Projet LM206
(Introduction à Scilab) :
Linguistique

Introduction :

Le but du projet était de réaliser un programme qui permette de mesurer la distance entre deux textes, que ce soit deux textes de différents auteurs, ou encore l'évolution des textes d'une même personne dans le temps à partir de la fréquence de l'occurrence des mots. Le texte donné en exemple était celui des voeux du président, nous avons choisi de continuer dans la même fibre avec l'évolution dans le temps, et la distance des discours pour les présidentielles de 1988, 2002 et 2006 d'Arlette Laguillier et de Jean-Marie Lepen. Nous nous attendions à un certain rapprochement au fil du temps puisque que le Front National utilise une phraséologie de plus en plus socialisante.

Notre étude sera réalisée à l'aide du logiciel *Scilab*, pseudo clone libre de *Matlab* développé par l'INRIA, qui permet de faire du calcul numérique, matriciel, et des applications graphiques telles que visualisation de courbes et surfaces.

Programme utilisé :

Plaçons nous dans *Scilab*, et ouvrons *Scipad* (en tapant la commande scipad). Ecrire les commandes (instructions et fonctions) dans *Scipad* pour les exécuter ensuite dans *Scilab* permet de pouvoir revenir aisément sur sa commande si l'on se trompe, et la reformuler, mais aussi de copier/coller les instructions, et surtout de les enregistrer, pour garder une trace de son travail, qui, qui plus est, ne sera pas brouillée par les "résultats" renvoyés lorsqu'on exécute les instructions.

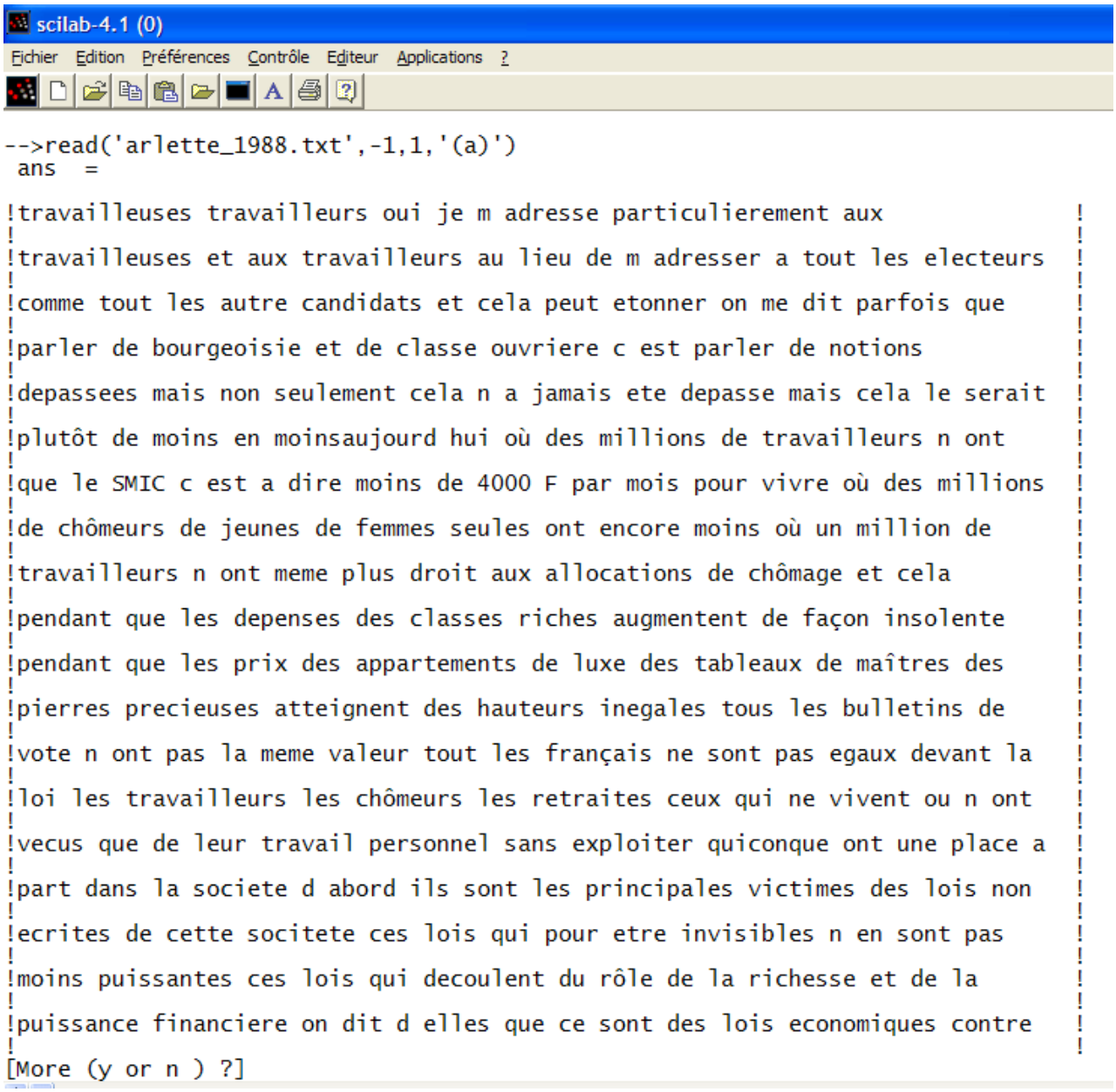
Il nous faut tout d'abord avoir accès aux fichiers que nous avons au préalable épurés (grâce au logiciel de traitement de texte *emacs*) de leurs ponctuations, accents, majuscules, et autres retours à la ligne gênants. Pour cela, il existe la fonction *Scilab* read. Plaçons-nous dans le dossier où se trouvent les documents,

```
cd
cd Documents
cd textes_du_projet
```

puis affectons les textes à des variables aux noms choisis et adaptés, nous permettant de reconnaître rapidement les valeurs qu'elles contiennent.

```
arl_88_1=read('arlette_1988.txt',-1,1,'(a)')
arl_95_1=read('arlette_1995.txt',-1,1,'(a)')
arl_02_1=read('arlette_2002.txt',-1,1,'(a)')
arl_06_1=read('arlette_2006.txt',-1,1,'(a)')
pen_88_1=read('lepen_1988.txt',-1,1,'(a)')
pen_02_1=read('lepen_2002.txt',-1,1,'(a)')
pen_06_1=read('lepen_2006.txt',-1,1,'(a)')
```

On obtient alors, si l'on demande ce que contient la variable arl_88_1, par exemple :

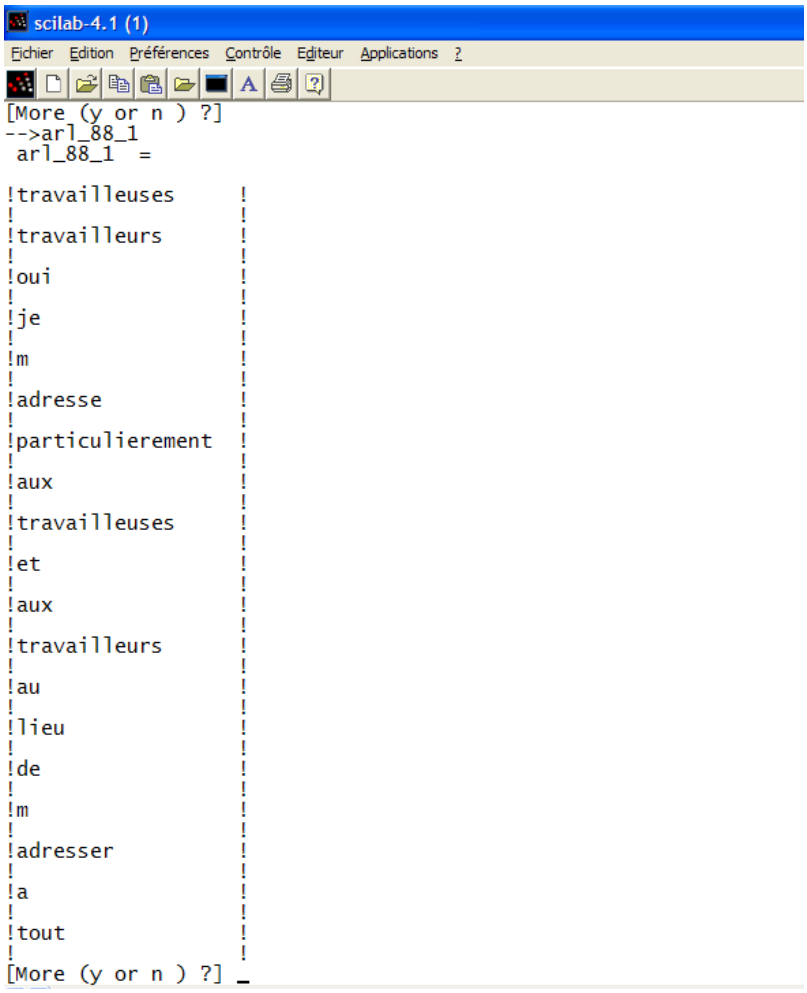


Pour un texte dont il ne resterait des "éléments gênants" que les retours à la ligne permettant des séparations de paragraphes, comme par exemple pour les vœux de Jacques Chirac que nous avions sur votre site http://nicolas.limare.net/tmp/, nous serions amenés à utiliser la fonction suivante. Elle ajoute des espaces à la fin de chaque ligne (dans une boucle allant de la première ligne à la dernière ligne, prod(size(t)) rendant ici le nombre de lignes de la matrice t, car c'est le produit du nombre de lignes et de colonnes de la matrice t) puis concatène toutes les lignes du texte pour les réduire à une seule ligne (fonction strcat).

```
function x=mise_forme(t)
  txt=t
  for i=1:prod(size(txt))
     txt (i)=txt(i)+' '
     end;
  txt=strcat(txt)
endfunction
```

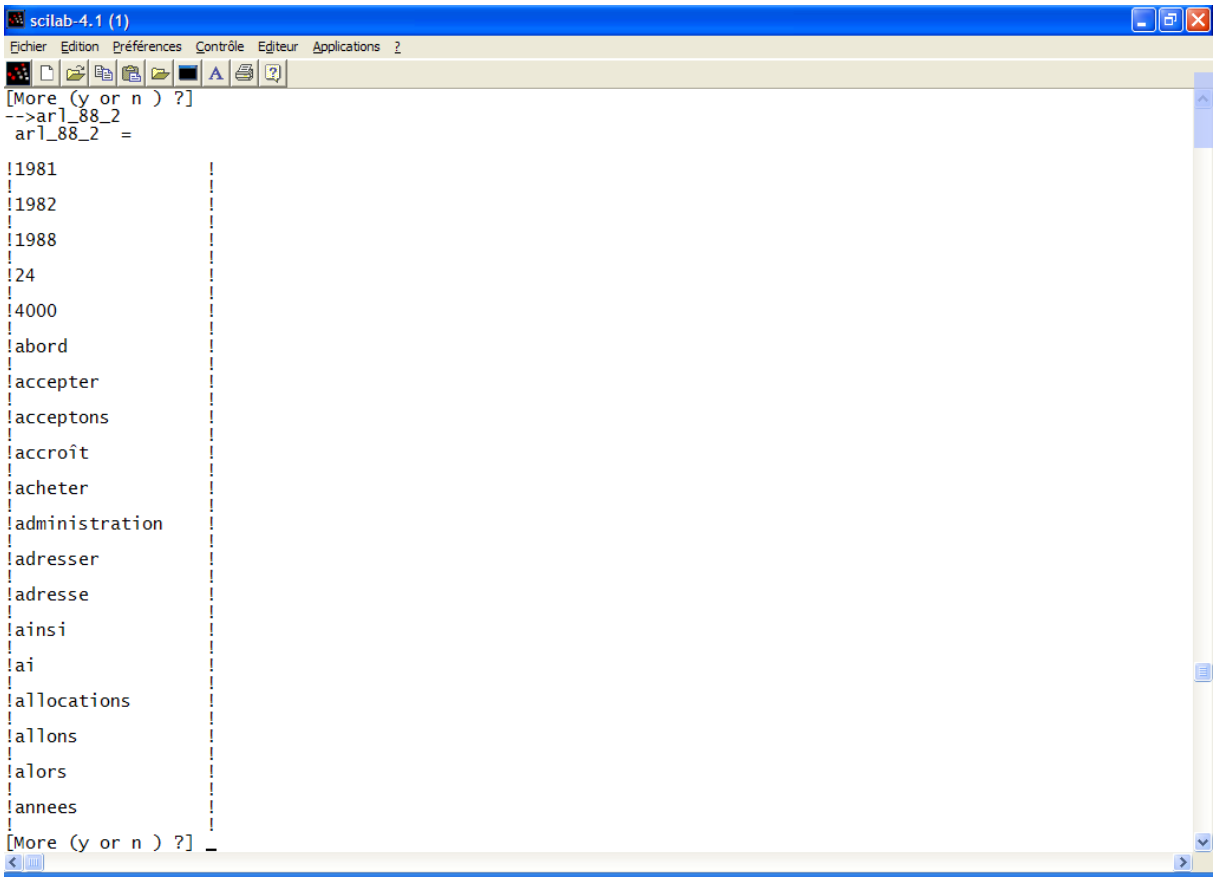
Nous n'en avons ici pas l'utilité, il nous faut néanmoins pour notre étude séparer tous les mots du texte, c'est à dire obtenir une seule colonne avec un mot par ligne uniquement (fonction tokens). (Voir exemple du nouvel arl_88_1 ci-dessous :)

```
arl_88_1=tokens(arl_88_1)
arl_95_1=tokens(arl_95_1)
arl_02_1=tokens(arl_02_1)
arl_06_1=tokens(arl_06_1)
pen_88_1=tokens(pen_88_1)
pen_02_1=tokens(pen_02_1)
pen_06_1=tokens(pen_06_1)
```



Créons d'autres variables, contenant les textes épurés et mis en colonne comme précédemment, en supprimant les multiples occurences du même mot – fonction unique -, et en les triant - fonction sort -, afin d'obtenir une colonne où n'apparaîtra qu'une fois chaque mot présent dans le texte, et où ils seront triés par ordre alphabétique.

```
arl_88_2=sort(unique(arl_88_1))
arl_95_2=sort(unique(arl_95_1))
arl_02_2=sort(unique(arl_02_1))
arl_06_2=sort(unique(arl_06_1))
pen_88_2=sort(unique(pen_88_1))
pen_02_2=sort(unique(pen_02_1))
pen_06_2=sort(unique(pen_06_1))
```



Créons la fonction frequencemot binaire, qui prendra comme arguments un mot et un texte "mis en colonne" comme nous l'avons fait plus haut (variables du type arl_88_1). find(t==mot) trouve les places des occurrences de 'mot' dans le texte 't'. exemple :

```
-->find(arl_88_1=='travailleuses')
 ans  =

    1.    9.    839.    1321.    1571.

-->_
```

length(find(t==mot)) trouve donc le nombre d'occurence du mot 'mot' dans le texte 't'. Pour trouver la fréquence, on divise donc ce nombre par le nombre total de mots. La fonction rendra donc la fréquence d'UN mot 'mot' dans le texte 't'.

```
function [x]=frequencemot(mot,t)
  x=length(find(t==mot))/prod(size(t))
endfunction
```

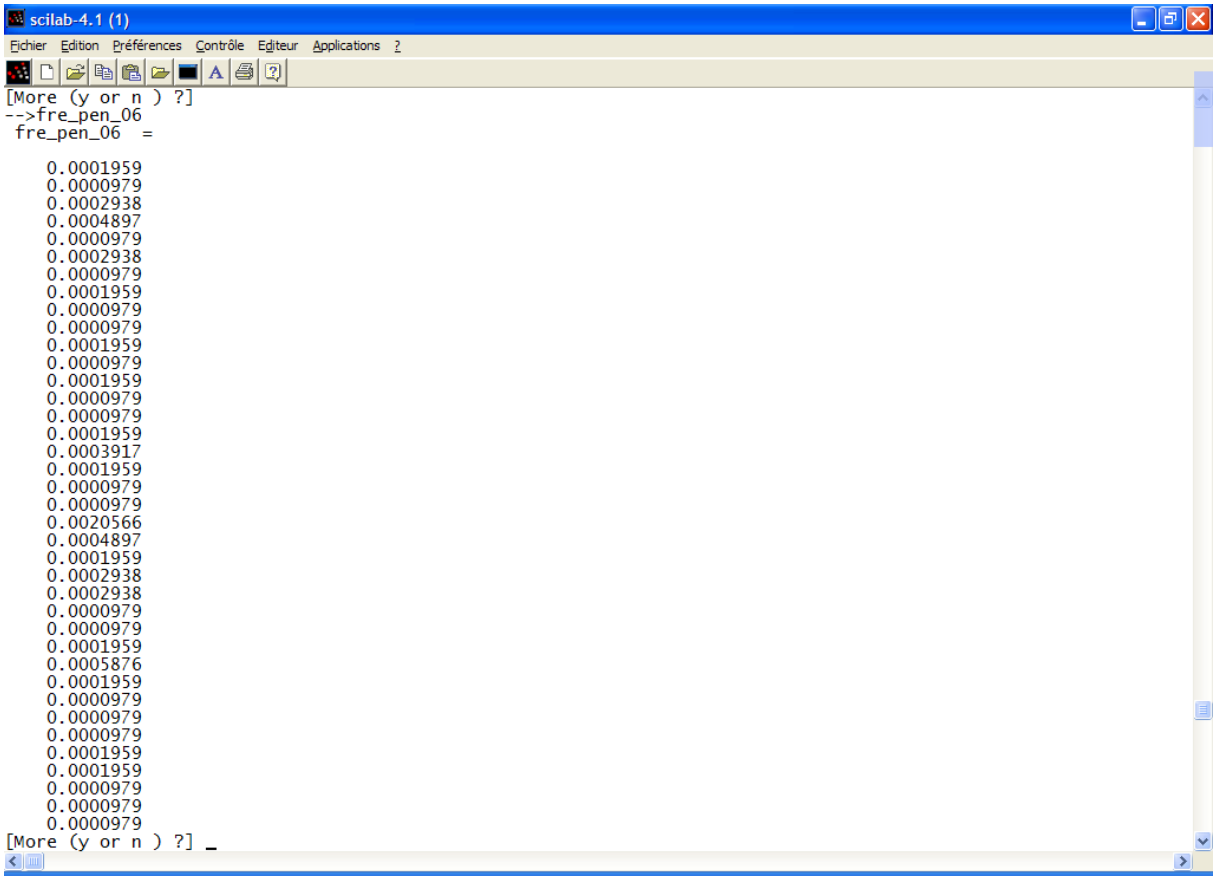
A présent, on peut s'aider de cette fonction pour en créer une plus importante, calculant la fréquence de chaque mot d'un texte 'txt2' dans un texte 'txt', et les mets dans une matrice colonne, la position des fréquences étant la position du mot correspondant dans 'txt2'.

```
function x=frequence(txt2,txt)
  for i=1:prod(size(txt2))
    fre(i)=frequencemot(txt2(i),txt)
  end
  x=fre
endfunction
```

Calculons ainsi les fréquences pour tous les textes.

```
fre_arl_88=frequence(arl_88_2,arl_88_1)
```

```
fre_arl_95=frequence(arl_95_2,arl_95_1)
fre_arl_02=frequence(arl_02_2,arl_02_1)
fre_arl_06=frequence(arl_06_2,arl_06_1)
fre_pen_88=frequence(pen_88_2,pen_88_1)
fre_pen_02=frequence(pen_02_2,pen_02_1)
fre_pen_06=frequence(pen_06_2,pen_06_1)
```



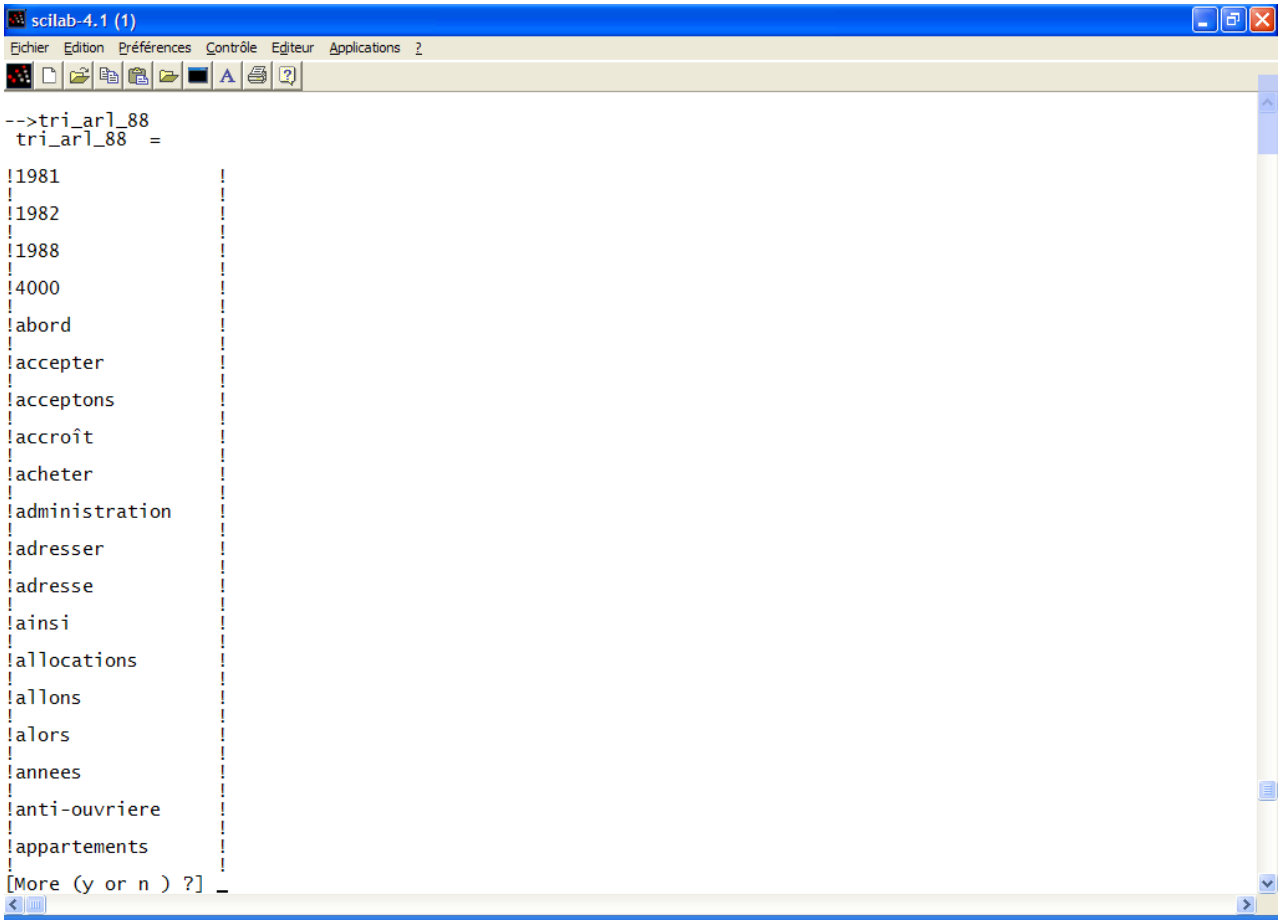
Nous obtenons bien des matrices colonnes contenant les fréquences de chaque mot des textes.

A présent, nous nous rendons compte que tous ces mots ne sont pas significatifs. Les mots de liaison, articles, pronoms, et plein d'autres mots ne peuvent avoir une fréquence intéressante par rapport aux autres mots (noms communs, verbes ou adjectifs pour la plupart ont un sens précis). Nous allons donc évincer de notre compte de mots, tous ceux qui font strictement moins de quatre lettres, grâce à la fonction tri. Les mots très courts ne sont en effet que très rarement utiles dans ce type d'étude. Si length(txt2(i))>3, c'est-à-dire si la longueur du i-ème mot de 'txt2' est supérieur à 3, on le garde, et on incrémente k de 1.

```
function x=tri(txt2)
  k=1
  for i=1:prod(size(txt2))
    if (length(txt2(i))>3) then
      tritxt(k)=txt2(i)
      k=k+1
    end
  end
  x=tritxt
endfunction


tri_arl_88=tri(arl_88_2)
```

```
tri_arl_95=tri(arl_95_2)
tri_arl_02=tri(arl_02_2)
tri_arl_06=tri(arl_06_2)
tri_pen_88=tri(pen_88_2)
tri_pen_02=tri(pen_02_2)
tri_pen_06=tri(pen_06_2)
```



Nous appliquons la fonction tritxt à tous nos textes. On observe effectivement qu'il n'y a plus de mot inférieur à quatre lettres.

Cependant, après avoir calculé la fréquence d'apparition de chaque mot et n'avoir conservé que les mots de 4 lettres et plus, nous avons remarqué que de nombreux mots avaient la même racine, provenaient du même lexique, mais étaient comptés comme des mots différents, tel: chômeur, chômeurs, chômeuse, chômage etc... Nous avons donc décidé de créer un tableau où tous les mots ayant la même racine ne compteraient que pour un seul mot, et leurs fréquences seraient ajoutées. Pour cela nous avons dû faire un choix approximatif et, sachant que la racine d'un mot est le plus souvent au début de celui-ci, considérer que la racine d'un correspondant à ses 4 premières lettres. Nous avons donc "fusionné" les mots ayant les 4 premières lettres identiques et en comparant les tableaux originaux et les tableaux ainsi créés, on voit que ce choix n'entraîne que peu d'erreurs, malgré que ce système puisse donner le même sens à "anti-termite" et "antisèche", par exemple (ou, ici, 'allocations' et 'allons'.

Observons pas à pas le sens de cette fonction :

```
function red=reduction(t)
  txt=t
  tritxt=tri(t)
  tritxt2=tri(t)
```

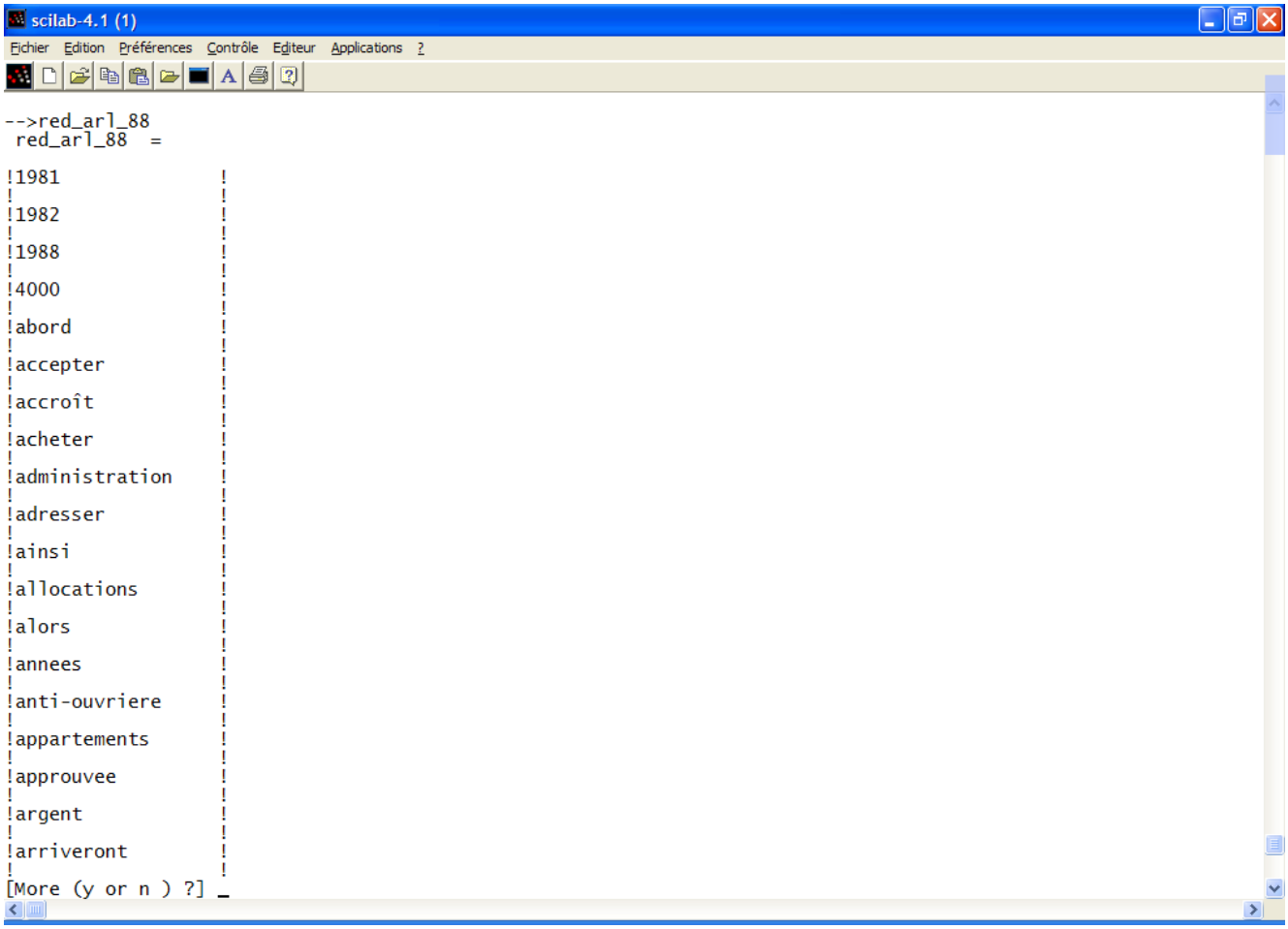
Ci-dessus, on a créé 2 tableaux, l'original et celui que l'on va modifier.

```
n=prod(size(tritxt))
k=1
```

La fonction part permet d'extraire d'une suite de caractères ceux que l'on veut. On sait que tous les mots ayant "la même racine" seront à la suite dans le tableau trié puisqu'il est ordonné par ordre alphabétique, on compare donc 2 mots consécutifs, si ils ont la même racine alors on ôte le second du tableau que l'on modifie. Puis on passe au mot suivant.

```
for i=1:(n-1)
  if (part(tritxt(i),1:1:4)==part(tritxt(i+1),1:1:4))
    tritxt2=[tritxt2(1:1:k,:);tritxt((i+2):1:n,:)]
  else k=k+1
  end
 end
 red=tritxt2
endfunction

red_arl_88=reduction(tri_arl_88)
red_arl_95=reduction(tri_arl_95)
red_arl_02=reduction(tri_arl_02)
red_arl_06=reduction(tri_arl_06)
red_pen_88=reduction(tri_pen_88)
red_pen_02=reduction(tri_pen_02)
red_pen_06=reduction(tri_pen_06)
```



Nous observons dans l'exemple ci-dessus la disparition du mot "acceptons", dont la fréquence sera ajoutée à celle d'"accepter", ce qui est tout à fait logique.

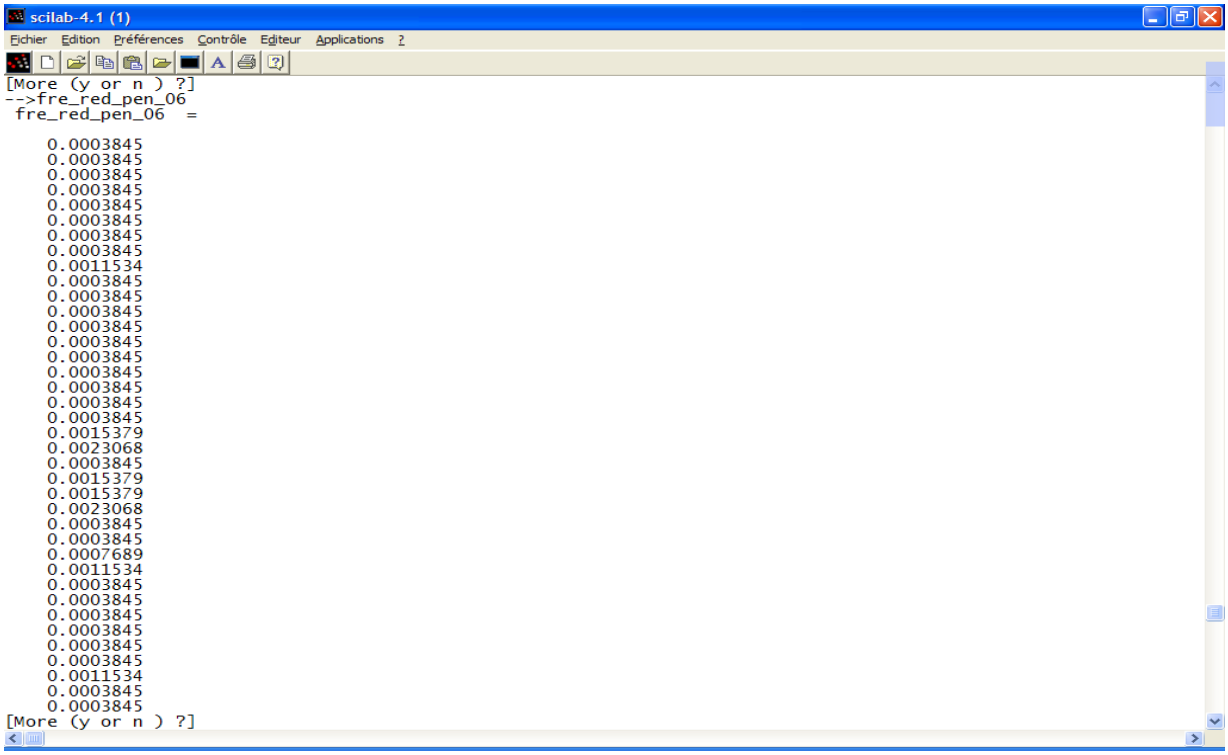
Créons la fonction fre_red qui permettra de donner un tableau fréquence correspondant au texte réduit.

```
function fre=fre_red(t)
  tritxt=tri(t)
  fre_txt=frequence(tritxt,t)
  fre_txt2=fre_txt
  n=prod(size(tritxt))
  k=1
```

Pour créer le tableau fréquence on procède de la même manière que précédemment sur la fonction reduction sauf qu'au lieu de retirer simplement la fréquence du mot ayant "la même racine" que le précédent, on fait la somme des fréquences de ces 2 mots consécutifs, que l'on place dans la case du premier.

```
  for i=1:(n-1)
    if (part(tritxt(i),1:1:4)==part(tritxt(i+1),1:1:4))
      fre_txt2(k)=fre_txt2(k)+fre_txt(i+1)
      fre_txt2=[fre_txt2(1:1:k,:);fre_txt((i+2):1:n,:)]
    else k=k+1
    end
  end
  fre=fre_txt2
endfunction

fre_red_arl_88=fre_red(arl_88_2)
fre_red_arl_95=fre_red(arl_95_2)
fre_red_arl_02=fre_red(arl_02_2)
fre_red_arl_06=fre_red(arl_06_2)
fre_red_pen_88=fre_red(pen_88_2)
fre_red_pen_02=fre_red(pen_02_2)
fre_red_pen_06=fre_red(pen_06_2)
```



Une fois les tableaux textes quelques peu "épurés" et les tableaux fréquence correspondants créés, nous pouvons passer au calcul de la distance entre 2 textes à partir de ceux-ci.

Il est possible de comparer deux textes par rapport à la fréquence des mots qu'ils contiennent, uniquement sur les mots qu'ils ont en commun. Il faut donc avant toute chose créer le tableau de leurs mots communs.
Néanmoins, plutôt que de prendre les mots strictement identiques, nous avons choisis d'utiliser le même procédé que pour la réduction et avons considéré comme identiques les mots ayant la "même racine". Lorsqu'on utilise cette fonction, on suppose txt1 et txt2 triés et réduits.

```
function x=mots_communs(txt1,txt2)
  k=1
  for i=1:prod(size(txt1))
     for j=1:prod(size(txt2))
        if (part(txt1(i),1:1:4)==part(txt2(j),1:1:4))
           txt3(k)=txt1(i)
           k=k+1
        end
     end
  end
  x=txt3
endfunction
```

Nous avons ensuite calculé ce qu'on appelle la distance euclidienne entre les fréquences des mots communs à ces deux textes. C'est à dire que si n est la taille du tableau des mots communs, nous avons considéré que l'ensemble des fréquences possibles de ces mots était l'espace vectoriel Rn, le tableau des fréquences des mots communs dans un des textes constitue un point de Rn et la distance euclidienne est alors :

racine carrée de $((x1-y1)^2+(x2-y2)^2+...+(xn-yn)^2)$

si x1,...,xn et y1,...yn constituent respectivement les fréquences des mots communs de 1 à n dans le premier et le second texte.
Une fois que les tableaux des fréquences des mots communs sont créés, faire ce calcul devient très simple par des opérations sur les tableaux.

```
function x=distance(txt1,txt2)
  tritxt1=tri(txt1)
  tritxt2=tri(txt2)
    com=mots_communs(tritxt1,tritxt2)              //création du tableau des mots communs
     fre_com_1=frequence(com,txt1)
     fre_com_2=frequence(com,txt2)
```

Les deux dernières lignes correspondent à la créations des tableaux de fréquences des mots communs dans chacun des textes.

```
      dist_com=(fre_com_1-fre_com_2).*(fre_com_1-fre_com_2)
    distance=sqrt(sum(dist_com))                   //calcul de la distance euclidienne
  x=distance
endfunction
```

Comme prévu, on peut ensuite mesurer l'évolution des textes d'une personne, et l'évolution de la distance entre les textes de 2 personnes. C'est ce que nous avons réalisé, tout d'abord entre plusieurs discours d'Arlette Laguillier, puis entre plusieurs discours de Jean-Marie Lepen, puis enfin le discours de l'un par rapport à celui de l'autre à la même date.

Evolution d'Arlette Laguillier entre 1988 et 2006

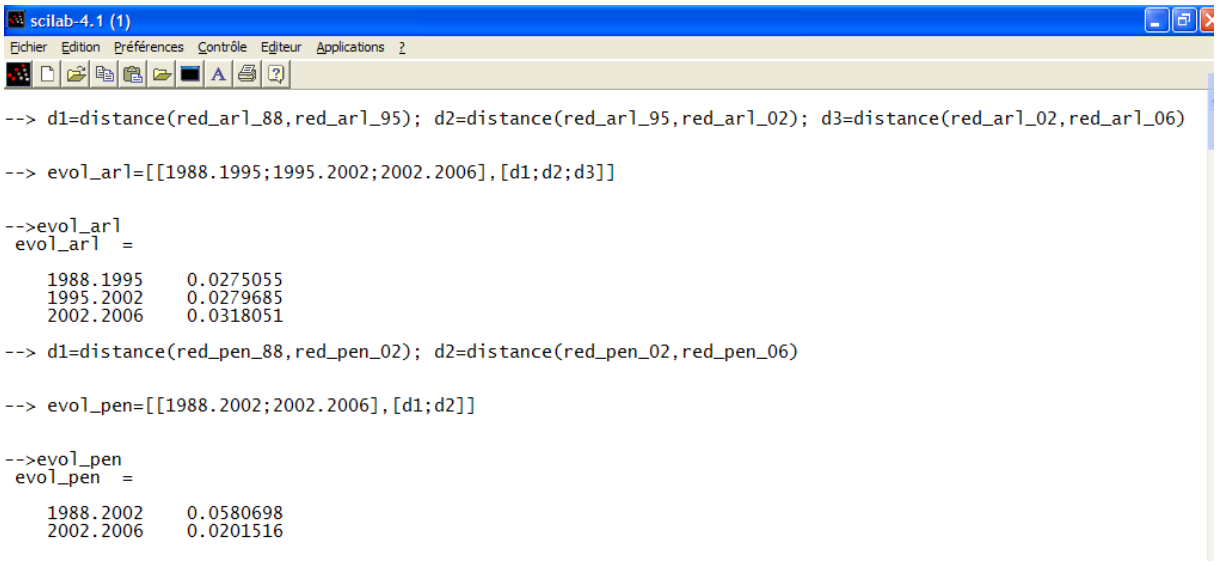
```
d1=distance(red_arl_88,red_arl_95)
d2=distance(red_arl_95,red_arl_02)
d3=distance(red_arl_02,red_arl_06)
```

```
evol_arl=[[1988.1995;1995.2002;2002.2006],[d1;d2;d3]]
```

Evolution de Jean-Marie Lepen entre 1988 et 2006

```
d1=distance(red_pen_88,red_pen_02)
d2=distance(red_pen_02,red_pen_06)

evol_pen=[[1988.2002;2002.2006],[d1;d2]]
```



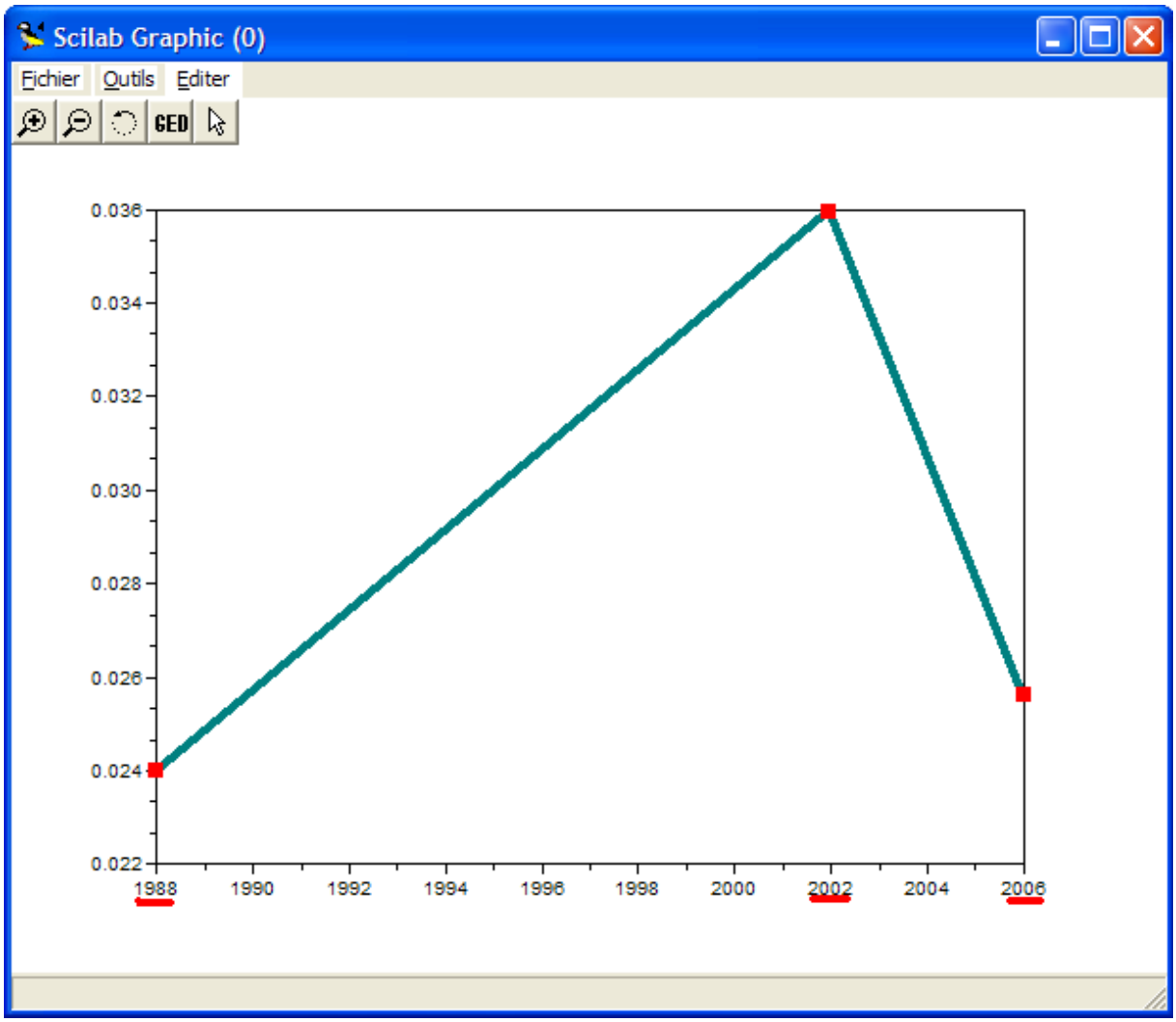
Pour ces deux derniers, étant donné que nous n'avons que très peu de dates, nous n'aurions obtenu que des courbes à 2 ou 3 points, c'est pourquoi nous avons favorisé l'idée du "tableau" (matrice), plus représentatif et agréable à comprendre.

En revanche, pour l'évolution de la distance entre les textes de Le Pen et Laguillier entre 1988 et 2006, nous avons choisi de former une courbe, un peu plus lisible que pour les dernières études.

Evolution Lepen/Arlette

```
d1=distance(red_arl_88,red_pen_88)
d2=distance(red_arl_02,red_pen_02)
d3=distance(red_arl_06,red_pen_06)

plot2d([1988,2002,2006],[d1,d2,d3])
```



Les résultats confirment à peu près nos prévisions bien que cette technique ne prenne en compte que les mots eux-mêmes et non leur sens.

Conclusion :

Il est pratique d'utiliser *Scilab* lorsque nous sommes en contact d'opérations matricielles. On dispose avec ce logiciel d'une bibliothèque simple mais complète. Le langage est proche du langage mathématique et on s'y retrouve assez facilement. Il est également pratique et avantageux de pouvoir utiliser les graphiques rapidement, pour visualiser ses résultats.

Cependant, *Scilab* est un logiciel assez lent. De plus, puisqu'on ne déclare pas les variables ni même leurs types, on y fait moins attention et il est parfois peu simple de savoir si une fonction est applicable à telle ou telle valeur. Il est également perturbant de ne pouvoir revenir sur quelque chose qu'on a écrit, une instruction ratée… on est obligé de la ré-écrire. Il faut donc bien faire attention à toujours noter ce que l'on fait dans *Scipad* pour s'en sortir plus aisément, mais lorsqu'on exécute une fonction ou instruction à partir de *Scipad* directement, elle devient alors difficilement lisible dans la fenêtre *Scilab*.