Méthodes numériques

Recueil d'exercices (avec corrigés)

Manfred Gilli

Département d'économétrie
Université de Genève
Décembre 2003

Table des matières

Index

1 Introduction

1.1 sys-expl01

Afin de se familiariser avec le système opératif Windows et l'éditeur de texte Bloc-notes on suggère d'expérimenter les commandes suivantes :

1. Créer un répertoire `oiec` sur l'unité `h:`.
2. Copier le fichier `ex1.dat` qui se trouve sur l'unité `v:` dans le répertoire `metri\oiec` dans le répertoire crée sous 1.
3. Vérifier l'existence du fichier ainsi copié. Quelle est sa taille en octets ?
4. Renommer le fichier `ex1.dat` en `ex1.txt`.
5. A l'aide de l'éditeur de texte Bloc-notes parcourir le texte du fichier `ex1.txt` et quitter le fichier sans le modifier.
6. Créer un nouveau fichier. Insérer votre nom en première ligne suivi d'une ligne blanche puis du texte : "`Calcul symbolique sur ordinateur avec Maple V`". Introduire à nouveau une ligne blanche.
7. Insérer le fichier `ex1.txt` à la fin du fichier courant. On procédera avec "copier/coller" entre deux fenêtres Bloc-notes.
8. Corriger l'orthographe du mot mathématiques du texte inséré.
9. Détruire, puis restituer des caractères, mots et lignes de votre choix dans le texte.
10. Localiser toutes les occurences de la chaîne de caractères "`no`".
11. Déplacer le deuxième paragraphe à la fin du texte.
12. Remplacer la chaîne de caractères "`aussi capable`" du premier paragraphe (ligne 6) par l'expression "`egalement en mesure`".
13. Remplacer dans le texte le mot logiciel par le mot programme en utilisant les fonctions de recherche d'une chaîne de caractères.

2 Programmation

2.1 mlab00

Evaluer la précision de calcul de MATLAB. On rappelle que la précision est définie par la valeur de la plus petite différence entre deux nombres que le programme est capable de détecter.

Le problème dit "$3n + 1$" de la théorie des nombres s'énonce comme : Choisir un entier positif. Si cet entier est pair on le divise par 2, sinon on le multiplie par 3 et on additionne 1. Appliquer au resultat le même procédé jusqu'à ce que l'on obtienne 1. Question : Existe-t-il un entier positif pour lequel ce procédé continue indéfiniment ? Programmer le procédé avec MATLAB.

2.2 mlab01a

Expérimenter toutes les commandes MATLAB presentées au cours. Capter la séance de travail (essais et résultats) dans un fichier et le conserver.

2.3 mlab01c

Le modèle input-output de Leontief peut s'écrire

$$x = C\,x + d \tag{1}$$

où x $(n \times 1)$ est le vecteur des productions des n secteurs de l'économie, d $(n \times 1)$ est le vecteur des demandes finales de chaque secteur et C $(n \times n)$ est la matrice d'input-output donnant, par son élément générique c_{ij}, la proportion de biens du secteur i utilisés pour produire une unité des biens du secteur j.

Lorsque C et d contiennent des valeurs non négatives et que la somme de chaque colonne de C est inférieure ou égale à un, le vecteur de production x solution de (1) existe et contient des éléments non négatifs.

1. Isoler x en l'exprimant en fonction de C, d et de la matrice identité.
2. Calculer le vecteur de production x pour une économie à trois secteurs décrite par

$$C = \begin{bmatrix} .2 & .2 & .0 \\ .3 & .1 & .3 \\ .1 & .0 & .2 \end{bmatrix} \quad \text{et} \quad d = \begin{bmatrix} 40 \\ 60 \\ 80 \end{bmatrix} .$$

   Utiliser l'opérateur \ de MATLAB.
3. On peut calculer $(I - C)^{-1}$ par la formule

$$(I - C)^{-1} = I + C + C^2 + C^3 + \cdots \tag{2}$$

   si $C^m \to 0$ lorsque $m \to \infty$.
   Cela signifie que l'on peut approcher $(I - C)^{-1}$ aussi précisément qu'on le désire par la formule (2) en prenant m suffisamment grand.
   (a) Soit $D_m = I + C + \cdots + C^m$. Trouver une relation liant D_{m+1} à D_m qui ne fait pas intervenir C^{m+1}.
   (b) Grâce à la relation précédente, proposer une procédure itérative pour calculer $(I - C)^{-1}$ selon (2).
   (c) Vérifier votre réponse avec MATLAB pour les données de la question 2.

2.4 mlab01b

Les problèmes donnés par la suite sont à résoudre à l'aide du logiciel MATLAB. On conservera les résultats dans un fichier.

1. On considère le vecteur $x^{(0)} = [.2\ .3\ .5]'$, qui représente les parts de marché de 3 entreprises. On admet que, pendant une période unitaire donnée, la part de marché cédée par l'entreprise j à l'entreprise i est donnée par le coefficient a_{ij}. Soit $A = \|a_{ij}\|$

$$A = \begin{bmatrix} .8 & .2 & .1 \\ .1 & .7 & .3 \\ .1 & .1 & .6 \end{bmatrix}$$

   la matrice de transition ; on obtient l'état des parts de marché après une période $x^{(1)}$, en effectuant le produit

$$x^{(1)} = A x^{(0)} .$$

   En considérant le processus de transition stationnaire, c'est-à-dire A constante durant les périodes successives, expérimenter numériquement à partir de combien de périodes le marché sera en équilibre. (On négligera les variations à partir du troisième digit après la décimale.)
2. Expérimenter numériquement l'évolution la matrice A^n pour $n = 1, 2, \ldots$ entiers positifs croissants, si A vaut :

$$a)\ \begin{matrix} .6 & .5 \\ -.2 & 1.2 \end{matrix} \qquad b)\ \begin{matrix} .6 & .5 \\ -.16 & 1.2 \end{matrix} \qquad c)\ \begin{matrix} .6 & .5 \\ -.1 & 1.2 \end{matrix}$$

$$d)\ \begin{matrix} .9 & 1.0 \\ 0 & .9 \end{matrix} \qquad e)\ \begin{matrix} .99 & 1.0 \\ 0 & .99 \end{matrix} \qquad f)\ \begin{matrix} 1.0 & 1.0 \\ 0 & 1.0 \end{matrix}$$

2.5 mlab01d

Dire si les propositions suivantes sont vraies ou fausses. Donner un contre-exemple spécifique lorsque la proposition est fausse. Répondez en vous aidant de MATLAB.

1. Si la 1ère et la 3ème colonne de B sont les mêmes, alors la 1ère et la 3ème colonne du produit $A\,B$ sont aussi les mêmes.
2. Si la 1ère et la 3ème ligne de B sont les mêmes, alors la 1ère et la 3ème ligne de $A\,B$ sont aussi les mêmes.
3. Si la 1ère et la 3ème ligne de A sont les mêmes, alors la 1ère et la 3ème ligne de $A\,B$ sont aussi les mêmes.
4. $(A\,B)^2 = A^2\,B^2$.

2.6 mlab01e

Soient $A_{m\times n}$ et $B_{n\times p}$ deux matrices quelconques. Dire si les propositions suivantes sont vraies ou fausses. Donner un contre-exemple spécifique lorsque la proposition est fausse. Répondez en vous aidant de MATLAB.

1. Si la 1ère et la 3ème colonne de B sont les mêmes, alors la 1ère et la 3ème colonne du produit $A\,B$ sont aussi les mêmes.
2. Si la 1ère et la 3ème ligne de B sont les mêmes, alors la 1ère et la 3ème ligne de $A\,B$ sont aussi les mêmes.
3. Si la 1ère et la 3ème ligne de A sont les mêmes, alors la 1ère et la 3ème ligne de $A\,B$ sont aussi les mêmes.
4. $(A\,B)^2 = A^2\,B^2$.

2.7 mlab16

On note

$$u_1 = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}, \quad u_2 = \begin{pmatrix} -5 \\ 2 \\ 1 \end{pmatrix}, \quad u_3 = \begin{pmatrix} -1 \\ -3 \\ 7 \end{pmatrix}$$

1. Définir ces vecteurs sous Matlab.
2. Calculer $u_1 + u_2$, $u_1 + 3u_2 - 5u_3$, $u_3/5$.
3. Calculer $\|u_1\|_2$, $\|u_2\|_1$, $\|u_3\|_\infty$. (Utilisez la fonction Matlab `norm`.)

2.8 MatMult

1. A l'aide d'une boucle `for` programmer le produit scalaire de deux vecteurs $a \in \mathbb{R}^n$ et $b \in \mathbb{R}^n$

$$s = \sum_{i=1}^{n} a_i \, b_i \, .$$

   Pour $n = 5$, générer les vecteurs a et b avec la commande `rand` et vérifier votre résultat en vous servant de la commande Matlab `dot`.

2. A l'aide de trois boucles `for` programmer le produit $C = AB$ de deux matrices $A \in \mathbb{R}^{n \times r}$ et $B \in \mathbb{R}^{r \times m}$

$$c_{ij} = \sum_{k=1}^{r} a_{ik} \, b_{kj} \, .$$

   Pour $n = 10$, $m = 8$ et $r = 5$ générer les matrices A et B avec la commande `rand` et vérifier le résultat de votre procédure.

2.9 mlab15

On considère la matrice tridiagonale d'ordre n définie par

$$A_n = \begin{bmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 2 & -1 \\ & & & -1 & 2 \end{bmatrix}$$

1. Que fait la séquence d'instructions suivante ?

```
n = 7;
S = [eye(n) zeros(n,1)];
S = S(:, 2: n + 1);
A = 2 * eye(n) - S - S'
```

2. Même question avec la séquence d'instructions

```
D = diag(ones(n,1));
S = diag(ones(n-1,1),1);
A = 2 * D - S - S'
```

3. Utiliser une des procédures décrites ci-dessus pour construire la matrice A_n pour n quelconque. Résoudre ensuite un système linéaire $Ax = b$ à l'aide de l'opérateur \, où b est un vecteur colonne unitaire de dimension n. Mesurer le temps de calcul pour $n = 100, 200, 400, 800$ à l'aide des commandes `tic` et `toc`.

2.10 mlab02

Les observations de trois variables y, x_1 et x_2 sont données dans la matrice X_o ci-après :

$X_o =$

y	x_1	x_2
0	−1	0
NaN	2	2
2	0	1
1	1	0
4	1	NaN
2	2	1
6	−1	3
−1	0	−1
1	0	0

Afin d'estimer les paramètres du modèle

$$y = \beta_o + \beta_1 x_1 + \beta_2 x_2 + u$$

il faut éliminer les lignes de X_o qui contiennent des observations manquantes (NaN). D'autre part, on soupçonne la présence d'observations aberrantes. On considèrera une observation d'une variable z comme aberrante, si sa valeur z_i n'est pas comprise dans l'intervalle $[\mu_z - 2\sigma_z \ ; \ \mu_z + 2\sigma_z]$. A l'aide de MATLAB :

1. Ecrire une procédure (fichier .m) pour construire une nouvelle matrice X, à partir de X_o, en épurant toutes les lignes qui contiennent soit une observation manquante (NaN), soit une observation aberrante telle que définie ci-dessus.
2. Ecrire une procédure pour estimer les paramètres d'un modèle linéaire quelconque $y = X\beta + u$, par la méthode des moindres carrés ordinaires. Imprimer le vecteur des paramètres estimés et la valeur des variables de Student qui lui sont associées.

2.11 mlab02a

Les observations de trois variables y, x_1 et x_2 sont données dans la matrice X_o ci-après :

$X_o =$

y	x_1	x_2
0	−1	0
NaN	2	2
2	0	1
1	1	0
4	1	NaN
2	2	1
6	−1	3
−1	0	−1
1	0	0

Afin d'estimer les paramètres du modèle

$$y = \beta_o + \beta_1 x_1 + \beta_2 x_2 + u$$

il faut éliminer les lignes de X_o qui contiennent des observations manquantes (NaN). D'autre part, on soupçonne la présence d'observations aberrantes. On considèrera une observation d'une variable z comme aberrante, si sa valeur z_i n'est pas comprise dans l'intervalle $[\mu_z - 2\sigma_z \ ; \ \mu_z + 2\sigma_z]$ avec μ_z la moyenne et σ_z l'écart type de la variable z. A l'aide de MATLAB ecrire une procédure (fichier .m) pour construire une nouvelle matrice X, à partir de X_o, en épurant toutes les lignes qui contiennent soit une observation manquante (NaN), soit une observation aberrante telle que définie ci-dessus.

2.12 mlab02b

Les observations de trois variables y, x_1 et x_2 sont données dans la matrice X_0 ci-après :

$$X_0 = \begin{array}{|ccc|}\hline y & x_1 & x_2 \\ \hline 0 & -1 & 0 \\ \text{NaN} & 2 & 2 \\ 2 & 0 & 1 \\ 1 & 1 & 0 \\ 4 & 1 & \text{NaN} \\ 2 & 2 & 1 \\ 6 & -1 & 3 \\ -1 & 0 & -1 \\ 1 & 0 & 0 \\ \hline \end{array}$$

Afin d'estimer les paramètres du modèle

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + u$$

il faut éliminer les lignes de X_0 qui contiennent des observations manquantes (NaN). D'autre part, on soupçonne la présence d'observations aberrantes. On considèrera une observation d'une variable z comme aberrante, si sa valeur z_i n'est pas comprise dans l'intervalle $[\mu_z - 2\sigma_z \ ; \ \mu_z + 2\sigma_z]$ avec μ_z la moyenne et σ_z l'écart type de la variable z.

Avev MATLAB ecrire une procédure (fichier .m) pour construire une nouvelle matrice X, à partir de X_0, en épurant toutes les lignes qui contiennent soit une observation manquante (NaN), soit une observation aberrante telle que définie ci-dessus.

Facultatif : Ecrire une procédure pour estimer les paramètres d'un modèle linéaire quelconque $y = X\beta + u$, par la méthode des moindres carrés ordinaires et l'appliquer au cas ci-dessus. Imprimer le vecteur des paramètres estimés et la valeur des variables de Student qui lui sont associées.

2.13 mlab02c

Les observations de trois variables y, x_1 et x_2 sont données dans la matrice X_0 ci-après :

$$X_0 = \begin{array}{|ccc|}\hline y & x_1 & x_2 \\ \hline 0 & -1 & 0 \\ \texttt{NaN} & 2 & 2 \\ 2 & 0 & 1 \\ 1 & 1 & 0 \\ 4 & 1 & \texttt{NaN} \\ 2 & 2 & 1 \\ 9 & -1 & -4 \\ -1 & 0 & -1 \\ 1 & 0 & 0 \\ \hline \end{array}$$

Afin d'estimer les paramètres du modèle

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \varepsilon$$

il faut éliminer les lignes de X_0 contenant des observations manquantes (NaN). D'autre part, on soupçonne la présence d'observations aberrantes. On considérera qu'une observation d'une variable z est aberrante, lorsque sa valeur z_i n'appartient pas à l'intervalle $[\mu_z - 2\sigma_z; \mu_z + 2\sigma_z]$, où μ_z désigne la moyenne des observations de z et σ_z l'écart-type de ces mêmes observations.

1. Utiliser des fonction MATLAB telles que `any` et `isnan` afin de trouver les lignes contenant au moins une observation manquante.
   Eliminer ces lignes de la matrice X_0.
2. Dans le but d'éliminer les lignes ayant des éléments considérés comme aberrants, on propose de suivre les étapes suivantes :
   (a) Calculer une matrice de mêmes dimensions que X_0 contenant les écarts des observations par rapport à la moyenne de leur colonne.
   (*Indication* : On peut utiliser d'une part la fonction `mean` sur X_0 et d'autre part le vecteur $[1\ 1\ \cdots\ 1]'$ que l'on construit grâce à `ones(n,1)` où `n` est le nombre de lignes du vecteur. Lire le help concernant ces fonctions.)
   (b) Calculer une matrice 0-1 de mêmes dimensions que X_0 qui contient des 1 lorsque l'écart calculé au point précédent est supérieur (en valeur absolue) à deux fois l'écart-type de la colonne de X_0.
   (*Indication* : La fonction `std` de MATLAB calcule les écart-types. Consulter le help de `std`. Utiliser une comparaison de deux matrices afin d'obtenir la matrice 0-1 voulue.)
   (c) Eliminer, similairement au point 1, les lignes contenant au moins un élément ayant un écart considéré trop grand.

2.14 mlab02d

Les observations de trois variables y, x_1 et x_2 sont données dans la matrice X_0 ci-après :

$$X_0 = \begin{array}{|ccc|}\hline y & x_1 & x_2 \\ \hline 0 & -1 & 0 \\ \texttt{NaN} & 2 & 2 \\ 2 & 0 & 1 \\ 1 & 1 & 0 \\ 4 & 1 & \texttt{NaN} \\ 2 & 2 & 1 \\ 9 & -1 & -4 \\ -1 & 0 & -1 \\ 1 & 0 & 0 \\ \hline \end{array}$$

Afin d'estimer les paramètres du modèle

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \varepsilon$$

il faut éliminer les lignes de X_0 contenant des observations manquantes (NaN). D'autre part, on soupçonne la présence d'observations aberrantes. On considérera qu'une observation d'une variable z est aberrante, lorsque sa valeur z_i n'appartient pas à l'intervalle $[\mu_z - 2\sigma_z; \mu_z + 2\sigma_z]$, où μ_z désigne la moyenne des observations de z et σ_z l'écart-type de ces mêmes observations.

1. Utiliser des fonctions MATLAB telles que `find` et `isnan` afin de trouver les lignes contenant au moins une observation manquante.
   Eliminer ces lignes de la matrice X_0.
2. Dans le but d'éliminer les lignes ayant des éléments considérés comme aberrants, on propose de suivre les étapes suivantes :
   (a) Calculer une matrice de mêmes dimensions que X_0 contenant les écarts des observations par rapport à la moyenne de leur colonne.

(*Indication* : On peut utiliser d'une part la fonction `mean` sur X_0 et d'autre part le vecteur $[1\ 1\ \cdots\ 1]'$ que l'on construit grâce à l'instruction `ones(n,1)` où `n` est le nombre de lignes du vecteur. Lire le help concernant ces fonctions.)

(b) Calculer une matrice 0-1 de mêmes dimensions que X_0 qui contient des 1 lorsque l'écart calculé au point précédent est supérieur (en valeur absolue) à deux fois l'écart-type de la colonne de X_0.
(*Indication* : La fonction `std` de MATLAB calcule les écart-types. Consulter le help de `std`. Utiliser une comparaison de deux matrices afin d'obtenir la matrice 0-1 voulue.)

(c) Eliminer, similairement au point 1, les lignes contenant au moins un élément ayant un écart considéré trop grand.

3. **Facultatif** : Ecrire une procédure pour estimer les paramètres d'un modèle linéaire quelconque $y = X\beta + u$, par la méthode des moindres carrés ordinaires et l'appliquer au cas ci-dessus. Imprimer le vecteur des paramètres estimés et les valeurs de la statistique de Student qui leurs sont associées.

2.15 prog00

Ecrire une fonction Matlab qui évalue la fonction suivante

$$y = \cos(x^x) - \sin(e^x)\,. \tag{3}$$

On appellera cette fonction `ex9f` et elle comportera les arguments suivants :

```
y = ex9f(x)
```

Programmer cette fonction de sorte que y soit aussi évalué si x est un vecteur.

A l'aide de votre fonction, générer les valeurs de y qui correspondentaux valeurs de $0.5 \leq x \leq 2.5$. Les valeurs de x (vecteur x) pourront être générées avec la commande `linspace` de Matlab.

Faire un graphique de la fonction.

2.16 prog07

1. On considère les instructions Matlab

```
x = ceil(8*rand(1,10));
p = (x > 4);
```

qui produisent le vecteur `p` défini comme

$$p(i) = \begin{array}{ll} 1 & \text{si} \quad x(i) > 4 \\ 0 & \text{sinon} \end{array}\,.$$

A l'aide d'une boucle `for` programmer une procédure qui construit le vecteur `p`. Vérifier qu'elle fournit le même résultat que `p = (x > 4)`.

2. Les instructions Matlab

```
x = ceil(8*rand(1,10));
p = find(x > 4);
```

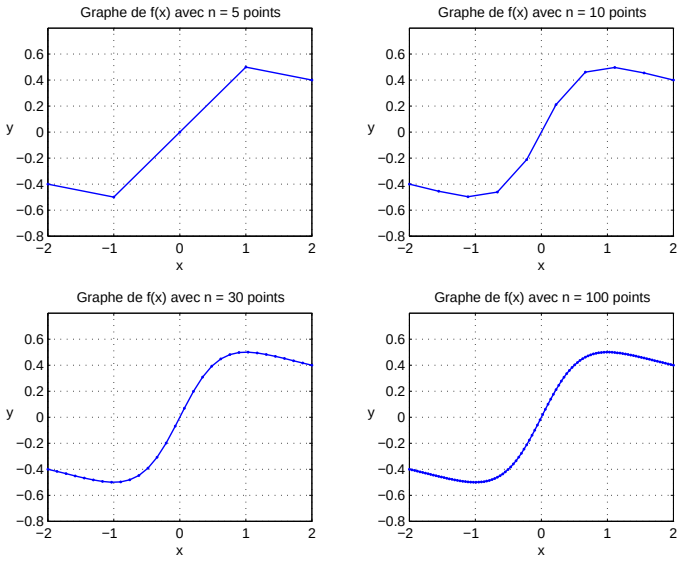
produisent le vecteur `p` des indices des éléments du vecteur `x` qui sont supérieurs à 4. A l'aide d'une boucle `for` programmer une procédure qui construit le vecteur `p`. Vérifier qu'elle fournit le même résultat que `find`.

2.17 prog08

Boucles `for` et vectorisation :
- Créer une matrice $A = [a_{ij}]$ de dimension 10×10 où $a_{ij} = \sin(i)\cos(j)$ à l'aide de boucles `for`.
- Comment peut-on obtenir le même résultat sans utiliser de boucle `for` ?
- Avec la commande `mesh` produire un graphique de la surface $z = \sin(x)\cos(y)$ pour $x \in [1, 10]$ et $y \in [1, 10]$.

Opérateurs logiques :
- Créer un vecteur ligne x de longueur 7 dont les éléments sont des nombres tirés au hazard de la distribution normale standard.
- Trouver les éléments de x qui sont supérieurs à 1 ou inférieurs à -0.2.
- Trouver les éléments de x qui sont supérieurs à 0 et inférieurs à 2.
- Quelle est la position des éléments supériurs à 3 dans le vecteur x ?

Graphisme :
Dessiner le graphe de la fonction $f(x) = \frac{x}{1+x^2}$ pour 5, 10, 30 et 100 valeurs équidistantes de x comprises dans l'intervalle $[-2, 2]$ en reproduisant la figure suivante.



2.18 prog09

Programmer la fonction `BS` ci-après, qui calcule le prix d'un *call* Européen :

```
function call = BS(S,E,r,T,sigma)
% Call Européen avec Black-Scholes
%
d1 = (log(S./E) + (r + sigma.^2 /2) .* T) ...
     ./ (sigma .* sqrt(T));
```

```
d2 = d1 - sigma .* sqrt(T);
Ee = E .* exp(-r .* T);
call = S .* normcdf(d1) - Ee .* normcdf(d2);
```

1. Faire un graphique du prix du *call* en fonction de r, avec $r \in [0.001, 0.10]$.
2. Faire un graphique du prix du *call* en fonction de σ, avec $\sigma \in [0.001, 0.50]$.
3. Faire un graphique du prix du *call* en fonction de r et σ, avec $r \in [0.001, 0.10]$ et $\sigma \in [0.001, 0.50]$.
4. Faire un graphique du prix et des courbes de niveau du *call* en fonction de r et σ, avec $r \in [0.001, 0.10]$ et $\sigma \in [0.001, 0.50]$.

2.19 evalexp

On sait que l'exponentielle est définie comme

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!}$$

Ecrire un programme qui calcule e^x suivant ce schéma. Proposer un critère d'arrêt pour la somme.

Compter le nombre d'opérations élémentaires que nécessite votre algorithme en vous servant de la fonction Matlab `flops`.

Expérimenter votre algorithme pour $x = 13.7$. (Il est possible de ne pas dépasser les 330 opérations élémentaires pour cet exemple en choisissant comme critère d'arrêt $\frac{x^n}{n!} < \epsilon_{\text{mach}}$).

2.20 Horner

On appelle polynôme, ou fonction rationnelle entière, une fonction de la forme

$$y = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n \tag{4}$$

où les a_k, $k = 0, 1, \ldots, n$ sont des nombres constants donnés, appelés coefficients, et où n est un entier positif que l'on appelle degré du polynôme.

En se servant du symbole de sommation, l'expression (4) peut aussi être écrite

$$y = \sum_{k=0}^{n} a_k \, x^k \,. \tag{5}$$

1. Programmer une procédure qui évalue un polynôme de degré n. Exécuter la procédure pour $n = 5$, $x = 2$ et $a = \begin{array}{cccccc} 3 & 4 & 2 & 5 & 6 & 5 \end{array}$.

L'évaluation de l'expression (4) nécessite n additions, n multiplications et $n - 1$ calculs de puissance.

2. En tenant compte que les puissances successives de x peuvent être obtenues par des multiplications

$$\begin{aligned} x^2 &= x\,x \\ x^3 &= x^2 x \\ x^4 &= x^3 x \\ &\vdots \\ x^n &= x^{n-1} x \end{aligned} \tag{6}$$

réécrire une procédure pour évaluer (4), dans laquelle les puissances sont calculées au moyen de la récurrence (6). Avec cette procédure, quel est le nombre de multiplications nécessaires pour évaluer un polynôme de degré n.

Considérons maintenant un polynôme de degré 4 :

$$y = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + a_4 x^4 \,.$$

Il est alors possible de regrouper les termes du polynôme comme suit :

$$y = a_0 + x(a_1 + x(a_2 + x(a_3 + x a_4))) \,. \tag{7}$$

L'évaluation de l'expression (7) ne nécessite que n additions et n multiplications[1].

3. Programmer une procédure qui évalue un polynôme de degré n selon la méthode de Horner illustrée en (7). Vérifier avec la fonction Matlab `flops` que la procédure nécessite $2n$ opérations élémentaires.

2.21 LotkaVolterra

On considère une population de prédateurs P et une population de proies H. L'évolution dans le temps de ces deux populations peut être décrite par les deux équations[2]

$$H_{t+\triangle_t} = H_t + (\alpha_1 H_t - \beta P_t H_t)\, \triangle_t \tag{8}$$

$$P_{t+\triangle_t} = P_t + (-\alpha_2 P_t + \beta P_t H_t)\, \triangle_t \tag{9}$$

ou α_1 est le taux de croissance de la population des proies, α_2 est le taux de déclin de la population des prédateurs et β est un paramètre qui quantifie l'interaction entre les deux populations. La variation des effectifs entre l'instant t et $t+\triangle_t$ s'interpréte :

- Pour la population des proies : croissance de $\alpha_1 H_t \triangle_t$ et diminution de $\beta P_t H_t \triangle_t$ du à l'interaction des prédateurs ;
- Pour la population des prédateurs : déclin de $\alpha_2 P_t \triangle_t$ (en l'absence de proies) et augmentation de $\beta P_t H_t \triangle_t$ du à la présence des proies.

Pour un effectif H_{t_0} et P_{t_0} au temps t_0 donné on désire calculer l'évolution de ces deux populations pour une durée T. La variable t est discrétisée en M valeurs équidistantes

$$t_i = i \triangle_t, \quad i = 0, 1, \ldots, M, \quad \triangle_t = T/M$$

Pour la programmation on fera correspondre à la valeur de P_{t_i} l'élément `P(i)` du vecteur `P` de longueur $M+1$. Comme Matlab ne permet pas que l'indice d'un vecteur prenne la valeur zéro on initialise une variable `f7 = 1` et on défini un indice rélatif

```
P(f7+i)
```

et ainsi l'indice i pourra prende les $M+1$ valeurs $i = 0, 1, \ldots, M$.

1. Avec Matlab programmer la procédure (ésquissé ci-après) qui calcule les M valeurs P_{t_i}, H_{t_i}, $i = 1, 2, \ldots, M$. Pour l'initialisation on utilisera les valeurs suivantes : $\alpha_1 = 0.03$, $\alpha_2 = 0.30$, $\beta = 0.000\,3$, $T = 100$, $M = 1\,000$, $P_{t_0} = 100$ et $H_{t_0} = 500$.

[1] Cette façon d'évaluer un polynôme est due à Horner.
[2] Il s'agit d'une version discrète du modèle de Lotka-Volterra.

1: Initialiser les paramètres α_1, α_2 et β
2: Initialiser P_{t_0} et H_{t_0}
3: Initialiser T et M
4: Calculer $\Delta_t = T/M$
5: **pour** $i = 0$ à $M-1$ **faire**
6: Calculer $H_{t_{i+1}}$ en utilisant l'équation (8)
7: Calculer $P_{t_{i+1}}$ en utilisant l'équation (9)
8: **finfaire**

2. Dans un même graphique traçer l'évolution de l'effectif des deux populations. Exécuter le graphique suivant les spécifications ci-après :
   - temps en absisse et effectifs en ordonnée ;
   - courbe effectif des proies en couleur verte ;
   - courbe effectif des prédateurs en couleur rouge ;
   - légende expliquant les courbes tracées ;
   - titre comportant le texte "Evolution des effectifs".
3. Traçer le diagramme de phase, c'est-à-dire le graphique des couples (P_{t_i}, H_{t_i}), $i = 0, 1, \ldots, M$. Marquer le premier point avec le symbole $\circ$ et le dernier avec le symbole $\star$. Anoter les axes et le graphique.
4. Présenter la procédure programmé au point 1 sous la forme d'une fonction

   ```
   [P,H] = LotkaVolterra(P0,H0)
   ```

   ou les arguments `P0` et `H0` représentent les valeurs initiales et `P` et `H` les trajectoires calculées.
5. Au moyen d'une boucle calculer les trajectoires qui correspondnet aux trois couples de valeurs initiales

   ```
   P0 = 100;  H0 =  500;
   P0 = 100;  H0 = 1000;
   P0 = 100;  H0 = 1500;
   ```

   et traçer dans un graphique les trois paires de courbes d'effectifs et dans un autre graphique les trois diagrammes de phase. Commenter.

2.22 BreakEven

Le coût de production total pour un bien x est donné par l'èquation

$$f(x) = 1000 + 350\log(x) + (x-5)^{1.15} + (0.05\,x)^3\,,$$

le prix de vente pour une unité de x est de 37.5 unitées monétaires ce qui définit la recette totale comme

$$g(x) = 37.5\,x$$

d'où on déduit la fonction profit

$$h(x) = g(x) - f(x)\,.$$

1. Chercher le *break-even point*, c'est-à-dire le niveau de production pour lequel la recette égalise les coûts. (Indication : Il s'agit de chercher la valeur de x pour laquelle $h(x) = 0$).
2. On désire déterminer le niveau de production pour lequel le profit est maximum. Avec Maple dériver la fonction $h(x)$.
3. En vous servant de $h'(x)$ obtenu avec Maple calculer avec Matlab la solution de $h'(x) = 0$.

2.23 BreakEven2

La fonction de coût total d'une entreprise, pour la production d'un bien x est donné par l'équation

$$f(x) = 200 + 210\,\log(x^2) + (x+4)^{0.9} + \left(\frac{x}{40}\right)^5\,,$$

le prix de vente p pour une unité de x est de 50 unités monétaires et la recette totale est définie comme

$$g(x) = p\,x$$

d'où on déduit la fonction profit

$$h(x) = g(x) - f(x)\,.$$

1. Chercher les niveaux de production x_1 et x_2 pour lesquels la recette égalise les coûts. Il s'agit de chercher les valeurs de x pour lesquelles $h(x) = 0$.
   (a) Dans un premier temps vous pouvez identifier les intervalles qui contiennent un zero en faisant un graphique de la fonction (Indication : Produire le graphique pour $x \in [4, 300]$). Ensuite utiliser la méthode de la bissection avec une tolérance $tol = 10^{-6}$.
   (b) Une autre demarche consiste à determiner les intervalles qui contiennent un zero avec la méthode dite *bracketing* et d'appliquer ensuite la méthode de la bissection. (Indication : Pour la méthode de *bracketing* considerer 40 intervalles équidistants entre 4 et 300.)

   Comparer les résultats obtenus en *a)* et *b)*.
2. On désire déterminer le niveau de production x^* pour lequel le profit est maximum. Avec Maple dériver la fonction $h(x)$.
3. En vous servant de $h'(x)$ obtenu avec Maple et en utilisant la même méthode qu'au point 1(a), chercher avec Matlab la solution de $h'(x) = 0$ et ainsi determiner le niveau de production x^* pour lequel la fonction de profit est maximisée. Calculer le montant du profit maximum $h(x^*)$ réalisé par l'entreprise.
4. Représenter dans un même graphique la fonction de profit $h(x)$ et les couples $(x_1, h(x_1))$, $(x_2, h(x_2))$ et $(x^*, h(x^*))$.

2.24 Determinant

Le determinant d'une matrice A d'ordre n peut s'écrire comme

$$|A| = \sum_{j=1}^{n} (-1)^{i+j}\, a_{ij}\, |A^{ij}| \tag{10}$$

ou i est l'indice d'une ligne quelconque de A et A^{ij} est la sous-matrice de A obtenue en supprimant la ligne i et la colonne j de A.

A titre d'exemple considérons la matrice

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 0 \end{bmatrix}$$

et développons l'expression (10) pour $i = 3$:

$$|A| = (-1)^{3+1}\,7\,\begin{vmatrix} 2 & 3 \\ 5 & 6 \end{vmatrix} + (-1)^{3+2}\,8\,\begin{vmatrix} 1 & 3 \\ 4 & 6 \end{vmatrix} + (-1)^{3+3}\,0\,\begin{vmatrix} 1 & 2 \\ 4 & 5 \end{vmatrix}$$

Pour l'application qui suit on construira la matrice A avec la procédure aléatoire suivante

```
n = 3;
A = fix(10*rand(n));
```

1. Avec Matlab programmer le calcul du déteminant en appliquant la formule (10). Pour le programme on suggère la structure suivante :

Algorithm 1 (Calcul du déterminant)

1: Initialiser i et n
2: $D = 0$
3: **pour** $j = 1$ à n **faire**
4: Definir la sousmatrice A^{ij}
5: $D = D + (-1)^{i+j}\, a_{ij}\, |A^{ij}|$
6: **finfaire**

Pour le calcul de la sousmatrice A^{ij} à la ligne 4 de l'algorithme 1 on rappelle que les vecteur d'indices des lignes et des colonnes peuvent êtres définies comme

```
lin = [1:i-1 i+1:n];
col = [1:j-1 j+1:n];
```

et la sousmatrice A^{ij} s'obtient après avec la commande `Aij = A(lin,col)`. Pour le calcul du déterminant de A^{ij} à la ligne 5 de l'algorithme on utilisera la commande Matlab `det`.

2. Générer une matrices d'ordre 3 et tester le programme (vérifier le résultat à l'aide de la commande Matlab pour le calcul du déterminant).
3. Présenter le programme élaboré au point 1 sous la forme d'une fonction Matlab

```
D = Determinant(A)
```

Pour $n = 3, 4, 5$ générer aléatoirement A et tester la fonction `Determinant`.

pour $n = 3,\ 4,\ 5$ **faire**
Construire A
`D = Determinant(A)`
Vérifier le résultat
finfaire

2.25 DerivNum

On considère la fonction

$$y = \sin(x) \tag{11}$$

et sa dérivée $y' = \cos(x)$. On vérifie que pour $x = 1$ on a $y' = 0.5403\ldots$.

Écrire un programme qui calcule une approximation numérique de la dérivée de (11) au point $x = 1$, en utilisant la définition

$$y'(1) = \lim_{h \to 0} \frac{\sin(1+h) - \sin(1)}{h}\,.$$

Calculer cette approximation pour $h = 2^{-i}$, $i = 1, \ldots, 52$.

Faire un graphique qui représente la valeur de l'approximation de la dérivée en fonction de $\log(h)$.

Calculer, pour les mêmes valeurs de h, une approximation de la dérivée par différence centré et reporter le résultat dans le même graphique.

2.26 Bracketing

Avec Matlab réaliser l'algorithme qui suit sous la forme d'une fonction

```
ab = bracketing(f,xmin,xmax,n)
```

ou `ab` est une matrice avec 2 colonnes contenant respectivement le debut et la fin d'un intervalle susceptible de contenir un zero, et le nombre de lignes correspond au nombre d'intervalles identifiés. `f` est la fonction à analyser, `xmin` et `xmax` définissent le domaine de recherche et `n` est le nombre d'intervalles à analyser.

Algorithm 2 Étant donné $f(x)$, $x_{\min}$, $x_{\max}$ et n l'algorithme identifie les intervalles susceptibles de contenir des zeros.

$dx = (x_{\max} - x_{\min})/n$
$a = x_{\min}$
$i = 0$
tant que $i < n$ **faire**
$i = i + 1$
$b = a + dx$
si $sign(f(a)) \neq sign(f(b))$ **alors**
$[a, b]$ peut contenir un zéro, imprimer a et b
finsi
$a = b$
finfaire

Application : $f(x) = \cos(1/x^2)$, $x \in [0.3,\ 0.9]$ et $n = 25$.

2.27 fact

Soit $n! = 1 \cdot 2 \cdot \ldots \cdot n$ la notation de factorielle n. Il est immédiat que factorielle n peut être défini comme

$$\text{factorielle } n = n \cdot \text{ factorielle } (n-1)$$

ce qui suggère que l'on peut calculer $n!$ avec un algorithme récursif.

Avec MATLAB, écrire une fonction récursive `r = fact(n)` qui calcule $n!$. *(10 points)*
(Rappel : Par convention on a $0! = 1$).

2.28 MaxMin

On considère le problème de la recherche du maximum et du minimum des éléments d'un vecteur s. Afin de simplifier le problème, on admet que le nombre d'éléments du vecteur est une puissance de 2, c'est-à-dire qu'il peut s'exprimer sous la forme 2^k avec $k > 1$. Ecrire, puis implémenter l'algorithme suivant :

Algorithm 3 Recherche des éléments maximum et minimum d'un vecteur. Input : Vecteur s avec n elements, $n = 2^k$, $k > 1$. Output : Elément maximum et élément minimum du vecteur s. Application récursive de la procédure `maxmin` au vecteur s. Le résultat est constitué par le couple (a, b) avec a l'élément maximum et b l'élément minimum du vecteur s.

```
(a, b) = maxmin(s)
si card(s) = 2 alors
    a = max(s_1, s_2)
    b = min(s_1, s_2)
sinon
    partitionner s en s^1 et s^2 de même cardinalité
    (a_1, b_1) = maxmin(s^1)
    (a_2, b_2) = maxmin(s^2)
    a = max(a_1, a_2)
    b = min(b_1, b_2)
finsi
```

Application : Générer un vecteur $s_1, s_2, \ldots, s_{2^k}$ avec $k = 8$ et vérifier le fonctionnement de l'algorithme.

Une recherche du maximum et du minimum par simple comparaison nécessite $n - 1$ opérations de comparaison pour le maximum et ensuite $n - 2$ opérations de comparaison pour le minimum, donc au total $2n - 3$ opérations de comparaison.

Etablir la fonction $C(n)$ qui donne le nombre de comparaisons effectuées par l'algorithme 3. Quelle est la complexité de l'algorithme 3. Comparer la performance des deux approches. Vérifier le resultat empiriquement.

3 Précision finie et stabilité numérique

3.1 apf00

Soit F l'ensemble des nombres en virgule flottante que l'on peut representer avec des mots de 8 bits dont $t = 4$ bits sont reservées pour la mantisse. Donner :
- le cardinal de F,
- les nombres m et M,
- la précision machine *eps*
- le nombre de digits significatifs.

3.2 nstab01

Avec Matlab générer un vecteur y contenant 8000 observations d'une variable aléatoire normale. L'élément générique de y est défini comme :

$$y_i = z_i + 10^8$$

où z_i est un nombre aléatoire tiré d'une loi normale centrée réduite. On choisira 15207 comme valeur initiale pour le générateur des nombres au hasard, en exécutant la commande `randn('seed',15207)`.

1. Calculer une estimation de la variance de y en utilisant la formule suivante (one-pass algorithm) :
$$\hat{\sigma}_y^2 = \frac{1}{n-1}\left\{ \sum_{i=1}^{n} y_i^2 - \frac{1}{n}\left(\sum_{i=1}^{n} y_i\right)^2 \right\}$$
Commenter le résultat.
2. Calculer la moyenne $\bar{y}$ du vecteur y
$$\bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$$
puis effectuer la somme
$$e = \sum_{i=1}^{n} (y_i - \bar{y})$$
Commenter e.
3. On sait que la transformation $x_i = y_i - d$ avec d constant, conduit à un vecteur x de même variance que y. Calculer une estimation de la variance de x par la méthode définie sous 1 pour différentes valeurs de d.
Commenter et donner un critère de choix pour d.
4. Une méthode alternative pour calculer $\hat{\sigma}_y^2$ consiste à calculer d'abord $\bar{y}$, puis l'expression
$$\frac{1}{n-1}\sum_{i=1}^{n} (y_i - \bar{y})^2 \ .$$
Cette méthode est appelée "two-pass algorithm".
Calculer $\hat{\sigma}_y^2$ avec cette méthode.

Commenter les avantages et inconvénients de chacune de ces méthodes.

3.3 nstab02

On considère le polynôme $p(x) = (x-1)^6$. En développant on peut écrire ce même polynôme sous la forme

$$f(x) = x^6 - 6x^5 + 15x^4 - 20x^3 + 15x^2 - 6x + 1$$

1. Avec Matlab évaluer $p(x)$ et $f(x)$ pour x allant de .998 à 1.002 en choisissant des pas de .0001.
2. Faire un graphique de $f(x)$ et $p(x)$. Commenter.

3.4 nstab03

Soit la suite numérique $u_1 + u_2 + \ldots + u_n + \ldots$ et sa somme partielle

$$S_n = u_1 + u_2 + \ldots + u_n$$

avec le terme général $u_i = (\frac{1}{3})^i$. On peut alors écrire la relation suivante :

$$R_n = \frac{1 - 2S_n}{u_n} = 1 \quad .$$

Pour n allant de 1 à 33 calculer u_n, S_n et R_n. On se servira de la récurrence :

$$\begin{aligned} u_o &= 1 \\ S_o &= 0 \\ u_n &= \tfrac{1}{3}\, u_{n-1} \\ S_n &= S_{n-1} + u_n \quad . \end{aligned}$$

Representer sur un même graphique R_n, S_n et u_n. Commenter les résultats.

3.5 nstab06

On désire calculer les 37 premiers termes de la récurrence

$$x_{i+2} = -\frac{13}{6}x_{i+1} + \frac{5}{2}x_i \qquad i = 1, 2, \ldots \tag{12}$$

avec $x_1 = 30$ et $x_2 = 25$.

On peut facilement vérifier que (1) s'exprime également comme

$$x_i = 36 \quad \frac{5}{6}^{\ i} \qquad i = 1, 2, \ldots . \tag{13}$$

Calculer avec MATLAB :

a) la suite $\{x_i\}_{i=3,\ldots,37}$ à partir de l'expression (1) ;
b) la suite $\{x_i\}_{i=1,\ldots,37}$ à partir de l'expression (2).

Représenter les résultats graphiquement et commenter.

Facultatif : Montrer comment l'on obtient l'expression (2) à partir de l'expression (1).

3.6 nstab07

On désire calculer les puissances entières successives du nombre d'or $\phi = (\sqrt{5} - 1)/2$. On peut facilement vérifier que les puissances ϕ^i, $i > 0$ satisfont la relation de récurrence

$$\phi^{i+1} = \phi^{i-1} - \phi^i \ . \tag{14}$$

Ainsi, en connaissant les valeurs de $\phi^0 = 1$ et de $\phi^1 = \phi$, on peut calculer les puissances suivantes de ϕ en utilisant la relation (14).

Calculer la suite $\{\phi^i\}_{i=2,3,\ldots,20}$ avec MATLAB par
(a) la relation de recurrence (14),
(b) la relation $\phi^{i+1} = \phi^i \cdot \phi$.
Représenter graphiquement les erreurs entre les deux méthodes.

3.7 nstab08

Soit le système linéaire $Ax = b$ suivant :

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 3 & 4 \\ 2 & 2 & 1 & 1 \\ 4+e & 5 & 4 & 5 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 10 \\ 30 \\ 13 \\ 46+e \end{bmatrix} \quad .$$

On vérifie que pour tout $e \neq 0$, $x_1 = 1$, $x_2 = 2$, $x_3 = 3$ et $x_4 = 4$ est la solution exacte de ce système d'équations. Lorsque $e = 0$, le système est singulier et possède une infinité de solutions comprenant la solution $x_1 = 1$, $x_2 = 2$, $x_3 = 3$, $x_4 = 4$.

Pour un problème bien conditionné, MATLAB produit des résultats avec approximativement 15 digits significatifs.

Le but de cet exercice est d'illustrer la relation qui existe entre la condition d'une matrice A et la précision de la solution x du système $Ax = b$. A cette fin, on posera $e = 10^{-2(k-1)}$ et on calculera pour $k = 1, 2, \ldots, 8$

1. la condition de la matrice A,
2. la solution du système $Ax = b$ en `format long`,
3. le nombre de digits non significatifs (c'est-à-dire le nombre de chiffres qui différent de la vraie solution donnée).

D'après ces résultats, établir quelle est la relation approximative qui existe entre la condition d'une matrice et le nombre de digits non significatifs dans la solution calculée.

3.8 ApproxNumCos

On considère l'expression

$$\cos(x) \approx \sum_{k=0}^{n} (-1)^k \frac{x^{2k}}{(2k)!} \tag{15}$$

qui pour n fini constitue une approximation de $\cos(x)$.

1. Écrire un script (il n'est pas nécessaire de le presenter sous forme de fonction) qui évalue l'expression (15).

   Proposer un critère pour le choix de n dans (15). Suggestion : On pourra arrêter la sommation lorsque en arithmétique flottante on vérifie $S_{i+1} = S_i$ où $S_i = \sum_{k=0}^{i} (-1)^k \frac{x^{2k}}{(2k)!}$.
   Exécuter la procédure pour $x = 1.3$, vérifier le résultat en utilisant la fonction Matlab `cos`, imprimer le nombre de flops (pour le comptage on utilisera la fonction `flops` et n le nombre de termes additionnées.
   Rappel : Avec Matlab on peut calculer $n!$ avec la commande `prod(1:n)`.
2. On peut remarquer que le rapport dans l'expression (15) peut être décomposé comme suit :

   $$\frac{x^{2k}}{(2k)!} = 1 \times \frac{x^2}{1 \cdot 2} \times \frac{x^2}{3 \cdot 4} \times \cdots \times \frac{x^2}{(2k-1) \times 2k} \tag{16}$$

   Completer votre script avec une procédure qui tient compte de l'expression (16) pour le calcul de $\frac{x^{2k}}{(2k)!}$.
   Exécuter, toujours pour $x = 1.3$, les deux procédures (celle du point 1. et la nouvelle) et comparer les résultats ainsi que le nombre d'opérations élémentaires (`flops`).
3. Exécuter les procédures pour $x = 50$. Que constat-t-on ? Commenter et expliquer les résultats.
4. On sait que $\cos(x) = \cos(\lfloor x/2\pi \rfloor)$ où $\lfloor x/2\pi \rfloor$ représente le reste de la division entière. Vérifier que les deux procédures fonctionnent correctement si l'on applique ce changement de variable.

4 Complexité des algorithmes

4.1 Strassen

On considère le produit matriciel $C = AB$ avec $A \in \Re^{n \times n}$ et $B \in \Re^{n \times n}$.

1. Quel est le nombre d'opérations élémentaires nécessaires pour effectuer ce produit. Justifier la réponse à partir d'une esquisse de l'algorithme .

En admettant que n est pair, on considère la partition suivante des matrices C, A et B

$$\begin{matrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{matrix} = \begin{matrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{matrix} \quad \begin{matrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{matrix} \tag{17}$$

avec $A_{11} \in \Re^{\frac{n}{2} \times \frac{n}{2}}$ et $B_{11} \in \Re^{\frac{n}{2} \times \frac{n}{2}}$. Si l'on considère le produit par blocs, on a $C_{ij} = A_{i1}B_{1j} + A_{i2}B_{2j}$. On vérifie alors aisément qu'en appliquant la multiplication par blocs, le calcul de C fait intervenir 8 multiplications de matrices et 4 additions de matrices.

2. Quel est le nombre d'opérations élémentaires pour effectuer le produit par blocs défini en (1). Existe-t-il un avantage à recourir au produit par blocs ?

Strassen[3] a montré qu'il est possible d'effectuer le produit par bloc (1) en faisant intervenir seulement 7 produits et 18 additions de matrices en organisant les calculs comme suit :

$$\begin{aligned} P_1 &= (A_{11} + A_{22})(B_{11} + B_{22}) \\ P_2 &= (A_{21} + A_{22})B_{11} \\ P_3 &= A_{11}(B_{12} - B_{22}) \\ P_4 &= A_{22}(B_{21} - B_{11}) \\ P_5 &= (A_{11} + A_{12})B_{22} \\ P_6 &= (A_{21} - A_{11})(B_{11} + B_{12}) \\ P_7 &= (A_{12} - A_{22})(B_{21} + B_{22}) \\ C_{11} &= P_1 + P_4 - P_5 + P_7 \\ C_{12} &= P_3 + P_5 \\ C_{21} &= P_2 + P_4 \\ C_{22} &= P_1 + P_3 - P_2 + P_6 \end{aligned}$$

3. Combien d'opérations élémentaires nécessite la méthode de Strassen ?
4. A partir de quelle valeur de n la méthode de Strassen nécessite-t-elle moins d'opérations élémentaires que le produit défini en (1). (On accepte une solution obtenue à partir d'un graphique).
5. Programmer avec MATLAB la méthode de Strassen sous la forme d'une fonction

   $$C = \texttt{strassen0}(A, B)$$

   et vérifier que le nombre d'opérations élémentaires pour la valeur de n trouvée avant correspond au nombre d'opérations élémentaires pour le produit défini sous 1.

On remarque que le calcul des matrices P_1 à P_7 dans l'algorithme de Strassen fait intervenir un produit de matrices, lequel peut à son tour être évalué moyennant l'algorithme de Strassen. Ainsi, il apparaît que l'algorithme de Strassen a une structure récurrente.

6. Programmer une version récursive de l'algorithme de Strassen. On appellera cette fonction `strassen`. On lui donnera la structure suivante :

   function $C = \textbf{strassen}(A, B, n_{min})$
   if $\textbf{size}(A, 1) < n_{min}$
   $C = AB$
   else
   Appliquer l'algorithme de Strassen
   end

7. Exécuter l'algorithme pour quelques valeurs de $n = 2^d$, $d = 5, \ldots, 7$. Pourquoi a-t-on défini n comme une puissance de 2 ?

[3] Strassen, V. (1969) : 'Gaussian Elimination is not Optimal', *Numer. Math.* 13, 354–356.

Comme on l'a remarqué au point 2., la méthode de Strassen devient avantageuse lorsque les matrices dépassent une certaine taille. Dès lors, il convient d'arrêter la procédure récurrente lorsque le produit implique des sous-matrices de taille inférieure à un certain seuil.

8. Explorer l'algorithme afin de découvrir la valeur optimale à partir de laquelle l'algorithme doit commuter au produit normal.

5 Recherche des zeros d'une fonction

5.1 fzero00

On désire trouver la valeur de ξ qui satisfait l'expression suivante

$$\xi^3 e^{\xi^2/4} \int_{-\infty}^{\xi} e^{-s^2/4} ds = 2(2 - \xi^2)\,. \tag{18}$$

Ce problème est aussi appellé *recherche du zéro d'une fonction.*

1. Ecrire l'équation (18) sous la forme $f(\xi) = 0$ et construire une fonction Matlab qui évalue $f(\xi)$. On pourra structurer la fonction comme suit :

   `function f = fzero00f(xi)`
   Définir la fonction $g(s) = e^{-s^2/4}$ avec `inline` (s peut être un vecteur)
   Calculer $I = \int_{-\infty}^{\xi} e^{-s^2/4} ds$ avec la fonction `quad`
   Calculer $f = \xi^3 e^{\xi^2/4} I - 2(2 - \xi^2)$

2. Faire un graphique de la fonction f pour des valeurs de $\xi \in [-1, 2]$.
3. Rechercher numériquement le zéro de la fonction f dans ce même intervalle (de préférence utiliser l'algorithme de la bissection).

5.2 bissect

Zéro d'une fonction dans $\mathbb{R}$ par bissection.

Soit $f : \mathbb{R} \to \mathbb{R}$ une fonction non linéaire. Le problème considéré est celui de trouver un point x^* dans un intervalle borné tel que $f(x^*) = 0$. Ce point x^* est appelé "zéro de f". Par hypothèse, nous nous restreindrons au cas où la fonction f change de signe en x^* et nous ne considérerons pas le cas où f est tangente à l'axe horizontal.

Un intervalle $[a, b]$ recouvrant le point cherché x^* est appelé *intervalle d'incertitude* et cet intervalle *encadre* x^* lorsque $f(a)f(b) < 0$, c'est-à-dire lorsque f change de signe dans $[a, b]$.

Pour approcher x^* on se contente généralement d'un intervalle d'incertitude suffisamment petit, par exemple $|a - b| < \delta$ avec $\delta = 10^{-5}$.

Une méthode permettant de trouver une approximation numérique de cet intervalle (et donc de x^*) est la *méthode de la bissection.* Cette méthode procède comme suit.

- On se donne un intervalle de départ $[a, b]$ tel que $f(a)f(b) < 0$ et une valeur pour δ.
- On répète les opérations suivantes tant que la condition $|a - b| < \delta$ n'est pas satisfaite :
  - On évalue f en $c = (a + b)/2$.
  - Si $f(c)$ a le même signe que $f(a)$, on remplace a par c.
  - Sinon, on remplace b par c (puisque alors b a le même signe que c).

Lorsque la condition est satisfaite, on possède un intervalle suffisamment petit qui encadre x^*.

On désire programmer à l'aide de Matlab la méthode de la bissection.

1. Créer une fonction Matlab nommée `f` qui évalue
$$f : x \to \cos(x^x) - \sin(e^x) .$$
La figure 1 donne la représentation graphique de f dans l'intervalle $[0.5, 2.5]$.

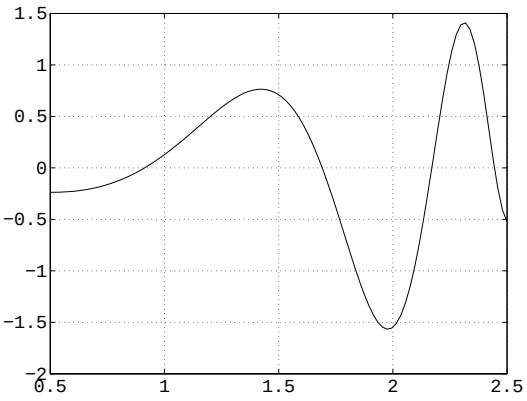


FIG. 1 – Graphe de la fonction $\cos(x^x) - \sin(e^x)$.

2. Créer un script-file nommé `bissect.m` qui implémente la méthode décrite plus haut.
3. Exécuter le script avec $\delta = 10^{-5}$ et différentes valeurs de départ pour a et b, par exemple

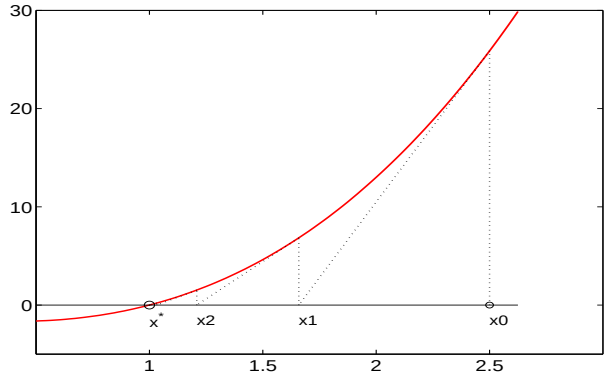
a	0.5	1.5	2.0	2.3
b	1.5	2.0	2.3	2.5

Commenter les résultats obtenus.
4. Afin de compter le nombre d'itérations exécutées (i.e. le nombre de fois que la condition $|a - b| < \delta$ n'est pas satisfaite), on introduit un *compteur*.
Avant la boucle, on initialise une variable à zéro : `iter=0`. Puis, à chaque passage dans la boucle on augmente la valeur de `iter` de 1 : `iter=iter+1`. On peut alors imprimer la valeur de `iter` avec celles de a et b à la fin du programme.
En gardant l'intervalle de départ fixé à $[0.5, 1.5]$, exécuter le script pour différentes valeurs de δ, par exemple $\delta = 10^{-3}, 10^{-4}, 10^{-5}, 10^{-6}, 10^{-7}$.
Comment évolue le nombre d'itérations effectuées en fonction de δ ?

5.3 Newton0

La recherche des zéros d'une fonction peut s'effectuer au moyen de différents algorithmes. Une possibilité est la méthode dite de la bissection.

Une autre méthode, très utilisée, est la méthode de Newton. Pour cette méthode, on ne donne pas un intervalle d'incertitude comme c'est le cas pour la méthode de la bissection, mais un point de départ x_0 dans un voisinage du zéro x^*. Ensuite, on calcule avec un procédé itératif une suite d'approximations $\{x_k\}_{k=1,2,\ldots}$ qui tend vers x^*. La figure qui suit illustre la méthode de Newton :



On arrête le processus lorsqu' une précision fixée δ est atteinte. Cette méthode nécessite en plus que la fonction dont on cherche les zéros soit dérivable et que sa dérivée soit continue. Formellement, la méthode se résume à :

1. Initialiser x_0, $\delta = 10^{-6}$, $k = 0$ et $conv = 0$
2. Tant que $conv$ est faux
3. Calculer $x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$
4. Incrémenter le compteur $k = k + 1$
5. Mettre à jour $conv$ (voir ci-dessous)
6. Fin

La variable $conv$ prend la valeur "vrai" (i.e. $conv \neq 0$) si $|x_{k+1} - x_k| < \delta$, et la valeur "faux" ($conv = 0$) sinon.

1. Pour $x \in [-3, 4]$ faire un dessin de la fonction
$$f(x) = x^3 + 3x^2 - 3x - 1 .$$
2. Programmer l'algorithme de Newton pour trouver les zéros de la fonction proposée (on passera la fonction comme argument).

5.4 USA

On appelle *point fixe* le vecteur x^* qui satisfait le système d'équations
$$g(x^*) = x^*.$$
Considérons la fonction $g : \mathbb{R} \to \mathbb{R}$ (une seule équation et une seule variable)
$$g(x) = ax - bx^2$$
qui est parfois utilisée pour décrire l'évolution d'une population. Le paramètre a représente le taux de reproduction et le paramètre b le taux d'extinction de la population.

1. En choisissant $a = b = 3.5$ évaluer $g(x)$ pour $x \in [0, 1]$. Faire un graphique de $g(x)$ et de la fonction $y = x$.
L'intersection de $g(x)$ avec $y = x$ définit le point fixe de $g(x)$. Relever sa valeur approximative sur le graphique.
Avec la commande MATLAB `hold on` il est possible de tracer successivement des fonctions dans la même fenêtre

graphique. Pour abandonner une fenêtre graphique et créer une nouvelle fenêtre, on précédera la commande `plot` de la commande `hold off`.

2. On désire maintenant étudier la fonction composée $g(g(x))$. Evaluer $g(g(x))$ avec a, b et x, définie ci-dessus, avant et l'ajouter dans la fenêtre graphique créée précédemment.

Khilnani et Tse[4] ont proposé un algorithme, appelé Updated Successive Approximation (USA), pour résoudre numériquement le problème du point fixe. Cet algorithme s'énonce ainsi :

Définir ϵ (critère de convergence)

Définir γ (paramètre de contraction, $0 < \gamma < 1$))

Choisir x_0 et x_1 (valeurs initiales)

Evaluer $g_0 = g(x_0)$ et $g_1 = g(x_1)$

Evaluer $w_0 = x_0 - g_0$ et $w_1 = x_1 - g_1$

`si` $\|w_0\| < \|w_1\|$, `alors`

permuter x_0 avec x_1

`finsi`

`tant que` $\|w_k\| > \epsilon$, `faire`

$\Delta w_k = w_k - w_{k-1}$

$p_k = \frac{(x_k - x_{k-1})' \Delta w_k}{\|w_k\| \, \|\Delta w_k\|} \gamma$

$x_{k+1} = (1 - p_k)x_k + p_k g_k$

`finfaire`

$x^* = x_k$

3. Programmer l'algorithme USA avec MATLAB. Etant donné un vecteur v, on note $\|v\| = \sqrt{v'v}$ la norme 2 du vecteur. Avec MATLAB on la calcule avec la commande `norm(v)`. Notons que dans le cas où v est un scalaire `norm(v) = abs(v)`.
4. Tester l'algorithme programmé en cherchant le point fixe de la fonction donnée plus haut. On choisira 0.5 comme valeur de γ.

On peut aussi envisager la recherche du point fixe comme la recherche du zéro d'une fonction. Soit

$$f(x) = 0 \qquad \text{avec} \qquad f(x) = g(x) - x$$

cette transformation.

5. Appliquer la méthode de Newton pour rechercher le zéro de cette fonction. Comparer et commenter les résultats.
6. Chercher les points fixes de $g(g(x))$ avec l'algorithme USA.
7. On considère n compositions de la fonction $g(x)$

$$\underbrace{g(g(g(\cdots g(x))))}_{n}$$

Ecrire une fonction récursive

$$\texttt{g} = \texttt{compg}(\texttt{a}, \texttt{b}, \texttt{x}, \texttt{n})$$

qui effectue cette évaluation. Les arguments `a` et `b` correspondent aux paramètres de la fonction $g(x)$, `x` étant le vecteur pour lequel on désire évaluer la fonction et `n` représente le nombre de fois la fonction est composée.

Compléter la fenêtre graphique avec l'image de $g(g(g(g(x))))$.

5.5 racine

Afin de calculer numériquement la racine carrée r d'un réel positif x, $r = \sqrt{x}$, on considère la procédure itérative[5]

$$r_k = \frac{1}{2} \quad r_{k-1} + \frac{x}{r_{k-1}} \qquad k = 1, 2, \ldots$$

où l'on peut choisir comme point de depart $r_0 = x$.

Dans les questions qui suivent on proposera différentes alternatives pour programmer le calcul numérique de cette approximation avec Matlab.

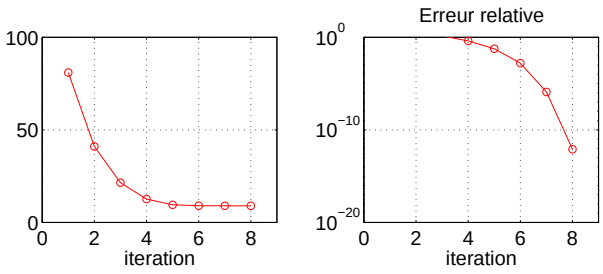
1. Programmer le pseudo-code[6] ci-dessous et l'exécuter pour $x = 81$.

   Initialiser x (pour l'exemple $x = 81$)
   Initialiser un vecteur r de dimension 1×10
   Poser $r_1 = x$
   $r_2 = \frac{1}{2} \quad r_1 + \frac{x}{r_1}$
   $r_3 = \frac{1}{2} \quad r_2 + \frac{x}{r_2}$

2. Calculer l'erreur relative de l'approximation.[7] Compléter le code précédant avec une instruction qui permet l'impression du résultat et de l'erreur relative sous la forme :

```
Racine carré de 81.00 = 21.49
(erreur relative = 1.39e+000)
```

3. Programmer la même procédure en vous servant de l'instruction `for`.
   - Exécuter le programme pour 7 itérations.
   - Faire un graphique des valeurs succesives r_i, $i = 1, \ldots, 8$. (voir figure de gauche)
   - Produire le graphique des erreurs relatives avec une échelle logarithmique. (voir figure de droite)



[4] Khilnani, A. et Tse, E. (1985) : A Fixed Point Algorithm with Economic Applications, *Journal of Economic Dynamics and Control* **9**, 127-137.

[5] Il s'agit de l'algorithme de Newton.

[6] On appelle *pseudo-code* un programme dont la syntaxe ne correspond pas exactement à un langage de programmation particulier. Ainsi un tel code ne fait qu'expliciter la logique d'un programme et doit encore être adapté aux règles de syntaxe de Matlab, par exemple.

[7] On rappelle que l'erreur relative de la $i$eme approximation est définie comme $|\, r_i - \sqrt{x} \,| / \sqrt{x}$.

4. On désire poursuivre l'itération jusqu'à ce que la différence, en valeur absolue, de deux approximations successives soit inférieure à une valeur δ donnée.
   Programmer cette procédure en vous servant de l'instruction `while`.
   Exécuter le code pour $\delta = 10^{-6}$ et $x = 81$.
   Indication : On attire votre attention sur le fait que dans ce cas il convient de ne pas stocker les valeurs successives des approximations (vecteur r). On ne conservera que deux valeurs successives dans deux scalaires r_0 et r_1. L'itération peut alors se schématiser comme :

   tant que *critère d'arrêt pas satisfait* **faire**
   $r_0 = r_1$
   $r_1 = \frac{1}{2} \quad r_0 + \frac{x}{r_0}$
   finfaire

5. Modifier la procédure de sorte à ce que l'on calcule au plus `maxit` approximations.
   Exécuter le code pour `maxit = 10`.
6. Modifier la procédure de sorte à ce que les itérations soient arretées lorsque l'approximation est égale à la valeur de la racine obtenue avec la fonction `sqrt` de Matlab. Imprimer le nombre d'iteration.
7. Presenter l'algorithme programmer sous le point 5) comme une fonction Matlab :

```
r = myracine(x)
```

Note : La racine correspond au zero de la fonction $f(r) = r^2 - x$. On a $f'(r) = 2r$. Cherchons le zero de la fonction avec l'algorithme de Newton :

$$\begin{aligned} r_k &= r_{k-1} - \frac{f(r_{k-1})}{f'(r_{k-1})} \\ &= r_{k-1} - \frac{r_{k-1}^2 - x}{2r_{k-1}} \\ &= r_{k-1} - \frac{r_{k-1}}{2} + \frac{x}{2r_{k-1}} \\ &= \frac{r_{k-1}}{2} + \frac{x}{2r_{k-1}} \\ &= \frac{1}{2} \quad r_{k-1} + \frac{x}{r_{k-1}} \quad . \end{aligned}$$

5.6 FPI00

Avec la méthode du point fixe rechercher les zeros de la fonction

$$e^x - x^3 - 1 = 0\,.$$

5.7 FPI01

On considère le problème de la recherche du zero d'une fonction $f(x) = 0$. Soit la fonction

$$x^3 - \exp(x) + \cos(x) + 1 = 0 \tag{19}$$

dont le graphique est donné dans la figure 2.

1. Définir la fonction $f(x)$ donné en (19) avec la commande Matlab `inline`. Faire un plot de $f(x)$ pour $x \in [-1.5, 5]$ et completer le graphique par une ligne horizontale pour le niveau zero. Redimensionner la fenêtre graphique aux dimensions $x \in [-1.5, 5]$ et $y \in [-2, 11]$ avec la commande `axis`. Sauvegarder le graphique dans un fichier en format `eps` à l'aide de la commande Matlab `print -depsc FPI01.eps`.

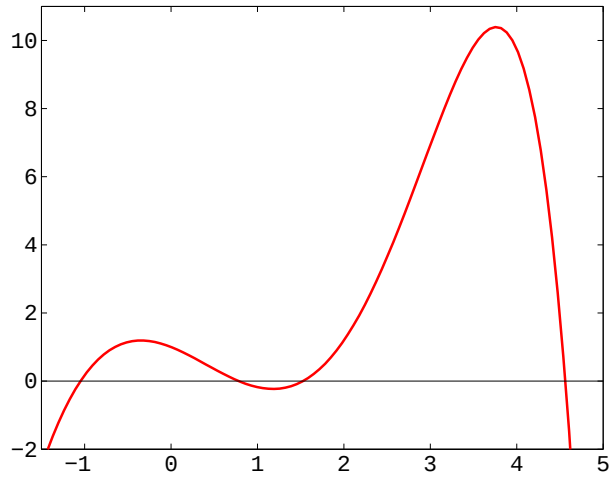


FIG. 2 – Graphique de $x^3 - \exp(x) + \cos(x) + 1$.

2. On envisage d'abord d'utiliser la méthode du point fixe. Pour ce faire on définit les deux fonctions d'itération

$$g1 = \exp(x) - \cos(x) - 1^{\,1/3} \tag{20}$$
$$g2 = \log(x^3 + \cos(x) + 1) \tag{21}$$

   Appliquer la méthode du point fixe en utilisant successivement les fonctions d'itération $g1$ et $g2$ et en choisissant comme point de départ 1 puis 3. Completer le graphique avec les solutions z trouvées. On plottera pour chaque solution le point $z, f(z)$. Commenter le résultat.

3. Avec Maple calculer la dérivée de chacune des fonctions d'itération et faire un graphique des dérivées. Pour $g1'$ on fera varier x dans l'intervalle $[0.8, 4]$ et pour $g2'$ x variera dans l'intervalle $[-0.8, 5]$. Quel commentaire peut-on maintenant faire quant à la convergence de la méthode du point fixe pour le problème étudié.
4. Un des avantages de la méthode de la bisection est qu'elle cherche la solution dans un intervalle donné. En utilisant la méthode de la bisection montrer qu'elle identifie sans problème le zero qui se trouve dans l'intervalle $x \in [-2, -1]$. Completer le graphique par la solution trouvée.
5. Programmer une fonction Matlab qui réalise la méthode de Newton et chercher la solution qui correspond au point de départ $x0 = -2$. Completer le graphique par la solution trouvée. L'Algorithme 4 rappelle la méthode de Newton.

Algorithm 4 Recherche des zeros d'une fonction avec la méthode de Newton.

1: Initialiser $x^{(0)}$
2: **pour** $k = 1, 2, \ldots$ jusqu'à convergence **faire**
3: $\quad x^{(k)} = x^{(k-1)} - \dfrac{f(x^{(k-1)})}{f'(x^{(k-1)})}$
4: **finfaire**

6 Systèmes linéaires

6.1 mat-triu

Définition On appelle une matrice triangulaire avec une diagonale unitaire une matrice triangulaire unitaire.

Vérifier les propriétés suivantes du produit et de l'inverse des matrices triangulaires :
- L'inverse d'une matrice triangulaire inférieure (supérieure) est triangulaire inférieur (supérieur).
- Le produit de deux matrices triangulaires inférieures (supérieures) est triangulaire inférieur (supérieur).
- L'inverse d'une matrice triangulaire inférieure (supérieure) unitaire est triangulaire inférieure (supérieure) unitaire.
- Le produit de deux matrices triangulaires inférieures (supérieures) unitaires est triangulaire inférieure (supérieure) unitaire.

6.2 rsl06

Programmer les algorithmes 4.1 et 4.2 pour la résolution de systèmes triangulaires sous la forme de fonctions MATLAB. La structure des fonctions sera la suivante :
- `b = fsub(L,b)`, avec L une matrice triangulaire inférieure. La solution est retournée dans b.
- `b = fsub1(L,b)`, version particulière de `fsub` où L est une matrice triangulaire unitaire inférieure pour laquelle il ne sera pas nécessaire de donner la diagonale. La solution est retournée dans b.
- `b = bsub(U,b)`, avec U une matrice triangulaire supérieure. La solution est retournée dans b.

Etablir la fonction du nombre d'opérations élémentaires (flops) pour chaque algorithme ? Vérifier le résultat empiriquement. De quel ordre est la complexité de ces algorithmes ?

6.3 rsl01c

Soit le système linéaire $Ax = b$ de dimension n. La règle de Cramer permet de calculer la solution x de ce système en ayant recours aux déterminants par la formule suivante,

$$x_i = \frac{|A_i|}{|A|} \quad , \quad i = 1, 2, \ldots, n \tag{22}$$

où A_i est la matrice A dans laquelle on a remplacé la i-ème colonne par le vecteur b.

On suppose que chaque déterminant est calculé selon le schéma d'expansion des mineurs. Le déterminant de la matrice A s'écrit alors,

$$|A| = \sum_{p \in P(n)} s(p) \prod_{i=1}^{n} a_{i\,p_i} \tag{23}$$

où $P(n)$ est l'ensemble des $n!$ permutations de l'ensemble $\{1, 2, \ldots, n\}$, p_i est le i-ème élément de la permutation p et $s(p)$ est une fonction prenant les valeurs $+1$ ou -1 selon p.

1. Développer une formule donnant approximativement le nombre d'opérations en virgule flottante à effectuer pour obtenir la valeur du déterminant d'une matrice d'ordre n d'après l'équation (80).
2. Donner une formule pour le nombre d'opérations utilisées pour résoudre le système linéaire avec la règle de Cramer donnée sous (79).
3. Quel temps nécessite la résolution d'un système de taille $n = 90$ sur un ordinateur pouvant effectuer :
   (a) 1 Gflops (Pentium IV 2.8 GHz),
   (b) 1 Tflops, c'est-à-dire 10^{12} flops/seconde (Ordinateur parallèle ?).

   Commenter.
4. Avec MATLAB, générer un système de taille $n = 90$ et le résoudre. Combien d'opérations en virgule flottante a-t-on effectué et combien de temps cela a-t-il pris ?

6.4 rslmi01

Soit le système linéaire $Ax = b$ avec

$$A = \begin{bmatrix} -1 & 0 & 0 & 0 & a_1 & 0 & a_2 & 0 \\ 0 & -1 & 0 & 0 & a_3 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 & 0 & a_4 \\ 1 & 1 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 1 & -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & -1 \end{bmatrix}, \quad b = \begin{bmatrix} -20 \\ -.3 \\ -14 \\ 2.2 \\ 0 \\ -200 \\ -8 \\ -1.6 \end{bmatrix},$$

$a_1 = .2$, $a_2 = .8$, $a_3 = .5$ et $a_4 = .47$.

On désire résoudre ce système avec les différentes méthodes itératives. Afin d'éviter des opérations redondantes impliquant les éléments nuls dans la matrice A on codera chaque produit vectoriel $\sum_{j=1}^{8} a_{ij} x_j$, $i = 1, \ldots, 8$ de sorte à ne faire intervenir que les produits non nuls. Pour ce faire on construira une matrice caractères comme suit :

```
eq = str2mat('a1*x(5)+a2*x(7)','a3*x(5)',...
             'a4*x(8)','x(1)+x(2)','   ...     ');
```

dont la i$^{\text{ème}}$ ligne sera évalué à l'aide de la fonction `eval(eq(i, :))`.

1. Pour la méthode de Jacobi, calculer le rayon spectral de la matrice qui gouverne la convergence et programmer la résolution du système linéaire. Compter le nombre d'opérations élémentaires à l'aide de la fonction `flops`.
2. Pour la méthode de Gauss-Seidel, calculer le rayon spectral de la matrice qui gouverne la convergence et programmer la résolution du système linéaire. Compter le nombre d'opérations élémentaires à l'aide de la fonction `flops`.
3. En faisant varier ω déterminer la valeur qui minimise le rayon spectral de la matrice qui gouverne la convergence pour la méthode de sur-relaxation successive. Programmer la résolution du système linéaire avec la méthode de sur-relaxation successive. Compter le nombre d'opérations élémentaires à l'aide de la fonction `flops`.

6.5 Obstacle

On considère la fonction

$$f(x) = \frac{1}{(0.6\,x+0.2)^2+0.14} + \frac{1}{(0.7\,x-0.4)^2+0.16} - 6\,. \quad (24)$$

1. Avec `inline` définir la fonction (24) et faire un graphique dans une fenêtre allant de -1 à 1 pour l'abscisse et de 0 à 3.5 pour l'ordonnée. Completer le graphique avec un maillage (fonction `grid`).
2. En se servant des fonctions Matlab, déterminer les points maximum, minimum et zeros de la fonction dans l'intervalle $x \in [-1, 1]$. Completer le graphique en marquant ces points avec le symbole $\circ$.

On fixe maintenant un fil élastique aux points $x = -1$ et $x = 1$ et on le fait passer par dessus l'obstacle défini par la fonction (24). Il s'agit alors de déterminer les 4 points à partir desquels le fil élastique n'adhere plus à l'obstacle. En notant $u(x)$ la position du fil et $f(x)$ l'hauteur de l'obstacle respectivement, ce problème se formalise comme

$$\begin{aligned} u''(u-f) &= 0 && (25)\\ -u'' &\geq 0 && (26)\\ u-f &\geq 0 && (27) \end{aligned}$$

avec $u(-1) = u(1) = 0$ et u, u' continues.

Dans cette formulation la relation (26) s'interprète que le trajet de l'élastique ne peut être que concave. La relation (25) s'interprète que, soit l'élastique adhère (dans ce cas on a $u - f = 0$), soit il n'adhère pas (dans ce cas il dessine une droite et $u'' = 0$). Le fait que l'élastique ne peut passer par dessous l'obstacle est exprimé par la relation (27).

On envisage la solution de ce problème à l'aide d'une méthode aux différences finies. Pour ce faire on discrétise la variable x en divisant le domaine $x \in [-1, 1]$ en $2N$ intervalles équidistants de longueur $\triangle_x = 2/2N = 1/N$. Ainsi x pourra prendre les $2N + 1$ valeurs

$$(-N+i)\,\triangle_x, \qquad i = 0, 1, \ldots, 2N$$

et les valeurs des fonctions u et f aux point définis par la discrétisation sont notés

$$u_i = u((-N+i)\,\triangle_x) \qquad \text{et} \qquad f_i = f((-N+i)\,\triangle_x)\,.$$

La dérivée u'' est approximée comme

$$u'' \approx \frac{u_{i-1} - 2u_i + u_{i+1}}{\triangle_x^2}$$

et les relations (25–27) s'écrivent maintenant

$$\begin{aligned} (u_{i-1} - 2u_i + u_{i+1})(u_i - f_i) &= 0\\ -u_{i-1} + 2u_i - u_{i+1} &\geq 0\\ u_i - f_i &\geq 0 \end{aligned}$$

pour $i = 1, 2, \ldots, 2N-1$ et $u_0 = u_{2N} = 0$. Avec une notation matricielle on écrit

$$\begin{aligned} (u-f)\cdot Bu &= 0\\ Bu &\geq 0\\ u &\geq f \end{aligned}$$

ou B est une matrice d'ordre $2N-1$ et u, f sont des vecteurs

$$B = \begin{bmatrix} 2 & -1 & 0 & \cdots & 0\\ -1 & 2 & -1 & & \vdots\\ 0 & -1 & 2 & \ddots & 0\\ \vdots & & \ddots & \ddots & -1\\ 0 & \cdots & 0 & -1 & 2 \end{bmatrix} \quad u = \begin{bmatrix} u_1\\ \vdots\\ \vdots\\ \vdots\\ u_{2N-1} \end{bmatrix} \quad f = \begin{bmatrix} f_1\\ \vdots\\ \vdots\\ \vdots\\ f_{2N-1} \end{bmatrix}$$

et $\cdot$ désigne la multiplication élément par élément. Avant de présenter l'algorithme de resolution donnons une formalisation plus générale

$$\begin{aligned} (x-c)\cdot(Ax-b) &= 0\\ Ax &\geq b\\ x &\geq c \end{aligned}$$

dans laquelle il suffit de poser $b = 0$, $x = u$, $A = B$ et $c = f$ pour retomber sur notre problème. Pour construire la solution on résoudra $Ax \geq b$ avec une méthode itérative, de sorte à pouvoir vérifier à chaque itération k pour tous les éléments du vecteur x^k la contrainte $x_i^k \geq c_i$.

Choisir $x^1 > c$, η et ω et poser $x^0 = 0$
tant que $|x^1 - x^0| > \eta$ **faire**
  $x^0 = x^1$
  pour $i = 1 : 2N-1$ **faire**
    $y_i^{\mathrm{GS}} = b_i - \sum_{j=1}^{i-1} A_{ij}x_j^1 - \sum_{j=i+1}^{2N-1} A_{ij}x_j^0 \quad /A_{ii}$
    $x_i^1 = \max\ c_i,\ x_i^0 + \omega(y_i^{\mathrm{GS}} - x_i^0)$
  finfaire
finfaire

L'algorithme ne conserve que les vecteurs x^0 et x^1 de deux itérations successives. A la fin x^1 correspond à la solution. La méthode itérative pour la solution du système linéaire est du type SOR.

3. Programmer cet algorithme et chercher la solution du problème en posant $N = 200$, $\eta = 10^{-5}$ et $\omega = 1.95$.
4. Completer le graphique en traçant les segments de l'élastique qui n'adherent pas à l'obstacle en noir.

7 Factorisations

7.1 complex1

On considère l'algorithme `tgm1` qui applique la transformation de Gauss à la première colonne d'une matrice :

Algorithme (`tgm1`) Soit une matrice $C \in \mathbb{R}^{m\times m}$. L'algorithme remplace $C_{2:m,1}$ par le vecteur des multiplicateurs τ, et la sous-matrice $C_{2:m,2:m}$ par le produit M_1C. Les éléments de $C_{1,1:m}$ restent inchangés.

function $C = \mathbf{tgm1}(C)$
$m = \mathbf{size}(C, 1)$
if $C_{1,1} = 0$, **then** Pivot est nul, arrêt.
$C_{2:m,1} = C_{2:m,1}/C_{1,1}$
$C_{2:m,2:m} = C_{2:m,2:m} - C_{2:m,1}\ C_{1,2:m}$

1. Programmer l'algorithme sous la forme d'une fonction Matlab.
2. Quel est le nombre d'opérations élémentaires que nécessite l'exécution de cet algorithme ?

La factorisation d'une matrice A en une matrice triangulaire inférieure L et une matrice triangulaire supérieure U est obtenue avec l'algorithme suivant :

Algorithme (`eg`) Factorisation $A = LU$. L'algorithme remplace A avec U et L à l'exception des éléments de la diagonale de L.

function $A = \mathbf{eg}(A)$
for $k = 1 : n-1$
  $A_{k:n,k:n} = \mathbf{tgm1}(A_{k:n,k:n})$
end

3. Programmer l'algorithme sous la forme d'une fonction Matlab.
4. Générer une matrice aléatoire A d'ordre 10 et vérifier le fonctionnement de l'algorithme en effectuant le produit LU.
5. Quel est le nombre d'opérations élémentaires que nécessite l'exécution de l'algorithme `eg`? (On pourra se servir de MAPLE pour établir cette expression). Vérifier le résultat empiriquement avec la fonction Matlab `flops`.

7.2 TriLinv

On désire programmer un algorithme qui recherche l'inverse d'une matrice L triangulaire inférieure. Une approche possible consiste à résoudre les n systèmes triangulaires

$$LX = I\,.$$

1. Programmer un algorithme qui recherche l'inverse d'une matrice triangulaire inférieure L en appliquant successivement la "forward substitution".
   Etablir la fonction du nombre d'opérations élémentaires (flops). De quel ordre est la complexité de cet algorithme ?

On envisage maintenant une autre approche. Ecrivons les équations qui définissent le système $LT = I$, avec T la matrice inverse de L.

$$\begin{bmatrix} \ell_{11} & & & & \\ \ell_{21} & \ell_{22} & & & \\ \ell_{31} & \ell_{32} & \ell_{33} & & \\ \vdots & \vdots & \vdots & \ddots & \\ \ell_{n1} & \ell_{n2} & \ell_{n3} & \dots & \ell_{nn} \end{bmatrix} \begin{bmatrix} t_{11} & & & & \\ t_{21} & t_{22} & & & \\ t_{31} & t_{32} & t_{33} & & \\ \vdots & \vdots & \vdots & \ddots & \\ t_{n1} & t_{n2} & t_{n3} & \dots & t_{nn} \end{bmatrix} = \begin{bmatrix} 1 & & & \\ & 1 & & \\ & & \ddots & \\ & & & 1 \end{bmatrix}$$

$$\begin{aligned} \ell_{11}\cdot t_{11} = 1 &\quad\Rightarrow\quad t_{11} = 1/\ell_{11} \\ \ell_{21}\cdot t_{11} + \ell_{22}\cdot t_{21} = 0 &\quad\Rightarrow\quad t_{21} = -\ell_{21}\cdot t_{11}/\ell_{22} \\ \ell_{22}\cdot t_{22} = 1 &\quad\Rightarrow\quad t_{22} = 1/\ell_{22} \\ \ell_{31}\cdot t_{11} + \ell_{32}\cdot t_{21} + \ell_{33}\cdot t_{31} = 0 &\quad\Rightarrow\quad t_{31} = -(\ell_{31}\cdot t_{11} + \ell_{32}\cdot t_{21})/\ell_{33} \\ \ell_{32}\cdot t_{22} + \ell_{33}\cdot t_{32} = 0 &\quad\Rightarrow\quad t_{32} = -\ell_{32}\cdot t_{22}/\ell_{33} \\ \ell_{33}\cdot t_{33} = 1 &\quad\Rightarrow\quad t_{33} = 1/\ell_{33} \\ \vdots & \end{aligned}$$

On voit qu'il s'agit d'un système d'équations récursives qui définissent les éléments de la matrice inverse T comme

$$t_{ij} = \begin{cases} 1/\ell_{ii} & i = j \\ -\sum_{k=j}^{i-1} \ell_{ik}\, t_{kj} \;/\ell_{ii} & i \neq j \end{cases}$$

2. Programmer un algorithme qui inverse une matrice triangulaire inférieure L en exploitant cette récurrence. Donner au programme la forme de la fonction Matlab :

```
T = TrilInv0(L)
```

   Etablir la fonction du nombre d'opérations élémentaires (flops). De quel ordre est la complexité de cet algorithme ?
3. Réécrire l'algorithme `TrilInv0` de sorte que la matrice L soit remplacée par son inverse. On aura

```
L = TrilInv(L)
```

4. Expliquer exactement d'où provient la différence du nombre d'opérations élémentaires entre l'algorithme du point 1) et l'algorithme qui se sert de la récurrence (point 2).

7.3 rsl09

1. Programmer l'algorithme 5.3 (`egpp`), l'algorithme 5.4 (`rsl`) et `perm` sous forme de fonctions MATLAB.
2. Générer une matrice

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 10 \end{bmatrix}$$

   et appliquer `egpp` à cette matrice. Construire L, U et P et vérifier que $PA = LU$.
3. Utiliser la fonction `rsl` pour résoudre les systèmes linéaires $Ax_i = b_i$, $i = 1, 2$ où A est définie comme précédemment et $b_1 = [10\ 20\ 30]'$, $b_2 = [11\ 12\ 13]'$.

7.4 rsl09b

1. Programmer l'algorithme `egpp` et `perm` sous forme de fonction Matlab.
2. Générer une matrice

$$A = \begin{bmatrix} 1 & 2 & 3 & 4 \\ -6 & 5 & 3 & 6 \\ 4 & -3 & 9 & 8 \\ 3 & 5 & 1 & 8 \end{bmatrix}$$

   et appliquer `egpp` à cette matrice. Construire L, U et P et vérifier que $PA = LU$.
3. Programmer l'algorithme `rsl` sous forme de fonction Matlab et utiliser `rsl` pour résoudre les systèmes linéaires $Ax^{(i)} = b^{(i)}$, $i = 1, 2$ où A est définie comme précédemment et $b^{(1)} = \begin{bmatrix} 10 & 20 & 30 & 40 \end{bmatrix}'$, $b^{(2)} = \begin{bmatrix} 11 & 12 & 13 & 15 \end{bmatrix}'$.

7.5 GaussJordan

L'élimination dite de *Gauss-Jordan* transforme un système linéaire $Ax = b$ en un système $Dx = c$ avec D une matrice diagonale.

La transformation qui annule tous les éléments, à l'exception de l'élément x_k d'un vecteur x se formalise comme

$$M_k\, x = \begin{bmatrix} 1 & \cdots & 0 & -\tau_1^{(k)} & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \vdots & & \vdots \\ 0 & \cdots & 1 & -\tau_{k-1}^{(k)} & 0 & \cdots & 0 \\ 0 & \cdots & 0 & 1 & 0 & \cdots & 0 \\ 0 & \cdots & 0 & -\tau_{k+1}^{(k)} & 1 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & -\tau_n^{(k)} & 0 & \cdots & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_{k-1} \\ x_k \\ x_{k+1} \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ x_k \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

où $\tau_i^{(k)} = x_i/x_k$, $i = 1, \ldots, n$.

1. Ecrire une procédure qui pour n et k donné génère un vecteur aléatoire $x \in \mathbb{R}^n$, construit la matrice M_k et calcule le vecteur $z = M_k\, x$. On choisira $n = 8$, $k = 3$. Vérifier que $z_i = 0$, $\forall i \neq k$.
2. Ecrire la procédure qui résoud un système linéaire par la méthode de Gauss-Jordan. On procédera de la manière suivante : Pour n donné générer une matrice aléatoire $A \in \mathbb{R}^{n\times n}$ et un vecteur b unitaire et former la matrice bordée $B = \begin{matrix} A & b \end{matrix}$. Pour $k = 1, \ldots, n$ effectuer le produit $B = M_k\, B$. La solution du système linéaire $Ax = b$ s'écrit alors $x_i = B_{i,n+1}/B_{ii}$, $i = 1, \ldots, n$.
3. Résoudre le système linéaire avec la factorisation LU et calculer la norme 2 du vecteur de différences entre la solution obtenue avec LU et celle obtenue par la méthode de Gauss-Jordan.
4. A l'aide de la commande `flops` comparer le nombre d'opérations élémentaires nécessaire à chacune de ces méthodes.

7.6 Chol0

Programmer l'algorithme `cholesky` du cours. Comparer votre version avec `chol` de Matlab. Etudier de manière analytique et empirique la complexité de votre algorithme.

7.7 Chol1

1. Pour $n = 200$ et $k = 400$ créer une matrice $Z \in \mathbb{R}^{k\times n}$ dont les éléments sont des variables pseudo-aléatoires uniformément distribués dans $[-10, 10]$. Calculer la matrice de covariance $W = \mathrm{Cov}(Z)$ à l'aide de la fonction Matlab `cov`.
2. Factoriser la matrice W et vérifier qu'elle est définie positive.
3. Générer une matrice aléatoire $X \in \mathbb{R}^{n\times m}$ avec $m = 10$ ainsi qu'un vecteur aléatoire $y \in \mathbb{R}^n$.
4. Evaluer l'expression
$$\hat{\beta} = (X'W^{-1}X)^{-1}X'W^{-1}y \tag{28}$$
en calculant les inverses des matrices avec la fonction Matlab `inv`. A l'aide de la fonction `flops` donner le nombre d'opérations élémentaires nécessaires à cette évaluation.
5. Evaluer l'expression (28) sans le calcul explicit des inverses et comparer le nombre d'opérations élémentaires.

7.8 stosim

Pour des exercices de simulation stochastique notamment on est souvent amené à générer des vecteurs aléatoires qui suivent une loi normale donnée $N(\hat{\beta}, \hat{V})$ avec $\hat{\beta}$ l'estimation de l'espérance et $\hat{V}$ l'estimation de la matrice des variances et covariances. Pour générer un vecteur β^* tiré de cette loi on peut procéder de la façon suivante :

- Factoriser $\hat{V}$ en $R'R$ avec R triangulaire supérieure
- Générer un vecteur e dont chaque composante e_i suit une loi $N(0, 1)$
- Calculer $\beta^* = \hat{\beta} + R'e$

On vérifie alors que $E(\beta^*) = \hat{\beta}$ et $E\left[(\beta^* - \hat{\beta})(\beta^* - \hat{\beta})'\right] = \hat{V}$.

Application : Soit

$$\hat{\beta} = \begin{bmatrix} 16.55 \\ .0173 \\ .2162 \\ .8102 \end{bmatrix} \qquad \hat{V} = 10^{-4} \times \begin{bmatrix} 17450 & -15.3 & -4.7 & -32.7 \\ -15.3 & 139.4 & -95.7 & -15.3 \\ -4.7 & -95.7 & 115.1 & -5.3 \\ -32.7 & -15.3 & -5.3 & 16.2 \end{bmatrix}$$

Générer 1000 vecteurs β^* et calculer la moyenne et la matrice des variances et covariances de ces vecteurs.

7.9 givens00

L'algorithme 5 qui suit calcule les coefficients c et s de la matrice

$$G = \begin{matrix} c & s \\ -s & c \end{matrix} \tag{29}$$

telle que

$$\begin{matrix} c & s \\ -s & c \end{matrix}' \begin{matrix} a \\ b \end{matrix} = \begin{matrix} r \\ 0 \end{matrix}.$$

Algorithm 5 (Rotation de Givens)

si $b = 0$ **alors**
  $c = 1$; $s = 0$
sinon
  si $|b| > |a|$ **alors**
    $\tau = -a/b$; $s = 1/\sqrt{1+\tau^2}$; $c = s\,\tau$
  sinon
    $\tau = -b/a$; $c = 1/\sqrt{1+\tau^2}$; $s = c\,\tau$
  finsi
finsi

1. Écrire une fonction Matlab

   ```
   [c,s] = givens(a,b)
   ```

   qui réalise l'algorithme 5.
2. Dans la suite :
   (a) Genérer une matrice aléatoire $A \in \Re^{m\times n}$ avec $m = 7$ et $n = 5$.
   (b) On désire annuler l'élément A_{63} en appliquant une rotation de Givens. Pour $i = 2$ et $k = 6$ construire

la matrice

$$G(i,k,\theta) = \begin{array}{c} \\ \text{i} \\ \\ \text{k} \\ \\ \end{array} \begin{bmatrix} 1 & \cdots & 0 & \cdots & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & & \vdots & & \vdots \\ 0 & \cdots & c & \cdots & s & \cdots & 0 \\ \vdots & & \vdots & \ddots & \vdots & & \vdots \\ 0 & \cdots & -s & \cdots & c & \cdots & 0 \\ & & \vdots & & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & 0 & \cdots & 1 \end{bmatrix}.$$

(Tous les éléments de la diagonale valent 1 à l'exception des éléments en position i et k).

(c) Effectuer le produit $G'A$ et vérifier que A_{63} et seules les lignes $i = 2$ et $k = 6$ de la matrice A ont étés modifiées.

(d) Quel est le nombre d'operations exécutés ?

3. Afin d'éviter des opérations redondantes dans le produit $G'A$ on propose l'algorithme 6 :

Algorithm 6 (Rotation de 2 lignes de la matrice $A \in \Re^{2\times n}$). L'algorithme remplace la matrice A avec $G'A$. (G défini en (29)).

pour $j = 1 : n$ **faire**
  $\tau_1 = A_{1,j}$; $\tau_2 = A_{2,j}$;
  $A_{1j} = c\,\tau_1 - s\,\tau_2$; $A_{2j} = s\,\tau_1 + c\,\tau_2$;
finfaire

Écrire une fonction Matlab

```
A = rowrot(A,c,s)
```

qui réalise l'algorithme 6.

4. On désire maintenant rendre triangulaire supérieure la matrice A. Considérons d'abord la colonne $j = 1$. Alors la procédure suivante annulera successivement les éléments des lignes m, $m-1$ jusqu'à 2 de la colonne 1 de la matrice A :

```
pour i = m : -1 : 2 faire
  [c,s] = givens(A(j,j),A(i,j)) ;
  A([j i], :) = rowrot(A([j i], :),c,s) ;
finfaire
```

Réaliser cette procédure et vérifier le résultat.

5. En répétant la procédure précédente successivement pour les colonnes $j = 1, \ldots, n$ de la matrice A on obtiendra une matrice triangulaire.

```
pour j = 1 : n faire
  pour i = m : -1 : j + 1 faire
    [c,s] = givens(A(j,j),A(i,j)) ;
    A([j i], :) = rowrot(A([j i], :),c,s) ;
  finfaire
finfaire
```

Réaliser cette procédure et vérifier le résultat.

6. Soit la matrice $Q = G_1 \cdots G_r$. Réaliser dans la procédure précédente la construction des matrices G_i, $i = 1, \ldots, r$ ou r est le nombre total de rotations. Calculer Q, vérifier que $Q'Q = I$ et que $A = QR$.

7.10 svdprox

Une application possible pour l'approximation d'une matrice par une matrice de rang inférieur constitue la compression de données lors de la transmission d'un très grand nombre d'images prises par une sonde spatiale.

Pour ce faire, on digitalise au préalable une image en lui appliquant une grille de n lignes et n colonnes puis on assigne à chaque carré de la grille un nombre qui correspond à son niveau de luminosité suivant par exemple une échelle de 0 à 10. Ainsi l'image est traduite en n^2 entiers qu'il faudra par la suite transmettre sur Terre.

En procédant de la sorte, le volume de données à transmettre peut cependant devenir rapidement prohibitif lorsqu'il s'agit d'un très grand nombre d'images provenant d'une grande distance. Une possibilité consiste alors à appliquer à la matrice des luminositées une décomposition singulière et à ne transmettre que les $2k$ vecteurs singuliers ainsi que les k valeurs singulières qui permettent une approximation satisfaisante de la matrice originale. Ainsi, si une approximation de rang 6 pour une matrice d'ordre 1000 s'avère suffisante, le nombre de données à transmettre serait de $2 \times 6 \times 1000 + 6 = 12'006$ au lieu de 10^6, ce qui correspond à un gain de 98.8%.

Application : Le fichier `v:\metri\svdEx2.bmp` (à charger avec la commande `imread`) contient une matrice A de dimension 223×300 qui correspond à une image digitalisée avec deux niveaux de luminosité codés avec 0 et 1. Pour visualiser cette image avec Matlab on peut procéder comme suit :
- Changer le format de codage des éléments de A avec la commande `double` ;
- utiliser la commande `spy`.

En se basant sur l'étude des valeurs singulières rechercher l'approximation B de plus petit rang de la matrice A qui soit acceptable pour la qualité de cette image. (Pour la matrice approchée B on imprimera une marque si $b_{ij} > .4$).

8 Systèmes non-linéaires

8.1 JacNum

Soit un système de n fonctions non linéaires

$$F(x) \quad \Leftrightarrow \quad \begin{cases} f_1(x) \\ f_2(x) \\ \vdots \\ f_n(x) \end{cases}$$

et sa matrice Jacobienne

$$J = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & & \vdots \\ \frac{\partial f_n}{\partial x_1} & \cdots & \frac{\partial f_n}{\partial x_n} \end{bmatrix}.$$

Une approximation numérique de la matrice Jacobienne peut être obtenue avec une *différence progressive* (forward difference), qui est définie par

$$J_{ij} = \frac{f_i(x + \Delta_j e_j) - f_i(x)}{\Delta_j}$$

où Δ_j est choisi suffisamment petit et e_j est le j-ème vecteur de la matrice identité.

En utilisant la notation vectorielle, on peut évaluer la j-ème colonne de la matrice Jacobienne par

$$J_{.j} = \frac{F(x + \Delta_j e_j) - F(x)}{\Delta_j}$$

Soit $F(x)$ définie ainsi :

$$F(x) = \begin{bmatrix} x_1^2 + x_2^3 - x_3^4 \\ x_1 x_2 x_3 \\ 2x_1x_2 - 3x_2x_3 + x_1x_3 \end{bmatrix} .$$

1. Écrire une fonction Matlab qui, pour un vecteur x donné, retourne la valeur de $F(x)$

   ```
   F = func(x)
   ```

2. Écrire le code Matlab qui évalue une matrice Jacobienne par la méthode des différences progressives. On choisira $\Delta_j = \sqrt{eps}\,|x_j|$, avec `eps` la précision machine dans Matlab.
3. Coder les expressions analytiques qui définissent les éléments de la matrice Jacobienne. Pour $x = \begin{matrix} 1.2 & 9.3 & 25.7 \end{matrix}$ comparer les valeurs qui correspondent à l'évaluation analytique avec les résultats de l'évaluation numérique.
4. Une approximation plus précise des éléments de la matrice Jacobienne s'obtient avec la méthode dite *différence centrée* (central difference). L'évaluation de la j-ème colonne de la matrice Jacobienne s'écrit alors

   $$J_{.j} = \frac{F(x + \Delta_j e_j) - F(x - \Delta_j e_j)}{2\,\Delta_j}$$

   Écrire le code Matlab qui évalue une matrice Jacobienne par la méthode différence centrée. Pour $x = \begin{matrix} 1.2 & 9.3 & 25.7 \end{matrix}$ comparer les résultats quant à leur précision et quant au nombre d'opérations élémentaires qu'ils nécessitent.
5. Réécrire votre code en faisant usage de la fonction Matlab `feval`.

8.2 rsnl01

Soit le système d'équations non-linéaires suivant :

$$\begin{aligned} a_1 \log(x_2) - a_2 x_1/(x_5 + a_3) &= 0 \\ a_4 x_2 x_5 + a_5 x_4 - x_3 + a_6 &= 0 \\ x_4 x_5 - x_2 + a_7 &= 0 \\ x_3 - x_4 + a_8 &= 0 \\ a_9 x_1 - x_3 + a_{10} &= 0 \end{aligned}$$

avec $a = \begin{matrix} 1 & 2 & 3 & \frac{1}{2} & 4 & 5 & 6 & 7 & 8 & 9 \end{matrix}$.

1. Résoudre ce système à l'aide de la fonction `fsolve` de MATLAB.
2. Résoudre ce système à l'aide de la méthode de Gauss-Seidel.
3. Résoudre ce système à l'aide de la méthode de Newton.

9 Moindes carrés

9.1 mc01

Soit le problème $Ax \cong b$ avec

$$A = \begin{bmatrix} 1 & 1 \\ 1 & 1 \\ 1 & 1+\epsilon \end{bmatrix} \quad \text{et} \quad b = \begin{bmatrix} 2 \\ 3 \\ 2 \end{bmatrix} .$$

Par la méthode des équations normales, donner la solution analytique pour x. Pour quelles valeurs de ϵ une résolution numérique avec ordinateur sera-t-elle impossible.

9.2 mc02

Soit le modèle linéaire classique

$$y = X\beta + \epsilon \quad \text{avec} \quad \epsilon \sim iid(0, \sigma^2 I) \quad .$$

On désire trouver l'estimation des moindres carrés du vecteur des paramètres β et l'estimation de la variance des erreurs $\hat{\sigma}^2$. Les observations sont :

$$X = \begin{bmatrix} 1 & -1 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 2 & 1 \\ 1 & 0 & -1 \\ 1 & 0 & 0 \end{bmatrix} \qquad y = \begin{bmatrix} 0 \\ 2 \\ 1 \\ 2 \\ -1 \\ 1 \end{bmatrix} .$$

Procéder à l'estimation en utilisant, successivement :
– la méthode des équations normales,
– la factorisation QR de X.

9.3 mc-eqn00

On considère le problème des moindres carrés $Ax \approx b$. On rappelle que pour la résolution des équations normales et le calcul de la matrice des variances covariances on procède suivant les étapes suivantes :

- Former la matrice $\overline{A} = \begin{matrix} A & b \end{matrix}$
- Calculer le triangle inférieur de $\overline{C} = \overline{A}'\overline{A}$
- Calculer la factorisation de Cholesky $\overline{G} = \begin{matrix} G & \\ z' & \rho \end{matrix}$ qui correspond à $\overline{C}$
- Résoudre $G'x = z$
- Calculer $\sigma^2 = \rho^2/(m-n)$
- Calculer $T = G^{-1}$
- Calculer $S = \sigma^2 T'T$

Donner (sans le démontrer) le nombre d'opérations élémentaires nécessaires à l'exécution de chaque étape de cette procédure pour résoudre un problème comportant m observations et n variables. Donner le nombre total des opérations élémentaires.

9.4 lauchli3

Afin de pouvoir vérifier la précision des calculs numériques, on construit des problèmes pour lesquels la solution exacte est connue. Il devient alors possible de juger de la précision des calculs en comparant les résultats numériques avec la solution exacte.

Un problème imaginé par Läuchli[8] consiste à estimer le paramètre β dans le modèle linéaire classique

$$y = X\beta + u$$

avec $X \in \mathbb{R}^{n\times k}$, $n \geq 3$, $k = n-1$ et $y \in \mathbb{R}^n$ définies comme

$$X_{ij} = \begin{cases} 1 \text{ pour } i = 1 \text{ et } j = 1,\ldots,k \\ 1 \text{ pour } j = 1 \text{ et } i = 1,\ldots,n \\ e \text{ pour } i = j \text{ et } i \neq 1 \\ 0 \text{ sinon} \end{cases} \qquad y_i = \begin{cases} n-1+e & \text{pour } i = 1 \\ e & \text{pour } i = 2,\ldots,n-1 \\ n-1-e & \text{pour } i = n \end{cases}$$

Pour ce problème on peut vérifier que la solution analytique de l'estimation de β par les moindres carrés ordinaires correspond à

$$\hat{\beta} = \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix}.$$

1. Pour $n = 41$ et $e = 2^{-23}$ construire la matrice X et le vecteur y proposé par Läuchli. Calculer la condition de la matrice X.
2. Afin d'évaluer la qualité des deux méthodes numériques, proposées ci-après, pour calculer $\hat{\beta}$, effectuer les opérations ci-dessous pour le système avec $n = 41$:
   (a) Calculer la condition de la matrice X.
   (b) Calculer $\hat{\beta}$ par les équations normales $(X'X)\,\hat{\beta} = X'y$ en utilisant la décomposition de Cholesky de $X'X$.
   (c) Utiliser la décomposition QR de X pour résoudre le système surdéterminé $X\hat{\beta} = y$.
   (d) Comparer dans un même graphique les valeurs obtenus pour $\hat{\beta}$ par les deux méthodes.

   On effectuera les 4 points ci-dessus pour les valeurs successives de $e = 2^{-18}$, 2^{-23}, 2^{-24} et 2^{-25}.
   Commenter.

9.5 lauchli

Afin de pouvoir vérifier la précision des calculs numériques, on construit des problèmes pour lesquels la solution exacte est connue. Il devient alors possible de juger de la précision des calculs en comparant les résultats numériques avec la solution exacte.

Un problème imaginé par Läuchli[9] consiste à estimer le paramètre β dans le modèle linéaire classique

$$y = X\beta + u$$

avec $X \in \mathbb{R}^{n\times k}$, $n \geq 3$, $k = n-1$ et $y \in \mathbb{R}^n$ définies comme

$$X_{ij} = \begin{cases} 1 \text{ pour } i = 1 \text{ et } j = 1,\ldots,k \\ 1 \text{ pour } j = 1 \text{ et } i = 1,\ldots,n \\ e \text{ pour } i = j \text{ et } i \neq 1 \\ 0 \text{ sinon} \end{cases} \qquad y_i = \begin{cases} n-1+e & \text{pour } i = 1 \\ e & \text{pour } i = 2,\ldots,n-1 \\ n-1-e & \text{pour } i = n \end{cases}$$

Pour $n = 4$ par exemple, vérifier avec MAPLE que l'estimation de β par les moindres carrés ordinaires correspond à

$$\hat{\beta} = \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix}.$$

Utiliser les commandes suivantes de la librairie `linalg` : `matrix`, `transpose`, `multiply` et `inverse`.

9.6 lauchli2

Reprenons les données X $(n\times(n-1))$ et y $(n\times 1)$ proposées par Läuchli, pour lesquelles on vérifie que l'estimation du modèle linéaire classique par les moindres carrés ordinaires produit le vecteur $((n-1)\times 1)$ $\hat{\beta} = [1\ 1\ \ldots 1]'$ (cf TP #13).

Afin d'évaluer la qualité des deux méthodes numériques suivantes pour calculer $\hat{\beta}$, effectuer les opérations ci-dessous pour un système avec $n = 41$:

1. Calculer $\hat{\beta}$ par les équations normales $(X'X)\,\hat{\beta} = X'y$ en utilisant la décomposition de Cholesky de $X'X$.
2. Utiliser la décomposition QR de X pour résoudre le système surdéterminé $X\hat{\beta} = y$.

On effectuera les 2 points ci-dessus pour des valeurs successives de $e = 2^{-15}$, 2^{-16}, 2^{-17},..., 2^{-27}.

Afin de visualiser les résultats, reporter les estimations de β sur un graphique avec sur l'axe horizontal l'indice $i = 1, 2, 3, \ldots, 40$ et en correspondance les valeurs de $\hat{\beta}_i$ trouvées par les deux méthodes.

Commenter les résultats trouvés.

Indication : Utilisez une boucle sur e dans laquelle vous calculerez les estimations de β et dessinerez ce vecteur. Vous pourrez insérer les 3 commandes suivantes après la fonction `plot` afin de mieux observer ce qui ce produit :

```
drawnow       % rafraîchit la fenêtre graphique de MATLAB
figure(gcf)   % amène la fenêtre graphique au dessus des a
pause         % attend que l'on presse une touche pour
              % continuer l'exécution
```

9.7 MCnl01

On considère le modèle non-linéaire suivant

$$C_t = \alpha + \beta Y_t^{\gamma} + \epsilon_t \qquad t = 1950,\ldots,1985. \tag{30}$$

Le fichier `v:\metri\mne\mcnl01data.m` contient les observations de la variable C et de la variable Y.

On peut aussi formaliser le modèle comme

$$y = f(X, b) + \epsilon \tag{31}$$

avec $y = \quad C \quad$, $X = \quad \iota \quad Y \quad$ et $b = \quad \alpha \quad \beta \quad \gamma \quad$.

[8] Läuchli, P., (1961), Jordan-Elimination und Ausgleichung nach kleinsten Quadraten, *Numerische Mathematik* 3, 226–240.

[9] Läuchli, P., (1961), Jordan-Elimination und Ausgleichung nach kleinsten Quadraten, *Numerische Mathematik* 3, 226–240.

1. Ecrire une fonction MATLAB qui évalue la partie déterministe $\hat{y} = f(X, b)$ du modèle. On appellera cette fonction `model01` et on spécifiera les arguments suivants :

$$\texttt{yhat} = \texttt{model01(X,b)}$$

2. A l'aide de la fonction `model01` évaluer la somme des carrés des résidus $r = y - \hat{y}$ en posant $b = \begin{array}{ccc} 1 & 1 & 1 \end{array}$.

La matrice Jacobienne du modèle est définie comme

$$J = \begin{bmatrix} \frac{\partial f_1(X_{1\,.}, b)}{\partial b_1} & \frac{\partial f_1(X_{1\,.}, b)}{\partial b_2} & \frac{\partial f_1(X_{1\,.}, b)}{\partial b_3} \\ \vdots & \vdots & \vdots \\ \frac{\partial f_T(X_{T\,.}, b)}{\partial b_1} & \frac{\partial f_T(X_{T\,.}, b)}{\partial b_2} & \frac{\partial f_T(X_{T\,.}, b)}{\partial b_3} \end{bmatrix}.$$

Une approximation numérique de la matrice Jacobienne peut être obtenue avec une *différence progressive* (forward difference), qui est définie par

$$J_{ij} = \frac{f_i(X_{i\,.}, b + \delta_j e_j) - f_i(X_{i\,.}, b)}{\delta_j}$$

où δ_j est choisi suffisamment petit et où e_j est le j-ème vecteur de la matrice identité.

En utilisant la notation vectorielle, on peut évaluer la j-ème colonne de la matrice Jacobienne par

$$J_{.j} = \frac{f(X,\, b + \delta_j e_j) - f(X, b)}{\delta_j}$$

3. En choisissant $\delta_j = \sqrt{\varepsilon}\, b_j$ programmer la procédure qui calcule la matrice jacobienne (ε représente la précision machine). Ci-après la structure de la procédure à programmer.

```
1: Initialiser le vecteur δ
2: Construire la matrice diagonale Δ = diag(δ)
3: Calculer ŷ = f(X, b)
4: pour k = 1 : p faire
5:    Calculer ŷ⁺ = f(X, b + Δ_{.k})
6:    J_{.k} = (ŷ⁺ − ŷ)/δ_k
7: finfaire
```

On utilisera `feval` pour évaluer le modèle aux instructions 3 et 5.

On désire maintenant implémenter la méthode de Gauss-Newton pour l'estimation des paramètres b du modèle (31). La structure de l'algorithme de Gauss-Newton est la suivante :

```
1: Initialiser b, it, maxit et η
2: tant que critère de convergence pas satistait faire
3:    Incrémenter it
4:    Evaluer ŷ, r et J
5:    Résoudre Js = r
6:    Mettre à jour le pas b = b + s
7:    Tester si le nombre maximal d'itérations a été atteint
8: finfaire
```

Le critère de convergence est défini comme

$$\frac{|\texttt{sse0} - \texttt{sse}|}{\texttt{sse} + \sqrt{\varepsilon}} < \eta\,.$$

4. Implémenter l'algorithme de Gauss-Newton et l'exécuter en choisissant $b = \begin{array}{ccc} 1 & 1 & 1 \end{array}$ comme valeur initiale. On posera `maxit` $= 50$ et $\eta = 10^{-4}$.

5. La matrice des variances et covariances des paramètres b peut être approchée par l'expression

$$\hat{\Sigma} = \sigma^2 (J'J)^{-1}$$

avec $\sigma^2 = r'r/n$.
Evaluer $\hat{\Sigma}$ et imprimer la sousmatrice qui correspond aux paramètres β et γ.

6. Implémenter l'algorithme de Levenberg-Marquardt.

9.8 mcnl-gn

Ecrire une fonction MATLAB qui implémente la méthode de Gauss-Newton pour l'estimation d'un modèle non-linéaire par la méthode des moindres carrés. On désire que la fonction se presente comme :

$$[\texttt{b}, \texttt{r}, \texttt{J}] = \texttt{mc\_gn}(\texttt{X}, \texttt{y}, \texttt{model}, \texttt{b0})$$

où les arguments d'entrée sont :

`X`	Matrice des observations des variables exogènes
`y`	Vecteur des observations de la variable dépendante
`model`	Fonction qui calcule les $\hat{\texttt{y}}$ du modèle
`b0`	Valeur initiale du vecteur des paramètres

et les arguments de sortie

`b`	Valeur du vecteur des paramètres
`r`	Vecteur des résidus
`J`	Approximation numérique de la matrice jacobienne à la so

On suggère la structure suivante pour le programme :

```
Initialiser J, iter, maxiter, b, tol
tant que  critère de convergence pas satisfait, faire
    Incrémenter iter
    Evaluer yhat  avec la fonction model
    Evaluer la matrice jacobienne J
    Résoudre le pas de Gauss-Newton s
    b = b + s
    Tester si le nombre d'itérations maximales est atteint
finfaire
```

On considère que le critère de convergence n'est pas satisfait ausssi longtemps que

$$\frac{|\texttt{sse0} - \texttt{sse}|}{\texttt{sse} + \sqrt{\texttt{eps}}} > \texttt{tol}.$$

La fonction `model` retourne `yhat` à partir d'une valeur de `b` et la matrice des observations `X`.

Estimer les paramètres x_1 et x_2 du modèle $y = x_1 e^{x_2 t}$ avec $t = \begin{array}{cccc} 0 & 1 & 2 & 3 \end{array}$ et $y = \begin{array}{cccc} 2 & 0.7 & 0.3 & 0.1 \end{array}$.

Sur le modèle de l'algorithme de Gauss-Newton implémenter la méthode de Levenberg-Marquardt.

10 Option pricing

10.1 prog09

Programmer la fonction `BS` ci-après, qui calcule le prix d'un *call* Européen :

```
function call = BS(S,E,r,T,sigma)
% Call Européen avec Black-Scholes
%
d1 = (log(S./E) + (r + sigma.^2 /2) .* T) ...
     ./ (sigma .* sqrt(T));
d2 = d1 - sigma .* sqrt(T);
Ee = E .* exp(-r .* T);
call = S .* normcdf(d1) - Ee .* normcdf(d2);
```

1. Faire un graphique du prix du *call* en fonction de r, avec $r \in [0.001, 0.10]$.
2. Faire un graphique du prix du *call* en fonction de σ, avec $\sigma \in [0.001, 0.50]$.
3. Faire un graphique du prix du *call* en fonction de r et σ, avec $r \in [0.001, 0.10]$ et $\sigma \in [0.001, 0.50]$.
4. Faire un graphique du prix et des courbes de niveau du *call* en fonction de r et σ, avec $r \in [0.001, 0.10]$ et $\sigma \in [0.001, 0.50]$.

10.2 BSplot

On rappelle que le prix d'un *call* pour une option Européenne avec prix d'exercise E, taux du marché r, T temps restant jusqu'à la maturité et σ la volatilité du sous-jacent S est donné par :

$$C = S\,N(d_1) - E\,e^{-rT}\,N(d_2) \tag{32}$$

où $N(x)$ est la fonction de distribution cummulée de la variable aléatoire normale centrée réduite, définie comme

$$N(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-\frac{1}{2}y^2} dy$$

et

$$\begin{aligned} d_1 &= \frac{\log(S/E) + (r + \frac{1}{2}\sigma^2)T}{\sigma\sqrt{T}} \\ d_2 &= d_1 - \sigma\sqrt{T}. \end{aligned}$$

Le prix du *put* se déduit à partir de la "put-call parity" et s'écrit

$$P = E\,e^{-rT}\,N(-d_2) - S\,N(-d_1)\,. \tag{33}$$

1. Programmer une fonction Matlab qui retourne la valeur d'un call Européen. Donner à la fonction la structure suivante :

   ```
   function call = BScall(S,E,r,T,sigma)
   % Evaluation du prix d'un call Européen avec Black-Scholes
   % call = BScall(S,E,r,T,sigma)
   %
   ```

   Tester la fonction en vérifiant que l'on obtient 11.2460 pour le call lorsqu'on choisit $S = 100$, $E = 95$, $r = 0.05$, $T = 0.40$ et $\sigma = 0.30$. Pour tester la fonction préparer un fichier script avec les commandes :

   ```
   S = 100;
   E = 95;
   r = 0.05;
   T = 0.40;
   sigma = 0.30;
   call = BScall(S,E,r,T,sigma)
   ```

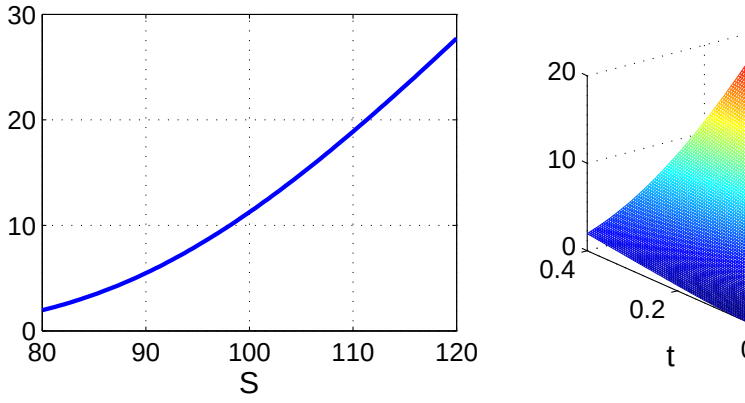
2. Modifier votre fonction de sorte à ce qu'elle retourne un vecteur pour le call lorsqu'un des arguments d'entrée est un vecteur. Pour tester la fonction modifiée on pourra ajouter dans le script précédant les commandes :

   ```
   S = linspace(80,120);
   call = BScall(S,E,r,T,sigma);
   plot(S,call)
   xlabel('S')
   ```

3. Lorsque deux des arguments d'entrée de la fonction `BScall` sont des matrices de coordonnées l'argument de sortie constitue une matrice dont l'élément de la ligne i et de la colonne j correspond au call évalué au coordonnées correspondantes des matrices d'entrée. Completer le script avec les instructions :

   ```
   S = linspace(80,110);
   T = linspace(0,0.40);
   [SS,TT] = meshgrid(S,T);
   c = BScall(SS,E,r,TT,sigma);
   figure
   mesh(S,T,c)
   xlabel('S'); ylabel('t');
   ```

   La figure qui suit illustre les type de graphique obtenu.



4. Programmer une fonction Matlab qui retourne la valeur d'un put Européen. Donner à la fonction la structure suivante :

   ```
   function call = BSput(S,E,r,T,sigma)
   % Evaluation du prix d'un put Européen avec Black-Scholes
   % call = BSput(S,E,r,T,sigma)
   %
   ```

   Tester la fonction en vérifiant que l'on obtient 4.3649 lorsqu'on choisit les mêmes arguments que pour le call.

10.3 FiniteDiffMethod

On considère le problème du calcul du prix d'une option européenne. A un instant donné, le prix de l'option est fonction de la valeur du titre sous-jacent et du temps qu'il reste à parcourir jusqu'à la maturité de ce titre. La méthode dite des différences finies offre une possibilité de calculer une approximation du prix pour de telles options. On procède de la façon suivante.

La variable temps, qui va de T à 0 est découpée en n parties égales $\Delta t = T/n$. On suppose que le prix S du titre sous-jacent peut varier entre 0 et $S_{\max}$. Cet intervalle est découpé en m parties égales $\Delta S = S_{\max}/m$. Il en résulte une grille avec $(n+1)(m+1)$ noeuds telle que illustrée dans la figure 3.

A partir du modèle de Black and Scholes[10], il est possible d'établir la relation suivante

$$a_j\,f_{i-1,j-1} + b_j\,f_{i-1,j} + c_j\,f_{i-1,j+1} = f_{i,j} \tag{34}$$

[10] Hull, J.C., (1993) : Options, Futures, and other Derivative Securities. Prentice Hall, page 355.

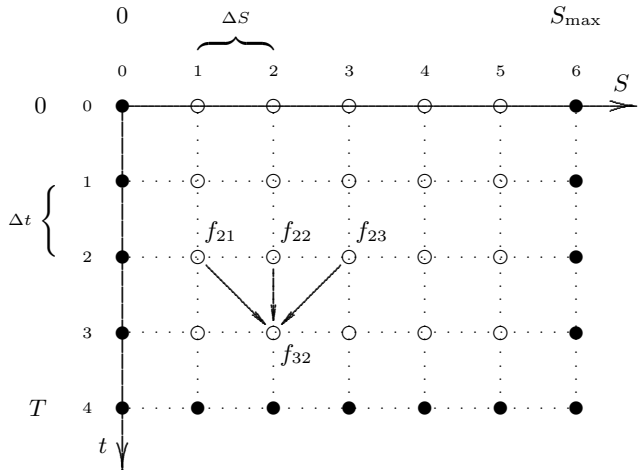


FIG. 3 – Grille pour $n = 4$ et $m = 6$.

pour $i = 0, 1, \ldots, n-1$ et $j = 1, 2, \ldots, m-1$ entre les différents noeuds.

Les coefficients a_j, b_j et c_j pour $j = 1, 2, \ldots, m-1$ sont définis comme suit :

$$\begin{aligned} a_j &= \frac{1}{2} r j \Delta t - \frac{1}{2} \sigma^2 j^2 \Delta t \\ b_j &= 1 + \sigma^2 j^2 \Delta t + r \Delta t \\ c_j &= -\frac{1}{2} r j \Delta t - \frac{1}{2} \sigma^2 j^2 \Delta t \end{aligned}$$

où r est le taux d'intérêt annuel hors risque et σ la volatilité sous-jacente du titre.

On vérifie facilement que pour $n = 4$ et $m = 6$ les équations (34) définissent le système

$$\left[\begin{array}{cccc|c} a_1 & b_1 & c_1 & & & \\ & a_2 & b_2 & c_2 & & \\ & & a_3 & b_3 & c_3 & \\ & & & a_4 & b_4 & c_4 \\ & & & & a_5 & b_5 \end{array}\right] \left[\begin{array}{cccc} f_{00} & f_{10} & f_{20} & f_{30} \\ \hline f_{01} & f_{11} & f_{21} & f_{31} \\ f_{02} & f_{12} & f_{22} & f_{32} \\ f_{03} & f_{13} & f_{23} & f_{33} \\ f_{04} & f_{14} & f_{24} & f_{34} \\ f_{05} & f_{15} & f_{25} & f_{35} \\ \hline f_{06} & f_{16} & f_{26} & f_{36} \end{array}\right] = \left[\begin{array}{cccc} f_{11} & f_{21} & f_{31} & f_{41} \\ f_{12} & f_{22} & f_{32} & f_{42} \\ f_{13} & f_{23} & f_{33} & f_{43} \\ f_{14} & f_{24} & f_{34} & f_{44} \\ f_{15} & f_{25} & f_{35} & f_{45} \end{array}\right] \tag{35}$$

Pour pouvoir résoudre le système (35), il faut connaître les valeurs des noeuds aux limites de la grille (noeuds dessinés en noir). Pour notre problème, ces conditions aux limites sont

$$\begin{aligned} f_{i0} &= 0 \quad i = 0, 1, \ldots, n && (36) \\ f_{im} &= 0 \quad i = 0, 1, \ldots, n && (37) \\ f_{nj} &= \max(X - j\Delta S, 0) \quad j = 1, \ldots, m-1 && (38) \end{aligned}$$

avec X le prix d'exercice du titre. Etant donné (36) et (37), le système (35) se simplifie et on a

$$\underbrace{\left[\begin{array}{cccc} b_1 & c_1 & & \\ a_2 & b_2 & c_2 & \\ & a_3 & b_3 & c_3 \\ & & a_4 & b_4 & c_4 \\ & & & a_5 & b_5 \end{array}\right]}_{P} \left[\begin{array}{cccc} f_{01} & f_{11} & f_{21} & f_{31} \\ f_{02} & f_{12} & f_{22} & f_{32} \\ f_{03} & f_{13} & f_{23} & f_{33} \\ f_{04} & f_{14} & f_{24} & f_{34} \\ f_{05} & f_{15} & f_{25} & f_{35} \end{array}\right] = \left[\begin{array}{cccc} f_{11} & f_{21} & f_{31} & f_{41} \\ f_{12} & f_{22} & f_{32} & f_{42} \\ f_{13} & f_{23} & f_{33} & f_{43} \\ f_{14} & f_{24} & f_{34} & f_{44} \\ f_{15} & f_{25} & f_{35} & f_{45} \end{array}\right] . \tag{39}$$

Etant donné que la dernière colonne de la matrice du membre de droite est définie par la condition (38), le système (39) peut être résolu en cherchant la solution de

$$P f_{ij} = f_{i+1,j} \qquad j = 1, \ldots, m-1 \tag{40}$$

pour $i = 3, 2, 1, 0$, dans l'ordre.

Application

On désire appliquer cette méthode au problème défini par les données suivantes :

Prix d'exercice du titre $X = 50$
Temps restant jusqu'à la maturité du titre $T = 5/12$
Taux d'intérêt annuel hors risque $r = 0.1$
Volatilité du titre $\sigma = 0.4$

On suppose $S_{\max} = 100$ le prix maximum que le titre sous-jacent peut atteindre et pour lequel le prix de l'option est nul. Pour le découpage temporel on choisit $n = 10$, et $m = 20$ pour le découpage du prix du titre.

1. Ecrire les instructions pour évaluer la matrice P. (L'assemblage de la matrice P peut se faire facilement, à l'aide de la fonction `diag`.)
2. Initialiser avec des zéros la matrice f qui représente les noeuds de la grille. (Etant donné que la valeur des noeuds de la première et la dernière colonne de la grille est nulle, la matrice f peut être définie comme ayant $n+1$ lignes et $m-1$ colonnes.)
3. Initialiser la dernière ligne de f avec les conditions données sous (38).
4. Résoudre le système d'équations (40) en se servant de la commande \ de Matlab. La théorie économique dicte que à chaque étape $i = 3, 2, 1, 0$, la solution f_{ij}, $j = 1, 2, \ldots, m-1$ du système linéaire doit être corrigée comme suit
$$f_{ij} = \max(f_{ij},\ f_{n+1,j}) \qquad j = 1, 2, \ldots, m-1.$$
Ceci s'interprète comme suit : à aucun instant le prix de l'option ne peut être inférieur à $\max(X - S, 0)$.

La matrice P a une structure particulière étant donné que $p_{ij} = 0$ pour $|i - j| > 1$. Une telle matrice est appelée une matrice par bande, et dans notre cas, il s'agit du cas particulier d'une matrice tridiagonale. On vérifie aisément que les matrices L et U correspondant à la factorisation d'une matrice tridiagonale T sont de la forme

$$\left[\begin{array}{cccc} 1 & & & \\ l_1 & 1 & & \\ & l_2 & 1 & \\ & & l_3 & 1 \\ & & & l_4 & 1 \end{array}\right] \left[\begin{array}{cccc} u_1 & q_1 & & \\ & u_2 & q_2 & \\ & & u_3 & q_3 \\ & & & u_4 & q_4 \\ & & & & u_5 \end{array}\right] = \left[\begin{array}{cccc} d_1 & q_1 & & \\ p_1 & d_2 & q_2 & \\ & p_2 & d_3 & q_3 \\ & & p_3 & d_4 & q_4 \\ & & & p_4 & d_5 \end{array}\right] \tag{41}$$

avec

$$\begin{aligned} u_1 &= d_1 && \\ l_i &= p_i / u_i && i = 1, \ldots, n-1 \\ u_i &= d_i - l_{i-1} q_{i-1} && i = 2, \ldots, n. \end{aligned}$$

Ainsi il apparaît que cette factorisation ne nécessite que le calcul de deux vecteurs l et u.

4. Programmer une fonction Matlab `lu3diag` qui calcule les vecteurs l et u en fonction des vecteurs p, d et q qui définissent la matrice tridiagonale T. On donnera à cette fonction la structure suivante
$$[\mathtt{l},\mathtt{u}] = \mathtt{lu3diag}(\mathtt{p},\mathtt{d},\mathtt{q})$$
5. Quel est le nombre d'opérations élémentaires que nécessite l'exécution de la fonction `lu3diag` ? (Donner une expression en fonction de la taille de T.)
6. Sous quelles conditions l'algorithme programmé dans `lu3diag` est-il numériquement stable ?
7. Programmer une fonction Matlab `res3diag` qui résoud un système linéaire tridiagonal $Tx = b$ par *forward* et *backward substitution*, la factorisation LU étant donnée (vecteurs l, u et q). On donnera à cette fonction la structure
$$\mathtt{b} = \mathtt{res3diag}(\mathtt{l},\mathtt{u},\mathtt{q},\mathtt{b})$$
8. Quel est le nombre d'opérations élémentaires que nécessite l'exécution de la fonction `res3diag` ? (Donner une expression en fonction de la taille du système.)
9. Evaluer la matrice f à l'aide des fonctions `lu3diag` et `res3diag`.
10. Quel est le nombre d'opérations élémentaires que nécessite l'évaluation de la matrice f lorsqu'on utilise les fonctions `lu3diag` et `res3diag` ? Donner une expression en fonction du découpage défini par m et n.

10.4 AsianOpt

Ci-après le code Matlab, présenté au cours, pour l'évaluation d'un call Européen avec la méthode de Monte Carlo.

Code /Teaching/MNE/Ex/MC00.m:

```
function [P,IC] = MC00(S0,E,r,T,sigma,NIncr,NRepl)
% Evaluation d'un call Europeen avec Monte Carlo (brut)
dt = T/NIncr;
trend = (r - sigma^2/2)*dt;
sigdt = sigma * sqrt(dt);
S1 = 0;
S2 = 0;
for j = 1:NRepl
   z = log(S0);
   for i = 1:NIncr
      z = z + trend + sigdt * randn;
   end
   Payoff = max( (exp(z)-E), 0);
   S1 = S1 + Payoff;
   S2 = S2 + Payoff^2;
end
P  = exp(-r*T) * S1 / NRepl;
SE = ( exp(-r*T) * sqrt( (S2 - S1^2/NRepl) )/(NRepl-1) );
IC(1) = P - 1.96*SE;
IC(2) = P + 1.96*SE;
```

Dans ce code les `NIncr` incréments du mouvement brownien géométrique sont programmés à l'aide d'une boucle `for`. Etant donnée que Matlab interpréte les instructions dans une boucle à chaque passage, l'éxécution d'un tel code est relativement lent comparé à l'éxécution d'instructions vectorielles qui font appel à un code compilé lors de l'exécution. La fonction qui suit génère, sans faire appel à une boucle `for`, une matrice dont chaque ligne correspond à une trajectoire.

Code /Teaching/MNE/Ex/MBG.m:

```
function S = MBG(S0,r,T,sigma,NIncr,NRepl)
% Mouvement Brownien Geometrique
% Simulation de NRepl trajectoires de prix avec NIncr increments
% Chaque trajectoire a NIncr+1 points (S(1)=S0).
dt = T/NIncr;
trend = (r - sigma^2/2)*dt;
sigdt = sigma * sqrt(dt);
Increments    = trend + sigdt*randn(NRepl,NIncr);
LogTrajectoires = cumsum([repmat(log(S0),NRepl,1) Increments],2);
S = exp(LogTrajectoires); % Trajectoires du prix
```

Les répétitions `for` sont évitées en créant d'abord une matrice d'incréments à laquelle on rajoute une première colonne avec le prix initial. A cette matrice on applique la commande `cumsum` qui calcule la somme cumulée des éléments de chaque ligne, ce qui constitue les trajectoires browniennes.

On se propose maintenant de réecrire le code qui évalue un call Européen en utilisant la fonction `MBG` pour la simulation des trajetoires de prix. Voici ce code revu :

Code /Teaching/MNE/Ex/MC01.m:

```
function [P,IC] = MC01(S0,E,r,T,sigma,NIncr,NRepl)
% Evaluation d'un call Europeen avec Monte Carlo (brut)
S = MBGV(S0,r,T,sigma,NIncr,NRepl);
Payoff = max( (S(:,end)-E), 0);
[P,ignore,IC] = normfit( exp(-r*T) * Payoff );
```

On utilise ici la fonction Matlab `normfit` pour caluler la moyenne des *payoffs* escomptés ainsi qu'un intervalle de confiance. Le deuxième argument de retour de la fonction `normfit` contient l'ecart-type, que nous n'utilisons pas dans la suite des calculs (c'est la raison pour laquelle nous l'avons nommé `ignore`). Le troisième argument de sortie est un vecteur de deux éléments qui contient les valeurs qui définissent l'intervalle de confiance à 95%.

On désire maintenant comparer la performance de ces codes en calculant le prix d'un call Européen correspondant à un prix du sous-jacent de $S0 = 120$, un prix d'exercice de $E = 130$, un taux du marché de $r = 0.05$, une échéance à 9 mois et une volatilité de $\sigma = 0.35$.

1. Calculer le prix du call avec la formule analytique de Black-Scholes.
2. Calculer le prix du call avec la fonction `MC00` en choisissant un nombre d'incréments de `NIncr=100` et un nombre de réplications de `NRepl=10000`. Comme on désire pouvoir comparer les résultats on initialisera le générateur des nombres au hasard et on mesurera le temps d'éxécution. Pour ce faire on procédera comme :
   ```
   randn('seed',0);
   t0 = clock;
   [P,IC] = MC00( ... );
   dispMC('MC00',P,IC,t0)
   ```
   La fonction `dispMC` permet l'impression organisé des résultats et se trouve sur la disquette fournie.
3. Calculer le prix du call avec la fonction `MC01` en choisissant les mêmes valeurs pour `NIncr` et `NRepl` que précédemment. Comme avant on initialisera le générateur des nombres au hasard et on imprimera les résultats à l'aide de `dispMC`.
4. Pour le calcul du *payoff* il n'est pas necessaire de connaître toute la trajectoire des prix. Le prix final suffit et de ce fait on peut exécuter `MC01` en posant `NIncr=1`. Quelle est le temps d'exécution dans ce cas.
5. Le gain de temps obtenu en posant `NIncr=1` permet d'augmenter le nombre de réplications. Multiplier la valeur précédente de `NRepl` par 100 et exécuter `MC01`.
6. Commenter les résultats.

On sait que l'écart-type des prix calculés avec la méthode de Monte Carlo est proportionelle à $1/\sqrt{\texttt{NRepl}}$. On désire vérifier empiriquement ce fait.

7. Programmer la procédure suggérée ci-après :

   Initialiser `ndeb` $= 1000$; `nfin` $= 100000$; `incr` $= 10000$;
   Generer le vecteur `NReplvec` $=$ `ndeb :incr :nfin` ;
   pour $i = 1$: `length(NReplvec)` **faire**
   Avec `MC01` calculer c_i (largeur de l'intervalle de confiance)
   finfaire
   `n = linspace(ndeb,nfin) ;`
   Calculer le facteur de proportionalité $p = c_1\sqrt{n_1}$
   Calculer l'intervalle théorique $cth_i = p/\sqrt{n_i}$
   Plotter c et cth `plot(NReplvec,c,'r*',n,cth,' :')`

8. Ecrire la fonction `[P,IC] = MCAT(S0,E,r,T,sigma,NIncr,NRepl)` qui implémente la technique des variables antithétiques. On procédera en deux etapes : D'abord on construit `[S1,S2] = MBGAT(S0,r,T,sigma,NIncr,NRepl)` qui est une nouvelle version de la fonction qui génère les trajectoires browniennes, où la ligne i de la matrice `S2` contient la trajectoire de prix qui est parfaitement corrélée avec la trajectoire contenu dans la ligne i de la matrice `S1`. Les instructions qui construisent ces deux matrices sont données ci-après :

```
Dif   = sigdt*randn(NRepl,NIncr);
Increments1 = trend + Dif;
Increments2 = trend - Dif;
LogTrajectoires1 = cumsum([log(S0)*ones(NRepl,1) Increments1],2);
LogTrajectoires2 = cumsum([log(S0)*ones(NRepl,1) Increments2],2);
S1 = exp(LogTrajectoires1);
S2 = exp(LogTrajectoires2);
```

   La fonction `MCAT` aura alors la structure de la fonction `MC01` dans laquelle on aura remplacé l'appel à `MBG` par un appel à `MBGAT` et dont le vecteur du payoff sera calculé comme :

```
Payoff = ( max( (S1(:,end)-E), 0) + max( (S2(:,end)-E), 0) ) / 2;
```

   Calculer le prix du call avec la fonction `MCAT` en choisissant les mêmes valeurs pour `NIncr` et `NRepl` que précédemment. Commenter le résultat.

9. Est-ce-que l'écart-type des prix calculés avec variables antithétiques est aussi proportionelle à $1/\sqrt{\texttt{NRepl}}$?

10. Quel est le nombre de replications `NRepl` nécessaires si l'on veut obtenir un intervalle de confiance de 0.01.

On considère maintenant un option Asiatique dont le *payoff* dépend de la moyenne discrète du sousjacent. Plus précisément le payoff est défini comme

$$\frac{1}{N}\sum_{i=1}^{N} S(t_i) - E\Bigg)_{+} \tag{42}$$

où $t_i = i\ \triangle_t$ avec $\triangle_t = T/N$. Ainsi il s'agit de considérer une moyenne sur N valeurs du sousjacent, le nombre N faisant partie de la définition de l'option.

11. Ecrire la fonction `[P,IC] = AsianMC(S0,E,r,T,sigma,N,NRepl)` qui implémente ce payoff. La structure de la procédure est la suivante :

    pour $i = 1$: `NRepl` **faire**
    Calculer avec MBG une trajectoire avec `N` incréments
    Calculer le `Payoff` comme défini en (42)
    finfaire
    Avec `normfit` calculer la moyenne de `Payoff` et un intervalle de confiance

12. Calculer le prix du call de l'option Asiatique en choisissant $N = 5$. Les autres paramètres sont ceux de l'exemple précédent.

10.5 BarrierOpt

Rappellons l'équation différentielle partielle

$$V_{\mathrm{t}} + \frac{1}{2}\sigma^2 S^2 V_{\mathrm{SS}} + (r-q)S\,V_{\mathrm{S}} - r\,V = 0$$

et sa discrétisation aux noeud de la ligne i et la colonne j de la grille aux différences finies

$$\begin{aligned}&\frac{v_{i,j+1} - v_{ij}}{\triangle_t} + \\ &\quad\frac{1}{2}\sigma^2 S_i^2\,\frac{v_{i+1,j} - 2v_{ij} + v_{i-1,j}}{\triangle_S^2} + \\ &\quad r\,S_i\,\frac{v_{i+1,j} - v_{i-1,j}}{2\,\triangle_S} - r\,v_{ij} = 0\,.\end{aligned}$$

Lorsque $S_0 = 0$ on substitue souvent S_i par $S_i = i\ \triangle_S$ pour $i = 0, 1, \ldots, N$ et $\triangle_S = (S_N - S_0)/N$. Si l'on veut aussi pouvoir considérer des valeurs positives pour S_0 il convient cependant de procéder à la substitition suivante $s_i = S_i/\ \triangle_S$

$$\begin{aligned}v_{i,j+1} \;=\; & -\frac{1}{2}\sigma^2\, s_i^2\,\triangle_t\ v_{i-1,j} + \frac{1}{2} r\, s_i\,\triangle_t\ v_{i-1,j} \\ & - v_{ij} + \sigma^2\, s_i^2\,\triangle_t\ v_{ij} + r v_{ij} \\ & - \frac{1}{2}\sigma^2\, s_i^2\,\triangle_t\ v_{i+1,j} - \frac{1}{2} r\, s_i\,\triangle_t\ v_{i+1,j}\,.\end{aligned}$$

En regroupant les variables on obtient l'expression

$$v_{i,j+1} = v_{ij} - a_i v_{i-1,j} - b_i v_{ij} - c_i v_{i+1,j}$$

où les coefficient sont définies comme

$$\begin{aligned}a_i &= \triangle_t\,(\sigma^2 s_i^2 - r s_i)/2 \\ b_i &= -\triangle_t\,(\sigma^2 s_i^2 + r) \\ c_i &= \triangle_t\,(\sigma^2 s_i^2 + r s_i)/2\,.\end{aligned}$$

1. En vue d'une structuration plus claire de l'implémentation des algorithmes basés sur les méthodes aux différences finies on se propose d'écrire une fonction Matlab `FD1A` qui calcule les matrices de coefficients de la méthode theta. On suggère d'écrire une fonction qui se presente comme :

```
function [Am,Ap,S,dt,a1,cN] = FD1A(r,T,sigma,S0,Smax,M,N,t
%
f7 = 1;
dt = T / M;
S  = linspace(S0,Smax,N+1);
s  = S(f7+(1:N-1)) / (S(2)-S(1));
```

   Calculer les vecteurs de coefficients `a, b` et `c` et les matrices `Am` et `Ap`

```
a1 = a(1);
cN = c(N-1);
```

   Remarquez que les vecteurs a, b et c peuvent être calculés sans recours aux boucles `for`. Le calcul des matrices A_m et A_p est identique à ce qui a été programmé dans les procédures vues au cours.

2. Modifiez la fonction `EuPutCrNi` présentée au cours afin d'utiliser la fonction `FD1A` et appellez la fonction modifiée `EuPutCN`.

3. Tester la fonction `EuPutCN` en calculant le prix d'un put Européen pour $S = 50$, $E = 50$, $r = 0.10$, $T = 5/12$ et $\sigma = 0.40$. Pour les paramètres qui sont spécifiques à la méthode numérique on choisira $S_{\max} = 100$, $M = 200$ et $N = 500$.

On considère maintenant des options *barrière*. Dans ce cas le prix de l'option dépend du fait que le sous-jacent S traverse (ou ne traverse pas) une certaine *barrière* S_b durant le temps qui s'écoule jusqu'à l'exercice. Dans le cas ou le prix initial S_0 du sous-jacent vérifie $S_b < S_0$ on parle de *up options* et dans le cas contraire ($S_b > S_0$) on parle de *down options*.

Nous considérons un *down and out put* qui est une option put qui cesse d'exister si le prix du sous-jacent descend en dessous de la barrière S_b. Pour une telle option on a $S_b < S_0$ et $S_b < E$ avec E le prix d'exercice. Le prix d'une telle option peut être calculé avec une méthode aux différences finies en ne considérant que le domaine $S > S_b$ avec les conditions aux bords

$$V(S_b, t) = 0 \qquad \text{et} \qquad \lim_{S\to\infty} V(S,t) = 0$$

et les conditions terminales

$$V(S,T) = (E-S)_+ .$$

4. Ecrire une fonction `DOPutCN` qui implémente la méthode aux différences finies pour le calcul du prix de l'option *down and out put*.
   Indications : Il suffira de modifier `EuPutCN` comme suit : a) compléter les argument d'entrée avec S_b ; b) enlever le calcul de la condition au bord à chaque pas temporel (cette condition vaut maintenant zéro et a déjà été initialisée avec la commande `V = zeros(N+1,M+1)`) ; c) la correction de `z(1)` est inutile étant donné que $V_{0j} = 0$.
5. Calculer le prix de l'option pour $S_b = 40$. Pour les autres paramètres on utilisera les valeurs données ci-dessus.
6. Dans un même graphique représenter le prix du put Européen et du *down and out put*.
   Indication : Utiliser les commandes `hold on` pour tracer les courbes successives, et redimensionner la fenêtre graphique avec `axis([30 110 0 3])`.
7. Augmenter la volatilité en posant $\sigma = 0.50$. Recalculer les prix du put Européen et du *down and out put* et compléter le graphique précédent avec les nouveaux prix. Commenter.
8. Il est connu[11] que dans certaines conditions la méthode de Crank-Nicolson produit des solutions qui oscillent et que dans ces cas une méthode implicite est preférable. Illustrer ce phénomène en choisissant $\sigma = 0.40$, $S_b = 47$ et $S_{\max} = 80$.
   Indication : A partir de `DOPutCN` construire `DOPutIm`. La seule modification consiste à modifier la valeur de `theta`. Redimensionner la fenêtre graphique avec `axis([49 51 4*1e-3 6*1e-3])`.

10.6 OptionPricing00

Le 28 mai 2004 le prix d'un call Européen sur une action UBS était coté 6 SFr alors que l'action s'échangeait à 89.90 SFr. Le prix d'exercise est de 87.50 SFr et le call expire le 17 septembre 2004. On sait que la volatilité moyenne σ est de 0.239 mais on ignore quel est le taux d'intérêt du marché r.

1. Déterminer le taux r en faisant l'hypothèse que le call correspond au modèle de Black-Scholes.
   Suggestion : On peut établir le taux approximativement en traçant l'évolution du prix du call en fonction de r et relever la valeur de r qui correspond à un prix de 6 SFr (les autres paramètres restant constants).
2. On admets maintenant que $r = 0.0064$. Recalculer le prix Black-Scholes du call avec cette valeur de r.
3. Calculer le prix du call avec la méthode de Crank-Nicolson.
4. Calculer le prix du call avec la méthode de Monte-Carlo.
5. Calculer le prix du call avec la méthode de Monte-Carlo en applicant la méthode de réduction de variance avec variables antithétiques.
6. Calculer le prix du call en utilisant le modèle binomial multiplicatif. On rappelle que le coefficient du mouvement à la hausse u et le coefficient du mouvement à la baisse d sont définies comme

$$u = \exp(\sigma\sqrt{\triangle t}) \qquad d = 1/u$$

On considère maintenant des *options barrières* qui ont pour caractéristique que le payoff prend la valeur zéro lorsque le sous-jacent traverse un certain niveau défini au préalable.

Une option *Up-and-out* call a un payoff qui vaut zéro lorsque le sous-jacent traverse une barrière prédefinie $B > S_{t=0}$ à un moment $t \in [0,T]$. Si la barrière n'est pas traversé le payoff correspond au payoff d'un call Européen, c'est-à-dire $(S_T - E)_+$.

7. Quelle méthode proposez vous utiliser pour le calcul d'un call *up-and-out*. (Proposer la méthode qui s'adapte le plus facilement à ce problème).
8. Calculer le prix d'un call *up-and-out* avec une barrière $S_b = 110$, les autres paramètres sont ceux du call Européen introduit plus haut.
9. Dans un même graphique tracer le prix du call Européen et le prix du call *up-and-out* pour $S \in [70, 115]$.

10.7 mnf-cc1

1. Expliquer brièvement en quoi consiste une méthode numérique et quel est son intérêt pratique.
2. On considère les méthodes aux différences finies pour la solution d'équations différentielles partielles. Expliquer brièvement le principe de ces méthodes, leurs avantages et leurs inconvénients.
3. Soit l'équation de Black-Scholes et les conditions aux limites pour un put européen $P(S,t)$:

$$\frac{\partial P}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 P}{\partial S^2} + rS\frac{\partial P}{\partial S} - rP = 0$$

   avec

$$P(0,t) = Ee^{-r(T-t)} \qquad \lim_{S\to\infty} P(S,t) = 0 \qquad P(S,T) = (E-S)_+$$

   (a) De quel type d'équation différentielle s'agit-il ?
   (b) De quelles façons, numériques et analytiques, peut-on envisager sa solution ? (Expliquer sans présenter les développements mathématiques.)
   (c) Quelles critiques peut-on formuler quant à l'application directe d'une méthode numérique à l'équation de Black-Scholes.
4. On considère maintenant la méthode explicite pour le calcul du prix du put d'une option européenne à 3 mois d'échéance avec un prix d'exercice de 20, un taux de marché de 6% et une volatilité sous-jacente de 0.4.
   (a) Quelle valeur de $\delta\tau$ est optimale pour la précision de l'approximation de la dérivée par rapport au temps. On indiquera l'ordre p, c'est-à-dire l'exposant de $\delta\tau = 10^{-p}$, dans un environnement MATLAB sur PC. Qu'en est-il de la valeur du pas pour le calcul de la dérivée par différence centrée ? Commenter.

[11] Voir Zvan, R., K.R. Vetzal et P.A. Forsyth (2000) : PDE methods for pricing barrier options, *Journal of Economic Dynamics and Control* 24, 1563–1590.

(b) On désire maintenant calculer le prix du put qui correspond à une valeur du sous-jacent de 25. En utilisant la valeur de $\delta\tau$ retenue au point précédent, définir la dimension de la grille aux différences, de sorte que ce prix calculé ne dépende pas des conditions aux limites. (On ne demande pas de calculer le prix.)

10.8 mnf-cc2

Soit l'équation de Black-Scholes

$$\frac{\partial V}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} + rS\frac{\partial V}{\partial S} - rV = 0.$$

Ecrire cette équation après les changements de variables suivants :

$$\begin{aligned} x &= \log(S/E) \\ \tau &= T - t \\ v(x,\tau) &= V/E. \end{aligned}$$

Discrétiser l'équation différentielle obtenue en remplaçant $\partial v/\partial\tau$ et $\partial v/\partial x$ par une différence progressive et $\partial^2 v/\partial x^2$ par une différence centrée.

Définir une grille de différences finies en choisissant N pas temporels. Pour la variable x on centrera la grille au point $x = 0$, puis on choisira une borne inférieure $-L$, une borne supérieure de L et un nombre de pas M dans la direction positive et dans la direction négative.

Ecrire et implémenter l'algorithme qui calcule soit un call, soit un put pour une option européenne. Pour l'application on choisira $r = 0.05$, $\sigma = 0.2$, $T = 0.5$, $E = 10$, $L = 1$, $M = 12$ et $N = 25$.

Comparer les résultats obtenus avec la solution analytique pour l'équation de Black-Scholes.

10.9 mnf-cc3

Soit l'équation de Black-Scholes

$$\frac{\partial V}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} + rS\frac{\partial V}{\partial S} - rV = 0.$$

Ecrire cette équation après les changements de variables suivants :

$$\begin{aligned} x &= \log(S/E) \\ \tau &= T - t \\ v(x,\tau) &= V/E. \end{aligned}$$

Discrétiser l'équation différentielle de sorte qu'elle puisse être résolue avec la méthode dite $\theta-$method.

Définir une grille de différences finies en choisissant N pas temporels. Pour la variable x on centrera la grille au point $x = 0$, puis on choisira une borne inférieure $-L$, une borne supérieure de L et un nombre de pas M dans la direction positive et dans la direction négative.

Ecrire et implémenter la méthode dite $\theta-$*method* qui calcule soit un call, soit un put pour une option européenne. Pour l'application on choisira $r = 0.05$, $\sigma = 0.2$, $T = 0.5$, $E = 10$, $L = 1$, $M = 12$ et $N = 25$.

Exécuter l'algorithme pour les différentes valeurs de θ qui caractérisent soit la méthode explicite, soit la méthode implicite ou celle de Crank-Nicolson.

Comparer les résultats obtenus avec la solution analytique pour l'équation de Black-Scholes.

10.10 compare-FDM

Comparaison des méthodes aux différences finies

1. Enumerer les principales méthodes pour la résolution numérique d'équations différentielles partielles.
2. Pour chaque méthode indiquer brievement le principe, les avantages et les inconvenients.
3. Donner pour chaque méthode le nombre d'opérations élémentaires en fonction du nombre de pas temporels M et du nombre des pas pour la discretisation du sous-jacent. Detailler votre réponse.

Sur la disquette jointe vous trouvez une procédure Matlab `CompareFDM`. Cette procédure fait appel à des fonctions qui calculent le prix d'une option Européenne suivant la méthode explicite, implicite et la méthode de Crank-Nicolson pour ensuite comparer la précision de ces méthodes avec la solution analytique de Black-Scholes.

1. Fournir la fonction pour le calcul de la solution analytique et coder son appel dans la procédure `CompareFDM`.
2. Pour $N = 100$, $M = 30$ et $I = 3$ comparer la précision des trois méthodes. Commneter.
3. Augmenter M ($M = 500$) et observer le comportement des méthodes. Commneter.
4. Pour $M = 30$ et $N = 200$ commneter les résultats.
5. Poser $\sigma = .4$ en gardant M et N comme avant.
6. On désire maintenant que la solution ne s'écarte pas plus que de 0.001 de la solution analytique. Choisir des valeurs de N, M restant à 30, pour obtenir cette précision avec au moins une des méthodes. Commenter.
7. Peut-on atteindre une précision de .0001 ? Commneter.
8. Donner un commnetaire général sur cet exercice.

10.11 FDMCiba

On considère le call Européen sur l'action CIBN qui expire le 17 juin 2005 (64 jours) avec un prix d'exercise de 80. Sachant que le prix actuel de CIBN est de 81 et en considérant un taux du marche de $r = 0.005$ et une volatilité de $\sigma = 0.13$, calculer le prix du call avec le modèle de Black-Scholes en utilisant la méthode :

- analytique,
- explicite,
- implicite,
- Crank-Nicolson.

Comparer dans un même graphique les prix obtenus avec les trois méthodes numériques avec la solution analytique.

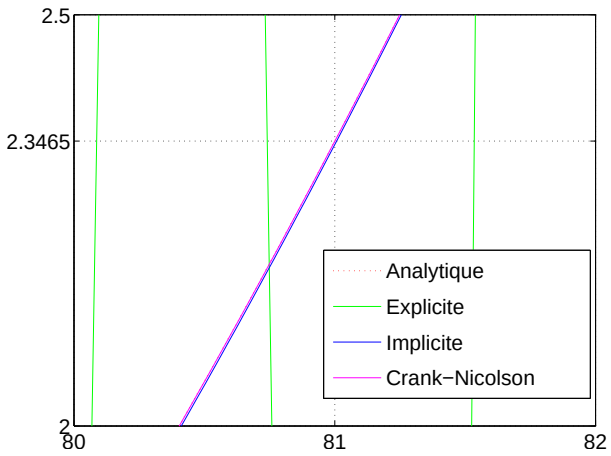


FIG. 4 – Exemple de presentation graphique des résultats.

10.12 MCBasketPricing

On considère une option européenne sur n sousjacents dont le gain P au temps d'expiration T, aussi appelé *payoff*, est définie comme

$$P = \max \quad E - \sum_{i=1}^{n} S_i^{(T)} w_i, 0$$

avec E le prix d'exercise, $S_i^{(T)}$, $i = 1, 2, 3$ les prix des sousjacents au temps T et w_i, $i = 1, 2, 3$ des poids donnés. Une telle option est appelée un *put* et permet de couvrir les pertes lorsque la valeur de la combinaison pondérée des trois sousjacents, appelé le *basket*, tombe en dessous de E.

On peut évaluer le prix d'une telle option avec la méthode de Monte Carlo en simulant les prix des sousjacents par un mouvement brownien géométrique. A partir d'un vecteur des prix donné S_0 au temps 0, et en divisant l'intervalle de temps $[0, T]$ en M pas de longueur $\triangle_t = T/M$ on peut simuler des trajectoires de prix avec la relation

$$S^{(i\triangle_t)} = \text{diag}(S^{((i-1)\triangle_t)}) \exp \quad v\triangle_t + \sqrt{\triangle_t}\, z \qquad i = 1, 2, \ldots, N \tag{43}$$

avec

$$v = r - q - \text{diag}(\Sigma)/2 = r - \begin{bmatrix} q_1 \\ q_2 \\ \vdots \\ q_n \end{bmatrix} - \begin{bmatrix} \sigma_1^2/2 \\ \sigma_2^2/2 \\ \vdots \\ \sigma_n^2/2 \end{bmatrix}$$

et ou r est le taux du marché, q le taux de dividende, z est un vecteur aléatoire distribué comme $N(0, \Sigma)$ et ou Σ est la matrice des variances et covariances des prix S. L'opérateur diag transforme un vecteur en une matrice diagonale. Lorsqu'on l'applique à une matrice il construit un vecteur avec les éléments de la matrice.

En remplaçant dans (43) $\triangle_t$ par T on peut directement simuler les prix $S^{(T)}$ à partir de $S^{(0)}$ en un seul pas

$$S^{(T)} = \text{diag}(S^{(0)}) \exp \quad vT + \sqrt{T}\, z \quad . \tag{44}$$

On simule alors N vecteurs $S^{(T)}$ et on calcule les *payoffs* correspondants et on escompte leur valeur à l'instant $t = 0$. La moyenne des ces valeurs actualisés constitue alors le prix de l'option. L'algorithme 7 résume la méthode de Monte Carlo pour l'évaluation d'une option *basket* européenne sur n actifs.

Algorithme 7 Méthode de Monte Carlo pour option *basket* européenne.

Initialiser $S_0 \in \mathbb{R}^n$, w, E, T, Σ, r, et N
$v = r - q - \text{diag}(\Sigma)/2$
Calculer $U'U = \Sigma$ (décomposition de Cholesky)
pour $i = 1 : N$ **faire**
  Générer $\epsilon \sim N(0, I_n)$ et calculer $z = U'\epsilon$
  $S^T = \text{diag}(S^0) \exp\left(v\,T + \sqrt{T}\, z\right)$
  $P = \max\left(E - \sum_j S_j^T w_j, 0\right)$ (put)
  $p_i = \exp(-rT)\, P$
finfaire
$\bar{p} = \frac{1}{N} \sum_{i=1}^{N} p_i$
(Calculer un intervalle de confiance pour $\bar{p}$)

1. Calculer le prix d'une option *put* pour un *basket* de trois sousjacents avec :

$$S^{(0)} = \begin{bmatrix} 80 \\ 100 \\ 50 \end{bmatrix}, \qquad w = \begin{bmatrix} 0.38 \\ 0.22 \\ 0.40 \end{bmatrix}, \qquad \sigma = \begin{bmatrix} 0.25 \\ 0.35 \\ 0.20 \end{bmatrix}, \qquad q$$

$\rho_{21} = -0.65$, $\rho_{31} = 0.25$, $\rho_{32} = 0.50$, $r = 0.045$, $T = 1$ et $E = 70$. On rappelle que la matrice Σ est définie comme

$$\begin{bmatrix} \sigma_1^2 & \rho_{21}\sigma_1\sigma_2 & \rho_{31}\sigma_1\sigma_3 \\ \rho_{21}\sigma_2\sigma_1 & \sigma_2^2 & \rho_{32}\sigma_2\sigma_3 \\ \rho_{31}\sigma_3\sigma_1 & \rho_{32}\sigma_2\sigma_3 & \sigma_3^2 \end{bmatrix}.$$

On pourra choisir $N = 10^4$ pour le nombre des trajectoires simulées.

11 Simulation

11.1 simul00

La fonction Matlab `rand` génère des variables pseudo-aléatoires uniformement distribuées dans l'intervalle $(0, 1)$. Avec la commande

```
u = rand;
```

on génère une réalisation de cette variable aléatoire qui sera assignée à la variable `u`.

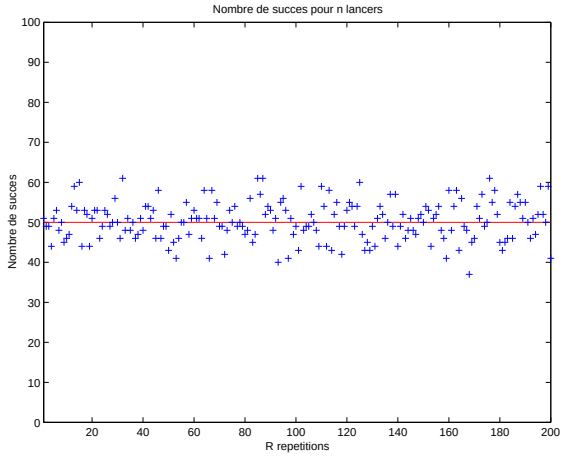
1. En utilisant une boucle `for` générer un vecteur `u` de 1000 éléments représentant chacun une réalisation de cette variable aléatoire. Faire un histogramme des valeurs générées.
2. On désire simuler le résultat d'un lancer de dés. On rappelle que chaque résultat possible, c'est-à-dire 1, 2, 3, 4, 5 et 6 a la même probabilité de 1/6 de se réaliser. Pour simuler le résultat x d'un lancer, on construira la procédure suivante :
   - Générer une réalisation u d'une variable aléatoire uniforme
   - Si $u < 1/6$, alors $x = 1$ et arrêt
   - Sinonsi $u < 2/6$, alors $x = 2$ et arrêt
   - Sinonsi $u < 3/6$, alors $x = 3$ et arrêt
   - Sinonsi $u < 4/6$, alors $x = 4$ et arrêt
   - Sinonsi $u < 5/6$, alors $x = 5$ et arrêt
   - Sinon $x = 6$

   Ecrire le programme qui traduit cette procédure. (On se servira des instructions `if`, `elseif` et `else`).

3. Reprendre le programme de la question précédente et le modifier de sorte que l'on puisse simuler un vecteur x de 1000 résultats.
4. Construire un vecteur f dont l'élément $f(i)$, $i = 1, 2, \ldots, 6$ contient le nombre de fois où l'on a obtenu $x = i$ dans la simulation précédente. Faire un diagramme en batons qui représente les éléments de f.
5. On se demande maintenant combien d'essais sont nécessaires, en moyenne, pour obtenir le résultat $x = 6$. Ecrire un programme qui simule la succession de lancers jusqu'à l'obtention de $x = 6$. (On se servira de l'instruction `while`).
6. Repéter 1000 fois la simulation précédente et calculer le nombre moyen de lancers.

11.2 simul01

1. On considère l'expérience où l'on compte le nombre de fois que l'on obtient "face" en lançant n fois une pièce de monnaie. On appellera S ce nombre. Ecrire une procédure qui calcule S en fonction de n.
2. On désire répéter l'expérience précédente R fois. Ecrire une procédure qui construit un vecteur S dont les R éléments contiennent les résultats des R répétitions de l'expérience. Choisir $n = 100$ et $R = 200$.
   Essayer de représenter les résultats avec un graphique semblable à celui qui suit :



3. Reprendre la procédure du point 2 et la compléter afin de calculer la moyenne

$$\texttt{ES} = \frac{1}{R}\sum_{i=1}^{R} S_i$$

du nombre de "faces" obtenu et l'écart quadratique moyen

$$\texttt{VS} = \frac{1}{R-1}\sum_{i=1}^{R} (S_i - \texttt{ES})^2$$

de ce nombre.
Transformer les observations S_i comme

$$V_i = \frac{S_i - \texttt{ES}}{\sqrt{\texttt{VS}}} \qquad i = 1, \ldots, R.$$

Faire un histogramme avec 9 classes qui représentent les éléments du vecteur V.

4. On considère maintenant l'expérience où l'on compte le nombre de lancers pour obtenir la première fois "face". On appellera n ce nombre. Ecrire une procédure qui calcule n.
5. On désire répéter l'expérience précédente N fois. Ecrire une procédure qui construit un vecteur R dont les N éléments contiennent les résultats des N répétitions de l'expérience. Choisir $N = 320$. Faire un histogramme des résultats.
   On sait que la probabilité d'obtenir "face" au cinqième lancer est égale à
   $$(1-p)^4 p$$
   avec $p = 1/2$ s'il s'agit d'une pièce honnête. Ainsi, en moyenne, on devrait observer théoriquement $N(1-p)^4 p = 10$ expériences pour lesquelles "face" a été obtenu la première fois au cinquième lancer.
   Vérifier empiriquement ce résultat.

11.3 VaMv

Le fichier `H:\input\RetFNS.mat` contient les rendements journaliers des indices FTSE, Nikkei et S&P500 observées sur la période du 1 avril 1986 au 15 décembre 2003. On peut lire ce fichier avec la commande `load` et voir les variables chargées avec `whos`. Pour la suite de l'exercise on ne considère que les deux premières colonnes de la matrice des rendements, c'est-à-dire celles qui correspondent aux rendements du FTSE et Nikkei (colonne 1 et 2 de la matrice R).

Pour répondre aux questions qui suivent on pourra se servir d'un script unique.

1. Calculer la moyenne $\overline{r} = [\overline{r}_x \ \overline{r}_y]$ des rendements et la matrice des variances covariances Q des ces rendements (Q sera calculée à partir d'un sous-ensemble de colonnes de R).
2. On considère maintenant la variable aléatoire normale multivarié $z \backsim N(\overline{r}, Q)$, $z = [x \ y]'$. Rappelons que
   $$S = (z - \overline{r})' \, Q^{-1} \, (z - \overline{r}) \backsim \chi^2_\nu$$
   et
   $$S - \chi^2_{\alpha,\nu} = 0$$
   définit l'intervalle de confiance simultané au seuil de α avec $\chi^2_{\alpha,\nu}$ le $1-\alpha$ quantile de la distribution χ^2 à ν degrés de liberté. Rappellons que le nombre de degrés de liberté est égal à la dimension du vecteur z et qu'en général on utilise $\alpha = 0.05$.
   Pour dessiner l'intervalle de confiance on procède comme suit :
   - définir le domaine pour z, c'est-à-dire $x \in [L, U]$ et $y \in [L, U]$, (on pourra choisir $L = -0.07$ et $U = 0.07$)
   - construire le vecteur $x_i = L + i \ \triangle$, $i = 0, 1, \ldots, n$ et $\triangle = (U - L)/n$, (choisir $n = 80$)
   - construire le vecteur $y_i = L + i \ \triangle$, $i = 0, 1, \ldots, n$ et $\triangle = (U - L)/n$
   - évaluer la forme quadratique $(z - \overline{r})' \, Q^{-1} \, (z - \overline{r})$ pour les $n \times n$ vecteurs $z = [x_i - \overline{r}_x \quad y_j - \overline{r}_y]'$
   - dessiner la ligne de niveau correspondant à $S = \chi^2_{\alpha,\nu}$

   Ceci est résumé avec le pseudo-code qui suit :

   Choisir L, U, n et α
   Initialiser $S \in \mathbb{R}^{n \times n}$
   $c = \chi^2_{\alpha,\nu}$ (utiliser `chi2inv`)
   $x_i = L + i \ \triangle$, $i = 0, 1, \ldots, n$ et $\triangle = (U - L)/n$ (utiliser `linspace`)

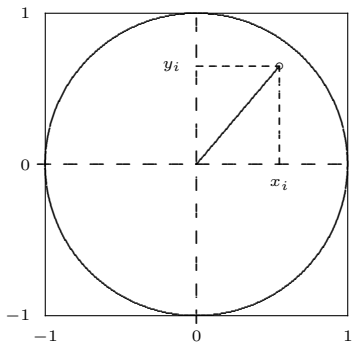
$y_i = L + i\ \triangle$, $i = 0, 1, \ldots, n$ et $\triangle = (U - L)/n$ (utiliser `linspace`)
pour $i = 1 : n$ **faire**
  pour $j = 1 : n$ **faire**
    $z = [x_i - \overline{r}_x \quad y_j - \overline{r}_y]'$
    $S_{ji} = z'\, Q^{-1}\, z$
  finfaire
finfaire
`plot(`$\overline{r}_x$`,`$\overline{r}_y$`,'r+'), hold on`
`contour(`x`,`y`,`S`,[c c])`
`xlabel('x'), ylabel('y')`

Dans la commande `contour` une colonne de la matrice S représente les valeurs de la forme quadratique pour un x_i donné, c'est pourquoi dans la construction de la matrice S l'indice des lignes varie avec y.

3. Générer $n = 1000$ réalisations du vecteur aléatoire $z \backsim N(\overline{r}, Q)$.
4. Completer le graphique de l'intervalle de confiance en traçant un point pour chaque vecteur générée. Écrire une procédure qui compte les points n_α situés en dehors de l'intervalle de confiance et imprimer la proportion n_α/n de ces points par rapport au total. Commenter.
5. Réexécuter le script en posant $\alpha = 0.001$. Commenter.

11.4 MCpi

Afin d'obtenir le nombre π par une méthode de Monte Carlo, considérons la figure suivante :



On vérifie que le rapport de l'aire du cercle sur l'aire carrée est $r^2\pi/4 = \pi/4$.

En générant aléatoirement n points $(x_1, y_1), \ldots, (x_n, y_n)$, uniformément distribués dans le carré de base 2, on peut estimer la quantité $\pi/4$ en calculant le rapport n_c/n du nombre des points n_c qui se trouvent à l'intérieur du cercle sur le nombre total des points n.

Pratiquement, on peut procéder comme suit :
- Poser $n = 50$.
- Initialiser le générateur de nombres au hasard avec la valeur 123456789.
- Générer 2 vecteurs u et v uniformément distribués dans [0, 1[.
- Transformer u et v en x et y uniformément distribués dans [−1, 1[.
- Calculer $r = x^2 + y^2$.
- Calculer n_c le nombre d'observations pour lesquelles $r_i \leq 1$.
- Calculer $\hat{\pi} = 4\,\frac{n_c}{n}$.
- Calculer l'erreur relative entre $\hat{\pi}$ et $\pi = 4\,\text{atan}(1)$.
- Refaire les calculs pour les valeurs $n = 100,\ 1000,\ 10000$.

11.5 RandU

Soit la récursion

$$x_{i+1} = c\,x_i \bmod M \qquad i = 1, \ldots, n \tag{45}$$

avec $c = 65539$ et $M = 2^{31}$ et où $a \mod b$ est le reste de la division entière a/b. A partir de cette récursion on peut construire des variables dites *pseudo-aléatoires*

$$u_i = x_i/M \qquad i = 1, \ldots, n$$

uniformement distribuées dans l'intervalle $[0, 1]$.

1. Ecrire une fonction Matlab `RandU` qui génère un vecteur u de variables pseudo-aléatoires. La fonction aura la structure
   `u = Randu(n,seed)`
   où n le nombre d'éléments que l'on désire générer et `seed` la valeur de départ de la récurrence. Si la fonction est appellée sans le deuxième argument on posera `seed` $= 1$.
2. Générer un vecteur u avec $n = 10000$ et vérifier la distribution uniforme des éléments dans u avec un histogramme. Que peut-on conclure ?
3. A partir du même vecteur u générer les coordonnées
   $$(u_{2i+1},\ u_{2i+2}) \qquad i = 0, 1, \ldots, \lfloor (n-1)/2 \rfloor$$
   de $\lfloor (n-1)/2 \rfloor$ points dans $\mathbb{R}^2$ et faire un graphique.[12] Que peut-on conclure ?
4. Toujours à partir du vecteur u générer les coordonnées
   $$(u_{3i+1},\ u_{3i+2},\ u_{3i+3}) \qquad i = 0, 1, \ldots, \lfloor (n-2)/3 \rfloor$$
   de $\lfloor (n-2)/3 \rfloor$ points dans $\mathbb{R}^3$ et faire un graphique (utiliser `plot3`). Que peut-on conclure ?[13] Essayez d'effectuer une rotation du graphique. Que peut-on constater ?

11.6 Moca-00

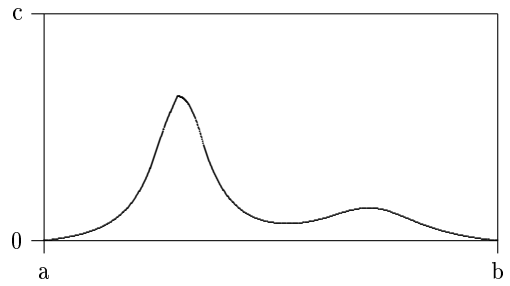
Soit la fonction

$$f(x) = \frac{1}{(x - 0.3)^2 + 0.01} + \frac{1}{(x - 0.9)^2 + 0.04} - 6 \tag{46}$$

1. Écrire la fonction Matlab `y = bosses(x)` qui évalue (46) pour un vecteur x donné.
2. Faire un graphique de la fonction (46) dans le domaine $x \in [-2,\ 2]$ en se servant de la fonction Matlab `fplot`.
3. A l'aide de l'algorithme de la bissection rechercher les zéros de la fonction (46).
4. Évaluer l'intégrale
   $$I = \int_a^b f(x)dx \tag{47}$$
   avec a et b les zéros de la fonction.

Une façon d'obtenir numériquement une approximation pour l'intégrale (47) consiste à définir un domaine $D \subset \mathbb{R}^2$, comme il est montré dans la figure qui suit.

[12] On note $\lfloor x \rfloor$ le plus proche entier de x vers zéro.
[13] Pour une discussion détaillée voir Marsaglia, G. (1968) : Random numbers fall mainly in the planes. *Proc. National Academy of Sciences USA* **61**, 25–28.



Le domaine D est le rectangle de base $b-a$ et de hauteur c, ayant pour surface

$$S = c\,(b-a)\,.$$

On génère n points aléatoires, donnés par les couples (x_i, y_i), $i = 1, \ldots, n$, où x et y sont uniformément distribués dans le domaine D. Soit p le nombre de points se trouvant en-dessous de la fonction, alors

$$S\,\frac{p}{n}$$

est une approximation de l'intégrale (47). On appelle cette technique, utilisée ici pour approcher notre intégrale, la *méthode de Monte-Carlo*[14].

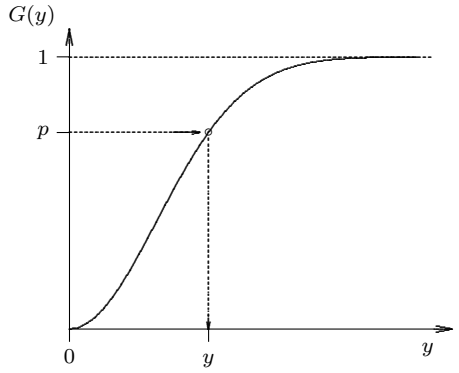
5. En choisissant $c = 150$, générer n points dans D comme décrit plus haut, en construisant notamment les deux vecteurs de coordonnées x et y. En choisissant, par exemple $n = 10$, vérifier que les points se trouvent dans D. On peut visualiser les points dans le domaine avec les commandes : `figure(clf); fplot('bosses',[a b 0 c]); hold on; plot(x,y,'+');`
6. Calculer une approximation de l'intégrale (47), avec $c = 150$ et $n = 100$.
7. Répéter 1000 fois le calcul de l'intégrale (47) avec la méthode de Monte-Carlo et calculer la moyenne ainsi que l'écart-type des résultats obtenus.
8. En posant $c = 100$ répéter la simulation de la question précédente. Que constate-t-on ? Commenter brièvement.
9. On peut montrer que l'approximation de cette intégrale par la méthode de Monte-Carlo suit une variable aléatoire distribuée selon une loi normale. Construire l'histogramme correspondant aux résultats des 1000 repetitions (commande `hist` et `bar`). Commenter brièvement.

11.7 GPDex

On considère la fonction de distribution de Pareto généralisée qui s'écrit

$$G(y) = 1 - (1+\xi y)^{-1/\xi} \tag{48}$$

avec $\xi = 1/2$. On désire générer des réalisations de cette distribution. Pour ce faire, on inverse la fonction G, c'est-à-dire que l'on cherche la valeur de l'argument y correspondant à une valeur p de G. La figure qui suit illustre ce procédé.



Comme G est une fonction de distribution, elle vérifie $0 \leq G \leq 1$. Ainsi les valeurs de p peuvent être générées comme des variables aléatoires uniformes.

On vérifie aisément que l'inverse de la fonction G s'écrit

$$y = \frac{e^{-\xi \log(1-p)} - 1}{\xi}. \tag{49}$$

1. Initialiser le générateur des nombres pseudo-aléatoires avec la valeur 99999.
2. Générer $n = 100$ réalisations d'une variable aléatoire uniforméément distribuée dans $[0, 1[$.
3. Générer $n = 100$ réalisations de la distribution définie en (48) en utilisant la transformation definie en (49).

On considère maintenant que les n réalisations de G sont données dans un ordre croissant, $y_1^n \leq \cdots \leq y_n^n$. La distribution empirique est alors définie comme

$$\hat{G}_n(y_i^n) = \frac{i}{n} \quad i = 1, \ldots, n. \tag{50}$$

4. Pour $y \in [0, 20]$ faire un graphique de la fonction $G(y)$ définie en (48).
5. Dans la même fenêtre, dessiner les couples

   $$(y_i^n,\, \frac{i}{n}) \qquad i = 1, \ldots, n$$

   qui définissent la distribution empirique $\hat{G}_n$ (on ne marquera que les points de la fonction, sans les relier). Ne pas oublier que les variables y doivent être rangées dans l'ordre croissant.

La fonction de densité de la distribution de Pareto généralisée s'obtient en dérivant (48) et s'écrit

$$g(y) = (1+\xi\, y)^{\frac{-1}{\xi}-1}. \tag{51}$$

Etant données n observations, la vraisemblance est définie comme

$$V = \prod_{i=1}^{n} g(y_i) \tag{52}$$

et le logarithme de la vraisemblance comme

$$L = \sum_{i=1}^{n} \log(g(y_i)). \tag{53}$$

6. Pour 40 valeurs de ξ allant de 0.3 à 0.7, évaluer les fonctions (52) et (53). Faire un graphique de ces deux fonctions dans deux fenêtres juxtaposées (`subplot(211)` et `subplot(212)`).
   Vérifier que les fonctions attaignent leur maximum pour une valeur ξ proche de 1/2. Commenter.

[14] C'est en raison du recours à la génération de nombres au hasard, pour laquelle on pourrait se servir d'une roulette, que l'on a donné à cette technique le nom de "méthode de Monte-Carlo".

7. Selon la procédure indiquée plus haut, générer maintenant $n = 500$ réalisations de la distribution généralisée de Pareto. Faire les graphiques de la vraisemblance et du logarithme de la vraisemblance commme défini en (52) et (53).

   Commenter les résultats.

11.8 MCVar

On vérifie aisément que l'intégrale définie I de la fonction exponentielle e^x sur l'intervalle [0, 1] vaut approximativement

$$I = \int_0^1 e^x \, dx = e - 1 \approx 1.7183 \,. \tag{54}$$

Une approximation numérique $\widehat{I}_n$ de I peut être obtenu par la méthode de Monte Carlo. Cette approximation est définie comme

$$\widehat{I}_n = \frac{1}{n} \sum_{i=1}^{n} e^{u_i} \tag{55}$$

où les u_i sont des réalisation d'une variable aléatoire uniformement distribuée dans l'intervalle [0, 1].

1. A l'aide de l'instruction Matlab `for` écrire la procédure qui calcule $\widehat{I}_n$ pour $n = 100$.

Etant donné que les u_i sont des réalisations d'une variable aléatoire, l'approximation $\widehat{I}_n$, qui est une fonction des u_i, est également une variable aléatoire. Concrétement cela veut dire que la valeur de $\widehat{I}_n$ change d'une exécution à l'autre.

2. Reprendre la procédure de la question précédante et la compléter avec les instructions nécessaires pour pouvoir l'exécuter R fois. Poser $R = 5$, exécuter la procédure et observer les différents résultats obtenus pour $\widehat{I}_n$.
3. Une mesure de la variabilité est donnée par l'écart quadratique moyen qui est défini comme

$$V(\widehat{I}_n) = \frac{1}{R} \sum_{r=1}^{R} (\widehat{I}_n^r - I)^2 \tag{56}$$

   où $\widehat{I}_n^r$ sont $r = 1, \ldots, R$ approximations de $\widehat{I}_n$. Reprendre la procédure du point précédant et la compléter afin de calculer l'écart quadratique moyen de l'approximation de l'intégrale (54) pour $n = 100$ et $R = 200$.

La racine de l'écart quadratique moyen est appellée *écart-type* et est noté $\sigma_{\widehat{I}_n}$. L'écart-type sert à construire des intervalles de confiance. Ainsi on établit, qu'en moyenne, 5% des approximations $\widehat{I}_n$ générées avec notre procédure s'écarteront de plus de $2\,\sigma_{\widehat{I}_n}$ de la moyenne qui est I.

4. Question facultative : Pour $n = 100$ et $R = 200$ compter le nombre d'approximations qui se trouvent en dehors de cet intervalle. On peut faire un graphique et compter les approximations qui se trouvent en dehors de l'intervalle (voir la Figure 5).

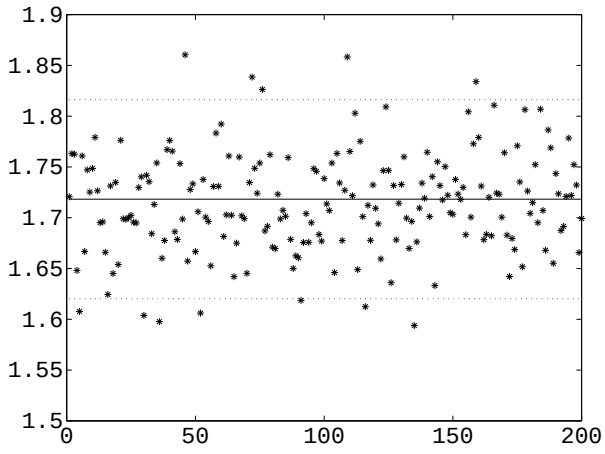


FIG. 5 – Approximations et intervalle de confiance.

On peut s'attendre que la précision de notre approximation $\widehat{I}_n$ augmente lorsque n augmente (ou vice-versa que $\sigma_{\widehat{I}_n}$ diminue lorsque n augmente). On se propose maintenant d'étudier empiriquement la relation qui existe entre n et $\sigma_{\widehat{I}_n}$.

5. Pour $n = 2^i$, $i = 7, \ldots, 12$ calculer $\sigma_{\widehat{I}_n}$. Faire un graphique des 6 couples $(n, \sigma_{\widehat{I}_n})$. Dans le même graphique tracer la fonction $y = c/\sqrt{n}$ pour $2^7 \leq n \leq 2^{12}$ et $c = 0.5$ (voir la Figure 6).

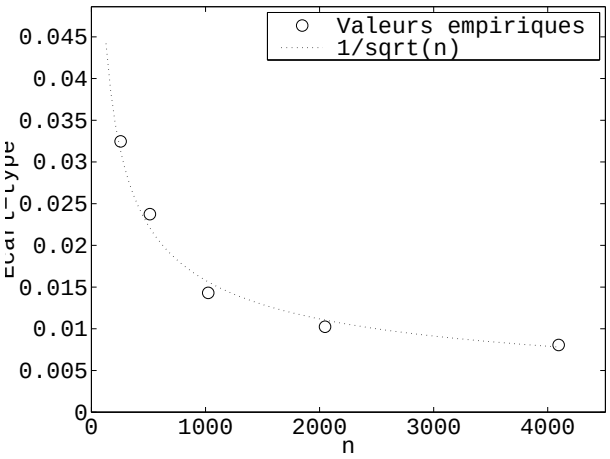


FIG. 6 – Evolution de l'écart-type empirique.

Que peut-on conclure ?

12 Optimisation

12.1 PL00

Une entreprise qui produit de la nourriture pour chiens en gros décide d'entrer dans le marché de détail avec 3 nouveaux produits. Ces produits sont :
– Seniordog nourriture pour vieux chiens,
– Topdog nourriture pour animaux en pleine maturité,
– Pupdog nourriture pour chiots.
Les ingrédients de ces trois produits sont constitués par les 4 produits A, B, C et D vendus jusque-là en gros. Du point de vue nutritionnel on a établi que :

- Seniordog doit contenir au moins 25% de B et pas plus de 20% de C,
- Topdog doit contenir au moins 50% de A et pas plus de 25% de D,
- Pupdog doit contenir au moins 25% de A, au moins 25% de B et 0% de C.

Les quantités disponibles de A, B, C et D sont limitées à 1000, 1000, 750 et 800 kg par semaine. Les prix respectifs sont : 2.00 fr/kg, 2.40 fr/kg, 1.60 fr/kg et 1.80 fr/kg.

Le prix de vente des trois nouveaux produits a été fixé à 3.60 fr/kg pour le produit Seniordog, 3.40 fr/kg pour le produit Topdog et 3.60 fr/kg pour le produit Pupdog. On fait l'hypothèse qu'à ces prix de vente la demande sera supérieure à l'offre, c'est-à-dire que toute la production pourra être vendue.

Il se pose dès lors la question du mélange de A, B, C et D à choisir afin de maximiser le profit.

1. Formuler le programme linéaire en procédant selon les étapes suivantes :
   - définition des variables de décision,
   - définition de la fonction objectif,
   - définition des contraintes.
2. Chercher la solution numérique.

,

12.2 ARML

On désire effectuer une estimation du modèle suivant par la méthode du maximum de vraisemblance :

$$y = X\beta + \varepsilon \,, \tag{57}$$

avec $y \in \mathbb{R}^{T\times 1}$, $X \in \mathbb{R}^{T\times K}$ et

$$\begin{aligned} &\varepsilon_t = \rho\,\varepsilon_{t-1} + u_t \quad , \quad t = 1,\ldots,T \,, \\ &|\rho| < 1 \,, \\ &u_t \overset{\text{iid}}{\sim} \mathcal{N}(0,\sigma^2) \,, \end{aligned}$$

ainsi que les autres hypothèses classiques.

Ce modèle est connu sous le nom de "*modèle linéaire avec autocorrélation des erreurs*". Dans ce cadre, les estimateurs des moindres carrés ordinaires ne sont plus efficaces et une alternative consiste à recourir aux estimateurs du maximum de vraisemblance.

La vraisemblance du modèle (57) s'écrit

$$\mathcal{L}(\theta; X, y) = \prod_{t=1}^{T} f_t(\theta; X, y) \,,$$

où f_t désigne la densité des observations et θ est le vecteur des paramètres à estimer contenant β, σ et ρ.

On maximise en général $\ln\mathcal{L}$ plutôt que $\mathcal{L}$, car son expression est plus simple à utiliser et les solutions sont invariants par rapport à cette transformation.

Dans notre cas on a

$$\ln\mathcal{L}(\theta; X, y) = \frac{1}{2}\ln(1-\rho^2) - \frac{T}{2}\ln(2\pi\sigma^2) - \frac{1}{2\sigma^2}(y-X\beta)'R'R(y-X\beta) \,,$$

avec

$$R = \begin{bmatrix} \sqrt{1-\rho^2} & 0 & \cdots & \cdots & 0 \\ -\rho & 1 & 0 & \cdots & 0 \\ 0 & -\rho & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & 1 & 0 \\ 0 & \cdots & 0 & -\rho & 1 \end{bmatrix} \,.$$

Le problème est de trouver la solution de

$$\max_{\theta=[\beta'\ \sigma\ \rho]'} \ln\mathcal{L}(\theta; X, y) \,.$$

On utilisera la fonction `fminsearch` de Matlab pour résoudre ce problème et de ce fait on minimisera $-\ln\mathcal{L}$.

On aimerait appliquer cette méthode pour estimer une équation d'investissement pour les Etats-Unis[15].

On explique l'investissement réel ($\bar{I}$) par une fonction linéaire du produit national brut en termes réels ($\overline{GNP}$) et du taux d'intérêt réel ($\bar{r}$) :

$$\bar{I}_t = \beta_0 + \beta_1\,\overline{GNP}_t + \beta_2\bar{r}_t + \varepsilon_t \quad , \quad t = 1964,\ldots,1982 \,. \tag{58}$$

Le fichier `v:\metri\mme\arml.dat` contient un tableau de données[16] sur l'investissement nominal (I), le produit national brut nominal (GNP), l'indice des prix (P), et le taux d'intérêt nominal (r) pour les Etats-Unis sur la période 1963–1982.

1. Calculer les variables utilisées dans l'équation (58). Le produit national et l'investissement en termes réels s'obtiennent en divisant les données nominales par l'indice des prix de la période. Une approximation du taux d'intérêt réel est obtenue en soustrayant le taux de variation de l'indice des prix au taux d'intérêt nominal :
$$\bar{r}_t = r_t - 100\,\frac{(P_t - P_{t-1})}{P_{t-1}} \quad , \quad t = 1964,\ldots,1982 \,.$$
2. Développer analytiquement le produit Ry afin de faire apparaître la transformation opérée par R sur les éléments de y. Donner une expression Matlab calculant $y^* = Ry$ sans effectuer de produit matriciel.
   Donner aussi l'expression correspondante pour $X^* = RX$.
3. Programmer une fonction Matlab `logl(theta,X,y)` qui calcule $-\ln\mathcal{L}$ en fonction de $\theta = [\beta'\ \sigma\ \rho]'$ de dimension $(K+2)\times 1$, de X et de y.
   Dans `logl`, on programmera d'abord le calcul X^* et y^* pour ensuite utiliser ces résultats dans l'expression de la vraisemblance.
   Indication : `logl([-10 0.2 -1 10 0.5]',X,y)=130.225`.
4. Utiliser la fonction `fminsearch` de Matlab de la façon suivante afin d'obtenir l'estimation voulue :
   `theta = fminsearch('logl',theta0,[ ],X,y)` .
   Le vecteur `theta0` contient les valeurs initiales suivantes pour les paramètres :
   $\hat{\beta}$ = estimation de β par les m.c.o. ,
   $\hat{\sigma}$ = $\sqrt{\hat{\varepsilon}'\hat{\varepsilon}/(T-K)}$, où $\hat{\varepsilon} = y - X\hat{\beta}$,
   $\hat{\rho}$ = $\sum_{t=2}^{T}\hat{\varepsilon}_t\hat{\varepsilon}_{t-1}$ / $\sum_{t=2}^{T}\hat{\varepsilon}_t^2$.
   Vérifier que l'on obtient les résultats suivants :
   $$\begin{aligned} \hat{\beta}_{\text{ML}} &= [-14.4921 \ \ 0.1701 \ \ -0.8249]' \,, \\ \hat{\sigma}_{\text{ML}} &= 15.2655 \,, \\ \hat{\rho}_{\text{ML}} &= 0.2795 \,. \end{aligned}$$
5. Imprimer les résultats sous la forme suivante :

```
-----------------------------------------------
 Estimateur Constante   GNP   Intérêt   rho
-----------------------------------------------
    OLS      -12.533   0.169  -1.001   0.222
    ML       -14.492   0.170  -0.824   0.279
-----------------------------------------------
```

[15] Greene, W.H., *Econometric Analysis*, MacMillan, New York, 1990, p.430.

[16] Source : *Economic Report of the President*, Government Printing Office, Washington D.C., 1984.

13 A classer

13.1 graph00

Soit la fonction

$$y = x^3 + 3x^2 - 3x - 1$$

et sa dérivée

$$d = 3x^2 + 6x - 3\,.$$

1. Construire un vecteur x avec 40 éléments. Le prémier élément a la valeur -4, le dernier la valeur 2 et la différence de valeur entre deux éléments successifs est constante.
2. Calculer le vecteur y qui correspond au valeurs du vecteur x.
3. Calculer le vecteur d qui correspond au valeurs du vecteur x.
4. Dans une même fenêtre, faire le graphique de la fonction et de sa dérivée.
5. Refaire le graphique de la fonction et de sa dérivée en dessinant le tracé de la fonction en rouge et celui de la dérivée en vert.
6. Dans le même graphique, tracer une ligne entre les deux points (x_1, y_1) et (x_2, y_2) avec $x_1 = -1$, $x_2 = -1$, $y_1 = -6$ et $y_2 = 4$. Les deux points seront marqués avec un petit cercle.
7. Ecrire le titre «Inflexion de f» sur le graphique et marquer l'absisse avec le symbole x.

13.2 kron-0

Soit deux matrices A et B non-singulières et d'ordre p respectivement q. Le produit de Kronecker vérifie alors les propriétés suivantes :

$$(A \otimes B)^{-1} = A^{-1} \otimes B^{-1} \tag{59}$$

$$det(A \otimes B) = (det(A))^q\,(det(B))^p \tag{60}$$

1. Construire deux matrices A et B de rang $p = 5$, respectivement $q = 9$, dont l'élément générique a_{ij}, respectivement b_{ij} est défini comme suit :

$$\begin{aligned} a_{ij} &= 1/(i+j-1) \\ b_{ij} &= 1/(i+j-1) \quad . \end{aligned}$$

   Indication : La $j$ème colonne d'une telle matrice d'ordre n s'écrit

$$\texttt{ones(n,1)./(j : n + j - 1)}'$$

2. Evaluer à l'aide de la fonction `kron` et `inv` l'expression à gauche et à droite dans (1). A l'aide de la fonction `flops` comparer le nombre d'opérations élémentaires nécessaires à chaque évaluation.
3. Multiplier le terme de gauche, puis celui de droite par son inverse $(A \otimes B)$. Comparer les diagonales du produit. Commenter.
4. Evaluer l'expression à gauche et à droite dans (2). Comparer le nombre d'opérations élémentaires nécessaires à chaque évaluation.

13.3 kron-1

Soit deux matrices A et B non-singulières et d'ordre p respectivement q. Le produit de Kronecker vérifie alors les propriétés suivantes :

$$(A \otimes B)^{-1} = A^{-1} \otimes B^{-1} \tag{61}$$

$$det(A \otimes B) = [det(A)]^q [det(B)]^p \tag{62}$$

1. Construire deux matrices A et B de rang $p = 5$, respectivement $q = 7$, dont l'élément générique a_{ij}, respectivement b_{ij} est défini comme suit :

$$\begin{aligned} a_{ij} &= 1/(i+j-1) \\ b_{ij} &= 1/(i+j-1) \quad . \end{aligned}$$

   Indication : La $j$ème colonne d'une telle matrice d'ordre n s'écrit

$$\texttt{ones(n,1)./(j : n + j - 1)}'$$

2. Evaluer à l'aide de la fonction `kron` et `inv` l'expression $K1 = (A \otimes B)^{-1}$ à gauche dans (1) et l'expression $K2 = A^{-1} \otimes B^{-1}$ à droite dans (1). A l'aide de la fonction `flops` comparer le nombre d'opérations élémentaires nécessaires nécessaires pour évaluer $K1$ et $K2$. Commenter.
3. Evaluer $K = A \otimes B$. Calculer $I1 = K * K1$ et $I2 = K * K2$. Visualiser les matrices à l'aide de la commande `mesh(I1)` et `mesh(I2)`. Commenter.
4. Evaluer l'expression à gauche et à droite dans (2). Comparer le nombre d'opérations élémentaires nécessaires à chaque évaluation. Commenter.

13.4 Maple-00

Avec Maple répondre aux questions suivantes :

1. Calculer l'intégrale définie

$$I = \int_0^1 e^x\,dx$$

2. Calculer la somme

$$\sum_{i=7}^{12} 2^i$$

13.5 Maple-01

A l'aide de MAPLE résoudre les problèmes suivants :

1. Ramanujan se demandait si le résultat de l'expression

$$\frac{\pi\sqrt{310}}{\log((2+\sqrt{2})\cdot(3+\sqrt{5})\cdot(5+2\sqrt{10}+\sqrt{61+20\sqrt{10}})/4)}$$

   était entier. Vérifier numériquement que ce n'est pas le cas.

2. Evaluer l'expression suivante :

$$2\sum_{k=1}^{n-1}(n-k)^2 + \sum_{k=1}^{n-1}(n-k)$$

3. Soit la matrice

$$A = \begin{bmatrix} 0 & a & 0 & b & 0 & c \\ d & 0 & e & 0 & 0 & 0 \\ 0 & f & g & 0 & 0 & 0 \\ h & k & 0 & 0 & l & 0 \\ m & 0 & n & 0 & i & 0 \\ 0 & j & 0 & r & 0 & p \end{bmatrix}$$

Calculer l'expression du déterminant et simplifier le résultat. Calculer la matrice inverse.

4. Diviser le polynome $x^5 - x^4 + 8x^3 - 8x^2 + 16x - 16$ par le polynome $x^5 - x^4 + 12x^3 - 12x^2 + 36x - 36$ et simplifier le résultat.

13.6 Maple-02

A l'aide de MAPLE résoudre les problèmes suivants :

1. Ramanujan se demandait si l'expression qui suit était un entier.

$$\frac{\pi\sqrt{310}}{\log((2+\sqrt{2})\,(3+\sqrt{5})\,(5+2\sqrt{10}+\sqrt{61+20\sqrt{10}})/4)}$$

Vérifier numériquement que ce n'est pas vrai.

2. Factoriser l'expression

$$x + \frac{2}{x} + \frac{1}{x^2} - \frac{2}{x-1} \quad .$$

3. Dériver le produit $f(x)g(x)$ par rapport à x.
4. Dériver le quotient $\frac{f(x)}{g(x)}$ par rapport à x.
5. Dériver la fonction composée $f(g(x))$ par rapport à x.
6. Définir une fonction f telle que $f(x) = 2x(x^2+1)$.
   - Evaluer $f(x)$ pour x égal aux expressions suivantes : i) 27 ii) -1 iii) $a+b$ iv) $\sqrt{5}$ v) $\frac{1+s}{1-s}$
   - Qu'observe-t-on si l'on dérive f en utilisant `diff(f,x)` ? Commenter.
   - Trouver la dérivée de f par rapport à x. Pourquoi n'est il pas nécessaire de réassigner x comme un symbole avant de dériver ?
   - Evaluer $f'(3)$.

Avec MATLAB programmer l'algorithme 3.2.4 et à l'aide de MAPLE établir la fontion du nombre d'opérations élémentaires.

13.7 Maple-03

A l'aide de Maple résoudre le problème qui suit.

Soit la fonction $f : \mathbb{R} \to \mathbb{R}$ définie par

$$f(x) = \frac{x}{x^2+1}.$$

- Calculer la valeur de f pour $x = 0$, $x = 1$, $x = -1$.
- Calculer $\lim_{x\to\infty} f(x)$ et $\lim_{x\to-\infty} f(x)$.
- Trouver les maximums, les minimums et les points d'inflexion de la fonction f.
- Tracer le graphe de f.

13.8 Maple-04

Ce problème est à resoudre à l'aide de Maple.

Soit la fonction de densité de la variable aléatoire normale de paramètre μ et σ.

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp \quad -\frac{(x-\mu)^2}{2\sigma^2} \quad . \tag{63}$$

1. Calculer l'expression de la dérivée de $f(x)$.
2. Determiner le maximum de f en cherchant la solution de $f'(x) = 0$.
3. Determiner les point d'inflection de f en cherchant les solutions de $f''(x) = 0$.
4. Dans un même graphique dessiner les fonctions de densité pour $x \in [-15, 15]$ et qui correspondent aux deux choix de paramètres suivants : $\mu = 3$, $\sigma = 4$ et $\mu = 5$, $\sigma = 2$.

13.9 mlab-xxa

Soit `erf(x)`, appelée "fonction d'erreur" et $\phi(x)$, la fonction de répartition de la loi normale centrée-réduite. Ces deux fonctions sont définies comme suit :

$$\texttt{erf(x)} = \int_0^x \frac{2}{\sqrt{\pi}} e^{-t^2} dt \quad \text{et} \quad \phi(x) = \int_{-\infty}^x \frac{1}{\sqrt{2\pi}} e^{-\frac{t^2}{2}} dt$$

Dans le but d'établir la relation entre ces deux fonctions, on propose de suivre les étapes suivantes :

1. A l'aide de `lookfor` rechercher la fonction de MATLAB qui calcule la valeur de $\phi(x)$ (on remarquera que la moyenne et la variance sont des paramètres en entrée).
2. En explorant le code de cette fonction, établir la relation qui existe entre $\phi(x)$ et `erf`(x).
3. Montrer que l'on peut obtenir le même resultat numérique avec la fonction `quad`.

13.10 mlab04

A l'aide du logiciel MATLAB résoudre le problème proposé ci-dessous.

Soit la fonction :

$$y = x^3 + 3x^2 - 3x - 1.$$

1. Faire un graphique de cette fonction, avec la commande **plot** de MATLAB.

On désire maintenant trouver les zéros de cette fonction. On utilisera la méthode itérative de Newton, qui à partir d'un point initial x_o donnée, calcule une meilleure approximation du point cherché, en remplaçant la valeur de la fonction en un point par sa tangente. Les valeurs successives de x sont déterminées de la façon suivante :

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)} \qquad k = 1, 2, 3, \ldots$$

La précision du résultat est jugée suffisante lorsque la condition

$$|f(x_{k+1}) - f(x_k)| < \varepsilon$$

est satisfaite, pour une valeur de ε donnée.

2. En choisissant différentes valeurs initiales x_o, calculer les zéros de cette fonction.
3. Déterminer numériquement les extrema en appliquant cette même procédure à la dérivée de la fonction.

Rendre un listing du programme et des résultats.

13.11 mlab05

Dans les tests statistiques du type dit *lambda* on cherche les intersections de deux fonctions suivantes :

$$c = ln(u)$$

$$d = \frac{u}{n} + k$$

1. Construire un algorithme qui, pour des valeurs de $n \in N_*$ et $k \in R_+$, determine les valeurs u_{inf} et u_{sup} correspondant aux deux points d'intersection.
2. Programmer cet algorithme sous la forme d'une fonction MATLAB.
3. Representer les fonctions et les solutions avec un graphique.

13.12 mlab07

Reprendre la fonction MATLAB qui recherche les zéros d'une fonction par la méthode itérative de Newton.

Compléter l'algorithme de telle sorte qu'il reproduise le déroulement de la recherche d'une solution en une séquence animée de graphiques.

13.13 mlab07b

Générer l'axe des x et l'axe des y d'un graphique défini de -10 à 10 pour l'ordonnée et l'abscisse. A partir du point $(x = 0, y = 0)$ dessiner des déplacements définis par des accroissements Δx et Δy qui correspondent à des réalisations d'une variable aléatoire normale centrée réduite. Lorsqu'un pas conduit à se déplacer au delà de la limite des axes on ne l'effectue pas.

Créer le programme qui dessine ces déplacements en une séquence animée.

13.14 mlab08

On considère la fonction de Rosenbrock :

$$f(x_1, x_2) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2 \quad .$$

Donner une représentation en trois dimensions de cette fonction ainsi que un graphique avec les lignes de niveau. Que peut-on conclure quant au domaine pour lequel la fonction est minimum.

13.15 mlab09

On pense qu'un phénomène suit une loi de probabilité de Poisson. On observe 4 réalisations qui sont :

$$X = \{0, 1, 1, 3\} \quad .$$

On désire estimer le paramètre λ qui caractérise cette loi de Poisson. Soit alors la probabilité conjointe

$$P(X \mid \lambda) = \prod_{x \in X} \frac{e^{-\lambda}\lambda^x}{x!}$$

qui, pour l'ensemble des observations X, s'écrit :

$$P(\lambda) = \frac{e^{-4\lambda}\lambda^5}{6} \quad .$$

L'estimateur du maximum de vraisemblance de λ correspond à la valeur de λ qui maximise $P(\lambda)$. Pratiquement on maximisera le logarithme de P en omettant les termes constants. On a alors

$$L(\lambda) = -4\,\lambda + 5\,\log\lambda \quad .$$

Rechercher λ qui correspond au maximum de L.

Ce problème de maximisation peut se ramener à un problème de recherche de zéros d'une fonction, qui dans ce cas est la dérivée première.

1. Appliquer l'algorithme de la bissection, ou un autre algorithme de votre choix, pour trouver la valeur de λ qui satisfait la condition du premier ordre.
2. Vérifier le résultat à l'aide de la fonction `fminbnd` de MATLAB.

13.16 mlab10

On désire générer T observations d'une variable aléatoire binomiale de paramètres n et p.

1. Avec MATLAB écrire une fonction qui, pour n, p et T donnés, génère le vecteur x des T observations, ainsi que le vecteur f des fréquences relatives. On a :

   $f(i) =$ (nombre de fois que x a pris la valeur $i+1)/T$

   La fonction s'écrira :

   `function` $[\mathtt{f}, \mathtt{x}] = \mathtt{binom}(\mathtt{n}, \mathtt{p}, \mathtt{T})$

   et l'on prévoira avec "nargout" le cas où seul le vecteur f est désiré.
2. Construire une fonction qui calcule la probabilité d'une variable aléatoire binomiale de paramètres n et p. On aura :

   `function` $\mathtt{prob} = \mathtt{p\_binom}(\mathtt{n}, \mathtt{p})$

3. Pour des valeurs de n, p et T de votre choix, faire un plot superposé de f et de *prob*. Commenter.

On sait que la distribution d'une variable aléatoire binomiale tend vers une variable aléatoire normale pour des valeurs de n et p telles que $np > 30$.

4. Pour des valeurs (n=10, p=.5), (n=50, p=.1) et (n=500, p=.01) faire un plot superposé de la probabilité binomiale et de la densité normale correspondante. Commenter.

13.17 mlab10a

Un tirage de Bernoulli consiste en une expérience dont le résultat est soit un succès, soit un échec. On note p la probabilité de succès et $1-p$ la probabilité d'échec.

Avec MATLAB, il est facile de simuler une telle expérience. On génère une réalisation z d'une variable aléatoire uniforme et si $z \leq p$, on considère le résultat comme un succès (échec dans le cas contraire).

Considérons maintenant n tirages indépendants de Bernoulli. Le nombre total de succès obtenus y suit alors une loi dite binomiale, que l'on note $y \sim B(n,p)$. Pour générer une telle variable, on considère un vecteur aléatoire uniformément distribué de dimension n et la variable binomiale correspond au nombre d'éléments inférieurs à p.

1. Avec MATLAB, écrire une fonction qui génère un vecteur x de T observations d'une variable binomiale de paramètres n et p donnés, ainsi que le vecteur f des fréquences relatives. On a :

   $f(i) =$ (nombre de fois que x a pris la valeur $i-1)/T$

   fonction s'écrira :

   $$\texttt{function} \quad [\texttt{x},\texttt{f}] = \texttt{binom}(\texttt{n},\texttt{p},\texttt{T})$$

   l'on prévoira avec `nargout` le cas où seul le vecteur x est désiré.
2. Construire une fonction qui calcule les probabilités d'une variable aléatoire binomiale de paramètres n et p. On écrira :

   $$\texttt{function} \quad \texttt{prob} = \texttt{p\_binom}(\texttt{n},\texttt{p})$$

   ou `prob(i)` $= P\{y = i-1\}$, $i = 1,\ldots,n+1$. On a $P\{y=k\} = C_n^k p^k q^{n-k}$ et on rappelle que les coefficients binomiaux vérifient la récurrence suivante :

   $$C_n^{k+1} = \frac{n-k}{k+1} C_n^k \quad .$$

3. Pour des valeurs de n, p et T de votre choix, faire un graphique qui contenant les valeurs de f et de *prob*. Commenter. (Pour le graphique, on pourra utiliser la commande `plot(0 :n,f,'+',0 :n,prob,'o')`).
4. On sait que la distribution d'une variable aléatoire binomiale tend vers une variable aléatoire normale pour des valeurs de n et p telles que $\min(np, nq) > 15$. Pour des valeurs (n=10, p=.5), (n=50, p=.1), (n=500, p=.01) et (n=500, p=.6) faire un graphique de la probabilité binomiale et de la densité normale de paramètres $\mu = np$ et $\sigma = np(1-p)$. Commenter.

13.18 mlab12

On considère les instructions suivantes destinées à être exécutées avec MATLAB :

```
A = randn(n,m);
U = A - diag(sum(A')/m) * ones(n,m);
for i = 1:n
   for j = 1:n
      VC(i,j) = sum(diag(diag(U(i,:)) * diag(U(j,:)) )) / (m-1);
   end
end
VC
```

1. Que représente VC ?
2. Existe-t-il une façon plus efficace, pour MATLAB, de calculer VC ? Si oui, proposer une alternative.
   Pour $n = 5$ et $m = 50$ comparer la performance du programme original avec celui proposé. On comparera notamment le temps de calcul et le nombre d'opérations élémentaires.
   Présenter les résultats exactement sous la forme que voici :

```
                    Matrice VC :

| a(1,1)= 0.8 | a(1,2)=-0.1 | a(1,3)= 0.1 | a(1,4)=-0.2 | a(1,
| a(2,1)=-0.1 | a(2,2)= 0.9 | a(2,3)= 0.1 | a(2,4)=-0.1 | a(2,
| a(3,1)= 0.1 | a(3,2)= 0.1 | a(3,3)= 1.0 | a(3,4)= 0.0 | a(3,
| a(4,1)=-0.2 | a(4,2)=-0.1 | a(4,3)= 0.0 | a(4,4)= 1.3 | a(4,
| a(5,1)= 0.0 | a(5,2)= 0.0 | a(5,3)= 0.1 | a(5,4)=-0.4 | a(5,

Méthode originale : flops = ..... elapsed time = ......
      Alternative : flops = ..... elapsed time = ......
```

   Le traçage du cadre est facultatif.

13.19 mlab13

Un des problèmes courants rencontrés dans la modélisation input/output consiste, partant d'une matrice A donnée, à construire une matrices S, dont la somme des lignes est égale à un vecteur u et la somme des colonnes égale à un vecteur v.

La méthode RAS[17] constitue un algorithme efficace pour obtenir une approximation de la solution de ce problème. Cet algorithme s'énonce comme suit :

- $S^o = A$
- Pour $i = 1, 2, \ldots$, faire
  $$\begin{aligned} a^i &= (\widehat{S^{i-1}\imath})^{-1} u \\ S_a^i &= \widehat{a^i} S^{i-1} \\ b^i &= \widehat{(\imath' S_a^i)}^{-1} v \\ S^i &= S_a^i \widehat{b^i} \end{aligned}$$
- Critère d'arrêt :
  $$||a^i - a^{i-1}||_2 < \varepsilon \quad \text{et} \quad ||b^i - b^{i-1}||_2 < \varepsilon$$
- La matrice S^i correspond alors à la matrice S rec

Le symbole $\imath$ désigne le vecteur unitaire et le symbole $\widehat{\ }$ désigne la transformation d'un vecteur en une matrice diagonale.

Programmer la méthode RAS décrite ci-dessus.

Application

$$A = \begin{bmatrix} 60 & 20 & 42 & 35 & 80 & 23 \\ 36 & 10 & 84 & 70 & 29 & 44 \\ 24 & 70 & 14 & 19 & 32 & 65 \\ 10 & 60 & 30 & 20 & 40 & 50 \\ 62 & 22 & 11 & 52 & 12 & 39 \\ 44 & 92 & 33 & 71 & 31 & 82 \end{bmatrix} \quad u = \begin{bmatrix} 230 \\ 290 \\ 225 \\ 215 \\ 170 \\ 280 \end{bmatrix} \quad v = \begin{bmatrix} 240 \\ 270 \\ 220 \\ 270 \\ 100 \\ 310 \end{bmatrix}$$

Donner le nombre de "flops" exécutées pour obtenir l'approximation de la solution.

[17] Bacharach M. (1970) : *Biproportional Matrices and Input-Output Change*, Cambridge University Press.

Vers quelles valeurs tendent les éléments des vecteurs a^i et b^i pour $i \to \infty$. Quel autre critère d'arrêt ceci suggère-t-il ?

13.20 mlab14

Le theorème de Caley-Hamilton dit qu'une matrice vérifie sa propre équation caractéristique, c'est-à-dire, pour une matrice $A \in \mathbb{R}^{n \times n}$, on a la fonction caractéristique

$$\lambda^n + c_1\lambda^{n-1} + \ldots c_n = 0$$

et

$$A^n + c_1 A^{n-1} + \ldots c_n I = 0.$$

1. En utilisant la fonction MATLAB qui permet de calculer le polynôme caractéristique, écrire une procédure qui permette de vérifier le théorème numériquement pour une matrice A d'ordre n, dont les éléments sont tirés d'une loi normale $N(0,1)$. *(6 points)*
2. Tester la procédure pour quelques valeurs de $n < 20$ et visualiser le résultat (matrice nulle) à des fins de vérification. *(2 points)*

13.21 Mlab-intro00

1. Générer une matrice aléatoire A avec la commande `A = rand(4)` et vérifier son rang avec la commande `r = rank(A)`. Genérer une deuxième matrice aléatoire B de la même façon.

   Calculer le produit `C = A * B` et l'inverse D de la matrice C avec la commande `D = inv(C)`.

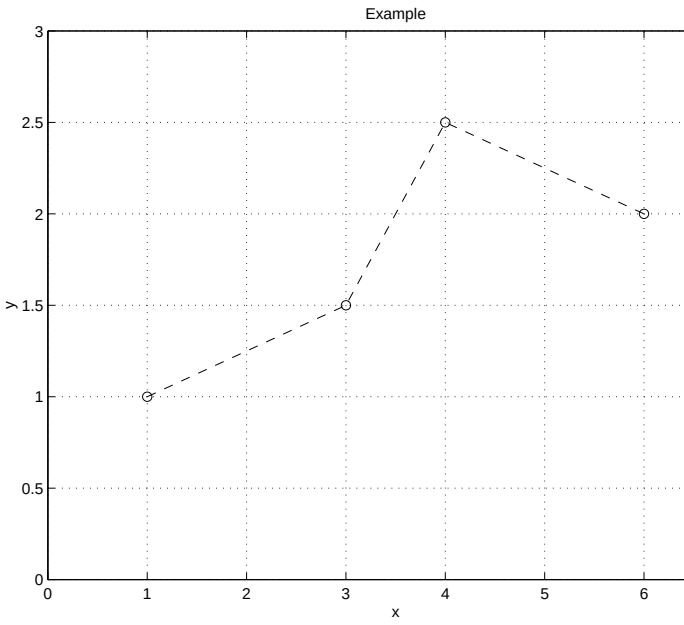
   Calculer la matrice $E = B^{-1}A^{-1}$. Vérifier que les matrices E et D sont identiques aux erreurs d'arrondi près.

2. Soit la matrice

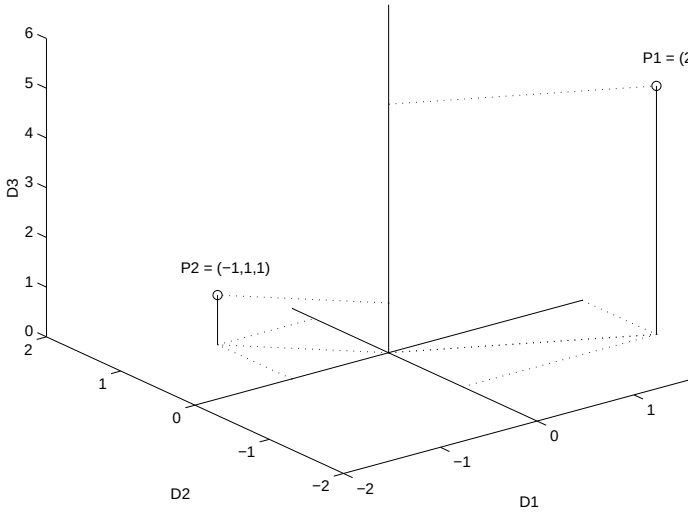
$$A = \begin{bmatrix} 1 & 2 \\ 3 & 5 \end{bmatrix}$$

   calculer la matrice $B = A^3$. Pour la matrice A calculer la matrice des vecteurs propres V et la matrice des valeurs propres D avec la commande `[V,D] = eig(A)`. Vérifier que la matrice $E = VD^3V^{-1}$ est identique à la matrice B.

3. Reproduire le graphique à deux dimensions ci-après. On utilisera les commandes `plot`, `grid`, `axis`, `xlabel`, `ylabel` et `title`.



4. Réproduire le graphique à trois dimensions ci-après. On utilisera les commandes `plot3`, `hold on`, `axis`, `xlabel`, `ylabel` et `zlabel`.



13.22 Mlab-intro01

1. Générer une matrice aléatoire A avec la commande `A = rand(4)` et vérifier son rang avec la commande `r = rank(A)`. Genérer une deuxième matrice aléatoire B de la même façon.

   Calculer le produit `C = A * B` et l'inverse D de la matrice C avec la commande `D = inv(C)`.

   Calculer la matrice $E = B^{-1}A^{-1}$. Vérifier que les matrices E et D sont identiques aux erreurs d'arrondi près.

2. Soit la matrice

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 5 \end{bmatrix}$$

calculer la matrice $B = A^3$. Pour la matrice A calculer la matrice des vecteurs propres V et la matrice des valeurs propres D avec la commande `[V,D] = eig(A)`. Vérifier que la matrice $E = VD^3V^{-1}$ est identique à la matrice B.

13.23 nlp00

I.

On considère la fonction à une variable explicative suivante :

$$f(x) = -x^4 + 12x^3 - 15x^2 - 56x + 60 \quad .$$

Rechercher le maximum, soumis à aucune contrainte, de cette fonction moyennant la fonction `fmin` de MATLAB.

II.

Soit la fonction

$$f(x_1, x_2) = (x_1^2 + x_2 - 11)^2 + (x_1 + x_2^2 - 7)^2 \quad .$$

1. Donner une representation en 3 dimensions de la fonction $f(x_1, x_2)$ dans le domaine $x_{\in}[-4; 4]$ et $x_2 \in [-4; 4]$.
2. Representer graphiquement les courbes de niveau pour le domaine défini sous 1).
3. Avec la fonction `fmins` de MATLAB rechercher les extremas.

13.24 ode-ex1

Soit léquation différentielle ordinaire (la fonction ne fait intervenir qu'une seule variable)

$$\frac{\mathrm{d}\, y}{\mathrm{d}\, x} = -\frac{y}{x} \tag{64}$$

pour laquelle la solution générale est donnée par

$$y = \frac{C}{x}. \tag{65}$$

En remplaçant la dérivée par une approximation, on peut écrire l'équation différentielle (64) comme

$$\frac{f(x+h) - f(x-h)}{2h} = -\frac{f(x)}{x}$$

d'où l'on tire

$$f(x) = \frac{x}{2h}(f(x-h) - f(x+h)). \tag{66}$$

Pour pouvoir résoudre cette expression, il faut connaître deux points de la solution $f(x-h)$ et $f(x+h)$. Ecrivons (66) pour quatre points successifs :

$$\begin{aligned} y_1 &= x_1(y_0 - y_2)/2h \\ y_2 &= x_2(y_1 - y_3)/2h \\ y_3 &= x_3(y_2 - y_4)/2h \\ y_4 &= x_4(y_3 - y_5)/2h \end{aligned}$$

et l'on remarque qu'il s'agit d'un système linéaire

$$\begin{bmatrix} 1 & c_1 & & \\ -c_2 & 1 & c_2 & \\ & -c_3 & 1 & c_3 \\ & & -c_4 & 1 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix} = \begin{bmatrix} c_1 y_0 \\ 0 \\ 0 \\ -c_4 y_5 \end{bmatrix}$$

où $c_i = x_i/2h$, $h = x_i - x_{i-1}$ et que l'on peut résoudre si l'on connaît y_0 et y_5.

Ecrire un programme qui calcule, en se servant de l'expression (66), k approximations de l'équation (64), les points x_0, y_0 et x_{k+1}, y_{k+1} étant donnés.

Application : $x_0 = 2$, $y_0 = 1$, $x_{k+1} = 10$, $y_{k+1} = 0$ et $k = 10, 20, 50$. Faire un graphique qui compare la solution analytique avec les approximations calculées.

13.25 Poisson

On rappelle que la probabilité d'une variable aléatoire de Poisson Y de paramètre λ est définie comme

$$P(Y = y) = \frac{e^{-\lambda}\lambda^y}{y!}\,.$$

On désire générer n réalisations d'une variable aléatoire de Poisson de paramètre $\lambda = 2$.

1. Avec la fonction Matlab `poissrnd` générer le vecteur x de $n = 220$ réalisations.

A partir du vecteur x on désire obtenir une éstimation $\hat{\lambda}$ et puis la comparer au λ utilisé pour la génération. Cette estimation peut se faire en maximisant la probabilité conjointe

$$P = \frac{e^{-\lambda}\lambda^{x_1}}{x_1!} \cdots \frac{e^{-\lambda}\lambda^{x_n}}{x_n!}$$

par rapport au paramètre λ. Cette probabilité conjointe P pour le vecteur de réalisations x peut aussi s'écrire

$$P = \frac{e^{-n\lambda}\lambda^s}{x_1! \cdots x_n!} \quad \text{avec} \quad s = \sum_{i=1}^{n} x_i$$

2. Pour 100 valeurs de λ comprises entre 1.5 et 2.5 calculer les valeurs de P et faire le graphique de P en fonction de λ. (Utiliser `linspace`).
3. Créer une fonction Matlab `d = poissondv(lambda,x)` qui evalue la dérivée de la probabilité conjointe. Cette dérivée s'écrit

$$D = \frac{-e^{-n\lambda}\left(n\lambda^s - s\,\lambda^{s-1}\right)}{x_1! \cdots x_n!}$$

4. Produire le graphique de D en fonction de λ. Pour une génération de x donné les graphiques de P et D sont illustrés dans la figure 7.
5. Le maximum de la fonction P correspond au zero de la fonction D. Utiliser l'algorithme de la bissection pour rechercher le zero de D. Afin de pouvoir directement estimer le parameter λ en fonction d'un vecteur d'observations donnée, on le programmera comme suit :

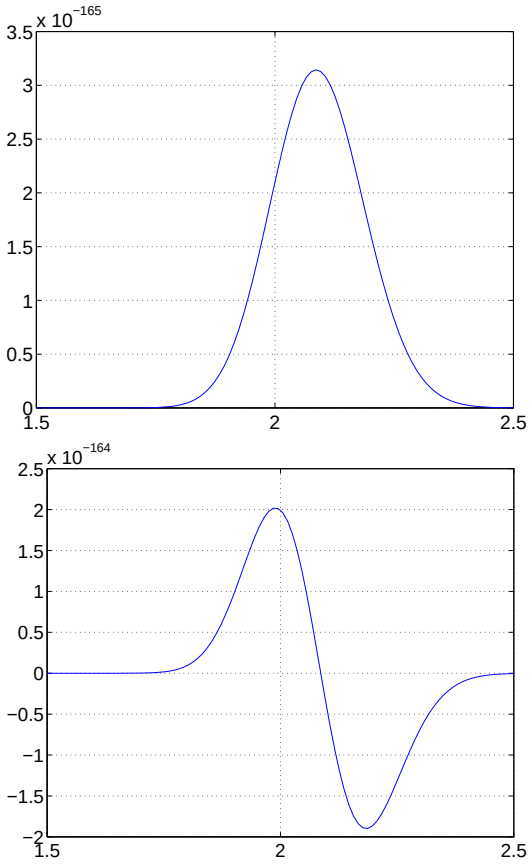


FIG. 7 – Graphiques de P et D.

```
function c = bissectp(a,b,x)
fa = poissondv(a,x);
fb = poissondv(b,x);
if sign(fa) == sign(fb)
   error('fonction n''est pas de signe oppose en a et b');
end
fait = 0;
c = a + (b - a) / 2;       % Chercher centre intervalle
%
while abs(b-a) > 2*tol & ~fait
   fc = poissondv(c,x);    % Evaluer f au centre
   if sign(fa) ~= sign(fc) % zero à gauche de c
      b  = c;
      fb = fc;
      c = a + (b - a) / 2;
   elseif sign(fc) ~= sign(fb) % zero a droite de c
      a  = c;
      fa = fc;
      c = a + (b - a) / 2;
   else                    % on tombe exactement sur zero
      fait = 1;
   end
end
```

Les évalutions de la fonction aux points a, b et c se font donc avec la fonction `poissondv` construite au point 3.

Etant donné que les valeurs de la fonction D sont très petites (de l'ordre de 10^{-150}) on a modifié le test `fa * fb < 0` par le test equivalent `sign(fa) ~= sign(fb)`. Que se passerait si l'on avait utilisé `fa * fb < 0` ?

6. Générer succesivement 10 vecteurs x et calculer le $\hat{\lambda}$ correspondant. Faire un graphique de $\hat{\lambda}_i$, $i = 1, \ldots, k$. Que constat'on ? Quel est l'écart type des $\hat{\lambda}_i$.

13.26 prog01

Ecrire une procédure Matlab qui évalue numériquement la dérivée d'une fonction. On rappelle qu'étant donné une fonction $y = f(x)$, on peut approcher la valeur de la dérivée au point x comme

$$y'_x = \frac{f(x + dx) - f(x)}{dx} .$$

Pour la fonction du problème précédent, évaluer la dérivée au point $x = 1.3$. On choisira $dx = 10^{-8}$.

13.27 prog02

Ecrire une fonction Matlab qui évalue numériquement les dérivées d'une fonction quelconque. On donnera à cette fonction la forme

```
d = derivnum('nom_fonction',x)
```

où `nom_fonction` est la chaîne de caractères qui correspond au nom de la fonction Matlab que l'on désire dériver. Indication : Il faudra faire appel à la fonction `feval` dans `derivnum`.

13.28 prog03

Soit l'expérience qui consiste à lancer deux dés. Le résultat auquel on s'intéresse est la somme des points des deux dés. L'ensemble des résultats possibles est donc constitué par

$$\Omega = \{\, 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ 10\ 11\ 12 \,\} .$$

On sait que les résultats n'ont pas tous la même probabilité étant donné qu'ils se composent des réalisations suivantes :

Résultat du dé 1 \ Résultat du dé 2	1	2	3	4	5	6
1	2	3	4	5	6	7
2	3	4	5	6	7	8
3	4	5	6	7	8	9
4	5	6	7	8	9	10
5	6	7	8	9	10	11
6	7	8	9	10	11	12

Sachant qu'il y a indépendence entre le résultat du dé 1 et le résultat du dé 2, on a la probabilité suivante pour chaque résultat possible :

i	2	3	4	5	6	7	8	9	10	11
$P(X = i)$	$\frac{1}{36}$	$\frac{2}{36}$	$\frac{3}{36}$	$\frac{4}{36}$	$\frac{5}{36}$	$\frac{6}{36}$	$\frac{5}{36}$	$\frac{4}{36}$	$\frac{3}{36}$	$\frac{2}{36}$

Générer 1000 résultats et comparer, à l'aide d'un graphique, la distribution empirique avec les probabilités théoriques.

13.29 prog05

On considère une variable aléatoire u uniformément distribuée dans l'intervalle $(0,1)$. On sait que l'espérance mathématique d'une telle variable aléatoire est $E(u) = 1/2$ et que sa variance est $V(u) = 1/12$.

On désire vérifier empiriquement ce résultat. Pour ce faire on génère n réalisations de u et on calcule la moyenne

$$\mathrm{Eu} = \frac{1}{n}\sum_{i=1}^{n} u_i \tag{67}$$

et l'écart quadratique moyen

$$\mathrm{Vu} = \frac{1}{n-1}\sum_{i=1}^{n}(u_i - \mathrm{Eu})^2 . \tag{68}$$

Le vecteur des n réalisations de u peut être généré avec la commande MATLAB

```
u = rand(n,1) ;
```

1. Générer un vecteur u avec 1000 réalisations de la variable aléatoire uniforme.
2. Programmer la procédure pour calculer Eu définie en (67).
3. Programmer la procédure pour calculer Vu définie en (68).
4. En développant l'expression (68) on obtient

$$\mathrm{Vu} = \frac{1}{n-1}\left(\sum_{i=1}^{n} u_i^2 - n\,\mathrm{Eu}^2\right) \tag{69}$$

$$\mathrm{Vu} = \frac{1}{n-1}\left(\sum_{i=1}^{n} u_i^2 - \frac{(\sum_{i=1}^{n} u_i)^2}{n}\right) \tag{70}$$

   A partir de l'expression (70) il est possible de calculer Vu (et Eu) avec une seule boucle.

   Ecrire le programme qui calcule Vu à partir de la définition donnée en (70).

   Indication : On accumulera dans la même boucle les u_i^2 et les u_i. Ainsi on aura, après avoir terminé la boucle, la somme des u_i^2 et la somme des u_i, ce qui permet de calculer Vu selon (70).

13.30 prog06

Programmer une fonction MATLAB qui calcule le produit des éléments d'un vecteur x. On appellera cette fonction `vprod` et on lui donnera la structure suivante :

```
function p = vprod(x)
si x est vide alors
  p = 1
sinon
  p = x(1)
  Calculer à l'aide d'une boucle p = p · x_2 ··· x_n
finsi
```

On pourra utiliser la fonction MATLAB `isempty` pour tester si le vecteur x est vide. Cette fonction retourne la valeur 1 si x est vide et la valeur 0 sinon.

En définissant `x = 1:n` l'appel `vprod(1:n)` retourne la valeur de $n!$.

1. Vérifier que $10! = 3628800$.
2. Combien d'opérations élémentaires (additions, soustractions, multiplications et divisions) exécute la fonction `vprod` pour un vecteur x avec n éléments ?

On considère l'expression du coefficient binomial

$$C_n^k = \frac{n!}{(n-k)!\,k!} \tag{71}$$

3. Ecrire les instructions MATLAB (et en vous servant de la fonction `vprod`) qui permettent d'évaluer C_n^k suivant (75) pour n et k données. Evaluer C_n^k pour $n = 50$ et $k = 5$.
4. Quel est le nombre d'opérations élémentaires nécessaires pour évaluer C_n^k.

On peut évaluer C_n^k plus efficacement comme

$$C_n^k = \frac{n\cdot(n-1)\cdots(n-k+1)}{k!} \tag{72}$$

5. Ecrire un code MATLAB qui evalue (76) pour $k = 0,1,\ldots,n$ et conserver les valeurs dans un vecteur C. Vérifier que

$$\sum_{k=0}^{n} C_n^k = 2^n$$

   et imprimer un message de confirmation. On structurera le code comme :

```
Initialiser le vecteur c
Calculer C_n^k pour k = 0,1,...,n et conserver les valeurs dans c
si ∑_{i=1}^{n+1} c_i = 2^n , alors
  Imprimer un message de confirmation
sinon
  Imprimer un message d'erreur
finsi
```

L'ensemble des coefficients binomiaux C_n^k, $n = 0,1,\ldots,m$ et $k = 0,1,\ldots,n$ est appellé *triangle de Pascal* étant donné que l'on peut les representer sous la forme

$$P = \begin{array}{c|cccccccc} & k & & & & & & & \\ n & 0 & 1 & 2 & 3 & 4 & \cdots & m \\ \hline 0 & C_0^0 & & & & & & \\ 1 & C_1^0 & C_1^1 & & & & & \\ 2 & C_2^2 & C_2^1 & C_2^2 & & & & \\ 3 & C_3^0 & C_3^1 & C_3^2 & C_3^3 & & & \\ 4 & C_4^0 & C_4^1 & C_4^2 & C_4^3 & C_4^4 & & \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \\ m & C_m^0 & C_m^1 & C_m^2 & C_m^3 & C_m^4 & \cdots & C_m^m \end{array}$$

On sait que l'on a

$$C_n^0 = C_n^n = 1, \qquad n = 0,1,\ldots,m\,, \tag{73}$$

et que

$$C_n^k = C_{n-1}^{k-1} + C_{n-1}^k, \qquad k = 1, \ldots, n-1\,. \tag{74}$$

6. En se servant des relations (73) et (74) écrire un code MATLAB qui évalue tous les coefficients du triangle de Pascal pour $m = 50$. On pourra s'inspirer du pseudo-code donné ci-après. On rappelle qu'avec MATLAB les indices d'un tableau (vecteur ou matrice) commencent avec 1.

Algorithme 8 Triangle de Pascal

```
Initialiser la matrice P d'ordre m+1
Initialiser l'élément P_11
pour i = 2 à m+1 faire
  Initialiser l'élément P_i1
  pour j = 2 à i-1 faire
    Calculer P_ij selon (74)
  finfaire
  Initialiser l'élément P_ii
finfaire
```

7. Quel est le nombre d'additions que necessite cette procédure. Donner une expression en fonction de m.

13.31 prog06-b

Programmer une fonction MATLAB qui calcule le produit des éléments d'un vecteur x. On appellera cette fonction `vprod` et on lui donnera la structure suivante :

```
function p = vprod(x)
si x est vide alors
  p = 1
sinon
  p = x(1)
  Calculer à l'aide d'une boucle p = p · x_2 ··· x_n
finsi
```

On pourra utiliser la fonction MATLAB `isempty` pour tester si le vecteur x est vide. Cette fonction retourne la valeur 1 si x est vide et la valeur 0 sinon.

En définissant `x = 1:n` l'appel `vprod(1:n)` retourne la valeur de $n!$.

1. Vérifier que $10! = 3628800$.
2. Combien d'opérations élémentaires (additions, soustractions, multiplications et divisions) exécute la fonction `vprod` pour un vecteur x avec n éléments ?

On considère l'expression du coefficient binomial

$$C_n^k = \frac{n!}{(n-k)!\,k!} \tag{75}$$

3. Ecrire les instructions MATLAB (et en vous servant de la fonction `vprod`) qui permettent d'évaluer C_n^k suivant (75) pour n et k données. Evaluer C_n^k pour $n = 50$ et $k = 5$.
4. Quel est le nombre d'opérations élémentaires nécessaires pour évaluer C_n^k.

On peut évaluer C_n^k plus efficacement comme

$$C_n^k = \frac{n \cdot (n-1) \cdots (n-k+1)}{k!} \tag{76}$$

5. Ecrire un code MATLAB qui evalue (76) pour $k = 0, 1, \ldots, n$, $n = 50$, et conserver les valeurs dans un vecteur C. Vérifier que

$$\sum_{k=0}^{n} C_n^k = 2^n\,.$$

13.32 prog07

1. On considère les instructions Matlab

```
x = ceil(8*rand(1,10));
p = (x > 4);
```

qui produisent le vecteur `p` défini comme

$$p(i) = \begin{cases} 1 \text{ si } & x(i) > 4 \\ 0 \text{ sinon} & \end{cases}\,.$$

A l'aide d'une boucle `for` programmer une procédure qui construit le vecteur `p`. Vérifier qu'elle fournit le même résultat que `p = (x > 4)`.

2. Les instructions Matlab

```
x = ceil(8*rand(1,10));
p = find(x > 4);
```

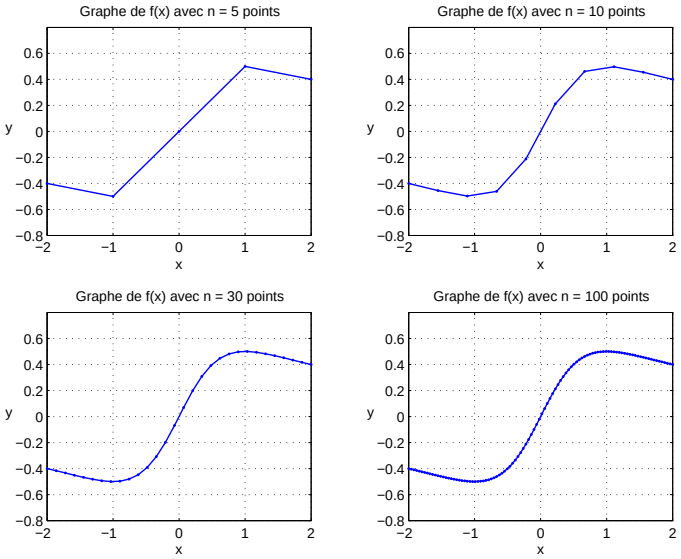
produisent le vecteur `p` des indices des éléments du vecteur `x` qui sont supérieurs à 4. A l'aide d'une boucle `for` programmer une procédure qui construit le vecteur `p`. Vérifier qu'elle fournit le même résultat que `find`.

13.33 prog08

Boucles `for` et vectorisation :
- Créer une matrice $A = [a_{ij}]$ de dimension 10×10 où $a_{ij} = \sin(i)\cos(j)$ à l'aide de boucles `for`.
- Comment peut-on obtenir le même résultat sans utiliser de boucle `for` ?
- Avec la commande `mesh` produire un graphique de la surface $z = \sin(x)\cos(y)$ pour $x \in [1, 10]$ et $y \in [1, 10]$.

Opérateurs logiques :
- Créer un vecteur ligne x de longueur 7 dont les éléments sont des nombres tirés au hazard de la distribution normale standard.
- Trouver les éléments de x qui sont supérieurs à 1 ou inférieurs à -0.2.
- Trouver les éléments de x qui sont supérieurs à 0 et inférieurs à 2.
- Quelle est la position des éléments supériurs à 3 dans le vecteur x ?

Graphisme :
Dessiner le graphe de la fonction $f(x) = \frac{x}{1+x^2}$ pour 5, 10, 30 et 100 valeurs équidistantes de x comprises dans l'intervalle $[-2, 2]$ en reproduisant la figure suivante.



13.34 ras

Un des problèmes courants rencontrés dans la modélisation input-output consiste, partant d'une matrice A donnée, à construire une matrice S dont la somme des lignes est égale à un vecteur u et la somme des colonnes égale à un vecteur v.

La méthode RAS[18] constitue un algorithme efficace pour obtenir une approximation de la solution à ce problème. Cet algorithme s'énonce comme suit :

Initialiser $S^{(0)} = A$
`Pour` $i = 1, 2, \ldots,$ *jusqu'à convergence*, `faire`
$a^{(i)} = (\text{diag}(S^{(i-1)}\imath))^{-1}\, u$
$S^{(i)} = \text{diag}(a^{(i)})\, S^{(i-1)}$
$b^{(i)} = (\text{diag}(\imath' S^{(i)}))^{-1}\, v'$
$S^{(i)} = S^{(i)}\, \text{diag}(b^{(i)})$
`finfaire`

Critère d'arrêt : $\|a^{(i)} - a^{(i-1)}\|_2 < \epsilon$ et $\|b^{(i)} - b^{(i-1)}\|_2 < \epsilon$

Lorsque l'algorithme se termine la matrice $S^{(i)}$ correspond à la matrice S recherchée.

Le symbole $\imath$ désigne le vecteur unitaire et l'opérateur diag désigne la transformation d'un vecteur en une matrice diagonale.

1. Programmer la méthode RAS telle qu'elle est décrite à l'aide de MATLAB en utilisant les fonctions `diag` et `inv` notamment.

[18] Bacharach M. (1970) : *Biproportional Matrices and Input-Output Change*, Cambridge University Press.

2. Exécuter le programme avec les données proposées ci-après. Imprimer la matrice S calculé et donner le nombre de flops utilisés.

$$A = \begin{bmatrix} 60 & 20 & 42 & 35 & 80 & 23 \\ 36 & 10 & 84 & 70 & 29 & 44 \\ 24 & 70 & 14 & 19 & 32 & 65 \\ 10 & 60 & 30 & 20 & 40 & 50 \\ 62 & 22 & 11 & 52 & 12 & 39 \\ 44 & 92 & 33 & 71 & 31 & 82 \end{bmatrix} \quad u = \begin{bmatrix} 230 \\ 290 \\ 225 \\ 215 \\ 170 \\ 280 \end{bmatrix}$$
$$v = \begin{bmatrix} 240 & 270 & 220 & 270 & 100 & 310 \end{bmatrix}$$

3. Programmer une nouvelle version de l'algorithme dans laquelle les différentes instructions sont codées de sorte à engendrer moins d'opérations élémentaires. (On évitera notamment l'utilisation de `diag` et `inv`.)

4. Exécuter ce nouveau programme sur les données et donner le nombre de flops utilisés. (En choisissant $\epsilon = 0.01$ pour le critère d'arrêt le résultat devrait être obtenu en moins que 1500 flops.)

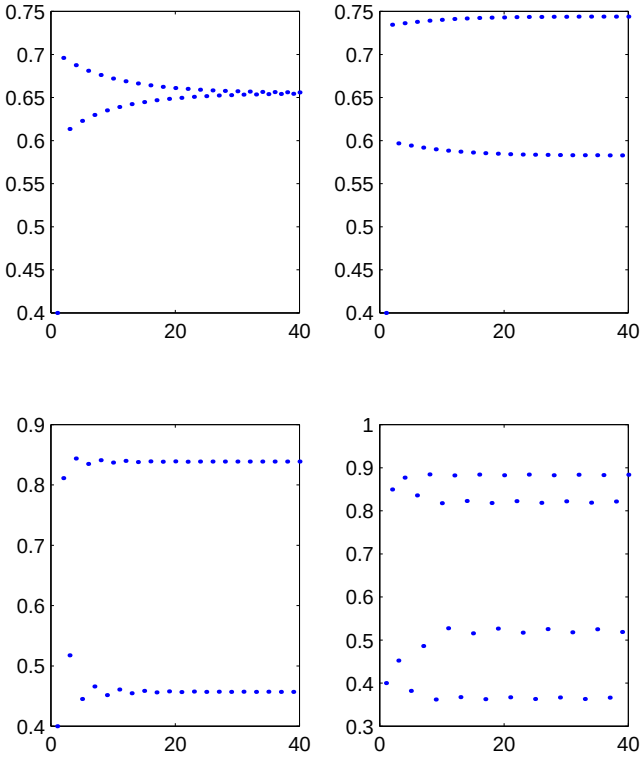
5. Comparer les résultats trouvés en 2 et 4. Expliquer de manière générale d'où provient la différence du nombre d'opérations.

13.35 Recurrence

Soit la suite récurrente (x_n) avec

$$\begin{cases} x_1 = 0.4 \\ x_{n+1} = r\, x_n\, (1 - x_n) \qquad n = 1, 2, \ldots \end{cases} \tag{77}$$

En vous servant de la commande `subplot` produire pour des valeurs equidistants de r allant de 2.9 à 3.7 les graphiques suivants :



13.36 rsl00

On rappelle q'un polynôme de degré p s'écrit

$$y = c_p x^p + c_{p-1} x^{p-1} + \cdots + c_0 \,.$$

Considérons quatre points (x_i, y_i), $i = 1, 2, 3, 4$ qui appartiennent à un polynôme de degré $p = 3$. Les coefficients de ce polynôme correspondent à la solution du système linéaire $Ac = y$ avec

$$A = \begin{bmatrix} x_1^3 & x_1^2 & x_1 & 1 \\ x_2^3 & x_2^2 & x_2 & 1 \\ x_3^3 & x_3^2 & x_3 & 1 \\ x_4^3 & x_4^2 & x_4 & 1 \end{bmatrix} \quad c = \begin{bmatrix} c_3 \\ c_2 \\ c_1 \\ c_0 \end{bmatrix} \quad y = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix} .$$

L'application se fera avec $x = \begin{bmatrix} 1 & 3 & 5 & 9 \end{bmatrix}'$ et $y = \begin{bmatrix} 3 & 7 & 1 & -3 \end{bmatrix}'$.

1. Ecrire le code qui construit la matrice A pour x et p donné.

```
n = length(x)
pour i = 1 : n faire
   pour j = 0 : p faire
      A_{i,j+1} = ...
   finfaire
finfaire
```

2. Résoudre le système linéaire $Ac = y$ en procédant comme suit :[19]

   Factoriser $A = LU$ (afin de simplifier on aura pas besoin de pivoter)
   Résoudre $Lz = y$
   Résoudre $Uc = z$

3. Faire le graphique des points donnés et du polynôme calculé.

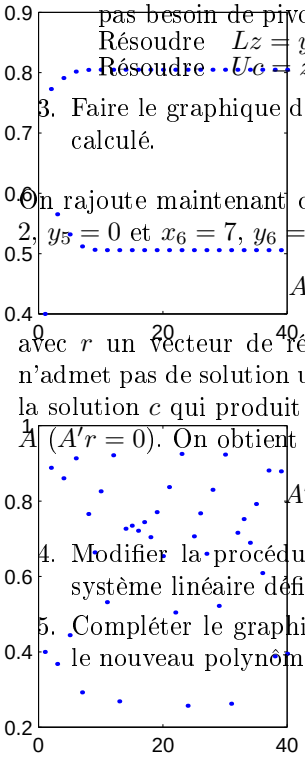
On rajoute maintenant deux point supplémentaires $x_5 = 2$, $y_5 = 0$ et $x_6 = 7$, $y_6 = 3$. On peut alors écrire l'égalité

$$Ac = y + r$$

avec r un vecteur de résidus. Ce système surdéterminé n'admet pas de solution unique, mais on privilégie souvent la solution c qui produit un résidu r qui est orthogonal à A ($A'r = 0$). On obtient alors le sytème linéaire

$$A'Ac = A'y \tag{78}$$

4. Modifier la procédure du point 2) pour résoudre le système linéaire défini en (78).
5. Compléter le graphique avec les nouveaux points et le nouveau polynôme.



13.37 rsl01b

Soit le système linéaire $Ax = b$ de dimension n. La règle de Cramer permet de calculer la solution x de ce système en ayant recours aux déterminants par la formule suivante,

$$x_i = \frac{|A_i|}{|A|} \quad , \quad i = 1, 2, \ldots, n \tag{79}$$

où A_i est la matrice A dans laquelle on a remplacé la i-ème colonne par le vecteur b.

On suppose que chaque déterminant est calculé selon le schéma d'expansion des mineurs. Le déterminant de la matrice A s'écrit alors,

$$|A| = \sum_{p \in P(n)} s(p) \prod_{i=1}^{n} a_{i\,p_i} \tag{80}$$

où

$P(n)$ est l'ensemble des $n!$ permutations de l'ensemble $\{1, 2, \ldots, n\}$,
p_i est le i-ème élément de la permutation p,
$s(p)$ est une fonction prenant les valeurs $+1$ ou -1 selon p.

1. Développer une formule donnant approximativement le nombre d'opérations en virgule flottante à effectuer pour obtenir la valeur du déterminant d'une matrice d'ordre n d'après l'équation (80).
2. Donner une formule pour le nombre d'opérations utilisées pour résoudre le système linéaire avec la règle de Cramer donnée sous (79).
3. Quel temps nécessite la résolution d'un système de taille $n = 90$ sur un ordinateur pouvant effectuer :
   (a) 522 Kflops/seconde = 522000 flops/seconde (PC-386),

[19] Il n'est pas permis d'utiliser l'opérateur \ de Matlab.

(b) 5200 Kflops/seconde (DEC AXP 3000-800S),

(c) 33000 Kflops/seconde (Cray X-MP),

(d) 1 Teraflops/seconde = 10^{12} flops/seconde (Ordinateur parallèle ?).

Commenter.

4. Avec MATLAB, générer un système de taille $n = 90$ et le résoudre. Combien d'opérations en virgule flottante a-t-on effectué et combien de temps cela a-t-il pris ?

13.38 rsl01

Soit la règle de Cramer pour la résolution d'un système linéaire $Ax = b$

$$x_i = \frac{|A_i|}{|A|} \qquad i = 1, \ldots, n$$

où A_i est la matrice A dans laquelle on a remplacé la $i^{\text{ème}}$ colonne par le vecteur b. On suppose que chaque déterminant est évalué selon le schéma de l'expansion des mineurs.

1. Estimer le nombre d'opérations élémentaires nécessaires pour résoudre le système $Ax = b$ de taille n=90.
2. Quel temps nécessite une machine parallèle, capable d'effectuer 1 téraflops (10^{12}), pour effectuer ce calcul ?
3. Quelle conclusion peut-on tirer de ce résultat ?

Rappel : Le déterminant d'une matrice A peut s'écrire comme

$$|A| = \sum_{p \in P(n)} s(p) \prod_{i=1}^{n} a_{ip_i}$$

où $P(n)$ est l'ensemble des $n!$ permutations de l'ensemble $I = \{1, 2, \ldots, n\}$, p_i est le $i^{\text{ème}}$ élément de la permutation p et $s(p)$ est une fonction qui prend la valeur de $+1$ ou -1.

13.39 rsl02

On considère un modèle économétrique constitué d'un système de 8 équations, qui pour une période t donnée s'écrivent :

$$\begin{aligned} C &= a_o + a_1 P + a_2 P_{-1} + a_3 W \\ I &= b_0 + b_1 P + b_2 P_{-1} + b_3 K_{-1} \\ W1 &= c_0 + c_1 E + c_2 E_{-1} + c_3 TM \\ Y &= C + I + G - T \\ P &= Y - W1 \\ K &= K_{-1} + I \\ W &= W1 + W2 \\ E &= Y + T - W2 \end{aligned}$$

En écrivant les équations sous la forme implicite, c'est à dire par exemple pour la première équation

$$a_0 + a_1 P + a_2 P_{-1} + a_3 W - C = 0$$

on peut formuler le système comme

$$Ay + Ey_{-1} + Fz = 0$$

avec y le vecteur des 8 variables endogènes $[C \;\; I \;\; W1 \;\; Y \;\; P \;\; K \;\; W \;\; E]'$, y_{-1} est le vecteur des mêmes variables mais considérées avec un retard d'une période et $z = [1 \;\; W2 \;\; T \;\; G \;\; TM]'$ est le vecteur des variables exogènes.

1. Donner les matrices A, E et F.

Une quantification du modèle attribue les valeurs suivantes aux paramètres a, b et c du modèle :

a_o	16.5498	b_o	20.3886	c_o	4.8182
a_1	.0209	b_1	.1466	c_1	.4701
a_2	.2137	b_2	.6190	c_2	.1076
a_3	.8098	b_3	-.1583	c_3	.3064

et on a observé les valeurs suivantes pour les variables aux périodes $t-1$ et t :

	C	I	W1	Y	P	K	W	E	W2	T
t-1	61.6	1.3	41.6	60.6	19.0	201.4	49.4	61.7	7.8	8.
t	65.0	3.3	45.0	66.1	21.1	204.5	53.0	67.7	8.0	9.

2. Ecrire le modèle sous la forme

$$Ay = b$$

et déterminer numériquement b.

On se propose maintenant de résoudre ce système avec les méthodes itératives de Jakobi et Gauss-Seidel respectivement, soit :

$$\begin{aligned} y^{(k+1)} &= By^{(k)} + g \qquad k = 0, 1, 2, \ldots \\ (L + D)y^{(k+1)} &= -Uy^{(k)} + b \qquad k = 0, 1, 2, \ldots \end{aligned}$$

L, U et D sont des matrices triangulaires inférieures, supérieures et diagonales respectivement, telles que l'on vérifit

$$A = L + D + U$$

3. Quels sont la matrice B et le vecteur g pour la méthode de Jakobi.
4. Pour la méthode de Gauss-Seidel, donner les matrices L, D et U et montrer qu'étant une particularité de D on peut écrire

$$y^{(k+1)} = Ly^{(k+1)} + Uy^{(k)} - b \qquad k = 0, 1, 2, \ldots$$

Commenter la structure des matrices B et U.

5. Donner, pour les deux méthodes, les matrices qui "*gouvernent*" la convergence. Donner le module des valeurs propres des ces deux matrices. (On rappelle que le module d'une valeur propre λ est égale à $\sqrt{r^2 + i^2}$, où r est la partie réelle et i la partie imaginaire de la valeur propre).

Que peut-on dire quant à la façon dont les 2 méthodes vont converger ?

6. Trouver une solution pour y en utilisant la méthode de Jakobi puis celle de Gauss-Seidel.
   On choisira pour $y^{(o)}$ les valeurs des variables observées pour la période $t-1$.
   Comme critère d'arrêt on choisira
   $$Max \mid \frac{y^{(k+1)} - y^{(k)}}{y^{(k)}} \mid < \epsilon$$
   où Max est le maximum des éléments du vecteur et $\epsilon = .001$.

La structure causale de ce système $Ay = b$, peut être représentée au moyen d'un graphe. L'ensemble des sommets est donné par les éléments du vecteur y et un élément a_{ij} non-null de la matrice A correspond à l'éxistence d'un arc $j \rightarrow i$. On ne tiendra pas compte des éléments diagonaux de A qui introduisent simplement des boucles sur chaque sommet.

7. Donner le schéma fléché du graphe qui correspond au système linéaire $Ay = b$ défini sous 2).

On décide maintenant de permuter les lignes et les colonnes de la matrice A dans l'ordre suivant : 4 8 3 7 5 2 1 6.

8. A quoi correspond cette opération de permutation ? Résoudre ce nouveau système avec Jakobi et Gauss-Seidel.
9. On considère maintenant la permutation des lignes et des colonnes de la matrice originale A dans l'ordre : 6 1 2 5 7 3 8 4. Que peut-on attendre quand à la performance de Gauss-Seidel ? Résoudre le système avec Gauss-Seidel.

13.40 rsl03

Soit l'expression suivante :

$$x = B^{-1}(2A + I)(C^{-1} + A)b \quad .$$

1. Montrer comment on peut calculer le vecteur x sans avoir recours à aucune inversion de matrice.
2. Comment peut-on obtenir l'inverse de A en résolvant n systèmes linéaires du type $Ax = b$ avec l'élimination de Gauss ? Donner un exemple numérique.

13.41 rsl05

Avec MATLAB programmer une fonction qui transforme, selon le schéma de l'élimination de Gauss, une matrice A donnée en une matrice triangulaire supérieure. La fonction doit pouvoir s'appeler comme suit :

$$A = \mathbf{trisup}(A)$$

Application : Chercher la matrice triangulaire supérieure qui correspond à la matrice de Toeplitz $a_{ij} = 1/(i+j-1)$ d'ordre 10.

Quel est le nombre d'opérations élémentaires nécessaires pour effectuer cette triangularisation ?

13.42 rsl07

1. Programmer l'algorithme 3.2.1 pour le calcul du vecteur de la transformation de Gauss et l'algorithme 3.2.2, qui applique la transformation de Gauss à une matrice C, sous la forme de fonctions MATLAB. La structure des fonctions sera la suivante :
   - `t = vecgauss(x)`, avec $x = A(k : n, k)$ le sous-vecteur de la matrice A à l'étape k.
   - `C = trgauss(C,t)`, avec C la matrice à laquelle on applique la transformation de Gauss.
2. Programmer l'algorithme 3.2.3 qui transforme une matrice A en une matrice de forme triangulaire supérieure. Donner à la fonction la structure suivante :
   $$\texttt{A} = \texttt{trisup(A)}$$
3. Programmer l'algorithme 3.2.4 qui applique l'élimination de Gauss à une matrice A. Donner à la fonction la structure suivante :
   $$\texttt{A} = \texttt{elgauss(A)}$$
   Générer une matrice aléatoire A d'ordre 10 et tester l'algorithme. Imprimer L et U et vérifier que $LU = A$.
4. Pour chaque algorithme établir à l'aide de MAPLE la fonction du nombre d'opérations élémentaires (flops) pour transformer une matrice d'ordre $n \geq 3$. Vérifier le résultat empiriquement. De quel ordre est la complexité de ces algorithmes ?

13.43 rsl10

Programmer l'algorithme 4.1.2 qui, pour une matrice symétrique A produit la factorisation $A = LDL'$. La structure de cette fonction MATLAB sera la suivante :

$$[\texttt{A}] = \texttt{ldl(A)} \quad .$$

Comparer empiriquement, en factorisant quelques matrices aléatoires, le nombre d'opérations élémentaires entre les algorithmes `egpp` et `ldl`. Quel est le rapport des `flops` respectifs.

13.44 rsl11

1. Programmer l'algorithme 4.1.3 de Cholesky qui factorise une matrice A symétrique définie positive en $A = GG'$. La structure de cette fonction MATLAB sera la suivante :
   $$[\texttt{A}] = \texttt{cholesky(A)} \quad .$$

2. Etablir la fonction du nombre d'opérations élémentaires (flops) de cet algorithme.
3. Générer une matrice aléatoire X d'ordre 100×5 et un vecteur aléatoire y de dimension 100. Estimer par la méthode des moindres carrés les paramètres du modèle

$$y = Xb + u \quad .$$

Comparer le nombre de flops de votre procédure avec le nombre de flops nécessaires pour évaluer `b=inv(X'*X)*X'*y`.

13.45 rsl12

1. Programmer l'algorithme 3.2.4 qui applique l'élimination de Gauss à une matrice A. Donner à la fonction la structure suivante :

$$\texttt{A} = \texttt{elgauss(A)}$$

Générer une matrice aléatoire A d'ordre 10 et tester l'algorithme. Imprimer L et U et vérifier que $LU = A$.

2. Pour l'algorithme 3.2.4 établir à l'aide de MAPLE la fonction du nombre d'opérations élémentaires (flops) pour transformer une matrice d'ordre $n \geq 3$. Vérifier le résultat empiriquement. De quel ordre est la complexité de cet algorithme ?

13.46 rsl13

1. Programmer l'algorithme 3.2.5 qui applique l'élimination de Gauss avec pivotage partiel à une matrice A. Donner à la fonction la structure suivante :

$$\texttt{A} = \texttt{egpp(A)}$$

Générer une matrice aléatoire A d'ordre 10 et tester l'algorithme. Imprimer L et U et vérifier que $LU = A$.

13.47 rsl14

Créer une matrice A d'ordre $n = 7$ et un vecteur colonne b de dimension $n = 7$. Compter le nombre d'opérations élementaires effectuées à l'aide de la fonction `flops`, lorsque l'on résoud le système linéaire $Ax = b$:

1. en utilisant la fonction `rsl` vu au cours,
2. en utilisant l'opérateur \ de MATLAB,
3. en utilisant l'inverse de A calculée par la fonction `inv`.

Répéter ces opérations pour des systèmes de taille $n = 10, 20, 30, 40, 50$ et dessiner l'évolution du nombre de flops en fonction de n.

Que peut-on dire ?

13.48 SprintModel

Pritchard[20] a suggeré le modèle suivant

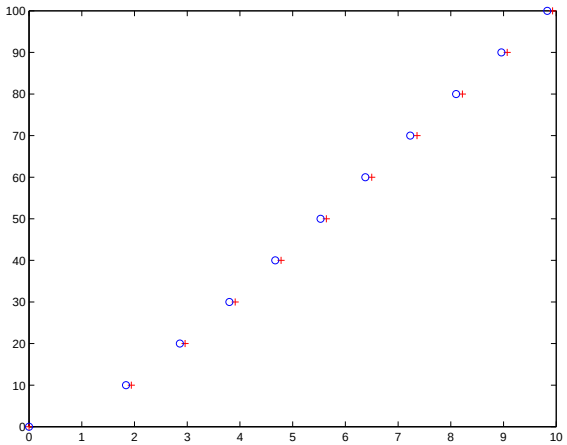
$$x(t) = \alpha\tau\left(t - \tau(1 - e^{-t/\tau})\right) \tag{81}$$

pour décrire la performance d'un sprinter. Ce modèle explique la distance x (en mètres) parcourue en fonction du temps t en secondes. Les coefficients α et τ sont des paramètres.

Lors des championnats du monde à Rome en 1987 on a mésuré, lors de la finale, les temps suivants pour Carl Lewis et Ben Johnson :

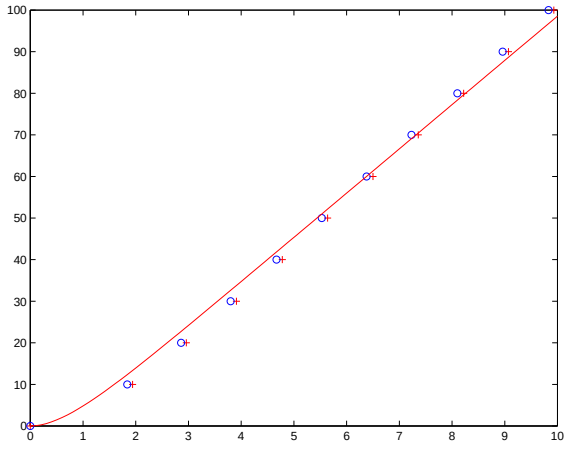
x en mètres	0	10	20	30	40	50	60	7
Lewis (sec)	0	1.94	2.96	3.91	4.78	5.64	6.50	7
Johnson (sec)	0	1.84	2.86	3.80	4.67	5.53	6.38	7

1. Construire trois vecteurs `x`, `tL` et `tJ` qui contiennent la distance (pour `x`) et les temps de Lewis (`tL`) respectivement de Johnson (`tJ`). Produire le graphique suivant :

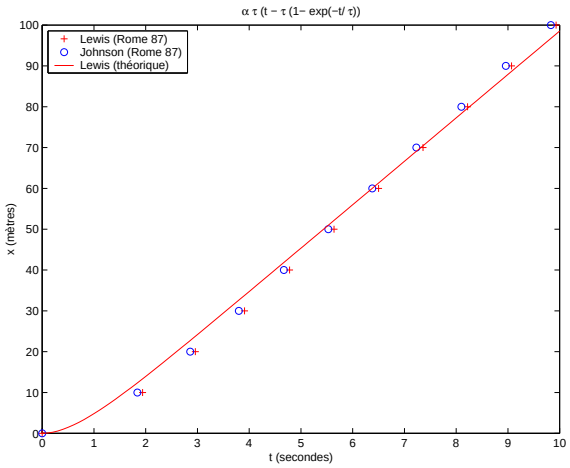


2. Pour Lewis on établi qu'on a $\alpha = 14.4$ et $\tau = 0.739$. Avec la commande Matlab `inline` construire la fonction qui permet d'évaluer l'expression (81) pour les valeurs de α et τ qui correspondent a Lewis. En se servant de la commande Matlab `linspace` construire un vecteur `t` avec 100 valeurs equidistantes allant de 0 à 100. Evaluer les distances théoriques `xth` qui correspondent aux valeurs du vecteur `t`. Completer le graphique précédant avec cette courbe.

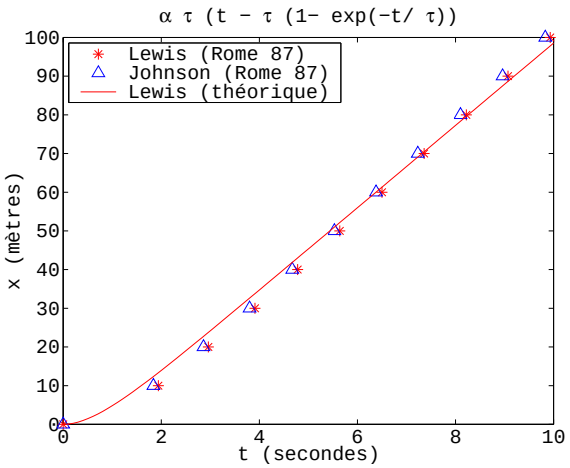
[20] W.G. Pritchard, (1993). Mathematical Models of Running. *SIAM Review* **35** :3, 359–379.



3. Annoter le graphique comme illustré ci-après :



4. Changer la taille et la fonte des annotations du graphique afin d'obtenir un résultat qui ressemble au graphique qui suit :



13.49 SparseLinSys01

Soit le système linéaire $Ax = b$ avec

$$A = \begin{bmatrix} -1 & 0 & 0 & 0 & a_1 & 0 & a_2 & 0 \\ 0 & -1 & 0 & 0 & a_3 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 & 0 & a_4 \\ 1 & 1 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 1 & -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & -1 \end{bmatrix}, \quad b = \begin{bmatrix} -20 \\ -.3 \\ -14 \\ 2.2 \\ 0 \\ -200 \\ -8 \\ -1.6 \end{bmatrix}$$

avec $a_1 = .2\ a_2 = .8\ a_3 = .5\ a_4 = .47$.

On envisage la résolution de ce système linéaire avec la méthode de Gauss-Seidel, c'est-à-dire en procédant de la manière suivante :

$B = A - I$
Initialiser le vecteur x
Jusqu'à convergence, `faire`
$x = -Bx + b$
`finfaire`

Afin d'éviter des opérations redondantes, celles qui impliquent les éléments nuls lors du produit de la matrice avec le vecteur, on pourrait proposer la construction de la matrice caractères suivante

```
eq=['a1*x(5)+a2*x(7)'
    'a3*x(5)        '
    'a4*x(8)        '
    'x(1)+x(2)      '
      ...
      ...           '];
```

ce qui permettrait d'évaluer le i$^{\text{ème}}$ produit scalaire, sans effectuer des opérations redondantes, à l'aide de la commande `eval(eq(i, :))`.

Programmer une version de cet algorithme de Gauss-Seidel qui évite les opérations redondantes en utilisant les matrices creuses de MATLAB (c'est-à-dire sans recourir à la matrice définie ci-dessus, ni à la fonction `eval`).

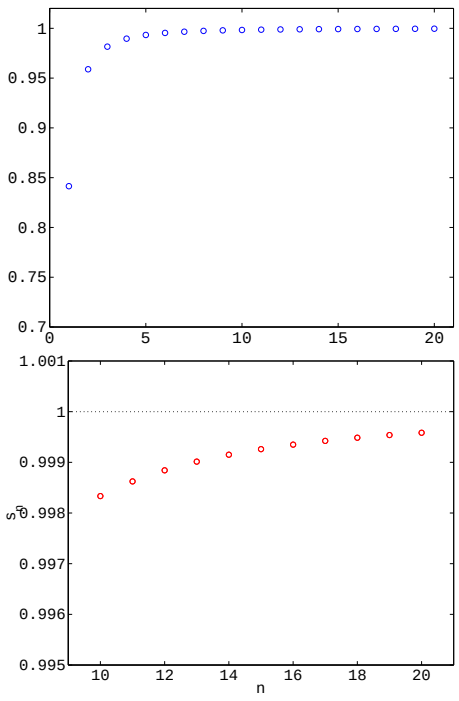
Vérifier avec la fonction `flops` le nombre d'opérations élémentaires effectivement exécutées.

13.50 Suites

On sait que la suite de terme général $s_n = n \sin(1/n)$ on a

$$\lim_{n\to\infty} s_n = 1\,. \tag{82}$$

1. Avec `inline` definir la fonction qui evalue s_n.
2. Calculer s_n pour quelques valeurs de n afin de vérifier la relation (82).
3. Calculer les valeurs de s_n pour $n = 1$ à 20. Reproduire les graphiques ci-dessous.



13.51 svd-ex1

Une application possible pour l'approximation d'une matrice par une matrice de rang inférieur constitue la compression de données lors de la transmission d'un très grand nombre d'images prises par un satellite, par exemple.

Pour ce faire, on digitalise au préalable une image en lui appliquant une grille de n lignes et n colonnes puis on assigne à chaque carré de la grille un nombre qui correspond à son niveau de luminosité suivant une échelle de 0 à 10 par exemple. Ainsi l'image est traduite en n^2 entiers qu'il faudra par la suite transmettre sur terre.

En procédant de la sorte, le volume de données à transmettre peut cependant devenir rapidement prohibitif lorsqu'il s'agit d'un très grand nombre d'images. Une possibilité consiste alors à appliquer à la matrice des luminosités une décomposition singulière et à ne transmettre que les $2k$ vecteurs singuliers ainsi que les k valeurs singulières qui permettent une approximation satisfaisante de la matrice originale. Ainsi, si une approximation de rang 6 pour une matrice d'ordre 1000 s'avère suffisante, le nombre de données à transmettre serait de $2 \times 6 \times 1000 + 6 = 12006$ au lieu de 10^6, ce qui correspond à un gain de $98,8\%$.

Application : Le fichier `v:\metri\mne\svd_ex1.mat` (à lire avec la commande `load`) contient une matrice A d'ordre 100 qui correspond à une image digitalisée avec deux niveaux de luminosité codés avec 0 et 1. Pour visualiser cette image avec MATLAB on peut se servir des instruction `axis('ij'), spy(flipud(A))`.

En se basant sur l'étude des valeurs singulières rechercher l'approximation B de plus petit rang de la matrice A qui soit acceptable pour la qualité de cette image. (Pour la matrice approchée B on imprimera une marque si $b_{ij} > .4$)

13.52 svd-ex2

On désire effectuer une estimation d'un modèle de régression multiple par les moindres carrés ordinaires en utilisant les résultats de la décomposition singulière.

Soit X une matrice de données de rang complet k et de dimension $n \times k$ et soit y un vecteur d'observations de dimension $n \times 1$. On cherche à calculer l'estimation $\hat{\beta}$ pour le modèle $y = X\beta + \varepsilon$ comme la solution du problème $\min_{\beta} \|X\beta - y\|_2^2$.

D'autre part, la décomposition singulière de $X = USV'$ est supposée connue avec $U = [u_1\, u_2 \cdots u_n]$ et $V = [v_1\, v_2 \cdots v_k]$, deux matrices orthogonales. On peut alors calculer $\hat{\beta}$ par la formule

$$\hat{\beta} = \sum_{i=1}^{k} \frac{u_i' y}{\sigma_i} v_i \ ,$$

et la somme des carrés des résidus $\mathrm{SSR} = \hat{\varepsilon}'\hat{\varepsilon}$ par

$$\mathrm{SSR} = \sum_{i=k+1}^{n} (u_i' y)^2 \ .$$

1. En utilisant la commande `svd` de MATLAB, vérifier que $X = U_1 S_1 V_1'$ avec $U_1 = [u_1\, u_2 \cdots u_k]$, $V_1 = V = [v_1\, v_2 \cdots v_k]$, $S_1 = \mathrm{diag}(\sigma_1, \sigma_2, \ldots, \sigma_k)$ sur les données pour X chargées grâce à l'instruction `load v:\metri\mne\test2` . (3 points)
2. Ecrire l'expression matricielle de $\hat{\beta}$ en fonction de U_1, V_1, S_1 et y. (15 points)
3. Ecrire l'expression matricielle de SSR en fonction de $U_2 = [u_{k+1}\, u_{k+2} \cdots u_n]$ et y. (10 points)
4. En utilisant les expressions trouvées aux points 2 et 3, programmer l'algorithme pour calculer $\hat{\beta}$ et SSR sur les données X et y chargées au point 1.
   (Vous pouvez vérifier vos résultats en utilisant l'une des autres façons de calculer $\hat{\beta}$ et SSR) (5 points)
5. En utilisant le fait que le pseudo-inverse de X, noté X^+, peut se calculer, lorsque X est de rang complet, soit par $(X'X)^{-1}X'$, soit par $V_1 S_1^{-1} U_1'$, donner une expression pour le calcul de $(X'X)^{-1}$ qui ne nécessite que les matrices V_1 et S_1. (Indication : Calculer $X^+(X^+)'$) (10 points)
6. Donner une estimation non biaisée de la matrice des variances et covariances de $\hat{\beta}$ en appliquant la formule trouvée au point précédent et calculer les valeurs des statistiques t de Student pour les paramètres estimés. (Utilisez `inv` si vous n'avez pas résolu le point 5) (5 points)

13.53 vis-00

On considère le modèle linéaire suivant :

$$I = \tilde{\beta}_1 + \tilde{\beta}_2 GNP + \tilde{\beta}_3 p + \tilde{\beta}_4 r + e \qquad (83)$$

dont les observations sont données dans le fichier `v:\metri\arml.dat`.

L'estimation par la méthode des moindres carrées des paramètres β_i, $i = 0, 1, 2, 3$ de ce modèle correspond à la solution du problème de minimisazion suivant :

$$\min_{\beta} e'e = (y - X\beta)'(y - X\beta)$$

En vous servant uniquement des outils de visualisation de MATLAB donner les valeurs (approximatives) des paramètres qui minimisent cette somme de carrées.

13.54 xxx-00

Avec LaTeX réproduire l'énoncé du problème 1 jusqu'au point 1.