Ministère de l'Education Nationale,
de l'Enseignement Supérieur, de la Formation
des Cadres et de la Recherche Scientifique

Fondation ACADEMIA

Les Journées Informatiques
des Classes Préparatiores aux Grandes Ecoles
Edition 2009

29 Octobre-2 Novembre 2009
Rabat-Maroc

Documents de travail

TP : Matlab©

Said. EL HAJJI
PhD en Mathématiques Appliquées, Université Laval, Canada.
Professeur de l'enseignement supérieur
Faculté des Sciences de Rabat
http://www.fsr.ac.ma/mia/elhajji.htm
elhajji@fsr.ac.ma

Université Mohammed V Agdal-
Faculté des Sciences– Rabat
Laboratoire Mathématiques, Informatique et Applications

Exercice 1 :

On note u1=[1 ,2, 3], u2=[-5, 2 ,1], u3=[-1, -3, 7]

- **Définir ces vecteurs sous Matlab.**
- **Calculer u1+ u2, u1 + 3u2 - 5u3, u3/5**
- **Calculer norm1(u1); norm2(ul); norm1(u2); norminf(u3);**
- **Calculer le cosinus de l'angle formé par les vecteurs U1et U2.**

Exercice 2 :

Calculer les déterminants, inverses, valeurs propres et vecteurs propres de chacune des matrices

A=[2 3 ; 6 5] et B=[2 3 4 ;7 6 5 ;2 8 7]

Exercice 3 On note

$$A = \begin{pmatrix} 5/8 & -1/4 & 1/8 \\ 1/4 & 0 & 1/4 \\ 1/8 & -1/4 & 5/8 \end{pmatrix}, b = \begin{pmatrix} 1 \\ -1 \\ 1 \end{pmatrix}, u_0 = \begin{pmatrix} 5 \\ 2 \\ -4 \end{pmatrix}$$

et on définit, pour $n \geq 0$, la suite de vecteurs $u_{n+1} = Au_n + b$.

1. Calculer les premiers termes de la suite u_n, qu'observez-vous?
2. Même question avec $_{\text{et}}$ Interpréter les résultats

$$B = \begin{pmatrix} 5 & 6 & 3 \\ -1 & 5 & -1 \\ 1 & 2 & 0 \end{pmatrix}, b = \begin{pmatrix} 1 \\ -1 \\ 1 \end{pmatrix}, u_0 = \begin{pmatrix} 2 \\ 1 \\ 0 \end{pmatrix}$$

Exercice 4

1. Soit la matrice A symétrique définie par:

$$A = \begin{pmatrix} 1.4025 & 0.5975 & -1.3404 & 1.0921 \\ * & 0.538 & 1.2141 & -0.774 \\ * & * & 0.0096 & 2.907 \\ * & * & * & 0.0499 \end{pmatrix}$$

   (a) Calculer le déterminant, la trace et l'inverse de la matrice A.

   (b) Calculer le polynôme caractéristique de A, chercher ses racines.

   (c) Rechercher les valeurs propres et les vecteurs propres de A. Vérifier sur un de ces vecteurs qu'il est bien vecteur propre.

   (d) Calculer le conditionnement de A.

2. Soit $x = (1111)$. Calculer sa norme 1, sa norme 2, sa norme ∞.

Exercice 5 On considère la matrice tridiagonale d'ordre n définie par

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

- Que fait la séquence d'instructions suivante ?

1

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

2

$$D = diag(ones(n,1));$$
$$SD = diag(ones(n-1,1),1);$$
$$A = 2 * D - SD - SD';$$

1. **construire *An*** par concaténation et extraction !
Pour plus d'informations, taper help **eye**, help **zeros** dans la ligne de commande.
2. **construire*An*** en utilisant la commande **diag**,
Pour plus d'informations, taper help **diag** dans la ligne de commande.
3. Il *est* recommandé de sauvegarder dans un fichier une suite d'instructions que l'on veut utiliser plusieurs fois ; ainsi la procédure "**proc**" est un ensemble d'instructions contenues le fichier de nom "**proc.m**", et qui seront exécutées en tapant simplement proc dans la ligne de commande de **Matlab.**
4. Ecrire une procédure pour construire la matrice *A* pour *n* quelconque et résoudre ensuite un système linéaire $\boldsymbol{Ax = b}$.
5. Mesurer le temps calcul pour plusieurs valeurs de *n,* à l'aide de la commande **cputime** (ou encore **tic** et **toc**).
6. Faire l'expérience pour une matrice *A* définie en stockage plein (tableau à deux dimensions), puis en stockage creux **(A=sparse(n,n)).**

7. Déterminer de manière expérimentale le nombre d'opérations *N(n)* que nécessite une résolution de système en fonction de *n*, pour le stockage plein et le stockage creux. Tracer la courbe *N(n)* en échelle naturelle (utiliser la commande plot), puis en échelle log-log (utiliser la commande loglog).
   • A l'aide de Matlab, proposer une méthode expérimentale permettant de montrer que la **matrice *A* est définie positive**.
   • **Représenter graphiquement en échelle log-log** le conditionnement de la matrice *A* en fonction de *n.* Commenter.

Université Mohammed V Agdal-
Faculté des Sciences– Rabat
Laboratoire Mathématiques, Informatique et Applications

1. Matrice compagnon: Soit $P(X) = X^n + a_{n-1}X^{n-1} + ... + a_0$ un polynôme de degré n. La matrice compagnon de ce polynôme $C(P) \in M_n(R)$ est définie par:

$$C(P) = \begin{pmatrix} 0 & 1 & 0 & .. & 0 \\ 0 & 0 & 1 & .. & 0 \\ & & & .. & \\ 0 & 0 & 0 & .. & 1 \\ -a_0 & -a_1 & -a_2 & .. & -a_{n-1} \end{pmatrix}$$

Nous allons écrire la matrice compagnon du polynôme $P1(X) = X^7 + 12 * X^6 + 2 * X^4 + X^3 - X^2 + 6$ de manière simple avec les commandes de Matlab.

(a) Définir le polynôme $P1$ avec Matlab (attention a l'ordre des termes $P1 = (\ a_n \quad a_{n-1} .. \ a_0\ )$)

(b) Construire le vecteur ligne $(\ -a_0 \quad -a_1 \quad .. \quad -a_{n-1}\ )$ à partir du polynôme $P1$.

(c) Construire la matrice identité de taille 6*6.

(d) Construire le vecteur colonne nul de R^6.

(e) Construire, à partir de ces 3 objets la matrice compagnon de P1.

(f) regarder l'aide sur compan. La matrice compagnon a des propriétés particulières:

(g) Calculer le polynôme caractéristique de la matrice compagnon $C(P1)$. Que remarque-t-on?

2. Tracer sur un même graphique le graphe des fonctions $P_k : x \mapsto x^k$ pour $k = 0$ à $k = 3$.

Exercice 2

Soit les polynômes: $a(x)=x^3+2x^2+3x+4$, $b(x)=x^3+4x^2+9x+16$

a- Multiplier et Additionner ces deux polynômes.

b- Soit c le polynôme obtenu après la multiplication de a et b, faire la division de c par b.

c- tracer la courbe: $b(x)= x^3+4x^2+9x +1$

Trouver les racines du polynôme: $P=x^4-12x^3+25x+116$

Exercice 3

1. Ecrire un script qui affiche la somme, puis la différence et enfin le produit des Matrices A et B .
2. Programmer en faisant appel à l'instruction for le Calcul de l'aire d'une courbe par la méthode des trapèzes.
3. Ecrire un script affichant sur le même graphe les deux fonctions $f_1(x)$= x.*cos(x) et $f_2(x)$= x.*x sin(x)

a) Ecrire une fonction sous le nom %fonction marche f(x) =1 si x>O et f(x)=0 si x<O

b) Ecrire une fonction %fonction en forme de N: f(x)= -x si x<1 et f(x)=0 sinon

4. Ecrire une fonction qui calcule la somme de chacune des colonnes de A, A doit être une matrice *(4*5)*.

5. Ecrire une fonction résolvant l'équation x.*x -x -1 ou n est le Nombre d'itération demandée.

6. Ecrire une fonction qui résout un équation du deuxième degré, ax.*x+bx+c=0 en prenant comme donnée ses coefficients a,b et c Les solutions sont renvoyées dans x1 et x2.

7. Les fichiers .m

construire les fonctions suivantes (un fichier.m par fonction)

a) f: [0, 1] dans R, fonction en escalier avec au moins une discontinuités dans] 0, 1[.

b) f: [0, 1] dans R, fonction affine par morceaux (non C' sur]0, 1[).
Tracer le graphe (voir fplot) des fonctions précédentes et les fonctions suivantes:
c) f: [0, 1] dans R, fonction régulière,
d) f: [0, 1] dans R, fonction régulière périodique.

Exercice 8:
a)donner l'algorithme de Méthode de Newton pour résoudre f(x)=0

b) Ecrire la fonction Newton.m : function x=Newton(x0, n)

% Remarque: commencer toujours le programme par des lignes de commentaires

% initialisation a x0, on fait n itérations.

%la fonction f et sa différentielle df sont respectivement définies dans les fichiers fnewton.m et df.m.

c) Ecrire la fonction convergence_newton.m :

% pour le polynôme P=x^4+x^3-23x^2+3x+90, étudier la vitesse de convergence de la méthode de Newton % a partir de x0=0, on tombe sur la racine -5.

Université Mohammed V Agdal-
Faculté des Sciences– Rabat
Le Laboratoire Mathématiques, Informatique et Applications

Objectif : On veut résoudre des Systèmes linéaires.

Méthodes directes - Méthodes itératives.

1. Ecrivez un programme pour résoudre le système d'équations suivant

$\mathbf{x_1+2x_2+3x_3+4x_4=1}$
$\mathbf{2x_1-3x_2+4x_3+5x_4=2}$
$\mathbf{3x_1+4x_2-5x_3+6x_4=3}$
$\mathbf{4x_1+5x_2+6x_3-7x_4=4}$

Par la méthode de décomposition $\mathbf{U^tDU}$ de **Choleski** de sa matrice. La matrice du système est-elle définie positive? Comparez votre résultat à celui que fournit La procédure standard de Matlab.

2. Sans écrire un programme général résolvez le système suivant

$\mathbf{9x_1-2x_2-2x_3=11}$
$\mathbf{2x_1+2x_2+8x_3=13}$
$\mathbf{2x_1-9x_2+2x_3=15}$

a) par la méthode de **Jacobi**,
b) par la méthode de **Gauss Seidel**

Comparez les résultats obtenus du point de vue de la vitesse de convergence. Concluez.

3. Les méthodes itératives de Jacobi et Gauss-Seidel s'écrivent respectivement

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

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

Ecrivez un programme qui permet de résoudre un système linéaire par la méthode de **Jacobi** et résolvez le système suivant à l'aide de ce programme:

$\mathbf{9x_1-2x_2-2x_3+2x_4+2x_5=1}$
$\mathbf{2x_1+0x_2+8x_3-2x_4+2x_5=2}$
$\mathbf{2x_1-9x_2+2x_3+2x_4-2x_5=3}$
$\mathbf{0x_1-2x_2+2x_3+2x_4+8x_5=4}$
$\mathbf{2x_1+2x_2+2x_3-9x_4-2x_5=5}$

Modifiez votre programme pour appliquer la méthode de **Gauss- Seidel** au système précédent. Comparez les résultats obtenus.

Outils mathématiques

Ahmed Kanber
Professeur agrégé de mathématiques
Chargé d'inspection de mathématiques en CPGE
kanber@ucam.ac.ma

Outils Mathématiques

A.Kanber: chargé d'Inspection

kanber@ucam.ac.ma.

Journée D'informatiques II, Rabat, Maroc, 29octobre - 2 Novembre

Sommaire

1: Dénombrement

Le nombre d'éléments d'un ensemble fini E, appelé cardinal sera noté $|E|$.

Rappels : on suppose que $|E| = n, |F| = m$ et que $p \in [|1; n|]$.

- Le nombre d'application de E vers F est égal à m^n : ainsi $|F^E| = |F|^{|E|}$
- Lorsque $m = n$, le nombre de bijection de E sur F est égal à $n!$.
- Le nombre d'injection de $[|1; p|]$ vers $[|1; n|]$ est égal à $A_n^p = \frac{n!}{(n-p)!} = n(n-1)\dots(n-p+1)$.
- Le nombre de parties de E ayant p éléments est égal à $C_n^p = \frac{A_n^p}{p!} = \frac{n!}{p!(n-p)!}$

2: Ensembles dénombrables

Definition

un ensemble est dit dénombrable si et seulement si il existe u ne bijection de $\mathbb{N}$ sur E.
Ainsi, on peut alors théoriquement "numéroter" tous les éléments de E et écrire que $E = \{x_n, n \in \mathbb{N}\}$.

Proposition

- *Toute partie d'un ensemble dénombrable est finie ou dénombrable.*
- *Si $E_1, \ldots, E_p$ sont dénombrables, alors $E_1 \times E_2 \times \ldots \times E_p$ est dénombrable*
- *Si I est dénombrable et si $\forall i \in I, E_i$ est dénombrable, alors $\bigcup_{i \in I} E_i$ est dénombrable*

Exemples:

- $\mathbb{Z}, \mathbb{Z} \times \mathbb{Z}, \mathbb{Q}$ sont dénombrables;
- $\{0,1\}^{\mathbb{N}}, \mathcal{P}(\mathbb{N}), \mathbb{R}, [0,1]$ ne sont pas dénombrable.

Coefficients binomiaux

- $C_n^{n-p} = C_n^p$ et $C_{n-1}^{p-1} + C_{n-1}^p = C_n^p$ (Triangle de Pascal)
- $\sum_{k=0}^{n} C_n^k = 2^n$, $\sum_{k=1}^{n} kC_n^k = n2^{n-1}$, $\sum_{k=1}^{n} k^2 C_n^k = n(n+1)2^{n-2}$
- $\sum_{k=0}^{p} C_m^k C_n^{p-k} = C_{n+m}^p$

Suites géométriques

Pour *pourx* $\neq 1$,, $\sum_{k=0}^{n} x^k = \frac{1-x^{n+1}}{1-x}$.

- $\sum_{k=0}^{n} x^k = \frac{1-x^{n+1}}{1-x}$
- $\sum_{k=0}^{n} kx^k = \frac{nx^{n+2}-(n+1)x^{n+1}+1}{(1-x)^2}$
- $\sum_{k=0}^{n} k^2x^k =$ *dériver*

Comportement Asymptotique de Suites

On se limitera dans ce chapitre au cas des suites réelles positives.

Definition

On dit que (u_n) est dominé par (v_n), ce qu'on écrit $u_n = O(v_n)$

$$\exists \lambda > 0,\ \exists n_0\ \in \mathbb{N}/\ \forall n \geq n_0\ \ u_n \leq \lambda v_n$$

Proposition

$$O(O(u_n)) = O(u_n),\ O(u_n)+O(u_n) = O(u_n),\ \ \lambda O(u_n) = O(\lambda u_n) = O(u_n)$$

et

$$v_n O(u_n) = O(u_n v_n) = O(u_n)O(v_n)$$

Définition

On dit que deux suites (u_n) et (v_n) sont semblable ce qu'on note $u_n = \theta(v_n)$ $u_n = O(v_n)$ et $v_n = O(u_n)$

Exemple : Si $\lim \frac{u_n}{v_n} = l \neq 0$ alors, $u_n = \theta(v_n)$

Définition

On dit (u_n) est négligeable devant (v_n), ce qu'on écrit $u_n = o(v_n)$

$$\forall \epsilon > 0,\ \exists n_0\ \in \mathbb{N}/\ \forall n \geq n_0\ \ u_n \leq \epsilon v_n$$

Ce quit est équivalent à dire :

$$\exists (\epsilon_n) \to 0 \text{ et } n_0\ \in \mathbb{N}/\ \forall n \geq n_0 u_n = \epsilon_n v_n$$

Exemples de Références

Pour $\alpha, \beta > 0$ et $a > 1$ on a :

$$n^\alpha = o(n^\beta) \Leftrightarrow \alpha < \beta,\ ln^\beta(n) = o(n^\alpha),\ n^\alpha = o(a^n),\ a^n = o(n!)$$

Equivalence

Définition

On dit que (u_n) *et* (v_n) *sont équivalente, ce qu'on écrit* $u_n \sim v_n$ $|u_n - v_n| = o(v_n)$ *ce qui est équivalent à dire :*

$$\exists(\epsilon_n) \to 0 \text{ et } n_0 \in \mathbb{N} / \ \forall n \geq n_0 u_n = (1+\epsilon_n)v_n$$

Remarque

La relation $\sim$ *définie une relation d'équivalence sur* $\mathbb{R}^{\mathbb{N}}$

Proposition

Si $u_n \sim u'_n$ *et* $v_n \sim v'_n$ *alors*

- $u'_n = O(v_n) \Rightarrow u_n = O(v_n)$
- $u'_n = o(v_n) \Rightarrow u_n = o(v_n)$

Sommation des Relations de Comparaisons

Soit (u_n) et (v_n) deux suites positives. On note $V_n = \sum_{k=0}^{n} v_k$ et $U_n = \sum_{k=0}^{n} u_k$. Si $\lim_{n\to+\infty} V_n = +\infty$ On a les résultats suivants :

- si $u_n = O(v_n)$, alors $U_n = O(V_n)$
- si $u_n = o(v_n)$, alors $U_n = o(V_n)$
- si $u_n \sim v_n$, alors $U_n \sim V_n$

Comportement Asymptotique de certain somme

On suppose que f est une fonction à valeurs positive décroissante, continue sur $[p, +\infty[$. On pose $S_n = \sum_{k=p+1}^{n} f(k)$. On a alors l'encadrement suivant :

$$\int_{p+1}^{n+1} f(x)dx \leq S_n \leq \int_{p}^{n} f(x)dx$$

Exemples classiques :

$$1 + \frac{1}{2} + \ldots + \frac{1}{n} \sim \ln n, \quad lnn! \sim n \ln n, \quad \sum_{k=1}^{n} k^{\alpha} \sim \frac{n^{\alpha+1}}{\alpha+1}$$

Suites Récurrentes Linéaire de Premier Ordre

Se sont des suites vérifiant une relation de récurrence sous la forme $u_{n+1} = a(n)u_n + b(n) \quad (E)$.// On se propose de donner des méthodes permettant, lorsque c'est possible, de calculer explicitement u_n en fonction de n, ou à défaut de donner un équivalent de u_n.

- Principe $\mapsto$ au cas particulier $a(n) = 1$.
- On pose $p(n) = \prod_{k=0}^{n} a(k)$ $\circlearrowleft$ le changement de $v_n = \frac{v_n}{p(n)}$.
- On obtient alors

$$v_{n+1} = v_n + c(n), \quad c(n) = \frac{b(n)}{p(n+1)}$$

Récurrence $u_{n+1} = au_n + br^n$

- Méthode rapide: Ici

$$p(k) = a^k, v_k = \frac{u_k}{a^k}$$

On obtient :

  - si $r \neq a$ alors $u_k = \alpha a^k + \beta r^k$
  - si $r = a$ alors $u_k = (\alpha + \beta k) r^k$

- En se ramenant au $2^{ième}$ ordre : A partir des relations $u_k = au_{k-1} + br^k$ et $u_{k-1} = au_{k-2} + br^{k-1}$ on obtient :

$$u_k - (a + r)u_{k-1} + aru_{k-2} = 0$$

Application à des suites fréquentes : Coûts d'Algorithmes

$T(n)$ une suite croissante positive vérifiant :

$$\forall n = 2^k, \;\; T(n) = aT(\frac{n}{2}) + \theta(n^p)$$

On s'intéresse d'abord à la suite extraite $kT(2^k)$
On introduit les deux suites définie par :

$$u_0 = T(0) \quad u_{k+1} = au_k + \gamma(2^{k+1})^p \quad \forall k \geq 0 \qquad (1)$$
$$v_0 = T(0) \quad v_{k+1} = av_k + \delta(2^{k+1})^p \quad \forall k \geq 0 \qquad (2)$$

Référence

Référence : D'après un cours d'Informatique que j'ai suivi dans la classe de mon ancien ami et collègue DANIEL GENOUD au lycée Lamartinière Mon plaisir à Lyon

Equations Non Linéaires

A.Kanber : chargé d'Inspection

kanber@ucam.ac.ma.

Journée D'informatiques II, Rabat, Maroc, 29octobre - 2 Novembre

Sommaire

1: Introduction

Le numéricien est souvent confronté à la résolution d'équation algébrique

$$f(x) = 0$$

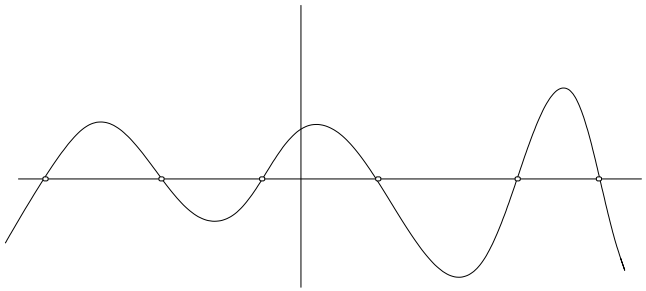
1: Position du problème

Soit $f : \mathbb{R} \longrightarrow \mathbb{R}$ une fonction donnée. On désire trouver une ou plusieurs solutions de l'équation $f(x) = 0$.
f est supposé continu et dérivable autant de fois qu'il est nécessaire sur l'intervalle où sont cherché les racines.

1: Séparation des racines

Definition

Une racine s de f est dite séparé dans un intervalle I si s est la seul solution de $f(x) = 0$ dans I



Séparation des Racines

Méthode de la bissection : Dichotomie

Considérons l'équation $f(x) = 0$ où la fonction f vérifie $f(a)f(b) < 0$. On pose $x_0 = \frac{a+b}{2}$ on a trois cas :

- $f(x_0) = 0$
- $f(a)f(x_0) < 0$, on pose alors $x_1 = \frac{x_0+a}{2}$
- $f(b)f(x_0) < 0$, on pose alors $x_1 = \frac{x_0+b}{2}$

- 1. Ètant donné $[x_1, x_2]$ ou f possède un changement de signe.
- 2. Ètant donné $\epsilon > 0$ (critère d'arrêt), N le nombre maximum d'itération.
- 3. Posons $x_m = \frac{x_1+x_2}{2}$
- 4.Si $\frac{|x_2-x_1|}{2x_m} < \epsilon$
  - convergence atteinte.
  - écrire $f(x_m)$
  - arrêt.
- 5. écrire $f(x_1), f(x_2), f(x_m)$
- 6.si $f(x_m)f(x_1) < 0$, alors $x_2 = x_m$
- 7.si $f(x_m)f(x_2) < 0$, alors $x_1 = x_m$
- 8. si le nombre d'itération est N est atteint :
  - Convergence non atteint
  - Arrêt
- 9.Retour à 3

Méthode des approximations successives

Definition

Un point fixe d'une f sur un intervalle I est une valeur $x \in I$ telle que $f(x) = x$

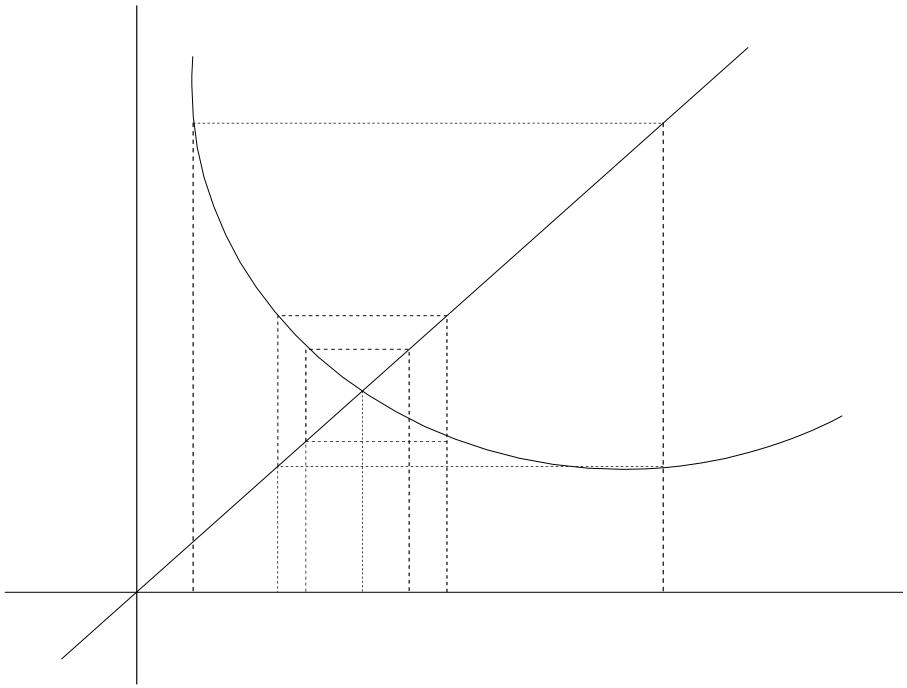
Theorem

Si f est continu sur $[a, b]$ à valeur dans $[a, b]$ alors f admet un point fixe.
si f est dérivable sur $[a, b]$ et s'il existe $L \in]0, 1[$ tel que $\forall x \in [a, b] \quad |f'(x)| < L$ alors f admet un unique point fixe dans $[a, b]$

Principe

L'algorithme des approximation successive consiste à construire la suite (x_n) définie par :

$$\begin{cases} x_0 \in I, & \text{arbitraire;} \\ x_{n+1} = f(x_n), & . \end{cases}$$



Algorithme

- 1. Ètant donné $\epsilon > 0$(critère d'arrêt), N nombre maximum d'itération.
- 2. x_0 une valeur estimée initiale du point fixe
- 3. Effectuer $x_{n+1} = f(x_n)$
- 4.Si $\frac{|x_{n+1}-x_n|}{x_{n+1}} < \epsilon$
  - convergence atteinte.
  - écrire x_{n+1}
  - Arrêt.
- 5. si le nombre d'itération est N est atteint :
  - Convergence non atteint
  - Arrêt
- 6.Retour à 3

Importance

La méthode de Newton est l'une des méthode les plus utilisées pour la résolution des équations non linéaires

Principe

à partir d'une valeur initial x_0 de la solution, on cherche une correction δ telle que

$$f(x_0 + \delta) = 0$$

En faisant un développement de Taylor autour de x_0 on trouve :

$$0 = f(x_0) + \delta f'(x_0) + \frac{\delta^2}{2} f''(x_0) + \dots$$

Si on néglige les termes d'ordre supérieure à 2 on obtient :

$$0 \simeq f(x_0) + \delta f'(x_0)$$

ce qui donne :

$$\delta = -\frac{f(x_0)}{f'(x_0)}$$

Algorithme

- 1. Ètant donné $\epsilon > 0$, le critère d'arrêt
- 2. Ètant donné N le nombre maximum d'itération.
- 3. x_0 une valeur initial de la solution
- 4. Effectuer $x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$
- 5.Si $\frac{|x_{n+1}-x_n|}{x_{n+1}} < \epsilon$
  - convergence atteinte.
  - écrire x_{n+1}
  - Arrêt.
- 6. si le nombre d'itération est N est atteint :
  - Convergence non atteint
  - Arrêt
- 7.Retour à 4

Programmes MAPLE

Voir TP MAPLE

Référence

Analyse numérique pour l'ingénieur André Fortin

Structures de données avec Maple©

Boujaida Sadik
Professeur agrégé de mathématiques
Enseignant en classes de PSI
CPGE My Youssef
Rabat, Maroc
jsadikster@gmail.com

Structures de données avec Maple

Par Sadik BOUJAIDA

Introduction

Le typage des données sous Maple est trés varié. Il fournit tous les types utilisés dans les langages classiques et offre différentes méthodes de construction et de manipulation pour les types composites tels les les listes, tableaux et chaines de caractères. Maple en tant qu'interpreteur de commande n'offre pas de commandes de gestion de la mémoire (à la manière d'un langage compilé et même parfois interpreté), cette fonction est dévolue à son "ramasse-miettes" (gabrage collector).
Une conséquence de ce choix est que la declaration des types de données, bien que possible avec Maple, n'est pas obligatoire. C' est au moment de l'initiation d'une variable que son type est reconnu. En outre les regles de conversion de types sont trés permissives. Une variable qui, par exemple, à l'origine a reçu une donnée de type tableau, peut se voir changer de valeur en un nombre réel par une simple affectation. Bien que ceci puisse nuire à la robustesse et à la rigueur d'un programme écrit sous Maple, ce comportement est bien adapté à une utilisation intéractive.
Un autre aspect de la gestion des types de données sous Maple est la manière dont les variables sont évaluées à l'éxecution : certaines sont immédiatement remplacées par leurs contenus, alors que pour d'autre cette substitution ne se fait que jusqu'au dernier nom reçu par la variable.
Maple offre des procédures de contrôle du type de donnée d'une variable (type, whattype , ...) , ce qui est très utile pour effectuer des branchements par exemple. La procédure *convert* est specialisée dans la conversion explicite d'un type vers un autre, d'un tableau vers un ensemble, ou d'une écriture décimale vers une écriture en binaire par exemple. D'autres procédure permettent de sélectionner parmis plusieurs objet ceux répondant à certains critères, notament le type de donnée (*select*,*remove* , ...).

Types de données simples : les types booliens et les nombres.

Comme pour tout langage de programmation, le type boolien est bien présent, il prend les valeurs logiques *true* ou *false*.

En plus des variables de type boolien, Maple gére nativement tous les types de nombres (entiers, rationnels, réels et complexes) sans autre limite pour leur taille que la quantité de mémoire disponible dans le système.
Chaque type de nombres peut être representé sous différents types de données (pour répondre aux besoins en calcul formel). Une variable donnée, qui peut être l'un des argument d'une procédure peut être asservie à un type bien précis en utilisant la procédure *assume*.

Les nombres entiers

Les entiers (type *integer*), peuvent être de taille arbitrairemnt grande. il se subdivisent en plusieurs sous--type: *posint, negint*, respectivement pour entiers strictement positif ou negatif , et leurs négations *nonposint* et *nonnegint*.

```
> type(2,posint);type(-3,nonposint);type(0,negint);type(0,
  nonposint);
```

true
true **(1)**

false

true

Les nombres rationnels

Les rationnels (type *rational*), sont manipulés avec leurs valeurs exactes sous forme de fractions, et non avec des valeurs approchées en virgules flottantes. Signalons qu'il existe un type *fraction* qui est en fait un rationnel qui n'est pas un entier.

```
> 1/2+1/3;1/(5!);
```

$$\frac{5}{6}$$

$$\frac{1}{120} \quad \textbf{(2)}$$

```
> type(1/2,rational);type(0.5,rational);
```

true

false **(3)**

```
> type(1/2,fraction);type(2,fraction);
```

true

false **(4)**

Nombres réels

Pour les nombres réels, on dispose de plusieurs types qui se chevauchent dans leurs domaines de validité :

realcons : pour toute constante réelle, même representée sous forme symbolique tels que pi.
numeric : pour toute variable representée sous forme numérique.
float : pour toute variable representant un nombre à virgule flottante.

```
> restart:
> type(Pi,realcons);type(Pi,numeric);type(Pi,float);
```

true

false

false **(5)**

```
> type(.5,realcons);type(.5,numeric);type(.5,float);
```

true

true

true **(6)**

```
> type(1/2,realcons);type(1/2,numeric);type(1/2,float);
```

true

true

false **(7)**

la procédure evalf peut donner une valeurs approchée en virgules flotantes de n'importe quel nombre, avec une précision par défaut de 8 chiffres flottants
evalf(a,n) permet en outre d'evaluer *a* avec *n* chiffres flottants.

```
> evalf(Pi);
```

3.141592654 **(8)**

```
> evalf(Pi,75);
```

3.14159265358979323846264338327950288419716939937510582097494459230781640629 **(9)**

On peut fixer pour toute la session le nombre de chiffre flottant dans la représentation des floats grace à la variable d'environnement *Digits*.

```
> Digits:=25:
```

```
> evalf(Pi);
```

$$3.14159265358979323846264 \tag{10}$$

Nombres complexes

Les nombres complexes, comme pour les réels, peuvent être representés sous différents type de données;
complex : un complexe écrit obligatoirement sous forme algebrique $x+Iy$, où x et y sont des réels de type realcons, à signaler que si l'une des composantes x ou y est de type float, l'autre est automatiquement convertie en ce type.
complexcons : constante complexe qui peut se presenter sous forme algebrique ou trigonometrique.

```
> type(exp(2*I*Pi/5),complex);type(exp(2*I*Pi/5),complexcons);
```

$$\mathit{false}$$
$$\mathit{true} \tag{11}$$

```
> exp(2*I*Pi/5);
```

$$e^{\frac{2}{5} I \pi} \tag{12}$$

evalc(z) permet d'ecrire z sous forme algebrique quelquesoit la représentation d'origine de z avec la particularité que tout nom intervenant dans la définition de z est considéré comme un réel. *Im(z)* et *Re(z)* remplissent le rôle que leurs noms indiquent, mais z doit être fourni sous forme algébrique.

```
> z:=exp(2*I*Pi/5)/(a+2*I);
```

$$z := \frac{e^{\frac{2}{5} I \pi}}{a + 2 I} \tag{13}$$

```
> evalc(z);
```

$$\frac{\cos\left(\frac{2}{5}\pi\right) a}{a^2+4} + \frac{2\sin\left(\frac{2}{5}\pi\right)}{a^2+4} + I\left(\frac{\sin\left(\frac{2}{5}\pi\right) a}{a^2+4} - \frac{2\cos\left(\frac{2}{5}\pi\right)}{a^2+4}\right) \tag{14}$$

```
> Im(z);Im(evalc(z));
```

$$\Im\left(\frac{e^{\frac{2}{5} I \pi}}{a + 2 I}\right) \tag{15}$$
$$\Im\left(\frac{\cos\left(\frac{2}{5}\pi\right) a}{a^2+4} + \frac{2\sin\left(\frac{2}{5}\pi\right)}{a^2+4} + I\left(\frac{\sin\left(\frac{2}{5}\pi\right) a}{a^2+4} - \frac{2\cos\left(\frac{2}{5}\pi\right)}{a^2+4}\right)\right)$$

```
> Z:=x+I*y;
```

$$Z := x\sim + I\, y\sim \tag{16}$$

```
> Im(Z);
```

$$y\sim \tag{17}$$

```
> type(Z,complex);type(Z,complex(name));
```

$$\mathit{false}$$
$$\mathit{true} \tag{18}$$

```
> assume(x,real);assume(y,real);
> type(Z,complex);Im(Z);
```

$$\mathit{false}$$
$$y\sim \tag{19}$$

Séquences, Listes et ensembles:

Les séquences, listes et ensembles sont des objets composite, mais dont la gestion reste simple: ils

obéissent aux même regles d'evaluation que les types numériques, substitution immédiate du nom d'une variable par son contenu.

Séquences

Une séquence est une liste (au sens littéral, non au sens Maple) d'objets de types quelconques, séparés par des virgules.

```
> S:=0,I,sin,matrix(2,2,1);
```

$$S := 0, I, \sin, \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \quad (20)$$

On peut accéder aux opérandes d'une séquence en utilisant l'opérateur d'indexation [ ]

```
> S[3];S[1..3];S[-1];S[-2..-1];
```

$$\sin \quad (21)$$

$$0, I, \sin$$

$$\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$$

$$\sin, \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$$

la procédure *seq* sert à la construction de séquences structurées . Son utilisation peut être sous la forme *seq(f(k),k=n..m)* ou *seq(f(k),k=obj)* .

Dans le premier cas , la sequence *f(n),f(n+1),...,f(m)* sera construite en incrémentant la variable m, d'une unité jusqu'à atteindre m, m et n peuvent être des rationals.

Dans le second sera construite la séquence *f(op1),f(op2),...,f(opN)* où *op1,op2,...,opN* sont les operandes de l'objet obj, obj étant une entité Maple quelconque.

```
> seq(k,k=1..10);seq(k,k=1/2..5);seq(k^2,k=a+b+c+d);seq(k,k="Hello
  World!");
```

$$1, 2, 3, 4, 5, 6, 7, 8, 9, 10 \quad (22)$$

$$\frac{1}{2}, \frac{3}{2}, \frac{5}{2}, \frac{7}{2}, \frac{9}{2}$$

$$a^2, b^2, c^2, d^2$$

"H", "e", "l", "l", "o", " ", "W", "o", "r", "l", "d", "!"

L'opérateur *$* permet aussi de construire des séquences simples, son utilisation est moins polyvalente que *seq*

```
> x$5;$1..10;
```

$$x, x, x, x, x \quad (23)$$

$$1, 2, 3, 4, 5, 6, 7, 8, 9, 10$$

```
> x^k$k=1..10;
```

$$x, x^2, x^3, x^4, x^5, x^6, x^7, x^8, x^9, x^{10} \quad (24)$$

```
> k$k=a+b+c+d;
Error, wrong number (or type) of parameters in function $
```

Le type d'une séquence :

```
> whattype(S);
```

$$exprseq \quad (25)$$

La séquence vide est désigné par le mot clé *NULL,* utile pour initialiser une séquence qui va être remplie par une boucle ensuite.

On peut récuperer la séquence des opérandes de n'importe quels objet Maple avec la procédure *op* , ce

qui expliquerait le comportement dans l'instruction *seq(f(k),k=obj)*

```
> x:=a+b+c+1/2;y:=(a+b)/c+1/2;z:=(a+b)*c*(1/2);
```

$$x := a + b + c + \frac{1}{2} \tag{26}$$

$$y := \frac{a+b}{c} + \frac{1}{2}$$

$$z := \frac{1}{2}\,(a+b)\,c$$

```
> op(x);op(y);op(z);
```

$$a, b, c, \frac{1}{2} \tag{27}$$

$$\frac{a+b}{c}, \frac{1}{2}$$

$$\frac{1}{2}, a+b, c$$

Listes

Une liste est une séquence Maple mise entre crochets. De ce fait on utilise souvent le constructeur de séquences *seq* pour construire des listes.

```
> L:=[seq(x^k,k=1..10)];
```

$$L := [x, x^2, x^3, x^4, x^5, x^6, x^7, x^8, x^9, x^{10}] \tag{28}$$

```
> L[5];L[-5..-1];
```

$$x^5 \tag{29}$$

$$[x^6, x^7, x^8, x^9, x^{10}]$$

On peut accéder aux opérandes d'une liste en utilisant aussi la procédure *op*. Cette dernière renvoyerait une erreur si on l'utilise avec une séquence.

```
> op(5,L);
```

$$x^5 \tag{30}$$

```
> op(4,S);
Error, wrong number (or type) of parameters in function op
```

op(L) retourne la séquence de toutes les opérandes de L :

```
> op(L);
```

$$x, x^2, x^3, x^4, x^5, x^6, x^7, x^8, x^9, x^{10} \tag{31}$$

Le nombre d'opérandes d'une liste L est donné par *nops(L)*, une fois encore nops provoquera une erreur si elle est utilisée avec une séquence.

```
> nops(L);
```

$$10 \tag{32}$$

```
> nops(S);
Error, wrong number (or type) of parameters in function nops
```

Petite astuce, si on tient à avoir le nombre d'opérande d'une séquence.

```
> nops([S]);
```

$$4 \tag{33}$$

N.B: *?nops(obj)* retourne en fait le nombre d'opérandes de n'importe quel objet Maple (sauf une séquence bien sur).

Type d'une liste :

```
> whattype(L);
```

(34)

$$list \tag{34}$$

Ensembles

Un ensemble au sens Maple est une séquence mise entre accolades {}. La différence entre un ensemble et une liste tient esentiellement en deux points. Dans un ensemble il n'y a pas d'ordre préalable des opérandes (l'ordre d'entrée des opérandes, n'est pas forcément celui de la sortie), l'autre point est que les doublons sont automatiquement éliminé dans un ensemble.

```
> E:={1,(-1)^2,sin,matrix(2,2,1)};
```

$$E := \left\{1, \sin, \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}\right\} \tag{35}$$

```
> op(E);nops(E);
```

$$1, \sin, \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \tag{36}$$

$$3$$

Type d'un ensemble :

```
> whattype(E);
```

$$set \tag{37}$$

Conversion entre types de données

La procédure *convert* permet de convertir un objet Maple quelconque en un autre avec un autre type mais construit à partir des mêmes opérandes.

```
> L;E;
```

$$[x, x^2, x^3, x^4, x^5, x^6, x^7, x^8, x^9, x^{10}] \tag{38}$$

$$\left\{1, \sin, \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}\right\}$$

```
> convert(L,set);convert(E,list);
```

$$\{x, x^2, x^3, x^4, x^5, x^6, x^7, x^8, x^9, x^{10}\} \tag{39}$$

$$\left[1, \sin, \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}\right]$$

```
> convert(a+b+c+2*d,set);
```

$$\{a, b, c, 2\,d\} \tag{40}$$

```
> convert([a,b,c,d],`+`);
```

$$a + b + c + d \tag{41}$$

? Détail interessant , *?convert* permet aussi de faire de la conversion entre systèmes de numération.

```
> convert(17092009,binary);convert(17092009,octal);convert
  (17092009,hex);
```

$$1000001001100110110101001 \tag{42}$$

$$101146651$$

$$104CDA9$$

Ou plus interessant encore, puisque la sortie est presentée sous forme d'une liste, les coéfficients étant pris à partir des unités (de droite vers la gauche) :

```
> convert(17092009,base,2);convert(17092009,base,8);convert
  (17092009,base,16);
```

$$[1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1] \tag{43}$$

$$[1, 5, 6, 6, 4, 1, 1, 0, 1]$$

[9, 10, 13, 12, 4, 0, 1]

Tableaux et tables

Tableaux et tables sont des structures de données composites, capable de stocker des données discrètes sur une ou deux dimensions pour les tableaux, ou avec un système clé/valeur, à la mainère d'un dictionnaire pour les tables.

Tableaux

Un tableau Maple peut être à une ou à deux dimensions. A la différence d'une liste où aucune indication sur la taille n'est fournie lors de la définition, il est nécéssaire de déclarer la taille d'un tableau. Son initialsation peut se faire lors de sa définition ou plus tard (grace à une boucle par exemple).

Pour la construction de tableaux, la syntaxe est assez permissive :
array(1..n, list_coeff): tableau à une dimension avec n coéfficient, list_coeff, non obligatoire, devrait être une liste d'objets Maple, de taille inferieure ou égale à n.
array(list_coeff) : On omet de préciser la taille, mais elle est toujours obligatoire, elle est calculée à partir de la taille de list_coeff.
array(1..n,1..m,list_list_coeff) : tableau à deux dimensions, n lignes et m colonnes. list_list_coeff, devrait être une liste de listes d'objets Maple, chacune representant une ligne du tableau. La taille de chaque liste intérieure doit être inférieure ou égal à m, le nombre de ces listes doit être inférieur ou égal à n.
array(list_list_coeff) : la taille est calculée à partir de list_list_coeff, la taille de la première list donne le nombre de colonnes, le nombre de listes celui des lignes.

```
> T_one:=array(1..3,[1,Pi]);T_two:=array([[7,8],[1],[]]);
```

$$T\_one := \begin{bmatrix} 1 & \pi & T\_one_3 \end{bmatrix} \quad (44)$$

$$T\_two := \begin{bmatrix} 7 & 8 \\ 1 & T\_two_{2,2} \\ T\_two_{3,1} & T\_two_{3,2} \end{bmatrix}$$

On accéde aux coéfficients par leurs indices, y compris pour les initier par afféctation directe.

```
> T_one[1];T_one[3]:=0;
```

$$T\_one_3 := 0 \quad (45)$$

```
> print(T_one);
```

$$\begin{bmatrix} 1 & \pi & 0 \end{bmatrix} \quad (46)$$

```
> T_two[2,2]:=0:T_two[3,1]:=0:T_two[3,2]:=0:
> print(T_two);
```

$$\begin{bmatrix} 7 & 8 \\ 1 & 0 \\ 0 & 0 \end{bmatrix} \quad (47)$$

Une bizarrerie liée au mode d'évaluation des tableaux, qu'on abordera plus en avant.

```
> whattype(T_one);op(T_one);
```

$$symbol \quad (48)$$

$$\begin{bmatrix} 1 & \pi & 0 \end{bmatrix}$$

```
> whattype(op(T_one));op(op(T_one));
```

$$array \qquad (49)$$

$$1..3, [1 = 1, 2 = \pi, 3 = 0]$$

Tables

Une table Maple, est ce qu'on appelle parfois dictionnaire sous d'autres langages de programmation. C'est une structure composite capable de stocker des données en utilisant un systéme d'indexation clé/valeur. A la différence d'un tableau, il n'y a pas d'obligation de préciser la taille d'une table. Une table est de fait de taille indéterminée.

Pour créer une table :

```
> Tab_1:=table([un=sup,deux=spe]);
```

$$Tab\_1 := table([deux = spe, un = sup]) \qquad (50)$$

```
> Tab_2:=table([sup,spe]);
```

$$Tab\_2 := table([1 = sup, 2 = spe]) \qquad (51)$$

```
> Tab_3:=table([(1,1)=1,(1,2)=0,(2,1)=0,(2,2)=1]);
```

$$Tab\_3 := table([(1, 2) = 0, (2, 1) = 0, (2, 2) = 1, (1, 1) = 1]) \qquad (52)$$

Pour créer la table vide :

```
> table();
```

$$table([]) \qquad (53)$$

Le fait d'utiliser un nom T suivi de l'opérateur d'indexation / (quelque chose comme $T[a]$) dans une affectation, crée automatiquement une table de nom T , et initie la clé a avec la valeur qu'on a affecté à $T[a]$. Souvent on utilise l'indexation pour définir des termes d'une suites, on crée en fait une table.

```
> T[1]:=0;
```

$$T_1 := 0 \qquad (54)$$

```
> print(T);
```

$$table([1 = 0]) \qquad (55)$$

Une autre utilisation des tables dans Maple, consiste en la possibilité d'associer à une procédure une table, dite *table remember*, pour stocker les valeurs calculées de cette procédure.

Evaluation dans Maple

Evalution des variables

Les variables simples, les séquences et donc les listes et les ensembles sont immédiatement remplacés par leurs valeurs quand on fait intervenir leurs noms dans une expression. Les tableaux, les tables (et les procédures) ne sont évalués que jusqu'à leurs derniers noms.

Par exemple si on définit les deux variables:

```
> a:=Pi;A:=array(1..2,[1,1]);
```

$$a := \pi \qquad (56)$$

$$A := \begin{bmatrix} 1 & 1 \end{bmatrix}$$

Un appel de ces variables par leurs noms donne :

```
> a;A;
```

$$\pi \qquad (57)$$

$$A$$

a est completement évaluée, alors que la valeur retournée pour le tableau est son derniers nom A.

```
> B:=A;C:=B;
```

$$B := A \qquad (58)$$

$$C := A$$

```
> C;
```

$$A \tag{59}$$

Pour faire référence au contenu de A et non simplement à son nom, il faut utiliser la procédure d'evalution *eval*

```
> eval(A);
```

$$\begin{bmatrix} 1 & 1 \end{bmatrix} \tag{60}$$

Ce comportement s'explique par le fait, que lors de l'évaluation d'une expression qui fait intervenir plusieurs variables, Maple effectue d'abords des simplifications automatiques selon les régles applicables au domaine des variables en jeu (commutativité, associativité, calcul algebrique ...). Ces regles seront donc appliquées uniquement sur les noms des variables composites complexes, exigentes en mémoire, au lieu de le faire sur leurs contenus.

Un autre aspect de l'évaluation des variables complexes sous Maple, est que le nom d'un tableau (ou d'une table) A est un pointeur qui pointe vers son contenu. lorsque on donne un autre nom B à A, les pointeurs A et B continuent de pointer vers le même objet en mémoire. Si on change B, A va aussi changer. Pour une variable simple, lors d'une deuxième affectation une copie du contenu est crée pour le nouveau nom, de sorte que les deux variables deviennent complétement indépendantes.

```
> b:=a;
```

$$b := \pi \tag{61}$$

```
> b;a;
```

$$\pi \tag{62}$$

$$\pi$$

```
> b:=0;a;
```

$$b := 0 \tag{63}$$

$$\pi$$

```
> C;eval(A);
```

$$A \tag{64}$$

$$\begin{bmatrix} 1 & 1 \end{bmatrix}$$

```
> C[1]:=0;eval(A);
```

$$C_1 := 0 \tag{65}$$

$$\begin{bmatrix} 0 & 1 \end{bmatrix}$$

Si on tient à créer une copie d'un objet complexe, il faut le demander explicitement en utilisant *copy*

```
> eval(A);E:=copy(A);E[1]:=1;eval(E);eval(A);
```

$$\begin{bmatrix} 0 & 1 \end{bmatrix} \tag{66}$$

$$E := \begin{bmatrix} 0 & 1 \end{bmatrix}$$

$$E_1 := 1$$

$$\begin{bmatrix} 1 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 1 \end{bmatrix}$$

On peut pour une variable donnée (même simple), retarder son évaluation en mettant son nom entre quotes ' '.

```
> a;'a';
```

$$\pi \tag{67}$$

$$a$$

Ceci permet par exemple de libérer le nom d'une variable qui a déja reçu une valeur par affctation, en lui affectant à nouveau son nom.

```
> X:=1;
```

$$X := 1 \quad (68)$$

```
> X:='X';X;
```

$$X := X$$
$$X \quad (69)$$

Avec plusieurs niveaux de quotations chaque évaluation diminue la protection d'un niveau.

```
> c:=a+b;d:='a'+'b';e:=''a'';f:=e;
```

$$c := \pi$$
$$d := a + b$$
$$e := 'a' \quad (70)$$
$$f := a$$

Structure interne d'une expression Maple :

Une expression Maple est entièrement déterminée par son type et par la séquences de ses opérandes, qui elles même sont des expressions. Une expression est donc construite de façon récursive sous forme d'un arbre, avec des sommets qui sont les types des variables intermédiaires et les terminaisons des branches sont les opérandes élémentaires.

```
> expr:=x+y*z;
```

$$expr := x + y\,z \quad (71)$$

en respectant les regles de précédences des opérateurs, l'expression *expr* est representée par l'arbre :

On peut demander le type d'une expression expr par l'instruction *whattype(expr)* ou *op(0,expr)*. La séquence complète de ses opérandes est données par *op(expr)*. la k eme opérande est désignée par *op(k, expr)*. la h eme opérande de la k eme est désignée par op([k,h],expr), ainsi de suite.

```
> whattype(expr);op(0,expr);
```

$$`+` \quad (72)$$
$$`+`$$

```
> op(expr);op(1,expr);op(1..2,expr);
```

$$x, y\,z \quad (73)$$
$$x$$
$$x, y\,z$$

```
> op([2,1],expr);
```

$$y \quad (74)$$

```
> op(0,op(2,expr));
```

$$`*` \quad (75)$$

On voit ainsi que les séquences representent la structure de donnée la plus élémentaire dans Maple.

Pour un tableau (une table ou une procédure aussi)

```
> Tab:=array(1..2,[0,1]);
```

$$Tab := \begin{bmatrix} 0 & 1 \end{bmatrix} \quad (76)$$

N'oublions pas, un tableau est évalué jusqu'à son dernier nom.

```
> op(0,Tab);
```

$$symbol \quad (77)$$

Il faut plutôt faire :

```
> op(0,eval(Tab));
```

$$array \quad (78)$$

```
> op(eval(Tab));
```

$$1..2, [1 = 0, 2 = 1] \quad (79)$$

Les procédures *map*, *select* et *remove*

select permet de séléctionner, parmi les opérandes d'une expression, celles qui répondent à un certain critère .
select(fct_test, expr), fct_test est une procédure qui doit retourner forcément une valeurs logique (true ou false). Chaque opérande de l'expression est testé par *fct_test*, ne sont retenues que celles auquels la réponse est *true,* Maple construit alors un nouvel objet de même type que *expr* avec les opérandes retenues.

Liste des nombres premiers inférieur à 100 :

```
> select(isprime,[2,seq(2*k+1,k=1..49)]);
```

$$[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97] \quad (80)$$

Pour éléminer d'une liste toutes les opérandes qui sont nulles :

```
> select(x->if x<>0 then true else false fi,[1,0,2,0,9,0,0,0,8,7,6,
  5,0,0,1]);
```

$$[1, 2, 9, 8, 7, 6, 5, 1] \quad (81)$$

remove agit de la même façon mais ne retient que les opérandes qui ne répondent pas au test.

map, bien utile, permet d'appliquer une procédure à toutes les opérandes d'une expression.
map(fct,expr), *fct* étant une procédure quelconque qui pend un seul argument, *fct* est appliquée à chaque opérande de l'expression *expr*, un objet de même type que *expr* est retourné.
map(fct,expr,arg2,arg3,...,argN) applique cette fois la procédure *fct* qui prend cette fois *N* argument, aux opérandes en tant que premier argument, les autres arguments *arg2,..,argN* devant être précisé explicitement .

```
> map(x->x^2,a+b+c+d);
```

$$a^2 + b^2 + c^2 + d^2 \quad (82)$$

```
> map((x,y)->(x+y)^2,a+b+c+d,1);
```

$$(a+1)^2 + (b+1)^2 + (c+1)^2 + (d+1)^2 \quad (83)$$

```
> map(convert,[629,42,6598,27,9,47,820],binary);
```

$$[1001110101, 101010, 1100111000110, 11011, 1001, 101111, 1100110100] \quad (84)$$

Les chaines de charactères

Une chaine de charactère (type *string)* est une suite de charactères qui n'a d'autre signification qu'elle même, elle ne peut faire l'objet d'une affectation et sera toujours évaluée en elle même (d'après l'aide Maple).

les chaines de charactères seront définies comme des suites de charactères quelconques mises entre double-quotes. Elles serviront en général sous Maple à afficher des messages. Elles deviendront en outre indispensable dés qu'il s'agit d'écrire des programmes pour la manipulation de texte, ce qui n'est pas forcément le terrain de prédilection de Maple.

Définition d'une chaine de charactère

```
> Strg:="Une chaine qui se termine avec des charactères spéciaux /,
  $ et &";
```

Strg := "Une chaine qui se termine avec des charact  res spciaux /, $ et &" (85)

```
> whattype(Strg);
```

string (86)

Une chaine de charactère est un type indexé (en fait un tableau dans certains langage), on peut accéder aux charactères d'une chaines par leurs indices

```
> Strg[1];Strg[-1];Strg[1..10];Strg[-9..-1];
```

(87)

"U" **(87)**
"&"
"Une chaine"
"/, $ et &"

On peut chercher un motif dans une chaine de charactères en utilisant *searchtext* (indifférence à la casse) ou *SearchText* (respect de la casse). La valeur retournée est l'indice du premier charactère du motif dans la chaine si le motif est retrouvé dans la chaine, 0 sinon.

```
> searchtext(cde,"abcdefgh");SearchText(abc,"Abcdefgh");
```

3 **(88)**
0

On peut concaténer plusieurs chaine de charactères avec cat

```
> Strg_1:="Une première chaine"; Strg_2:=" et une deuxième";
```

Strg_1 := "Une premire chaine" **(89)**
Strg_2 := " et une deuxime"

```
> cat(Strg_1,Strg_2);
```

"Une premi  re chaine et une deuxime" **(90)**

cat peut servir pour formater une sortie texte basique (voir les fonctions *printf*, *fprintf* et *sprintf* à la manière du langage C, pour un formatage plus élaboré)

```
> for i to 5 do cat("La valeur de i est ",i) od;
```

"La valeur de i est 1" **(91)**
"La valeur de i est 2"
"La valeur de i est 3"
"La valeur de i est 4"
"La valeur de i est 5"

Fonction intéressante, on peut convertir une chaine de charactère en une liste, la liste des codes héxadécimaux (exprimé en décimal !) des charactère de la chaine

```
> L:=convert(Strg,bytes);
```

L := [85, 110, 101, 32, 99, 104, 97, 105, 110, 101, 32, 113, 117, 105, 32, 115, 101, 32, 116, **(92)**
101, 114, 109, 105, 110, 101, 32, 97, 118, 101, 99, 32, 100, 101, 115, 32, 99, 104, 97, 114,
97, 99, 116, 232, 114, 101, 115, 32, 115, 112, 233, 99, 105, 97, 117, 120, 32, 47, 44, 32, 36,
32, 101, 116, 32, 38]

La réciproque, la liste convertie en une chaine, avec la même syntaxe.

```
> convert(L,bytes);
```

"Une chaine qui se termine avec des charact  res spciaux /, $ et &" **(93)**

La liste des charactères de la table de code utilisée par le système

```
> L_char:=convert([$1..255],bytes);
```

L_char := " **(94)**

!"#$%&'()* +,-./0123456789:;<=>?
@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_abcdefghijklmnopqrstuvwxyz{|}
~  "

Stocker tout dans une table indéxée par les codes des charactères

```
> Tab_char:=table(convert(L_char,list));
```

Tab_char := *table*([120 = "x", 241 = "  ", 121 = "y", 1 = "", 242 = "", 122 = "z", 2 = "" , **(95)**
243 = "", 123 = "{", 3 = "" , 244 = "", 124 = "|", 4 = "" , 245 = "", 125 = "}", 5

= "" , 246 = "", 126 = "~", 6 = "" , 247 = "", 127 = "", 7 = "" , 248 = "", 128
= "", 8 = "", 249 = "", 129 = "", 9 = " ", 250 = "", 130 = "", 10
= "
", 251 = "", 131 = "", 11 = " ", 252 = "", 132 = "", 12 = " ", 253 = "", 133 = "",
13 = "
", 254 = "", 134 = "", 14 = "" , 255 = "", 135 = "", 15 = "" , 136 = "", 16 = "" , 137
= "", 17 = "" , 138 = "", 18 = "" , 139 = "", 19 = "" , 140 = "", 20 = "" , 141
= "", 21 = "" , 142 = "", 22 = "" , 143 = "", 23 = "" , 144 = "", 24 = "" , 145
= "", 25 = "" , 146 = "", 26 = "" , 147 = "", 27 = "" , 148 = "", 28 = " ", 149
= "", 29 = "", 150 = "", 30 = " ", 151 = "", 31 = " ", 152 = "", 32 = " ", 153
= "", 33 = "!", 154 = "", 34 = """, 155 = "", 35 = "#", 156 = "", 36 = "$", 157
= "", 37 = "%", 158 = "", 38 = "&", 159 = "", 39 = "", 160 = "", 40 = "(", 161
= "", 41 = ")", 162 = "", 42 = "*", 163 = "", 43 = "+", 164 = "", 44 = ",", 165
= "", 45 = "-", 166 = "", 46 = ".", 167 = "", 47 = "/", 168 = "", 48 = "0", 169
= "", 49 = "1", 170 = "", 50 = "2", 171 = "", 51 = "3", 172 = "", 52 = "4", 173
= "", 53 = "5", 174 = "", 54 = "6", 175 = "", 55 = "7", 176 = "", 56 = "8", 177
= "", 57 = "9", 178 = "", 58 = ":", 179 = "", 59 = ";", 180 = "", 60 = "<", 181
= "", 61 = "=", 182 = "", 62 = ">", 183 = "", 63 = "?", 184 = "", 64 = "@", 185
= "", 65 = "A", 186 = "", 66 = "B", 187 = "", 67 = "C", 188 = "", 68 = "D", 189
= "", 69 = "E", 190 = "", 70 = "F", 191 = "", 71 = "G", 192 = "", 72 = "H", 193
= "", 73 = "I", 194 = "", 74 = "J", 195 = "", 75 = "K", 196 = "", 76 = "L", 197
= "", 77 = "M", 198 = "", 78 = "N", 199 = "", 79 = "O", 200 = "", 80 = "P", 201
= "", 81 = "Q", 202 = "", 82 = "R", 203 = "", 83 = "S", 204 = "", 84 = "T", 205
= "", 85 = "U", 206 = "", 86 = "V", 207 = "", 87 = "W", 208 = "", 88 = "X", 209
= "", 89 = "Y", 210 = "", 90 = "Z", 211 = "", 91 = "[", 212 = "", 92 = "\", 213
= "", 93 = "]", 214 = "", 94 = "^", 215 = "", 95 = "_", 216 = "", 96 = "`", 217
= "", 97 = "a", 218 = "", 98 = "b", 219 = "", 99 = "c", 220 = "", 100 = "d", 221
= "", 101 = "e", 222 = "", 102 = "f", 223 = "", 103 = "g", 224 = "", 104 = "h", 225
= "", 105 = "i", 226 = "", 106 = "j", 227 = "", 107 = "k", 228 = "", 108 = "l", 229
= "", 109 = "m", 230 = "", 110 = "n", 231 = "", 111 = "o", 232 = "", 112 = "p",
233 = "", 113 = "q", 234 = "", 114 = "r", 235 = "", 115 = "s", 236 = "", 116 = "t",
237 = "", 117 = "u", 238 = "", 118 = "v", 239 = "", 119 = "w", 240 = ""])

```
> Tab_char[k]$k=65..90;
```

"A", "B", "C", "D", "E", "F", "G", "H", "I", "J", "K", "L", "M", "N", "O", "P", "Q", "R", "S", "T", "U", "V", "W", "X", "Y", "Z" **(96)**

Signification des quotes

Il y a trois types de quotations, chacune à une signification spéciale: la quote ', la backquote ` et la double quote ".
La quote sert à retarder l'évaluation d'une variable, comme on l'a vu.
La double quote sert à construire des chaines de charactères.
La backquote sert à construire des noms de variables. Elle permet d'utiliser des charactères illégitime dans les noms, tel l'espace

```
> Un nom:=NULL;
Error, missing operator or `;`

> `Un nom`:=NULL;
```

Un nom := **(97)**

Il faut faire la différence entre un nom de variable et une chaine de charactère. Un nom pointe vers une

valeur qui n'est pas le nom lui même, une chaine de charactère n'a d'autre valeur qu'elle même.

TP : Programmation en Maple©

Mamouni My Ismail
Professeur agrégé de mathématiques
Enseignant en classes de MP
CPGE My Youssef
Rabat, Maroc
mamouni.myismail@gmail.com
myismail.chez.com

Dans le cadre des Journées Informatiques des Classes Prépas, du 29 Octobre au 2 Novembre 2009, j'ai la plaisir de vous présente cette initiation à la programmation en Maple©

- ***1ère séance:: Jeudi 29 Octobre 2009, 15h :***
  Intitulé : Initiation sur des exemples simples.
  Contenu :
  - **Cours de programmation en Maple**
  - **Tester si un nombre est premier**
  - **Décomposer un entier en base *b***
  - **Methode de dichotomie**
  - **Calcul approché de π**
  - **Cryptographie RSA**

- ***2ème séance : Vendredi 30 Octobre 2009, 15h :***
  Intitulé : Programmation avancéé.
  Contenu :
  - **Corrigé de l'épreuve d'informatique, Centrale-Sup Elec, 2004.**
  - **Corrigé de l'épreuve d'informatique, X-Polytechnique, 2003.**

2 Cours résumé.

1) Objects manipulés
- Objects élémentaires : numériques et caractères.
- Objects structurés : chaines de caractères, tableaux,..

2) Actions utilisés.
- Affectation
- Appel d'un algorithme prédefini.

3) Instructions de contrôle.

a) La boucle for.
On lutilise pour exécuter des instructions dépendant dun indice entier ou pour exécuter une même instruction n fois.
Syntaxe :
for k from valeur initiale de k to valeur finale de k by le pas
do instructions ;
od ;
Exemple : Algorithme d'Euclide, les restes successives de la division euclidienne.

```
> a:=15487:b:=1254:liste:=NULL:
> from 1 to 5 do
> r:=irem(a,b):
> liste:=liste,r:
> a:=b:
> b:=r:
> od:
> [liste];
```

$$[439, 376, 63, 61, 2]$$

Ce petit programme présente un défaut à chaque fois qu'on veut l'appliquer pour de nouvelles valeurs de $a, b, n = 5$ on doit recharger le programme. Pour y rémedier on va le nommer, lui associer des variables, ici a, b, n, appelons le "restes", le syntaxe en Maple©est le suivant :

```
> restes:=proc(c,d,n)
> a:=c:b:=d:liste:=NULL:
> from 1 to n do
> r:=irem(a,b):
> liste:=liste,r:
> a:=b:
> b:=r:
> od:
> RETURN([liste]);
> end:
```

```
\begin{Maple Warning}{\normalsize{Warning, `a` is implicitly declared local to procedu
```

```
\begin{Maple Warning}{\normalsize{Warning, `b` is implicitly declared local to procedu

\begin{Maple Warning}{\normalsize{Warning, `liste` is implicitly declared local to pro

\begin{Maple Warning}{\normalsize{Warning, `r` is implicitly declared local to procedu

\begin{Maple Warning}{\normalsize{Warning, `a` is implicitly declared local to procedu

\begin{Maple Warning}{\normalsize{Warning, `b` is implicitly declared local to procedu

\begin{Maple Warning}{\normalsize{Warning, `liste` is implicitly declared local to pro
```

Remarque : Maple©déclare automatiquement toutes les variables locales quand on oublie de le faire.
Exemple numérique :

```
> restes(15487,1254,6);
```

$$[439, 376, 63, 61, 2, 1]$$

Remarque : Ce programme présente à son tour un autre défaut, les restes successives ne sont plus possibles dés que l'un des reste est nul.
Exemple numérique :

```
> restes(15487,1254,8);
```

```
\begin{Maple Error}{\normalsize{Error, (in restes) numeric exception: division by zero
```

Ce qui nous amène une autre solution : "test d'arrêt"

b) Le test d'arrét while.
Syntaxe : while le test d'arrêt n'est pas remplie do instructions od :
Exemple : Algorithme d'Euclide.

```
> euclide:=proc(c,d) local a,b,r,Restes:
> a:=c:b:=d:
> r:=irem(a,b):Restes:=r:
> while r>0 do
> a:=b:b:=r:r:=irem(a,b):Restes:=Restes,r:
> od:
> [Restes];
> end:
```

Remarque : Ici on a déclaré nous même nous variables comme étant locales.
Application numérique :

```
> c:=156989:d:=165:euclide(c,d);
```

$$[74, 17, 6, 5, 1, 0]$$

Remarque : Le dernier reste non nul dans l'algorithme d'Euclide n'est autre que le pgcd, chose qu'on peut vérifier rapidement avec le programme prédéfini en Maple©appelé igcd.
Application numérique :

```
> igcd(c,d);
```

c) Les branchements if.
Syntaxe : If condition réalisé then instructions else instructions fi ;
Exemple :Méthode de dichotomie
Objectif :Localiser une racine de l'équation $f(x) = 0$.
Principe : Si $f(a)f(b) < 0$ alors $f(x) = 0$ admet une solution entre $[a, b]$. On pose $c = \frac{a+b}{2}$, si $f(a)f(c) < 0$ alors la solution dans $[a, c]$, sinon elle est dans $[c, b]$.
Application numérique :

```
> f:=x->x^2-2*x+1:
> a:=0:b:=3/2:c:=(a+b)/2:
> if f(a)*f(c)<0 then b:=(a+b)/2
> else a:=(a+b)/2
> fi:
> [a,b];
```

$$[3/4, 3/2]$$

3 Initiation sur des exemples simples.

3.1 Tester de primalité.

Objectif. Ecrire un programme qui verifie si un nombre est premier à l'aide du crible d'Eratosténe, tou d'abord on écrit un programme qui donne tous les nombres premiers inferieurs a un entier donné.

```
> liste:=proc(n)local a, list:
> a:=prevprime(n):
> list:=a:
> while a<>2 do
> a:=prevprime(a):
> list:=a,list:
> od:
> [list]:
> end:
> liste(54);
```

$$[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53]$$

```
> test:=proc(n)local A,i,a:
> A:=liste(trunc(sqrt(n)));
> for i from 1 to nops(A) do
> a:=irem(n,op(i,A)):
> if a<>0 then next
> else break
> fi:
> od:
```

```
> if i<nops(A) then print('n-non-premier'):
> else print('n-premier'):
> fi:
> end:
> test(102);
```

$$102 - non - premier$$

```
> test(101);
```

$$101 - premier$$

3.2 Numération en base b.

Objectif : Écrire un nombre en base b.
Principe : On sait que la décomposition d'un entier naturel en base b quelconque repose sur les restes successifs de sa division euclidienne par b.
Programme :

```
> decomposer:=proc(n,b) local r,q,liste,p:
> r:=irem(n,b):q:=iquo(n,b):liste:=r:
> while q>=b do r:=irem(q,b):p:=iquo(q,b):q:=p:liste:=r,liste:od:[q,liste];end:
> decomposer(159871,10);
```

$$[1, 5, 9, 8, 7, 1]$$

```
> decomposer(7,2);
```

$$[1, 1, 1]$$

3.3 Calcul approché de π avec la méthode des isopêrimétres

. **Principe**.

1) Soit $(a, b) \in (\mathbb{R}^{*+})^2$ tel que $a < b$, on pose :

$$\begin{cases} a_0 = a & , b_0 = b \\ a_{n+1} = \dfrac{a_n + b_n}{2} & , b_{n+1} = \sqrt{a_{n+1} b_n} \end{cases}$$

Montrer que ces suites sont bien définies et adjacentes.

2) Exprimer leurs limites communes en fonction de $\theta = \arccos\left(\dfrac{a}{b}\right)$.
Indication : On pourra d'abord commencer par exprimer les a_n, b_n en fonction de θ.

3) soit $n \in \mathbb{N}$, et P_n le polygône régulier à 2^n côtés et de périmètre 2, soit r_n le rayon du cercle inscrit et R_n celui du cercle circonscrit à P_n, montrer que $\forall n \geq 2$ on a :

$$r_{n+1} = \frac{r_n + R_n}{2}, R_{n+1} = \sqrt{r_{n+1} R_n}$$

4) En déduire qu'elle sont adjacentes puis en remarquant que :
$\dfrac{1}{R_n} \leq \pi \leq \dfrac{1}{r_n}$ en déduire que $\dfrac{1}{R_n}, \dfrac{1}{r_n}$ convergent vers π.

Programme.

```
> n := 0 : R := 1/sqrt(8) : r := 1/4 : N := 10 :
> r1:=1/r:R1:=1/R:
> epsilon:=evalf(10^(-N)):
> erreur:=evalf(abs(r1-R1)):
> while erreur>evalf(epsilon) do
> r:=evalf((r+R)/2);
> R:=evalf(sqrt(R*r));
> r1:=1/r;R1:=1/R;
> erreur:=evalf(abs(r1-R1)):n:=n+1:
> od:
> print(‘la valeur approchée par defaut est‘=evalf(r1));
        ‘la valeur approchée par defaut est‘ = 3.141592655
> print(‘la valeur approchée par exces est‘=evalf(R1));
         ‘la valeur approchée par exces est‘ = 3.141592655
> print(‘le nombre d’operation‘=n);
                    ‘le nombre d’operation‘ = 17
```

3.4 Cryptographie RSA.

Principe. Soit p et q deux nombres premiers, on pose $n = pq$. Soit M un entier naturel premier avec pq, qui représente le message à décoder, et C le message codé envoyé.

1) Dites pourquoi $\varphi(n) = (p-1)(q-1)$.

2) Soit e premier avec $\varphi(n)$, justifier l'existence de $d \in \mathbb{Z}$ tel que $ed \equiv 1 \pmod{\varphi(n)}$.

3) Le message M est codé en C tel que $C \equiv M^e \pmod{n}$. En déduire que : $C^d \equiv M \pmod{n}$.

Indication : On pourra penser à utiliser le théorème d'Euler.
Remarque : Le couple (n, e) est appelé clef publique alors que le couple (n, d) est appelé clef privée. On constate que pour chiffrer un message, il suffit de connaitre e et n. En revanche pour déchiffrer, il faut d et n. Ainsi il suffit de connaitre p, q et e puisque $\varphi(n) = (p-1)(q-1)$ et $d \equiv e^{-1} \quad [\varphi(n)]$. **Programme.** ***On se donne d'abord des nombres premiers, plus qu'ils sont grand plus que c'est mieux, on comprendra dans la suite pourquoi.***

```
> p:=13:q:=17:
```

On factorise le produit (p-1)(q-1) pour en déduire un nombre premier avec.

```
> ifactor((p-1)*(q-1));
```

(0)

```
> e:=5:
```

On cherche les coéfficient de Bezout u,d tels que u(p-1)(q-1)+ed=1.

```
> igcdex((p-1)*(q-1),e,'u','d'):d;
```

On se donne le message à coder.

```
> Message:=162:
```

On code le message.

```
> Code:=Message^e mod p*q;
```

$$Code := 93$$

On décode le message codé.

```
> Decodage:=Code^d mod p*q;
```

$$Decodage := 162$$

Oh ça marche,... mais si on prend le message assez grand ?

```
> Message:=1452:
> Code:=Message^e mod p*q;
```

$$Code := 198$$

```
> Decodage:=Code^d mod p*q;
```

$$Decodage := 126$$

Oh ça ne marche plus, la raison c'est que le message dépasse pq, alors que le décodage non , parceque c'est son reste mod (pq), vérifions.

```
> Message mod p*q;
```

126

Pour y rémedier on découpera le message initial en blocs tous inférieurs à pq. On définit une fonction appellée, hachage qui extrait le 1ér bloc.

```
> hachage:=proc(M,p,q) local n,a,b;
> for n from 1 to nops(M)  do
> a:=sum(M[k]*10^(k-1),k=1..n):b:=a-M[n]*10^(n-1):
> if a>=p*q then break fi:
> od:return([b,n-1]):
> end proc:
```

Puis on définit une fonction appellée, décomposition qui extrait le 2èm bloc à partir du 1ér et ansi de suite jusqu'à extraire tous les blocs..

```
> decomposition:=proc(M,p,q) local M1,n,m,Blocs,M2;
> M1:=M;n:=0;m:=0;Blocs:=NULL;
> while m<nops(M) do
> M2:=op(1,hachage(M1,p,q)):
> Blocs:=M2,Blocs:
> n:=op(2,hachage(M1,p,q)):
> m:=m+n:
> M1:=[seq(M[i],i=m+1..nops(M))]:
> od:
> return(convert(Message,base,10)[1],Blocs);
> end proc:
```

Vérifions sur un exemple.

```
> p:=13;q:=17;Message:=50698465;M:=convert(Message,base,10):
```

$$p := 13$$
$$q := 17$$
$$Message := 50698465$$

```
> Blocs:=decomposition(M,p,q);
```

$$Blocs := 5, 0, 69, 84, 65$$

```
> Code:=seq([Blocs][i]^e mod p*q,i=1..nops([Blocs]));
```

$$Code := 31, 0, 205, 67, 182$$

```
> Decodage:=seq([Code][i]^d mod p*q,i=1..nops([Code]));
```

$$Decodage := 5, 0, 69, 84, 65$$

Ah Enfin ça marche, c'est joli la programmation non, pour terminer je vous laisse un exercice.
Ecrire un programme qui transforme les lettres en chiffres et un autre qui fait l'inverse.
C'est pas difficile, il suffit de s'y mettre.

Fin.

Centrale SupElec, 2004

L'épreuve est constituée de deux parties indépendantes. Le candidat peut les traiter dans l'ordre de son choix à condition de respecter les numérotations.

Partie I - Algorithmique

On appelle *graphe* un ensemble fini de points du plan (nommés nœuds). Certains de ces nœuds sont reliés par un arc orienté. Un graphe permet de représenter simplement une relation binaire définie sur un ensemble fini.

I.A - Affectation de candidats à des postes

Dans cette partie, on s'intéresse au problème de l'affectation de candidats à des postes ouverts par des écoles. Chaque candidat classe les écoles dans lesquelles il souhaite obtenir un poste par ordre de préférence strictement décroissante. Chaque école offre un nombre connu de postes, et classe tous les candidats qui postulent par ordre de préférence strictement décroissante. Les choix des candidats et des écoles peuvent être représentés par un graphe dans lequel chaque nœud représente une candidature : les nœuds du graphe sont sur une grille à deux dimensions, les candidats étant placés en abscisses et les écoles en ordonnées ; ainsi les arcs verticaux représentent la relation de préférence des candidats pour les écoles et les arcs horizontaux la relation de préférence des écoles pour les candidats. Ces relations sont des relations d'ordre : elle sont donc transitives.

I.B - Notations

On note $\langle C_i, E_j \rangle$ la candidature du candidat C_i à un poste ouvert par l'école E_j. On note P_c la relation de préférence des candidats pour les écoles, et P_e la relation de préférence des écoles pour les candidats. Ainsi $P_c(\langle C_i, E_j \rangle, \langle C_i, E_k \rangle)$, indique que le candidat C_i préfère l'école E_j à l'école E_k, et $P_e(\langle C_j, E_i \rangle, \langle C_k, E_i \rangle)$ indique que l'école E_i préfère le candidat C_j au candidat C_k. On note N_i le nombre de postes ouverts par l'école E_i.

Dans toute cette partie $[1, n]$ désigne l'ensemble $\{1, ..., n\}$.

Filière MP

Concours Centrale SupElec 2004

I.C - Exemple

Considérons le graphe ayant pour sommets :

$\langle C_1, E_2\rangle$, $\langle C_1, E_3\rangle$, $\langle C_2, E_1\rangle$, $\langle C_2, E_2\rangle$, $\langle C_2, E_3\rangle$,

$\langle C_3, E_2\rangle$, $\langle C_3, E_3\rangle$, $\langle C_4, E_1\rangle$, $\langle C_4, E_2\rangle$

pour arcs « verticaux » :

$P_c(\langle C_1, E_3\rangle, \langle C_1, E_2\rangle)$,

$P_c(\langle C_2, E_3\rangle, \langle C_2, E_2\rangle)$, $P_c(\langle C_2, E_2\rangle, \langle C_2, E_1\rangle)$,

$P_c(\langle C_3, E_2\rangle, \langle C_3, E_3\rangle)$,

$P_c(\langle C_4, E_1\rangle, \langle C_4, E_2\rangle)$

et pour arcs « horizontaux » :

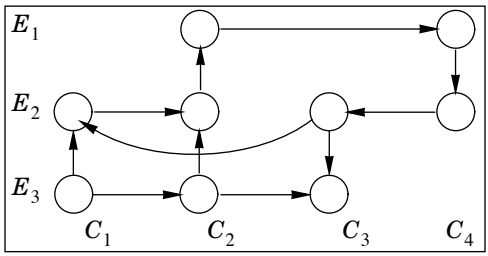
$P_e(\langle C_2, E_1\rangle, \langle C_4, E_1\rangle)$,

$P_e(\langle C_4, E_2\rangle, \langle C_3, E_2\rangle)$, $P_e(\langle C_3, E_2\rangle, \langle C_1, E_2\rangle)$, $P_e(\langle C_1, E_2\rangle, \langle C_2, E_2\rangle)$,

$P_e(\langle C_1, E_3\rangle, \langle C_2, E_3\rangle)$, $P_e(\langle C_2, E_3\rangle, \langle C_3, E_3\rangle)$

avec, comme nombres de postes ouverts, $N_1 = 1$, $N_2 = 2$ et $N_3 = 1$.

Ce graphe peut être représenté comme suit :



Ce graphe indique que le candidat C_1 postule pour les écoles E_2 et E_3, et qu'il préfère E_3 à E_2. De même, le candidat C_2 postule pour les 3 écoles et préfère E_3 à E_2 et E_2 à E_1 et donc, par transitivité, il préfère E_3 à E_1. Le candidat C_3 postule pour E_2 et E_3, dans cet ordre de préférence décroissante, et C_4 postule pour E_1 et E_2 dans cet ordre. L'école E_1 ouvre un seul poste, et elle préfère la candidature de C_2 à celle de C_4. L' école E_2 ouvre 2 postes, elle préfère C_4 à C_3, C_3 à C_1 et C_1 à C_2 ; par transitivité, elle préfère donc C_4 à C_1, C_4 à C_2 et C_3 à C_2. Enfin E_3 n'ouvre qu'un poste et préfère C_1 à C_2 qu'elle préfère à C_3.

I.D - Affectations méritoires

Une affectation $\mathscr{A}$ est un ensemble de nœuds tel que dans chaque colonne, au plus un nœud appartient à l'affectation (un candidat ne peut pas être affecté à plusieurs postes) et tel que sur chaque ligne, le nombre de nœuds appartenant à l'affectation est au plus égal au nombre de postes ouverts par l'école correspondante. Une affectation vérifie donc les propriétés suivantes :

$A1$ $\quad \forall i, \quad \left(\langle C_i, E_j\rangle \in \mathscr{A} \text{ et } \langle C_i, E_k\rangle \in \mathscr{A} \Rightarrow j = k\right)$

$A2$ $\quad \left(\forall j, \exists n > N_j \,;\, \forall k \in [1,n],\, \langle C_{i_k}, E_j\rangle \in \mathscr{A}\right) \Rightarrow \exists p, q \in [1,n] \,,\, \begin{cases} p \neq q \\ i_p = i_q \end{cases}.$

Une affectation est dite « totale » si tous les postes ouverts sont attribués, *ou* si tous les candidats obtiennent un poste (le nombre de postes ouverts et le nombre de candidats ne sont pas forcément égaux). Une affectation $\mathscr{A}$ est dite « méritoire » si et seulement si pour tout nœud $\langle C_i, E_j\rangle$ du graphe l'une des propositions suivantes est vraie :

$M1$ $\quad \langle C_i, E_j\rangle \in \mathscr{A}$

$M2$ $\quad \exists \langle C_i, E_k\rangle \in \mathscr{A}, k \neq j$ et $P_c(\langle C_i, E_k\rangle, \langle C_i, E_j\rangle)$

$M3$ $\quad \exists n_1, \ldots, n_{N_j}$ distincts, $\forall k \in [1, N_j]$, $\begin{cases} n_k \neq i \\ \langle C_{n_k}, E_j\rangle \in \mathscr{A} \\ P_e(\langle C_{n_k}, E_j\rangle, \langle C_i, E_j\rangle) \end{cases}$

l'accolade dans $M3$ signifiant que les 3 propriétés sont vraies simultanément.

I.D.1) Que signifie en langage courant la définition d'une affectation méritoire ?

I.D.2) Une affectation méritoire est-elle nécessairement totale ?

I.E - Nœuds inutiles pour les écoles

Dans cette section on cherche un algorithme conduisant à une affectation méritoire privilégiant les vœux des candidats en donnant à chaque candidat son choix préféré.

On appelle « nœud inutile pour les écoles » tout nœud $\langle C_i, E_j\rangle$ tel qu'il existe N_j nœuds distincts $\langle C_{n_1}, E_j\rangle \ldots \langle C_{n_{N_j}}, E_j\rangle$, avec $n_k \neq i$ pour tout k, qui vérifient les deux propriétés suivantes :

$$\forall k \in [1, N_j],\ P_c(\langle C_{n_k}, E_p\rangle, \langle C_{n_k}, E_j\rangle) \Rightarrow (p = j) \qquad (1)$$

$$\forall k \in [1, N_j],\ P_e(\langle C_{n_k}, E_j\rangle, \langle C_i, E_j\rangle) \qquad (2)$$

I.E.1) Montrer que les affectations méritoires d'un graphe sont exactement celles du graphe obtenu en supprimant les nœuds inutiles pour les écoles du graphe initial, à condition que, lors de la suppression des nœuds inutiles, on prenne garde de maintenir les chaînes des préférences concernant les noeuds restants.

I.E.2) Déduire de la question précédente un algorithme pour trouver une affectation méritoire.

I.E.3) Appliquer cet algorithme (pas à pas) au graphe donné en exemple.

On va maintenant s'intéresser à l'affectation qui privilégie les vœux des écoles.

I.F - Dualité candidat-école

I.F.1) Donner la définition d'un « nœud inutile pour les candidats ».

I.F.2) Montrer que les nœuds inutiles pour les candidats peuvent eux-aussi être supprimés du graphe sans que cela change les affectations méritoires.

I.F.3) En déduire un algorithme pour obtenir l'affectation méritoire qui privilégie le choix des écoles.

I.F.4) Appliquer cet algorithme au graphe donné en exemple.

I.G - Graphe réduit

I.G.1) Donner un algorithme permettant de supprimer tous les nœuds inutiles (aussi bien pour les écoles que pour les candidats) d'un graphe.

I.G.2) Appliquer cet algorithme au graphe donné en exemple, et en déduire toutes les affectations méritoires de ce graphe.

Mamouni My Ismail
Professeur agrégé de mathématiques
Enseignant en classes de MP
CPGE My Youssef
Rabat, Maroc
mamouni.myismail@gmail.com
myismail.chez.com

Corrigé abrégé
Epreuve Informatique
Concours Centrale Sup Elec, 2004

```
STUDENT > restart:
```

On commence par déclarer les écoles, le nombre de postes ouverts pour chaque école

```
STUDENT > Ecoles:=[E1,E2,E3];N:=[1,2,1];candidats:=[C1,C2,C3,C4];
```

$$Ecoles := [E1, E2, E3]$$

$$N := [1, 2, 1]$$

$$candidats := [C1, C2, C3, C4]$$

Puis le classement de chaque école pour les candidats

```
STUDENT > E1:=[C2,C4];E2:=[C4,C3,C1,C2];E3:=[C1,C2,C3];
```

$$E1 := [C2, C4]$$

$$E2 := [C4, C3, C1, C2]$$

$$E3 := [C1, C2, C3]$$

Les voeux de chaque candidat classés dans l'ordre de préference, la 1ére case répresente un numéro pour idéntifier le candidat

```
STUDENT > C1:=[1,3,2];C2:=[2,3,2,1];C3:=[3,2,3];C4:=[4,1,2];
```

$$C1 := [1, 3, 2]$$

$$C2 := [2, 3, 2, 1]$$

$$C3 := [3, 2, 3]$$

$$C4 := [4, 1, 2]$$

Pour chaque école, on va chercher les candidats à éliminer, en commençant bien sûr par le dernier classée pour cette même école. Pour cela on dénombre les candidats les mieux classés pour lesquels cette école représente le 1ér choix, si ce nombre dépasse le nombre de poste ouverts par la dite école, alors notre candidat est élinminé.

```
STUDENT > choix_ecole:=proc(i) local
            Ecole,j,Nbr,k,candidat,a,b,NewEcole;
STUDENT > Ecole:=op(i,Ecoles):NewEcole:=NULL:
STUDENT > for j from nops(Ecole) to 1 by -1 do
STUDENT > Nbr:=0;
STUDENT > for k from j-1 to 1 by -1 do
STUDENT > candidat:=op(k,Ecole):
STUDENT > if op(2,candidat)=i then Nbr:=Nbr+1:
STUDENT > else
```

```
STUDENT > fi:
STUDENT > od:
STUDENT > if Nbr>=op(i,N) then print(`Le candidat eliminé est
          numéro`=op(1,op(j,Ecole)));
STUDENT > else print(`Le candidat non eliminé
          `=op(1,op(j,Ecole)));NewEcole:=NewEcole,C[op(1,op(j,Ecol
          e))]:
STUDENT > fi:
STUDENT > od:
STUDENT > print(`Le choix de l'école est
          `=[NewEcole]);RETURN([NewEcole]);
STUDENT > end:
```

On applique notre petit programme pour le 1ére école

```
STUDENT > E[1]:=choix_ecole(1);
```

$$\text{Le candidat non eliminé} = 4$$

$$\text{Le candidat non eliminé} = 2$$

$$\text{Le choix de l'école est} = [C_4, C_2]$$

$$E_1 := [C_4, C_2]$$

On applique notre petit programme pour le 2éme école

```
STUDENT > E[2]:=choix_ecole(2);
```

$$\text{Le candidat non eliminé} = 2$$

$$\text{Le candidat non eliminé} = 1$$

$$\text{Le candidat non eliminé} = 3$$

$$\text{Le candidat non eliminé} = 4$$

$$\text{Le choix de l'école est} = [C_2, C_1, C_3, C_4]$$

$$E_2 := [C_2, C_1, C_3, C_4]$$

On applique notre petit programme pour le 3éme école

```
STUDENT > E[3]:=choix_ecole(3);
```

$$\text{Le candidat eliminé est numéro} = 3$$

$$\text{Le candidat eliminé est numéro} = 2$$

$$\text{Le candidat non eliminé} = 1$$

$$\text{Le choix de l'école est} = [C_1]$$

$$E_3 := [C_1]$$

Maintenant chaque candidat va éliminer toute école qu'elle juge inutile, c'est à dire pour la quelle il est sûr d'avoir mieux. Comment? il commence par la derniére et il vérifie si une école mieux classée va le prendre dans sa liste prinicipale.

```
STUDENT > for i from 1 to nops(Ecoles) do
STUDENT > liste_principale[i]:=[seq(op(j,E[i]),j=1..N[i])];
STUDENT > od;
```

$$\text{liste\_principale}_1 := [C_4]$$

$$liste\_principale_2 := [C_2, C_1]$$

$$liste\_principale_3 := [C_1]$$

STUDENT >

On s'inspirant de cet exemple on inverse les role écoles-candidats pour trouver les écoles à éliminer pour chaque candidat....A vous de le faire

ÉCOLE POLYTECHNIQUE

ÉCOLE SUPÉRIEURE DE PHYSIQUE ET CHIMIE INDUSTRIELLES

CONCOURS 2003 — FILIÈRE MP - OPTION SCIENCES INDUSTRIELLES

FILIÈRE PC

ÉPREUVE FACULTATIVE D'INFORMATIQUE

(Durée : 2 heures)

L'utilisation des calculatrices **n'est pas autorisée** pour cette épreuve.
Le langage de programmation choisi par le candidat doit être spécifié en tête de la copie.

***

Avertissement On attachera une grande importance à la clarté, à la précision, à la concision de la rédaction.

L'enclos du robot

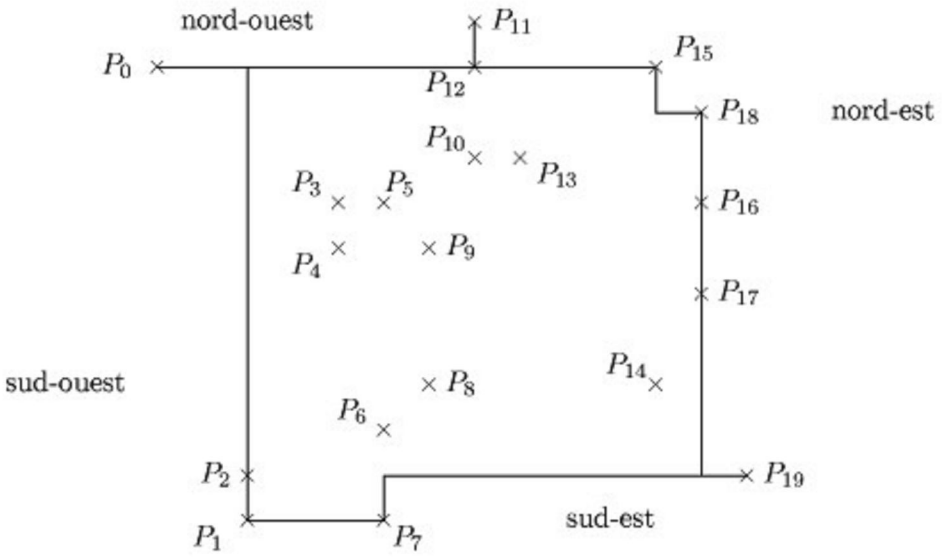
Frontière Sud-Ouest

Un robot rend visite à n points P_i $(0 \leq i < n)$ de coordonnées (a_i, b_i) $(a_i \in \mathbf{N}, b_i \in \mathbf{N}, n \geq 0)$. Le robot ne fait que des déplacements parallèles à l'axe des abscisses ou à l'axe des ordonnées. Ses déplacements sont toujours de longueur minimale entre deux points. Toutefois le robot n'est pas très fiable. On veut délimiter l'espace minimal nécessaire pour ses déplacements en tendant une ficelle autour du périmètre strictement nécessaire pour les déplacements du robot.

L'*intérieur Manhattan* des n points est l'ensemble des points de coordonnées (x, y) vérifiant les quatre conditions suivantes :

$\exists i.\ 0 \leq i < n$ et $a_i \leq x$ et $b_i \leq y$ (sud-ouest)
$\exists i.\ 0 \leq i < n$ et $a_i \geq x$ et $b_i \leq y$ (sud-est)
$\exists i.\ 0 \leq i < n$ et $a_i \geq x$ et $b_i \geq y$ (nord-est)
$\exists i.\ 0 \leq i < n$ et $a_i \leq x$ et $b_i \geq y$ (nord-ouest)

L'enveloppe Manhattan est un polygone dont l'intérieur est l'intérieur Manhattan de ces n points. Par exemple sur les 20 points de la figure suivante, c'est le polygone suivant :



On se propose de calculer les points définissant l'enveloppe Manhattan dans un premier temps, puis de tracer cette enveloppe dans un deuxième temps.

Question 1 Écrire la fonction *sudouest* qui retourne le résultat vrai si et seulement si le point de coordonnées (x_1, y_1) est au sudouest du point de coordonnée (x_2, y_2), c'est-à-dire si $x_1 \leq x_2$ et $y_1 \leq y_2$. Écrire de même les fonctions *nordouest*, *sudest*, *nordest*. (Dans les langages de programmation où les valeurs booléennes n'existent pas, on rendra l'entier 0 pour la valeur *faux* et 1 pour la valeur *vrai*)

Nous décomposons le calcul de l'enveloppe en quatre fonctions : la première calcule les points définissant la partie sud-ouest de l'enveloppe, la deuxième calcule les points définissant la partie nord-ouest, la troisième et quatrième font de même sur les parties sud-est et nord-est.

On suppose les coordonnées des n points P_i rangés dans deux tableaux a et b d'entiers contenant l'abscisse a_i et l'ordonnée b_i du point P_i pour tout i $(0 \leq i < n)$. En outre, on suppose les points rangés par ordre d'abscisses croissantes, c'est-à-dire $a_i \leq a_j$ pour $0 \leq i < j < n$.

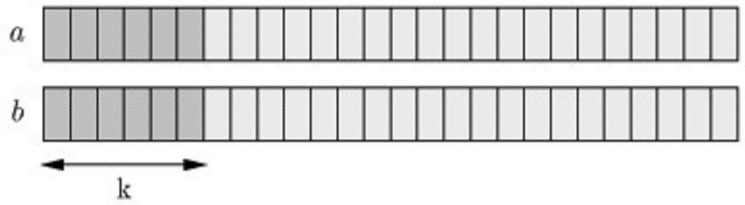
Question 2 Écrire une fonction *echange* prenant comme arguments les tableaux a et b, les indices i et j et qui échange, dans chacun des tableaux a et b, les valeurs contenues aux indices i et j.

Les points définissant la partie sud-ouest de l'enveloppe sont les points P_i tels que : $a_j \leq a_i$ et $b_j \leq b_i$ implique $a_j = a_i$ et $b_j = b_i$ pour tout j $(0 \leq j < n)$.

Question 3 Dans l'exemple précédent, donner parmi les 20 points P_i, les points définissant la partie sud-ouest de l'enveloppe.

Question 4 Écrire une fonction *frontiereSO* prenant en argument les tableaux a et b et retournant le nombre k de points définissant la partie sud-ouest de l'enveloppe Manhattan des n

points de coordonnées a_i et b_i. On modifiera les tableaux a et b pour qu'ils contiennent dans leur k premières places les coordonnées des points définissant la partie sud-ouest de l'enveloppe, rangées en ordre d'abscisses croissantes.



Après avoir exécuté la fonction précédente, on suppose qu'une fonction range dans deux tableaux *aSO*, *bSO* les coordonnées des points précédemment trouvés, qui définissent la partie sud-ouest. Une variable globale *nSO* a pour valeur le nombre de ces points.

Tracé de l'enveloppe

Les points définissant la partie nord-ouest de l'enveloppe sont les points P_i tels que $a_j \leq a_i$ et $b_j \geq b_i$ implique $a_j = a_i$ et $b_j = b_i$ pour tout j $(0 \leq j < n)$.

Question 5 Donner les points définissant la partie nord-ouest de l'enveloppe sur l'exemple. Modifier la fonction précédente pour obtenir la fonction *frontiereNO* correspondante pour la partie nord-ouest de l'enveloppe.

Question 6 Écrire également les fonctions *frontiereSE* et *frontiereNE* correspondant aux parties sud-est et nord-est.

On suppose à présent que les coordonnées des points précédemment trouvés, qui définissent les partie sud-ouest, nord-ouest, sud-est et nord-est, sont rangées respectivement dans des tableaux *aSO*, *bSO*, *aNO*, *bNO*, *aSE*, *bSE*, *aNE*, *bNE*, et toujours classées par ordre d'abscisses croissantes. Soient *nSO*, *nNO*, *nSE*, *nNE* les nombres de ces points. On suppose également qu'il existe deux fonctions graphiques *moveTo* et *lineTo* telles que :

- *moveTo(x,y)* déplace le point courant au point (x, y),
- *lineTo(x,y)* trace un segment du point courant jusqu'au point (x, y). Après le tracé, le point courant devient le point de coordonnées (x, y).

Question 7 Écrire une fonction qui dessine l'enveloppe Manhattan. (Cette fonction utilise les 12 variables globales *aSO*, *bSO*, *aNO*, *bNO*, *aSE*, *bSE*, *aNE*, *bNE*, *nSO*, *nNO*, *nSE*, *nNE*.)

Question 8 L'enveloppe peut-elle produire un polygone croisé ? Si oui, donner une piste pour résoudre ou éviter ce problème. $O(1)$ opérations.

* *
*

Mamouni My Ismail
Professeur agrégé de mathématiques
Enseignant en classes de MP
CPGE My Youssef
Rabat, Maroc
mamouni.myismail@gmail.com
myismail.chez.com

ECOLE POLYTECHNIQUE
ECOLE SUPERIEURE DE PHYSIQUE ET CHIMIE INDUSTRIELLES

CONCOURS 2003 **FILIERE MP-OPTION SCIENCES INDUSTRIELLES**

FILIERE PC

EPREUVE FACULTATIVE D'INFORMATIQUE

***

l'enclos du robot

Frontière Sud-Ouest

```
STUDENT > restart;
```

Question 1.

```
STUDENT > sudouest:=proc(P,Q) local x,y;
STUDENT > x:=[P[1],Q[1]]:y:=[P[2],Q[2]]:
STUDENT > if x[1]<=x[2] and y[1]<=y[2] then 1
STUDENT > else 0
STUDENT > fi;
STUDENT > end:
STUDENT > nordouest:=proc(P,Q) local x,y;
STUDENT > x:=[P[1],Q[1]]:y:=[P[2],Q[2]]:
STUDENT > if x[1]<=x[2] and y[1]>=y[2] then 1
STUDENT > else 0
STUDENT > fi;
STUDENT > end:
STUDENT > sudest:=proc(P,Q) local x,y;
STUDENT > x:=[P[1],Q[1]]:y:=[P[2],Q[2]]:
STUDENT > if x[1]>=x[2] and y[1]<=y[2] then 1
STUDENT > else 0
STUDENT > fi;
STUDENT > end:
STUDENT > nordest:=proc(P,Q) local x,y;
STUDENT > x:=[P[1],Q[1]]:y:=[P[2],Q[2]]:
STUDENT > if x[1]>=x[2] and y[1]>=y[2] then 1
STUDENT > else 0
STUDENT > fi;
STUDENT > end:
```

Question 2

```
STUDENT > echange:=proc(a,b,i,j) local p,q,a1,b1;
```

```
STUDENT > p:=min(i,j);q:=max(i,j);
STUDENT > if p=1 then
STUDENT >      if q=nops(a) then
          a1:=[a[q],seq(a[k],k=2..nops(a)-1),a[1]];
STUDENT >
          b1:=[b[q],seq(b[k],k=2..nops(b)-1),b[1]];
STUDENT >      else
          a1:=[a[q],seq(a[k],k=2..q-1),a[1],seq(a[k],k=q+1..nops(a
          ))];
STUDENT >
          b1:=[b[q],seq(b[k],k=2..q-1),b[1],seq(b[k],k=q+1..nops(b
          ))];
STUDENT >      fi;
STUDENT > else
STUDENT >      if q=nops(a) then
          a1:=[seq(a[k],k=1..p-1),a[q],seq(a[k],k=p+1..q-1),a[p]];
          b1:=[seq(b[k],i=1..p-1),b[q],seq(b[k],k=p+1..q-1),b[p]];
STUDENT >      else
STUDENT > a1:=[seq(a[k],k=1..p-1),a[q],seq(a[k],k=p+1..q-1),a[p],s
          eq(a[k],k=q+1..nops(a))];
          b1:=[seq(b[k],k=1..p-1),b[q],seq(b[k],k=p+1..q-1),b[p],s
          eq(b[k],k=q+1..nops(b))];     fi;
STUDENT > fi;
STUDENT > RETURN([a1,b1]);
STUDENT > end:
Warning, `k` in call to `seq` is not local
Warning, `k` in call to `seq` is not local
Warning, `k` in call to `seq` is not local
Warning, `k` in call to `seq` is not local
Warning, `k` in call to `seq` is not local
Warning, `k` in call to `seq` is not local
Warning, `k` in call to `seq` is not local
Warning, `k` in call to `seq` is not local
Warning, `i` in call to `seq` is not local
Warning, `k` in call to `seq` is not local
Warning, `k` in call to `seq` is not local
Warning, `k` in call to `seq` is not local
Warning, `k` in call to `seq` is not local
Warning, `k` in call to `seq` is not local
Warning, `k` in call to `seq` is not local
Warning, `k` in call to `seq` is not local
```

Question 3 *c'est le point P1*

Question 4

Ecrivons d'abord la fonction testSO qui retourne 0 si un point donné est dans la frontière SudOuest, et une valeur non nulle dans le cas contraire

```
STUDENT > testSO:=proc(a,b,i) local a1,b1,N,j;
STUDENT > a1:=op(1,echange(a,b,1,i));
STUDENT > b1:=op(2,echange(a,b,1,i));
STUDENT > N:=0:for j from 2 to nops(a) do
          N:=N+sudouest([a1[j],b1[j]],[a1[1],b1[1]]);
STUDENT > od;
```

```
STUDENT > RETURN(N);
STUDENT > end:
```

Terminons enfin par récupérer les points qui se trouvent sur la frontière SudOuest, c'est à dire pour lesquels la valeur retournée par testSO est 0

```
STUDENT > frontiereSO:=proc(a,b) local aSO,bSO,i; global nSO;
STUDENT > aSO:=NULL:
STUDENT > bSO:=NULL:
STUDENT > for i from 1  to nops(a) do
STUDENT > if testSO(a,b,i)=0 then aSO:=aSO,a[i];bSO:=bSO,b[i];
STUDENT > else fi;
STUDENT > od;
STUDENT > nSO:=nops([aSO]);
STUDENT > RETURN(seq([op(i,aSO),op(i,bSO)],i=1..nSO));
STUDENT > end:
```

Question 5

```
STUDENT > testNO:=proc(a,b,i) local a1,b1,N,j;
STUDENT > a1:=op(1,echange(a,b,1,i));
STUDENT > b1:=op(2,echange(a,b,1,i));
STUDENT > N:=0:for j from 2 to nops(a) do
          N:=N+nordouest([a1[j],b1[j]],[a1[1],b1[1]]);
STUDENT > od;
STUDENT > RETURN(N);
STUDENT > end:
STUDENT > frontiereNO:=proc(a,b) local aNO,bNO,i; global nNO;
STUDENT > aNO:=NULL:
STUDENT > bNO:=NULL:
STUDENT > for i from 1  to nops(a) do
STUDENT > if testNO(a,b,i)=0 then aNO:=aNO,a[i];bNO:=bNO,b[i];
STUDENT > else fi;
STUDENT > od;
STUDENT > nNO:=nops([aNO]);
STUDENT > RETURN(seq([op(i,aNO),op(i,bNO)],i=1..nNO));
STUDENT > end:
```

Question 6

```
STUDENT > testSE:=proc(a,b,i) local a1,b1,N,j;
STUDENT > a1:=op(1,echange(a,b,1,i));
STUDENT > b1:=op(2,echange(a,b,1,i));
STUDENT > N:=0:for j from 2 to nops(a) do
          N:=N+sudest([a1[j],b1[j]],[a1[1],b1[1]]);
STUDENT > od;
STUDENT > RETURN(N);
STUDENT > end:
STUDENT > frontiereSE:=proc(a,b) local aSE,bSE,i; global nSE;
STUDENT > aSE:=NULL:
STUDENT > bSE:=NULL:
STUDENT > for i from 1  to nops(a) do
STUDENT > if testSE(a,b,i)=0 then aSE:=aSE,a[i];bSE:=bSE,b[i];
```

```
STUDENT > else fi;
STUDENT > od;
STUDENT > nSE:=nops([aSE]);
STUDENT > RETURN(seq([aSE[i],bSE[i]],i=1..nSE));
STUDENT > end:
STUDENT > testNE:=proc(a,b,i) local a1,b1,N,j;
STUDENT > a1:=op(1,echange(a,b,1,i));
STUDENT > b1:=op(2,echange(a,b,1,i));
STUDENT > N:=0:for j from 2 to nops(a) do
          N:=N+nordest([a1[j],b1[j]],[a1[1],b1[1]]);
STUDENT > od;
STUDENT > RETURN(N);
STUDENT > end:
STUDENT > frontiereNE:=proc(a,b) local aNE,bNE,i; global nNE;
STUDENT > aNE:=NULL:
STUDENT > bNE:=NULL:
STUDENT > for i from 1  to nops(a) do
STUDENT > if testNE(a,b,i)=0 then aNE:=aNE,a[i];bNE:=bNE,b[i];
STUDENT > else fi;
STUDENT > od;
STUDENT > nNE:=nops([aNE]);
STUDENT > RETURN(seq([aNE[i],bNE[i]],i=1..nNE));
STUDENT > end:
STUDENT > frontiereSO(a,b);
```

$$[1, 1]$$

```
STUDENT > with(plots):
```

Question 7

On définit maintenant la fonction SO qui permet de tracer la frontière SudOuest en joignant ses point à l'aide de la fonction plot

```
STUDENT > tracer:=proc(P) local Pts,i;
STUDENT > Pts:=NULL:
STUDENT > for i from 1 to nops(P)-1 do
STUDENT > Pts:=Pts,P[i],[P[i][1],P[i+1][2]];
STUDENT > od;
STUDENT > Pts:=Pts,P[nops(P)];
STUDENT > end:
STUDENT > P:=[[0,11],[2,0],[2,1],[3,8],[3,7],[4,8],[4,2],[4,0],[5,
          3],[5,6],[6,9],[6,12],[6,11],[7,9],[9,3],[9,11],[10,8],[
          10,6],[10,10],[11,1]];a:=[seq(P[i][1],i=1..nops(P))];b:=
          [seq(P[i][2],i=1..nops(P))];
```

$$P := [[0, 11], [2, 0], [2, 1], [3, 8], [3, 7], [4, 8], [4, 2], [4, 0], [5, 3], [5, 6], [6, 9], [6, 12], [6, 11], [7, 9], [9, 3], [9, 11], [10, 8], [10, 6], [10, 10], [11, 1]]$$

$$a := [0, 2, 2, 3, 3, 4, 4, 4, 5, 5, 6, 6, 6, 7, 9, 9, 10, 10, 10, 11]$$

$$b := [11, 0, 1, 8, 7, 8, 2, 0, 3, 6, 9, 12, 11, 9, 3, 11, 8, 6, 10, 1]$$

```
STUDENT > P1:=[frontiereNO(a,b),frontiereNE(a,b),frontiereSE(a,b),
          frontiereSO(a,b),frontiereNO(a,b)];
```

$$P1 := [[6, 12], [6, 12], [9, 11], [10, 10], [11, 1], [4, 0], [11, 1], [2, 0], [6, 12]]$$

```
STUDENT > plot([tracer(P1)]);
```