Dans matlab, on peut réaliser soit des scripts soit des fonctions:

- **Les scripts** sont des programmes sans argument dont le contenu s'exécute comme un copier-coller dans le command window.
- **Les fonctions** sont des programmes qui reçoivent des arguments en entrée et qui renvoie un ou plusieurs résultats en sortie. Pour réaliser une fonction, il faut que la première ligne du fichier soit :

```
function [output1,output2, ...]=NomDeFonction(Input1,Input2,...)
```

Le fichier contenant la fonction doit avoir le même nom que celle-ci. Il est possible d'insérer plusieurs fonctions dans un fichier mais celles-ci ne seront accessibles qu'à l'intérieur même du fichier.

Les commentaires insérer dans les premières lignes (Signalés avec un "%") seront afficher lorsque vous taperez "**help NomDeFichier**".

I. Instructions de contrôle

- **Boucle for** (parcours d'un intervalle)

```
for  indice = borne_inf : incrément: borne_sup
  Séquence d'instructions ;
end
```

Exemples :

```
for x=0:0.5:2*pi
      y =cos(x);
end

for s=[1,2,5,7]
      disp(s)
end
```

- **Boucle while** (tant que…faire)

```
while expression_logique
  Séquence d'instructions ;
end
```

Exemple :

```
n=10;
while n~=0
  n=n-1;
end
```

- **Instruction conditionnée if**

```
if expression logique
   séquence d'instructions 1
elseif  expression logique 2
   séquence d'instructions 2
…
else
   séquence d'instructions
```

```
end
```

Exemple :

```
if (x<2.5)
   y=x^4
elseif ((x>=2.5)&(x<9.5))
   y=5*x^3
else
   y=x^3+1
end
```

- **Instruction switch** (choix ventilé)

```
switch var
case cst1,
  séquence d'instructions 1
case cst2,
  séquence d'instructions 2
…
case cstN,
  séquence d'instructions N
otherwise
  séquence d'instructions
end
```

Exemple :

```
Val=input('entrer une valeur')
switch val
  case 1
   disp('un')
  case 2
   disp('deux')
  case 3
   disp('trois')
  otherwise
   disp('autre valeur')
end
```

II. Différentes façons de définir une fonction

1. Fonctions définies dans un fichier function

Exemple :

Créer un fichier f99.m définissant la fonction :

`f(x)=1/((x-0.3)^2+0.01)+1/((x-0.9)^2+0.04)-6`

Représenter graphiquement cette fonction

Ecrire : `f=@f99` puis `f(2.0)`

2. Fonctions anonyme

Elles sont définies par le symbole **@**

Exemple :

Ecrire `fh=@(x) 1/((x-0.3)^2+0.01)+1/((x-0.9)^2+0.04)-6`

Pour calculer la valeur de la fonction en un point :

Ecrire : `fh(2.0);`

On peut également créer une fonction anonyme à plusieurs variables :

```
fh=@(x,y) y*sin(x)+x*cos(y)
fh(pi,2*pi)
```

III. Représentation graphique de fonctions

La commande fplot représente une fonction sur un intervalle donné :

```
fplot(@f99,[-5 5])
grid on
```

On peut faire un zoom selon un axe :

```
fplot(@f99,[-5 5 -10 25])
grid on
```

On peut également représenter graphiquement une fonction anonyme

```
fplot(@(x) 2*sin(x+3), [-1 1]);
```

On peut représenter plusieurs courbes sur le même graphe

```
fplot(@(x) [2*sin(x+3), f99(x)],[-5 5]);
```

Option intéressante : La fonction `fh=@(x) [2*sin(x+3), f99(x)]` permet d'obtenir une matrice deux colonnes contenant les deux fonctions définies

Exemple :

Ecrire `fh([1;2;3])`

Commenter le résultat obtenu

IV. Recherche des minima d'une fonction

Écrire et commenter

```
[xmin ymin]=fminbnd(@f99,0.3,1)
fplot(@f99,[-5 5])
hold;
plot(xmin,ymin,'square ','MarkerEdgeColor', 'r','MarkerFacecolor','r')
```

Rechercher le minimum pour une fonction de plusieurs variables

Ecrire le fichier function suivant

```
function b = trois_var(v)
x = v(1);
y = v(2);
z = v(3);

b = x.^2 + 2.5*sin(y) - z^2*x^2*y^2;
```

Rechercher le minimum de cette fonction en utilisant comme valeur de démarrage `x=-0.6`, `y=-1.2 et z=0.135`

```
v = [-0.6 -1.2 0.135];
a = fminsearch(@trois_var,v)
```

V. Recherche des zéros d'une fonction

Exemple 1 :

Calculer le zéro de sin(x) au voisinage de 3.

```
x = fzero(@sin,3)
```

Exemple 2 :

Trouver le zéro de cos(x) entre 1 et 2

```
x = fzero(@cos,[1 2])
```

Exemple 3 :

Trouver le zéro de la fonction `f(x)=x^3-2*x-5`;

Ecrire une fonction anonyme

```
f = @(x)x.^3-2*x-5;
```

puis trouver le zéro de cette fonction au voisinage de 2:

```
z = fzero(f,2)
```

Comme cette fonction est un polynôme, que donnerait la commande roots ?

VI. Dérivation

Y = diff(X) calcule les différences entre les éléments adjacents de X.

- Si X est un vecteur de longueur N, alors diff(X) renvoie un vecteur de longueur N-1 défini par : [X(2)-X(1) X(3)-X(2) ... X(n)-X(n-1)]
- Si X est une matrice alors diff(X) renvoie une matrice constituée par les différences entre les lignes, définie par [X(2:m,:)-X(1:m-1,:)]

Y = diff(X,n) applique diff(X) de manière récursive n fois.

diff(X,2) est équivalent à diff(diff(X)).

Remarque : la quantité diff(y)./diff(x) est une expression approchée de la dérivée dy/dx.

Exemple 1 :

```
x = [1 2 3 4 5];
y = diff(x)
z = diff(x,2)
```

Exemple 2 :

Définir y=sin(x) pour x compris entre 0 et 4pi.

Calculer dy/dx et représenter le graphe.

Noter et commenter les messages d'erreurs éventuels.

VII. Intégration

Commande `quadl`

syntaxe `q=quadl(@mafonction,a,b)`

`mafonction` : fichier fonction définissant la fonction ou bien fonction anonyme

`a, b` : bornes d'intégration

Exercice 1 : Intégrer la fonction `f99` entre `0` et `2`

Exercice 2 : Proposer deux méthodes pour intégrer la fonction

```
y = 1./(x.^3-2*x-5);
```

Exercice 3 : Calculer et représenter l'intégrale suivante sur l'intervalle `0<x<pi/2` de la fonction :

```
cos(x)/sqrt(1+sin(x));
```

MATLAB possède deux types de fonctions, fonctions numériques et fonctions à expressions symboliques.

Une méthode simple, pour construire une fonction numérique, utilise la fonction MATLAB `inline`.

Exemple :

```
>> f = inline('x^3 + x - 1')
```

Les fonctions de ce type acceptent, comme variables d'entrée, des vecteurs ou des matrices.

Pour évaluer la fonction $f(x)=x^3+x-1$ au point $x=2$, on écrit `f(2)`.

```
>> A = [1 2 3; 4 5 6]
>> B = f(A)
>> g = inline('cos(x) + x')
```

Utiliser et commenter la commande :

```
>> h = f + g
>> h = inline('x.^3 + 2*x - 1 + cos(x)')
```

Representation graphique

```
>> x = linspace(0, 5, 101);
>> plot(x, f(x))
>> hold on
>> plot(x,g(x))
```

Calcul symbolique

Les variables peuvent être déclarées comme étant des variables symboliques.

Exemple :

```
>> syms x y z a b c
```

On peut par la suite définir des expressions symboliques en utilisant ces variables.

```
>> A = [a b 1; 0 1 c; x 0 0]
>> d = det(A)
```

Une fonction $f(x)$ peut être définie en termes d'une expression symbolique.

```
>> f = a*x^2 + b*x + c + 2*cos(x)
>> diff(f)
>> diff(f,2)
```

On peut différentier f par rapport à la variable a par la commande :

```
>> diff(f,a)
```

Représentation graphique

```
>> ezplot(f, [1,5])
```

1. Evaluation d'une expression symbolique

Par exemple si on veut évaluer la fonction *f* en $x = 2$:

```
>> subs(f,x,2)
```

Si on veut affecter aux paramètres a = 2; b = 3; c = 9 :

```
>> g = subs(f, [a b c], [2 -3 9])
```

Si on veut convertir l'expression symbolique en nombre en double précision :

```
>> double(subs(g,x,2))
```

2. Résolution symbolique

Exemple 1 : l'équation $ax^2 + bx + c = 0$

```
>> syms x a b c
>> f = a*x^2 + b*x + c
>> solve(f)
```

Exemple 2 : l'équation $ln(y) - ln(r - y) = kt + C$

```
>> syms t y r k C
>> f = log(y) - log(r-y) - k*t - C
>> y = solve(f,y)
```

Exemple 3: Système de deux équations à deux inconnues

- Considérons les fonctions : $f(x,y) = y - 4x^2 + 3$ et $g(x,y) = y^2 + \frac{1}{4}x^2 - 1$.
- Représenter sur la même figure les graphes de $f(x,y) = 0$ et $g(x,y) = 0$, pour $x \in [-2,2]$ et $y \in [-3,3]$.

```
clear;clc
[x,y]= meshgrid ([ -2 : 0.02 : 2 ] , [ -3 : 0.03 : 3 ]);
fxy = y - 4*x.^2 + 3;gxy = y.^2 + 0.25*x.^2 -1;
v = [0 , 0];
contour (x,y,fxy,v);hold on
xlabel('x');ylabel('y');
contour (x,y,gxy,v);hold off
```

- Que représentent les points d'intersection des deux graphes ?

```
syms x y
f = y - 4*x^2 +3;
g = y^2 + .25*x^2 -1;
[a b] = solve(f,g);
[a b]
double([a b])
```

Exemple 4 : Résolution d'une équation à deux variables

L'équation $f(x,y)=0$ peut être résolue pour *y* en fonction de *x* en utilisant le calcul symbolique.

```
syms x y
f = x - exp(x*y);
solve(f,y)
```

3. Solutions numériques à une dimension

Une des méthodes numériques les plus connues pour résoudre les équations $f(x)=0$ est la méthode de Newton. La suite récurrente de Newton est définie par une valeur initiale x_0 proche de la racine et par la relation :

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

Exemple

Considérons la fonction $f(x)=sin(x)-\frac{x}{2}$.

Ecrire des m-files (fichiers function), dont les noms *f*.m et *df*.m pour la fonction et sa dérivée.

Dans la suite un simple m-file (script) qui implémente la méthode de Newton.

```
x0 = input(' enter la valeur initiale ')
N = input(' enter le nombre d''itérations ')
% On crée l'espace mémoire pour les itérations.
x = zeros(N,1);
x(1) = x0;
for n = 1:N-1
x(n+1) = x(n) - f(x(n))/df(x(n));
end
[x, f(x)]
```

Un problème d'optimisation a la forme standard suivante :

$$
\begin{aligned}
&Minimise \quad f(x_1, x_2, ..., x_n) \quad fonction\ objective\\
&Contraintes: \quad h_1(x_1, x_2, ..., x_n) = 0\\
&\qquad h_2(x_1, x_2, ..., x_n) = 0\\
&\qquad h_l(x_1, x_2, ..., x_n) = 0\\
&\qquad g_1(x_1, x_2, ..., x_n) \le 0\\
&\qquad g_2(x_1, x_2, ..., x_n) \le 0\\
&\qquad g_m(x_1, x_2, ..., x_n) \le 0\\
&\quad x_i^l \le x_i \le x_i^m, \quad i = 1, 2, ..., n
\end{aligned}
$$

Exemple :

L'exemple suivant permettra la création de courbes représentatives des fonctions objectives ainsi de définir l'optimisation graphique de la solution du problème.

$$
\begin{aligned}
&Minimise \quad f(x_1, x_2) = (x_1 - 3)^2 + (x_2 - 2)^2\\
&Contraintes: \quad h_1(x_1, x_2): 2x_1 + x_2 = 8\\
&\qquad h_2(x_1, x_2): (x_1 - 1)^2 + (x_2 - 4)^2 = 4\\
&\qquad g_1(x_1, x_2): x_1 + x_2 \le 7\\
&\qquad g_2(x_1, x_2): x_1 - 0.25.x_2^2 \le 0\\
&\qquad 0 \le x_1 \le 10; \quad 0 \le x_2 \le 10
\end{aligned}
$$

Créer des fichiers function dans lesquels :

- la fonction objective `f(x1, x2) = (x1 -3).^2 + (x2 -2).^2` doit être évaluer en tout point `[X1,X2]` et enregistrer en `f`

```
function f = obj_fonc(X1,X2)
f = (x1 -3).^2 + (x2 -2).^2;
```

- l'équation `h1(x1, x2): 2*X1 + X2` doit être évaluer en tout point `[X1,X2]` et enregistrer en `h1`

```
function h1 = eq1(X1,X2)
h1 = 2*X1 + X2;
```

- l'équation `h2(x1, x2): (X1 - 1).^2 + (X2 - 4).^2` doit être évaluer en tout point `[X1,X2]` et enregistrer en `h2`

```
function h2 = eq2(X1,X2)
h2 = (X1 - 1).^2 + (X2 - 4).^2;
```

- l'inéquation `g1(x1, x2): X1 + X2` doit être évaluer en tout point `[X1,X2]` et enregistrer en `g1`

```
function g1 = ineq1(X1, X2)
g1 = X1 + X2;
```

l'inéquation `g2(x1, x2): X1 - 0.25*X2.^2` doit être évaluer en tout point `[X1,X2]` et enregistrer en `g2`

```
function g2 = ineq2(X1,X2)
g2 = X1 - 0.25*X2.^2;
```

Le script suivant permet de représenter la solution graphique du problème.

```
x1 = 0:0.1:10;
x2 = 0:0.1:10;
[X1 X2] = meshgrid(x1,x2);

f = obj_fonc(X1,X2)
h1  =  eq1(Xl,X2);
h2  =  eq2(X1,X2);
g1 = ineql(Xl,X2);
g2 = ineq2(Xl,X2);

[C,h] = contour(x1,x2,f,'g');
clabel(C,h);
set(h,'LineWidth',1)
xlabel('Valeurs x_1','FontName','times','FontSize',12,'FontWeight','bold');
ylabel('Valeurs x_2','FontName','times','FontSize',12,'FontWeight','bold');

[C1,hh1] = contour(x1,x2,h1,[8,8],'b-');
clabel(C1,hh1);
set(hh1,'LineWidth',2)
k1 = gtext('h1');
set(k1,'FontName','Times','FontWeight','bold','FontSize',14,'Color','blue')

[C2,hh2] = contour(x1,x2,h2,[4,4],'b--');
clabel(C2,hh2);
set(hh2,'LineWidth',2)
k2 = gtext('h2');
set(k2,'FontName','Times','FontWeight','bold','FontSize',14,'Color','blue')

[C3,gg1] = contour(x1,x2,g1,[8,8],'r-');
clabel(C3,gg1);
set(gg1,'LineWidth',2)
hold on
k3 = gtext('g1');
set(k3,'FontName','Times','FontWeight','bold','FontSize',14,'Color','red')
```

```
[C4,gg2] = contour(x1,x2,g2,[0,0],'r--');
clabel(C4,gg2);
set(gg2,'LineWidth',2)
k4 = gtext('g2');
set(k4,'FontName','Times','FontWeight','bold','FontSize',14,'Color','blue')
```

1. Modélisation

En Recherche Opérationnelle (RO), modéliser un problème consiste à identifier :

- Les variables intrinsèques (inconnues).
- Les différentes contraintes auxquelles sont soumises ces variables.
- L'objectif visé (optimisation).

Dans un problème de programmation linéaire (PL) les contraintes et l'objectif sont des fonctions linéaires des variables. On parle aussi de programme linéaire.

2. Formes standards d'un programme linéaire (PL)

$$\begin{aligned}&Minimiser \quad f(x): c^T x \quad fonction \quad objective\\ &Contraintes: \quad g(x): Ax = b,\\ &\qquad\qquad x \geq 0, b \geq 0, A \in R^{mxn}, \quad rang(A) = m \leq n\end{aligned}$$

3. Transformations sous forme standard

Il est toujours possible de revenir à la forme standard

1. Si $b_i \prec 0 \rightarrow multiplier\ b_i\ et\ ligne\ A_i\ par -1$.

2. $max\, c^T x \Leftrightarrow min - c^T x$.

3. $Ax \geq b \Leftrightarrow Ax - y = b, \quad y \geq 0$.

4. $Ax \leq b \Leftrightarrow Ax + y = b, \quad y \geq 0$.

5. $$\begin{aligned}&x_{min} \leq x \leq x_{max} \Rightarrow z = x - x_{min} \geq 0, \quad y \geq 0\\ &\qquad\qquad z + y = x - x_{min}, \quad y \geq 0\end{aligned}$$

4. Résolution graphique (PL à 2 variables)

Pour modéliser des problèmes à l'aide de la PL :

- Plusieurs algorithmes existent, dont le simplexe.
- Pour des problèmes avec deux variables, on peut résoudre graphiquement (aide à comprendre la structure du problème)

Les contraintes où apparaissent des inégalités correspondent géométriquement à des demi-plans. Intersection de ces demi-plans représente l'ensemble des variables satisfaisant à toutes les contraintes. L'ensemble des contraintes est un polygone convexe.

Exemple

$$\begin{aligned}&Maximiser \quad f(X): 990x_1 + 900x_2 + 5250\\ &Contraintes: \quad g_1(X): 0.4x_1 + 0.6x_2 \leq 8.5\\ &\qquad\qquad g_2(X): 3x_1 - x_2 \leq 25\\ &\qquad\qquad g_3(X): 3x_1 + 6x_2 \leq 70\\ &\qquad\qquad x_1 \geq 0; \quad x_2 \geq 0\end{aligned}$$

A partir du PL de l'exemple écrit sous forme normale, on peut construire un PL sous forme standard :

$$\begin{aligned}
&\textit{Minimiser} \quad f(X): -990x_1 - 900x_2 - 5250 \\
&\textit{Contraintes :} \quad g_1(X): 0.4x_1 + 0.6x_2 + x_3 = 8.5 \\
&\qquad g_2(X): 3x_1 - x_2 + x_4 = 25 \\
&\qquad g_3(X): 3x_1 + 6x_2 + x_5 = 70 \\
&\qquad x_1 \geq 0, x_2 \geq 0, x_3 \geq 0, x_4 \geq 0, x_5 \geq 0
\end{aligned}$$

Ou sous forme matricielle :

$$c = [-990 \quad -900 \quad 0 \quad 0 \quad 0]^T$$

$$A = \begin{bmatrix} 0.4 & 0.6 & 1 & 0 & 0 \\ 3 & -1 & 0 & 1 & 0 \\ 3 & 6 & 0 & 0 & 1 \end{bmatrix}, \quad X = [x_1 \quad x_2 \quad x_3 \quad x_4 \quad x_5]^T, \quad b = [8.5 \quad 25 \quad 70]^T$$

Une fonction MATLAB pour tracer des lignes droites de type $ax + by = c, (c \geq 0)$ est donnée par le code suivant :

```
function ret = trac_ligne(x1,x2,a,b,c,type)
% x1,x2 représentent deux abscisses de x
% type indique le type de la ligne à tracer
% inf pour (<=), sup pour (>=), ega pour (=) et non pour (rien)

if (type == 'non')
   str1 = 'b';
   str2 = 'b';
   cmult = 1;
elseif (type == 'ega')
      str1 ='m';
      str2 ='m';
else
   str1 = 'g';
   str2 = 'r';
end

% Les valeurs pour tracer les tirets
% en fonction de la direction de l'inégalité
if (type ~= 'non' | 'ega')
   if (type == 'inf')
      cmult = +1;
   else cmult = -1;
   end
end
% Couleur (rouge) est utilisé pour dessiner une ligne parallèle à 10%
% Sup/Inf à la valeur de c (selon la direction de l'inégalité)
% cfac est un facteur pour tracer les contraintes en tirets
% Conditions sur c
if (abs(c) >= 10)
   cfac = 0.025;
elseif (abs(c)> 5) & (abs(c) <  10)
   cfac = 0.05;
else
   cfac = 0.1;
end
```

```
if (c == 0 )
   cdum = cmult*0.1;
else
   cdum = (1 + cmult* cfac)*c;
end

% Conditions sur b
if ( b ~= 0)
   y1 = (c - a*x1)/b;
   y1n = (cdum - a* x1)/b;
   y2 = (c - a* x2)/b;
   y2n = (cdum - a*x2)/b;
else
   % Si b = 0, on utilise la couleur (magenta) pour tracer les contraintes
   str1 = 'm';
   str2 = 'm';
   y1 = x1; % y1 est la même que x1
   y2 = x2;  % y2 est la même que x2
   x1 = c/a;
   x2 = c/a;
   y1n = 0;  % y = 0;
   y2n = 0;  % y = 0
end

if (a == 0)
   str1 = 'm'; %
   str2 = 'm'; %
end;

% Tracer des axes horizontal et vertical en noir
hh = line([x1,x2],[0,0]);
set(hh,'LineWidth',1,'Color','k');
hv = line([0,0],[x1,x2]);
set(hv,'LineWidth',1,'Color','k');

% Tracer les lignes droites

h1 = line([x1 x2], [y1,y2]);

if (type == 'non')
   set(h1,'LineWidth',2,'LineStyle','--','Color',str1);
else
   set(h1,'LineWidth',1,'LineStyle','-','Color',str1);
end

if (b ~= 0)&(a ~= 0)
   text(x1,y1,num2str(c));
end
if( b == 0)|(a == 0)| (type == 'non') | (type == 'ega')
   grid
   ret = [h1];
   return,
end
grid;
h2 = line([x1 x2], [y1n,y2n]);
set(h2,'LineWidth',0.5,'LineStyle',':','Color',str2);
grid
hold on
ret = [h1 h2];
```

Ecrire un programme MATLAB "fichier script", faisant appel à la fonction `trac_ligne` permettant la représentation graphique de l'exemple ci-dessus.

Méthode du pivot

Soit à résoudre le système suivant de 3 équations à 3 inconnues par la méthode du pivot (dans Matlab) :

$$\begin{bmatrix} -0{,}04 & 0{,}04 & 0{,}12 \\ 0{,}56 & -1{,}56 & 0{,}32 \\ -0{,}24 & 1{,}24 & -0{,}28 \end{bmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 3 \\ 1 \\ 0 \end{pmatrix}$$

déterminer x_1, x_2, x_3.

1. On définit tout d'abord la matrice argument dans Matlab par :

```
A=[-0.04 0.04 0.12 3;0.56 -1.56 0.32 1; -0.24 1.24 -0.28 0] ;
```

2. On choisit la ligne1 comme ligne pivot : `a(1,1)=-0.04`,

on divise la ligne1 par `a(1,1)` → `b(1,:)=a(1,:)/a(1,1)` → nouvelle ligne,

on annule le 1er terme de la ligne2 et le 1er terme de la ligne3 :

```
b(2,:)=a(2,:)-b(1,:)*a(2,1);
b(3,:)=a(3,:)-b(1,:)*a(3,1);
```

On obtient après calculs :

```
b =
1 -1 -3 -75
0 -1  2  43
0  1 -1 -18
```

3. On choisit la ligne2 comme ligne pivot.

On divise cette ligne par `b(2,2)=-1`, on obtient :

`c(2,:)=b(2,:)/b(2,2)` → nouvelle ligne → `0 1 -2 -43;`

on annule le terme `b(1,2)` de la ligne1 et le terme `b(3,2)` de la ligne3 :

`c(1,:)=b(1,:)-c(2,:)*b(1,2)` → `1 0 -5 118;`

`c(3,:)=b(3,:)-c(2,:)*b(3,2)` → `0 0  1 25;`

4. On choisit la ligne3 comme ligne pivot `c(3,:)`.

On divise cette ligne par `c(3,3)=1`, on obtient :

`d(3,:)=c(3,:)` → nouvelle ligne → `0 0 1 25;`

on annule dans la ligne1 `c(1,3)` et dans la ligne2 `c(2,3)` :

`d(1,:)=c(1,:)-d(3,:)*c(1,3)` → `1 0 0 7,`

`d(2,:)=c(2,:)-d(3,:)*c(2,3)` → `0 1 0 7.`

D'où la matrice d s'écrit :

```
d =
    1  0  0  7
    0  1  0  7
    0  0  1  25
```

et la solution est : $\begin{cases} x_1 = 7 \\ x_2 = 7 \\ x_3 = 25 \end{cases}$;

Le programme suivant (dans Matlab) permet de résoudre le système précédent :

```
 clear %effacer toutes les variables en mémoire dans Matlab
    a=[-0.04 0.04 0.12 3;0.56 -1.56 0.32 1;-0.24 1.24 -0.28 0];a
    x=[0 0 0]'; % le vecteur colonne x de composantes xi est initialisé
% 1er point pivot ligne1 et calculs sur les lignes 2 et 3
    b(1,:)=a(1,:)/a(1,1);
    b(2,:)=a(2,:)-b(1,:)*a(2,1);
    b(3,:)=a(3,:)-b(1,:)*a(3,1);b
% 2ème pivot ligne2 et calculs sur les lignes 1 et 3
    c(2,:)=b(2,:)/b(2,2);
    c(1,:)=b(1,:)-c(2,:)*b(1,2);
    c(3,:)=b(3,:)-c(2,:)*b(3,2);c
% 3ème pivot ligne3 et calculs sur les lignes 1 et 2
    d(3,:)=c(3,:)/c(3,3);
    d(1,:)=c(1,:)-d(3,:)*c(1,3);
    d(2,:)=c(2,:)-d(3,:)*c(2,3);d
%Solutions recherchées
    x(1)=d(1,4);
    x(2)=d(2,4);
    x(3)=d(3,4);x
```

Méthode du SIMPLEXE

Principe

Evaluer la fonction aux sommets de l'ensemble admissible.

Les sommets sont obtenus en calculant les solutions de base.

Opérations élémentaires sur les lignes

1. Permutation de deux lignes

Ex. Permutation ligne1 et 3 : $E = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \longrightarrow \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$;

2. Multiplication d'une ligne par un scalaire

Ex. Multiplication ligne2 par α : $E = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$;

3. Ajout d'une ligne multipliée par un scalaire à une autre ligne

Ex. Ajout de 2*ligne1 à la ligne4 : $E = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 2 & 0 & 0 & 1 \end{bmatrix}$.

Soit à appliquer la méthode du SIMPLEXE sur l'exemple :

$$\begin{aligned} &Maximiser \quad f(X): 990x_1 + 900x_2 + 5250 \\ &Contraintes: \quad g_1(X): 0.4x_1 + 0.6x_2 \le 8.5 \\ &\qquad\qquad g_2(X): 3x_1 - x_2 \le 25 \\ &\qquad\qquad g_3(X): 3x_1 + 6x_2 \le 70 \\ &\qquad\qquad x_1 \ge 0; \quad x_2 \ge 0 \end{aligned}$$

La méthode du SIMPLEXE est appliquée sur le problème écrit sous forme standard :

$$\begin{aligned} &Minimiser \quad f(X): -990x_1 - 900x_2 - 5250 \\ &Contraintes: \quad g_1(X): 0.4x_1 + 0.6x_2 + x_3 = 8.5 \\ &\qquad\qquad g_2(X): 3x_1 - x_2 + x_4 = 25 \\ &\qquad\qquad g_3(X): 3x_1 + 6x_2 + x_5 = 70 \\ &\qquad x_1 \ge 0, x_2 \ge 0, x_3 \ge 0, x_4 \ge 0, x_5 \ge 0 \end{aligned}$$

Ou sous forme matricielle :

$$A=\begin{bmatrix} 0.4 & 0.6 & 1 & 0 & 0 \\ 3 & -1 & 0 & 1 & 0 \\ 3 & 6 & 0 & 0 & 1 \end{bmatrix},\quad b=\begin{bmatrix} 8.5 \\ 25 \\ 70 \end{bmatrix},\quad c=\begin{bmatrix} -990 & -900 & 0 & 0 & 0 \end{bmatrix}^T.$$

Tableau 1 :

La première ligne indique les noms des variables et la dernière est reservée à la fonction objective

Base initiale $B=\begin{bmatrix} x_3 & x_4 & x_5 \end{bmatrix}$**;**

Solution de base initiale $X=\begin{bmatrix} 0 & 0 & 8.5 & 25 & 70 \end{bmatrix}^T$ **;**

Fonction de coût initiale $f=-5250$

x_1	x_2	x_3	x_4	x_5	**b**	**V.B**
0.4	*0.6*	*1*	*0*	*0*	*8.5*	x_3
$\boxed{3}$	*-1*	*0*	*1*	*0*	*25*	x_4
3	*6*	*0*	*0*	*1*	*70*	x_5
-990	*-900*	*0*	*0*	*0*	*5250*	*-f*

```
% les coefficients du tableau 1

A=[0.4 0.6 1 0 0 8.5;3 -1 0 1 0 25;3 6 0 0 1 70;-990 -900 0 0 0 5250]
```

Tableau 2 :

x_1 entre dans la base

La ligne du pivot est obtenue à partir de la ligne2 du tableau 1:

x_1 remplace x_4 dans la base

on divise la ligne2 par 3:

Le premier élément de la premiere ligne doit se réduit à *0* par ***ligne1* - [*0.4*(ligne2)*]**

La troixieme ligne est obtenue par ***ligne3* - [*3*(ligne3)*]**

La troixieme ligne est obtenue par ***ligne4* + [*990*(ligne2)*]**

x_1	x_2	x_3	x_4	x_5	**b**	**V.B**
0	*0.7333*	*1*	*-0.1333*	*0*	*5.1667*	x_3
1	*-0.3333*	*0*	*0.3333*	*0*	*8.3333*	x_1
0	$\boxed{7}$	*0*	*-1*	*1*	*45*	x_5
0	*-1230*	*0*	*330*	*0*	*13530*	*-f*

La valeur négative du coefficient dans la ligne4 du tableau 2 indique que d'autre itération est nécessaire.

```
% Tableau 2 :
% On divise la sixième colonne par la colonne du coefficient entrant dans
% la base
format short
A(:,6)./A(:,1)
% deuxième ligne est le pivot et x4 est l'élément sortant de la base
A
% l'élément A(2,1) doit être 1 :
A(2,:)= A(2,:)/ A(2,1);
% l'élément A(1,1) doit être 0 :
A(1,:)= A(1,:)- 0.4*A(2,1);
% l'élément A(3,1) doit être 0 :
A(3,:)= A(3,:)- A(3,1)* A(2,:);
% l'élément A(4,1) doit être 0 :
A(4,:)= A(4,:)- A(4,1)* A(2,:);
% fin du tableau 2 : Solution ne converge pas à cause de l'élément A(4,2)
% négative
```

Tableau 3 :

x_2 entre dans la base

La ligne du pivot est obtenue à partir de la ligne3 du tableau 2:

x_2 remplace x_5 dans la base

ligne3 = (ligne3/7)

ligne1=ligne1 - [*0.7333(*ligne3*)]**

ligne2=ligne2 + [*0.3333(*ligne3*)]**

ligne4=ligne4 + [*1230(*ligne3*)]**

x_1	x_2	x_3	x_4	x_5	***b***	**V.B**
0	*0*	*1*	*-0.0285*	*-0.1047*	*0.4524*	x_3
1	*0*	*0*	*0.2857*	*0.0476*	*10.4762*	x_1
0	*1*	*0*	*-0.1428*	*0.1428*	*6.4285*	x_2
0	*0*	*0*	*154.28*	*175.71*	*21437.14*	*-f*

```
% Tableau 3 :
% x2 est le coefficient entrant dans la base
% calcul du coefficient sortant de la base
A(:,6)./A(:,2)
% troisième ligne est le pivot et x5 est l'élément sortant de la base
% l'élément A(3,2) doit être 1 :
A(3,:)= A(3,:)/ A(3,2);
% l'élément A(1,2) doit être 0 :
A(1,:)= A(1,:)- A(1,2)* A(3,:);
```

```
% l'élément A(2,2) doit être 0 :
A(2,:)= A(2,:)- A(2,2)* A(3,:);
% l'élément A(4,2) doit être 0 :
A(4,:)= A(4,:)- A(4,2)* A(3,:);
A
% créer un fichier nommé TP5 dans "current directory".
diary('TP5')
```