentielle).

Enfin, pour calculer les factorielles, on sera amené à utiliser la fonction Γ . Cette fonction est en fait définie sur $\mathbb{C} \setminus \mathbb{Z}^-$, mais on utilisera surtout le fait que pour tout entier n , $n! = \Gamma(n + 1)$, qui se calcule dans SCILAB par `gamma(n+1)` (attention au décalage). Cependant, le calcul n'est pas possible pour de grandes valeurs de n ; nous allons utiliser les outils de la notation scientifique et le fait que le logarithme transforme produits en sommes. Ainsi, pour calculer le nombre :

$$P = 500!,$$

il peut être judicieux de constater que :

$$Q = \log_{10}(P) = \sum_{k=1}^{500} \log_{10}(k).$$

Exercice 1.2.2 *Calculez Q par la commande `Q=sum(log10(1:500))` et en déduire l'exposant et la mantisse de P . Combien de chiffres y a-t-il dans P ?*

Les calculs précédents s'étendent au cadre complexe. Le nombre complexe i se note `%i`.

```
> (2+3*%i)/(5+2*%i)
```

Passons aux parties réelles, imaginaires, modules et argument.

```
> a=7-2*%i
> real(a) // partie réelle
> imag(a) // partie imaginaire
> abs(a) // module
> phasemag(a) // argument en degres
> conj(a) // complexe conjugué
> a+conj(a)-2*real(a)
```

Les fonctions réelles s'étendent au cas complexe. Par exemple, le sinus est défini pour tout $z \in \mathbb{C}$ par la série convergente :

$$\sin z = \sum_{k=0}^{+\infty} \frac{(-1)^k z^{2k+1}}{(2k+1)!}.$$

```
> sin(2+%i)
```

On fera attention au logarithme complexe et aux puissances non entières qui sont des fonctions multiformes et qui ne vérifient pas toutes les formules du cas réel :

```
> log(%i^4)-4*log(%i)
```

Dans ce cas, on n'obtient pas zéro comme attendu si la formule $\log(a^4) = 4 \log(a)$ était vraie. La raison en est la suivante : tout nombre complexe non nul s'écrit de manière unique $z = \rho e^{i\theta}$ avec $\theta \in]-\pi, \pi]$ et on pose alors $\log(z) = \log(\rho) + i\theta$. Au vu de ceci, expliquez le résultat obtenu.

1.3 Affectations et désaffectations

La variable prédéfinie `ans` contient le dernier résultat calculé et peut être utilisée :

```
> 5-4
> ans+2
```

Rappelez la dernière ligne par la flèche \uparrow et ré-exécutez la dernière ligne. Que se passe-t-il ? Commentaire.

Il peut être utile d'enregistrer le résultat d'un calcul dans une variable autre que `ans`. On le fait en mettant le nom de la variable à gauche d'un signe égal qui à droite a le résultat du calcul :

```
> a=sin(2)
```

Pour ensuite supprimer l'affectation à la variable nommée ici `a`, on utilise l'instruction `clear` :

```
> clear a
```

Étudiez ce qui suit :

```
> clear // ôte toutes affectations.
> a=2
> a/5
> A/5
```

Comme le montre le dernier exemple, SCILAB distingue majuscules et minuscules.

1.4 Fichiers de données, scripts et fonctions

1.4.1 Fichiers de données

Vous pouvez tout d'abord sauvegarder vos données dans un fichier. Supposons que vous ayez créé deux matrices `A` et `B` et que vous souhaitez les sauvegarder. La commande `save` enregistre ces matrices dans un fichier nommé ici `data.sav`, ces variables peuvent être récupérés ultérieurement :

```
> A=2, B=sqrt(7)
> save('data.sav',A,B)
> clear
> A
> load('data.dat','A','B')
> A
```

1.4.2 Fichiers scripts

Un fichier script est une suite de commandes SCILAB que vous avez tapé dans votre éditeur de texte favori et sauvegardé dans votre répertoire de travail avec une extension qui est par défaut `.sce`. SCILAB dispose de l'éditeur SCIPAD (que l'on peut ouvrir par le menu **Editor**), mais rien ne vous empêche d'en utiliser un autre si vous le souhaitez. L'appel en ligne de commande de ce fichier provoque alors son exécution.

Par exemple, ouvrez l'éditeur et enregistrez les lignes suivantes dans un fichier intitulé `courbe.sce` :

```
x=-2 : 0.01 : 2 ; y=x.^3-1;
plot2d(x,y)
```

Ces lignes sont destinées à tracer la fonction $x \mapsto x^3 - 1$ sur $[-2; 2]$. Revenez à l'environnement de travail et tapez :

```
> exec("courbe.sce");
```

alors vous verrez le graphe se dessiner. La commande `exec` se trouve aussi dans le menu `File`.

Attention :

- après chaque modification d'un script, il faut le charger à nouveau grâce à `exec`,
- dans un fichier script, les variables sont *globales*, ce qui signifie qu'elles sont utilisables une fois le programme exécuté.

Pour expliquer la notion de variable globale, faites l'expérience suivante. On donne à une variable `x` une valeur avant l'exécution du script, puis on lance ce script qui lui affecte une autre valeur ; l'ancienne valeur est écrasée par la nouvelle. Essayez et commentez :

```
> x=8
> exec("courbe.sce");
> x
```

1.4.3 Création d'une fonction

Une fonction peut tout d'abord être créée en ligne de commande. Ainsi pour créer la fonction `phi` telle que $\phi(z) = z^3 - z + 1$, on écrit :

```
> deff('t=phi(z)', 't=z.^3-z+1');
```

et l'on peut alors utiliser directement la fonction :

```
> phi(5)
```

Il est à noter cette fois que les variables sont *locales*, l'appel de `phi` ne modifiera pas les anciennes valeurs de `t` et `z` :

```
> t=123, z=456
> phi(0)
> t,z
```

Pour une fonction plus longue, on peut la rentrer sur plusieurs lignes, mais en général il est recommandé d'écrire les commandes dans un fichier de fonction, qui est un fichier texte d'extension par défaut `sci`, qui commence par le mot clef `function` et se termine par `endfunction`. Il est recommandé de donner le même nom à la fonction qu'au fichier. Par exemple, le fichier suivant `facto.sci` retourne la valeur de la factorielle lorsqu'en entrée, l'utilisateur fournit un entier au moins égal à 1.

Nous allons entrer une fonction `facto` dans ces deux modes, permettant le calcul de la factorielle. La première solution est d'entrer en mode interactif :

```
function res=facto(n)
res=prod(1:n);
endfunction
```

On notera que les invites se sont rapprochées, jusqu'à l'entrée du mot clé `endfunction`. La fonction est alors immédiatement utilisable. On remarque que la première ligne est composée du mot clé `function`, suivi de la variable qui va contenir la sortie, puis du signe `=`, du nom de la fonction puis entre parenthèse la variable d'entrée. Ces variables peuvent être des matrices (cf. plus loin) et s'il n'y en a pas, écrire `[]`.

Si l'on rentre ces lignes dans un fichier à part, il faut l'enregistrer avec l'extension `sci` et reprendre le nom (ici `facto`) de la fonction. L'enregistrement se fait alors dans `facto.sci`'. Une fois ce programme enregistré, pour calculer $12!$, l'utilisateur devra taper en ligne de commande² :

```
> exec("facto.sci") // pour charger la fonction
> facto(12)
```

Attention : après chaque modification d'une fonction, il faut la charger à nouveau grâce à `exec`.

1.5 Calcul vectoriel et matriciel.

Une spécificité de SCILAB est son traitement des matrices. **Cette section est d'ailleurs certainement la plus importante de ce chapitre.** Il est possible d'appliquer de manière simple et en générale efficace en terme de temps de calcul des opérations élément par élément d'une matrice. Si par exemple on veut calculer une somme :

$$\sum_{k=1}^{100} F(k)$$

avec une fonction F explicite (disons $F(t) = \sin(t)/(1 + t^2)$ pour donner un exemple), le point de vue usuel consisterait à suivre l'ordre suivant :

on fait tout d'abord une boucle – pour k variant de 1 à 100, on calcule $F(k)$ – puis on ajoute les valeurs. Même si en SCILAB on peut suivre cette démarche, celle qui est plus dans la philosophie de ce logiciel, et qui est plus efficace en terme de temps et moyen de calcul, consiste plutôt à faire :

on crée un vecteur de dimension 100 contenant les entiers de 1 à 100, on applique élément par élément la fonction F , puis on somme les composantes.

²Je rappelle que `exec` est accessible par les menus

C'est un nouveau mode de pensée auquel vous devez vous habituer pour devenir des utilisateurs efficaces de SCILAB. Mais avant de revenir sur ce point, passons en revue les premières commandes utiles sur les matrices.

1.5.1 Saisie.

Saisie d'un vecteur ligne :

```
> x=[1 3 4 9]
```

Observez l'effet de : sur les exemples suivants :

```
> y=[1:10]
```

```
> z=[1:2:7]
```

```
> z=[15:-2:3]
```

Quel est-il ? Il s'agit donc d'une première commande très utile en vue d'appliquer le principe général exposé dans l'introduction de cette section. Exercez vous si besoin en prenant des petits exemples.

Assez proche, on dispose de `linspace`. Si a et b sont des réels tels que $a < b$ et n un entier naturel non nul, `linspace(a,b,n)` est le vecteur ligne à n composantes séparant l'intervalle $[a, b]$ en $(n - 1)$ intervalles égaux, i.e. le k -ème terme est $a + \frac{k-1}{n-1}(b - a)$.

Saisie d'un vecteur colonne. Un vecteur colonne est vu comme une matrice donc chaque ligne est composée d'un seul élément, et dans SCILAB on sépare les lignes par des points-virgules :

```
> z=[1 ; 3 ; 4 ;9] ;
```

En fin de ligne, le symbole ; permet de ne pas afficher le résultat. x' désigne la transposée dans le cas réel (et la transconjuguée dans le cas complexe).

```
> x'-z
```

Donc $z=x'$. On peut accéder aux éléments individuels :

```
> z(3).
```

Pour un vecteur, on peut obtenir la somme, le produit, la moyenne, etc. de ses coefficients par les commandes `sum`, `prod`, `mean`, etc. Nous verrons d'autres commandes plus loin dans cette section, et nous verrons leurs applications aux matrices. Exemple :

```
> sum(z), prod(z), mean(z) ;
```

Passons aux matrices.

La première matrice à signaler est la matrice vide : `[]`.

On entre les matrices ligne à ligne (espaces ou virgules entre éléments d'une même ligne), puis on change de ligne par un point-virgule.

```
> A=[16 2 3 13 ; 5 11 10 8 ; 9 7 6 12 ; 4 14 15 1]
```

Tester les commandes suivantes, et en déduire le sens des commandes `ones`, `eye`, `zeros` :

```
> eye(3), eye(3,3), eye(3,5)
> ones(3), ones(3,3), ones(3,5)
> zeros(3), zeros(3,3), zeros(3,5)
```

Ces commandes peuvent aussi prendre un vecteur ou une matrice comme entrée au lieu d'un couple de réels. Cela revient à mettre la dimension de la matrice entrée comme argument³. Par exemple, tapez la commande `B=eye(A)` qui va vous créer une matrice B de même taille que A.

La syntaxe s'applique aussi pour entrer des matrices par blocs. Essayez :

```
> a=[ 1 2 ; 4 5 ], b=[3 ; 6], c=[7 8], d=[9]
> [a b ; c d]
```

1.5.2 Premières opérations.

Passons aux déterminants et à l'inverse :

```
> C=A+B; det(C)
> inv(C)
```

On dispose des opérateurs `*` et `^` pour calculer le produit de deux matrices et pour élever à une puissance :

```
> C*A
> A^2
```

Observez les résultats suivants et en déduire le sens de `/` et de `\` pour les matrices carrées de bonnes dimensions et régulières⁴.

```
> A/C-A*inv(C)
> C\A-inv(C)*A
```

Commentaire :

```
> spec(A) // val propres de A
```

³dans `eye(3)`, `ones(3)`, `zeros(3)`, SCILAB interprète le chiffre 3 comme une matrice de taille (1,1), le comportement de MATLAB est différent ici

⁴Il existe des extensions dans d'autres cas, nous en verrons une dans le cadre des moindres carrés

```
> x=[1;1;1;1] // verifions que x est v.p.
> A*x
```

1.5.3 Extraction de sous matrices.

Pour extraire la première ligne de **A**, on écrit :

```
> A(1,:)

```

et pour extraire la seconde colonne, on écrit :

```
> A(:,2)

```

L'élément en troisième ligne et quatrième colonne s'obtient par :

```
> A(3,4)

```

Pour mettre sous forme d'un vecteur colonne les colonnes de **A** les unes après les autres, on écrit :

```
> A(:)

```

Pour obtenir la sous matrice formées des lignes 1 et 3 et des colonnes 1 et 2, on écrit :

```
> A([1 3],[1 2])

```

Enfin, pour supprimer des lignes ou des colonnes d'une matrice, on les remplace par une matrice vide. Par exemple, la commande suivante supprime la première ligne de **C** :

```
> C(1,:)=[]

```

Exercice 1.5.1 *On introduit la matrice $D = A/34$. Vérifier à l'aide de SCILAB que les sommes de chaque ligne et de chaque colonne font 1 (la matrice D peut donc s'interpréter comme une matrice de transition d'une chaîne de Markov). On utilisera la fonction `sum` qui s'applique à une matrice (taper `help sum`). Calculez les puissances successives de D . Que constatez-vous ?*

1.5.4 Opération élément par élément (ou vectorisation).

On en vient à l'une des spécificités de SCILAB signalée en introduction : il est possible d'effectuer des opérations élément par élément, ce qu'il est conseillé de faire dès que le problème s'y prête. Par exemple, la commande :

```
> log(A)

```

calcule le logarithme de chaque élément de la matrice **A**. L'effet d'application d'une fonction usuelle ou spéciale est identique. Par exemple, un vecteur **d1** contenant les exponentielles des nombres 0, 0.001, ..., 0.999, 1 se fera par :

```
> d1=exp(0 : 0.001 : 1) ;

```

(le point-virgule est mis pour éviter l'affichage de **d1**). Créez de même le vecteur **d2** de dimension 1001 dont les composantes sont $\sin(k)$ pour k variant de 0 à 1000.

Pour deux matrices de même dimension $A = (a_{ij})$ et $B = (b_{ij})$, il est possible d'effectuer une multiplication, une division ou une exponentiation élément par élément. Plus précisément, les matrices $(a_{ij}b_{ij})$, (a_{ij}/b_{ij}) et $(a_{ij}^{b_{ij}})$ s'obtiennent respectivement par :

```
> A.*B
> A./B
> A.^B
```

Essayez avec :

```
> A=[16 2 3 13 ; 5 11 10 8 ; 9 7 6 12 ; 4 14 15 1]; B=A/10 ;
```

Ainsi, ne confondez pas les opérations $.*$ et $*$ (ou $./$ et $/$). A l'aide de $d1$ et $d2$, créez le vecteur :

$$d3 = (\exp(k/1000)^{\sin(k)})_{0 \leq k \leq 1000}.$$

Ces opérations s'étendent au cas où l'une des matrices est un nombre. Dans ce cas, tout se passe comme si on avait à la place de ce nombre une matrice de même dimension que l'autre matrice et composée uniquement du nombre en question. Par exemple, pour retrancher 1 à chacune des composantes de $d1$, et élever au cube chacune des composantes de $d2$, il suffit d'écrire :

```
> e1=d1-1 ; e2=d2.^3 ;
```

Exercice 1.5.2 Appliquez ces raccourcis pour créer :

- Un vecteur contenant les carrés des entiers de 1 à 10.
- Un vecteur contenant les 2^p pour p impair variant de 1 à 11.

Remarque 1.5.3 En fait SCILAB accepte, lorsqu'il n'y a pas d'ambiguïté, que l'utilisateur ommette le point pour ces opérations. C'est une habitude que je vous déconseille de prendre, car d'une part ceci n'est pas vrai de tous les logiciels, et d'autre part il est toujours bon de réfléchir précisément à l'opération que l'on souhaitait faire. Je serai exigeant sur ce point à l'examen.

Attention : pour calculer les inverses des éléments d'un vecteur u , il faut écrire $(1)./u$ et non $1./u$, ce dernier étant interprété comme $(1.)/u$ et (si u est en ligne) fournit un vecteur v tel que $uv=1$.

Reprenons l'exemple du calcul de la somme :

$$\sum_{k=1}^{100} F(k) \text{ avec } F(t) = \sin(t)/(1+t^2).$$

Le plus efficace, lorsque F est une fonction s'appliquant élément par élément (par exemple \sin) est de créer le vecteur $k=[1:100]$, de lui appliquer notre fonction

F , puis de sommer à l'aide de `sum` qui pour un vecteur donne la somme des composantes. Ici, opérons autrement. Je donne d'abord une version détaillée pour expliquer puis une version courte (sans affectations inutiles) telle que vous devriez l'écrire.

version détaillée. On génère un vecteur dont les composantes sont les entiers de 1 à 100. On calcule les numérateurs des termes $F(k)$, puis leurs dénominateurs, on fait le quotient élément par élément pour obtenir le vecteur $(F(k))_k$, puis on somme ses composantes :

```
> k=[1:100];
> num=sin(k);
> den=1+k.^2;
> Fk=num./den;
> sum(Fk)
```

version abrégée. Je choisis quand même de créer le vecteur nommé avant `k` car il apparaît deux fois, mais les autres affectations sont inutiles :

```
> k=[1:100];sum(sin(k)./(1+k.^2))
```

Citons un autre exemple. Dans l'esprit de SCILAB, la somme $\sum_{i=1}^{200} i^{-2}$ se calculera par :

```
> sum([1:200].^(-2)) ;
```

Exercice 1.5.4 *En suivant l'esprit des raccourcis SCILAB, écrivez la formule la plus courte possible pour calculer ces expressions (regarder l'aide pour `prod`) :*

1. $1 \times 2 \times \dots \times 20$.
2. $\prod_{k=1}^{100} (1 + 1/k^2)$.

Remarque : dans l'écriture d'une fonction, l'idéal est de profiter au maximum de la vectorisation (revoir l'exemple de la fonction `phi`) afin de pouvoir appliquer la fonction à un vecteur. Ainsi, si j'écris :

```
> deff("y=f(x)","y=x*sin(x)")
```

je ne pourrai pas calculer `f(1:5)`, alors que c'eût été possible en écrivant :

```
> deff("y=f(x)","y=x.*sin(x)")
```

Applications courantes (vecteurs réels) :

1. Si \mathbf{x} et \mathbf{y} sont deux vecteurs (réels) lignes de même taille, on peut calculer la somme $\sum_i x_i y_i$ en posant `x*y'`. Si l'on a affaire à des vecteurs colonnes, il faut transposer le premier `x'*y`. Noter que l'on ne peut pas se tromper dans l'ordre de transposition (s'aider des dimensions).

2. Si \mathbf{x} et \mathbf{y} sont deux vecteurs (réels) lignes de taille respectives $(1, p)$ et $(1, q)$ (p pouvant être ou non égal à q), la matrice de taille (p, q) , $A = (x_i y_j)$ se calcule en posant `A=x'*y`. Là encore, en s'aidant des dimensions, il n'est pas possible de se tromper pour mettre la transposition.

Exercice 1.5.5 *Mettons ceci en pratique. Partons du vecteur ligne $\mathbf{x}=[1 \ 2 \ 3]$.*

1. *Je veux créer la matrice \mathbf{A} de taille $(3,4)$ où toutes les colonnes de \mathbf{A} sont des vecteurs \mathbf{x} mis en colonnes. La commande $\mathbf{x}'*\mathbf{ones}(1,4)$ convient (bien comprendre pourquoi).*

2. *Créer de même la matrice \mathbf{B} de taille $(3,4)$ où toutes les lignes de \mathbf{B} sont des vecteurs $\mathbf{y}=[0.5 \ 1 \ 1.5 \ 2]$.*

3. *A l'aide de \mathbf{A} et \mathbf{B} , créer une matrice $(3,4)$ dont le terme d'indices (i, j) est $x_i^{y_j}$.*

On peut aussi créer des matrices ou des vecteurs dont les composantes sont aléatoires. Par exemple, `rand(p,q)` (ou `rand(p,q,"uniform")`) crée une matrice $p \times q$ dont toutes les composantes suivent une loi uniforme sur $[0;1]$. La commande `rand(p,q,"normal")` crée une matrice dont toutes les composantes suivent une loi normale centrée réduite. Nous verrons plus tard comment créer des matrices dont les coefficients suivent d'autres lois.

1.5.5 Fonctions complémentaires.

Voici quelques autres fonctions utiles sur les matrices :

- > `[m,n]=size(A)` : `m` donne le nombre de lignes et `n` le nombre de colonnes.
- > `length(A)` : donne le produit `m n` (i.e. le nombre d'éléments). > `mean(A)` : retourne la moyenne des éléments de `A`.
- > `st_deviation(A)` : retourne l'écart-type anglo-saxon – on divise par $n - 1$ et non n – des éléments de `A`.
- > `max(A)` : retourne le maximum des éléments de `A`.
- > `min(A)` : retourne le minimum des éléments de `A`.
- > `sum(A)` : retourne la somme des éléments de `A`.
- > `prod(A)` : retourne le produit des éléments de `A`.

Toutes les commandes listées de `mean` à `prod` peuvent s'appliquer ligne à ligne ou colonne à colonne. Par exemple, pour obtenir un vecteur ligne (=row en anglais) dont le i -ème terme est la somme des termes de la colonne i , entrez : `sum(A,'r')`. De même, il est possible de créer un vecteur colonne dont le i -ème terme est la somme des termes de la ligne i , grâce à : `sum(A,'c')`.

Exercice 1.5.6 *Calculez le plus rapidement possible la moyenne des entiers de 1 à 100, de leurs carrés et de leurs cubes.*

1.5.6 Moindres carrés et projections orthogonales.

La commande `\` permet de résoudre un système au sens des moindres carrés. C'est utile si l'on veut appliquer la méthode des Moindres Carrés Ordinaires (par

exemple en économétrie) à un système linéaire :

$$Y = Xb + u$$

où Y est de taille $N \times 1$, X de taille $N \times K$ (avec $N > K$), b de taille $K \times 1$ et u est le vecteur des résidus $N \times 1$.

Rappelons sur un exemple (non issu de l'économétrie) l'origine de ce type de problème. Supposons que vous estimez que trois variables x , y , z soient reliées par une relation du type $z = \alpha x + \beta y^2$ avec α et β inconnues que vous cherchiez à estimer. On observe n triplets (x_i, y_i, z_i) avec $n \geq 2$ (2 est le nombre de paramètres) si possible n beaucoup plus grand. Il est bien entendu peu probable que pour chaque i , on ait une relation exacte $z_i = \alpha x_i + \beta y_i^2$ avec les mêmes α et β . On écrit donc cette relation sous la forme :

$$z_i = \alpha x_i + \beta y_i^2 + u_i,$$

le terme u_i étant à interpréter comme un terme d'erreur. Si l'on pose $b = (\alpha, \beta)'$ (vecteur 2×1), $u = (u_1, \dots, u_n)'$ (vecteur $n \times 1$ dit vecteur des résidus) et $Y = (z_1, \dots, z_n)'$ (vecteur $n \times 1$), on peut écrire notre relation sous la forme :

$$Y = Xb + u$$

où X est une matrice $n \times 2$ dont la i -ème ligne est : $(x_i \ y_i)$. Si l'on cherche le couple (α, β) minimisant la norme 2 du vecteur des résidus (donc minimisant $\sum_{i=1}^n u_i^2$), on obtient l'estimateur des MCO (moindres carrés ordinaires).

Formule théorique et calcul dans SCILAB de l'estimateur des MCO. On suppose que X est de plein rang colonne⁵. Rappelons que dans ce cas, l'estimateur des M.C.O. de b existe, est unique et est donné par :

$$\hat{b} = (X'X)^{-1}X'Y.$$

En SCILAB, il se calcule immédiatement par la séquence :

```
> X\Y
```

Remarque : Etant donné dans \mathbb{R}^n , p vecteurs libres, si l'on note X la matrice dont les colonnes sont ces vecteurs, l'ensemble des Xb est en fait le s.e.v. F engendré par nos p vecteurs. Ainsi, le projeté orthogonal de u sur F est $X\hat{b}$. Utiliser cette remarque pour calculer le projeté orthogonal de $u = (1, 2, 1)'$ sur le s.e.v. engendré par $((4, 5, 6)', (7, 8, 9)')$.

⁵ce qui dans le cas gaussien homocédastique est la C.N.S. d'identifiabilité des paramètres

1.5.7 Valeurs propres et exponentielle de Matrice.

Valeurs propres.

La commande `spec` permet le calcul de valeurs propres et de vecteurs propres.

Dans la syntaxe :

```
> spec(A)
```

la commande `spec` retourne les valeurs propres de la matrice A . Pour avoir aussi les vecteurs propres (dans le cas diagonalisable), on tape :

```
> [P,D]=spec(A)
```

ce qui donne un couple de matrices P et D , la seconde étant diagonale, telles que $AP=PD$. Lorsque la matrice P est inversible, il s'ensuit que la matrice A est diagonalisable, et l'on rappelle que dans ce cas, la i -ème colonne de P donne un vecteur propre associé à la i -ème valeur propre située sur la diagonale de D .

Exercice 1.5.7 Prenez la matrice $A=[16 \ 2 \ 3 \ 13; 5 \ 11 \ 10 \ 8; 9 \ 7 \ 6 \ 12; 4 \ 14 \ 15 \ 1]$ et à l'aide de la commande `spec`, indiquez les éléments propres de A et si A est diagonalisable.

Prenons maintenant l'exemple d'une matrice non diagonalisable.

```
A=[-14 -25 ; 9 16]; [P,D]=spec(A); rcond(P)
```

La faible valeur de `rcond(P)` nous fait penser qu'elle n'est pas inversible. Vérifiez que l'on a bien $AP=PD$ (aux erreurs d'arrondis près), mais tentez le calcul de `inv(P)*A*P-D`.

Exercice 1.5.8 (calcul approché des valeurs propres et vecteurs propres). Supposons que l'on ait une matrice A réelle ayant ses valeurs propres $\lambda_1, \dots, \lambda_n$ satisfaisant $|\lambda_1| > |\lambda_2| \geq |\lambda_3| \geq \dots \geq |\lambda_n|$.

1. Justifier que A est diagonalisable. On note (e_1, \dots, e_n) une base de vecteurs propres de sorte que pour tout i , e_i soit associé à λ_i .
2. Soit $x = \sum_{i=1}^n x_i e_i$ un vecteur non nul arbitraire, tel que $x_1 \neq 0$ (ce qui est le cas, à moins de vraiment mal choisir x). On introduit la suite de vecteurs $(y_p)_p$ définie par $y_p = A^p x$. Donner le terme dominant de y_p quand $p \rightarrow +\infty$, et en déduire une méthode de calcul de $|\lambda_1|$.
3. Déterminer la direction limite de $y_p / \|y_p\|$, et en déduire comment calculer la direction de e_1 .
4. Soit B une matrice, on pose $A = {}^t B B$. Expliquer pourquoi A est diagonalisable et ses valeurs propres sont des réels positifs. Appliquer la méthode précédente à A après avoir choisi une matrice B aléatoire d'ordre 3 (faire $B=\text{rand}(3,3)$; $A=B' * B$). La convergence semble-t-elle rapide sur votre exemple ? Quelle est-elle en théorie ?

Exponentielle, logarithme et racine carrée d'une matrice.

L'exponentielle d'une matrice carrée A est définie par la formule :

$$\exp(A) = \sum_{n=0}^{+\infty} \frac{A^n}{n!}$$

et intervient régulièrement en mathématiques, par exemple dans la résolution des équations différentielles linéaires (cf. la section sur la dynamique)). La matrice $\exp(A)$ n'est pas la même que la matrice dont les coefficients sont les exponentielles des coefficients de A , et se calcule par la fonction `expm`.

Attention : il ne faut donc pas confondre l'expression mathématique $\exp(A)$ qui se calcule en SCILAB par `expm(A)` avec l'expression SCILAB `exp(A)`.

Exercice 1.5.9 Prenez une matrice carrée d'ordre 2 quelconque, et calculez son exponentielle. Comparez-la à la matrice dont les coefficients sont les exponentielles des coefficients de A .

On dispose de même d'une fonction `sqrtm` et d'une fonction `logm` retournant respectivement une racine carrée et un logarithme de la matrice entrée. La première s'applique à une matrice symétrique, la seconde à une matrice symétrique ou diagonalisable (sans quoi les risques de réponses farfelues sont grands). Ces fonctions dépassent le cadre de notre programme.

1.6 Graphiques

Sauf pour l'utilisation basique de `xbasc()` et de `plot2d`, ce chapitre est hors programme, il vous sert de référence.

1.6.1 Ce qu'il faut savoir.

La commande de base utilisée est la commande `plot2d`. Dans sa syntaxe la plus simple, elle prend deux vecteurs x et y de même dimension en entrée et retourne un graphique où sont reliés continûment les points (x_i, y_i) . On peut s'en servir pour tracer des courbes paramétriques ou des graphes de fonctions (i.e. des équations de la forme $y = f(x)$).

Commençons par des graphes de fonctions. On veut tracer sur un même graphique les graphes des fonctions $x \mapsto x^2$ et $x \mapsto \sqrt{x}$. Le domaine de variation de x est l'intervalle $[0; 2]$. On crée tout d'abord trois vecteurs (remarquez que l'on utilise les raccourcis vectoriels sur lesquels on a déjà insisté) x, y, z :

```
> x=0:0.005:2; y=x.^2; z=x.^(1/2);
```

puis on trace avec cela le graphe de la fonction carrée.

```
> plot2d(x,y)
```

Une nouvelle fenêtre s'ouvre dans laquelle se trace le graphique. Revenez dans la ligne de commande SCILAB. Par défaut avec `plot2d`, le second graphique est superposé (alors qu'avec `plot` il y aurait effacement). On peut donc superposer la seconde courbe par :

```
> plot2d(x,z)
```

Pour effacer la fenêtre graphique on dispose de `xbasc()`.

Enfin, lorsque `x` et `y` sont des matrices (non vecteurs) de même taille, on trace simultanément plusieurs courbes, la i -ème étant la i -ème colonne de `y` en fonction de la i -ème colonne de `x` :

```
> x=linspace(0,2*pi,50); y=x.*sin(x); n=5; X=ones(n,1)*x; Y=[1:n]'*y;
> xbasc(); plot2d(X,Y)
> X=X'; Y=Y';
> xbasc(); plot2d(X,Y) // commenter la différence avec ce qui précède
```

Tout le reste de la section est hors programme.

1.6.2 Courbes du plan (suite).

Les fonctions graphiques sont très riches, je ne pourrais malheureusement pas tout détailler, si besoin n'hésitez pas à consulter l'aide.

A titre d'illustration de courbes paramétriques, on se propose de tracer :

$$\begin{cases} x(t) = \cos(2\pi t) \\ y(t) = \cos(3\pi t) \end{cases}$$

pour t parcourant $[0; 1]$. On commence par effacer le graphique précédent par :

```
> xbasc();
```

Il faut alors entrer :

```
> t=0:0.01:1 ; x=cos(2*pi*t) ; y=cos(3*pi*t);
> plot2d(x,y)
```

Signalons que dans sa nouvelle syntaxe, la commande `plot2d` peut prendre plusieurs options. Sa syntaxe est alors :

```
> plot2d(x,y,argplot)
```

où `x` et `y` sont deux matrices de même taille, et `argplot` une suite d'affections de la forme $key_1 = val_1, \dots, key_p = val_p$. Si `x` et `y` sont des vecteurs, on place les points (x_i, y_i) et par défaut, on les relie continûment. Sinon, on trace plusieurs courbes, la i -ème courbe étant le tracer de la i -ème colonne de `y` en fonction de

la i -ème colonne de \mathbf{x} .

Les arguments key_i peuvent prendre plusieurs valeurs :

- **style** est un vecteur ligne de taille q égale au nombre de colonnes de \mathbf{y} . Par défaut, **style**=1:q. Si le style est strictement positif, il définit la couleur de la courbe (noir : 1, bleu clair 4 et 12, bleu foncé 2, 9, 10 et 11, vert : 3, 13, 14, 15, marron : 25 et 27, etc.). Si le style est nul, on trace juste les points et si le style est strictement négatif, les points sont remplacés par des marques.
- **strf** est une chaîne de trois chiffres, **strf**="xyz". **x** vaut 0 ou 1 et régit la présence de légendes (oui si 1). **y** varie entre 1 et 8 et contrôle le calcul des bornes du dessin. Les valeurs courantes sont **y**=8 (bornes calculées à partir des courbes) et **y**=0 (on conserve les bornes du dessin précédent). **z** contrôle l'affichage des axes, les valeurs courantes étant **z**=1 (axes dessinés, l'axe des ordonnées est placé à gauche), et **z**=0 (pas d'axes dessinés, utile notamment pour superposer).
- **leg** définit les légendes (quand le premier chiffre de **strf** est 1). C'est une chaîne de caractères **leg**="leg1@...@legp", où **p** est le nombre de courbes. La chaîne **legi** est alors la légende de la courbe i .
- **nax** permet de graduer les axes (quand le premier chiffre de **strf** est 1). C'est un quadruplet **nax**=[nx,Nx,ny,Ny]. L'axe des **x** est divisés en **Nx** graduations, elles-mêmes divisées en **nx** sous-graduations (et idem pour **y**).

Exemples :

```
> x=linspace(0,2*pi,50); y=x.*sin(x); n=5; X=ones(n,1)*x; Y=[1:n]'*y;
> xbasc(); plot2d(X,Y,style=-[1:n])
> xbasc(); plot2d(x,y)
> plot2d(x,2*y)
> xbasc(); plot2d(x,y)
> xbasc(); plot2d(X,Y,strf="181",leg="c1@c2@c3@c4@c5")
```

Remarque : pour plus d'informations, n'hésitez pas à consulter l'aide.

1.6.3 Courbes de l'espace.

Soit à tracer la courbe paramétrique :

$$\begin{cases} x(t) = \sin(t) \\ y(t) = t^2 \\ z(t) = \tan(t) \end{cases}$$

pour t parcourant $[0; 1]$. On crée les vecteurs permettant le tracer :

```
> t=0:0.01:1; x=sin(t); y=t.^2; z=tan(t);
```

La commande permettant le tracé est :

```
> xbaso();param3d(x,y,z)
```

1.6.4 Surfaces de l'espace.

On va commencer par tracer une surface paramétrique, ce qui est le cas le plus général. Supposons que l'on veuille tracer la sphère grâce à une représentation paramétrique :

$$\begin{cases} x = \cos(u) \cos(v) \\ y = \cos(u) \sin(v) \\ z = \sin(u) \end{cases}$$

où $u \in [-\pi, \pi]$ et $v \in [0; 2\pi]$. On définit tout d'abord l'équation :

```
deff("[x,y,z]=sphere(u,v)",["x=cos(u).*cos(v)","y=cos(u).*sin(v)","z=sin(u)"])
```

ensuite, on précise le domaine de variation des paramètres :

```
> u=linspace(-%pi,%pi,50); v=u+%pi;
```

la commande suivante crée des matrices \mathbf{x} , \mathbf{y} , \mathbf{z} nécessaires pour le tracer :

```
> [x,y,z]=eval3dp(sphere,u,v);
```

enfin on trace la surface :

```
> xbaso(); plot3d(x,y,z);
```

Lorsque la surface est d'équation cartésienne $z = f(x, y)$, il suffit de prendre \mathbf{x} et \mathbf{y} comme paramètres. Par exemple, pour tracer la surface :

$$z = x^2 - y^2/2$$

pour x et y variant dans $[-2; 2]$, on définit en premier une fonction paramétrée de \mathbb{R}^2 vers \mathbb{R}^3 pour tracer la fonction que l'on va nommer \mathbf{fct} .

```
> deff("[x,y,z]=fct(u,v)",["x=u","y=v","z=u.^2-v.^2"]);
```

```
> u=-2:0.1:2; v=u;
```

```
> [x,y,z]=eval3dp(fct,u,v);
```

```
> xbaso(); plot3d(x,y,z)
```

1.6.5 Subdivision de la fenêtre.

Il est possible de partager l'écran en plusieurs parties pour mettre ensemble plusieurs graphiques, mais sans les superposer. La commande `subplot(m,n,p)` partage l'écran en un tableau de taille $m \times n$ (si ce n'est déjà fait) et place le graphique suivant en position p (en comptant ligne par ligne). Par exemple, pour tracer les fonctions $x \mapsto x^k$ pour $k \in \{1, \dots, 4\}$ sur des graphiques distincts, on

peut taper :

```
> x=-1:0.05:1 ; xbasec();
> y=x ; subplot(2,2,1) ; plot2d(x,y) ;
> y=x.^2 ; subplot(2,2,2) ; plot2d(x,y) ;
> y=x.^3 ; subplot(2,2,3) ; plot2d(x,y) ;
> y=x.^4 ; subplot(2,2,4) ; plot2d(x,y) ;
```

Tapez ces commandes et observez l'effet.

1.7 Booléens et opérateurs relationnels.

Un Booléen est une variable pouvant prendre deux valeurs : vraie (true en anglais) et fausse (false). Dans Scilab, vous écrivez respectivement %t et %f et dans les réponses du logiciel vous verrez apparaître T et F. Pour les calculs avec des nombres, il est convenu que T=1 et F=0.

Les opérateurs relationnels sont :

< > <= >= == ~=

et signifient respectivement : strictement inférieur, strictement supérieur, inférieur ou égal, supérieur ou égal, égal et différent (\sim est obtenu par la combinaison des touches AltGr et 2). Le résultat est 1 si l'assertion est vraie, 0 si l'assertion est fausse. Dans le cas de deux matrices de même dimension, ils donnent une matrice de même dimension où les relations sont vérifiées élément par élément. Dans le cas d'une matrice et d'un nombre, tout se passe comme si le nombre était une matrice de même taille que l'autre et dont tous les coefficients sont ce nombre.

On dispose des opérateurs logiques :

& | ~

signifiant respectivement **et**, **ou**, **non** (| est obtenu par la combinaison des touches AltGr et 6). Les syntaxes sont les suivantes : si A et B sont deux matrices de même format⁶, A & B (resp. A | B, retourne une matrice de même format dont le terme d'indices ij est 1 si et seulement si chacun (resp. au moins l'un) des termes ij des matrices A et B est non nul (et 0 sinon).

Exemple : essayez :

```
> x=[0 1 2]; y=[2 1 0]; x&y, x|y
```

Soit à tracer sur $[-2; 2]$ la fonction :

$$f(x) := \begin{cases} x^2 - 1 & \text{si } x \geq 0 \\ 0 & \text{sinon} \end{cases}$$

⁶Le cas d'une matrice et d'un nombre vous est maintenant familier.

Une solution peut être fournie par la liste d'instructions :

```
> x=-2: 0.01: 2;
> y=(x.^2-1).*(x>=0) ;
> plot2d(x,y)
```

Expliquons la seconde ligne. Quand x_i est positif, $(x_i \geq 0)$ est vraie donc donne la valeur T (i.e. 1 pour la multiplication), et sinon donne F (i.e. 0 pour la multiplication).

Voyons un autre exemple (avec un piège celui-ci). Soit à tracer la fonction :

$$f(x) = \begin{cases} x^2 + 1 & \text{si } x \geq 0 \\ \cos(x) & \text{si } x \leq 0 \end{cases}$$

Une solution est fournie par la liste d'instructions :

```
> x=-2: 0.01: 2;
> y=(x.^2-1).*(x>=0)+((cos(x)).*(x<0)) ;
> plot2d(x,y)
```

où cette fois il faut noter que j'ai dû mettre $x < 0$ et non $x \leq 0$ pour éviter de compter deux fois le cas $x = 0$. Essayez de bien comprendre ces exemples avant de traiter l'exercice suivant :

Exercice 1.7.1 Tracez la fonction définie sur $[0; 2\pi]$ par :

$$f(x) := \begin{cases} \sin(x) & \text{si } |\sin(x)| < 0.5 \\ 0.5 & \text{si } \sin(x) \geq 0.5 \\ -0.5 & \text{si } \sin(x) \leq -0.5 \end{cases} .$$

Vous pourrez superposer avec des symboles la fonction sinus (pour laquelle vous parcourrez l'axe des abscisses par pas de 0.2).

1.8 Boucles et structures conditionnelles.

On va illustrer ceci avec le calcul de factorielle 10, c'est-à-dire du produit $1 \times \dots \times 10$ (noté en général $10!$). Les solutions proposées ici ne sont pas les plus efficaces avec SCILAB, mais ont pour but d'illustrer le propos.

1.8.1 Boucles for.

La syntaxe d'une boucle `for` est la suivante :

```
for compteur = A,
instructions
end
```

où A est une matrice (un vecteur le plus souvent). Le `compteur` prend successivement les valeurs des coefficients de la matrice A . Le cas le plus courant est celui où $A = \text{debut}:\text{pas}:\text{fin}$, avec comme d'habitude, un pas qui peut être omis s'il

est de 1. Appelons z la variable dans laquelle nous allons stocker la réponse. Il ne faut pas oublier, et c'est une erreur classique en programmation, d'initialiser la variable. Voici donc le script correspondant au calcul de factorielle 10 par une boucle for :

```
z=1 ; for i=2:10 , z=z*i; end; z
```

Le dernier z est là pour provoquer l'affichage du résultat.

Remarque : vous pouvez, et cela est conseillé, rentrer les commandes sur plusieurs lignes. SCILAB n'effectuera aucune opération tant que vous n'aurez pas entré le `end`⁷. Cette remarque est valable pour les suivants.

Voici un script où l'on itère 100 fois la fonction logistique $x \mapsto 4x(1-x)$:

```
x=sqrt(2)/2 // initialisation
for i=1:100 // on débute ici la boucle qui sera répétée 100 fois
x=4*x*(1-x);
end // fin de la boucle
x,i // on affiche i pour vérifier
```

Exercice 1.8.1 On considère la matrice de taille $(4N) \times 6$ suivante :

$$A_N = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 2 & 1 & 0 & 0 & 0 \\ 1 & 2 & 0 & 1 & 0 & 0 \\ 1 & 2 & 0 & 0 & 1 & 0 \\ 1 & 2 & 0 & 0 & 0 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & N & 1 & 0 & 0 & 0 \\ 1 & N & 0 & 1 & 0 & 0 \\ 1 & N & 0 & 0 & 1 & 0 \\ 1 & N & 0 & 0 & 0 & 1 \end{pmatrix}$$

issue d'un problème économétrique. Construire la matrice en profitant des moyens SCILAB. On pourra proposer une boucle, en remarquant comment passer de A_{k-1} à A_k .

1.8.2 Instruction while.

La syntaxe d'une instruction `while` est la suivante :

```
while conditions,
```

⁷D'ailleurs les invites sont rapprochées tant que `end` n'est pas entré

```
instructions
end,
```

ce qui signifie que l'on répète les instructions tant que les conditions sont vraies. Notre problème s'écrit avec l'instruction while :

```
i=1 ; z=1 ; while i<=10, z=z*i; i=i+1; end; z
```

Exercice 1.8.2 *Exécutez à nouveau cette ligne en enlevant l'affectation $i=1$, et expliquez ce qui s'est passé.*

Revenons à notre fonction logistique. Si $x_0 = \sqrt{2}/2$, on veut connaître le premier indice k pour lequel $x_k \geq 0.999$:

```
x=sqrt(2)/2 // initialisation
i=1 // compteur
while(x<0.999) // tant que x<0.999 faire ce qu'il y a jusqu'au end
x=4*x*(1-x); // ici on calcule  $x_i$ 
i=i+1 // le compteur augmente, il devient n+1 au dernier coup
end // fin de la boucle
i-1 // on affiche n
```

1.8.3 Condition if.

La syntaxe est :

```
if conditions,
then instructions (si les conditions sont satisfaites)
else
instructions (si les conditions ne sont pas satisfaites)
end
```

La partie `else` est facultative. Le mot-clef `then` peut être remplacé par une virgule ou un passage à la ligne (que l'on a fait ici pour plus de lisibilité). De plus, si l'on veut remettre des conditions dans le cas où la première série de conditions n'est pas satisfaite, on dispose d'une instruction `elseif` (en un seul mot) dont la syntaxe est :

```
if conditions1
then instructions1 (si les conditions1 sont satisfaites)
elseif conditions2
then instructions2 (si les conditions2 sont satisfaites)
```

...

```
elseif conditionsp
then instructionsp (si les conditionsp sont satisfaites)
else
instructions (dans les autres cas)
end,
```

Cette fois, on part d'un entier N . Lorsqu'il est pair, on le divise par 2 et s'il est impair, on calcule $3N + 1$, et on arrête le processus lorsqu'on arrive à 1 :

```
N=247 // par exemple
i=1 // compteur du nombre d'étapes pour arriver à 1
while(N>1)
if (modulo(N,2)==0)
test
then N=N/2;
else N=3*N+1
i=i+1
end
end, i-1
```

1.8.4 Une application aux mathématiques financières.

Voici une petite application simple de la boucle `for` et des simplifications matricielles apportées par SCILAB. Le problème est de construire un tableau d'amortissement d'un emprunt.

Le principe est le suivant. Supposons que l'on emprunte sur n mois une somme S au taux mensuel i , et que l'emprunt donne lieu à des mensualités M_1, \dots, M_n en fin de chaque mois. Ces mensualités doivent satisfaire :

$$S = \sum_{t=1}^n \frac{M_t}{(1+i)^t}.$$

Par exemple, dans le cas de mensualités constantes, un calcul simple montre qu'elles valent :

$$M = \frac{Si}{1 - \frac{1}{(1+i)^n}}.$$

Le principe est qu'à chaque fin de mois, la mensualité paie tout d'abord les intérêts sur le capital restant dû, puis le reste de la mensualité constitue l'amortissement du capital, c'est-à-dire est diminué du capital restant dû.

Ecrivez une fonction prenant en entrée n , i et S et dressant un tableau d'amortissement où les colonnes correspondent respectivement au capital restant dû, à la mensualité, à la partie de celle-ci payant les intérêts puis à l'amortissement. Pour vous aider, voici comment on remplit la première ligne, après avoir calculé la mensualité M : on met en première colonne $S_1 = S$, dans la seconde $M_1 = M$ (la seconde contient toujours M dans un cas de mensualité constante), dans la troisième $I_1 = S_1 i$, dans la quatrième $A_1 = M - I_1$. On continue ainsi sauf que bien entendu $S_2 = S_1 - A_1$. Application : rédigez le tableau d'amortissement pour

un emprunt de 100000 sur 48 mois au taux mensuel de 0.00487 (représentant un taux annuel de 6%).

Voici maintenant, dans le cas de mensualités constantes, une solution plus rapide, dans laquelle vous vous efforcerez d'utiliser les spécificités de SCILAB :

1. Vérifiez la relation de récurrence $S_{t+1} = (1 + i)S_t - M$. Il s'agit d'une suite arithmético-géométrique dont on peut démontrer que l'expression est :

$$S_t = (1 + i)^{t-1}(S - M/i) + M/i.$$

Créez un vecteur colonne dont les composantes sont les S_t .

2. Créez le vecteur colonne des mensualités et utilisez ces deux vecteurs pour créer la fin du tableau.

1.9 Polynômes et fractions rationnelles.

Cette section sert de référence, elle n'est pas au programme.

On peut faire quelques opérations sur les polynômes à l'aide de SCILAB. Par défaut, la variable s'écrit `%s`. Je signale juste quelques commandes, voir `apropos poly` pour plus d'informations.

Tout d'abord, si v est un vecteur, `poly(v, 'x')` (ou `poly(v, 'x', 'r')`) crée le polynôme de variable x dont les racines sont les éléments de v . Le polynôme de variable x dont les coefficients (par ordre croissant) sont les éléments de v s'obtient par `poly(v, 'x', 'c')`. Si P est un polynôme, on peut réciproquement obtenir ses coefficients (resp. ses racines, resp. ses facteurs irréductibles sur \mathbb{R}) par `coeff(P)` (resp. `roots(P)`, resp. `factors(P)`).

Exemples :

```
> v=[1,2,3]
> P1=poly(v, 'z'), P2=poly(v, 'z', 'c')
> roots(P1), coeff(P1)
> roots(P2), coeff(P2)
> factors(%s^4-1)
```

On commentera le dernier résultat.

1.10 Gestion du temps

Pour évaluer le temps d'exécution d'un algorithme, on peut utiliser la commande `timer()`. On encadre les commandes dont on veut évaluer le temps d'exécution entre deux `timer()`, le premier déclenche un chronomètre et le second l'arrête. Exemple :

```
> timer(); sum(log10(1:10000));timer()
```

Vous pouvez comparer avec :

```
> timer(); s=0; for i=1:10000; s=s+log10(i); end;timer()
```

pour apprécier l'efficacité des raccourcis SCILAB.

Chapitre 2

Gestion des erreurs.

2.1 Erreurs relatives et absolues.

Cette section n'a pas encore été rédigée, elle sera traitée en détail en cours. Voici deux exercices d'application. Vous pouvez consulter le chapitre 13 de Mathématiques L2, chez Pearson (disponible à la bibli de Tolbiac).

Exercice 1. On note $x_1 = 1000$, $x_2 = \dots = x_{12} = 1$. Supposons que l'on travaille à trois chiffres significatifs. On veut calculer $s = \sum_{i=1}^{12} x_i$.

1. On calcule la suite $s_1 = x_1$, $s_j = x_j + s_{j-1}$, de sorte que $s = s_{12}$. Que donne un logiciel travaillant à trois chiffres significatifs ? Est-ce correct (à trois chiffres) ?
2. Mêmes questions en considérant la suite $s'_1 = x_{12}$, $s'_j = s'_{j-1} + x_{13-j}$ (de sorte que $s = s'_{12}$).

Exercice 2. On donne une valeur approchée de $\pi \approx 3,14159$.

1. Donner une valeur approchée de π à 10^{-2} près. Et à quatre chiffres significatifs.
2. On approche $\sqrt{\pi}$ par $\sqrt{3,14159}$. Donner une estimation de l'erreur commise. Vérifier dans SCILAB.
3. On cherche à évaluer $\sqrt[3]{\pi}$ à trois chiffres significatifs. Quelle précision doit-on choisir pour π ?

2.2 Propagation des erreurs.

Voici une première illustration¹ du phénomène de propagation des erreurs.

Considérons la suite d'intégrales, pour $n \geq 1$:

$$I_n = \int_0^1 \frac{t^{n-1}}{-10+t} dt.$$

¹empruntée à l'excellent livre de J.P. Demailly, Analyse Numérique et Equations Différentielles.

On remarque facilement les faits suivants :

- $\forall n, \quad -1/(9n) \leq I_n \leq 0.$
- $\forall n, \quad I_{n+1} = 10I_n + 1/n.$

Comme $I_{n+1} \leq 0$, la relation de récurrence combinée avec la première inégalité montre qu'en fait :

$$\forall n, \quad -\frac{1}{9n} \leq I_n \leq -\frac{1}{10n}$$

soit :

$$\forall n, \quad 9 \leq -90nI_n \leq 10.$$

De plus, on sait que $I_1 = \ln(9/10)$.

Exercice 2.2.1 *A l'aide de SCILAB, de la relation de récurrence et de I_1 , calculez les premiers termes de la suite $(I_n)_n$ et indiquez à partir de quelle valeur de n l'encadrement $9 \leq -90nI_n \leq 10$ est mis en défaut.*

Une autre piste pour calculer I_n serait de poser $u = t - 10$ et d'appliquer la formule du binôme. On obtient ainsi, pour tout $n \geq 2$:

$$I_n = 10^{n-1} \left[\ln(9/10) - \sum_{k=1}^{n-1} (-1)^k \frac{C_{n-1}^k}{k} (1 - (9/10)^k) \right]$$

Calculez ainsi I_{20} et que constatez-vous ? Commentaires (indication : combien de chiffres a le coefficient binomial C_{19}^{10} ?

A titre d'information, pour vérifier vos programmes, voici les valeurs approchées par troncature que l'on obtient pour les premières valeurs :

n	I_n	$-90nI_n$
1	-0.105	9.48
2	-0.053	9.64
3	-0.036	9.73

Expérimentalement, on constate que pour calculer I_{20} , il est meilleur de partir de l'égalité (fausse) $I_{40} = 0$ et de remonter à I_{20} par la récurrence renversée : $I_n = (I_{n+1} - 1/n)/10$. Expliquez pourquoi.

2.3 Erreurs dans les systèmes linéaires.

De très nombreux problèmes scientifiques, une fois modélisés et éventuellement linéarisés, se ramènent à un problème de résolution d'un système linéaire ou à un problème de valeurs propres. Le but de cette section est de vous introduire au problème des erreurs dans le premier problème.

2.3.1 Normes vectorielles et matricielles.

Cette section est introductive à la suivante. Heuristiquement, les normes sont des indicateurs de l'ordre de grandeur d'un vecteur ou d'une matrice. Chercher à minimiser un vecteur revient à minimiser sa norme. Rappelons que l'on dispose de plusieurs normes, équivalentes entre elles (car on est en dimension finie).

Commençons par les normes vectorielles. Si x est un vecteur de \mathbb{R}^n ou de \mathbb{C}^n , on a pour habitude de considérer plusieurs normes usuelles :

$$\|x\|_\infty = \max_i |x_i|$$

$$\|x\|_p = \left(\sum_{i=1}^n |x_i|^p \right)^{1/p} \quad (p \geq 1)$$

et la seconde étant considérée en général pour $p = 1$ ou $p = 2$. SCILAB retourne ces normes pour un vecteur \mathbf{x} à l'aide respectivement des commandes :

```
> norm(x,%inf) // ou bien norm(x,'inf')
```

```
> norm(x,p)
```

Dans le second cas, lorsque $p=2$, on peut omettre le p . Ainsi, `norm(x)` est identique à `norm(x,2)`.

On peut aussi mettre sur l'espace des matrices plusieurs normes. Les matrices représentent des applications linéaires. Si les espaces de départ sont munis de normes (que l'on notera toutes $\|\cdot\|$ pour éviter de surcharger), il peut être astucieux de mettre la norme subordonnée à ces normes définie par, pour toute matrice A :

$$\|A\| := \sup_{x \in \mathbb{C}^n, x \neq 0} \frac{\|Ax\|}{\|x\|}.$$

On notera que même dans le cas de matrices réelles, on prend le supremum sur les vecteurs non nuls de \mathbb{C}^n . Cette norme subordonnée a de bonnes propriétés :

- $\|I\| = 1$, où I est la matrice identité.
- $\|AB\| \leq \|A\| \|B\|$.

Lorsque les espaces de départ et d'arrivée sont munis de leurs normes p (p fini ou infini), SCILAB calcule la norme subordonnée par : `norm(A,p)`, avec toujours la possibilité de sous-entendre le p lorsqu'il vaut 2. Signalons d'ailleurs que pour $p \in \{1, 2, \infty\}$, cette norme a une expression explicite :

$$\|A\|_1 = \max_j \sum_i |a_{ij}|$$

$$\|A\|_\infty = \max_i \sum_j |a_{ij}|$$

$$\|A\|_2 = \sqrt{\text{plus grande valeur propre de } (A^*A)}.$$

Exercice 2.3.1 Prenez une matrice A carrée d'ordre 3 et proposez un script SCILAB donnant un minorant de $\|A\|_2$ en appliquant la définition à 10 vecteurs aléatoires. Comparez à la vraie valeur.

2.3.2 Effets des incertitudes.

Si les matrices A et b ne sont pas connues exactement, au lieu de résoudre le système :

$$Ax = b$$

on résout en réalité un système :

$$(A + \Delta A)y = (b + \Delta b)$$

où ΔA et Δb représentent les incertitudes, supposées de norme petites vis-à-vis de celles de A et b . Notons $x + \Delta x$ la solution du second système. Il se peut que même si ΔA et Δb sont de normes très petites (vis-à-vis de celles de A et b), que Δx soit très grand. Un tel système est dit mal conditionné.

Contrairement à ce que l'on pense en première intuition, un bon critère de conditionnement n'est pas le déterminant de la matrice. On trouve dans la littérature des contre-exemples à ce sujet. Je pense que le plus simple pour comprendre est encore de choisir un système linéaire de ce type, avec λ très grand :

$$\begin{cases} \lambda x_1 = y_1 \\ x_2/\lambda = y_2 \end{cases} .$$

La matrice de ce système linéaire, de déterminant 1, a une norme très grande devant la matrice :

$$\Delta A = \begin{pmatrix} 0 & 0 \\ 0 & -\frac{1}{2\lambda} \end{pmatrix}$$

qui pourtant modifie considérablement la valeur de x_2 (en la multipliant par 2).

Citons toutefois un exemple célèbre dû à R. S. Wilson. Considérons les trois systèmes linéaires "proches" :

$$AX = b, \quad AX = b + \Delta b, \quad (A + \Delta A)X = b$$

avec :

$$A = \begin{pmatrix} 10 & 7 & 8 & 7 \\ 7 & 5 & 6 & 5 \\ 8 & 6 & 10 & 9 \\ 7 & 5 & 9 & 10 \end{pmatrix}, \quad \Delta A = \begin{pmatrix} 0 & 0 & 0.1 & 0.2 \\ 0.08 & 0.04 & 0 & 0 \\ 0 & -0.02 & -0.11 & 0 \\ -0.01 & -0.01 & 0 & -0.02 \end{pmatrix}, \quad b = \begin{pmatrix} 32 \\ 23 \\ 33 \\ 31 \end{pmatrix},$$

$$\Delta b = \begin{pmatrix} 0.1 \\ -0.1 \\ 0.1 \\ -0.1 \end{pmatrix}$$

dont les solutions respectives sont :

$$\begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}, \quad \begin{pmatrix} 9.2 \\ -12.6 \\ 4.5 \\ -1.1 \end{pmatrix}, \quad \begin{pmatrix} -81 \\ 137 \\ -34 \\ 22 \end{pmatrix}.$$

On constate qu'elles sont très éloignées, alors que A est à coefficients entiers, de déterminant 1 (son inverse est donc aussi à coefficients entiers) ! Analysons plus précisément ceci.

Etant fixée une norme sur l'ensemble des vecteurs, qui donne une norme matricielle subordonnée, on peut démontrer que la bonne notion est *le conditionnement de la matrice* (relatif à la norme choisie), qui est défini pour toute matrice inversible A par :

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

On peut démontrer que ce nombre est toujours supérieur ou égal à 1. Un système mal conditionné en est un pour lequel ce nombre est très grand.

Que le conditionnement réponde bien à la question posée se déduit de la formule suivante (voir exercice final) :

$$\text{Si } \|\Delta A\| < \frac{1}{\|A^{-1}\|}, \text{ alors } \frac{\|\Delta x\|}{\|x\|} \leq \frac{\text{Cond}(A)}{1 - \|A^{-1}\| \cdot \|\Delta A\|} \left[\frac{\|\Delta A\|}{\|A\|} + \frac{\|\Delta b\|}{\|b\|} \right].$$

Enfin, en SCILAB, voici les commandes permettant de calculer le conditionnement d'une matrice : `cond(A)` calcule le conditionnement de la matrice A relativement à la norme 2 (pour les autres, le calculer à partir de la définition et de `norm`). `rcond(A)` donne une estimation de l'inverse de $\text{Cond}_1(A)$, un système mal conditionné donne donc un nombre proche de 0. Il arrive que si A est une matrice non inversible et que l'on demande `inv(A)`, on obtienne un message d'erreur faisant apparaître ce nombre en disant que le système est très mal conditionné. **Si un tel message apparaît, il y a toutes les chances pour que votre matrice soit non inversible et que l'inverse proposé soit farfelu** (il arrive que SCILAB propose quelque chose en raison des erreurs d'arrondis).

Exercice 2.3.2 Analysez l'exemple de Wilson.

Venons-en à la démonstration de la formule :

Exercice 2.3.3 On veut établir la formule d'erreur. Pour cela, on considère une matrice carrée inversible d'ordre n , A , un vecteur non nul $n, 1$, b et l'on note x la solution du système linéaire :

$$Ax = b.$$

1. On admet que si S est une matrice carrée d'ordre n satisfaisant $\|S\| < 1$, alors $(I + S)^{-1}$ est inversible, et que de plus (ces majorations seront utiles dans la question suivante) :

$$\|(I + S)^{-1}\| \leq \frac{1}{1 - \|S\|},$$

$$\|(I + S)^{-1} - I\| \leq \frac{\|S\|}{1 - \|S\|}.$$

Soit maintenant une matrice ΔA telle que $\|\Delta A\| < \frac{1}{\|A^{-1}\|}$. Montrer que $\|A^{-1}\Delta A\| < 1$, et que $A + \Delta A$ est inversible, et enfin que :

$$(A + \Delta A)^{-1} = (I + A^{-1}\Delta A)^{-1}A^{-1}.$$

2. On suppose toujours que $\|\Delta A\| < \frac{1}{\|A^{-1}\|}$, on se donne Δb un vecteur de taille n , et l'on note $x + \Delta x$ l'unique solution du système :

$$(A + \Delta A)(x + \Delta x) = b + \Delta b.$$

Le but est d'estimer $\frac{\|\Delta x\|}{\|x\|}$ en fonction de $\frac{\|\Delta A\|}{\|A\|}$ et de $\frac{\|\Delta b\|}{\|b\|}$.

2.a. Montrer que :

$$\Delta x = [(I + A^{-1}\Delta A)^{-1} - I] A^{-1}b + (I + A^{-1}\Delta A)^{-1}A^{-1}\Delta b,$$

puis en déduire que :

$$\|\Delta x\| \leq \frac{\|A^{-1}\|\|\Delta A\|\|x\| + \|A^{-1}\|\|\Delta b\|}{1 - \|A^{-1}\|\|\Delta A\|}.$$

2.b. On introduit $\text{cond}(A) = \|A\|\|A^{-1}\|$. Déduire de ce qui précède la formule :

$$\frac{\|\Delta x\|}{\|x\|} \leq \frac{\text{cond}(A)}{1 - \|A^{-1}\|\|\Delta A\|} \left[\frac{\|\Delta A\|}{\|A\|} + \frac{\|\Delta b\|}{\|b\|} \right].$$

2.c. Commenter cette formule, notamment lorsque $\|\Delta A\| \ll \frac{1}{\|A^{-1}\|}$.

Chapitre 3

Calcul Scientifique.

3.1 Interpolation.

Etant donnés des réels x_0, \dots, x_p distincts, on cherche à déterminer le polynôme de degré minimal prenant des valeurs prescrites en ces points, avec éventuellement des valeurs fixées pour les dérivées. Si en tout on a k conditions, on cherche parmi les polynômes de degré au plus $k - 1$, ce qui fait k coefficients à déterminer.

La situation fréquente est celle où l'on dispose d'une fonction f que l'on cherche à approcher par un polynôme. La valeur de la fonction en ces points et les valeurs des dérivées fournit les conditions que l'on cherche à réaliser.

Précisons un peu. On se donne des entiers $\alpha_0, \dots, \alpha_p$, et l'on cherche les polynômes satisfaisant :

$$\text{(Interp)} \quad \forall i = 0, \dots, p, \quad \forall j = 0, \dots, \alpha_i, \quad P^{(j)}(x_i) = f^{(j)}(x_i).$$

On notera que l'on a en tout N conditions, où $N = \sum_{i=0}^p (\alpha_i + 1)$.

Théorème 3.1.1 *Il existe un polynôme P et un seul de degré au plus $N - 1 = p + \sum_i \alpha_i$ satisfaisant les conditions (Interp). L'ensemble des solutions est alors :*

$$\left\{ P + Q \prod_{i=0}^p (X - x_i)^{\alpha_i}; \quad Q \in \mathbb{R}[X] \right\}.$$

De plus, si f est de classe C^N sur $[a; b]$, pour tout $x \in [a, b]$, il existe $\xi_x \in]a, b[$ tel que :

$$f(x) - P(x) = \frac{f^{(N)}(\xi_x)}{N!} \prod_{i=0}^p (x - x_i)^{\alpha_i}.$$

En voici une illustration informatique :

Exercice 3.1.2 *A l'aide de SCILAB, trouver les polynômes de degré minimaux satisfaisant :*

- $P(1) = P(3) = 1$, $P(2) = 2$ et $P(4) = 4$.
- $P(1) = 2$, $P'(1) = 3$ et $P(2) = -1$.

Nous allons spécifier et démontrer ce théorème dans deux cas particuliers. On notera qu'en pratique, le ξ_x n'est pas connu, et donc on préfère une majoration du type :

$$|f(x) - P(x)| \leq \frac{M_N}{N!} \left| \prod_{i=0}^p (x - x_i)^{\alpha_i} \right|,$$

ou encore :

$$|f(x) - P(x)| \leq \frac{M_N}{N!} (b - a)^{N+1},$$

avec $M_N = \sup_{x \in]a, b[} |f^{(N)}(x)|$.

3.1.1 Conditions en un seul point.

On commence par la situation simple où $p = 0$. On se donne un entier $k \geq 0$ et l'on cherche un polynôme P de degré au plus k de sorte que pour tout $j \leq k$, $P^{(j)}(x_0) = f^{(j)}(x_0)$.

1. Cherchez le polynôme P sous la forme : $P = \sum_{j=0}^k y_j (X - x_0)^j$. En déduire l'existence et l'unicité de la solution au problème. Reconnaissez-vous le polynôme obtenu ?
2. Soit I un intervalle contenant x_0 en son intérieur. Supposant f dérivable $p + 1$ fois sur I et à dérivée $(k + 1)$ -ème bornée, pouvez vous proposer un majorant de $|f(t) - P(t)|$ pour $t \in I$? Si $I = [a, b]$, majorez $|\int_a^b f - \int_a^b P|$.

3.1.2 Conditions en plusieurs points.

On se donne maintenant $p + 1$ couples (x_i, y_i) pour i variant de 0 à p , où les x_i sont deux à deux distincts, et l'on cherche un polynôme P de degré au plus p tel que pour tout i , on ait : $P(x_i) = y_i$. On cherche les coefficients a_p, \dots, a_0 de P :

$$P = \sum_{j=0}^p a_j X^j.$$

Par construction, les coefficients a_j doivent vérifier ces n équations :

$$\begin{cases} a_p x_0^p + a_{p-1} x_0^{p-1} + \dots + a_1 x_0 + a_0 = y_0 \\ a_p x_1^p + a_{p-1} x_1^{p-1} + \dots + a_1 x_1 + a_0 = y_1 \\ \vdots \\ a_p x_p^p + a_{p-1} x_p^{p-1} + \dots + a_1 x_p + a_0 = y_p \end{cases}$$

Question 1. Ecrire sous forme matricielle le système linéaire dont sont solutions les coefficients a_j .

Question 2. Ecrire une fonction `interpo` prenant en argument une matrice à deux colonnes dont la première colonne est celle des x_i et la seconde celle des y_i , et qui donne en résultat l'unique vecteur solution (a_p, \dots, a_0) . La taille p doit pouvoir varier (mais vous pouvez, si vous le souhaitez, commencer par écrire une procédure pour $p = 2$, puis la modifier en conséquence). L'application de la fonction à la matrice :

$$\begin{pmatrix} 0 & 2 \\ 0.5 & -1.25 \\ 1 & -4 \end{pmatrix}$$

devra fournir $[1 \ -7 \ 2]$, signifiant que l'unique polynôme P_0 de degré au plus 2 tel que $P_0(0) = 2$, $P_0(0.5) = -1.25$ et $P_0(1) = -4$ est $P_0 = X^2 - 7X + 2$.

Question 3. En fait Lagrange a introduit des polynômes particuliers associés aux x_i :

$$L_i = \prod_{j \neq i} \frac{X - x_j}{x_i - x_j}.$$

Calculez $L_i(x_k)$ (distinguer $i = k$ et $i \neq k$) et en déduire la solution à notre problème.

Question 4 (Formule d'erreur). Le but ici est d'estimer l'erreur produite en remplaçant une fonction f par le polynôme l'interpolant aux x_i (i.e. tel que $y_i = f(x_i)$). On supposera que f est $p + 1$ fois dérivable sur un intervalle I contenant les x_i . Soit un réel x fixé dans I distinct des x_i . On introduit la fonction :

$$R(t) = f(t) - P(t) - A \prod_{i=0}^p (t - x_i).$$

4.a. Vérifier que R est p fois dérivable et que $R^{(p)}(t) = f^{(p)}(t) - Ap!$.

4.b. On choisit A de sorte que $R(x) = 0$. En remarquant que R a au moins $p + 1$ racines dans I et en appliquant plusieurs fois le théorème de Rolle, montrer qu'il existe un $\theta \in I$ (dépendant de x) tel que $R^{(p)}(\theta) = 0$.

4.c. En déduire :

$$\forall x \in I, \quad \exists \theta \in I, \quad f(x) - P(x) = \frac{f^{(p)}(\theta)}{p!} \prod_{i=0}^p (x - x_i). \quad (3.1)$$

Question 5. On veut revenir au problème de la sous-section précédente lorsque $k = 1$. On cherche un polynôme P de degré au plus 1 tel que $P(x_0) = y_0$ et $P'(x_0) = z_0$ (bien entendu ce problème est assez trivial, mais on se limite au cas de $k = 1$ pour simplifier). Un utilisateur ayant lu la section propose d'utiliser cette section avec $x_1 = x_0 + \varepsilon$, $P(x_2) = y_0 + z_0\varepsilon$ et de faire dans l'expression du polynôme tendre ε vers 0. Justifier heuristiquement cette démarche. La tester.

Nous en venons à la démonstration de la formule d'erreur mixte, sur un exemple particulier, mais qui est suffisamment représentatif pour comprendre le cas général.

Exercice 3.1.3 Soit f une fonction définie sur un intervalle ouvert I , trois fois dérivable sur I , et P le polynôme d'interpolation de f , de degré minimal, satisfaisant $P(x_1) = f(x_1)$, $P'(x_1) = f'(x_1)$ et $P(x_2) = f(x_2)$. On fixe x dans l'intervalle I distinct des x_i , et l'on considère :

$$g(t) = f(t) - P(t) - A(t - x_1)^2(t - x_2),$$

où A va être choisi immédiatement.

1. On choisit A de sorte que $g(x) = 0$. Quelle est la valeur de A ?
2. Montrer que g' s'annule en trois points distincts (on remarquera que $g'(x) = 0$).
3. En déduire qu'il existe ξ_x de sorte que $g'''(\xi_x) = 0$, puis aboutir à la formule d'erreur.

3.2 Calcul de séries.

Dans cette section, on calcule des sommes partielles de différentes séries pour exhiber leur éventuelle convergence. Par la suite, on se sert de séries pour calculer des valeurs approchées de π . Enfin, on se rendra compte qu'un calcul numérique même très précis est souvent insuffisant pour connaître la nature d'une série.

3.2.1 Sommes partielles et calcul de π par la fonction ζ de Riemann.

Dans toute la suite, on considère une série $\sum u_n$ où $(u_n)_{n \in \mathbb{N}}$ est une suite de nombres réels. On considérera la suite $(S_n)_{n \in \mathbb{N}}$ des sommes partielles de la série définie par :

$$S_n = \sum_{k=0}^n u_k \quad \text{pour } n \in \mathbb{N}.$$

Lorsque la série est convergente, on considérera le reste d'ordre n de la série, défini par

$$R_n = \sum_{k=n+1}^{\infty} u_k \quad \text{pour } n \in \mathbb{N}.$$

1. Soit $u_n = \frac{1}{(n+1)^4}$ pour $n \in \mathbb{N}$. Rappeler le résultat théorique concernant la nature de cette série. Faire un programme permettant de tracer S_n en fonction de n avec n variant de 1 à N . La suite (S_n) semble-t-elle converger ? Faire la même chose avec $u_n = \frac{1}{(n+1)^2}$. Quelle série semble converger le plus rapidement ?
2. On peut montrer (à l'aide des séries de Fourier) que :

$$\sum_{k=1}^{\infty} \frac{1}{k^2} = \frac{\pi^2}{6} \quad \text{et} \quad \sum_{k=1}^{\infty} \frac{1}{k^4} = \frac{\pi^4}{90}.$$

En déduire, pour les deux séries, le tracé de R_n en fonction de n avec n variant de 1 à N (on fera les 2 tracés sur le même graphe). On conjecture que lorsque n tend vers l'infini, $R_n \sim \frac{a}{n^\alpha}$ et l'on cherche à trouver α (qui est un indicateur de la vitesse de convergence). On fait, pour les deux séries, le tracé de $\ln(R_n)$ en fonction de $\ln(n)$ avec n variant de 1 à 100 ; justifiez ce choix et quelles valeurs conjecturez-vous pour α dans chacun des cas. Justifiez théoriquement le résultat obtenu ; on pourra, en utilisant la décroissance de la fonction $t \mapsto 1/t^a$ ($a > 1$) constater que :

$$\int_{n+1}^{+\infty} \frac{dt}{t^a} \leq \sum_{k=n+1}^{\infty} \frac{1}{k^a} \leq \int_n^{+\infty} \frac{dt}{t^a}.$$

3. Sachant que lorsque p est un entier pair, $\sum_{k=1}^{\infty} \frac{1}{k^p}$ est de la forme $r_p \pi^p$ où r_p est un rationnel connu ($r_2 = 1/6$, $r_4 = 1/90$, $r_6 = 1/945$, $r_8 = 1/9450, \dots$), on peut pour tout p estimer π par la suite :

$$\pi_p(n) = \left(\frac{\sum_{k=1}^n \frac{1}{k^p}}{r_p} \right)^{1/p}.$$

Donner un équivalent lorsque n tend vers l'infini de $\pi - \pi_p(n)$ et en déduire le choix de p optimal parmi les valeurs 2, 4, 6, 8.

3.2.2 Autres séries convergeant vers π .

1. Soit la série $\sum_{n=0}^{+\infty} u_n$ avec $u_n = (-1)^n \frac{4}{2n+1}$. On admettra que cette série a pour somme π (ce qui se montre facilement avec le cours sur les séries entières). Vérifier numériquement que S_{10000} est proche de π (pour ce calcul on ne fera pas de boucle, mais on travaillera en faisant la somme du vecteur constitué des $(u_n)_{0 \leq n \leq 10000}$). Calculer R_n pour $n = 100, 1000, 10000$ et le comparer avec la majoration du reste obtenue à partir du critère de convergence des séries alternées. Cette majoration est-elle très éloignée de la vraie valeur ? Calculer une approximation de π à partir de la série à 10^{-4} près.
2. Soit la série $\sum_{n=0}^{+\infty} u_n$ avec $u_n = 6 \frac{(2n)!}{2^{4n+1} (n!)^2 (2n+1)}$. On admettra que cette série a pour somme π . Vérifier numériquement que S_{10} est proche de π (pour ce faire on calculera $n!$ à partir de la commande `gamma(n+1)` qui permet de travailler vectoriellement). Pour avoir à l'écran un plus grand nombre de décimales, on tapera la commande `SCILAB format('v', 20)`. Vérifier que S_{10} est une très bonne approximation de π .

3.2.3 Un exemple d'accélération de convergence.

On suppose connus les résultats de la section sur la fonction ζ ci-dessus. On veut calculer une valeur approchée de :

$$S = \sum_{n=0}^{+\infty} \frac{1}{(n+1)^2 + 1}.$$

On note $S_p = \sum_{n=0}^p \frac{1}{(n+1)^2 + 1}$ la somme partielle.

1. Trouver un entier N assurant que $|S - S_N| \leq 10^{-4}$. La convergence semble-t-elle rapide ?
2. On note $S'_p = \sum_{n=0}^p \frac{1}{(n+1)^2}$, et l'on rappelle que S'_p tend vers $\pi^2/6$. Exprimer en fonction de S la limite S'' de $S''_p = S_p - S'_p$.
3. Trouver un entier N assurant que $|S'' - S''_N| \leq 10^{-4}$. En déduire un nombre \hat{S} tel que $|S - \hat{S}| \leq 10^{-4}$.

3.2.4 Limites du calcul numérique.

Cette partie a pour but de montrer que quelque soient les performances des logiciels de calcul numérique, il y a encore un sens et de l'intérêt à faire des mathématiques "théoriques"...

1. Soit la série de terme général $u_n = \frac{1}{n+1}$. Rappeler le résultat théorique sur la convergence de cette série. Tracer S_n en fonction de n avec n variant de 1 à 10000. La suite (S_n) semble-t-elle converger ? tracer alors $S_n - \ln(n)$ en fonction de n avec n variant de 1 à 10000. Conclusions ? Expliquer théoriquement le résultat.
2. Soit les séries de termes généraux $u_n = \frac{n!}{1000^n}$ et $v_n = \frac{1000^n}{n!}$. Numériquement, déterminer quelle série converge et quelle série diverge. Répondre à la même question théoriquement. Conclusion ?

3.3 Racines et extrema de fonctions d'une variable.

On suppose que l'on veut résoudre une équation de type

$$f(x) = 0 \quad \text{pour } x \in [0, 1]. \quad (3.2)$$

Ce type d'équations recoupe un grand nombre de problèmes mathématiques, comme celui de trouver la solution de $y_0 = g(x)$ avec $x \in [a, b]$, de déterminer un extremum local à une fonction (la condition nécessaire du premier ordre sur un ouvert porte sur l'annulation de la dérivée),... Dans la plupart des cas (à part

les rares cas d'école habituels), on ne peut pas trouver une solution théorique à cette équation (voir l'exemple ci-dessous). Aussi est-on amené à calculer de façon approchée l'ensemble des solutions de cette équation. Par la suite, nous étudions différentes méthodes numériques classiques de résolution de (3.4).

3.3.1 Un exemple.

Dans toute la suite nous considérerons l'exemple de la fonction f suivante sur $[0, 1]$:

$$f(x) = \sin(2x) - x. \quad (3.3)$$

Étudier théoriquement les solutions de (3.4) dans ce cas (utiliser par exemple un tableau de variations...). En déduire que (3.4) possède alors 2 solutions, une que l'on peut donner théoriquement et notée x_0 , l'autre notée x_1 a priori inconnue. Vérifier en traçant le graphe de f sur $[0, 1]$. Déterminer une valeur approchée à 0.1 près de x_1 .

3.3.2 Méthode de dichotomie.

La première méthode, la plus naturelle, pour obtenir une valeur approchée de x_1 est une méthode d'approximation successive dite par "dichotomie". Elle consiste à d'abord fixer un intervalle $[a_0, b_0]$ dans lequel (3.4) admet une unique solution. Ainsi, $f(a_0) \cdot f(b_0) < 0$. Ensuite, on considère le milieu du segment $[a_0, b_0]$, soit $\frac{1}{2}(a_0 + b_0)$ et on calcule numériquement $f\left(\frac{1}{2}(a_0 + b_0)\right)$. Si $f(a_0) \cdot f\left(\frac{1}{2}(a_0 + b_0)\right) < 0$ alors x_1 est dans $[a_0, \frac{1}{2}(a_0 + b_0)]$, on pose $a_1 = a_0$, $b_1 = \frac{1}{2}(a_0 + b_0)$ et on recommence le même procédé. Sinon, x_1 est dans $[\frac{1}{2}(a_0 + b_0), b_0]$, on pose $b_1 = \frac{1}{2}(a_0 + b_0)$, $a_1 = a_0$ et on recommence le même procédé.

Programmer cette méthode (fichier *dichotomie.sci*) pour une fonction f quelconque définie dans un fichier *f.sci* et d'abord pour 10 itérations. Introduire ensuite dans le programme le calcul du nombre d'itérations nécessaires pour une approximation *Delta* donnée (par exemple $\Delta = 0.0001$) en fonction de Δ , a_0 et b_0 . Appliquer cette méthode pour obtenir une valeur approchée à 10^{-10} près de x_1 .

3.3.3 Méthode de la sécante et méthode de Newton.

On se place encore dans le cadre où l'on sait que (3.4) admet une unique solution dans $[a_0, b_0]$. Ces deux méthodes sont basées sur un principe commun au départ, qui consiste à remplacer la fonction par un polynôme d'interpolation dont on calcule les racines. Pour la facilité de calcul, on prend des polynômes de degré 1. La seule différence entre les deux méthodes consiste en le choix du polynôme.

Méthode de la sécante. On donne pour valeur approchée \tilde{x}_1 de x_1 le point d'intersection entre la droite passant par les points $(a_0, f(a_0))$ et $(b_0, f(b_0))$ et l'axe des abscisses, c'est-à-dire que l'on remplace f par son polynôme d'interpolation de Lagrange aux points a_0 et a_1 . Montrer qu'alors $\tilde{x}_1 = a_0 - f(a_0) \cdot \frac{b_0 - a_0}{f(b_0) - f(a_0)}$. On cherche maintenant à établir une formule d'erreur, en supposant que f est de classe \mathcal{C}^2 sur $[a_0, b_0]$. On pose $m_1 = \inf_{x \in [a_0, b_0]} |f'(x)| > 0$ et $M_2 = \sup_{x \in [a_0, b_0]} |f''(x)|$.

1. Soit P le polynôme de degré au plus 1 tel que $P(a_i) = f(a_i)$ ($i = 1, 2$). En utilisant la formule d'erreur vue dans la section interpolation, montrer que pour tout t , on a :

$$|f(t) - P(t)| \leq \frac{M_2}{2}(b_0 - a_0)^2,$$

et en déduire une majoration de $P(\tilde{x}_1)$.

2. A l'aide d'un dessin et du théorème de Thalès, écrire $|\tilde{x}_1 - x_1|$ en fonction de $|x_1 - a_0|$, de $P(\tilde{x}_1)$ et d'autres éléments. En déduire que :

$$|\tilde{x}_1 - x_1| \leq \frac{M_2}{m_1} \times \frac{(b_0 - a_0)^2}{8}.$$

Méthode de Newton. On donne pour valeur approchée \hat{x}_1 de x_1 le point d'intersection entre la tangente à f en a_0 et l'axe des abscisses (on prend donc le polynôme de Taylor de degré 1 en a_0). Montrer qu'alors $\hat{x}_1 = a_0 - \frac{f(a_0)}{f'(a_0)}$. En déduire, en adaptant la démarche précédente, que si f est de classe \mathcal{C}^2 sur $[a_0, b_0]$, si $m_1 = \inf_{x \in [a_0, b_0]} |f'(x)| > 0$ et $M_2 = \sup_{x \in [a_0, b_0]} |f''(x)|$ alors

$$|\hat{x}_1 - x_1| \leq \frac{M_2}{m_1} \times \frac{(b_0 - a_0)^2}{2}.$$

Pour ces 2 méthodes, on utilise une méthode itérative, que l'on peut rédiger dans le style de la méthode de dichotomie en remplaçant $\frac{1}{2}(a_0 + b_0)$ par \tilde{x}_1 ou \hat{x}_1 . Dans quel cas gagne-t-on en rapidité de convergence par rapport à la méthode de dichotomie ?

Méthode de Newton et point fixe

L'objectif de cette méthode est de déterminer une suite (u_n) convergeant vers x_1 . Pour cela, on considère une fonction ϕ telle que pour $x \in [a, b]$, $f(x) = 0 \iff \phi(x) = x$ et si ϕ vérifie des propriétés de type $\phi([a_0, b_0]) \subset [a_0, b_0]$ et $\sup_{x \in [a_0, b_0]} |\phi'(x)| < 1$. On définit alors (u_n) par $u_{n+1} = \phi(u_n)$ avec $u_0 \in [a_0, b_0]$ et on

a bien (u_n) qui converge vers x_1 , unique solution de $x = \phi(x)$. Cette méthode est appelée méthode de Newton lorsque la fonction ϕ considérée est :

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

Expliquer le choix d'une telle fonction ϕ à partir de considérations sur ϕ' . En déduire, en fonction de a_0 et de b_0 , et de $N_1 = \sup_{x \in [a_0, b_0]} |\phi'(x)|$, la vitesse de convergence de (u_n) vers x_1 . Appliquer cette méthode au cas particulier de f précédent. Combien faut-il calculer de termes de (u_n) pour obtenir une approximation à 10^{-10} près de x_1 ?

3.3.4 Conclusions

Comparez les différentes méthodes et mettez en évidence les avantages et inconvénients de chacune d'entre elles (on examinera en particulier les problèmes du choix de $[a_0, b_0]$, d'unicité de la solution de (3.4), de la régularité de la fonction). On pourra à titre d'exemple traiter le cas de $f(x) = \cos(1/(x+0.1)) - \sqrt{x}$.

3.4 Calcul d'intégrales.

En dehors de cas d'école présentés pendant les cours de mathématiques, il est en général impossible de calculer théoriquement la valeur numérique d'une intégrale définie. On doit alors utiliser des méthodes de calcul numérique pour donner une valeur approchée d'intégrales. Sauf dans le cas des méthodes de Gauss, nous considérerons des intégrales du type :

$$I(f) = \int_a^b f(t)dt,$$

avec $-\infty < a < b < +\infty$ et f intégrable au sens de Riemann sur $[a, b]$ (nous exigerons souvent une régularité plus forte pour obtenir des formules d'erreur).

Les méthodes que nous allons présenter ici partent toutes d'un principe général, qui consiste à choisir un entier $p \geq 1$ et des points¹ x_1, \dots, x_p deux à deux distincts dans $[a, b]$, et d'approcher l'intégrale $I(f)$ par une formule de la forme :

$$I_p(f) = \sum_{i=1}^p a_i f(x_i).$$

Pour conserver l'homogénéité, nous allons introduire les $\theta_i \in [0; 1]$ tels que pour tout i , $x_i = a + \theta_i(b - a)$. Les θ_i indiquent la position relative des x_i par rapport aux points extrêmes $[a, b]$. Ainsi, $\theta_i = 0$ indique que $x_i = a$, $\theta_i = 1$ indique que $x_i = b$, $\theta_i = 1/2$ indique que x_i est le milieu du segment.

¹On notera ici que l'indice commence à 1, il y a donc p points en tout.

3.4.1 Méthodes élémentaires.

Ces méthodes consistent, une fois p et les x_i fixés, à choisir les a_i de sorte que la formule d'approximation

$$I(f) \approx I_p(f)$$

donne une égalité pour f polynômiale de degré le plus grand possible. Le degré maximal pour lequel il y a égalité s'appelle *l'ordre* de la méthode.

Il paraît notamment raisonnable d'exiger que la méthode soit exacte pour les polynômes de Lagrange associés aux x_i , les L_i . L'égalité $I(L_i) = I_p(L_i)$ détermine les a_i de manière unique :

$$\forall i = 1, \dots, p, \quad a_i = I(L_i).$$

Avec ce choix des a_i , on montre que la méthode est au moins d'ordre $p - 1$. On peut montrer que l'ordre maximal d'une telle méthode est $2p - 1$ et qu'un seul choix des couples $(a_i, x_i)_{1 \leq i \leq p}$ permet d'atteindre cet ordre (voir la section méthodes de Gauss).

Remarque 3.4.1 *La méthode étant d'ordre au moins $p - 1$, on peut déterminer également les a_i en écrivant successivement que*

$$I(1) = I_p(1), I(X) = I_p(X), \dots, I(X^{p-1}) = I_p(X^{p-1}).$$

Cela donne un système linéaire de type Van der Monde en les a_i .

Exercice 3.4.2 1. *Lorsque $p = 1$, écrire les formules d'approximations obtenues.*

2. *Vérifier qu'elles sont toutes (au moins) d'ordre 0, et qu'il n'y en a qu'une d'ordre 1, qui correspond à $\theta_0 = 1/2$.*

3. *Spécifier les formules obtenues en 1 lorsque $\theta_1 = 0$ (formule des rectangles à gauche), $\theta_1 = 1/2$ (formule du point milieu), $\theta_1 = 1$ (formule des rectangles à droite).*

4. *Écrire la formule obtenue lorsque $p = 2$, $\theta_1 = 0$, $\theta_2 = 1$ (formule des trapèzes). Quel est l'ordre de la méthode ?*

5. *Écrire la formule obtenue lorsque $p = 3$, $\theta_1 = 0$, $\theta_2 = 1/2$, $\theta_3 = 1$ (formule de Simpson). On pourrait vérifier que la méthode est d'ordre 3 (ce qui n'est pas demandé).*

Venons en à la formule d'erreur. La méthode étant d'ordre au moins $p - 1$, en notant que $I_p(f) = I(P_0)$, où P_0 est le polynôme d'interpolation aux x_i , on obtient facilement une majoration du type :

$$|I(f) - I_p(f)| \leq C(\theta_1, \dots, \theta_p) \frac{M_p(b-a)^{p+1}}{p!},$$

pour f de classe C^p , où $C(\theta_1, \dots, \theta_p) \in]0; 1]$ est une constante² ne dépendant que des θ_i . Si la méthode est d'ordre q supérieur $p-1$, on peut obtenir une majoration du type :

$$|I(f) - I_p(f)| \leq C \frac{M_{q+1}(b-a)^{q+2}}{(q+1)!},$$

pour f de classe C^q , où C est une constante dans $]0; 1]$. Cette forme ne semble pas nécessairement à cet instant un gain, mais nous allons voir que dans les méthodes composées le fait d'augmenter la puissance de $(b-a)$ est essentiel.

Exercice 3.4.3 *Etablir les formules d'erreurs pour les méthodes précédentes. Pour la méthode du point milieu, on prendra le polynôme Q tel que $Q((a+b)/2) = f((a+b)/2)$ et $Q'((a+b)/2) = f'((a+b)/2)$. Pour la formule de Simpson, on prendra le polynôme Q tel que $Q(a) = f(a)$, $Q((a+b)/2) = f((a+b)/2)$, $Q'((a+b)/2) = f'((a+b)/2)$, et $Q(b) = f(b)$.*

3.4.2 Méthodes composées.

Les méthodes précédentes souffrent d'un défaut que font apparaître les formules d'erreur. Il faut augmenter le nombre de points et recalculer les a_i pour augmenter la précision. Aussi, au lieu de les appliquer directement, on applique souvent une méthode composée. Celle-ci consiste à choisir un entier n , à découper l'intervalle $[a, b]$ en des intervalles J_1, \dots, J_n d'intérieurs deux à deux disjoints et dont la réunion fait $[a, b]$. On a alors :

$$I(f) = \sum_{j=1}^n \int_{J_j} f(t) dt.$$

On applique alors une méthode élémentaire sur chaque J_j .

En général, on fait un découpage régulier : $J_j = [a + (j-1)/n(b-a); a + j/n(b-a)]$ et l'on applique la même méthode sur chaque intervalle. Suivant ce principe général, si la méthode appliquée sur chaque sous intervalle est d'ordre $q \geq p$, on obtient une erreur totale de la forme³ :

$$C \frac{M_{q+1}(b-a)^{q+2}}{n^{q+1}(q+1)!}.$$

On voit ici l'intérêt d'avoir q le plus grand possible (en raison du terme $1/n^{q+1}$).

Exercice 3.4.4 *Ecrire les formules composées issues des méthodes élémentaires vues précédemment et les formules d'erreurs correspondantes.*

²bien entendu, si on ne veut pas la calculer, on peut la prendre égale à 1.

³A ce niveau, le fait que C ne dépende que des θ_i est essentiel.

Exercice 3.4.5 On veut calculer une valeur approchée de l'intégrale :

$$I = \int_0^1 e^{-t^2} dt.$$

1. Donner une valeur approchée de I à 10^{-7} près (soit une valeur a t.q. $|a - I| \leq 5 \times 10^{-8}$) en exprimant cette intégrale sous forme d'une série rapidement convergente.
2. Tester les méthodes vues précédemment et les comparer (on admettra que $M_1 = \sqrt{2}e^{-1/2}$, $M_2 = 2$ et $M_4 = 12$). On cherchera à réaliser un calcul approché à 10^{-5} près, grâce au majorant de l'erreur, puis on comparera avec le choix de n qui convient en pratique.

Exercice 3.4.6 On veut calculer une valeur approchée de l'intégrale :

$$I = \int_0^1 f(t) dt,$$

où $f(t) = \min\{2t; 1\}$ (bien entendu, cette intégrale se calcule à vue). On veut faire un découpage avec $n = 3$ et appliquer la formule des rectangles à gauche sur chaque sous-intervalle.

1. (**Répondre sans aucun calcul**). Expliquer pourquoi le découpage en parties égales n'est pas optimal. Quel découpage proposeriez vous ?
2. Connaissant la valeur exacte, trouver un meilleur découpage que le régulier.

3.4.3 Calcul d'intégrales : méthodes de Gauss.

Les méthodes de Gauss relèvent d'un esprit assez différent, puisque cette fois nous n'allons pas fixer avant les x_i . On a vu auparavant que si l'on fixe les x_i , on a un choix et un seul des a_i assurant que la méthode est au moins d'ordre p , et que parfois la méthode est d'ordre supérieur. Nous allons voir qu'il y a un choix et un seul des couples (x_i, a_i) assurant que la méthode soit (au moins) d'ordre $2p - 1$. Ce choix fournit alors une méthode d'ordre $2p - 1$ exactement. Il s'agit donc d'une méthode élémentaire, mais il est bien entendu possible de la composer.

La formule de Gauss est en fait d'une grande souplesse. Elle s'applique à des calculs d'intégrales :

$$I(f) = \int_a^b f(t)\omega(t)dt$$

où l'intervalle d'intégration n'est pas nécessairement borné⁴. On supposera la fonction ω continue, positive et non constamment nulle sur $]a, b[$. On suppose enfin que l'on connaît explicitement les intégrales suivantes, et qu'elles sont finies :

$$J_k = \int_a^b t^k \omega(t) dt, \quad k \in \mathbb{N}.$$

⁴Le cas antérieur consiste à choisir $\omega = 1$ et $[a, b]$ compact.

On cherche donc une formule d'interpolation de la forme :

$$I(f) \approx I_p(f) = \sum_{i=1}^p a_i f(x_i)$$

(avec pour tout i , $a_i > 0$ et $x_i \in]a, b[$), de sorte que la formule soit juste pour les polynômes de degré le plus grand possible. Heuristiquement, on dispose de $2p$ valeurs à choisir, et on peut donc espérer avoir une formule valide pour les polynômes de degré (au plus) $2p - 1$. Bien entendu ceci n'est pas rigoureux (on a notamment un système non linéaire en les x_i). Il est remarquable que l'intuition soit correcte :

Théorème 3.4.7 *Il existe un choix unique des a_i et des $x_i \in]a, b[$ de sorte que pour tout polynôme P de degré au plus $2p - 1$, on ait :*

$$I(P) = I_p(P).$$

De plus, les a_i sont strictement positifs. Enfin, il existe un polynôme de degré $2p$ tel que $I(P) \neq I_p(P)$ (donc on ne peut pas faire mieux).

Aussi surprenant que cela puisse paraître au premier abord, la méthode utilise l'algèbre bilinéaire de L^2 . On munit l'espace vectoriel des polynômes du produit scalaire :

$$\langle P, Q \rangle = I(PQ) = \int_a^b P(t)Q(t)\omega(t)dt.$$

Le procédé de Gram-Schmidt appliqué à la base canonique permet de fabriquer l'unique famille $(P_n)_n$ de polynômes de sorte que la famille soit orthogonale et que pour tout k , P_k soit unitaire de degré k . On montre alors que les racines de P_k sont toutes réelles distinctes dans l'intervalle $]a, b[$. Le fait que pour tout $j \leq p-1$, $0 = \langle X^j, P_p \rangle = I(X^j P_p) = I_p(X^j P_p)$ impose de choisir pour x_i les racines de P_p d'où l'unicité des x_i . En écrivant alors que pour tout $j \leq p-1$, $J_j = I(X^j)$ on obtient un système linéaire de van der Monde ayant une solution unique en les a_i . Par construction, on vérifie que l'unique formule possible convient. Le polynôme P_p^2 montre que $I = I_p$ n'est pas valable pour tous les polynômes de degré $2p$. Enfin, désignant par $(L_j)_j$ les polynômes de Lagrange associés aux (x_i) , la formule appliquée à L_j^2 montre que $a_j = I(L_j^2) > 0$.

Pour un certain nombre d'exemples, les polynômes $(P_n)_n$ (éventuellement modifiés à une constante multiplicative près) sont connus et tabulés. Voici quelques exemples :

1. Lorsque $a = 0$, $b = 1$ et $\omega = 1$, on obtient les polynômes de Legendre.
2. Lorsque $a = 0$, $b = +\infty$ et $\omega = [t \mapsto e^{-t}]$, on obtient les polynômes de Laguerre.

3. Lorsque $a = -\infty$, $b = +\infty$ et $\omega = [t \mapsto e^{-t^2}]$, on obtient les polynômes d'Hermite.
4. Lorsque $a = -1$, $b = 1$ et $\omega = [t \mapsto 1/\sqrt{1-t^2}]$, on obtient les polynômes de Tchebitchev de seconde espèce.

On notera le fait remarquable que sauf dans le premier cas, on traite essentiellement des situations d'intégrales impropres. C'est d'ailleurs ce qui justifie le choix des exemples d'applications vus ci-dessous.

Cas de Gauss-Laguerre

Ici on a donc

$$I(f) = \int_0^{+\infty} e^{-t} f(t) dt.$$

Les polynômes de Laguerre sont connus notamment par la formule suivante :

$$Lag_n(x) = e^x \frac{d^n}{dx^n} (e^{-x} x^n),$$

mais nous ignorerons cet aspect des choses. On va écrire un programme permettant de calculer, p étant fourni en entrée, les x_i et les a_i . L'exercice ci-dessous est décomposé afin de vous aider à écrire le programme. Enfin, il ne me reste plus qu'à vous dire que dans ce cas $J_j = j!$ (je rappelle que $j!$ se calcule (vectoriellement !) par `gamma(j+1)` dans SCILAB).

Exercice 3.4.8 Soit n un entier fixé.

1. Dans cette première étape, on cherche le polynôme P_p sous la forme :

$$P_p = X^p + \sum_{j=0}^{p-1} c_j X^j$$

(bien entendu les c_j dépendent aussi de p). Pour tout $k \leq p-1$, X^k et P_p sont orthogonaux, cela fournit un système linéaire en les c_j . L'écrire puis en déduire le moyen, à partir de p , de récupérer les c_j , puis P_p (utilisez la commande SCILAB `poly`).

2. En déduire comment récupérer les x_i (utilisez la commande SCILAB `roots`). Poser le système en les a_i puis écrire comment le résoudre dans SCILAB.
3. Terminer l'écriture de la procédure.
4. L'utiliser pour calculer $I_p(f)$ avec $f(t) = \sqrt{t}$. Comparer les valeurs obtenues à la valeur exacte ($\sqrt{\pi}$).

Cas de Gauss-Tchebitchev

Ici on a donc

$$I(f) = \int_{-1}^1 \frac{f(t)}{\sqrt{1-t^2}} dt = \int_0^\pi f(\cos(\theta)) d\theta.$$

Les polynômes de Tchebitchev de seconde espèce sont connus notamment par le fait que T_n est l'unique polynôme (de degré n) tel que :

$$\forall \theta \in \mathbb{R}, \quad T_n(\cos(\theta)) = \cos(n\theta).$$

Par exemple, $T_2(X) = 2X^2 - 1$ puisque pour tout θ , $\cos(2\theta) = 2\cos^2(\theta) - 1$.

Les intégrales J_j se calculent, lorsque p est pair, par les intégrales de Wallis. On trouve :

$$J_j = \begin{cases} 0 & \text{si } j \text{ est impair} \\ \frac{\Gamma((j+1)/2)\sqrt{\pi}}{\Gamma(j/2+1)} & \text{si } j \text{ est pair} \end{cases}$$

Dans SCILAB vous pourrez calculer J_j par :

```
deff("z=Jj(j)", "z=(1-modulo(j,2)).*sqrt(%pi).*gamma((j+1)/2)./gamma(j/2+1)")
```

qui fonctionne avec des vecteurs j .

Toute personne à l'aise avec la trigonométrie n'aura aucun mal à montrer que les T_n sont orthogonaux, qu'ils sont de norme $\sqrt{\pi/2}$ (pour $n \geq 1$) et que leur coefficient dominant est $2^{n-1}X^n$ (ainsi $P_n = T_n/2^{n-1}$). Enfin les racines de T_n sont les :

$$x_i = \cos\left(\frac{2i+1}{2n}\pi\right), \quad i = 1, \dots, n.$$

On admettra ces points.

1. Le polynôme $T_n(X)$ se calcule dans SCILAB par `chepol(n, 'x')`. A partir de là, reprendre les questions 2 et 3 de l'exercice précédent (en prenant directement les valeurs des x_i indiquées ci-dessus). Testez plusieurs petites valeurs de n . Que constatez-vous pour les a_i ? Peut-on prévoir leur valeur (déterminer leur somme).

2. En fait ce qui constaté dans la question 1 concernant les a_i est vrai. Ainsi dans le cas de Tchebitchev, on a une formule explicite :

$$I_n(f) = \frac{\pi}{n} \sum_{i=1}^n f\left(\cos\left(\frac{2i+1}{2n}\pi\right)\right).$$

3. Utilisez directement cette formule pour approcher

$$I = \int_{-1}^1 \frac{\cos(t)}{\sqrt{1-t^2}} dt = \int_0^\pi \cos(\cos(\theta)) d\theta.$$

La convergence semble t-elle rapide ?

Estimation de l'erreur dans la méthode de Gauss.

On suppose que la fonction f est de classe C^{2p} sur l'intervalle d'intégration. Soit P le polynôme de degré $2p-1$ au plus vérifiant :

$$\forall i = 1, \dots, p, \quad P(x_i) = f(x_i) \quad \text{et} \quad P'(x_i) = f'(x_i).$$

On rappelle la formule d'erreur suivante :

$$\forall x \in]a, b[, \exists \xi_x \in]a, b[, \quad f(x) - P(x) = \frac{f^{(2p)}(\xi_x)}{(2p)!} \left[\prod_{i=1}^p (x - x_i) \right]^2 = \frac{f^{(2p)}(\xi_x)}{(2p)!} P_p^2(x).$$

En pratique ξ_x n'étant pas connu, on majore $|f^{(2p)}(\xi_x)|$ par $M_{2p} = \sup_{t \in]a, b[} |f^{(2p)}(t)|$. En notant que $I_p(P) = I_p(f)$ (car P et f ont les mêmes valeurs aux x_i) et $I_p(P) = I(P)$ car la formule est exacte pour P , on arrive finalement à la majoration :

$$|I(f) - I_p(f)| \leq \frac{M_{2p}}{(2p)!} \|P_p\|^2.$$

Exercice 3.4.9 1. On a vu un avantage de la méthode de Gauss, son applicabilité à des intégrales impropres. Mais que pensez vous de celle-ci au vu de la formule d'erreur ? Est-elle intéressante pour l'exemple de la racine carrée dans le cas de Gauss-Laguerre ?

2. Spécifier la formule d'erreur dans le cas de Gauss-Tchebichev, puis dans le cas particulier de $f = \cos$ (traité dans l'exercice précédent). Que pensez vous de la convergence de la méthode dans ce cas ?

3.5 Résolution d'Equations Différentielles Ordinaires.

On suppose maintenant que l'on veut résoudre l'équation différentielle suivante

$$\begin{cases} y'(x) = f(x, y(x)) & \text{pour } x \in [0, T], \\ y(0) = y_0 & \text{avec } y_0 \in \mathbb{R}, \end{cases} \quad (3.4)$$

où f est une fonction de classe \mathcal{C}^1 sur $[0, T] \times \mathbb{R}$ avec $T > 0$. En général, on ne peut pas trouver la solution théorique à une telle équation différentielle (en revanche, on sait qu'il existe une solution unique à cette équation, au moins si T n'est pas trop grand). Aussi est-on amené à utiliser des méthodes numériques d'approximation de la solution. Nous voyons dans ici la méthode d'Euler.

3.5.1 La fonction ode de SCILAB.

SCILAB propose plusieurs routines de résolution numérique approchée, dont nous ne détaillerons pas les différences qui seraient complexes à exposer. Prenons l'exemple de la routine `ode`. La syntaxe de base est :

```
> ode(y0, t0, t, F0)
```

où `F0` est une fonction contenant l'équation (voir plus bas), `t` est le vecteur des instants ou sera calculée la solution, `y0` est la condition initiale en `t0`.

A titre de premier exemple, résolvons $y' = y^2$ avec la condition initiale $y(0) = 1$ sur $[0; 1/2]$. Vous pourrez comparer à la solution exacte qui est $t \mapsto \frac{1}{1-t}$. Première étape, on définit la fonction :

```
> deff("ydot=fct(t,y)", "ydot=y.^2")
```

Vous noterez qu'il est indispensable de faire figurer la variable t , même si l'équation n'en dépend pas. On donne ensuite les paramètres :

```
> y0=1 ; t0=0; t=0:0.05:0.5;
```

enfin on calcule la solution :

```
> ode(y0,t0,t,fct)
```

3.5.2 Méthode d'Euler (explicite).

Nous allons maintenant programmer la méthode d'Euler, qui est la plus méthode la plus simple pour résoudre de manière approchée une équation différentielle.

On s'intéresse à l'équation particulière suivante :

$$\begin{cases} y'(x) = -2y(x) & \text{pour } x \in [0, T], \\ y(0) = y_0 & \text{avec } y_0 \in \mathbb{R}. \end{cases} \quad (3.5)$$

1. Résoudre théoriquement cette équation.
2. En utilisant la commande `ode`, déterminer une approximation de la solution sur l'intervalle $[0, 5]$ avec une résolution de 100 (on prendra comme condition initiale $y_0 = 0$, puis $y_0 = 1$ et $y_0 = -5$).
3. Représenter sur le même graphe les solutions théorique et approchée de l'équation (3.5) pour une résolution de 100, puis de 1000. Déterminer la distance maximale entre ces 2 courbes.

Pour résoudre numériquement l'équation (3.4) par une méthode d'Euler, l'idée est d'abord de diviser l'intervalle $[0, T]$ en N sous-intervalles à partir de $0 \leq x_0 < x_1 < \dots < x_N = T$. On cherche alors une approximation \hat{y}_i (ou \tilde{y}_i) de $y(x_i)$ pour $0 \leq i \leq N - 1$, ce qui constituera notre approximation de y sur $[0, T]$. En notant $h_i = x_{i+1} - x_i$, on utilise les formules de Taylor pour $i = 0, 1, \dots, N - 1$:

$$y(x_{i+1}) \sim y(x_i) + h_i y'(x_i), \quad (\text{on choisit les } h_i \text{ suffisamment petits})$$

pour construire de façon récursive la solution de (3.4). Du développement limité précédent, on en déduit que

$$f(x_i, y(x_i)) = y'(x_i) \simeq \frac{y(x_{i+1}) - y(x_i)}{h_i} \simeq \frac{\hat{y}_{i+1} - \hat{y}_i}{h_i},$$

car y est solution de (3.4). On obtient donc une construction itérative d'une famille $(\widehat{y}_i)_{0 \leq i \leq N}$ de la façon suivante :

$$\widehat{y}_{i+1} = \widehat{y}_i + h_i f(x_i, \widehat{y}_i) \quad \text{avec} \quad \widehat{y}_0 = y_0.$$

Construire un programme *eulerexp.sci* qui détermine le vecteur $(\widehat{y}_i)_{0 \leq i \leq N}$ avec pour entrée T, N, y_0 et en utilisant une fonction *feuler.sci* contenant la fonction f de (3.4) (on choisira une discrétisation régulière de $[0, T]$ en N sous-intervalles égaux).

Exercice 3.5.1 (Exploitation de la méthode d'Euler)

1. Faire un programme qui trace la différence entre les solutions théorique et approchées de l'équation (3.5) pour les mêmes paramètres (T, N et y_0). En augmentant progressivement N , que remarquez-vous ?
2. Vérifier en calculant la suite des instants que $y(1)$ est bien égal à $y_0 \exp(-2)$.

3.6 Résolution d'EDP : Calcul approché de dérivées et différences finies.

Ce chapitre va être essentiellement illustré en dimension 1, pour faciliter l'écriture des programmes, ce qui fait que les exemples seront des équations différentielles, mais ce chapitre prend pleinement son ampleur en dimension supérieure.

3.6.1 Calcul approché de dérivées de fonctions d'une variable.

Le calcul approché de dérivées peut-être présenté à partir des formules de Taylor. Supposons que l'on cherche à approcher $f'(x_0)$ et que f soit deux fois dérivable. Les formules de Taylor nous apprennent que si k est petit, $f(x_0 + k)$ est approximativement égal à $f(x_0) + f'(x_0)k$, et que de plus l'erreur est majorée par $\frac{M_2 k^2}{2}$, c'est-à-dire que l'on a :

$$|f(x_0 + k) - (f(x_0) + f'(x_0)k)| \leq \frac{M_2 k^2}{2}.$$

Divisant par h supposé non nul, on obtient :

$$\left| f'(x_0) - \frac{f(x_0 + k) - f(x_0)}{k} \right| \leq \frac{M_2}{2} |k|.$$

Ainsi, une bonne approximation de $f'(x_0)$ est donnée par le taux d'accroissement $\frac{f(x_0+k)-f(x_0)}{k}$, avec une erreur majorée par $\frac{M_2}{2}|k|$ si f est deux fois dérivable au voisinage de x_0 .

3.6. RÉSOLUTION D'EDP : CALCUL APPROCHÉ DE DÉRIVÉES ET DIFFÉRENCES FINIES

En général, on se donne le pas $h > 0$ petit. Le calcul précédent donne alors deux formules possibles d'approximation de $f'(x_0)$ selon que l'on prenne directement h ou $-h$ pour k :

$$f'(x_0) \approx \frac{f(x_0 + h) - f(x_0)}{h},$$

$$f'(x_0) \approx \frac{f(x_0) - f(x_0 - h)}{h}.$$

Dans les deux cas, l'erreur est majorée par $\frac{M_2}{2}h$. En faisant la moyenne des deux, on trouve une troisième formule :

$$f'(x_0) \approx \frac{f(x_0 + h) - f(x_0 - h)}{2h}.$$

Cette formule symétrique donne de meilleurs résultats numériquement, comme on le voit dans l'exercice suivant :

Exercice 3.6.1 Appliquer les trois formules pour le calcul approché de $f'(1)$, avec $f(x) = x^3$. Commenter l'efficacité de chacune.

Pour obtenir une formule d'erreur sur la troisième, posons :

$$g(h) = f(x_0 + h) - f(x_0 - h) - 2hf'(x_0),$$

et supposons f deux fois dérivable. On a :

$$g'(k) = f'(x_0 + k) + f'(x_0 - k) - 2f'(x_0),$$

puis

$$g''(k) = f''(x_0 + k) - f''(x_0 - k).$$

Notant $\omega_{f''}$ le module de continuité de f'' sur $[x_0 - h, x_0 + h]$, on a pour $k \in [0; h]$:

$$|g''(k)| \leq \omega_{f''}(2k),$$

ce qui donne, par deux intégrations successives en tenant compte de $g(0) = g'(0) = 0$:

$$|g(h)| \leq \int_0^h \int_0^k \omega_{f''}(2\delta) d\delta dk.$$

Si par exemple f est 3 fois dérivable au voisinage de x_0 , on obtient comme majorant : $\frac{M_3}{3}h^3$. Ces majorants permettent d'obtenir les estimations :

$$\left| f'(x_0) - \frac{f(x_0 + h) - f(x_0 - h)}{2h} \right| \leq \frac{\int_0^h \int_0^k \omega_{f''}(2\delta) d\delta dk}{2h},$$

et dans le cas trois fois dérivable :

$$\left| f'(x_0) - \frac{f(x_0 + h) - f(x_0 - h)}{2h} \right| \leq \frac{M_3}{6}h^2.$$

La présence du terme en h^2 permet de comprendre pourquoi la formule symétrique est plus efficace.

Venons en au calcul approchée d'une dérivée seconde. La formule de Taylor à l'ordre 2 permet décrire que pour λh petit :

$$f(x_0 + \lambda h) \approx f(x_0) + f'(x_0)\lambda h + \frac{f''(x_0)}{2}\lambda^2 h^2.$$

On va se débarrasser du terme $f'(x_0)$ qui ne nous intéresse pas ici. Pour cela, on ajoute les formules obtenues avec $\lambda = 1$ et $\lambda = -1$, ce qui permet de dire que pour $h > 0$ petit :

$$f'(x_0 + h) + f'(x_0 - h) \approx 2f'(x_0) + f''(x_0)h^2,$$

d'où :

$$f''(x_0) \approx \frac{f(x_0 + h) + f(x_0 - h) - 2f(x_0)}{h^2}.$$

3.6.2 La méthode des différences finies en dimension 1.

Nous allons prendre un problème modèle, la recherche de solutions de l'équation différentielle :

$$-u'' + u = f(x),$$

où f est une donnée, continue sur l'intervalle d'étude $I = [a, b]$ et u est l'inconnue. On découpe l'intervalle $[a, b]$ en $N + 1$ sous-intervalles réguliers, en posant :

$$x_i = a + i \frac{b - a}{N + 1}, \quad i = 0, \dots, N + 1.$$

On note u_i les valeurs approchées que l'on prendra pour $u(x_i)$. Ici le pas est $h = \frac{b-a}{N+1}$, donc pour approximation de $-u'' + u$ on prend :

$$-\frac{u_{i+1} - 2u_i + u_{i-1}}{h^2} + u_i = \frac{1}{h^2} [-u_{i+1} + (2 + h^2)u_i - u_{i-1}].$$

Si notre problème est un problème de Cauchy :

$$\begin{cases} -u'' + u = f(x), \\ u(a) = \alpha, \quad u'(a) = \beta \end{cases}$$

alors il est raisonnable de poser $u_0 = \alpha$ et $u_1 = u(a+h) \approx u(a) + u'(a)h = \alpha + \beta h$. Connaissant u_0 et u_1 , la formule donnant l'équation approchée :

$$\frac{1}{h^2} [-u_{i+1} + (2 + h^2)u_i - u_{i-1}] = f(x_i), \quad i = 1, \dots, N$$

permet de calculer récursivement les termes u_i .

3.6. RÉSOLUTION D'EDP : CALCUL APPROCHÉ DE DÉRIVÉES ET DIFFÉRENCES FINIES

Considérons maintenant un problème très différent, il s'agit d'un problème aux limites :

$$\begin{cases} -u'' + u = f(x), \\ u(a) = \alpha, \quad u(b) = \beta \end{cases}$$

Dans ce cas, on remplace u_0 par α et u_{N+1} par β dans la série de relations :

$$\frac{1}{h^2} [-u_{i+1} + (2 + h^2)u_i - u_{i-1}] = f(x_i), \quad i = 1, \dots, N$$

ce qui donne le système :

$$A \begin{pmatrix} u_1 \\ \vdots \\ u_N \end{pmatrix} = \begin{pmatrix} f(x_1) + \frac{\alpha}{h^2} \\ f(x_2) \\ \vdots \\ f(x_{N-1}) \\ f(x_N) + \frac{\beta}{h^2} \end{pmatrix},$$

où la matrice A est tridiagonale, avec les termes $1 + \frac{2}{h^2}$ sur la diagonale, et $-1/h^2$ sur la sur et sous diagonale.

3.6.3 La méthode des différences finies en dimension 2 : calcul des dérivées partielles et résolution d'EDP.

Partie non rédigée cette année, voir cours oral.

Chapitre 4

Probabilités et Statistiques.

4.1 Nombres pseudo-aléatoires

4.1.1 Un système chaotique.

Creez le fichier `iterlogis.sci` :

```
function y=iterlogis(x,n);  
y=x;  
for i=2:n  
x=4x.*(1-x);  
y=[y;x];  
end; endfunction
```

Ce fichier prend en entrée un vecteur ligne x et un entier n au moins égal à 2, et sort un vecteur de taille $n \times 2$ de sorte que chaque colonne soit les itérées par la fonction logistique $x \mapsto 4x(1-x)$ de l'élément de la même colonne présent en première ligne. A l'aide de ce programme, comparez les suites issues des nombres 0.3 et 0.301 (on regardera l'évolution de l'erreur relative moyenne en fonction du nombre de termes calculés, et on pourra tracer sur un même graphique les deux orbites).

On peut tracer un histogramme des valeurs prises par la fonction par les 3000 premiers termes issus de $x_0 = 0.3$ en 30 classes par :

```
xbasc();X=iterlogis(0.3,3000);histplot(30,X)
```

En changeant le nombre de points et le x_0 , vous constaterez que sauf aux bornes, les valeurs prises semblent approximativement bien réparties (par exemple si l'on se restreint aux valeurs entre 0.3 et 0.7).

En fait, pour $a \in [0, 4]$, la fonction $x \mapsto ax(1-x)$ est bien un système dynamique sur $[0, 1]$, chaotique pour a assez voisin de 4. Ce qui est remarquable lorsque $a = 4$, c'est que l'on peut calculer x_t en fonction de x_0 :

Supposant que $x_t = \sin^2(\theta_t)$, trouver une relation de récurrence sur θ_t . En déduire

qu'en posant $\theta_0 = \arcsin(\sqrt{x_0})$, on a :

$$x_t = \sin^2(2^t \theta_0).$$

De cette formule, avec quelques connaissances élémentaires sur les sous groupes de \mathbb{R} , on explique pourquoi lorsque x_0 et x'_0 sont proches, pour tout N , il existe des indices plus grands pour lesquels x_n et x'_n seront éloignés (ou proches).

4.1.2 Nombres pseudo-aléatoires.

Le problème que nous nous posons maintenant est : comment générer des nombres pris “au hasard” entre 0 et 1 ? En termes plus mathématiques, la question se pose sous la forme suivante : si $X : \Omega \rightarrow [0, 1]$ est une variable aléatoire de distribution uniforme sur $[0, 1]$, comment produire une réalisation de X , c'est-à-dire $X(\omega)$ avec $\omega \in \Omega$? (ce qu'on appellera nombre pseudo-aléatoire par la suite) Et si (X_1, \dots, X_n) sont n variables indépendantes de même loi que X , comment générer une réalisation de (X_1, \dots, X_n) , soit $(X_1(\omega), \dots, X_n(\omega))$? Ces questions se posent car un processeur fonctionne de façon intégralement déterministe, donc a priori sans possibilité de produire du “hasard”. Pourtant, et c'est aussi ce que permet la fonction logistique vue plus haut, certaines fonctions déterministes peuvent approcher le “hasard”.

modulo(27, 10) (fonction modulo) : on calcule 27 modulo 10, soit le reste de la division euclidienne de 27 par 10 : on obtient 7 car $27 = 2 \times 10 + 7$. Testez aussi *modulo*(19, 7) ; faire suffisamment d'exemples jusqu'à avoir compris *modulo*.

Ouvrir un fichier `pseudo1.sci` et y créer une fonction `pseudo1` telle que : $y = \text{pseudo1}(x) = \text{modulo}(13x, 100)$. Retourner alors sur la fenêtre de commandes Scilab. Générer la suite (x_n) avec $n = 0, 1, \dots, 15$ telle que $x_{n+1} = \text{pseudo1}(x_n)$ et $x_0 = 1$. Par ce biais, on a généré des nombres entiers qui semblent se comporter comme des chiffres entre 0 et 99 pris au “hasard” de façon équiprobable et indépendante.

Cependant, on s'aperçoit, si on génère (x_n) avec $n = 0, 1, \dots, 25$ qu'un motif se répète de façon périodique. Lequel ? Quelle période ? Pourquoi ? Ceci met en défaut la prétention à générer ainsi des nombres “aléatoires”. SILAB dispose de deux fonctions permettant de générer des nombres “aléatoires”, `rand` et `grand`, ils sont basés notamment sur de telles suites congruentes, mais avec des nombres plus grands (voir l'aide de `grand`).

4.2 Les lois usuelles

4.2.1 Lois discrètes

On veut générer une réalisation d'une variable de Bernoulli X telle que $P(X = 1) = p$ et $P(X = 0) = 1 - p$, avec $p \in [0, 1]$. Que se passe-t-il si $p = 1$? Quelles sont les valeurs possibles pour X ? Quelle est la fonction de répartition de X , $F_X(x) = P(X \leq x)$? Créer alors une fonction appelée `bernoulli.sci` :

```
function x=bernoulli(p);
x=0;
u=rand() ;// On génère une réalisation u d'une va uniforme sur [0,1]
if (u>=p) // Test sur u
x=1; // Dans le cas où  $u < p$ , on reste avec  $x = 0$ 
end;
endfunction;
```

Essayons : `bernoulli(0.3)`. Faire d'autres essais. Comment faire pour vérifier empiriquement que l'on a bien généré une variable de Bernoulli ? Montrer théoriquement que c'est bien le cas (pour cela, il suffit de calculer F_X en posant $X = 1$ si $U \geq p$, et $X = 0$ sinon). Modifier votre fonction de façon à générer maintenant n réalisations indépendantes de X .

Passons maintenant à la génération d'une variable aléatoire discrète quelconque. On commence par un cas où la v.a. peut prendre trois valeurs (0, 1 et 2 pour fixer). Créer le fichier `VAdisc.sci` suivant, générant une réalisation d'une v.a. Y qui suit une loi discrète à valeurs dans $\{0, 1, 2\}$ étant données les probabilités d'avoir 0, 1 et 2 ($p0 = P(X = 0)$, $p1 = P(X = 1)$, $P(X = 2) = 1 - p0 - p1$) :

```
function [y]=VAdiscrete(p0,p1)
u=rand; y=0;
if (u>=p0) & (u<p0+p1)
y=1;
elseif (u>=p0+p1)
y=2;
end; endfunction;
```

Faire des essais, puis transformer la fonction pour simuler n réalisations indépendantes de Y .

Exercice 4.2.1 1. *Suivant la procédure précédente, simuler k réalisations indépendantes d'une v.a. suivant une loi binomiale $\mathcal{B}(2, 0.3)$. N'y-aurait-il pas un autre moyen de simuler une telle variable (penser à une somme de v.a.) ? En déduire une fonction simulant k réalisations indépendantes d'une v.a. suivant une loi binomiale $\mathcal{B}(n, p)$, où n et p sont des paramètres.*

2. *On aimerait simuler une réalisation d'une variable aléatoire X de Poisson*

de paramètre $\lambda > 0$ (on rappelle qu'alors $P(X = k) = \frac{\lambda^k}{k!} e^{-\lambda}$ pour $k \in \mathbb{N}$).
A quels problèmes sera-t-on confrontés ?

4.2.2 Lois uniformes

Si X suit une loi uniforme sur $[a; b]$, alors $(X - a)/(b - a)$ suit une loi uniforme sur $[0; 1]$. L'étude des lois uniformes se ramène donc à l'étude de la loi uniforme sur $[0; 1]$. Pour celle-ci, tout est simple :

- la densité de la loi est $1_{[0;1]}$,
- sa fonction de répartition est (sous forme résumée) $x \mapsto \max\{0; \min\{x; 1\}\}$.

De plus, on dispose dans SCILAB d'un générateur de matrices dont les coefficients suivent la loi uniforme, il s'agit de `rand`.

Vous aurez souvent à tracer des histogrammes d'observations issues de réalisation de lois. Vous disposez pour cela de la commande `histplot`. `histplot(n, x)` trace un histogramme en regroupant en n classes égales les valeurs de x .

Exercice 4.2.2 *Simulez plusieurs réalisations de la loi uniforme sur $[0; 1]$. Tracez des histogrammes, et comparez les moyennes et variances empiriques aux valeurs exactes, qui sont respectivement $1/2$ et $1/12$.*

4.2.3 Lois normales

Si X suit une loi normale $N(m, \sigma^2)$, alors $(X - m)/\sigma$ suit une loi $N(0; 1)$: on se ramène donc à la loi $N(0; 1)$. Cette fois, la situation est un peu moins agréable, puisque, si la densité est la fonction $x \mapsto \frac{1}{\sqrt{2\pi}} \exp(-x^2/2)$, sa fonction de répartition ne s'exprime pas à l'aide des fonctions usuelles, c'est :

$$\Phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^x \exp(-u^2/2) du.$$

SCILAB dispose de la fonction d'erreur, notée `erf`, qui est définie par :

$$\forall x \in \mathbb{R}, \quad \text{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x \exp(-t^2) dt$$

qui est donc une fonction régulière, impaire, strictement croissante et tendant vers 1 à l'infini. SCILAB dispose aussi de la fonction `erfinv` qui est la bijection réciproque de `erf`. Un calcul élémentaire montre que Φ peut s'exprimer à l'aide de `erf` et que plus précisément :

$$\Phi(x) = \frac{1}{2} (1 + \text{erf}(x/\sqrt{2})).$$

Rappelons que l'on peut simuler une matrice dont les coefficients suivent des lois normales centrées réduites par l'instruction `rand` avec l'option `"normal"`.

Une commande particulièrement pratique est la commande `cdfnor`. Il en existe d'ailleurs des équivalentes pour d'autres lois que l'on peut voir apparaître listées grâce à `apropos cdf`. Etant donnée une v.a. Y suivant une loi normale de moyenne `Mean` et d'écart-type `std`, on note pour chaque X , P la proba que $Y \leq X$ et $Q = 1 - P$. La connaissance de trois des quatre variables parmi `Mean`, `Std`, X , (P, Q) permet la connaissance de la dernière, par :

- `[P,Q]=cdfnor("PQ",X,Mean,Std)` (`cdfnor("PQ",X,Mean,Std)` donne P).
- `[X]=cdfnor("X",Mean,Std,P,Q)`
- `[Mean]=cdfnor("Mean",Std,P,Q,X)`
- `[Std]=cdfnor("Std",P,Q,X,Mean)`

Ainsi, l'utilisation de `erf` est en fait inutile.

Exercice 4.2.3 *On suppose que les notes des étudiants à un partiel sont réparties suivant une loi normale $N(11; 2)$.*

1. *Quelle est le pourcentage d'étudiants dépassant la note de 14 ?*
2. *Quelle est la note pour laquelle on a 60% de l'effectif au dessus ?*
3. *Simulez un échantillon de 1000 valeurs de la loi normale $N(11; 2)$ et comparez avec les valeurs obtenues ci-dessus.*

4.2.4 Simulation d'autres lois usuelles

On dispose tout d'abord d'un générateur généralisant `rand`, il s'agit de `grand` (cf. aide). Pour des raisons purement pédagogiques, je vais plutôt illustrer la méthode de la fonction de répartition. Supposons que l'on veuille simuler d'autres lois dont on connaisse la fonction de répartition F et explicitement son inverse. Alors, si X suit une loi uniforme sur $[0; 1]$, la loi de $F^{-1}(X)$ est la loi de fonction de répartition F (F^{-1} désigne la bijection réciproque de F et non $1/F$).

Exercice 4.2.4 *La loi exponentielle de paramètre λ a pour fonction de répartition $F_\lambda(x) = (1 - e^{-\lambda x})1_{\mathbb{R}^+}(x)$. Simulez pour plusieurs valeurs de λ plusieurs échantillons suivant la loi exponentielle de paramètre λ . Même question avec la loi de Cauchy dont la densité sur \mathbb{R} est : $x \mapsto \frac{1}{\pi(1+x^2)}$.*

4.3 Théorèmes limites en Calcul des Probabilités.

Sachant maintenant simuler des variables aléatoires, on commence par mettre en évidence la Loi des Grands Nombres et l'importance de ses hypothèses. Par la suite, on visualise des applications du Théorème de Limite Central. Enfin, on revient sur l'utilisation d'un histogramme pour visualiser la loi de probabilité d'une variable aléatoire.

4.3.1 Loi des Grands Nombres

On rappelle l'énoncé de ce théorème (appelé "Loi" car on a longtemps cru que c'était un résultat de physique et non de mathématiques). On considère des variables aléatoires $(X_n)_{n \in \mathbb{N}^*}$, indépendantes et de même loi qu'une variable X . Si on suppose que $\mathbb{E}|X| < \infty$, alors en probabilité :

$$\bar{X}_n = \frac{1}{n}(X_1 + X_2 + \dots + X_n) \rightarrow m \quad \text{avec } m = \mathbb{E}X.$$

Intuitivement, cela signifie que la moyenne empirique converge vers la moyenne théorique quand on considère un nombre croissant de variables aléatoires.

- Exercice 4.3.1**
1. *Faire un programme calculant la moyenne d'un nombre n (arbitraire) de réalisations de variables uniformes sur $[-2, 3]$, et qui trace en fonction de n cette moyenne. Que constate-t-on? Faites la même chose avec des réalisations indépendantes d'une v.a. distribuée d'abord selon une loi normale $\mathcal{N}(2, 3^2)$, puis suivant une loi de Cauchy (que l'on génère à partir de la tangente d'une variable uniforme sur $]-\pi/2, \pi/2[$). Conclusions ?*
 2. *L'exercice précédent montrait l'importance de l'hypothèse $\mathbb{E}|X| < +\infty$. Pour montrer l'importance de la notion d'indépendance, on suppose que pour tout i , on a $X_i = X_1$. La Loi des Grands Nombres est-elle encore vérifiée ? Maintenant, on considère que les X_i sont bien indépendantes, mais on veut appliquer la Loi des Grands Nombres aux variables $Y_i = \frac{X_1 + \dots + X_i}{i}$ (donc les Y_i ne sont pas, en général, indépendantes les unes des autres). La Loi des Grands Nombres est-elle vérifiée pour les lois considérées précédemment ?*

4.3.2 Application de la Loi des Grands Nombres : distribution d'une variable aléatoire.

Une application intéressante de la Loi des Grands Nombres est le fait que lorsque l'on a de nombreuses réalisations indépendantes d'une variable aléatoire, alors l'histogramme, s'il possède suffisamment de classes, se rapproche de la loi (densité) de probabilité. Cela s'explique par le fait que la proportion empirique de variables dans $[a, b]$ tend vers la probabilité théorique que la variable appartienne à $[a, b]$ (cette probabilité est en fait l'espérance de la fonction qui vaut 1 si la variable est dans $[a, b]$ et 0 sinon).

Répétez plusieurs fois cette ligne, en changeant éventuellement le nombre de v.a. simulées ou le nombre de classes de l'histogramme :

```
X=rand(100,1); xbaso(); hist(10,X)
```

Commentez le résultat obtenu. Reprendre la même question avec une loi normale

puis une loi de Cauchy. Expliquer pourquoi cela “marche” pour Cauchy alors que $\mathbb{E}|X| = \infty$.

4.3.3 Théorème Central Limite

Rappelons d’abord un énoncé de ce théorème, qui est en fait une sorte de raffinement par rapport à la Loi des Grands Nombres : on considère des variables aléatoires $(X_n)_{n \in \mathbb{N}^*}$, indépendantes et de même loi qu’une variable X . Si on suppose que $\mathbb{E}|X| < \infty$, et que $\mathbb{E}|X|^2 < \infty$, alors on a en loi :

$$\sqrt{n} \left(\frac{\bar{X}_n - m}{\sigma^2} \right) = \sqrt{n} \left(\frac{\frac{1}{n}(X_1 + X_2 + \dots + X_n) - m}{\sigma^2} \right) \rightarrow \mathcal{N}(0, 1),$$

avec $m = \mathbb{E}X$ et $\sigma^2 = \mathbb{E}X^2$. Cela signifie, de façon intuitive, que quand n est grand alors la moyenne empirique converge vers la moyenne théorique suivant une répartition gaussienne $\mathcal{N}(0, \frac{\sigma^2}{n})$.

On recommence la situation d’un jeu de pile ou face. Cela correspond aussi à la situation d’un sondage d’opinion pour choisir entre A et B . Ainsi, si la proportion d’intention de vote pour A sur l’ensemble de la population est p , on peut modéliser chaque votant par X_i , et lorsque $X_i = 1$ se traduit par un vote pour A . Alors la somme des X_i pour $i = 1$ à n représente le nombre total d’intentions de vote pour A parmi n personnes et la moyenne empirique \bar{X}_n représente la fréquence des intentions de vote pour A parmi les n personnes. On considère n fois 100 personnes sondées. La probabilité de voter pour le candidat A est p . La ligne suivante trace l’histogramme de X_n :

```
n=100;p=0.3;X=(rand(n,100)<p)+0; xbase();histplot(10,mean(X,'c'))
```

Faites plusieurs essais avec ces valeurs, puis augmentez n . Que constatez-vous ?
Mêmes questions en changeant p . Refaire des histogrammes (comme ci-dessus) sur la moyenne empirique dans le cas d’une loi exponentielle de paramètre 2, puis une loi de Cauchy. Conclusions ?

4.4 Estimation ponctuelle et ensembliste

Cette section illustre les résultats d’estimations ponctuelles et ensemblistes.

On rappelle que la moyenne d’une loi exponentielle est $1/\lambda$ et sa variance est $1/\lambda^2$. Désignant par \bar{X}_n la moyenne empirique dans le modèle d’échantillonnage, ceci implique que $\sqrt{n}(\bar{X}_n - 1/\lambda)$ tend en loi vers une loi normale $N(0, 1/\lambda^2)$. Posons $Y_n = 1/\bar{X}_n$. L’application du théorème Δ assure alors que $\sqrt{n}(Y_n - \lambda)$ tend en loi vers une loi normale $N(0, \lambda^2)$.

Exercice 4.4.1 *Simulez n réalisations de loi exponentielle de paramètre λ connu ($n \geq 1000$). Calculez les estimateurs \bar{X}_n et Y_n de $1/\lambda$ et de λ . Vous semblent-ils biaisés ?*

D'après ce que nous avons dit avant, $\Pi_n = \sqrt{n}(Y_n - \lambda)/Y_n$ tend en loi vers une $N(0; 1)$. C'est donc une fonction pivotale asymptotique pour λ . Etant donné un seuil d'erreur α , on sait que :

$$\lim_{n \rightarrow +\infty} P(|\Pi_n| \leq u_{1-\alpha/2}) = 1 - \alpha.$$

En général, on s'affranchit du passage à la limite. Prenons $\alpha = 5\%$, auquel cas $u_{1-\alpha/2} = 1.96$. On a alors, avec une probabilité d'erreur 5% :

$$Y_n \left(1 - \frac{1.96}{\sqrt{n}}\right) \leq \lambda \leq Y_n \left(1 + \frac{1.96}{\sqrt{n}}\right).$$

Exercice 4.4.2 *Simulez des lois exponentielles de paramètre connu, et donnez des intervalles de confiance. La vraie valeur fait-elle partie de l'intervalle de confiance obtenu ?*

4.5 La méthode de Monte-Carlo et ses applications.

Nous présentons ici la méthode de Monte-Carlo de calcul d'espérances, et nous donnons des illustrations de ses applications aux calculs approchés d'intégrales et à la résolution des EDP issues de la finance.

Le principe de la méthode de base est simple : on est amené à estimer une espérance $\mathbb{E}_P(\phi(X))$ où la loi P a une densité f , dans le but de calculer l'intégrale :

$$\int \varphi(x)f(x)dx = \int \varphi(x)dP(x) = \mathbb{E}_P(\phi(X)).$$

On simule n réalisations i.i.d. de notre v.a., X_1, \dots, X_n et on remplace l'intégrale par le moment empirique :

$$\frac{1}{n} \sum_{i=1}^n \phi(X_i).$$

La convergence et sa vitesse sont données par la loi des grands nombres et le théorème central-limite :

$$\sqrt{n} \left(\frac{1}{n} \sum_{i=1}^n \phi(X_i) - \mathbb{E}_P(\phi(X)) \right) \rightarrow N(0, V_P(\phi(X))).$$

Il est donc possible de construire des régions de confiance, puisque si l'on désigne par V_n la matrice de variance-covariance empirique de $\phi(X)$, la fonction :

$$\Pi_n = \sqrt{n}V_n^{-1/2} \left(\frac{1}{n} \sum_{i=1}^n \phi(X_i) - \mathbb{E}_P(\phi(X)) \right)$$

est pivotale asymptotique de loi limite $N(0, Id)$. La mise en oeuvre est donc en fait très simple :

1. On simule n v.a. X_i de loi P . On calcule $Y_i = \phi(X_i)$.
2. On calcule la moyenne des Y_i ce qui donne une approximation de $\mathbb{E}_P(\phi(X))$.
3. si l'on veut un intervalle de confiance, on calcule aussi la matrice de var-covar. des Y_i et on calcule alors Π_n d'où une région de confiance (nous ne le ferons ici qu'en dimension 1).

4.5.1 Application aux calculs d'intégrales

Sous la forme la plus simple, lorsqu'on calcule des intégrales sur des intervalles compacts de fonctions sympathiques, on peut considérer que la loi de P est la loi uniforme sur le compact considéré.

Exercice 4.5.1 *En simulant n v.a. uniformes sur $[0; 1]$ (augmenter n en partant de 500), donner plusieurs valeurs approchées de l'intégrale :*

$$\int_0^1 \sqrt{1-x^2} dx.$$

1. On comparera les valeurs obtenues à la valeur exacte qui est $\pi/4$ en calculant l'erreur relative.
2. Construire des intervalles de confiance à 95%. La vraie-valeur en fait-elle partie ?

On prend maintenant le problème d'une intégrale multiple (ici double). On simule donc à chaque étape un couple de v.a. uniformes.

Exercice 4.5.2 *Calculer de manière analogue une valeur approchée de l'intégrale double :*

$$\int_0^1 \int_0^1 x^y dx dy$$

et comparez avec la valeur exacte ($\ln(2)$). On comparera notamment la vitesse de convergence avec celle constatée dans la première question de l'exercice précédent.

On n'est pas obligé de considérer des lois uniformes. Par exemple, si l'on veut intégrer sur un intervalle non borné, il est plus judicieux de choisir d'autres lois :

Exercice 4.5.3 1. La loi exponentielle de paramètre λ a pour densité :

$$\lambda \exp(-\lambda x) 1_{\mathbb{R}^+}(x).$$

Utilisez cette remarque pour calculer pour plusieurs valeurs de λ :

$$\int_0^{+\infty} e^{-\lambda x} \sqrt{x} dx$$

et comparez à la valeur exacte $\sqrt{\pi}/(2\lambda^{3/2})$.

2. En utilisant des lois normales, calculez une valeur approchée de l'intégrale :

$$\int_{-\infty}^{+\infty} e^{-x^2} \cos(x) dx.$$

4.5.2 Application aux EDP issues de la finance

La formule de Feynman-Kac permet d'exprimer dans certains cas les solutions d'une EDP sous forme d'une espérance conditionnelle. Considérons par exemple le problème parabolique issu du modèle de Samuelson (Rational Theory of warrant prices, *Indust. Manag. Rev.*, 6, pp.13-31, 1965) :

$$\begin{cases} \frac{\partial v}{\partial t}(t, x) + \frac{\sigma^2}{2} \frac{\partial^2 v}{\partial x^2}(t, x) = 0 & \text{sur } [0; T[\times \mathbb{R} \\ v(T, x) = \phi(x) & \text{sur } \mathbb{R}. \end{cases}$$

La solution est donnée par :

$$v(t, x) = \mathbb{E}\phi(x + \sigma(W_T - W_t)),$$

où $(W_t)_t$ est un Brownien standard. Prenons le cas où $T = 1$ et $\phi(x) = (x - 0.5)^+$. On est donc intéressé par la fonction $x \mapsto v(0, x)$. Remarquant que $W_1 - W_0$ suit une loi normale $N(0, 1)$, on calcule une valeur approchée de $v(0, x)$ en simulant n v.a. X_i suivant $N(0, 1)$ et l'on calcule la moyenne des $\phi(x + \sigma X_i)$.

Exercice 4.5.4 Prenons $\sigma = 1$. Tracer les fonctions $x \mapsto v(0, x)$ et ϕ sur un même graphique.

Partie I

**Introduction à MAPLE et à
quelque-unes de ses applications.**

Chapitre 5

Généralités sur MAPLE

Avertissement : ce chapitre n'est pas au programme, et concerne la version très ancienne qui est sur le réseau de Paris 1.

5.1 Petit tour d'horizon et description de l'environnement de travail

Sur la feuille de calcul, les commandes apparaissent en rouge. Tout texte suivi de # est un commentaire. Pour exécuter une commande, taper RETURN après avoir tapé la ligne. Les résultats sont affichés en bleu. Respectez scrupuleusement la syntaxe, et en particulier les points-virgules ou deux-points, ainsi que les majuscules et minuscules.

Manipulation d'entiers :

```
> 25! ;  
> n:=% ;  
> n-12^6 ;
```

Essayez d'effectuer la feuille dans un ordre différent en terminant par la seconde ligne.

Manipulation de rationnels :

```
> a:=1/4 ; b:=5/6 ;  
> a+b ;  
> 350/60 ;
```

Manipulation de réels :

```
> sin(Pi/4) ; sin(PI/4) ; sin(pi/4) ;  
Commentaire ?
```

```
> sin(Pi/60) ;
```

```
> evalf(sin(Pi/60)) ;
```

Une manière d'invoquer l'aide sur une commande dont on connaît le nom est de taper ? suivi du nom de commande :

```
> ?evalf ;
```

(vous quitterez l'aide en allant dans la barre d'outils : Fichier, Close Help Topic).

```
> evalf(sin(Pi/60),60) ;
```

```
> a:=(sqrt(2)-1)*(sqrt(2)+1); evalf(a); evalf(a,30);
```

```
> expand(%) ;
```

```
> t:=1/(1+sqrt(5)) ;
```

```
> rationalize(t) ;
```

```
> sin(1/2) ; sin(0.5) ; evalf(sin(1/2)) ;
```

Nombre de chiffres significatifs :

Ce nombre est fixé par la variable d'environnement `Digits`.

```
> Digits ;
```

```
> x:=1/sqrt(3) ;
```

Quel est le rôle de : ?

```
> Digits:=3 ; evalf(x) ; Digits:=10 ; evalf(x) ;
```

```
> evalf(x,3) ; evalf(x) ;
```

Que vous inspirent ces lignes ?

Manipulation de nombres complexes :

L'unité imaginaire se note `I`.

```
> u:=(1+2*I) ; v:=(1/2-I) ;
```

```
> u+v ; u/v ; abs(u) ;
```

```
> u^3 ;
```

```
> (1+sqrt(2)*I)*(1/sqrt(2)-I) ;
```

Simplifiez cette expression en utilisant `evalc` (regardez l'aide en ligne).

```
> u^(1/2) ; u^(0.5) ;
```

Commentaires ?

Exercice 5.1.1 *Exercice : trouvez les réels a et b tels que $u^{1/2}=a+I*b$.*

5.2 Premières commandes

La première commande de survie est bien entendu l'aide en ligne. Elle est mieux documentée que celle de MATLAB, souvent il suffit de lire les exemples et d'utiliser un copier-coller. On peut bien entendu se débrouiller à l'aide du sommaire. Si l'on veut juste vérifier la syntaxe d'une commande dont on connaît

le nom, il suffit en fait de taper ? suivi du nom de la commande (par exemple ?plot) ou de taper plot, de le mettre en surbrillance, et d'aller dans le menu d'aide qui proposera une entrée "Help on plot".

Une commande se termine toujours par ; (si on veut afficher le résultat) ou par : dans le cas contraire ; sinon on s'expose à un message d'erreur.

Une affectation se fait par :=, ce qui est la notation informatique courante. Une variable à laquelle rien n'est affecté est dite libre. Une réinitialisation de l'ensemble des variables se fait par la commande restart, tandis qu'une réinitialisation de la variable var se fait par : var:='var' (cette syntaxe sera expliquée plus tard).

Il faut faire attention qu'en MAPLE, beaucoup de procédures (mais pas toutes) évaluent tous les arguments avant d'effectuer le calcul. La syntaxe pour calculer $\sum_{i=1}^{10} i^2$, lorsque i est libre, est :

```
> sum(i^2 , i=1..10) ;
```

mais lorsque i contient déjà une valeur, ceci aboutit à un message d'erreur. Ceci est dû au fait que lorsque $i = 2$ (par exemple), MAPLE remplace tous les i par 2 qui ne sont plus des indices de sommations. Il faut donc soit réinitialiser i , soit changer le nom de l'indice, soit encore utiliser une syntaxe expliquée plus bas ; essayez :

```
> i:=2 : sum(i^2 , i=1..10) ;
> i:=2 : sum('i'^2 , 'i'=1..10) ;
> i:=2 : sum('i^2' , 'i'=1..10) ;
```

Un commentaire sur une ligne de MAPLE input se met après #. Tout ce qui est après ce caractère n'est pas interprété.

Les symboles % (resp. %, %%) désignent le dernier (resp. l'avant dernier, resp. l'antépénultième) résultat calculé ayant donné un résultat non vide. Il faut faire attention au fait qu'il ne s'agit pas nécessairement du résultat immédiatement au dessus dans les lignes MAPLE. Exécutez par exemple :

```
> 1+1;
> %^2+1;
> 1+3;
```

puis remontez à la ligne 2 et réeffectuez. Commentaire ?

5.3 Expressions et fonctions

5.3.1 Fonctions

On peut définir avec MAPLE des fonctions d'une ou plusieurs variables. Dans le second cas, il ne faut pas oublier les parenthèses. La syntaxe est assez proche

de la notation mathématique, et les variables utilisées pour définir la fonction peuvent ne pas être libres.

```
> f:=x->x^2+1;
> g:=(x,y)->x^2-y^2;
> f(sqrt(2)) ; f(3) ;
> x:=2 ; f(x) ; f ; eval(f) ;
> g(1,y) ;
> t:=3 : h:=t->t^3+1 : h(2) ;
```

5.3.2 Expressions

Une expression n'est définie qu'à l'aide de variables libres, sans quoi elle est immédiatement évaluée.

```
> x:=2 ; P:=x^2+1 ;
> x:='x' ; P:=x^2+1 ;
> x:='x' : y:='y' : Q:=x^2-y^2 ;
```

Pour calculer une valeur d'une expression avec des valeurs particulières des variables, on peut soit affecter ces valeurs aux variables (qu'elles conservent alors), soit utiliser la fonction `subs` qui permet de conserver les variables libres en dehors du calcul de l'expression.

```
> x:=2 : P ; x ;
> x:='x' ; P:=x^2+1 ;
> x:='x' : subs(x=2,P) ; x ;
```

5.3.3 Passages entre fonctions et expressions

Pour passer d'une fonction à une expression (impliquant des variables libres), il suffit d'évaluer la fonction en ces variables :

```
> x:='x' : y:='y' ; g(x,y) ;
```

Pour passer au contraire d'une expression à une fonction, on utilise la fonction `unapply`.

```
> unapply(P,x) ;
> unapply(Q,x,y) ;
```

5.4 Quelques fonctions et constantes prédéfinies

Nous n'en signalons à titre informatif que quelques-unes. Ces objets sont protégés, c'est-à-dire qu'il n'est pas possible (par défaut) de leur attribuer autre chose via une affectation. Il faut faire attention au fait que dans tous les identificateurs, MAPLE distingue majuscules et minuscules. Ainsi, si `Pi` désigne le nombre d'Archimède (3.14...), `pi` désigne la lettre grecque minuscule et `PI` la lettre grecque majuscule.

Il y a d'abord trois constantes booléennes sur lesquelles nous reviendrons `true`, `false`, `FAIL`.

Il y a ensuite des constantes usuelles : `Pi` (3.14...), `I` (complexe qui est une racine carrée de -1), `infinity` (sans commentaire...), `gamma` (constante d'Euler 0.57721...). A partir de MAPLE version 5, on ne dispose plus de `E` pour désigner le nombre de Neper (2.718...) Il faut l'écrire `exp(1)`.

Fonctions logarithmiques et exponentielles : `exp`, `log` (ou `ln`), `log10` (après avoir tapé `readlib(log10)`), `log[b]`.

Fonctions de parties entières ou fractionnaires : `round`, `trunc`, `frac`, `floor`, `ceil` (essayez-les avec -1.72, -1.23, 1.23, 1.72 pour voir les différences).

Fonctions trigonométriques : `sin`, `cos`, `tan`, `cot`, `arcsin`, `arccos`, `arctan`, `arccot` (ajoutez un `h` pour les fonctions hyperboliques).

signes et fonctions complexes : `abs`, `conjugate`, `csgn`. (`csgn(z)` est le signe de `Re(z)` si ce terme est non nul, et celui de `Im(z)` sinon).

divers : `GAMMA` (fonction Eulérienne Γ), `factorial` (factorielle), `binomial` (coefficient C_n^p).

5.5 Types de variables

MAPLE dispose d'une part d'un certain nombre de types de base, et d'autre part de types déduits de ces types de base. Un type de base d'un objet MAPLE est donné par la commande `whattype`. Nous allons ci-dessous discuter les principaux types que vous rencontrerez.

MAPLE effectue de son propre chef les simplifications élémentaires (`x-x`) et les calculs sur les rationnels. Par exemple, il effectuera :

`(2+2/3)*(1-2/7)`

`(2-sqrt(2))*(1-4/5)` (car il effectuera la seconde soustraction)

mais laissera tel quel :

`(1-sqrt(2))*(1+sqrt(2))`

Les types discutés ci-après sont ceux obtenus après ces simplifications.

5.5.1 Expressions élémentaires

Le type d'une expression correspond toujours à son opérateur le plus externe. Les tous premiers types sont `+`, `*`, `^`, `function`. Voici à quoi ils correspondent sur une expression simple :

1. `+` : `x+y` ou `x-y` (qui est pour MAPLE `x+(-1)*y`).

2. `*` : `x*y` ou `x/y` (qui est pour MAPLE `x*y^(-1)`).
3. `^` : `x^y` ou `sqrt(x)` (que MAPLE voit comme `x^(1/2)`), ou `1/x` (qui est pour MAPLE `x^(-1)`).
4. `function` : n'importe quoi invoquant des fonctions : `sin(2)`, `GAMMA(x+2)`,...

Comme je vous l'ai dit plus haut, MAPLE considère le type par l'opérateur le plus externe. L'expression `2*x+sin(3)-7*y` est pour MAPLE de type `+`, composée des trois sous-expressions `2*x`, `sin(3)` et `-7*y` qui sont elles-mêmes respectivement de type `*`, `function`, `*`.

`nops` retourne le nombre de sous-expressions (ici 3), et l'on peut accéder à chacune d'elles par la commande `op`. Par exemple, `op(1,2*x+sin(3)-7*y)` retournerait ici `2*x`. `op(0,expr)` retourne le type élémentaire de `expr`, ici `+`. Dans le cas d'un type `function`, il retourne la fonction mise en jeu.

Il est souvent utile de dessiner des expressions complexes à l'aide d'un arbre informatique, dont la racine est vers le haut. A chaque noeud se trouve l'opérateur et sur chaque branche chacune des sous-expressions, pouvant donner elles-mêmes lieu à des sous branches. A l'aide de `op`, on peut alors accéder à chacune des sous-expressions.

Exercice 5.5.1 *En vous aidant de `op`, dessinez un arbre informatique pour chacune des expressions suivantes, puis à l'aide de MAPLE, accédez à la sous-expression demandée.*

- $x + 2y + z$, sous-expression z .
- $\sqrt{x^2 + 1}$, sous-expression 1.
- $\sqrt{t + \sin(z^2 - 1)} / (z - 2u)$, sous-expression z^2 .
- $\sin(x + \cos(x + y + z))$, sous-expression $y + z$ (il sera peut-être nécessaire d'ajouter deux sous-expressions).

5.5.2 Valeurs

On dispose de trois types de base, `integer`, `fraction`, `float`, qui désignent respectivement un entier (positif ou négatif), un nombre rationnel (quotient de deux entiers) et un flottant, c'est-à-dire un nombre muni d'un point décimal qui devient pour MAPLE une approximation. Ainsi, en MAPLE, 1.5 et 3/2 ne sont pas les mêmes termes, le premier est du type `float`, le second de type `fraction`. Le premier est pour MAPLE une approximation du second, tout autant d'ailleurs qu'il serait une approximation de $3/2 + 10^{-20}$ si l'on est avec 10 chiffres significatifs. Lorsque l'on dispose d'un ensemble de valeurs entières ou fractionnaires et que

l'on souhaite que MAPLE effectue un calcul approché (et non exact), il faut lui mettre un terme en flottant, par exemple on peut rajouter un point après un entier : si 1 est transformé en, 1., alors MAPLE effectue tous les calculs en flottant.

Comparez les réponses à ces deux instructions :

```
> 2^(1/2)
```

```
> 2^(0.5)
```

On peut également utiliser l'instruction `evalf` : `evalf(expr)` donne une valeur approchée de `expr` qui est celle-ci un flottant. MAPLE stocke les flottants suivant leur mantisse et leur exposant, que l'on récupère par `op(i,...)` avec `i=1` pour la mantisse et `i=2` pour l'exposant. L'appel d'`evalf` entraîne l'écriture d'un nombre de chiffres significatifs égaux à la variable d'environnement `Digits`, qui par défaut vaut 10. Elle est bien entendu modifiable, mais si l'on veut juste une évaluation d'une seule expression `p` avec `n` chiffres significatifs, il est préférable d'écrire `evalf(p,n)`. Comparez :

```
> evalf(Pi) ; evalf(sqrt(2)) ;
```

```
> Digits:=20 : evalf(Pi) ; evalf(sqrt(2)) ;
```

```
> Digits:=10 : evalf(Pi,30) ; evalf(sqrt(2)) ;
```

MAPLE propose aussi d'autres types, qui ne sont pas des types de base (qui ne sont pas renvoyés par `whattype`), que l'on peut tester par `type(expr,ty)`. Si `expr` est du type `ty`, on obtient `true`, sinon `false` (voir sous-section suivante). Ces types sont :

- `rational` : réunion de `fraction` et `integer` ;
- `numeric` : réunion de `rational` et de `float` ;
- `realcons` : n'importe quoi dont MAPLE peut donner une valeur approchée (réelle) par `evalf`, par exemple des choses type `numeric`, mais aussi des choses faisant intervenir des fonctions sur des types `numeric` et faisant intervenir des constantes que l'on peut approcher numériquement.

Par exemple, sont de type `realcons` les expressions suivantes :

```
Pi, sqrt(2), cos(7), sin(sqrt(2+gamma))
```

mais si `x` est libre, `sin(x)` n'est pas du type `realcons`, et `arcsin(2)` non plus (c'est un nombre complexe non réel).

Venons-en aux nombres complexes. Ils sont de la forme `a+I*b` avec `a` et `b` réels. Les types de base sont les mêmes que pour les réels (c'est l'opération la plus externe qui définit le type de base). On dispose d'un type (non de base), `complex` permettant de tester si un nombre est complexe. On peut lui ajouter entre parenthèses un sous-type (pas nécessairement de base), permettant de tester si partie réelles et imaginaires sont de ce type. Par exemple :

```
> type(expr,complex(numeric)) ;
```

donne `true` si `expr=a+I*b` où `a` et `b` sont tous deux de type `numeric`, et `false` dans tous les autres cas.

5.5.3 Relations et booléens

Au niveau des relations, MAPLE dispose du type de base `relation`, qui se décompose en plusieurs sous-types (non de base) : `=`, `<>`, `<`, `<=` que l'on peut tester par `type` ; il faut alors marquer ces types entre apostrophes inversées. Les signes supérieur (ou égal, ou bien strictement), rentrent dans les types des signes inférieurs en échangeant les deux membres. Testez par exemple :

```
> type(x>3 , `<`);
```

On récupère les membres de droite et de gauche par `rhs` et `lhs`. Ici, bien que `x` soit à gauche du signe, on le récupère par `rhs` puisque l'inégalité `x>3` est stockée `3<x`.

On dispose d'un type `boolean` comprenant notamment :

- le type `relation` testé ci-dessus.
- le type `logical`, contenant les connecteurs logiques `and`, `or`, `not`, toujours placés entre apostrophes inversées.
- les constantes logiques `true`, `false`, `FAIL` (la dernière correspondant au cas où MAPLE ne peut répondre sans hypothèse supplémentaire).

5.5.4 Intervalles

Leur type est `range` ou `..` (les points étant placés entre apostrophes inversées dans la définition du type). Par exemple, `In:=a..b` définit l'intervalle $[a, b]$ lorsque $a \leq b$ (testez un exemple dans l'autre cas). Pour récupérer `a` (resp. `b`), on tape `op(1,In)` (resp. `op(2,In)`).

Attention : si vous définissez un intervalle avec des bornes `a` et `b` qui sont du type `realcons` mais pas du type `numeric`, utilisez toujours `evalf`, par exemple, on écrira :

```
> Jn:=evalf(sin(2))..1 ;
```

5.5.5 Séquences

Une séquence est ici une suite de termes séparées par des virgules. On peut concaténer deux séquences en écrivant leurs noms successivement :

```
> s1 := Matin, Apres-midi ;
> s2 := Soir ;
> s := s1 , s2 ;
```

Leur type reconnu par `whattype` est `exprseq`. L'élément neutre s'appelle `NULL` et permet à MAPLE de toujours renvoyer quelque chose, même si MAPLE n'écrit pas explicitement `NULL` (voir un exemple plus bas).

La fonction `seq` est utile pour créer une séquence dont le i ème terme s'écrit $f(i)$. Plus précisément, `seq(f(i), i=a..b)` renvoie la séquence $f(a)$, $f(a+1)$, ..., $f(a+\text{floor}(b-a))$. Testez cette commande sur les exemples suivants :

```
> seq(i^2 , i=1..10) ;
> seq(2*i , i=0.5..7) ;
```

On dispose d'un générateur particulier de séquence, l'opérateur `$`. `expr$p` renvoie une séquence de p termes tous égaux à `expr`. C'est surtout utile avec la commande `diff`, que nous verrons dans un chapitre ultérieur. Pour calculer la dérivée quatrième de la fonction $x \mapsto \sin(x^2)$, il faudrait écrire en MAPLE :

```
> diff(sin(x^2), x, x, x, x) ;
```

que l'on abrège en :

```
> diff(sin(x^2), x$4) ;
```

Les séquences apparaissent notamment lors de réponses fournies par MAPLE. Lorsque P est un polynôme à coefficients réels, la commande `fsolve` sans paramètre donne des valeurs approchées des racines réelles de P . Essayez les deux exemples suivants :

```
> fsolve(x^2-5*x+6) ;
> S:=fsolve(x^2+3) ;
> evalb(S=NULL) ; # donne true ssi S=NULL
```

Dans le second, MAPLE donne en fait séquence `NULL` qu'il écrit sous forme d'une liste vide ; c'est l'élément neutre de l'opération de concaténation.

Signalons enfin que l'on a accès à l'élément d'indice i dans la séquence `s` en tapant simplement `s[i]`. Les fonctions `op` et `nops` ne peuvent pas être utilisées avec des séquences.

5.5.6 Listes et ensembles

On forme une liste (resp. un ensemble) en mettant une séquence entre crochets (resp. accolades). Dans un ensemble, l'ordre ne compte pas (contrairement au cas des listes), et MAPLE supprime automatiquement les éléments en double et peut éventuellement les réorganiser. Comparez ces exemples :

```
> s := 2 , 1 , 2 , 5 , 1 ; # séquence
> l := [2 , 1 , 2 , 5 , 1] ; # liste
> ens := {2 , 1 , 2 , 5 , 1} ; # ensemble
```

Alors que `op` et `nops` ne fonctionnent pas avec les séquences, ils fonctionnent avec les listes et ensembles.

On peut aussi utiliser des commandes `member`, `union`, `intersect`, `minus` avec les ensembles. `member(x, A)` donne `true` si $x \in A$ et `false` sinon, `A union B` donne l'ensemble $A \cup B$, etc...

Pour les listes, signalons les commandes d'affectation, d'extraction :

- `L[i]` : élément d'indice `i` de la liste `L`.
- `L[i]:=expr` : affectation de la valeur `expr` à l'élément d'indice `i` de la liste `L`.
- `[op(a..b, L)]` : extrait la sous-liste correspondant aux éléments d'indices `a` à `b` de la liste `L`.

5.5.7 Tables, tableaux et matrices

Une matrice est un tableau particulier, et un tableau est lui-même une table particulière ; on définit donc ces objets dans l'ordre inverse.

Tables. Une table associe à un indice permettant de se repérer dans la table une expression. Cet indice n'est pas nécessairement un nombre entier, ni un n -uplet de tels nombres. On peut par exemple définir une table `T` après avoir libéré `T` qui aux oui et non français associent leurs équivalents en anglais. Les deux syntaxes proposées ci-après sont équivalentes :

```
> T := 'T' : # libère T
> T[oui] := yes : T[non] := no ;
ou :
> T := 'T' : # libère T
> T:=table([oui=yes , non=no]) ;
```

Contrairement à la plupart des objets MAPLE, l'appel à une table ne l'évalue pas complètement, il faut demander explicitement l'évaluation grâce à `eval` (cf. chapitre sur évaluation) :

```
> T ; eval(T) ;
```

Pour supprimer l'entrée d'indice `non`, on utilise la commande `unassign` de syntaxe :

```
unassign('T[non]')
```

Si `T1` est une table, une commande type `T2:=T1` ou `T2:=eval(T1)` fait simplement pointer `T2` sur `T1`, ce qui fait que toute modification ultérieure de `T1` sera aussi

répertoriée sur T2. Pour créer une table indépendante dont les termes sont ceux de T1 (mais qui ne seront pas modifiés ultérieurement par les modifications effectuées sur T1), il faut utiliser la commande `copy` en écrivant `T2:=copy(T1)`.

Reprenons par exemple le cadre précédent avec la table T donnant un (mini !) dictionnaire français-anglais. On va ajouter l'allemand. Essayez les commandes suivantes et commentez :

```
> S:=T : T[oui]:=ja : eval(S) ; eval(T)
> R:=copy(T) : T[non]:=nein : eval(R) ; eval(T)
```

A titre informatif, signalons qu'outre les tableaux, les tables sont utilisées essentiellement dans MAPLE comme tables de remember, qui servent en programmation ou pour donner des valeurs ou identités remarquables pour des fonctions ; nous n'étudierons pas cet aspect dans ce cours.

Tableaux. Un tableau (type `array`) est une table dont les indices parcourent un sous-ensemble fini de \mathbb{Z}^p avec $p \geq 1$ et dont les indices croissent de 1 en 1.

La commande `T:=array(m..n , p..q , r..s)` crée par exemple un tableau T dont les indices sont des triplets parcourant l'ensemble $\{m, \dots, n\} \times \{p, \dots, q\} \times \{r, \dots, s\}$. Tant qu'une valeur n'est pas affectée à l'élément d'indice (i, j, k) (ce que l'on fait par `T[i,j,k]:=...`), MAPLE renvoie la variable $T_{i,j,k}$. Il est possible d'affecter plusieurs valeurs simultanément via une liste. Voici par exemple la syntaxe pour un tableau à deux indices :

```
> L:=[(i1,j1)=expr11 , (i2,j1)=expr21 , ... , (ik,j1)=exprk1] ;
> A:=array(m..n , p..q , L) ; # avec m<=ir<=n et p<=jr<=q
```

Exemple

```
> Liste:=[(3,2)=coucou,(4,3)=sqrt(2)] :
> A:=array(3..4 , 1..3,Liste);
```

Vecteurs et matrices. Un vecteur (resp. une matrice) est un tableau à un (resp. deux) indice(s) qui commence(nt) à 1.

On peut créer un vecteur par `array` ou par la commande `vector`. La seconde prend en argument soit une liste, soit la dimension et une fonction. Les deux syntaxes suivantes donnent le même résultat :

```
> v:=vector( [1,4,9,16,25] ) ;
```

ou

```
> v:=vector(5, i->i^2 ) ;
```

Similairement, une matrice peut se créer par `array` ou `matrix`. La seconde commande admet soit la liste des lignes (chacune étant une liste), soit les dimensions suivi de la liste des éléments ligne par ligne, soit les dimensions suivies d'une

fonction. Les trois syntaxes suivantes créent la matrice :

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix} :$$

```
> A:=matrix( [ [1,2] , [3,4] , [5,6] ] ) ;
> A:=matrix(3,2, [1,2,3,4,5,6] ) ;
> A:=matrix(3,2, (i,j)->2*i+j-2 ) ;
```

5.5.8 Polynômes, développements limités et asymptotiques

Un polynôme n'est pas un type de base de MAPLE ; son type de base dépend de comment il a été stocké. On peut cependant tester à l'aide de `type` le type `polynom` qui peut admettre des extensions entre parenthèses du type `integer`, `realcons`, ... dont la signification est analogue au cas du type `complex`.

Un développement limité est une expression de la forme :

```
> a1*(x-a)^b1+...+an*(x-a)^bn+O((x-a)^(n+1))
```

avec $b_1 < \dots < b_p < b(p+1)$. Il lui correspond un type de base MAPLE (`series`), et si p est le nombre de b_j non nuls, `nops` renvoie $(2p+2)$ avec dans l'ordre pour `op` : chaque coefficient non nul suivi de son exposant associé, puis $O(1)$, enfin $n+1$. Testez :

```
> S:=series(sin(x) , x=Pi/2 , 6) ;
> nops(S) ; op(1,S) ; op(2,S) ; op(3,S) ; # et d'autres ...
> op(0,S) ; # donne x-a avec ici a=Pi/2
```

On peut convertir un DL en polynôme (c'est-à-dire se débarrasser du O) par `convert` avec l'option `polynom`) :

```
> convert(S,polynom) ;
```

Ce qui vient de se dire pour les DL s'étend aux autres types de développement sans changement, même si cela peut parfois paraître (mathématiquement) curieux. Par exemple, la fonction pour supprimer le O est toujours `convert(S,polynom)`, même si dans l'expression de S , la partie régulière n'est plus polynomiale...

5.6 Premières règles d'évaluation

Nous présentons ici les premières règles, et nous parlerons plus loin de quelques spécificités. En général une évaluation d'une expression, à moins qu'elle ne se fasse par simple appel de cette expression, se fait à l'aide d'une fonction `eval`, éventuellement à laquelle est ajoutée une lettre désignant le type d'évaluation souhaitée (`evalf` pour une évaluation en flottant, `evalc` pour les complexes, `evalm` pour une évaluation de matrice, `evalb` pour une évaluation de booléen, etc...)

Sur la quasi-totalité des objets MAPLE, une évaluation se fait entièrement. Dans l'exemple ci-dessous, l'évaluation de $y-1$ fait immédiatement remplacer y par x^2 et x par 2, ce qui explique la valeur obtenue :

```
> y:=x^2 : x:=2 : y-1 ;
```

(ce qui est équivalent à `> x:=2 : y:=x^2 : eval(y-1) ;`). Une évaluation au premier niveau consisterait simplement à remplacer y par x^2 , puis celle au second niveau remplacerait de plus x par 2. On peut limiter une évaluation à un certain niveau en passant ce niveau comme paramètre. Essayez :

```
> y:=x^2 : x:=2 : eval(y-1,1) ;
```

et

```
> restart : x:=2 : y:='x^2' ; eval(y-1,1) ;
```

A cette règle générale font exception les tables (et tableaux et matrices...) ou procédures, où l'évaluation se fait au dernier nom rencontré. Une évaluation complète d'une table A se fait par `map(eval,A)` (ou par `evalm` dans le cas des matrices, cf. plus bas). Essayez :

```
> restart : T:=[x , x^2] ;
```

```
> x:=2 : T ; eval(T) ; map(eval , T) ;
```

Quand on veut retarder l'évaluation d'une sous-expression, on la met entre accolades. MAPLE se limite alors à enlever le premier niveau d'accolades. Essayez :

```
> restart : y:=2 : x:=' 'y+3' ' ;
```

```
> eval(%) ;
```

```
> eval(%) ;
```

Remplacez cependant la première ligne par :

```
> restart : x:=' '2+3' ' ;
```

et recommencez. Commentaire. Un retard d'évaluation peut être utile pour afficher un résultat. Par exemple, si après des opérations on trouve $x=2$, la commande `'x'=x` ; affichera $x=2$.

Maintenant, vous êtes en mesure de comprendre pourquoi chacune des deux lignes qui suivent ne provoque pas de message d'erreur et fournit le résultat attendu :

```
> i:=2 : sum('i'^2, 'i'=1..10) ;
```

```
> i:=2 : sum('i^2', 'i'=1..10) ;
```

Enfin, signalons qu'en général MAPLE ne met pas de lui-même les nombres complexes sous la forme $a+I*b$ (sauf dans les cas rationnels). On peut forcer cette écriture d'un nombre complexe z par `evalc(z)`.

5.7 Utilisation des bibliothèques (ou packages)

MAPLE dispose d'un certain nombre de packages contenant des fonctions plus spécialisées qui ne sont pas chargées au lancement du logiciel mais auxquelles

l'utilisateur peut avoir accès. Il y a des packages de géométrie, de théorie des nombres,... Nous utiliserons dans ce cours les quatre packages suivants :

- `linalg` contient la quasi-totalité des fonctions concernant l'algèbre linéaire.
- `DEtools` contient des outils pour les équations différentielles.
- `plots` contient des outils de tracé.
- `student` contient plusieurs outils pour faire des calculs pas à pas.

Suivant les besoins, on dispose de plusieurs manières de charger les fonctions dont on a besoin. Discutons-les par ordre croissant d'encombrement mémoire.

1. **On a besoin à l'occasion d'une fonction.** Par exemple, si l'on veut appliquer la fonction `rowdim` du package `linalg` à une matrice `A`, on peut l'utiliser sans la charger en mémoire en tapant : `linalg[rowdim](A)` ;

2. **On a besoin régulièrement de quelques fonctions d'un package.** On peut charger uniquement ces fonctions. Par exemple, pour charger `rowdim` et `coldim` du package `linalg`, on peut écrire `with(linalg,rowdim,coldim)` ;

3. **On a besoin régulièrement de toutes les (ou la plupart des) fonctions d'un package.** On charge alors le package. Pour `linalg`, on écrira `with(linalg)` ;

5.8 Programmation

Avertissement : on présente ici le syntaxe pour la version 5 de MAPLE (qui est la version disponible sur le serveur *bombadil* et que l'on utilise) ; cette syntaxe à été modifiée dans les versions ultérieures de MAPLE.

5.8.1 Instructions de contrôle

On retrouve les instructions `while`, `for`, `if` déjà vues dans MATLAB. La seule nouveauté est d'apprendre leur syntaxe dans MAPLE. Voici les syntaxes (en italique apparaissent les noms non réservés) :

```
for compt from deb to fin by step do instructions od:
if cond1 then instr1 elif cond2 then instr2 ... else intrp fi:
while booléen do instructions od:
```

Dans une boucle `for`, `from` 1 peut être sous-entendu, de même que `by` 1.

Dans chacun des cas, si `od` ou `fi` est suivi de deux points, MAPLE affiche aucun résultat intermédiaire de la procédure ou au contraire les affichent tous si ces

identificateurs sont suivis d'un point-virgule. On peut vouloir n'en afficher que quelques-uns. Dans ce cas, notre identificateur sera suivi de deux points et on affichera les résultats souhaités par `print`.

L'ensemble des termes du début à la fin d'une telle séquence doit faire partie du même groupe d'exécution. En cas de besoin, on passe à la ligne en appuyant simultanément sur `SCHIFT` et `ENTREE`.

Il existe une variable d'environnement, `printlevel`, fixée par défaut à 1, qui indique le niveau d'imbrication des boucles que l'on imprime. A l'entrée dans une procédure ou dans une fonction, on dispose d'un compteur valant 0, qui augmente de 1 à chaque rencontre d'un `if`, `for`, `while` et qui diminue de 1 à chaque rencontre d'un `od`, `fi`. Le résultat d'une boucle n'est affiché que si notre compteur à une valeur inférieure ou égale à `printlevel` après la prise en compte du mot d'entrée dans la boucle. Testez par exemple :

```
printlevel:=1: z:=0: for i to 3 do for j to 2 do z:=z+j od; z; od;
puis :
printlevel:=2: z:=0: for i to 3 do for j to 2 do z:=z+j od; z; od;
```

5.8.2 Fonctions et procédures

Une fonction associe à une ou plusieurs variables une expression. Cette expression peut être résultat d'un test. Typiquement, on peut définir une fonction par morceaux (quoique MAPLE dispose alors d'une fonction prédéfinie) :

```
f:= x-> if (x<>0) then sin(x)/x else 1 fi;
```

Une procédure est utilisée en général pour faire plusieurs actions. Elle peut aussi accessoirement retourner un résultat (comme une fonction), mais dans ce cas **elle retourne toujours la dernière quantité évaluée**. Une procédure utilise plusieurs types de variables, les variables en entrée, les variables locales et les variables globales. Les variables d'entrée sont celles que l'on passe en argument de la procédure. Ce sont donc des variables muettes dans lesquelles sont stockées les arguments d'entrée. Les variables locales et globales sont introduites à l'intérieur de la procédure. La différence est qu'une variable locale est libérée à la sortie de la procédure tandis d'une variable globale est conservée. Une procédure est déclarée comme suit :

```
proc(var. d'entrée)
local var. locales ;
global var. globales ;
Instructions
end
```

Reprenons le classique exemple de calcul de la factorielle. Un premier programme en variables locales est :

```
lfact:=proc(n) local i,y; y:=1; for i to n do y:=y*i od: end:
et en variables globales :
gfact:=proc(n) global i,y; y:=1; for i to n do y:=y*i od: end: Com-
parez :
i:='i' ; lfact(6); i ;
i:='i' ; gfact(6); i ;
lfact(7);
Commentaires ?
```

Par défaut, MAPLE considère les variables non déclarées comme locales et l'indique à l'utilisateur à l'aide d'un *Warning*.

Exercice 5.8.1 *Ecrire une fonction prenant en entrée un entier n au moins égal à 1, et donnant en sortie une matrice carrée d'ordre n dont les termes diagonaux sont des a et les termes non diagonaux sont des b . Ecrivez le programme le plus court possible.*

Chapitre 6

Utilisation de MAPLE en mathématiques

Ce chapitre présente quelques-uns des outils mathématiques de MAPLE. L'ensemble de ce chapitre est au programme de l'examen, sauf la section sur les équations et systèmes différentiels et la partie réduction en algèbre linéaire. Vous devez savoir vous débrouiller avec le reste. Je ne vous conseille pas cependant une lecture approfondie de la première section du reste du chapitre. Notamment, il n'est pas utile de connaître toute la première section, ni de connaître l'ensemble des options sur les graphiques : ces sections servent surtout à s'y référer en cas de besoin, il est juste utile de connaître quelques principes de base sur ces sections.

6.1 Développement, factorisation et simplification d'expressions

On dispose ici de plusieurs fonctions dont l'utilisation est particulièrement subtile... Les éléments indiqués ci-dessous ne sont que des bribes, il est fortement conseillé de consulter l'aide sur chacune des fonctions. L'une des difficultés tient notamment au caractère multiforme de la fonction argument dans le champ complexe (et des fonctions déduites de celle-ci : logarithme, puissance...) Nous faisons donc quelques rappels introductifs dans la première sous-section. Signalons enfin que par expression polynômiale (dans MAPLE), on entend une fonction qui est un polynôme en des fonctions pas nécessairement polynômiales. Par exemple, $\sin^2(x) + \tan^3(x)$ est $P(\sin(x), \tan(x))$ avec $P(X, Y) = X^2 + Y^3$, c'est donc une expression polynômiale.

6.1.1 Rappels mathématiques

Tout nombre complexe (non nul) z se décompose en module $|z|$ et argument $Arg(z)$. L'argument est une fonction multiforme, définie modulo 2π , et MAPLE choisit la représentation de sorte que $Arg(z) \in]-\pi, \pi]$, ce qui fixe l'argument de

manière unique. D'autre part, signalons que pour MAPLE, $Arg(0) = 0$!
Le logarithme d'un nombre complexe z non nul est défini (modulo $2\pi i$) par :

$$\ln(z) = \ln(|z|) + iArg(z).$$

MAPLE choisit donc une représentation particulière. Avec celle-ci, nous avons :

$$\ln(z_1..z_n) - \left(\sum_{i=1}^n \ln(z_i) \right) \in 2\pi i\mathbb{Z}$$

et cette différence ne vaut 0 que lorsque $\sum_{i=1}^n Arg(z_i) \in]-\pi, \pi]$. En particulier, $\ln(z^2)$ n'est pas toujours égal à $2\ln(z)$.

La fonction exponentielle est cependant définie sans ambiguïté :

$$\exp(z) = \exp(Re(z)) (\cos(Im(z)) + i \sin(Im(z))).$$

Les fonctions puissances sont définies par :

$$z_1^{z_2} = \exp(z_2 \ln(z_1)).$$

En particulier, elles dépendent du logarithme, et sont définies avec ambiguïté. En fait, il n'y a pas de problème si par exemple $(z_1, z_2) \in \mathbb{R}^+ \times \mathbb{R}$ ou $(z_1, z_2) \in \mathbb{R} \times \mathbb{Z}$. Sinon, les problèmes peuvent surgir... Par exemple, $(a^b)^c$ n'est pas toujours a^{bc} . De plus, lorsque p est un entier impair et $x \in \mathbb{R}^-$, il est usuel de poser $x^{1/p} = -(-x)^{1/p}$, de sorte que par exemple la racine cubique de -8 soit réelle et égale à -2. Ce n'est pas le cas avec MAPLE (essayez...) On dispose toutefois de la fonction `surd` définie sur le corps des complexes, mais qui remplit ce but dans le cas réel.

6.1.2 Quelques fonctions de manipulation d'expression

`expand` développe les expressions polynômiales, exprime les lignes trigonométriques de nx en fonction de celle de x , développe l'exponentielle d'une somme ou le logarithme d'un produit.

`collect` regroupe les termes d'expressions polynômiales ; cette fonction fait un appel préalable à `expand`, qu'il est donc inutile d'appeler.

`factor` factorise une expression polynômiale ou un quotient de deux telles expressions (expressions rationnelles) dans le corps engendré par les coefficients.

`normal` simplifie les expressions rationnelles en divisant par le PGCD du numérateur et du dénominateur.

`radnormal` et `rationalize` sont des fonctions de manipulations de racines (voir aide).

`convert`, `simplify` : voir ci-dessous.

6.1.3 La fonction `convert`

La fonction `convert` est d'une grande richesse et mérite à elle seule un paragraphe ; je ne saurais trop vous conseiller d'aller consulter l'aide pour de plus amples informations.

Sa syntaxe est `convert(expr, option)`. Il y a plusieurs options possibles, signalons-en quelques-unes :

- `exp` exprime les fonctions trigonométriques à l'aide d'exponentielles.
- `ln` exprime les fonctions trigonométriques réciproques à l'aide de logarithmes.
- `sincos` remplace `tan` et `tanh` en fonctions de sinus et cosinus.
- `tan` exprime $\sin(\theta)$ et $\cos(\theta)$ en fonction de $\tan(\theta/2)$.
- `trig` transforme les exponentielles en combinaison de fonctions trigonométriques (circulaires ou hyperboliques).

6.1.4 La fonction `simplify`

Cette fonction est encore plus riche, mais à manier avec précautions.

Une première syntaxe est `simplify(expr, opt, symbolic)` où `expr` est l'expression à simplifier, et `opt` est une option parmi `trig`, `sqrt`, `radical`, `power`. Sauf dans le cas de `trig`, les simplifications effectuées sont purement symboliques et ne sont pas valables pour toutes valeurs complexes ! Il faut donc utiliser cette fonction avec prudence ! Le paramètre `opt` est facultatif, s'il n'est pas indiqué MAPLE fera toutes les simplifications qu'il peut faire.

`trig` concerne les lignes trigonométriques, `sqrt` les racines carrées, `radical` les puissances rationnelles et `power` les autres puissances et fonctions associées (par exemple logarithme et exponentielle).

Si on enlève l'option `symbolic`, MAPLE n'effectue que les simplifications valables pour toutes les valeurs complexes des variables. On peut toutefois préciser à MAPLE qu'une variable `x` a certaines propriétés, par exemple un domaine de

variation ; ceci se fait par `assume`. La liste des hypothèses formulables et des exemples d'applications aux simplifications sont fournis dans l'aide `?assume` qu'il est fortement conseillé de consulter. Par défaut, $x \in A$ s'écrit à l'aide de `assume` par `assume(x,A)`. Dans certains cas, on peut remplacer le couple (x,A) par une relation satisfaite par x .

Voici quelques exemples d'utilisation qui pourraient vous être utiles :

`assume(x , real)` signifie $x \in \mathbb{R}$.

`assume(x < 1)` signifie $x < 1$.

`assume(x , RealRange(2 , 3))` signifie $x \in [2,3]$.

`assume(x , RealRange(2 , Open(3)))` signifie $x \in [2,3[$.

Par défaut, une variable sur laquelle est formulée une hypothèse est ensuite suivie d'un tilde. On peut retrouver l'hypothèse formulée par `about`. Par exemple, testez les lignes suivantes :

```
> x:='x' : y:='y' : z:='z' : t:=(x^y)^z :
> simplify(t) ;
> assume(x>0) : assume(y,real): assume(z:real) ;
> simplify(t) ;
> about(y) ;
```

Enfin, pour simplifier une expression, on peut proposer à MAPLE une ou plusieurs règles de simplifications, sous la syntaxe suivante :

```
> simplify(expr,[regle1, regle2,...,reglep]) ;
```

Les règles sont écrites sous forme d'égalité, où l'on met dans le membre de droite ce que l'on veut voir figurer au final. Une application pratique est le problème de l'expression d'une fonction rationnelle symétrique des racines d'un polynôme : si par exemple on appelle z_1, z_2, z_3 les racines du polynôme $X^3 + aX^2 + bX + c$ et que l'on veut exprimer :

$$\frac{z_1}{z_2 + z_3} + \frac{z_2}{z_3 + z_1} + \frac{z_3}{z_1 + z_2}$$

en fonction de a, b, c , on fera :

```
> simplify(z1/(z2+z3)+z2/(z1+z3)+z3/(z1+z2), [z1+z2+z3=-a, z1*z2+z2*z3+z3*z1=b, z1*z2*z3=-c]) ;
```

6.1.5 Quelques exercices de manipulation d'expressions

Exercice 6.1.1 *Dans cet exercice, les questions sont indépendantes, et il est conseillé de faire un `restart` entre chaque question.*

1. Essayez de simplifier (sans l'option `symbolic`) les expressions $\tan(\arctan(u))$

puis $\arctan(\tan(u))$. Expliquez les différences. Reprenez la seconde expression après avoir formulé l'hypothèse $u \in]-\pi/2; \pi/2[$.

2. Développez $\tan(a + b + c)$ puis $\ln(xy)$. Au besoin, faites des hypothèses.
3. Développez $\sin(5u)$, puis exprimez-le uniquement à l'aide de sinus en imposant une règle de simplification.
4. Utilisez `factor` pour factoriser les expressions $x^2 - 3x + 2$, $x^2 - 3$ puis $\sqrt{3}(x^2 - 3)$. Commentez¹.
5. On définit l'expression rationnelle :

$$Q := \frac{x^3 + x^2 - 9x - 9}{x^2 - 4x + 3}.$$

En repartant à chaque fois de cette expression, effectuez successivement les opérations suivantes : factorisation du numérateur et du dénominateur, factorisation du numérateur en laissant tel quel le dénominateur, puis le contraire, enfin simplification sans factoriser par le PGCD du numérateur et du dénominateur.

6. A l'aide de MAPLE, vérifiez que :

$$\frac{1}{(\cos(t))^2} = 1 + (\tan(t))^2.$$

7. Exprimez $\frac{1}{1+\sqrt{2}+\sqrt{3}}$ sans radicaux au numérateur (on utilisera en premier `rationalize` (cf. l'aide), puis on écrira cette expression sous la forme $a_1 + a_2\sqrt{2} + a_3\sqrt{3} + a_4\sqrt{2}\sqrt{3}$ où les a_i sont tous rationnels).

6.2 Résolution d'équations et de systèmes d'équations

On dispose de plusieurs commandes de résolution d'équations, dont le radical est `solve`. Je signale l'existence de la commande `rsolve` pour les équations de récurrence.

6.2.1 Résolution exacte : la commande `solve`

La commande `solve` s'attache à déterminer, si possible, une solution à l'équation donnée. Elle fournit une solution exacte, à moins que figure un flottant dans l'équation, auquel cas elle cherche une solution en flottant. Sa syntaxe pour une

¹Il peut être utile, en pratique pour factoriser, d'utiliser la commande `solve` qui sera vue au chapitre 2.

équation à une inconnue est :

```
solve(p,x)
```

où p est une expression dépendant de la variable libre x . Cette commande retourne parfois un objet "bizarre" nommé `RootOf`, mais qui est en fait utilisable par MAPLE (si l'équation dépend de paramètres, on peut obtenir un D.L. par rapport aux paramètres, et elle sert en théorie des corps pour travailler dans les extensions algébriques, on ne s'attardera pas sur ces points...). Illustrons toutefois l'illustration de l'utilisation d'un `RootOf`. Considérons l'équation :

$$x^5 + ax + 1 = 0$$

qui, lorsque $a = 0$ a une unique solution réelle, -1 . On veut savoir ce que devient cette racine lorsque a varie en restant près de 0 . Appelons z la racine :

```
z:=solve(x^5+a*x+1=0,x)
```

On effectue un développement limité de la solution, dont on ne garde que la partie polynômiale :

```
appz:=convert(series(z,a=0,6),polynom)
```

`appz` est donc une solution approchée de l'équation, valable lorsque a est proche de 0 . Substituons `appz` dans l'équation :

```
u:=expand(appz^5+a*appz+1);
```

Certes, ce n'est pas très joli... Ceci devrait faire 0 exactement si on n'avait pas tronqué le développement en série. Regardons si a vaut 0.1 :

```
subs(a=0.1,u);
```

`solve` sait résoudre toutes les équations polynômiales de degré inférieur ou égal à 4 , pour lesquelles elle donne toutes les solutions. Cependant, dans le cas des équations du 4ème degré, elle se contente souvent d'une solution en `RootOf`, si les radicaux ont des expressions trop longues ; dans ce cas, on peut toutefois forcer l'écriture en posant `_EnvExplicit:=true` (cette variable est par défaut fixée à `false`).

Pour les autres équations, MAPLE ne sait pas en général déterminer une solution analytique, *a fortiori* toutes les solutions. Il peut parfois se contenter de donner une solution (on peut lui demander toutes, s'il sait le faire en posant `_EnvAllSolutions:=true`), donner une solution à l'aide d'un `RootOf`, ou encore retourner une liste vide (ce dernier cas ne signifie pas pour autant qu'il n'y a pas de solutions).

`solve` peut aussi résoudre des systèmes d'équations simples (par exemple linéaires). Il arrive, par exemple dans le cas d'un système linéaire ayant plusieurs solutions, que MAPLE paramètre l'ensemble des solutions par une inconnue. La syntaxe est ici : `solve({eqns},{inc})`

Ce qu'il faut retenir, c'est que sauf dans les cas simples où l'on sait dénombrer le nombre de solutions (par exemple les cas polynômiaux), il faut avoir une confiance

relativement modérée sur le nombre de solutions retournées par cette commande. On se tourne en fait assez rapidement vers une évaluation en flottant (en mettant un flottant dans l'expression de p ou en invoquant `fsolve`). Même dans ces cas, il faut avoir une confiance relativement modérée en le nombre de solutions obtenues...

6.2.2 Résolution approchée : la commande `fsolve`

`fsolve(p,x)` retourne par défaut une solution approchée réelle de l'équation, et toutes les solutions dans le cas d'équations polynômiales. A nouveau, la variable x doit être libre. On peut forcer le système à chercher aussi les solutions complexes par en ajoutant comme option en troisième argument le mot `complex`, ou à chercher les solutions réelles comprises entre a et b par `fsolve(p,x=a..b)`. Cette commande s'emploie souvent avec un graphique de la courbe pour localiser *ex-ante* les (des) solutions.

Cette commande peut être utilisée pour un système, où l'on peut aussi préciser un domaine de recherche (`fsolve(p, {x=a..b,y=c..d})`).

6.2.3 Exercice avec ces commandes

Exercice 6.2.1 *Les questions sont indépendantes.*

1. Résolvez $x^4 + 2x^3 + 3x^2 + 2x + 1 = 0$.
2. Résolvez $x^4 + 2x^3 + 3x^2 + 4x + 5 = 0$. Forcez l'écriture de la solution. Commentaires ? Donnez des solutions approchées.
3. Résolvez le système linéaire :

$$\begin{cases} ax + y + z = 1 \\ x + ay + z = a \\ x + y + az = a^2 \end{cases}$$

d'inconnues x, y, z et où a est un paramètre. Au vu des solutions, il n'y aurait un problème uniquement lorsque $a = -2$. Écrivez le système obtenu lorsque $a = 1$ et résolvez-le. Commentaires ?

6.3 Limites, dérivées et développements

6.3.1 Limites

Les limites peuvent être calculées en un réel a , à gauche de a , à droite de a , en $+\infty$, en $-\infty$. Les commandes respectives pour effectuer ceci sur une expression p dépendant d'une variable libre x sont :

`limit(p,x=a)`, `limit(p,x=a,left)`, `limit(p,x=a,right)`,
`limit(p,x=infinity)`, `limit(p,x=-infinity)`

Pour une fonction f , la syntaxe est :

`limit(f(x), x=a)` (pour le premier, les autres étant analogues, et l'on peut enfin calculer des limites doubles (si elles existent) en mettant les valeurs des points sous forme d'ensemble :

`limit(f(x,y), { x=a, y=b })` ;

Lorsque la limite n'existe pas, la commande retourne `undefined`. Lorsqu'elle retourne un intervalle, cela signifie que toutes les valeurs de l'intervalle sont prises au voisinage du point.

Exemples :

`f:=x->x/abs(x): limit(f(x), x=0, right); limit(f(x), x=0, left);`

`limit(x*sin(1/x), x=0); limit(sin(1/x), x=0); limit(1/x*sin(1/x), x=0)`

(Vous remarquerez que dans le dernier résultat on attendait plutôt la réponse $-\infty, \dots, \infty$).

`limit((x^2-y^2)/(x^2+y^2), x=0, y=0);`

ce qui est juste ; par contre MAPLE est dépassé par cette limite :

`limit((x^3+y^4)/(x^2+y^2), x=0, y=0);`

alors que des manipulations élémentaires montrent que :

$$\left| \frac{x^3 + y^4}{x^2 + y^2} \right| \leq \sqrt{x^2 + y^2} + (x^2 + y^2)$$

et donc la limite est 0.

6.3.2 Dérivées

Si p est une expression dépendant de variables libres x, y, \dots , on utilise la commande `diff` pour obtenir les expressions dérivées. Les exemples suivants permettent de comprendre toutes les variations :

- $\frac{\partial p}{\partial x}(x, y)$ s'obtient par `diff(p, x)`.
- $\frac{\partial^3 p}{\partial x^3}(x, y)$ s'obtient par `diff(p, x, x, x)` (que l'on peut raccourcir en `diff(p, x$3)`).
- $\frac{\partial^2 p}{\partial y \partial x}(x, y)$ s'obtient par `diff(p, x, y)` qui est identique à `diff(diff(p, x), y)` (attention, dans certains cas, l'ordre compte).

Si maintenant f est une fonction, on souhaite récupérer une fonction pour la dérivée. La fonction dérivée se note $D(f)$. La dérivée partielle par rapport à la première variable est $D[1](f)$ (on peut faire de nombreuses variations, voici l'équation de Laplace à 3 variables : $D[1, 1](f) + D[2, 2](f) + D[3, 3](f) = 0$). Enfin, la dérivée n -ème s'obtient par $(D@@n)(f)$.

6.3.3 Développements limités et asymptotiques

Nous avons déjà parlé de la fonction `series` qui permet d'obtenir ces développements. Signalons simplement que lorsqu'il n'existe pas de développement au voisinage d'un point, on peut soit obtenir un message d'erreur, soit un développement seulement valable d'un côté sans message : la prudence est donc toujours de mise lorsque MAPLE renvoie quelque chose et que l'on n'est pas dans les cas usuels...

Lorsqu'une expression `p` dépend d'une variable libre `x`, on peut demander à juste obtenir un équivalent, en précisant éventuellement l'ordre maximal de recherche (argument facultatif `n`) ; la syntaxe est alors :

```
series(leadterm(p), x=a, n) ;
```

6.3.4 Petit exercice

L'exercice suivant n'a que peu d'intérêt pratique, il est juste présent pour vous faire manipuler les commandes.

Exercice 6.3.1 *On considère la fonction de deux variables t, x et dépendant de deux paramètres a et b suivante :*

$$f(t, x) := e^{-at} \cos(bx).$$

1. *Trouver une relation entre a et b de sorte que f soit solution de l'équation de la chaleur :*

$$-\frac{\partial f}{\partial t} + \frac{\partial^2 f}{\partial x^2} = 0.$$

Désormais, on remplace a par la fonction de b permettant cette relation, et l'on note fb la fonction obtenue $fb(t, x) := e^{-a(b)t} \cos(bx)$ qui ne dépend donc plus que du paramètre b .

2. *Faites un développement limité d'ordre 5 en fonction du paramètre b au voisinage de 0, à (t, x) fixé, de $fb(t, x)$.*
3. *Ecrivez en utilisant `op` la relation $t = \phi(x)$ permettant d'annuler le terme en b^2 . Remplacez t par $\phi(x)$ dans l'expression du développement. Quel est le terme d'ordre 4 ?*

6.4 Sommes, produits et intégrales

6.4.1 Sommes et produits sur des ensembles numériques

Lorsque l'on veut calculer une somme $\sum_{i \in L} f(i)$ avec un ensemble L composé uniquement de valeurs numériques, on dispose de deux syntaxes.

La première est basée sur `add`. On écrit :

```
> add(f(i) , i=L) ;
```

avec L liste, ou $i=m..n$ si i varie de m à n .

La seconde syntaxe consiste à construire tout d'abord une liste avec les valeurs que l'on souhaite sommer, puis à utiliser `convert` :

```
> S:= [ seq( f(i) , i=L) ] ;
> convert(S, '+' ) ;
```

En ce qui s'agit de produits, on dispose de commandes analogues à condition de remplacer dans le premier cas `add` par `mul` et dans le second cas l'option '+' par '*'.

Exemples :

- > `add(i^2 , i=0..10) ;`
- > `add(a[i]*x^i , i=0..5) ;`
- > `add(a[i] , i=[0,2,6]) ;`
- > `mul(i , i=1..7) ; # calcul de factorielle 7`

6.4.2 Sommes

MAPLE permet aussi de traiter certaines sommes particulières de manière indéfinie. Si f est une fonction (resp. si p est une expression de la variable libre k), la commande `sum(p,k)` ; (resp. `sum(f(k),k)`) retourne une expression g (resp. une fonction) telle que $g(k+1)-g(k)=p$ (ou $f(k)$). On peut spécifier des bornes de sommation en écrivant $k=a..b$ comme second argument au lieu de k . Ces indices ne sont plus nécessairement numériques, on peut avoir des lettres muettes ou l'infini. Par exemple :

```
> sum(1/n^6,n=1..infinity) ;
```

On dispose aussi de la commande inerte `Sum` dont l'effet est d'écrire la somme avec le symbole de sommation sans l'évaluer. On peut ensuite l'évaluer par `value`. Un intérêt de la forme inerte est par exemple de pouvoir écrire les résultats ou de faire des opérations avant l'évaluation. Testez :

```
> S:=Sum(i,i=1..n) ;
> S:=value(S) ;
```

MAPLE reconnaît les sommes polynômiales, certaines sommes de fractions rationnelles, et certaines trigonométriques. Il lui arrive d'exprimer les solutions à l'aide de fonctions que vous ne connaissez pas nécessairement ; dans ces cas, l'aide

est fort utile. Une que vous pourriez voir rapidement apparaître est la fonction Ψ qui sur les entiers vaut :

$$\Psi(n) = -\gamma + \sum_{i=1}^{n-1} \frac{1}{i}$$

où γ est la constante d'Euler 0.57721... Cette constante est la limite de la suite $(\sum_{k=1}^n 1/k - \ln(n))_n$.

Enfin, dans le cas où MAPLE ne sait pas quoi faire, il renvoie une forme non évaluée. On peut alors utiliser `evalf` (dans un cas numérique) pour une évaluation en flottant.

En ce qui concerne les produits, on dispose de `product` dont l'utilisation est tout à fait analogue à `sum` (même syntaxe, forme inerte avec une majuscule, etc...)

Exercice 6.4.1 On introduit la fonction ζ définie pour $\alpha > 1$ par :

$$\zeta(\alpha) = \sum_{n=1}^{+\infty} \frac{1}{n^\alpha}.$$

On sait que si p est un entier pair, le rapport $\frac{\zeta(p)}{\pi^p}$ est rationnel.

1. En utilisant `add`, et en remplaçant l'infini comme borne par un nombre fini, prévoir les valeurs de ces rationnels lorsque $p \in \{2, 4, 6, 8, 10\}$ (indication : ils sont tous de la forme $1/N$ avec N entier).
2. Vérifier en utilisant `sum`.

6.4.3 Intégration

Comme pour les sommes, on peut calculer des primitives et des intégrales par la commande `int` qui admet une forme inerte `Int`. Ce qui a été dit ci-dessus pour les sommes est également valable pour les intégrales.

La nouveauté est la possibilité d'utiliser dans le package `student` les commandes `intparts` et `changevar` qui permettent de faire respectivement une intégration par parties et un changement de variables. Rappelons que l'on commence par charger le package `student` en tapant `with(student)`. Pour intégrer par parties une intégrale $J = \int uv'$, on écrit :

`intparts(J,u)`

et pour poser $f(u)=g(x)$ dans $J = \int \phi(x)dx$, on écrit :

`changevar(g(x)=f(u) , J , u)`

Voici un exemple utilisant `intparts`. On veut calculer $\int x \ln(x)dx$ en dérivant le logarithme et en intégrant x .

```

> with(student,intparts) ; % charge intparts
> J:=Int(x*ln(x) , x) ;
> intparts(J, ln(x)) ;
> value(%) ;

```

Exercice 6.4.2 *Le but de cet exercice est de calculer l'intégrale :*

$$I_1 := \int_0^{\pi/2} \frac{\sqrt{\cos(t)}}{\sqrt{\sin(t)} + \sqrt{\cos(t)}} dt.$$

Pour cela, on introduit :

$$I_2 := \int_0^{\pi/2} \frac{\sqrt{\sin(t)}}{\sqrt{\sin(t)} + \sqrt{\cos(t)}} dt.$$

Vous devrez définir ces intégrales à l'aide de la forme inerte Int pour effectuer les opérations souhaitées.

1. Calculez $I_1 + I_2$ (pour mettre la somme de deux intégrales sous forme d'une seule intégrale, on utilise la commande `combine`).
2. Poser $t = \pi/2 - u$ dans I_1 . Que constatez-vous ?
3. En déduire la valeur de I_1 .

6.5 Graphiques en dimensions 2 et 3

6.5.1 L'environnement graphique

Par défaut, MAPLE trace le graphe sur la feuille de calcul. On peut lui demander de tracer ce graphe dans une fenêtre à part, soit pour un graphique particulier en double-cliquant dessus, soit dans tous les cas en modifiant l'option `Plot Display` qui se trouve dans le menu `Options`. Ceci a l'avantage d'ouvrir une barre d'outils particulière permettant d'accéder à certaines options. Il vous est recommandé, dans les exemples qui vont suivre, de double-cliquer sur le résultat et de tenter certaines modifications à l'aide des boutons de la barre d'outils graphiques.

6.5.2 Généralités sur les graphiques en dimension 2

Pour tracer le graphe d'une expression p dépendant d'une variable libre x sur un intervalle $[a, b]$, on écrit :

```
plot(p , x=a..b) ;
```

On peut mettre aussi un domaine pour les ordonnées. On peut spécifier un nom si l'on veut mettre un label, ou rien sinon. Comparez :

```
> plot(x^2 , x=0..2) ;
```

```
> plot(x^2 , x=0..2 , 0..1 ) ;
> plot(x^2 , x=0..2 , ordonnees=0..1 ) ;
```

Questions : que se passe t-il si l'on oublie $x=$ (en tapant toutefois $a..b$) ? Et si $a>b$?

Dans le cas d'une fonction, la syntaxe est : `plot(f, a..b)` ou `plot(f(x), x=a..b)`. Attention à soit mettre deux fois les x , soit aucune fois !

Dans le cas où l'on veut tracer plusieurs fonctions ou expressions, on les regroupe sous forme d'une liste. Cela est notamment pratique si l'on veut tracer plusieurs fonctions entrant dans une famille dépendant d'un paramètre.

Pour un tracé d'une courbe paramétrique donnée par un couple d'expressions (p, q) dépendant d'une variable libre t , la syntaxe est :

```
plot( [p , q , t=t0..t1] , abs=a..b , ord=c..d);
```

Les arguments relatifs à `abs` et `ord` sont facultatifs ; ils ne servent qu'à proposer une fenêtre et éventuellement à libeller les axes.

On peut bien entendu ne pas libeller les axes (`plot([p,q,t=t0..t1], a..b, c..d)` ;) voire laisser MAPLE créer une fenêtre automatiquement (`plot([p,q,t=t0..t1])` ;).

Pour tracer simultanément plusieurs courbes paramétriques, on fait une liste des différents éléments la composant, on obtient donc une liste de listes. On peut bien entendu tracer une famille dépendant d'un paramètre, ou adapter aux fonctions ce qui a été écrit pour les expressions.

Il peut être utile de tracer une courbe implicite, par exemple une courbe d'indifférence d'un consommateur de fonction d'utilité u est donnée par $u(c, d) = cste$ (dans une économie à deux biens). En général (c'est-à-dire sauf avec des u particulières), on ne peut pas tirer explicitement d en fonction de c le long d'une courbe d'indifférence, et le tracé des courbes passe par un tracé implicite. Il faut charger la fonction `implicitplot` pour pouvoir l'utiliser. On tape ainsi :

```
> with( plots , implicitplot ):
```

La syntaxe pour une expression p dépendant des variables libres x et y est `implicitplot(p, x=a..b, y=c..d)`. Je vous laisse le soin de l'adapter aux fonctions. Souvent, les tracés de fonctions implicites sont imprécis, il vaut mieux augmenter la grille ce que l'on fait par `grid=[m,n]` (par défaut $m=n=25$). Mais ceci ralentit largement le tracé de la courbe...

6.5.3 Les options de plot

Les options sont ajoutées après les renseignements déjà décrits. Nous en décrivons quelques-unes, il est conseillé d'aller voir l'aide pour voir l'ensemble des possibilités.

`linestyle=n` règle le type de tracé ($n=1$ pour un tracé continu, $n=2$ pour un trait, $n=3$ pour un point, enfin $n=4$ pour un trait-point).

`thickness` règle l'épaisseur du tracé.

`scaling=CONSTRAINED` force l'utilisation d'un repère normé.

`title=` Mon graphique`` (avec apostrophes inversées) pour donner un titre au graphique.

`color=` suivi d'un nom de couleur (voir la liste dans l'aide ; souvent le nom est le nom anglais).

`axes= BOXED, FRAME, NORMAL` ou `NONE` suivant les axes souhaités (essayez).

`numpoints, resolution` paramètres pour améliorer la qualité du tracé. Si une fonction a peu de singularités, il vaut mieux augmenter `resolution` que `numpoints`.

6.5.4 Tracés de surfaces

Afin de ne pas allonger, nous allons nous limiter aux surfaces d'équations cartésiennes.

Si p est une expression dépendant des variables libres x et y , on trace la surface d'équation p par la commande :

```
plot3d(p,x=a..b,y=c..d);
```

On peut vouloir restreindre le domaine des côtes à $[z1,z2]$. Ceci se fait en écrivant `view=z1..z2` (attention, le mot `view` est obligatoire). Ceci s'adapte bien entendu aux fonctions de manières naturelle, et l'on peut tracer plusieurs surfaces simultanément. On dispose ici de nombreuses options, dont certaines sont communes à `plot` et d'autres ont accessibles sur la barre d'outils graphique. En cas de besoin particulier, il est fortement conseillé d'utiliser l'aide.

Exercice 6.5.1 On considère la fonction de production CES :

$$Q = (\alpha K^a + (1 - \alpha)L^a)^{1/a}$$

($\alpha \in]0; 1[$ et $a > 0$ sont deux paramètres réels).

1. A l'aide de MAPLE, trouvez la fonction de production obtenue lorsque $a \rightarrow 0^+$.
2. Illustrez (par des graphiques) ce que vous avez obtenu lorsque $\alpha = 1/2$.

6.6 Algèbre linéaire

MAPLE présente beaucoup de fonctions d'algèbre linéaire ; nous allons ici étudier les plus courantes. A l'exception des quelques commandes déjà expliquées de création de matrices et de vecteurs, les commandes d'algèbre linéaire sont dans la bibliothèque `linalg` et doivent donc être chargées, soit en chargeant la commande dont vous avez besoin, soit en chargeant toute la bibliothèque `linalg`.

A ce niveau, le lecteur est supposé savoir créer une matrice et avoir assimilé le problème de l'évaluation dans le cas des tableaux. Signalons toutefois la commande `map`, utile pour effectuer des opérations sur chaque élément de la matrice. Par exemple, `map(expand,A)` développe chaque coefficient de A (alors que `expand(A)` est sans effet).

6.6.1 Premières commandes

Dimension. La commande `rowdim(A)` (resp. `coldim(A)`) s'applique à une matrice (et non pas un vecteur) donne respectivement le nombre de lignes et de colonnes de la matrice A . Pour un vecteur v , la commande est `vectdim(v)`.

Extraction. On extrait la sous-matrices contenant les lignes i_1 à i_2 et les colonnes j_1 à j_2 de la matrice A en tapant `submatrix(A,i1..i2,j1..j2)`. On peut extraire uniquement la pème ligne (resp. colonne) par `row(A,p)` (resp. `col(A,p)`).

Matrices aléatoires. La commande `randmatrix(n,p,entries=rand(a..b))` crée une matrice à n lignes et p colonnes dont les coefficients sont des entiers entre a et b . Le troisième argument est facultatif et alors $a=-99$, $b=99$. Pour créer une matrice aléatoire de nombres entre 0 et 1 on peut par exemple effectuer :

```
scalarmul( randmatrix(2,3,entries=rand(0..10000)) , 0.0001)
```

Pour créer des vecteurs aléatoires, la commande est `randvector(n,entries=rand(a..b))`.

Matrices diagonales. Pour créer une matrice diagonale de coefficients sur la diagonale a_1, \dots, a_p , on écrit `diag(a1, ..., ap)`. Pour créer la matrice identité d'ordre 12, on peut écrire `diag(1$12)`. Enfin, contrairement à MATLAB, si x est un nombre et A une matrice, le raccourci $A+x$ désigne pour MAPLE $A+xId$, c'est-à-dire qu'il ne rajoute x que sur la diagonale.

Base d'un e.v. Si L est une liste (resp. un ensemble) de vecteurs, alors `B:=basis(L)` retourne une liste (resp. un ensemble) de vecteurs formant une base du s.e.v. engendré par les vecteurs de L . On accède alors facilement à la dimension de ce sous-espace par la commande `nops(B)`.

Autres. Le rang (resp. le déterminant, resp. l'inverse) de A se calculent respectivement par `rank(A)`, `det(A)`, `inverse(A)`.

6.6.2 Opérations matricielles

Si A et B sont des matrices ou des vecteurs de même dimension, et si a et b sont des scalaires :

- `scalarmul(A,a)` calcule aA .
- `matadd(A,B)` calcule $A+B$.
- `matadd(A,B,a,b)` calcule $aA+bB$.
- `multiply(A,B)` calcule AB .

Les trois premières opérations peuvent être effectuées avec + et/ou * également. Cependant, le résultat ne sera pas complètement évalué (ce qui exige une évaluation par `evalm`) et d'autre part `2*u-u-u` donne 0 et pas le vecteur nul, tandis que l'on obtient le vecteur nul avec ces opérations. Il est cependant vrai que cette seconde syntaxe est moins lourde, et que le second inconvénient ne se fait guère sentir si l'on ne programme pas.

En revanche, il ne faut jamais effectuer un produit de matrices avec *. La commande à utiliser (à part `multiply`) est `&*` conjointement à `evalm`. La multiplication * est formelle et commutative, ce qui fait que pour MAPLE, `inverse(P)*A*P` donne toujours A ! De même, la puissance n-ème d'une matrice A ne note `A&n`.

6.6.3 Réduction des matrices

Voici les principales commandes pour récupérer les éléments propres de la matrice A :

- `charmat(A,x)` calcule `xId-A`.
- `charpoly(A,x)` calcule `det(xId-A)`, c'est-à-dire le polynôme caractéristique de A (éventuellement au $(-1)^n$ près, selon vos conventions).
- `minpoly(A,x)` calcule le polynôme minimal de A.
- `eigenvals(A)` donne les valeurs propres de A, ou fournit une approximation numérique de celles-ci dès que A contient un flottant.
- `eigenvects(A)` donne une séquence formées de listes $[\lambda, n_\lambda, B_\lambda]$ où λ est une valeur propre, n_λ est la dimension du sous espace propre associé et B_λ est une base de ce sous-espace propre.

La matrice est diagonalisable si et seulement si $\sum n_\lambda$ vaut la dimension de la matrice. Dans ce cas, on récupère la matrice P par les instructions (on fait ici pour le cas où il y a deux valeurs de λ :

```
> s:=eigenvects(A) ;
> vp:=op(3,s[1]) union op(3,s[2]) : # l'ens obtenu est une base de
v.p. ;
> P:=concat( op(vp) ) ; # op sert à transformer l'ens en sequence
;
> multiply( inverse(P), A, P ) ; # pour verifier
```

Une application de la diagonalisation est la possibilité d'écrire A^n avec n symbolique, ce que ne sait pas faire MAPLE. Il suffit en effet de récupérer les éléments propres P et D, et d'effectuer $P D^n P^{-1}$, sachant que D^n se calcule à vue. Par exemple, avec la matrice :

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

on récupère immédiatement :

$$P = \begin{pmatrix} 1 & 2 \\ 0 & 1 \end{pmatrix}$$

et :

$$D = \begin{pmatrix} 2 & 0 \\ 0 & 3 \end{pmatrix}$$

On calcule alors A^n par :

```
multiply(P, diag(2^n,3^n), inverse(P) )
```

Enfin, signalons que l'on peut obtenir une réduite de Jordan de A par `jordan(A)`. La commande `jordan(A,P)` calcule une réduite de Jordan et stocke la matrice de passage dans P . Dans le cas diagonalisable, on retrouve donc les matrices D et P directement par cette commande.

6.7 Equations et systèmes différentiels

MAPLE dispose de beaucoup de possibilités pour résoudre symboliquement, numériquement ou graphiquement des équations différentielles, des systèmes différentiels ou des équations aux dérivées partielles. Nous ne nous intéresserons ici qu'aux deux premiers types d'équations, et uniquement à quelques outils de MAPLE. De nombreuses options permettent de choisir la méthode de résolution. Nous laissons le lecteur souhaitant en savoir plus fouiller l'aide, et indiquons ici que le minimum.

6.7.1 Résolution exacte

La commande de résolution des équations et systèmes différentiels est la commande `dsolve`. Elle prend en argument un ensemble contenant la ou les équation(s) ainsi que les éventuelles conditions initiales, la fonction ou l'ensemble des fonctions que l'on cherche, et éventuellement des options.

Equations sans conditions initiales. Les équations peuvent être rentrées indifféremment avec le symbole `diff` ou `D`. Les deux syntaxes suivantes sont équivalentes :

```
> restart: Eq1:=diff(x(t),t)+x(t)=0 : Eq2:=(D@@2)(x)(t)+x(t)=0;
> Sol:=dsolve(Eq1,x(t));
> Sol:=dsolve(Eq2,x(t));
```

Notez l'apparition de constantes d'intégration que MAPLE note `_C` suivi d'un chiffre. Ici, en `x` la fonction solution n'est pas stockée. On la récupère par exemple en écrivant :

```
> x:=unapply(rhs(Sol),t);
```

Essayons :

```
> x(0); x(1);
```

Equations avec conditions initiales. Commencez par entrer `restart`. On va résoudre notre équation avec conditions initiales. Cette fois, pour écrire les conditions sur la dérivée, **il est obligatoire d'utiliser la syntaxe avec D**.

```
> Eq1:=diff(x(t),t$2)+x(t)=0:
> dsolve({Eq1,x(0)=0,D(x)(0)=1},x(t));
ou on peut aussi entrer :
> dsolve({Eq1} union {x(0)=0,D(x)(0)=1},x(t));
```

Systèmes différentiels Il faut cette fois entrer les équations sous forme d'ensemble et les inconnues aussi. En cas de conditions initiales, elles doivent être mises dans le même ensemble que les équations.

```
> restart:
> Eq:=diff(x(t),t)=(x(t)+y(t))/2, diff(y(t),t)=(x(t)-y(t))/2:
> dsolve({Eq}, {x(t),y(t)});
Rajoutons des conditions initiales :
> Eq1:= Eq union {x(0)=1, y(0)=1} : Sol:=dsolve(Eq1, {x(t),y(t)});
Notez qu'en revanche les deux syntaxes suivantes ne fournissent rien (même pas un message d'erreur) :
> dsolve({Eq, x(0)=1, y(0)=1} , {x(t),y(t)});
> Eq2:= {Eq, x(0)=1, y(0)=1} : dsolve(Eq2, {x(t),y(t)});
Voyons maintenant comment récupérer par exemple la fonction x. Il suffit d'entrer :
> x:=unapply(rhs(Sol[1]),t):
Vérifions :
> x(1);
```

6.7.2 Résolution numérique

MAPLE dispose de plusieurs routines de résolution numérique. Par défaut, il utilise une méthode de Runge-Kutta modifiée, mais on peut lui imposer d'autres choix. Il suffit d'entrer l'option `numeric` à `dsolve` pour qu'il fournisse une procédure qui s'applique pour obtenir ensuite une valeur approchée en tout point. On procède comme suit :

```
> restart: Eq:=diff(y(x),x)-sin(y(x)):
> Sol:=dsolve({Eq,y(0)=1} , y(x) , numeric);
```

L'écho obtenu à l'écran peut vous paraître peu explicite... En fait, MAPLE renvoie à une procédure. L'évaluation de `Sol` en un réel fournit ce réel et la valeur approchée de la solution en ce réel. Essayez :

```
> Sol(1) ;
```

Vous pouvez vouloir récupérer la solution approchée dans une fonction, ici que

l'on nommera `f`. La syntaxe est alors :

```
> f:=t->subs(Sol(t),y(x)):
```

Essayons :

```
> f(1);
```

```
> plot(f,0..1);
```

On peut aussi écrire :

```
> plot('f(x)',x=0..1);
```

mais ici les apostrophes sont obligatoires en raison d'un problème d'évaluation :

```
> plot(f(x),x=0..1);
```

6.7.3 Résolution graphique

On peut tracer une ou plusieurs courbes intégrales calculées de manière numérique à l'aide de la commande `DEplot` qui est disponible dans le package `DEtools`. Cette commande trace par défaut le champ de vecteurs ce qui permet de deviner l'aspect d'autres courbes intégrales.

Le premier argument est l'équation ou l'ensemble des équations, suivi de l'inconnue ou de l'ensemble des inconnues, puis l'intervalle d'intégration et enfin un ensemble de conditions initiales sous forme de liste. On peut aussi, de manière facultative, spécifier les domaines de tracé en `x` et `y`. Le premier exemple trace les deux courbes intégrales de :

$$\dot{x}(t) = \sqrt{t + x(t)}$$

qui vérifient $x(0) = 1$ et $x(0) = 2$.

```
> restart: with(DEtools,DEplot):
```

```
> Eq:=diff(x(t),t)=sqrt(t+x(t)):
```

```
> DEplot(Eq,x(t),t=0..1,{[x(0)=1], [x(0)=2]});
```

Le second exemple trace le diagramme de phase et une courbe intégrale pour le système de Goodwin :

```
> restart: with(DEtools,DEplot):
```

```
> Eq:={diff(x(t),t)=x(t)*(1-y(t)),diff(y(t),t)=y(t)*(x(t)-1)}:
```

```
> DEplot(Eq,{x(t),y(t)},t=0..7,{[x(0)=1.2,y(0)=1.5]});
```

Il est également possible de ne tracer aucune courbe intégrale, uniquement le portrait de phase. Dans ce cas, on n'indique pas de conditions initiales, mais le domaine de `x,y` est obligatoire :

```
> restart: with(DEtools,DEplot):
```

```
> Eq:={diff(x(t),t)=x(t)*(1-y(t)),diff(y(t),t)=y(t)*(x(t)-1)}:
```

```
> DEplot(Eq,{x(t),y(t)},t=0..7,x=0..2,y=0..2);
```


Chapitre 7

Quelques applications de MAPLE

7.1 Calculs numériques en grande précision

Même pour le calcul numérique, MAPLE peut être préféré à MATLAB puisqu'on peut contrôler avec MAPLE la précision des calculs. Lorsque l'on pose `Digits:=p`, MAPLE travaille avec `p` chiffres significatifs, ce qui signifie que l'évaluation d'une variable `x` donnera une valeur approchée `y` telle que $\text{abs}(1-y/x) \leq 5 \cdot 10^{-p}$ ou $\text{abs}(x-y) \leq 5 \cdot 10^{-p} \cdot \text{abs}(x)$.

Reprenons notre suite d'intégrales :

$$I_n := \int_0^1 \frac{t^{n-1}}{t-10} dt$$

pour laquelle on sait que $I_1 = \ln(9/10)$ et pour tout $n \geq 1$, $I_{n+1} = 10I_n + 1/n$.

On veut calculer I_{20} à 10^{-10} près en partant de la valeur approchée de I_1 fournie par `evalf(ln(9/10))`. On cherche le `p` que l'on doit choisir pour `Digits`. Posons $\varepsilon = 5 \times 10^{-p}$. L'erreur absolue sur I_n , ε_n vérifie donc l'inégalité :

$$\varepsilon_{n+1} \leq 10\varepsilon_n + \frac{\varepsilon}{n};$$

de plus, on sait que $\varepsilon_1 \leq \ln(10/9)\varepsilon$ et l'on cherche à réaliser $\varepsilon_{20} \leq 10^{-10}$.

1. On pose provisoirement $\varepsilon'_n = \varepsilon_n/\varepsilon$. Ecrire l'inéquation de récurrence dont est solution ε'_n .
2. Résoudre l'équation à l'aide de MAPLE (symboliquement, puis appliquez `evalf`). On utilisera la commande `rsolve` en s'aidant de l'aide en ligne.
3. Quelle valeur choisit-on pour `p` ? Calculez numériquement I_{20} .

Attaquons le problème par une autre méthode. On part de $I_{40} = 0$, ce qui nous fait commettre une erreur absolue au plus égale à $\varepsilon_{40} = 1/(9 \times 41)$, puis l'on

remonte jusqu'à I_{20} par :

$$I_n = \frac{I_{n+1}}{10} - \frac{1}{10n}.$$

L'erreur se propage en :

$$\varepsilon_n \leq \frac{\varepsilon_{n+1}}{10} + \frac{\varepsilon}{10n}$$

ce qui donne finalement :

$$\varepsilon_{20} \leq \varepsilon \left(\sum_{j=1}^{20} \frac{1}{10^j(19+j)} \right) + \frac{\varepsilon_{40}}{10^{20}}.$$

1. Quelle valeur choisit-on ici pour p ?
2. Comparez avec la première solution. Commentaires ?
3. Calculez numériquement I_{20} .

7.2 Systèmes dynamiques linéaires

Résolvez graphiquement quelques systèmes dynamiques linéaires et commentez la forme du diagramme de phase à l'aide des éléments propres de la matrice (essayez plusieurs matrices diagonalisables sur \mathbb{C}). Vous pourrez par exemple tester les matrices :

$$A = \begin{pmatrix} 1 & 2 \\ 4 & -1 \end{pmatrix}, \quad B = \begin{pmatrix} 0.433 & -0.5 \\ 0.5 & 0.433 \end{pmatrix}.$$

7.3 La méthode de Newton

La méthode de Newton est une procédure itérative de résolution d'une équation $f(x) = 0$. On part d'un x_0 proche de la solution, et connaissant x_n on calcule x_{n+1} comme étant l'intersection de l'axe des abscisses avec la tangente à la courbe en x_n , ce qui donne l'équation :

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

On se propose de calculer $\sqrt{2}$ en appliquant la méthode de Newton avec la fonction f définie par $f(x) := x^2 - 2$.

- Fixez la variable `Digits` à 1000.
- Ecrivez une procédure `newton` prenant en entrée le nombre d'itérations `n` et le point de départ `x0` et retournant les valeurs x_i pour i allant de 1 à `n`.

- Appliquez votre procédure avec `x0=2` et `n=5`. Affichez les résultats avec 15 chiffres significatifs. Commentaires ?
- On veut illustrer le caractère quadratique de la méthode de Newton selon lequel le nombre de décimales exactes (au moins localement) double approximativement à chaque itération. Améliorez votre procédure `newton` afin qu'elle retourne en sortie le nombre de décimales exactes à chaque étape.
- Application numérique : testez `newton(10,2)`. x_{10} contient donc 783 décimales exactes. Commentaires ?
- (plus difficile) trouver un exemple de fonction f donnant un cycle et illustrez-le avec MAPLE.

7.4 Une étude d'un modèle dynamique

On reprend à titre d'illustration le modèle simplifié sur la croissance et la pollution présenté page 102 du livre de D. Léonard et N. Van Long¹. Prenant $s = 1$ dans leur modèle, on peut résumer celui-ci au système différentiel :

$$\begin{cases} \dot{K} = K(K^{\alpha-1} - \delta) \\ \dot{P} = K^{\beta} - \gamma P \end{cases}$$

où les endogènes sont K et P et où les lettres grecques désignent des réels strictement positifs satisfaisant de plus : $\alpha < 1$ et $\beta > 1$.

1. Trouvez à l'aide de MAPLE les équilibres de ce système (on se limitera au quart de plan $K > 0$ et $P > 0$).
2. Tracez sur un même graphique² dans le plan (K, P) les courbes $\dot{K} = 0$, $\dot{P} = 0$ et un diagramme de phase dans le cas numérique $\alpha = 0.5$, $\beta = 2$, $\delta = 1.5$ (pour tracer la première courbe, on devra faire appel à une représentation paramétrique). Le point d'équilibre vous semble-t-il (localement) attractif ?
3. En revenant au cas symbolique, calculez la jacobienne au point d'équilibre, et montrez que ses deux valeurs propres sont réelles et strictement négatives. Ceci démontre que le point est effectivement localement attractif.

¹Optimal Control Theory and Static Optimization in Economics, Cambridge University Press, 1992.

²Pour superposer les trois graphiques, utiliser `display` du package `plots` (cf. aide).

Partie II

Compléments.

Chapitre 8

Correction des exercices et sujets précédents.

Avertissements : les corrections sont données pour la plupart des exercices, à titre indicatif. Plusieurs numéros de section n'ont pas été actualisés, et cela sera à vous de remettre les bons numéros ! Enfin, les sujets posés ont en général un barème très large, inutile de tout faire pour avoir 20.

8.1 Correction des exercices de SCILAB.

8.1.1 Section 1.5.

Si A et C sont carrées de même format, A/C calcule AC^{-1} tandis que $C\backslash A$ calcule $C^{-1}A$. Bien entendu, ces matrices sont en général différentes et vous ne devez pas les confondre. Retenez que l'on écrit les matrices dans l'ordre dans lequel elles apparaissent dans le produit.

Exercice 1.5.1. page 16

```
A=[16 2 3 13 ; 5 11 10 8 ; 9 7 6 12 ; 4 14 15 1] ; D=A/34;  
[sum(D,'r') sum(D,'c')]
```

```
pause
```

```
D^2
```

```
D^3
```

```
D^4
```

```
D^5
```

```
D^6
```

Exercice 1.5.2. page 17

```
[1:10].^2
```

```
2.^[1:2:11]
```

Exercice 1.5.3. page 17

```
prod(1:20)
prod(1+[1:100].^(-2))
```

Exercice 1.5.4. page 18

```
function X=ex211(T);
X=[];
// la boucle suivante créé la matrice par blocs de 4 lignes
for t=1:T
X=[X; [ones(4,1) t*ones(4,1) eye(4,4)]];
end; endfunction;
```

Exercice 1.5.5. page 18

```
x=[1 2 3]; y=[0.5 1 1.5 2]; (x'*ones(1,4).^(ones(3,1)*y))
```

Exercice 1.5.6.

```
x=[1 2 3]; y=[0.5 1 1.5 2]
(x'*ones(1,4).^(ones(3,1)*y))
```

8.1.2 Exercice 1.5.7

```
z=[1:100]'; mean([z, z.^2, z.^3])
```

8.1.3 Exercice 1.6.1. page 21

```
k=0:0.02:1 ; l=k;
deff("[K,L,Q]=CES(k,l)", "K=k", "L=l", "Q=((k.^0.1+l.^0.1)/2)^10")
deff("[K,L,Q1]=CB(k,l)", "K=k", "L=l", "Q1=sqrt(k.*l)")
xbasc(); [K,L,Q]=eval3p(CES,k,l); plot3d(K,L,Q)
[K,L,Q1]=eval3p(CB,k,l); plot3d(K,L,Q1)
xbasc(); plot3d(K,L,abs(Q-Q1)) //tracer de la difference
```

8.1.4 Exercice 1.7.1. page 23

```
x= 0: 0.05 : 6.3;
y=0.5*(sin(x)>=0.5)-0.5*(sin(x)<=-0.5)+ sin(x).*((abs(sin(x))<0.5)
plot2d(x,y);
z=sin(x); plot2d(x,z,style=7);
```

8.1.5 Exercice 1.8.2 page 24

```
function res=facto(n);
if (n==floor(n) & n>=0) // teste si n est entier et >=0
res=prod(1:n);
else res='erreur : n doit être un entier naturel'
end ; endfunction;
```

```
function res=factor(n);
// version récursive (non demandée)
// du calcul de la factorielle
if n>=2 res=n*factor(n-1) ;
else res=1;
end; endfunction;
```

8.1.6 Section 1.8.4 : amortissements**Premier programme**

```
function T=ammort1(S,i,n);
T=zeros(n,4); T(1,1)=S ; M=S*i/(1-(1+i).^(-n));
for j=1:n
T(j,2)=M ; T(j,3)=T(j,1)*i ;
T(j,4)=T(j,2)-T(j,3) ;
if j<n
T(j+1,1)=T(j,1)-T(j,4); end;
end ; endfunction;
```

Second programme

```
function T=ammort2(S,i,n);
T=zeros(n,4) ; M=S*i/(1-(1+i).^(-n));
T(:,1)=(1+i).^([0:n-1]')*(S-M/i)+M/i;
T(:,2)=M*ones(n,1) ; T(:,3)=T(:,1)*i ; T(:,4)=T(:,2)-T(:,3) ;
endfunction ;
```

Vous pouvez comparer les deux résultats en vérifiant que la différence relative est proche de 0 :

```
A=ammort1(100000, 0.00487 , 48); B=ammort2(100000, 0.00487 , 48);
norm(A-B)/norm(A)
```

8.1.7 Exercice 1.11.1 page 26

```

function [J,K]=scr114(n);
X=[1:n]; I=zeros(n,1);
I(1)=[log(9/10)];
for i=2:n
I(i)=10*I(i-1)+1/(i-1);
end;
J=1-(9>-90*X.*I)|(10<-90*X.*I) // T si OK, F sinon
pause;
K=log(9/10);
for v=1:n-1, K=K-(-1).^v/v*(1-0.9^v)*prod(n-v,n-1)/prod(1:v); end;
K=10^(n-1)*K; //valeur proposée
K=-90*n*K // verification de l'appartenance à l'intervalle

```

8.1.8 Exercice 2.1.1. page 28

```

A=rand(3,3);z=[];
for i=1:10;
x=rand(3,1);
z=[z; norm(A*x)/norm(x)];
end;
// plus ce qui suit est proche de 0, plus z est proche de norm(A)
1-max(z)/norm(A)

```

8.1.9 Exercice 3.2.1. page 31

```

A=[10 7 8 7 ; 7 5 6 5 ; 8 6 10 9 ; 7 5 9 10] ;
cond(A); // un nb très supérieur à 1 indique un système mal conditionné

```

8.1.10 Exercice sur l'interpolation page 32

```

function P=interpo(A);
X=A(:,1); Y=A(:,2); n=length(A)/2; Z=[];
// test d'égalité. donne 0 si 2 xi sont égaux, 1 sinon.
z=1;
for i=2:n
for j=1:(i-1)
if X(j)==X(i), z=0;
end;
end;
end;
// fin du test.

```

```

if z==1,
// si les xi sont distincts
for i=1:n
Z=[X.^(i-1) Z];
end;
P=(Z\Y)';
// si au moins 2 xi sont égaux
else P='erreur : les points doivent être distincts';
end; endfunction;

```

8.1.11 Exercice application des moindres carrés à Cobb-Douglas page 31

```

function par=CobbD(D);
s=size(D);
if s(1)>=3
Y=log(D(:,1)); X=[ones(s(1),1) log(D(:,2)) log(D(:,3))];
b=X\Y;
par=[exp(b(1)); b(2); b(3)];
else disp('Erreur : D doit avoir au moins 3 lignes');
end; endfunction;

```

8.1.12 Poly page 36

Avant d'écrire la fonction `phase`, notons à titre de solution à l'exercice 3.3.1. page 35 que les systèmes s'écrivent $\dot{\mathcal{X}}(t) = F(t, \mathcal{X}(t))$ avec $F(x, y) = (y, -\sin(x))$ pour le non-linéaire et $F(x, y) = (y, -x)$ pour le linéaire.

```

function X=phase(A,X0,T);
X=X0 ;
for i= 0.01 : 0.01 : 1
X=[X expm(i*T*A)*X0] ;
end;
// Graphique
plot(X(1,:),X(2,:))
// Solution
X=[[0 : 0.01 : 1]*T; X]; X=X';
endfunction;

```

Vous pourrez tester :

```

Y=phase([0,1;-1,0];[1;0],6.28);x=6.28*[0:0.01:1];plot2d(x,cos(x)-Y(:,2)');

```

8.1.13 Exercice 3.4.1 page 36

```

A=[0.7 0.5; 1 -1]; X0=[0.5;0.3]; T=5;
// solution exacte
// placer le graphique de phase en commentaire tout d'abord
exec('phase')
Ex=phase(A,X0,T);
Ex=Ex(:,2:3); // on ne garde que x et x'
// solution approchée
deff("ydot=linear(t,y,B)","ydot=B*y")
Ap=ode(X0,0,[0:0.01:1]*T,list(linear,A))';
// graphique des erreurs
plot([0:0.01:1]*T,(Ex(:,1)-Ap(:,1)).^2+(Ex(:,2)-Ap(:,2)).^2)

```

8.1.14 Système non linéaire de l'exercice 3.5.1 page 37

```

function dy=polsd(t,y);
dy=[-y(2)-(y(1)^2+y(2)^2)*y(1);y(1)-(y(1)^2+y(2)^2)*y(2)];

```

8.1.15 Linéarisé du précédent

```

function dy=polsdl(t,y);
dy=[0 -1 ; 1 0]*y;
// essayez avec X0=[0.1; 0] et T=10.

```

8.1.16 Exercice sur Fibonacci page 37

```

x=1;y=1;
// petit exo : raccourcissez la boucle ci-dessous
for i=2:10
z=x+y; x=y; y=z;
end; z
X=[0 1 ;1 1]^10*[1 ;1];
X(1)
// ou encore X=[0 1 ;1 1]^9*[1 ;1]; X(2)

```

8.1.17 Système chaotique page 38

```

function S=chaos(n,x0);
% chaos(n,x0) part de la c.i. x0

```

```

% et donne les solutions par 2 méthodes
% du sdd  $f(x)=4x(1-x)$  (recursion et formule exacte)
S=[]; theta0=asin(sqrt(x0)); x=x0;
for i=1:n
x=4*x*(1-x);
S=[S; x];
end;
S=[S (sin(2.^[1:n]*theta0).^2)'];
endfunction;

```

8.1.18 Exercice 4.1.2 page 40

```

// On centre et réduit
x=(14-11)/sqrt(2);
// Reponse à la question 1
1-0.5*(1+erf(x/sqrt(2))) // avec erf, ou bien :
1-cdfnor("PQ",14,11,sqrt(2)) // avec cdfnor
// reponse à la question 2
v=cdfnor("X",11,sqrt(2),0.4,0.6)
// simulation
X=11+sqrt(2)*rand(1000,1,"normal");
sum(X>14) // à comparer à 1000*réponse à la question 1
pause;
sum(X>v) // censé valoir 600.

```

8.1.19 Exercice 4.1.3 page 40

```

function X=loiexpo(n,lambda)
// loiexpo(n,lambda) simule n réalisations i.i.d.
// d'une loi exponentielle de par. lambda
X=-log(rand(n,1))/lambda;
// il faudrait en théorie prendre  $X=-\log(1-\text{rand}(n,1))/\lambda$ ;
// mais c'est équivalent (X suit  $U(0,1)$  ssi  $1-X$  suit cette loi)
endfunction;

```

Exercice 4.2.2., page 41

```

// par exemple avec  $\lambda=2$  et  $n=10000$  (vous pouvez augmenter n)
lambda=2; n=10000 ; y=mean(loiexpo(n,lambda));
y*(1-1.96/sqrt(n)), y*(1+1.96/sqrt(n)), 2*y/sqrt(n)

```

Augmentez n en le multipliant par 10. Les deux premières valeurs sont les bornes de l'intervalle de confiance. En général, 2 doit être compris entre ces valeurs.

La troisième valeur est l'amplitude de l'intervalle de confiance. Remarquez son comportement quand n varie.

8.1.20 Section 4.3. // les corrigés sont à refaire dans toute la 4.3

Section 4.3.1. page 41

```
% lambda=1 (n=100, p=0.01)
X=[];
for i=1:10;
X=[X; [i,binom(100,0.01,i),exp(-1)/gamma(i+1)]];
end;
disp('Erreurs pour lambda=1')
X(:,2)-X(:,3)
pause;
plot(X(:,1),X(:,2),'o',X(:,1),X(:,3),'rx')
pause;
% lambda=2 (n=100, p=0.02)
X=[];
for i=1:10;
X=[X; [i,binom(100,0.02,i),exp(-2)*2^i/gamma(i+1)]];
end;
disp('Erreurs pour lambda=2')
X(:,2)-X(:,3)
pause;
plot(X(:,1),X(:,2),X(:,1),X(:,3),'r')
pause;
```

Section 4.3.2. page 41

```
function X=sect422(n,m,eps);
% illustre la loi des gds nb
% renvoie le pourcentage des cas supérieurs au seuil
% n nb de simulations
% m nb de va dans la somme
% eps seuil
X=[];
for i=1:n
X=[X abs(mean(rand(m,1))-0.5)>eps];
end
X=100*mean(X);
```

On calcule ici pour chacune des m simulations :

$$\text{Card} \left\{ \left| \frac{1}{m} \sum_{i=1}^m X_i - 1/2 \right| > \varepsilon \right\}$$

et on fait la moyenne de ces nombres. Essayez par exemple `sect422(100,m,0.1)` pour $m=10$, $m=20$, $m=30, \dots$

Section 4.3.3. page 41

```
% 1. On simule 500 fois (X-1)/sqrt(n) avec
% X suit expo(1) et n=100
for i=1:500
S(i)=10*mean(loisexpo(100,1)-1);
end;
X=[];
% 1. Calcule proba<u pour plusieurs u
for u=-2.4:0.2:2.4
X=[X ; sum(S<u)/500];
end;
% 3. Trace le graphe et superpose celui de Phi en rouge
u=-2.4:0.2:2.4;
plot(u,X);
hold on
plot(u,(1+erf(u/sqrt(2)))/2,'ro')
```

8.1.21 Section 5.1.

Exercice 5.1.1., page 44

```
function mc=mc1(n);
// Monte-Carlo
x=rand(n,1); y=sqrt(1-x.^2); c=mean(y);
endfunction
```

c'est-à-dire que si $x = (x_1, \dots, x_n)'$, alors $y = (\sqrt{1-x_1^2}, \dots, \sqrt{1-x_n^2})$ et :

$$mc = \frac{1}{n} \sum_{i=1}^n \sqrt{1-x_i^2}.$$

Exercice 6.1.2 page 43 fonction mc=mc2(n);

```
// Monte-Carlo bis
x=rand(n,2); z=[];
for i=1:n;
z=[z x(:,1).^x(i,2)];
end;
```

```
mc=mean(z(:));
endfunction
```

ici, $x = (x_{ij})_{1 \leq i \leq n, 1 \leq j \leq 2}$, $z = (x_{i1}^{x_{j2}})_{1 \leq i, j \leq n}$ et ainsi :

$$mc = \frac{1}{n^2} \sum_{1 \leq i, j \leq n} x_{i1}^{x_{j2}}.$$

Exercice 6.1.3., question 1, page 43

Si f désigne la densité de la loi exponentielle, l'intégrale vaut :

$$\frac{1}{\lambda} \int_0^{+\infty} \sqrt{x} f(x) dx.$$

```
function int=mc3(n,lambda);
int=mean(sqrt(loexpo(n,lambda)))/lambda;
endfunction
```

Comparez avec la valeur exacte $\sqrt{\pi}/(2\lambda^{3/2})$, par exemple en étudiant l'évolution de l'erreur relative, pour plusieurs valeurs de λ , lorsque n augmente :

```
2*lambda^(3/2)*loexpo(n,lambda)/sqrt(%pi)
```

Exercice 6.1.3., question 2, page 46 Si f désigne la densité de la loi normale, l'intégrale vaut :

$$\sqrt{\pi} \int_{-\infty}^{+\infty} \cos(x/\sqrt{2}) f(x) dx.$$

```
function int=mc4(n);
int=sqrt(%pi)*mean(cos(rand(n,1,"normal")/sqrt(2))) ;
endfunction
```

8.1.22 Section 6.2

```
function y=v0(n,x);
z=x+rand(n,1,"normal")-0.5 ;
y=mean(z.*(z>=0));
endfunction
```

Après, en ligne de commande pour une étude sur $[0;3]$ on tape :

```
xbasc();X=[];t=[0:0.001:1]*3;
for i=t, X=[X,v0(1000,i)];end;
plot([t,t],[X',(t-0.5).*(t-0.5>=0)']) ;
```

8.2 Correction des exercices MAPLE

8.2.1 Section 7.1.5, manipulation d'expressions

```
simplify(tan(arctan(u))); simplify(arctan(tan(u)));
assume(u,RealRange(Open(-Pi/2),Open(Pi/2))); simplify(arctan(tan(u)));
tan(a+b+c);
expand(tan(a+b+c));
expand(ln(x*y));
assume(x>0): assume(y>0): expand(ln(y*x));
restart;
simplify(expand(sin(5*u)), [cos(u)^2=1-sin(u)^2]);
restart;
factor(x^2-3*x+2);
factor(x^2-2);
factor(sqrt(3)*(x^2-3));
Q:=(x^3+x^2-9*x-9)/(x^2-4*x+3);
factor(Q); # factorise les 2
op(1,Q); op(2,Q);
factor(op(1,Q))*op(2,Q); op(1,Q)*factor(op(2,Q));
# ne factorise que l'un des deux
normal(Q); #divise par PGCD
restart;
simplify((1+(tan(t))^2)-1/(cos(t))^2);
```

8.2.2 Section 7.2.3., résolution d'équations

```
P:=x^4+2*x^3+3*x^2+2*x+1; solve(P);
P:=x^4+2*x^3+3*x^2+4*x+5;
solve(P);
_EnvExplicit:=true:solve(P);
fsolve(P,x,complex);
solve({a*x+y+z=1,x+a*y+z=a,x+y+a*z=a^2},{x,y,z});
a:=1: solve({a*x+y+z=1,x+a*y+z=a,x+y+a*z=a^2},{x,y,z});
```

8.2.3 Section 7.3.1.

```
# Solution 1 : par les fonctions (plus lourd)
f:=(t,x)->exp(-a*t)*cos(b*x);
factor(-diff(f(t,x),t)+diff(f(t,x),x,x));
# On pose donc a=b^2.
fb:=unapply(subs(a=b^2,f(t,x)),(t,x));
```

```

S:=series(fb(t,x),b,5);
op(S);
solve(op(3,S),t);
S1:=subs(t=%,S);
op(3,S1);
# Solution 2 : par les expressions
t:='t' : x:='x' : f:=exp(-a*t)*cos(b*x);
factor(-diff(f,t)+diff(f,x,x));
fb:=subs(a=b^2,f);
S:=series(fb,b,5);
# et finit pareil

```

8.2.4 Section 7.4.1.

Suivons l'indication : $\pi^p/\zeta(p)$ doit être un entier. On en calcule une valeur approchée et on retient l'entier le plus proche.

```

for n to 5 do
evalf(Pi^(2*n)/add(1/i^(2*n),i=1..1000)); od;

```

On prévoit respectivement comme rationnels $1/6$, $1/90$, $1/945$, $1/9450$, $1/93555$. Vérifications.

```

for n to 5 do
sum(1/j^(2*n),j=1..infinity); od;

```

8.2.5 Section 7.4.2., intégrales

```

I1:=Int(sqrt(cos(t))/(sqrt(sin(t))+sqrt(cos(t))),t=0..Pi/2):
I2:=Int(sqrt(sin(t))/(sqrt(sin(t))+sqrt(cos(t))),t=0..Pi/2):
I1+I2: combine(I1+I2): value(%):

```

donc $I1 + I2 = \pi/2$.

```

with(student,changevar):
changevar(u=Pi/2-t,I1,u):

```

donc $I1 = I2$. Là, on peut conclure. Soyons plus royalistes que le roi et finissons avec MAPLE.

```

Sol:=solve({i1+i2=Pi/2,i1=i2},{i1,i2}):

```

On écrit la conclusion (ici il apparaît en seconde position dans ma session, d'où l'argument entre crochets. Vérifiez pour vous) : $I1=\text{rhs}(\text{Sol}[2])$:

8.2.6 Section 7.5.1., fonctions de production CES et Cobb-Douglas

```
restart: Q:=(alpha*K^a+(1-alpha)*L^a)^(1/a):
limit(Q,a=0,right):
```

La limite est donc une fonction de Cobb-Douglas à rendements constants.

```
# K^alpha L^(1-alpha).
Q1:=subs(alpha=1/2,Q): CD:=sqrt(K*L):
Q2:=subs(a=0.1,Q1): Q3:=subs(a=0.05,Q1):
plot3d(Q2,K=0..2,L=0..2);
plot3d(Q3,K=0..2,L=0..2);plot3d(CD,K=0..2,L=0..2);
```

Pour y voir plus clair, ajoutez des axes et choisissez la même orientation réglée par les angles theta et phi (double-cliquez sur une surface pour modifier ses paramètres).

8.2.7 Section 8.1., évaluation en grande précision

```
restart: ?rsolve
a:=rsolve({f(n) = 10*f(n-1) +1/(n-1), f(1)=ln(10/9)}, {f});
evalf(subs(n=20,rhs(a[1])));
eps:=10^(-10)/%;
Digits:=29; # proposé par ce qui précède.
In:=evalf(ln(9/10)): for i from 1 to 19 do In:=evalf(10*In+1/i) od:
In;
eps:=evalf((10^(-10)-1/(9*41*10^20))/(sum(1/(10^j*(19+j)),j=1..20)));
Digits:=8: # proposé par ce qui précède.
J:=0: for j from 39 to 20 by -1 do J:=evalf(J/10-1/(10*j)); od: J;
```

8.2.8 Section 8.2., quelques illustrations de systèmes dynamiques linéaires.

```
restart:
with(linalg,eigenvects):
A:=matrix(2,2,[1,2,4,-1]);
with(DEtools, DEplot):
Eq:={diff(x(t),t)=x(t)+2*y(t), diff(y(t),t)=4*x(t)-y(t)}:
DEplot(Eq,{x(t),y(t)}, t=0..3, x=-2..2, y=-2..2);
eigenvects(A);
Eq1:={ diff(x(t),t)=0.433*x(t)-0.5*y(t),
diff(y(t),t)=0.5*x(t)+0.433*y(t)};
```

```
DEplot(Eq1, {x(t), y(t)}, t=0..3, x=-2..2, y=-2..2);
```

8.2.9 Section 8.3., méthode de Newton

```
restart; Digits:=1000:
newton:=proc(x0,n)
local x,i;
global X,Y,f,g;
f:=x->x^2-2:
g:=D(f):
x:=x0:
X:=NULL : Y:=X:
for i to n do
x:=evalf(x-f(x)/g(x));
X:=X,x;
Y:=Y,evalf(floor(-log[10](abs(x-sqrt(2)))));
od: print(Y); end:
newton(2,10):
Z:=NULL: for i to 9 do Z:=Z,evalf(Y[i+1]/Y[i],3) od: Z :
```

8.2.10 Section 8.4., un système dynamique

```
f1:=(K,P)->K*(K^(alpha-1)-delta): f2:=(K,P)->K^beta-gamma*P:
Equil:=solve({f1(K,P),f2(K,P)},{K,P}):
```

On ne retient que le second équilibre (conditions $K > 0$ et $P > 0$).

```
Kbar:=rhs(op(1,Equil[2])): Lbar:=rhs(op(2,Equil[2])):
Bien sûr, Pbar=Kbar^beta/gamma.
Jac:=matrix(2,2,[diff(f1(K,P),K),diff(f1(K,P),P),diff(f2(K,P),K),diff(
f2(K,P),P)]):
```

A cause du zéro, les valeurs propres sont sur la diagonale. Vérifions.

```
eig:=linalg[eigenvals](Jac):
```

La première est strictement négative. Etudions la seconde en Kbar.

```
vp2:=subs(K=Kbar,eig[2]):vp2s:=simplify(vp2,symbolic):
factor(vp2s):
```

Elle est donc strictement négative. Le système est donc bien localement attractif.

8.3 Examen du 19 mai 2001.

Exercice 1. On veut calculer à l'aide de MAPLE la somme $\sum_{i=1}^5 i^2$ (la réponse est 55). Un étudiant propose de le faire via une boucle `for`, et propose le programme suivant :

```
restart: for i from 1 by 1 to 5 do z:=z+i^2: od: z;
```

1. Ce programme contient une erreur. Laquelle ? Corrigez le programme en conséquence. Dans la suite, on supposera que cette erreur a été corrigée.
2. Certains des paramètres sont optionnels. Ecrivez ce programme de la façon la plus courte possible (en conservant toutefois la boucle `for`).
3. Indiquez ce que l'on obtient comme réponse à cette ligne de commande. Puis répondez à la même question lorsque :

- on remplace dans le programme `od: par od;`
- on remplace dans le programme `z:=z+i^2: par z:=z+i^2;`

4. Proposez deux autres solutions par MAPLE sans faire de boucle. L'une de vos solutions permet-elle de calculer $\sum_{i=1}^n i^2$ lorsque n est un entier non spécifié (si oui, laquelle) ?
5. Ecrivez une version MATLAB de ce programme, puis proposez une solution sans boucle via MATLAB (réponse la plus courte possible).

Exercice 2. Voici un extrait de la procédure de l'interpolation. Etant donné une liste x à n termes deux à deux distincts, on effectue :

```
for k from 2 to n do
prod:=x[k]-x[1]
for i from 2 to k-1 do prod:=prod*(x[k]-x[i]) od;
recip[k]:=1/prod;
od;
```

Quelle est la valeur de `recip[k]` à l'issue de cette procédure ?

Exercice 3. On considère l'expression :

```
z:=tan(x^2-7*x+arcsin(sqrt(u+t)-2)) ;
```

1. Quelle est son type ? Quelles seront les réponses MAPLE à la ligne : `whattype(z)` ; `nops(z)` ; `op(1,z)` ; `op(0,z)` ;
2. Proposez un arbre informatique décomposant complètement cette expression en expressions élémentaires. En supposant que MAPLE ordonne les sous-expressions comme vous, comment feriez-vous pour récupérer `u+t` à partir de `z` ?

Exercice 4. Dans cet exercice, le découpage en questions est là pour vous aider à traiter les problèmes un par un. Vous pouvez ne donner sur votre copie que le programme final demandé à la question 4, et répondre aux questions 1, 2 et 3 sur

vosre brouillon. Si vous répondez sur votre copie à une question intermédiaire (1, 2 ou 3), indiquez-le explicitement. Le but est d'écrire une fonction MATLAB à l'aide des commandes vues en cours permettant de construire une matrice tridiagonale. Une telle matrice a la forme générale :

$$\begin{pmatrix} a_1 & b_1 & 0 & \dots & \dots & 0 \\ c_1 & a_2 & b_2 & 0 & \dots & 0 \\ 0 & c_2 & a_3 & b_3 & \dots & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \dots \\ 0 & \dots & \dots & 0 & c_{n-1} & a_n \end{pmatrix}$$

c'est-à-dire qu'elle a a_1, \dots, a_n sur la diagonale, b_1, \dots, b_{n-1} juste au dessus, c_1, \dots, c_{n-1} juste en dessous et des 0 ailleurs.

1. Ecrire une fonction MATLAB `trid` prenant en argument trois vecteurs $\mathbf{a}, \mathbf{b}, \mathbf{c}$ et un entier n avec : $n = \dim(\mathbf{a}) - 1 = \dim(\mathbf{b}) = \dim(\mathbf{c})$ et donnant la matrice ci-dessus.
2. Ecrire une autre fonction MATLAB `tridc` prenant en argument trois réels a, b, c et un entier n et donnant la matrice tridiagonale carrée d'ordre n correspondant au cas où tous les a_i (resp. b_i , resp. c_i) sont égaux à a (resp. b , resp. c).
3. Modifiez ensuite votre fonction `trid` de sorte qu'elle fasse ce qui est demandé en **1** lorsque les arguments sont trois vecteurs et un entier et qu'elle fasse ce qui est demandé en **2** si les arguments sont trois réels et un entier.
4. Terminez votre programme en ajoutant des messages d'erreurs si les arguments sont incorrects.

Exercice 5. On se propose ici d'appliquer la méthode de Gauss pour la quadrature d'intégrales. Le but est de calculer une valeur approchée d'intégrales du type :

$$\int_a^b f(t)\phi(t)dt$$

(où a, b, ϕ sont données et f varie) et l'on cherche, n étant un entier donné, une formule de la forme :

$$\sum_{i=1}^n A_i f(x_i).$$

La théorie nous apprend qu'il existe un seul choix possible des A_i et des x_i pour avoir l'égalité :

$$\int_a^b f(t)\phi(t)dt = \sum_{i=1}^n A_i f(x_i)$$

pour toute fonction f polynômiale de degré au plus $2n - 1$. De plus, les x_i doivent nécessairement être les racines d'un polynôme P_n connu, et sont des réels. On supposera que l'on connaît explicitement les valeurs des intégrales $\int_a^b t^n \phi(t)dt$ pour tout entier n , et l'on note $I_n = \int_a^b t^n \phi(t)dt$.

Par exemple, si $a = 0$, $b = +\infty$, $\phi(t) = \exp(-t)$, on cherche à approcher :

$$\int_0^{+\infty} e^{-t} f(t) dt$$

et ici on sait que $I_n = n!$. Dans la suite, on fixe un entier n .

Les questions 1, 2 et 3 sont indépendantes.

1. Etant donné l'expression du polynôme P_n , comment faites-vous avec MAPLE pour déterminer les x_i ?

2. Supposons que vous connaissiez les x_i . Ecrire une procédure MAPLE permettant de déterminer les A_i . Comment gérez-vous le problème des éventuelles erreurs d'arrondi en cas de système mal conditionné ?

3. On suppose dans la question 3 que $a = -1$, $b = 1$ et $\phi(t) = \frac{1}{\sqrt{1-t^2}}$. Dans ce cas, on peut démontrer que le polynôme P_n est l'unique polynôme vérifiant :

$$\forall \theta \in \mathbb{R}, \quad P_n(\cos(\theta)) = \cos(n\theta).$$

Comment faites-vous, à l'aide de ces informations, pour déterminer l'expression de P_n avec MAPLE, par exemple lorsque $n = 10$?

Exercice 6. Que feriez-vous pour étudier ce modèle dynamique de gestion de pêche à l'aide de MAPLE et (ou) MATLAB :

$$\begin{cases} \dot{R}(t) = R(t) - R(t)^2 - x(t) \\ \dot{K}(t) = x(t) - K(t) \end{cases}$$

où :

- $R(t)$ est le nombre de poissons à la date t .
- $x(t)$ est le nombre de poissons pêchés à la date t .
- $K(t)$ est la capacité du bassin de la pêche à la date t .

On supposera qu'il existe une relation entre x , K et R . Vous pourrez par exemple supposer que $x(t) = R(t)\sqrt{K(t)}$ pour tout t .

8.4 Examen du 4 mai 2002.

Exercice 1. Comment feriez-vous à l'aide de MATLAB et de MAPLE pour calculer le produit :

$$\prod_{i=1}^{100} \left(1 + \frac{\sin(i) + 1/i}{i^2} \right)$$

On attend, pour chaque logiciel, une réponse avec une boucle puis une réponse sans boucle, la seconde devant être la plus courte possible.

Exercice 2. On considère l'expression :

$$A := \sqrt{(\sin(x + 3y \cos(z) - t^2))^2 + \frac{1}{1 + (\tan(\varphi))^3}}.$$

1. Quelle est son type ? Quelles seront les réponses MAPLE à la ligne :
`whattype(A)` ; `nops(A)` ; `op(1,A)` ;
2. Proposez un arbre informatique décomposant complètement cette expression en expressions élémentaires. En supposant que MAPLE ordonne les sous-expressions comme vous, comment feriez-vous pour récupérer z et φ à partir de A ?

Exercice 3. Créer une procédure MAPLE et une fonction MATLAB prenant en entrée un entier $n \geq 1$ et donnant en sortie la matrice carrée d'ordre n dont le terme d'indices (i, j) est i^2/j . Dans MATLAB, une réponse rapide (sans boucle) sera préférée à une réponse avec boucle.

Exercice 4 On s'intéresse dans cet exercice à exprimer, lorsque n est un entier au moins égal à 1, l'expression $\sin(nt)$ en fonction de $\sin(t)$ et de $\cos(t)$.

1. L'entier n étant donné, comment feriez-vous à l'aide de MAPLE pour répondre à cette question ?
2. Voici les réponses obtenues pour n variant de 1 à 4 :

$$\begin{aligned} & \sin(t) \\ & 2 \sin(t) \cos(t) \\ & 4 \sin(t)(\cos(t))^2 - \sin(t) \\ & 8 \sin(t)(\cos(t))^3 - 4 \sin(t) \cos(t) \end{aligned}$$

Ces réponses font penser que pour tout entier n , $\sin(nt)$ est de la forme $\sin(t)U_n(\cos(t))$, où U_n est un polynôme de degré $n - 1$. On admettra que ceci est effectivement valable pour tout n . Au vu des premiers résultats obtenus, donnez les expressions des $U_n(X)$ pour n variant de 1 à 4 (pour $n = 3$, la réponse est : $4X^2 - 1$).

3. Donner une expression MAPLE qui à partir de n , fournit immédiatement l'expression $U_n(X)$. Ainsi, pour $n = 3$, on veut obtenir (à l'ordre près des termes) $4X^2 - 1$.

Exercice 5 On considère une fonction de production de Cobb-Douglas :

$$Q = K^\alpha L^\beta.$$

1. Comment feriez-vous à l'aide de MATLAB ou de MAPLE pour vérifier :
 - qu'elle est concave lorsque $\alpha = 1/2$ et $\beta = 1/3$.
 - qu'elle est convexe lorsque $\alpha = 1$ et $\beta = 3$.

2. Comment feriez-vous pour obtenir la condition nécessaire et suffisante sur le couple (α, β) pour qu'elle soit concave ?

Exercice 6. Expliquez les limites de l'utilisation d'un logiciel comme MAPLE et MATLAB pour résoudre des problèmes numériques et symboliques. Comment faire pour vérifier certains résultats qui pourraient vous paraître douteux ?

Chapitre 9

Sujets posés antérieurement.

Interrogation de décembre 2005.

On s'intéresse à donner une formule d'approximation de $I(f) = \int_{-1}^1 f(t)dt$ de la forme :

$$I(f) \approx I_5(f) = c_{-1}f(-1) + c_{-1/2}f(-1/2) + c_0f(0) + c_{1/2}f(1/2) + c_1f(1).$$

Pour cela, on appelle P_f le polynôme d'interpolation de f aux points $-1, -1/2, 0, 1/2, 1$ et l'on pose par définition $I_5(f) = I(P_f)$ (on rappelle que le degré de P_f est au plus égal à 4).

1. Montrer que la méthode est d'ordre 4 au moins.

2.

2.a. Calculer $I(1), I(X), I(X^2), I(X^3), I(X^4)$. En déduire que les coefficients c_i sont solution du système linéaire suivant :

$$\begin{cases} c_{-1} + c_{-1/2} + c_0 + c_{1/2} + c_1 = 2 \\ -c_{-1} - \frac{1}{2}c_{-1/2} + \frac{1}{2}c_{1/2} + c_1 = 0 \\ c_{-1} + \frac{1}{4}c_{-1/2} + \frac{1}{4}c_{1/2} + c_1 = \frac{2}{3} \\ -c_{-1} - \frac{1}{8}c_{-1/2} + \frac{1}{8}c_{1/2} + c_1 = 0 \\ c_{-1} + \frac{1}{16}c_{-1/2} + \frac{1}{16}c_{1/2} + c_1 = \frac{2}{5} \end{cases}$$

2.b. Quelle commande SCILAB proposez-vous pour résoudre ce système ?

2.c. La réponse SCILAB est $[0.1555556; 0.7111111; 0.2666667; 0.7111111; 0.1555556]$.

Que constatez-vous pour les c_{-i} par rapport aux c_i ? Démontrer ceci (on pourra écrire un système linéaire dont est solution $(c_1 - c_{-1}, c_{1/2} - c_{-1/2})$).

2.d. (**cette question n'est pas nécessaire pour la suite**). Sachant que $0,111\cdots = \frac{1}{9}$, quelles sont, au vu des résultats de SCILAB, les valeurs prévisibles pour les coefficients ?

3. Au vu du résultat à la question **2.c**, la formule d'approximation prend la forme :

$$I(f) \approx I_5(f) = c_0 f(0) + c_{1/2} (f(1/2) + f(-1/2)) + c_1 (f(1) + f(-1)).$$

3.a. Montrer que la méthode est d'ordre 5 (au moins).

3.b. On suppose désormais f de classe C^6 sur $[0, 1]$ et l'on pose $M_6 = \sup |f^{(6)}|$. On considère le polynôme Q de degré minimal tel que $Q(x) = f(x)$ pour $x \in \{-1, -1/2, 0, 1/2, 1\}$ et $Q'(0) = f'(0)$. On rappelle que pour tout t :

$$|f(t) - Q(t)| \leq \frac{M_6}{6!} \left| t \prod_{j=-2}^2 (t - j/2) \right|.$$

Montrer alors que l'erreur prend la forme :

$$|I(f) - I_5(f)| \leq CM_6,$$

où C est une constante que l'on exprimera à l'aide d'une intégrale dans laquelle le produit aura l'expression la plus simple possible (sans \prod et en groupant les termes opposés).

3.c. Donner un majorant simple de C . Comment feriez-vous pour calculer C à l'aide de SCILAB ?

Partiel de janvier 2006.

Exercice 1. Soit α un nombre réel strictement supérieur à 1. L'équation :

$$x^2 + 2\alpha x + 1 = 0$$

a deux racines réelles x_1 et x_2 distinctes (dépendant de α) telles que $x_1 + x_2 = -2\alpha$ et $x_1 x_2 = 1$.

1. On rappelle que pour ε petit, on a : $\sqrt{1 + \varepsilon} \approx 1 + \frac{\varepsilon}{2}$. D'où vient cette approximation (on ne demande pas de la démontrer, juste de donner l'argument crucial) ?

2. Donner les expressions de x_1 et x_2 , en prenant pour x_1 la plus petite des deux racines (on aura $x_1 < x_2 < 0$).

3. On suppose que α est très grand. Donner une valeur approchée de x_1 et de x_2 . Laquelle des deux racines est la plus facile à calculer dans un logiciel à précision limitée en utilisant les formules du **2** ? (on pourra tenter de prévoir ce que dirait un logiciel travaillant avec 3 chiffres significatifs lorsque $\alpha = 10^{10}$, si l'on fait le calcul direct).

4. Ayant déterminé la racine la plus simple, comment feriez-vous pour calculer la seconde ? Proposez une fonction SCILAB prenant α en entrée (supposé grand) et retournant les deux racines.

Exercice 2. Soit $I = [0; 1]$. A tout entier $n \geq 2$ et $k \in \{0, \dots, n-1\}$, on associe l'ensemble :

$$A_k^n = \left[\frac{k}{n}; \frac{k+1}{n} \right[.$$

On note $\chi_{n,k}$ la fonction définie sur I telle que $\chi_{n,k}(x) = 1$ lorsque $x \in A_k^n$ et 0 sinon. Ainsi, pour tout n , la fonction $\sum_{k=0}^{n-1} \chi_{n,k}$ est la fonction qui vaut 1 sur $[0; 1[$ et 0 en 1. On rappelle que pour f continue sur un compact K de \mathbb{R}^p à valeurs réelles, le module de continuité ω_f défini par :

$$\omega_f(\delta) = \sup\{|f(x) - f(y)|, (x, y) \in K^2, \|x - y\| \leq \delta\}$$

tend vers 0 lorsque δ tend vers 0, et que $|f(x) - f(y)| \leq \omega_f(\|x - y\|)$.

1. Soit f continue sur le compact I (ici $p = 1$). On approche f par la fonction :

$$P_n = \sum_{k=0}^{n-1} f(k/n) \chi_{n,k}$$

(c'est-à-dire que P_n vaut $f(k/n)$ sur A_k^n).

1.a. Montrer que :

$$\forall t \in I, |f(t) - P_n(t)| \leq \omega_f(1/n).$$

1.b. On approche $\int_0^1 f(t)dt$ par $\int_0^1 P_n(t)dt$. Ecrire la formule d'approximation obtenue. Reconnaissez-vous la méthode aboutissant à cette formule ? Retrouver à l'aide de **1.a** que lorsque n tend vers l'infini, l'approximation tend vers la valeur exacte. Prenant $f = \sqrt{\cdot}$ (pour lequel $\omega_f(\delta) = \sqrt{\delta}$), programmer dans SCILAB une fonction prenant la précision souhaitée ε en entrée et donnant une valeur approchée de l'intégrale à ε près en sortie.

2. Soit g continue sur le compact $I^2 = I \times I$ (ici, $p = 2$ et on munit I^2 de la distance $d((x_1, y_1), (x_2, y_2)) = |x_1 - x_2| + |y_1 - y_2|$). On approche g par la fonction :

$$Q_n : \left[(x, y) \mapsto \sum_{0 \leq k, l \leq n-1} f(k/n, l/n) \chi_{n,k}(x) \chi_{n,l}(y) \right]$$

(c'est-à-dire que Q_n vaut $g(k/n, l/n)$ sur $A_k^n \times A_l^n$).

2.a. Montrer que :

$$\forall (x, y) \in I^2, |g(x, y) - Q_n(x, y)| \leq \omega_g(2/n).$$

2.b. On approche $\int_0^1 (\int_0^1 g(x, y) dx) dy$ par $\int_0^1 (\int_0^1 Q_n(x, y) dx) dy$. Ecrire la formule d'approximation obtenue. Retrouver à l'aide de **2.a** que lorsque n tend vers l'infini, l'approximation tend vers la valeur exacte.

2.c. Programmer la méthode dans SCILAB.

Exercice 3. Dans le problème de la sensibilité aux erreurs d'arrondis pour le calcul des valeurs propres, on peut montrer que le critère important est la borne inférieure des conditionnements des matrices de passage de la base canonique à une base de vecteurs propres. On se donne une matrice carrée A de taille n dont on cherche à déterminer les valeurs propres. On note \mathcal{P} l'ensemble des matrices de passages de la base canonique à une base de vecteurs propres de A . On cherche donc à estimer $\inf\{\text{cond}(P), P \in \mathcal{P}\}$.

1. Soit P un élément de \mathcal{P} . On note C_1, \dots, C_n les colonnes de P . Montrer que pour tout n -uplet $(\alpha_1, \dots, \alpha_n) \in (\mathbb{R}^*)^n$, la matrice dont les colonnes sont $\alpha_1 C_1, \dots, \alpha_n C_n$ est aussi un élément de \mathcal{P} (indication : quelle est l'interprétation des colonnes de la matrice de passage ?).

2. Prenons par exemple $n = 2$. On a aboutit à la matrice

$$P = \begin{pmatrix} 2 & 1 \\ 1 & 3 \end{pmatrix}.$$

Déterminer le conditionnement de la matrice P relativement à la norme 1. Déterminer parmi les matrices :

$$P_{\alpha, \beta} = \begin{pmatrix} 2\alpha & \beta \\ \alpha & 3\beta \end{pmatrix}$$

(issues de la question 1) celle(s) de conditionnement minimal ; on se limitera à α et β strictement positifs.

Partiel de septembre 2006.

Exercice 1. Soit α un nombre réel strictement supérieur à 1. L'équation :

$$x^2 + 2\alpha x + 1 = 0$$

a deux racines réelles x_1 et x_2 distinctes (dépendant de α) telles que $x_1 + x_2 = -2\alpha$ et $x_1 x_2 = 1$.

1. On rappelle que pour ε petit, on a : $\sqrt{1 + \varepsilon} \approx 1 + \frac{\varepsilon}{2}$. D'où vient cette approximation (on ne demande pas de la démontrer, juste de donner l'argument crucial) ?

2. Donner les expressions de x_1 et x_2 , en prenant pour x_1 la plus petite des deux racines (on aura $x_1 < x_2 < 0$).

3. On suppose que α est très grand. Donner une valeur approchée de x_1 et de x_2 . Laquelle des deux racines est la plus facile à calculer dans un logiciel à précision limitée en utilisant les formules du 2 ?

4. Ayant déterminé la racine la plus simple, comment feriez-vous pour calculer la seconde ?

Exercice 2. Le but est de calculer numériquement la constante d'Euler γ , définie par :

$$\gamma = \lim_{n \rightarrow +\infty} \left(\left(\sum_{k=1}^n \frac{1}{k} \right) - \ln(n) \right).$$

Les deux premières questions sont indépendantes, et préliminaires à la suite de l'exercice.

1. Montrer que pour tout $x \in \mathbb{R}^+$, on a :

$$(1) \quad \frac{x^2}{2} - \frac{x^3}{6} \leq x - \ln(1+x) \leq \frac{x^2}{2}$$

et que pour $x \in [0; 3]$, on a :

$$(2) \quad 0 \leq x - \ln(1+x) \leq \frac{x^2}{2}.$$

2. Soit $\alpha > 1$. Montrer que :

$$\sum_{k=n+1}^{+\infty} \frac{1}{k^\alpha} \leq \frac{1}{(\alpha-1)n^{\alpha-1}}.$$

3. On introduit pour tout $k \geq 1$:

$$u_k = \frac{1}{k} - \ln\left(\frac{k+1}{k}\right).$$

3.a. Montrer que $\lim_{N \rightarrow +\infty} \sum_{k=1}^N u_k = \gamma$.

3.b. A l'aide de **1** (relation **(2)**) et de **2.**, donner un encadrement de :

$$\sum_{k=N+1}^{+\infty} u_k.$$

En déduire, $\varepsilon > 0$ étant donné, comment choisir $N(\varepsilon) \in \mathbb{N}^*$ de sorte que :

$$0 \leq \sum_{k=N+1}^{+\infty} u_k \leq \varepsilon.$$

3.c. En déduire, ε étant donné, comment fournir un encadrement d'amplitude ε de γ . La convergence de la méthode vous semble-t-elle rapide ?

3.d. Programmer la méthode dans SCILAB.

4. On rappelle que $\sum_{k=1}^{+\infty} \frac{1}{k^2} = \pi^2/6$. On introduit la suite :

$$v_k = u_k - \frac{1}{2k^2}.$$

4.a. Déterminer la limite de $\lim_{N \rightarrow +\infty} \sum_{k=1}^N v_k$.

4.b. En s'inspirant de **3.b-c.**, montrer comment obtenir un encadrement d'amplitude ε de γ à l'aide de ce qui précède (on fera les adaptations nécessaires, en utilisant notamment **(1)** et non **(2)**, et l'on prendra garde au fait que $v_k \leq 0$). Comparer la convergence des deux méthodes.

Exercice 1. On souhaite déterminer une valeur approchée de la somme :

$$S = \sum_{k=1}^{+\infty} \frac{1}{k^6}.$$

Pour cela, on introduit, pour $N \geq 1$ les sommes partielles :

$$S_N = \sum_{k=1}^N \frac{1}{k^6},$$

et le reste de la série : $R_N = S - S_N$.

Nous considérons aussi la série :

$$S' = \sum_{k=1}^{+\infty} \frac{(-1)^k}{k^6}.$$

avec, pour $N \geq 1$ ses sommes partielles :

$$S'_N = \sum_{k=1}^N \frac{(-1)^k}{k^6},$$

et les restes : $R'_N = S' - S'_N$.

1.

1.a. Déterminer, pour toute valeur de k :

$$\frac{1}{k^6} + \frac{(-1)^k}{k^6}$$

(on distinguera selon que k est pair ou impair), et en déduire la relation :

$$S + S' = \sum_{p=1}^{+\infty} \frac{2}{(2p)^6}.$$

1.b. En déduire que :

$$S = -\frac{32}{31}S'.$$

Pour calculer une valeur approchée de S , on peut donc aussi déterminer une valeur approchée de S' . Nous allons maintenant voir, entre les deux, la méthode qui semble la plus raisonnable. Pour la suite, il est suggéré de chercher des arguments simples. Presque tout se déduit du cours ou du TD.

2.

2.a. Montrer que $R_N \geq 0$ (suggestion possible : quelle est la monotonie de S_N ?), et que :

$$R_N \leq \frac{1}{6N^5}.$$

2.b. Justifier que

$$|R'_N| \leq \frac{1}{N^6}.$$

2.c. Au vu des résultats précédents, quelle série semble converger la plus vite ?

3. L'une des méthodes présente l'avantage que deux sommes partielles consécutives encadrent la somme totale. Préciser laquelle des deux, et expliquer si cela est pour vous un avantage ou un inconvénient. Au vu de la réponse à **2.c** et à **3**, laquelle des deux séries calculeriez-vous ? Justifier.

Exercice 2. On s'intéresse à résoudre $f(x) = 0$ sur un intervalle $I = [a, b]$ dans lequel on suppose que l'équation admet une unique solution.

1. Rappeler les avantages et inconvénients des méthodes suivantes : dichotomie, sécante et Newton. On s'intéressera notamment à leur convergence éventuelle, leur vitesse de convergence, les hypothèses qu'elles exigent sur f .

2. Donner un exemple où l'application de la méthode de Newton aboutit à la suite $((-1)^n)_n$.

3. On veut calculer $\sqrt{3}$ par la méthode de Newton appliquée à la fonction $f(x) = x^2 - 3$, en partant de $x_0 = 2$. On peut montrer que dans ce cas, la suite $(x_n)_n$ est strictement décroissante, et convergente de limite $\sqrt{3}$, ce qui n'est pas demandé. On admettra que $\sqrt{3} > 1,7$ pour la suite de l'exercice. La méthode aboutit à la suite :

$$x_{n+1} = \frac{x_n^2 + 3}{2x_n}.$$

3.a. On note $\varepsilon_n = x_n - \sqrt{3} \in]0; 3[$ l'erreur au rang n . Vérifiez que :

$$\varepsilon_{n+1} = \frac{\varepsilon_n^2}{2x_n},$$

et en déduire que :

$$\varepsilon_{n+1} \leq \frac{\varepsilon_n^2}{a},$$

avec $a = 3, 4$.

3.b. En déduire que :

$$\varepsilon_n \leq \frac{\varepsilon_0^{2^n}}{a^{2^n - 1}},$$

puis, en remarquant que $\varepsilon_0/a < 0,1$, que :

$$\varepsilon_n \leq \frac{a}{10^{2^n}}.$$

Quelle valeur de n choisir pour avoir $x_n - \sqrt{3} \leq 10^{-p}$?

Exercice 3.

1. Soit g une fonction dérivable en un point a . Rappeler pourquoi, si x est proche de a , $g(x)$ peut être approché par $g(a) + g'(a)(x - a)$ (on ne demande pas

d'explication rigoureuse, juste une intuition fondée).

2.a. On suppose maintenant que g est deux fois dérivable sur un intervalle ouvert I contenant a , et l'on note $M_2 = \sup_I |g'|$. On introduit :

$$K(h) = g(a+h) - (g(a) + g'(a)h).$$

Déterminer K' et K'' .

2.b. Montrer que :

$$|K''(h)| \leq M_2,$$

puis en déduire que :

$$|K(h)| \leq \frac{M_2}{2}h^2$$

(une démonstration dans le cas où $h \geq 0$ est suffisante).

2.c. Expliquer pourquoi ce que l'on vient de faire rend rigoureux le résultat de la question **1**.

3. On veut déterminer, sans calculatrice, une valeur approchée de 101. La formule du **1** avec $a = 1$ dit que si x est proche de 1, alors \sqrt{x} est proche de $1 + \frac{x-1}{2}$.

3.a. Vous paraît-il raisonnable de faire ce calcul avec $x = 100$? Pourquoi ?

3.b. On peut aussi remarquer que $\sqrt{101} = 10\sqrt{1,01}$, et approcher $\sqrt{1,01}$ par la formule précédente avec $x = 1,01$. Cela vous paraît-il plus raisonnable (pourquoi) ? Déterminer la valeur approchée ainsi obtenue, et un majorant de l'erreur commise (pour ce dernier point, on prendra $M_2 = \frac{1}{2}$).

Interrogation de janvier 2007.

Exercice 1. On considère x_1, x_2, x_3 trois réels distincts, $(y_1, y_2, y_3, z_1, z_2, z_3) \in \mathbb{R}^6$. On considère l'ensemble \mathcal{P} des polynômes P tels que :

$$\forall i \in \{1, 2, 3\}, \quad P(x_i) = y_i, \quad P'(x_i) = z_i.$$

1. On rappelle que dans \mathcal{P} , il y a un unique élément de degré minimal, que l'on note ici P_0 . Que peut-on dire du degré de P_0 ?

2. Soit $P \in \mathcal{P}$. Que peut-on dire de $P - P_0$?

3. Ecrire une fonction SCILAB dont la première ligne est :

```
function c=P0(x1,x2,x3,y1,y2,y3,z1,z2,z3)
```

retournant dans c les coefficients de P_0 .

Exercice 2. Soit $\sigma \in]0; 1]$. On note P_0 le polynôme d'interpolation (de degré au plus 1) d'une fonction f aux points σ et $-\sigma$, et l'approximation de $\int_{-1}^1 f(x)dx$ par $\int_{-1}^1 P_0(x)dx$ aboutit à une formule du type :

$$\int_{-1}^1 f(x)dx \approx c_1 f(-\sigma) + c_2 f(\sigma).$$

1.

1.a. Vérifier que :

$$P_0(x) = f(-\sigma) + \frac{f(\sigma) - f(-\sigma)}{2\sigma}(x + \sigma).$$

1.b. Calculer $\int_{-1}^1 P_0(x) dx$.

1.c. En déduire que $c_1 = c_2 = 1$.

2. Désormais, la formule d'approximation prend donc la forme :

$$\int_{-1}^1 f(x) dx \approx f(-\sigma) + f(\sigma),$$

On rappelle que la méthode est d'ordre q si et seulement si, pour tout $k \leq q$, on a :

$$\int_{-1}^1 x^k dx \approx (-\sigma)^k + \sigma^k.$$

2.a. Sachant que la méthode est basée sur 2 points, quel encadrement peut-on donner de l'ordre q de la méthode ?

2.b. Déterminer σ pour que la formule soit (au moins) d'ordre 2.

2.c. Vérifier que la formule ainsi obtenue est d'ordre 3.

2.d. Si f est de classe C^4 sur $[-1; 1]$, en notant Q le polynôme de degré au plus 3 tel que $Q(x_i) = f(x_i)$ et $Q'(x_i) = f'(x_i)$, aboutir à une formule d'erreur faisant intervenir $M_4 = \sup |f^{(4)}|$ et $\int_{-1}^1 (x^2 - \sigma^2)^2 dx$.

Partiel de janvier 2007.

Petite question. Quelles sont, selon vous, les avantages et les limites de l'utilisation d'un logiciel tel SCILAB pour résoudre des problèmes numériques ? Comment remédier à certaines limites ?

Exercice 1. On considère x_1, x_2, x_3, x_4 quatre réels distincts, $(y_1, y_2, y_3, y_4, z_1, z_2, z_3, z_4) \in \mathbb{R}^8$. On considère l'ensemble \mathcal{P} des polynômes P tels que :

$$\forall i \in \{1, 2, 3, 4\}, \quad P(x_i) = y_i, \quad P'(x_i) = z_i.$$

1. On rappelle que dans \mathcal{P} , il y a un unique élément de degré minimal, que l'on note ici P_0 . Que peut-on dire du degré de P_0 ?

2. Soit $P \in \mathcal{P}$. Que peut-on dire de $P - P_0$?

3. On considère maintenant une fonction f dérivable pour laquelle on a : $y_i = f(x_i)$ et $z_i = f'(x_i)$ pour tout i .

3.a. On considère la suite formée par les itérées issues de la méthode de Newton

pour la suite $(u_n)_n$: $u_{n+1} = g(u_n)$. Rappeler l'expression de g en fonction de f .

3.b. En déduire les valeurs z_1, z_2, z_3, z_4 en fonction de x_1, x_2, x_3, x_4 et de y_1, y_2, y_3, y_4 , de sorte que si $u_0 = x_1$, on a : $u_1 = x_2, u_2 = x_3, u_3 = x_4$, puis $u_4 = x_1$.

3.c. Que remarquez vous pour la suite $(u_n)_n$?

3.d. Déduire de **1** et de **3.b.** comment fabriquer une fonction f de sorte que la suite des itérées de Newton partant de x_1 soit $x_1, x_2, x_3, x_4, x_1, x_2, x_3, x_4, \dots$

Exercice 2. On veut calculer les intégrales :

$$I_A = \int_0^A e^{-x} x^{5/2} dx,$$

pour $A = 1$ et $A = +\infty$. Les questions sont indépendantes entre elles (mais ce n'est pas le cas d'une sous-question à l'autre).

1. On suppose ici que $A = +\infty$. Expliquer comment, à l'aide de lois exponentielles et de la méthode de Monte-Carlo, comment procéder. Donner la (ou les) ligne(s) de commande SCILAB que vous allez entrer. Quelles semblent être les limites de la méthode de Monte-Carlo. Comment y remédier ?

2. On suppose ici que $A = 1$. On va se servir d'un développement en série.

2.a. Justifier que

$$e^{-x} x^{5/2} = \sum_{n=0}^{+\infty} \frac{(-1)^n x^{5/2+n}}{n!}.$$

2.b. On admet que l'on peut intervertir la série et l'intégrale. Montrer qu'alors :

$$I_1 = \sum_{n=0}^{+\infty} \frac{(-1)^n}{n!(7/2+n)}.$$

2.c. On approche I_1 par une somme partielle S_N de la série. Que peut-on dire de l'erreur ? En déduire comment, dans SCILAB, trouver un encadrement de I_1 d'amplitude 10^{-5} (on écrira la ligne de commande).

3. On suppose toujours que $A = 1$. On envisage d'appliquer comme méthode composée la formule des rectangles à gauche. A quelle erreur aboutit-on ? Ecrire un fichier script SCILAB (ou une ligne de commande) permettant d'avoir I_1 à 10^{-3} près.