

Initiation Scilab
Agrégation de Mathématiques

G.Sallet
INRIA
Université de Metz

2006-2007

Table des matières

1	Introduction	7
2	Comment accéder à SCILAB	9
2.1	Machines UNIX	9
2.2	MacIntosh, Linux, Windows	9
2.3	Lancer Scilab	9
2.4	Le dossier de travail	10
2.4.1	Où se trouve-t-on ?	10
2.4.2	Comment naviguer dans l'arborescence	11
2.4.3	Chemin absolu et relatifs	12
2.5	Savoir ce qu'il y a dans un fichier	13
2.6	Edition dans la fenêtre de travail	13
2.7	L'utilisation du help	13
3	A propos de SCILAB	15
3.1	syntaxe <i>variable = expression</i>	15
3.2	Expression seule	17
3.3	Affichage ou non	17
3.4	$x = x + 1$	17
4	Matrices et vecteurs	19
4.1	Définitions	19
4.2	Les nombres flottants	19
4.3	Matrices	20
4.4	Attention à la saisie	21
4.5	Les vecteurs	22
5	Sauvegarde de variables	25
6	Manipulations de matrices et de vecteurs	27
6.1	L'opérateur « deux points » : les vecteurs réguliers	27
6.1.1	vecteur régulièrement espacé	28
6.2	Extraction d'éléments d'une matrice	29
6.2.1	Indexation à deux indices	29

6.2.2	Indexation à un seul indice	30
6.2.3	Le dernier indice	30
6.3	« Columnisation » et remodelage	30
6.3.1	Columnisation	30
6.3.2	Redimensionner une matrice	31
6.4	Création de matrices particulières	32
6.4.1	La matrice vide	32
6.4.2	Matrice nulle	32
6.4.3	Matrice de uns	32
6.4.4	Matrice identité	33
6.4.5	Matrices Aléatoires	33
6.5	Création par concaténation	33
6.6	Extraction, suppression de vecteurs lignes, de vecteurs colonnes .	34
6.6.1	Lignes et colonnes d'une matrice	34
6.6.2	Suppression d'une ligne, d'une colonne	34
6.7	Résumé des syntaxes du paragraphe	34
7	Opérations sur les matrices et les vecteurs	37
7.1	Addition, multiplication par un scalaire	37
7.1.1	Un raccourci bien pratique	38
7.2	Transposition	38
7.3	Produit de deux matrices	40
7.4	Résumé des syntaxes du paragraphe	40
8	Opérations sur les tableaux de même taille	43
8.1	Encore un danger	43
9	Fonctions de manipulations de données	45
9.1	Fonctions élémentaires	45
9.2	La commande <code>find</code>	46
10	Matrice particulières	47
10.1	Extraire la diagonale principale	47
10.2	Autres diagonales	47
10.3	Triangulaire extraite	48
10.4	Matrice de Toeplitz	48
10.5	Matrices test	49
11	Produit de Kronecker	51
12	Quelques astuces utiles	53
12.1	Une matrice dont toutes les colonnes sont égales	53
12.2	Une matrice dont toutes les lignes sont égales	53
12.3	Changer la diagonale principale d'une matrice	53
12.4	Matrice de Vandermonde	54

13 Exercices	55
13.1	55
13.2	55
14 Systèmes linéaires	57
14.1 Ajustement de données	57
14.2 Résolution d'un système linéaire	58
15 L'arithmétique de l'ordinateur	59
15.1 Les erreurs d'arrondis	59
15.1.1 Un premier exemple	59
15.1.2 Deuxième exemple	60
15.2 Les nombres dans l'ordinateur : Les nombres flottants	65
15.3 Arithmétique IEEE	67
15.4 Explication des cas précédents	68
16 Résolution aux moindres carrés	71
16.1 Un danger	72
16.2 Application à l'ajustement de données	73
16.3 Un exemple	74
17 Gestion de fichiers	79
17.1 Chaînes de caractères	79
17.2 Récupérer un fichier ASCII	79
17.3 Un exemple	80
17.4 Fichiers excel	81
18 Etude d'un cas réel	83
18.1 Phénomènes périodiques	83
18.2 Mise en forme des données	84
18.3 Interpolation des données manquantes	88
18.4 Interpolation linéaire	88
18.5 Tendances générale	90
18.6 Ajustement périodique	92
18.6.1 Première harmonique	92
18.6.2 Deuxième harmonique	94
19 Graphiques 2D	97
19.1 Les fenêtres graphiques	97
19.1.1 Créer une fenêtre	97
19.1.2 Effacer une fenêtre	97
19.1.3 Sélectionner une fenêtre	98
19.2 Diviser la fenêtre	98
19.3 Résumé	98
19.4 la commande simple <code>plot2d</code>	99
19.4.1 Utilisation simple	99

19.4.2	Exemple	99
19.5	Personnaliser un graphique	100
19.5.1	<code>style</code>	100
19.5.2	<code>rect</code>	103
19.5.3	<code>logflag</code>	103
19.5.4	<code>frameflag</code>	104
19.5.5	<code>axesflag</code>	107
19.5.6	<code>nax</code>	107
19.5.7	<code>leg</code>	107
20	Transformée de Fourier discrète	109
20.1	FFT	110
20.2	FFT inverse	110
20.3	Définitions	110
20.4	Application : le taux de CO2 à Maunaloa	111
20.5	FFT, Scilab et les touches du téléphone	116
20.5.1	Identifier la touche?	117
20.5.2	Procédure	117
20.6	El Niño	120
20.6.1	Identifier la tendance générale	120
20.6.2	Signal moins la tendance	120
20.7	Taches solaires	122

Chapitre 1

Introduction

SCILAB est un système interactif pour le calcul numérique, les calculs d'ingénierie, la programmation et la visualisation. Le logiciel SCILAB est un environnement complet. On peut résoudre des calculs complexes beaucoup plus rapidement qu'avec des langages comme le C ou le Fortran. C'est ce que l'on peut appeler un environnement pour la résolution de problèmes.

Il existe deux autres logiciels qui ont ces mêmes fonctionnalités : MATLAB et OCTAVE.

Le plus connu est MATLAB. Il est utilisé en écoles d'ingénieurs, dans les IUT, les universités, dans de nombreux bureaux d'études industriels. C'est un logiciel complet, qui possède de nombreuses boîtes à outils, il est de plus interfaçable avec le logiciel de mathématique symbolique Maple. Il est très coûteux, car protégé par un copyright commercial, chaque boîte à outils est payante, toute mise à jour l'est aussi et ce logiciel doit posséder une licence par poste de travail ou par groupe de postes. Vous n'avez pas accès au logiciel source bien que de nombreux programmes soient directement issus de bibliothèques mathématiques libres, comme par exemple la bibliothèque NETLIB, en particulier des programmes écrits en langages fortran ou en C, programmes que l'on obtient librement et gratuitement en se procurant le CD-rom de cette bibliothèque : <http://www.netlib.org>.

C'est un logiciel écrit à l'origine par des chercheurs de l'INRIA (Institut National de Recherche en Informatique et Automatique) ; ce logiciel a une licence proche de la licence GPL, il est libre et gratuit. Il est maintenant développé par un consortium comprenant l'INRIA et l'ENPC (Appedge, Axs Ingénierie, Cril Technology, CEA, CNES, Dassault Aviation, EDF, ENPC, Esterel Technologies, INRIA, PSA Peugeot Citroën, Renault et Thales). La syntaxe est très proche de MATLAB. L'interface graphique est moins professionnelle que MATLAB mais s'améliore. Vous pouvez le télécharger gratuitement sur [http : www.scilab.org](http://www.scilab.org), ainsi qu'une nombreuse documentation gratuite. Scilab tourne sous UNIX, LINUX, Window et Mac OSX.

La première version de MATLAB date de 1978 et a été écrite par Steve Moler. L'idée originale était de concevoir une aide à l'enseignement et de procurer

une introduction un package d'algèbre linéaire numérique LINPACK. Le nom MATLAB autrement dit MATRIX LABORATORY vient de là. A peu près au même moment (1980) le logiciel BLAISE, puis BASILE était développé à l'INRIA.

De par leur origine SCILAB et MATLAB sont très bien adapté à l'utilisation de l'algèbre linéaire. Ce cours a pour objectif de fournir des compléments d'algèbre linéaire ainsi qu'une introduction à SCILAB. Enfin ce cours donnera quelques applications de Scilab en ajustement de données et traitement du signal.

Un des problèmes les plus fréquemment rencontrés en calcul scientifique est la résolution de systèmes d'équations linéaires simultanées. Autrement dit de résoudre un système du type

$$Ax = b$$

où A , b sont des matrices et des vecteurs de dimension appropriée. On verra comment résoudre ce type de problème.

Chapitre 2

Comment accéder à SCILAB

Il faut d'abord que SCILAB soit installé sur votre machine !

2.1 Machines UNIX

Il suffit de taper Scilab dans la console

2.2 MacIntosh, Linux, Windows

En général il suffit de double-cliquer sur l'icône Scilab.

2.3 Lancer Scilab

Si on lance SCILAB une fenêtre de commande apparaît avec un symbole --> (un prompt) attendant des données tapées au clavier.

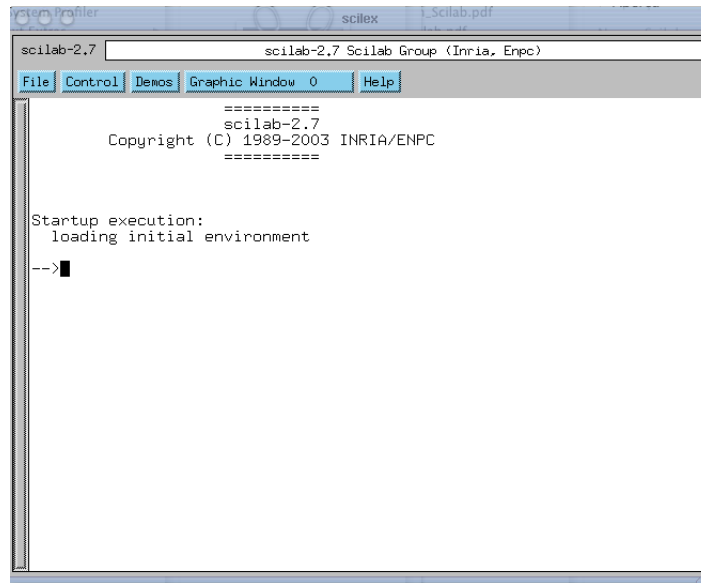


FIG. 2.1 – Une fenêtre de commande SCILAB

Dans la fenêtre de commande SCILAB vous tapez vos données (ou entrées) et quand c'est terminé sur la touche retour chariot (ou return)

ATTENTION SCILAB DISTINGUE ENTRE LES MINUSCULE ET MAJUSCULES

On en fera l'expérience tout à l'heure.

2.4 Le dossier de travail

Du moment où vous avez lancé Scilab, jusqu'à celui où vous quittez le logiciel, vous avez effectué ce que l'on appelle une session. Tout les calculs et opérations sont perdus quand vous quittez le logiciel à moins que vous les ayez sauvegardées. Scilab travaille toujours automatiquement dans un dossier (répertoire) par défaut dans lequel il cherche les données et également où il peut sauvegarder d'autres données. Par exemple si vous avez fait un calcul ou créé une courbe avec Scilab, vous pouvez sauvegarder ces données pour un usage futur.

Pour éviter une saturation il vaut mieux créer votre propre dossier, sinon chaque utilisateur laissera des traces dans le dossier par défaut. Ce sera vite le chaos dans ce dossier. Sans compter le risque que l'on modifie ou on écrase par inadvertance vos données.

2.4.1 Où se trouve-t-on ?

C'est donné par la commande `pwd`

```
-->pwd
ans =

/Users/sallet

-->
```

Cela signifie que l'on se trouve dans l'arborescence

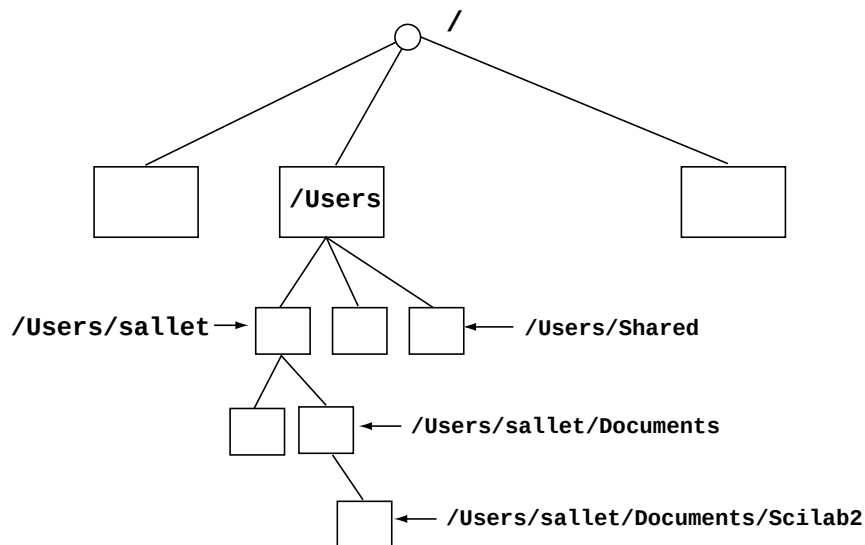


FIG. 2.2 – Détail de l'arborescence

la commande `pwd` signifie *print working directory* : afficher le répertoire de travail.

2.4.2 Comment naviguer dans l'arborescence

Cela se fait à l'aide de la commande `chdir` « change directory »
On l'utilise de la façon suivante :

```
chdir (' route d'accès')
```

Par exemple on veut aller de `/Users/sallet` à `/Users/Shared`. On tape alors

```
-->chdir('/Users/Shared')
ans =
```

0.

```
-->pwd
ans =

/Users/Shared
```

Le 0 indique que l'opération a réussi (sinon Scilab vous averti qu'il y a une erreur). On le vérifie en retapant `pwd`. Si maintenant on veut aller dans `Scilab2` il suffit d'indiquer la route

```
--> chdir('/Users/sallet/Documents/Scilab2')
ans =

0.
```

La chemin indiqué à partir de la racine s'appelle un chemin absolu.

2.4.3 Chemin absolus et relatifs

On peut aussi utiliser des raccourcis. Quand le dossier à atteindre est au-dessus, on remonte en utilisant `chdir('..')` ce qui fait remonter au dossier immédiatement au-dessus :

```
-->chdir('..')
ans =

0.

-->pwd
ans =

/Users/sallet/Documents
```

De même `chdir ('../..')` fait remonter de deux crans

```
--chdir('../..')
ans =

0.

-->pwd
ans =

/Users
```

Pour remonter à la racine la commande `chdir('/')` suffit.

Pour descendre à partir d'un dossier où l'on est il suffit d'écrire le chemin à partir du dossier où l'on est. Actuellement dans notre exemple nous sommes à `/Users`. Si l'on veut atteindre `Scilab2` il suffit de taper

```
-->chdir('sallet/Documents/Scilab2')
ans =
```

```
0.
```

```
-->pwd
ans =
```

```
/Users/sallet/Documents/Scilab2
```

et voilà.

2.5 Savoir ce qu'il y a dans un fichier

Pour savoir ce qu'il y a dans le fichier courant (working directory) on utilise la commande `ls`, qui signifie lister. Scilab donne la liste de ce qu'il y a dans le « working directory »

2.6 Edition dans la fenêtre de travail

Pour aller à la ligne (dans le cas où les instructions de commande sont trop longues) on tape ... puis la touche return ou enter.

Pour rappeler d'ancienne commande on utilise la touche « flèche vers le haut ». A chaque pression on remonte d'une commande. Pour redescendre on utilise la flèche vers le bas. Ceci est intéressant quand on a des instructions compliquées qui ressemblent à peu de chose près à une instruction passée. On rappelle cette commande que l'on peut ensuite modifier (éditer) en utilisant les flèches gauche et droite.

2.7 L'utilisation du help

Il suffit de taper la commande `help` suivie de la commande dont vous avez besoin de connaître la syntaxe.

L'aide de Scilab utilise la convention suivante Les termes entre crochets dans une syntaxe Scilab sont optionnels.

On peut aussi utiliser la commande `help` seulement qui vous donne accès aux chapitres de l'aide en ligne. On peut aussi accéder à cette fenêtre d'aide par le menu, en cliquant sur aide.

Chapitre 3

A propos de SCILAB

SCILAB est un langage interprété. Ceal veut dire qu'à chaque étape on obtient un résultat. Dans les langages informatiques usuels FORTRAN, C, ou Pascal on doit avoir un programme prêt avant d'avoir une réponse. Comme langage interprété SCILAB se comporte comme une calculette. Mais il y a quelque chose en plus.

Les calculettes ordinaires, et même les calculettes plus sophistiquées comme les calculettes programmables, ne connaissent que les nombres. SCILAB peut s'utiliser comme une calculette capable d'effectuer des opérations sur des vecteurs et des matrices.

Qu'est ce qu'une matrice? tout simplement un tableau rectangulaire de nombres comportant un certain nombre de lignes et de colonnes. Certains tableaux ont des significations plus particulières. C'est ainsi qu'un tableau de n lignes et 1 colonne, ou un tableau de 1 ligne, n colonnes sera interprété par SCILAB comme un vecteur. Les nombres habituels, encore appelés scalaires, seront des tableaux 1×1 !

On consacrera un chapitre particulier à ces deux notions vecteurs et matrices. Il n'est pas nécessaire de connaître les matrices pour utiliser ce tutoriel.

SCILAB permet également de visualiser vos résultats graphiquement par des courbes et des surfaces.

Très rapidement l'utilisation d'une calculette, même matricielle, ne suffit plus et l'on est très vite amené, pour éviter les tâches répétitives, à utiliser la puissance de SCILAB qui est aussi un langage. On écrit alors des scripts (autrement dit des suites d'instruction) et des routines appelées M-files.

SCILAB est un langage « à expression ». Nous allons expliquer ce que cela signifie.

3.1 syntaxe *variable = expression*

Le principe est

$$variable = expression$$

par exemple le fait de taper la commande

```
-->A=[ 1 2 3 ; 4 5 6]
```

quand on presse la touche « retour chariot »(encore appelée « return ») ou l'autre touche dénommée « enter »produit la sortie

```
A=[1 2 3 ; 4 5 6 ]
A  =
!   1.    2.    3.  !
!   4.    5.    6.  !
```

```
-->
```

Cela signifie que SCILAB a créé une variable nommée A , et qu'il lui a affecté pour valeur le tableau carré, la matrice 2 lignes-par-3 colonnes, donnée en exemple.

On assigne à la variable dont le nom est donné à gauche, la valeur de l'expression de droite.

Si la variable n'existe pas, SCILAB la crée c'est ainsi qu'il a créé la variable A . Cette variable est en mémoire et elle est facilement accessible :

En effet si maintenant on tape

```
--> A
```

et on appuie sur return, on obtient de nouveau

```
-->A
A  =
123456.

-->
```

Les noms de variables consistent en une lettre suivi d'un nombre quelconque de lettres, chiffres ou underscore. SCILAB utilise seulement (sic) les 24 premiers caractères d'un nom de variable. **ATTENTION** SCILAB distingue entre les minuscules et les majuscules. Un nom de variable doit commencer par une lettre ou un des caractères spéciaux $_$, $\#$, $!$, $\$$ ou $?$. Le caractère suivant doit être une lettre ou un des caractères suivants $_$, $\#$, $!$, $\$$ ou $?$ (remarquez que le $\%$ ne doit pas suivre.) Formellement un nom de variable est une chaîne de caractère. Certaines variables prédéfinies sont précédées par $\%$. Par exemple $\%pi$, $\%i$, e qui sont respectivement π , $i^2 = -1$ et e .

3.2 Expression seule

Si on omet le nom de la variable et le signe = , par exemple si on tape une seule expression

```
12.256
```

après avoir tapé return, on obtient

```
ans =
```

```
12.2560
```

```
-->
```

Comme nom de la variable et le signe = ont été omis, une variable ans a été créée par défaut, et la valeur 12.256 lui a été assignée (attribuée, affectée).

Le registre **ans** existe. On peut faire par exemple **2*ans** on obtient

```
-->2*ans
```

```
ans =
```

```
24.512
```

Vous noterez que comme sur les calculettes la « virgule » européenne décimale dans les nombres réels est un « point ». anglo-saxon.

Une conséquence est que si une variable existe, par exemple *aa* , si vous écrivez **aa=0** l'ancienne valeur de **aa** sera écrasée et remplacée par la nouvelle, ici 0.

3.3 Affichage ou non

Si l'on ne précise pas SCILAB est comme une calculette et affiche toujours le résultat des commandes après l'appui sur return. Si l'on veut ne pas afficher on ajoute un « point-virgule ». Cela peut être utile si l'on manipule de grandes matrices.

```
A=1000 ;
```

Il ne se passe apparemment rien, cependant SCILAB à créé la variable *A* où si elle existe remplacé sa valeur par 1000

3.4 $x = x + 1$

L'autre conséquence de la notion de *variable = expression* est que des commandes comme

```
x=x+1; y=2*y ;
```

sont parfaitement licites. Cela signifie que SCILAB remplace dans le registre **x** la valeur qui y était par cette valeur à laquelle on ajouté 1 ...

Chapitre 4

Matrices et vecteurs

Les types d'objet que Scilab peut considérer sont les matrices réelles ou complexes (on verra plus loin la définition), les matrices polynômiales, les matrices booléennes, les matrices creuses, les matrices de chaînes de caractères et les listes.

4.1 Définitions

Le principal type de données considéré par SCILAB sont les matrices.

Un des types de base de SCILAB est constitué par les tableaux rectangulaires de nombres réels ou complexes. Ces tableaux ont des propriétés particulières et s'appellent des matrices. Si vous connaissez déjà les matrices, ce paragraphe pourra être lu rapidement. Il introduit les définitions et les notations SCILAB nécessaires pour la suite.

Les matrices sont fondamentales dans SCILAB et même si vous n'avez pas l'intention d'utiliser SCILAB pour des calculs matriciels vous devez vous familiariser avec les matrices, leurs manipulations, leurs opérations. L'outil de base est la matrice.

Une Matrice est un tableau rectangulaire de nombre flottants réels ou complexes.

4.2 Les nombres flottants

Les flottants, c'est à dire que l'on utilise la convention décimale (avec un point décimal). Les nombres imaginaires utilisent i . Un nombre complexe s'écrit toujours $a + ib$ où i , appelée unité imaginaire, vérifie $i^2 = -1$, et a et b sont des réels. La multiplication se fait naturellement ... Si vous ne connaissez pas les nombres complexes SCILAB les connaît, et se chargera des opérations!

Tous les nombres sont conservés en mémoire interne en utilisant le format IEEE flottant standard, avec 16 chiffres décimaux significatifs. Voir plus loin la partie 15.2 sur les nombres flottants et l'arithmétique IEEE.

Exemples de quelques entrées acceptées et comprises par SCILAB.

```
-->-3
ans =

- 3.

-->1.4145
ans =

1.4145

-->1.2E-5
ans =

0.000012

-->1+2*%i
ans =

1. + 2.i
```

Dans SCILAB le i mathématique est noté %i. On a de même

```
-->%pi
%pi =

3.1415927

-->%e
%e =

2.7182818

-->
```

4.3 Matrices

Définition 4.3.1 (Matrice) :

Une matrice est donc un tableau rectangulaire de nombres réels ou complexes. La taille de la matrice est le couple d'entiers (m, n) où m est le nombre de lignes et n le nombre de colonnes.

On repère un élément d'une matrice par sa position en ligne et colonne. On désigne souvent par $a_{i,j}$ le terme à l'intersection de la i -ème ligne et j -ème colonne de la matrice A . On notera

$$A = (a_{i,j})$$

Dans ce chapitre on se limitera aux réels, mais cela ne change rien.

Une matrice se rentre en SCILAB ligne après ligne entre deux crochets. Le changement de colonne est marqué par une espace ou une virgule , et le changement de ligne par un point-virgule ; Par exemple soit la matrice

$$\begin{bmatrix} 8 & 1 & 6 \\ 3 & 5 & 7 \\ 4 & 9 & 2 \end{bmatrix}$$

Elle se rentre en SCILAB par les commandes

```
-->A=[8 1 6 ; 3 5 7 ; 4 9 2]
A =
```

```
!   8.    1.    6. !
!   3.    5.    7. !
!   4.    9.    2. !
```

```
-->
```

La virgule (comma en anglais est équivalente à l'espace)

```
-->A=[8 ,1, 6 ; 3, 5 ,7 ; 4, 9, 2]
A =
```

```
!   8.    1.    6. !
!   3.    5.    7. !
!   4.    9.    2. !
```

```
-->
```

4.4 Attention à la saisie

L'espace est considéré soit comme un espace de séparation, soit comme faisant partie d'une expression. Cela peut engendrer des confusions. L'exemple suivant est tiré de BOT. Considérons les trois expressions :

```
-->m1 = [1+%i -1+%i; -1+%i 1-%i]
m1 =
!   1. + i   - 1. + i   !
! - 1. + i    1. - i    !

-->m2 = [1 +%i - 1 + %i; - 1 + %i 1 - %i]
m2 =
!   1.          - 1. + 2.i !
```

```

! - 1. + i      1. - i      !

-->m3 = [1  +%i  -1 +  %i; - 1 + %i 1 -%i]
m3  =
!  1.          i      - 1. + i      !
! - 1. + i      1.      - i          !

```

Dans `m1` il n'y a pas d'espace dans les expressions. Les espaces sont donc des changements de colonne.

Dans `m2`, il y a un espace entre le 1 et `+%i`. Scilab interprète 1 comme la première entrée. On trouve ensuite `+%i - 1 + %i`. Il y a des espaces entre les opérations binaires `+` et `-`, Scilab interprète comme un espace entre expression $i - 1 + i$. Par exemple :

```

-->[1 +%i]
ans  =

!  1.      i      !

-->[1 + %i]
ans  =

1. + i

```

Dans la deuxième ligne de `m2`, l'espace entre `%i` et 1 est interprété comme un changement de colonne. Ce qui donne l'expression finale

Dans `m3`, les deux premiers espaces sont interprétés comme des changements de colonne. En effet il n'y a pas d'espace entre le `-` et 1. Donc dans `+%i -1` l'espace est interprété comme un changement de colonne. Mais ensuite on trouve `-1 + %i` des espaces entre le symbole binaire `+`. Ils sont interprétés comme des espaces entre expression.

Pour éviter des erreurs on peut soit ne pas utiliser d'espace dans les expressions. Soit utiliser des parenthèses :

```

-->[(1 +%i) (-1 + %i); (- 1 + %i) (1 -%i)]
ans  =
!  1. + i      - 1. + i      !
! - 1. + i      1. - i      !

```

4.5 Les vecteurs

Les vecteurs sont des matrices particulières.

Définition 4.5.1 :

Un vecteur colonne est une matrice $n \times 1$

Un vecteur ligne est une matrice $1 \times n$

Quand on ne précise pas, un vecteur sera un vecteur colonne.

La longueur d'un vecteur est n

Un scalaire (réel ou complexe) est un vecteur 1×1

Syntaxe 4.5.1 :

On obtient la taille d'une matrice A par la commande `size(A)`

La longueur d'un vecteur x par la commande `length(x)`

```
-->x=[1,3,5]
```

```
x  =
```

```
!   1.    3.    5.  !
```

```
-->size(A)
```

```
ans  =
```

```
!   3.    3.  !
```

```
-->length(x)
```

```
ans  =
```

```
3.
```


Chapitre 5

Sauvegarde de variables

Lors d'une session on crée souvent des variables. Ce sont les variables courantes. Pour savoir ce que l'on a créé il suffit d'utiliser la commande `who`. On obtient alors la liste des variables créées. La commande `clear` tue toutes les variables non protégées. Certaines variables sont protégées `%pi`, `%e`, `%F` par exemple. Par exemple

```
-->clear
```

```
-->n=1;A=rand(2,2);z=2;
```

```
-->who
```

your variables are...

z	A	n	scicos_pal	with_tk
demolist	%helps	LANGUAGE	MSDOS	home
TMPDIR	xdesslib	percentlib		polylib
elemlib	utllib	statslib	alglib	siglib
autolib	roplib	soundlib	metalib	armalib
tdcslib	s2flib	mtlplib	SCI	%F
%z	%s	%nan	%inf	COMPILER
%pvm	%tk	\$	%t	%f
%io	%i	%e		

using 5301 elements out of 1000000.
and 50 variables out of 1791

your global variables are...

LANGUAGE	%helps	demolist	%browsehelp
%toolboxes		%toolboxes_dir	INDEX
using	1442 elements	out of	1661.
and	7 variables	out of	255

La commande `clear A a` ne détruira que les variables `A` et `a`
 Si on veut sauvegarder des variables on utilise la fonction `save`

Syntaxe 5.0.2 :

la commande `save('toto',A,n)` va sauver dans le fichier binaire `toto` les valeurs de `A` et `n`. Si le fichier `toto` n'existe pas, il sera créé.

Si on écrit `save('toto')` toutes les variables courantes seront sauvegardées.

On récupère les variables dans la file `toto` par `load('toto')`

Ces fichiers sont des formats binaires indépendants.

On peut aussi sauvegarder des matrices sous format ASCII. La norme ASCII est une forme de codage des caractères. La plus ancienne et la plus universelle de ces normes de codage est le code A.S.C.I.I. (American Standard Code for International Interchange). Il utilise des nombres de 7 chiffres binaires. Il code donc 128 caractères.

```
-->A=rand(2,2)
A =

!   0.8782165    0.5608486 !
!   0.0683740    0.6623569 !

-->write('mat.dat',A)

-->clear

-->A=read('mat.dat',2,2)
A =

!   0.8782165    0.5608486 !
!   0.0683740    0.6623569 !
```

Chapitre 6

Manipulations de matrices et de vecteurs

6.1 L'opérateur « deux points » : les vecteurs réguliers

L'opérateur : permet de créer des suites de réels réguliers.

Syntaxe 6.1.1 :

a:b est le vecteur ligne

- vide, c'est à dire la matrice vide, soit encore $[]$ si $a > b$
- $[a, a + 1, a + 2, \dots, a + n]$ si $a \leq b$ avec $a + n \leq b$

a:h:b est le vecteur

$[a, a + d, a + 2 * d, \dots, a + n * d]$ avec $a + n * d \leq b$

Par exemple

```
-->1.2 : 2.5
```

```
ans =
```

```
! 1.2 2.2 !
```

```
-->1.2:6.3
```

```
ans =
```

```
! 1.2 2.2 3.2 4.2 5.2 6.2 !
```

```
-->1:5
```

```
ans =
```

```
! 1. 2. 3. 4. 5. !
```

```

-->1.2:3
ans =

!   1.2   2.2 !

-->1.1 : 0.5:2.3
ans =

!   1.1   1.6   2.1 !

-->3:-1:0
ans =

!   3.   2.   1.   0. !

-->1:-1:3
ans =

[ ]

```

6.1.1 vecteur régulièrement espacé

L'opérateur « deux points » permet de discrétiser des intervalles. Scilab offre une alternative. Si a et b sont deux scalaires alors on peut obtenir les k points régulièrement espacés entre a et b . C'est la commande `linspace`

Syntaxe 6.1.2 :

`linspace (a,b,k)` engendre un vecteur ligne de longueur k de points régulièrement espacés entre a et b .

Attention il y a k points, y compris les extrémités, la longueur des intervalles est $\frac{b-a}{k-1}$

Aux erreurs d'arrondis près

`linspace (a,b, k)` est équivalent à

`a:(b-a)/(k-1):b`.

Ce n'est pas complètement équivalent comme le montre le test suivant

```

-->d1=0:1/10:1;

-->d2=linspace(0,1,11);

-->d1==d2
ans =

! T T T F T T F F T T T !

```

```
-->d1(4)-d2(4)
ans =

5.551115123E-17
```

Explication : on a demandé à Scilab de faire le test si les vecteurs **d1** et **d2** sont égaux. C'est la commande `=`. Il a répondu par vrai (T) ou faux (F) composante par composante. On voit que, par exemple, la quatrième composante diffère.

6.2 Extraction d'éléments d'une matrice

6.2.1 Indexation à deux indices

Syntaxe 6.2.1 :

L'élément à l'intersection de la i -ème ligne, j -ème colonne de la matrice A s'obtient par

A(i,j)

```
-->A(2,2)
ans =

5.
```

Syntaxe 6.2.2 :

*Si $[i_1, i_2, \dots, i_p]$ et $[j_1, j_2, \dots, j_k]$ sont deux listes d'indices, alors la commande **A([i1,i2,...,ip], [j1,j2,...,jk])***

Donne la sous-matrice constituée des éléments

à l'intersection des lignes $(i_1, \dots, i_p)$ et des colonnes $(j_1, \dots, j_k)$, dans l'ordre où sont écrits les indices

Exemple

```
-->A([1 3],[2 3])
ans =

!   1.   6.  !
!   9.   2.  !

-->A([2 1],[2 3])
ans =

!   5.   7.  !
!   1.   6.  !
```

Comme SCILAB est un langage interprété, si x est un vecteur $x(i)$ est interprété par SCILAB comme la i -ème composante de x .

6.2.2 Indexation à un seul indice

Scilab repère les éléments d'une matrice aussi avec un seul indice. Les éléments sont comptés, en descendant les colonnes et en parcourant les colonnes de la gauche vers la droite :

Si A est une matrice $m \times n$ les éléments de la première colonne sont numérotés de 1 à n . $a_{i,1}$ a pour numéro i . L'élément de la première ligne, deuxième colonne a pour indice unique $m + 1$.

Proposition 6.2.1 :

L'élément $a_{i,j}$ est indicé par $(j - 1)m + i$

Syntaxe 6.2.3 :

La commande `min(X)` donne la valeur minimum des coefficients de la matrice X .

Exercice 6.2.1 :

Que font les deux commandes suivantes ?

```
--> [m,n]=size(A);
--> A(1:m+1:min(m,n)^2)
```

6.2.3 Le dernier indice

Scilab offre un raccourci très pratique. Le symbole $\$$ dans une indexation donne le dernier indice possible. C'est ainsi que si A est une matrice $A(\$)$ est le terme de la dernière ligne, dernière colonne, $A(\$,:)$ est la dernière ligne de A et $A(:, \$-1)$ l'avant dernière colonne. Vous remarquerez que ces syntaxes marchent pour toute matrice, quelle que soit la dimension. Cela est très pratique en programmation.

6.3 « Columnisation » et remodelage

6.3.1 Columnisation

L'opérateur `:` est lié à l'indexation en un seul indice.

Syntaxe 6.3.1 :

Si A est une matrice, la syntaxe `x=A(:)` crée un vecteur colonne x qui s'obtient en empilant les colonnes de A du début à la fin.

Autrement dit $x((j - 1)m + i) = a_{i,j}$. Les éléments de x ont pour indexation l'indexation à un seul indice de A . Mais attention $A(:)$ est un vecteur colonne. L'opérateur `:` transforme une matrice en un vecteur colonne (de longueur mn)

6.3.2 Redimensionner une matrice

Soit une matrice A de taille (m, n) et une matrice B de taille (p, q) . Si A et B ont même nombre d'éléments, soit $mn = pq$ alors

Syntaxe 6.3.2 :

La syntaxe `B(:)=A` remplit la matrice B par les éléments de A , colonne par colonne.

Ceci est conforme au principe *variable=expression*. En effet il dit que chaque élément de B doit être remplacé par l'élément correspondant de A .

Exemple

```
-->A=testmatrix('magi',4)
A  =

!   16.    2.    3.    13. !
!   5.     11.   10.    8.  !
!   9.     7.    6.    12. !
!   4.     14.   15.    1.  !

-->B=zeros(2,8);

-->B(:)=A
B  =

!   16.    9.    2.    7.    3.    6.    13.    12. !
!    5.    4.   11.   14.   10.   15.    8.     1.  !
```

Il existe un raccourci et une commande SCILAB

Syntaxe 6.3.3 :

La commande SCILAB `matrix(A,m,n)` reformate la matrice A en une matrice $m \times n$ en remplissant les entrées de A colonne par colonne.

Dans l'exemple précédent

```
>matrix(A,2,8)
ans  =

!   16.    9.    2.    7.    3.    6.    13.    12. !
!    5.    4.   11.   14.   10.   15.    8.     1.  !
```

6.4 Création de matrices particulières

6.4.1 La matrice vide

La matrice vide qui est donnée par `[]` est très utile.

```
-->x=1:-1:2
x =

     []

-->size(x)
ans =

     0     0
```

Supprimer une ligne, une colonne

La commande

```
--> A(2,:)=[] ;
```

supprime la deuxième ligne de la matrice A

6.4.2 Matrice nulle

Définition 6.4.1 :

La matrice $m \times n$ dont tous les éléments sont nuls s'appelle la matrice nulle. Elle s'obtient par la syntaxe

```
zeros(m,n)
```

La commande `zeros(A)` donne une matrice de zéros de la même taille que A

6.4.3 Matrice de uns

Définition 6.4.2 :

La matrice $m \times n$ dont tous les éléments sont valent 1 s'appelle la matrice de uns. Elle s'obtient par la syntaxe

```
ones(m,n)
```

La commande `ones(A)` donne une matrice de uns de la même taille que A

6.4.4 Matrice identité

Définition 6.4.3 :

On appelle éléments diagonaux d'une matrice A de taille $m \times n$ les éléments de la forme a_{ii} . C'est à dire les éléments dont l'indice de ligne est égal à l'indice de colonne.

La matrice $m \times n$ dont tous les éléments sont diagonaux valent 1, les autres étant nuls s'appelle la matrice identité. Elle s'obtient par la syntaxe

`eye(m,n)`

La commande `eye(A)` donne la matrice identité de la même taille que A

6.4.5 Matrices Aléatoires

Définition 6.4.4 :

La matrice $m \times n$ dont tous les éléments sont tirés au hasard s'appelle une matrice aléatoire.

1. Si le tirage est choisi uniformément dans l'intervalle $[0, 1]$, Elle s'obtient par la syntaxe

`rand(m,n)`

2. Si le tirage suit une loi de Gauss (loi normale) de moyenne nulle et d'écart type 1 elle s'obtient par la syntaxe

`rand(m,n,'normal')`

La commande `rand(A)` donne une matrice aléatoire uniforme de la même taille que A , `rand(A,'normal')` une matrice aléatoire de la même taille que A par un tirage suivant la loi normale.

6.5 Création par concaténation

On peut construire de grande matrices à partir de plus petites. Supposons que l'on a construit 5 matrices. Par exemple soit la série de commandes

```
-->A=eye(2,3);
```

```
-->A1=zeros(2,2);
```

```
-->E=1:5;
```

```
-->C=ones(3,1);
```

```
-->Z=rand(5,6);
```

On veut construire la matrice par concaténation à partir de ces blocs élémentaires :

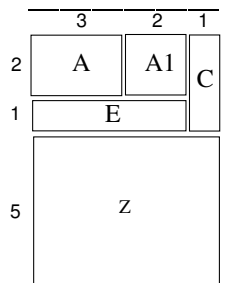


FIG. 6.1 – Concaténation de matrices

6.6 Extraction, suppression de vecteurs lignes, de vecteurs colonnes

6.6.1 Lignes et colonnes d'une matrice

Soit une matrice A de taille (m, n) , si les variables m et n ont été créées, par exemple `[m,n]=size(A)`, la commande

```
c1=A(1:m,1);
l2=A(2,1:n);
```

met en `c1` le premier vecteur colonne de A et en `l2` la deuxième ligne de A . SCILAB a un raccourci pour cette opération

Syntaxe 6.6.1 :

La syntaxe `A(:,j)` donne la j -ème colonne de A

La syntaxe `A(i,:)` donne la i -ème ligne de A

6.6.2 Suppression d'une ligne, d'une colonne

Syntaxe 6.6.2 :

La syntaxe `A(:,j)=[ ]` supprime la j -ème colonne de A

La syntaxe `A(i,:)=[ ]` supprime la i -ème ligne de A

6.7 Résumé des syntaxes du paragraphe

TAB. 6.1 – Commandes matricielles

<code>size(A)</code>	taille (m, n) de la matrice m lignes, n colonnes
<code>length(x)</code>	longueur n du vecteur $n \times 1$ ou $1 \times n$
<code>A(:,j)</code>	j -ème colonne
<code>A(i,:)</code>	i -ème ligne
<code>A([i1,...,ip],[j1,...,jk])</code>	matrice extraite de A , lignes $i_1, \dots, i_p$ et colonnes $j_1, \dots, j_k$
<code>A(:)</code>	grand vecteur colonne
<code>B(:)=A</code>	remplit B par les éléments de A colonnes après colonnes

TAB. 6.2 – Matrices particulières

<code>zeros (m,n)</code>	matrice de Zéros
<code>ones(m,n)</code>	matrice de 1
<code>eye(m,n)</code>	matrice identité
<code>rand(m,n)</code>	matrice dont les entrées sont uniformément distribuées
<code>rand(m,n, 'normal')</code>	matrice dont les entrées sont normalement distribuées

Chapitre 7

Opérations sur les matrices et les vecteurs

7.1 Addition, multiplication par un scalaire

Les matrices et les vecteurs forment ce que l'on appelle des espaces vectoriels. Autrement dit on peut les ajouter et les multiplier par un scalaire (i. e. réel ou complexe)

Définition 7.1.1 :

Soient $A = (a_{i,j})$ et $B = (b_{i,j})$ deux matrices de même taille. La matrice $A + B$ est la matrice dont le terme (i, j) est $a_{i,j} + b_{i,j}$. Autrement dit

$$(A + B)(i, j) = A(i, j) + B(i, j)$$

Soit λ un nombre scalaire, la matrice λA est la matrice dont le terme (i, j) est $\lambda a_{i,j}$. Autrement dit

$$(\lambda A)(i, j) = \lambda A(i, j)$$

SCILAB effectue sans problèmes ces opérations.

A =

```
!  8.    1.    6.  !
!  3.    5.    7.  !
!  4.    9.    2.  !
```

B =

```
!  1.    4.    7.  !
!  2.    5.    8.  !
```

```

!   3.   6.   9. !

-->A+B
ans  =

!   9.   5.   13. !
!   5.  10.  15. !
!   7.  15.  11. !

-->1.5*A
ans  =

!   12.   1.5   9.   !
!   4.5   7.5  10.5 !
!   6.   13.5   3.   !

```

7.1.1 Un raccourci bien pratique

On a vu que pour ajouter deux matrices celle-ci devaient avoir la même taille. Il y a cependant une exception. Si un des éléments de l'addition est un scalaire alors Scilab interprète.

Si A est une matrice si a est un scalaire alors

$A+a$

est la matrice A à laquelle on a ajouté a à chaque élément. C'est le raccourci de

$A+a * \text{ones}(A)$

7.2 Transposition

Définition 7.2.1 :

Etant donnée une matrice A de taille (m, n) on définit la matrice (n, m) transposée notée A^T obtenue en échangeant les lignes et les colonnes de A . Autrement dit la i -ème ligne de A^T est la i -ème colonne de A . La j -ème colonne de A^T est la j -ème ligne de A .

Cela est équivalent à : le terme (i, j) de A est $a_{j,i}$. Soit

$$A(i, j) = A^T(j, i)$$

Syntaxe 7.2.1 :

La syntaxe pour la transposition de A est $A.'$

La syntaxe A' donne la conjuguée de la transposée de A

Si la matrice est réelle alors ces syntaxes sont équivalentes.

Remarque 7.2.1 :

La transposée d'un vecteur ligne est un vecteur colonne et réciproquement

```
->B
```

```
B =
```

```
!  1.    4.    7.  !
!  2.    5.    8.  !
!  3.    6.    9.  !
```

```
-->B'
```

```
ans =
```

```
!  1.    2.    3.  !
!  4.    5.    6.  !
!  7.    8.    9.  !
```

```
-->(1:4)'
```

```
ans =
```

```
!  1.  !
!  2.  !
!  3.  !
!  4.  !
```

Mais attention !

```
x=x+%i*ones(4,1)
```

```
x =
```

```
!  1. + i  !
!  2. + i  !
!  3. + i  !
!  4. + i  !
```

```
-->x'
```

```
ans =
```

```
!  1. - i    2. - i    3. - i    4. - i  !
```

```
-->x.'
```

```
ans =
```

```
!  1. + i    2. + i    3. + i    4. + i  !
```

7.3 Produit de deux matrices

Scilab fait le produit de deux matrices sans problèmes. C'est l'opération `*`

```
A*B
```

Il faut évidemment que les matrices aient des dimensions compatibles. Sinon on obtient un message d'erreur.

```
!--error      10
inconsistent multiplication
```

Encore une fois il y a une exception, celui des matrices 1×1 , c'est à dire les scalaires. On a aussi une exception avec la matrice vide.

```
-->A* [ ]
ans =

[ ]
```

Le produit scalaire de deux vecteurs s'obtient de façon immédiate. Si x et y sont des vecteurs

```
--> x=x(:) ; y =y(:) ;

--> // on s'assure que ce sont des vecteurs colonnes

--> x.' * y
```

On peut toujours multiplier une matrice carrée ($n \times n$) par elle-même : `A*A`.

Il existe un raccourci qui est l'exposant `A^2`. La puissance n -ième d'une matrice se note `A^n`

7.4 Résumé des syntaxes du paragraphe

TAB. 7.1 – Opérations matricielles

+	addition
-	soustraction
*	multiplication
^	Exposant
'	transposition (matrice réelle) ou conjugué de la transposée (matrice complexe)
. '	Transposée (réelle ou complexe)

Chapitre 8

Opérations sur les tableaux de même taille

Les matrices étant des tableaux rectangulaires on peut définir des opérations sur des tableaux de même taille terme à terme. Si `op` est une opération arithmétique (addition, multiplication, division, exposant) alors l'opération sur les tableaux est `.op`

Définition 8.0.1 :

Si `op` est une opération arithmétique, si A et B sont deux matrices de même taille, la matrice $A \setminus .op \setminus B$ est la matrice dont le terme général est $a_{i,j} op b_{i,j}$

Il n'est pas nécessaire d'utiliser `.*` et `./` ces deux opérations étant déjà des opérations de tableau. Cela engendrerait d'ailleurs un message d'erreur. Par contre

si $C = A .* B$ alors $c_{i,j} = a_{i,j} * b_{i,j}$

si $C = A.^p$ alors $c_{i,j} = a_{i,j}^p$

si $C = p.^A$ alors $c_{i,j} = p^{a_{i,j}}$

si $C = A ./ B$ alors $c_{i,j} = a_{i,j} / b_{i,j}$

si $C = A \setminus B$ alors $c_{i,j} = a_{i,j} \setminus b_{i,j} = b_{i,j} / a_{i,j}$

Définition 8.0.2 :

Quand une fonction opère sur un tableau élément par élément on dit qu'elle est vectorisée. A peu près toutes les fonctions de SCILAB sont vectorisées.

Exemples : `log`, `exp`, `sqrt`, `sin`, `cos`, `tan`

Ces opérations sont très pratiques quand on veut vectoriser certains calculs.

8.1 Encore un danger

Malheureusement il faut faire attention dans l'utilisation du point. En effet le point peut signifier la séparation décimale (notre virgule) ou une opération `.op`.

44CHAPITRE 8. OPÉRATIONS SUR LES TABLEAUX DE MÊME TAILLE

C'est ainsi que si l'on veut obtenir les inverses des nombres de 1 à 5 (on choisit 5 pour des raisons d'affichage!) on va écrire

```
-->x=1:5;

-->x=1 ./ x
x  =

!    1.    0.5    0.333333    0.25    0.2 !
```

Vous remarquerez l'espace entre le 1 et le ./ . Si on l'avait oublié on obtiendrait

```
-->y=1./ x
y  =

!    0.6832416 !
!    0.3416208 !
!    0.2277472 !
!    0.1708104 !
!    0.1366483 !
```

On obtient un vecteur (1, 5) au lieu de (5, 1)! Que s'est-il passé? Tout simplement Scilab a considéré que le point était un point décimal de 1. et il effectué l'opération / au lieu de ./ . Pour voir ce que signifie / voir le chapitre 16 et la section 16.1.

On retrouve le même phénomène que dans la section 4.4.

Il faut faire attention à ce phénomène qui peut entraîner des effets indésirables en programmation.

Chapitre 9

Fonctions de manipulations de données

On a déjà rencontré, dans un exemple, de telles fonctions.

9.1 Fonctions élémentaires

Syntaxe 9.1.1 :

1. `max(A)` donne le plus grand élément de *A*.
Pour A , un vecteur ou une matrice réelle, `max(A)` est son plus grand élément. La commande `[m,k]=max(A)` donne la position du maximum dans A, en numérotation en double indice si A est une matrice, en simple indice si A est un vecteur.
2. `max(A,'r')` donne un vecteur ligne (d'où le 'r' pour row) , donc donne le plus grand élément de chaque colonne !
3. `max(A,'c')` donne le plus grand élément de chaque ligne.
Le deuxième argument optionnel peut être la chaîne de caractère 'r' ou 'c' . L'option 'r' permet d'obtenir un vecteur ligne.
4. `[val,ind]=max(A)` met le plus grand élément de A dans `val` et un double indice où cela est réalisé dans `ind`
5. On a l'analogie pour `min`
6. `abs(A)` matrice des valeur absolue des éléments de A, ou module des éléments de A
7. `sum(A)`, `sum(A,'r')`, `sum(A,'c')` somme des éléments
8. `prod(A)`, `prod(A,'r')`, `prod(A,'c')` produit des éléments
9. `mean(A)`, `mean(A,'r')`, `mean(A,'c')` moyenne des éléments
10. `st_deviation` est la commande pour l'écart type (standard deviation en anglais)

11. `cumsum(A)`, `cumsum(A,'r')`, `cumsum(A,'c')`
sommes cumulées des éléments
12. `cumprod(A)`, `cumprod(A,'r')`, `cumprod(A,'c')`
produit cumulés des éléments
13. `sort` range en ordre décroissant

`indexsum` `indexprod` `indexmean` `indexcumsum` `indexsort`

9.2 La commande `find`

Il s'agit là d'une commande très utile et puissante. C'est pour cette raison que nous l'avons mis à part.

Syntaxe 9.2.1 :

`I=find(x)`

Pour une matrice booléenne `x`, donne les indices `i` (numérotation à un seul indice) pour lesquels `x(i)` est vrai

Par exemple

```
--> I=find (A==max(A))
```

donne les indices pour lesquels les composantes de `A` sont maximales.

`find (A)` est un raccourci de `find(A~=0)`

Pour remplacer certaines valeurs d'une matrice par d'autres. Par exemple on veut remplacer les négatifs par des zéros :

```
I=find(A<0);
A(I)=0;
```

Chapitre 10

Matrice particulières

10.1 Extraire la diagonale principale

Syntaxe 10.1.1 :

la commande `diag(X)` si X est une matrice donne un vecteur colonne, qui est constitué des éléments diagonaux de X

Si X est un vecteur de longueur n , `diag(X)` est la matrice diagonale $n \times n$ dont la diagonale est constituée des éléments de X

Si X est une matrice `diag(diag(X))` est la matrice diagonale extraite de X

Exercice 10.1.1 :

Que vaut `A.*eye(A)`

10.2 Autres diagonales

Par convention la diagonale principale de A , c'est à dire les éléments de A qui ont même indice de ligne et de colonne, $j - i = 0$ est la diagonale 0.

La sur-diagonale, i.e. les éléments d'indice (i, j) tels que $j - i = 1$ est la diagonale 1.

Syntaxe 10.2.1 :

Si A est une matrice alors `diag(A,k)` est le vecteur colonne des éléments de A tels que leurs indices vérifient $j - i = k$

Si x est un vecteur de longueur n , alors `diag(x,k)` est la matrice $(n + |k|)^2$ dont la diagonale k est constituée des éléments de x

$k > 0$ donne des diagonales supérieures, $k < 0$ des diagonales inférieures.

10.3 Triangulaire extraite

Syntaxe 10.3.1 :

Partie triangulaire inférieure d'une matrice. `tril(x,k)` est composée des termes sous une diagonale : $k > 0$ (diagonale supérieure) et $k < 0$ (diagonale sous la diagonale principale).

Syntaxe 10.3.2 :

Partie triangulaire supérieure d'une matrice. `triu(x,k)` est composée des termes au dessus d'une diagonale : $k > 0$ (diagonale supérieure) et $k < 0$ (diagonale sous la diagonale principale).

10.4 Matrice de Toeplitz

Une matrice de Toeplitz est une matrice dont les composantes des diagonales sont constantes. Il suffit donc pour connaître une matrice de Toeplitz (m, n) de connaître la première ligne (qui va donner les diagonales de 0 à n et la première colonne qui va donner les diagonales de 0 à $-m$).

Syntaxe 10.4.1 :

`A=toeplitz(c, [r])`

Le deuxième terme `r` est optionnel

`A=toeplitz(c,r)` donne la matrice de Toeplitz de première colonne le vecteur `c` et de vecteur ligne `r`. Il faut évidemment `r(1)==c(1)`

La commande `A=toeplitz(v)` donne la matrice de Toeplitz symétrique.

```
-->A=toeplitz([1,2,0 0 0], [1,3 0 0 0])
```

```
A =
```

```
!   1.   3.   0.   0.   0. !
!   2.   1.   3.   0.   0. !
!   0.   2.   1.   3.   0. !
!   0.   0.   2.   1.   3. !
!   0.   0.   0.   2.   1. !
```

```
-->B=toeplitz(1:5)
```

```
B =
```

```
!   1.   2.   3.   4.   5. !
!   2.   1.   2.   3.   4. !
!   3.   2.   1.   2.   3. !
!   4.   3.   2.   1.   2. !
!   5.   4.   3.   2.   1. !
```

10.5 Matrices test

La commande

```
A=testmatrix ("nom", n)
```

engendre la matrice test carrée, correspondant à « nom ».

On dispose de 'magi' qui engendre une matrice magique, de frk matrice de Franck et de hilb qui engendre l'inverse de la matrice de Hilbert.

La matrice de Hilbert est la matrice H caractérisée par

$$H(i, j) = \frac{1}{i + j - 1}$$

La matrice de Franck est la matrice de Toeplitz de première ligne $\mathbf{n}:-1:1$ et de première colonne $\mathbf{n}, \mathbf{n}-1, 0, \dots, 0$

Exercice 10.5.1 :

Engendrer la matrice de Hilbert. On suppose que n est donné dans Scilab

```
-->a=1:n; b=a';
-->A=a(ones(a) , :);
-->B=b(: , ones(b));
-->H=A+B-1;
-->H=1 ./ H;
```

Autre solution

```
--> H1=toeplitz(n:-1:1);
-->H1=H1($:-1:1,:);
-->H1=1 ./ H1;
```


Chapitre 11

Produit de Kronecker

Le produit de Kronecker de deux matrices est un outil très puissant. Il est toujours défini

Définition 11.0.1 :

Soit $A = a_{i,j}$ une matrice $m \times n$ et $B = b_{j,k}$ une matrice $p \times q$. On définit le produit $A \otimes B$ encore noté `kron (A, B)` comme la matrice bloc de taille $mp \times nq$. La matrice est constituée sur le modèle de A . Elle contient m lignes et n colonnes de blocs, dont la taille est chacun de la taille de B , $p \times q$. Le bloc (i, j) est donné par

$$a_{i,j}B$$

La matrice B constitue la brique de base et la matrice A fourni le plan du pavage. Avec `kron` on peut constituer des motifs facilement.

Par exemple soit A une matrice, et on veut constituer la matrice bloc

$$M = \begin{bmatrix} A & A & A \\ A & A & A \end{bmatrix}$$

M s'obtient facilement par la commande `kron(ones(2,3),A)`

Chapitre 12

Quelques astuces utiles

12.1 Une matrice dont toutes les colonnes sont égales

Soit x un vecteur colonne : $x=x(:)$. On veut la matrice $A=[x, x, \dots, x]$ constituée de p colonnes identiques à x .

```
--> A=x*ones(1,p)
```

Cette solution utilise la multiplication des matrices. Il y a un calcul.

```
--> A=x(: , ones(1,p))
```

aucun calcul c'est de l'affectation mémoire. Cette astuce très utile est connue sous le nom de Tony's trick.

```
--> A=kron(ones(1,p),x)
```

12.2 Une matrice dont toutes les lignes sont égales

On a un vecteur ligne $x=x(:)$; $x=x.'$;

On veut la matrice $A=[x ; x ; \dots ; x]$

constituée de p lignes identiques à x .

Exercice !

12.3 Changer la diagonale principale d'une matrice

Soit A une matrice (m, n) . On a un vecteur x de taille $\min(m, n)$

```
[m,n]=size(A); p=min(m,n);
A(1:m+1:p^2)=x;
```

12.4 Matrice de Vandermonde

Définition 12.4.1 (Vandermonde) :

Si x est un vecteur colonne de longueur n , la matrice de Vandermonde $n \times p$ est la matrice

```
Vnp=[ones(x), x, x.^2, x.^3, . . . , x.^(p-1)]
```

On l'obtient par les commandes

```
x=x(:); n=length(x);
A=x(:, ones(1,p));
B=1:p; B=B(ones(1,p) , :);
Vnp=A.^B;
```

Chapitre 13

Exercices

Ces exercices servent à vérifier que vous avez bien compris ce qui précède

13.1

1. nombres de 1 à 3 avec un pas de 0.1
2. entiers de 1 à 10 par ordre décroissant
3. cubes des entiers de 1 à 10
4. somme pour $n = 1 : 10$ des entiers de 1 à n
5. nombres $(-1)^n n^2$ pour $n = 1 : 10$
6. dix 0 suivi de dix 1
7. trois 0 suivis de trois 1, suivis de trois 2, ..., suivis de trois 9
8. un 1, suivi de deux 2, ..., suivi de neuf 9.
9. un 1 suivi de un 0, suivi de 2 suivi de deux zéros, suivi de 3 , ...suivi de huit 0, suivi de 9

13.2

1. Ranger la matrice A par ordre croissant.
2. remplacer la diagonale d'une matrice carrée par des zéros.
3. Créer une matrice tridiagonale carrée, a sur la diagonale, b en dessous, c au dessus
4. Faire tourner de 90 degrés dans le sens horaire une matrice carrée.
5. Faire une symétrie sur une matrice carrée par rapport à l'antidiagonale

Chapitre 14

Systèmes linéaires

Un des problèmes les plus fréquent en calcul scientifique est la résolution d'un système linéaire. Un tel système s'écrit

$$Ax = b$$

où A est une matrice carrée $n \times n$ et b est un vecteur colonne donné de $\mathbb{R}^n$ et x est un vecteur de $\mathbb{R}^n$ inconnu, que l'on cherche.

L'ajustement de données est une source de problèmes linéaires.

14.1 Ajustement de données

On considère un système boîte noire : on rentre des données $(t_1, \dots, t_n)$ et il en résulte des sorties correspondantes $(y_1, \dots, y_n)$.

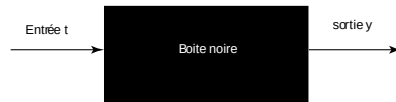


FIG. 14.1 – Une boîte noire

Par exemple les t_i peuvent être des temps et les y_i des mesures faites au temps correspondants. Les t_i peuvent être des températures et les y_i une intensité de courant etc etc ...

On a des raisons de penser que le phénomène observé peut se modéliser par une loi

$$y = a + bt + ct^2$$

On cherche alors paramètres (a, b, c) tels que pour tout indice i on ait

$$y_i = a + bt_i + ct_i^2$$

On a n équations. Ces équations peuvent se condenser en

$$Ax = b$$

où

$$x = \begin{bmatrix} a \\ b \\ c \end{bmatrix}$$

$$b = [y_1, \dots, y_n]^T$$

et

$$A = \begin{bmatrix} 1 & t_1 & t_1^2 \\ 1 & t_2 & t_2^2 \\ \vdots & \vdots & \vdots \\ 1 & t_n & t_n^2 \end{bmatrix}$$

14.2 Résolution d'un système linéaire

Beaucoup de méthodes ont été proposées pour résoudre les systèmes linéaires. Scilab utilise les algorithmes classiques et les plus adaptés pour résoudre un système linéaire.

Dans Scilab il existe une commande unique

`x=A \ b`

x est une solution de $A \star x = b$

1. Si A est carrée et inversible x est uniquement défini et est calculé avec l'algorithme le plus approprié
2. si A n'est pas carrée alors x est une solution aux moindres carrés. Autrement dit x minimise la quantité $\|A \star x - b\|_2$.
3. Si A est surjective (ie les colonnes sont linéairement indépendantes) alors la solution x est unique et Scilab la calcule numériquement
4. Si A n'est pas surjective et `A \ b` est une solution

Chapitre 15

L'arithmétique de l'ordinateur

L'algorithme de Gauss que l'on a proposé ne marche pas à tous les coups. Tout d'abord il peut arriver qu'un des pivots soit nul. Mais si le pivot est très petit il peut arriver des problèmes dus à la représentations des nombres dans une machine. Dans ce paragraphe on va revenir sur l'arithmétique IEEE et l'arithmétique de l'ordinateur (ou de votre calculatrice !)

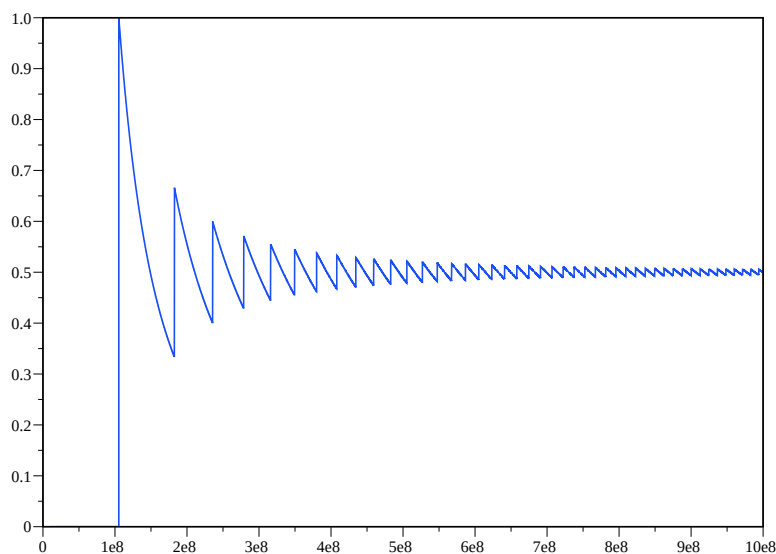
15.1 Les erreurs d'arrondis

15.1.1 Un premier exemple

On va essayer de représenter la fonction $\frac{1-\cos(x)}{x^2}$ sur l'intervalle $[10^{-9}, 10^{-7}]$.

```
-->t=1e-9:1e-11:1e-7;  
-->y=(1-cos(t))./(t.^2);  
-->plot2d(t,y)
```

On obtient la courbe

FIG. 15.1 – La fonction $\frac{1-\cos(x)}{x^2}$!!

Malheureusement on sait que $\frac{1-\cos(x)}{x^2} \leq \frac{1}{2}$

Exercice 15.1.1 :

Montrer que si $x \geq 0$ on a

$$1 - \cos(x) - \frac{1}{2}x^2 \leq 0$$

Si on calcule avec Scilab

```
-->a=sqrt(%eps/2);
```

```
-->(1-cos(a))/a^2
```

```
ans =
```

```
1.
```

Le résultat est manifestement faux. Que s'est-il passé ?

15.1.2 Deuxième exemple

On va considérer les polynômes. Un polynôme en X est une expression de la forme

$$P(X) = a_0 + a_1X + a_2X^2 + \cdots a_nX^n$$

a_i s'appelle le coefficient du terme de degré i et n le degré du polynôme.

Un réel ou complexe x_0 tel que $P(x_0) = 0$ s'appelle une racine du polynôme.

Un polynôme de degré n admet n racines réelles ou complexes. Si on se donne les racines d'un polynôme $\{x_0, x_1, \dots, x_n\}$ alors le polynôme peut s'écrire

$$P(X) = (X - x_0)(X - x_1) \cdots (X - x_n)$$

Par exemple

$$(X - 1)^2 = 1 + 2X + X^2$$

Les commandes Scilab sont

```
P=poly( a, 'X', 'coef')
```

Ce qui donne un polynôme de variable X dont les coefficients sont donnés par les éléments du vecteur $a = (a_0, a_1, \dots, a_n)$

```
-->P=poly([1 2 3], "X", "coef")
```

```
ans =
```

$$1 + 2X + 3X^2$$

La commande

```
Q=poly(a, "X", "roots")
```

donne le polynôme $(X - a_0) \cdots (X - a_n)$

```
-->Q=poly([1 2 3], "X", "roots")
```

```
Q =
```

$$-6 + 11X - 6X^2 + X^3$$

Si P est un polynôme la commande

```
roots(P)
```

donne les racines de P

```
-->roots(P)
```

```
ans =
```

```
! - 0.3333333 + 0.4714045i !
```

```
! - 0.3333333 - 0.4714045i !
```

```
-->roots(Q)
ans =

!   1. !
!   2. !
!   3. !
```

Enfin si on veut la valeur d'un polynôme en un point y_0 la commande est

```
horner(P,y0)
```

```
-->horner(P,1)
ans =

6.

-->horner(Q,1)
ans =

0.
```

La fonction horner est vectorisée. On va considérer le polynôme

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

Et l'on va dessiner sa courbe près de la valeur 1 avec différents agrandissements :

```
-->P=poly(ones(1,6),"X")
P =

          2      3      4      5      6
1 - 6X + 15X - 20X + 15X - 6X + X

-->t=0.95:1e-5:1.05;

-->xset("window",1)

-->plot2d(t,horner(P,t))

-->t=0.995:1e-5:1.005;

-->xset("window",2)

-->plot2d(t,horner(P,t))
```

```
-->t=0.998:1e-5:1.002;

-->xset("window",3)

-->plot2d(t,horner(P,t))

-->xset("window",4)

plot2d(t, (t-1).^6)
```

On obtient les quatre courbes suivantes

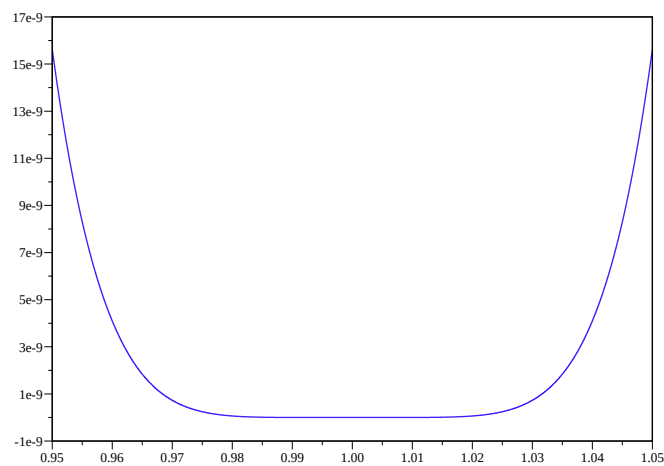
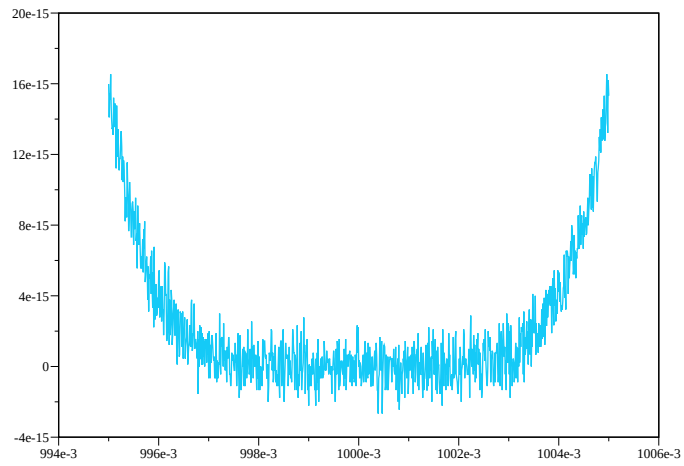
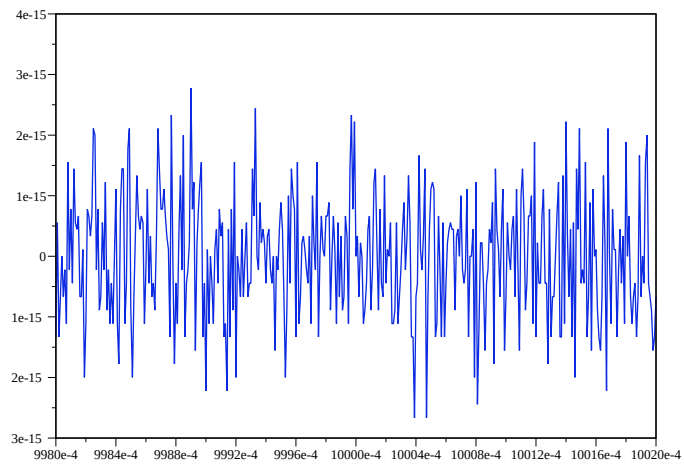
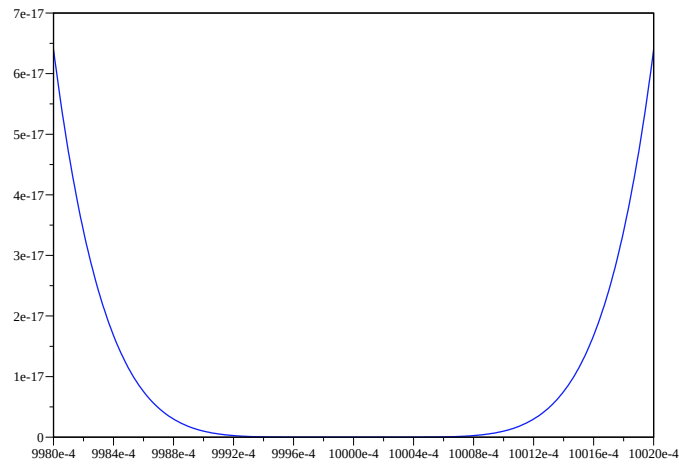


FIG. 15.2 – premier agrandissement $P=1-6X+15X^2-20X^3+15X^4-6X^5+X^6$

FIG. 15.3 – deuxième agrandissement $P=1-6X+15X^2-20X^3+15X^4-6X^5+X^6$ FIG. 15.4 – troisième agrandissement $P=1-6X+15X^2-20X^3+15X^4-6X^5+X^6$

FIG. 15.5 – même chose avec une autre représentation de $P = (X - 1)^6$

15.2 Les nombres dans l'ordinateur : Les nombres flottants

Les nombres réels dans une machine sont représentés comme des nombres flottants

$$x = \pm 0.b_1b_2 \dots b_t 10^e$$

La partie $0.b_1b_2 \dots b_t$ s'appelle la mantisse et e l'exposant.

Le nombre t s'appelle le nombre de chiffres significatifs.

Scilab utilise l'arithmétique IEEE pour laquelle $-308 \leq e \leq 308$. L'arithmétique IEEE est une norme publiée en 1985 par « Institute of Electronics and Electrical Engineers ». C'est le résultat de plusieurs années de travail d'un groupe de réflexion. Pratiquement tous les ordinateurs utilisent maintenant le standard IEEE.

Pour que Scilab utilise cette arithmétique il faut lui dire de l'utiliser :

```
ieee(2)
```

On teste ensuite le plus grand nombre possible

```
--1.797e308
```

```
ans =
```

```
1.797+308
```

```
-->1.798e308
```

```
ans =
```

```
Inf
```

au-delà de $1.79 \cdot 10^{308}$ Scilab répond `Inf` ce qui veut dire ∞ . On dit qu'on a un overflow.

```
-->a=1.7976E308
```

```
a =
```

```
1.798+308
```

```
-->1.1*a
```

```
ans =
```

```
Inf
```

```
-->0.9*a
```

```
ans =
```

```
1.618+308
```

Le plus petit nombre est de l'ordre de 10^{-324}

```
-->b=2.251e-308
```

```
b =
```

```
2.251-308
```

```
-->b*%eps
```

```
ans =
```

```
4.941-324
```

```
-->b*%eps/2
```

```
ans =
```

```
0.
```

La précision est défini par l' ϵ -machine. C'est la distance entre 1 et le plus proche nombre flottant qui lui est supérieur.

Il vaut

```
-->%eps
```

```
%eps =
```

2.220E-16

La machine arrondi tout résultat vers le plus proche nombre flottant. L'unité d'arrondi près de 1 est donc la moitié de l' ε -machine :

```
-->1+0.5*%eps==1
ans =
```

T

```
-->1+0.6*%eps==1
ans =
```

F

```
-->1+0.6*%eps==1+%eps
ans =
```

T

15.3 Arithmétique IEEE

L'arithmétique IEEE obéit aux principes suivants

TAB. 15.1 – Arithmétique IEEE

Type	Exemple	Résultat
Opération invalide	$0/0, 0 \times \infty, \infty/\infty$	Nan (not a number)
Overflow	$1/0$	$\pm\infty$

Une opération qui n'est mathématiquement définie produit une réponse Nan, c'est à dire « not a number ». La variable *Nan* est introduite par `%nan`

Une fois engendrée *Nan* se propage par toutes les opérations

```
-->a=%nan;
```

```
-->a/a
ans =
```

Nan

```

-->a*a
ans =

    Nan

-->b=[%nan,%inf,2]
b =

!   Nan   Inf   2.   !

-->min(b)
ans =

    2.

-->b=[b -%inf]
b =

!   Nan   Inf   2.  -Inf !

-->min(b)
ans =

  -Inf

```

15.4 Explication des cas précédents

Quand on introduit un nombre ou que l'on fait un calcul la machine arrondi : c'est à dire qu'elle utilise le nombre flottant le plus proche. (avec un processus pour choisir quand on est au milieu !)

c'est ainsi que si on calcule $1 - \cos(x)$ pour $x = \sqrt{\frac{\epsilon ps}{2}}$ on obtient

```

-->b=sqrt(%eps/2)
b =

    1.054E-08

-->1-cos(b)
ans =

    1.110E-16

```

```
-->b^2
ans  =

1.110E-16
```

Comme x est très petit $\cos(x)$ est très proche de 1. Il y a une simplification catastrophique.

On aurait pu éviter cela en calculant

$$1 - \cos(x) = 2 \sin^2 \frac{x}{2}$$

```
-->2*sin(b/2)^2
ans  =

5.551E-17
```


Chapitre 16

Résolution aux moindres carrés

On considère une équation linéaire $Ax = b$ où A est une matrice $m \times n$. On appelle norme d'un vecteur x la racine carrée du scalaire

$$x^T * x = \sum_{i=1}^n x_i^2$$

On note $\|x\|_2 = \sqrt{x^T * x}$

En général pour une équation linéaire $Ax = b$ il peut n'y avoir soit aucune solution, soit une seule solution ou enfin une infinité de solutions. Si $m > n$ très souvent il n'y a pas de solution.

S'il y a une solution alors $Ax - b = 0$ et donc $\|Ax - b\|_2^2 = 0$. Chercher à minimiser cette quantité $\|Ax - b\|_2^2$ s'appelle la solution aux moindres carrés de l'équation $Ax = b$

Théorème 16.0.1 :

Il existe une solution unique qui minimise $\|Ax - b\|_2^2$ et de norme minimale.

Cette solution est donnée numériquement en Scilab par

$$x=A \backslash b$$

Syntaxe 16.0.1 :

$x=A \backslash b$ l'expression $\backslash$ s'appelle *backslash*. C'est la division matricielle à gauche. C'est la solution aux moindres carrés de

$$A * x = b$$

Si A est inversible alors $x = A^{-1}b$. Autrement dit on multiplie à gauche par l'inverse de A , multiplier par un inverse dans $\mathbb{R}$ s'appelle diviser. D'où le nom de division à gauche.

On a de même la définition de la division matricielle à droite

Syntaxe 16.0.2 :

$x = A / b$ L'expression / s'appelle slash.

A / b est la solution aux moindres carrés de $x * b = A$.

Pour que ceci est un sens, si b est une matrice $p \times q$, A doit être de taille $n \times q$ et x sera un vecteur $n \times p$. On ne peut donc diviser à droite que si les matrices ont le même nombre de colonnes.

Il y a cependant toujours une exception quand A est un scalaire. Dans ce cas où A est un scalaire, A/b est la solution de $b * x = A$!! Ceci a une conséquence significative que l'on a déjà vu.

16.1 Un danger

Il faut faire attention aux commandes

```
-->a / b
```

et

```
-->a./b
```

```
-->1./b
```

En effet on obtient

```
-->a=1
```

```
a =
```

```
1.
```

```
-->a./[1 2 3]
```

```
ans =
```

```
! 1.    0.5    0.3333333 !
```

```
-->1./[1 2 3]
```

```
ans =
```

```
! 0.0714286 !
```

```
! 0.1428571 !
```

```
! 0.2142857 !
```

Le point avec le 1 est interprété comme `1.`, on a alors l'opérateur `/` et non `./`
Le mieux est d'utiliser des parenthèses pour éviter les confusions

```

-->x=1/1:6
x =

!   1.   2.   3.   4.   5.   6. !

-->1./(1:6)
ans =

!   0.0109890 !
!   0.0219780 !
!   0.0329670 !
!   0.0439560 !
!   0.0549451 !
!   0.0659341 !

-->(1) ./ (1:6)
ans =

!   1.   0.5   0.3333333   0.25   0.2   0.1666667 !

```

16.2 Application à l'ajustement de données

$y(t)$ est une fonction que l'on veut approximer. On suppose que l'on dispose de m observations

$$Y = [y_1; y_2; \dots; y_m]$$

mesurées pour des valeurs spécifiques de t

$$T = [t_1; t_2; \dots; t_m]$$

L'idée est de modéliser $y(t)$ à l'aide de n fonctions de $f_1, f_2 \dots f_n$.

$$y(t) \approx b_1 f_1(t) + b_2 f_2(t) + \dots + b_n f_n(t)$$

Le problème est de déterminer les meilleures valeurs des coefficients b_i compte-tenu des mesures. Si A est la matrice

$$a_{i,j} = f_j(t_i)$$

Les b_i les meilleurs au sens des moindres carrés seront ceux qui minimiseront

$$\|Ab - Y\|_2^2$$

La solution est donnée par

$$A \backslash Y$$

16.3 Un exemple

On a 25 observations pour des valeurs de t de 1 à 25. On pose $t = 1 : 25$. Les données sont présentées ligne après ligne :

```
-->y
y =
```

!	5.0291	6.5099	5.3666	4.1272	4.2948	!
!	6.2161	12.514	10.0502	9.1614	7.5677	!
!	7.292	10.0357	11.0708	13.4045	12.8415	!
!	11.9666	11.0765	11.7774	14.5701	17.044	!
!	17.0398	15.9069	15.485	15.5112	17.6572	!

Il faut donc transformer cela en vecteur colonne z qui sera le vecteur des observations

```
-->z=y';
-->z=z(:);
-->t=1:25;t=t(:);
```

On essaye le modèle $y(t) \approx \beta_1 + \beta_2 t$

```
-->A=[ones(t),t];
-->beta=A\z
beta =
```

!	4.022592	!
!	0.5321582	!

```
-->xset("window",1)
-->xbasc()
-->plot2d(t,z,-1)
-->plot2d(t,beta(1)+beta(2)*t)
```

On obtient le graphe suivant.

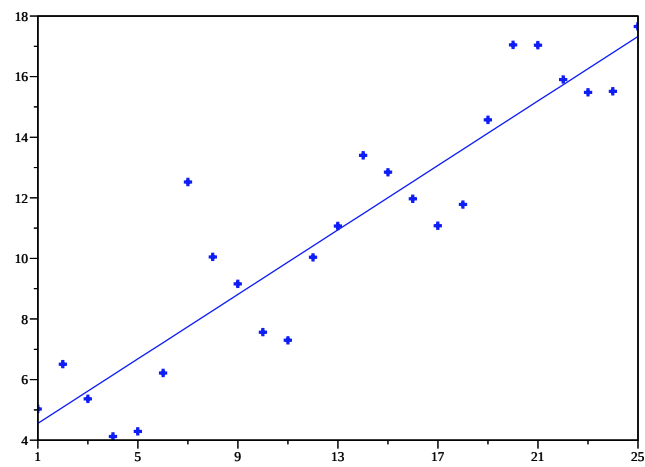


FIG. 16.1 – premier modèle

On dessine aussi les résidus

```
--xset("window",2)

-->xbasc()

-->r=z-beta(1)-beta(2)*t;

-->plot2d(t,r,-1)

-->[val,ind]=max(r)
I  =

    7.
v  =

    4.7663009
```

Ce qui donne la figure.

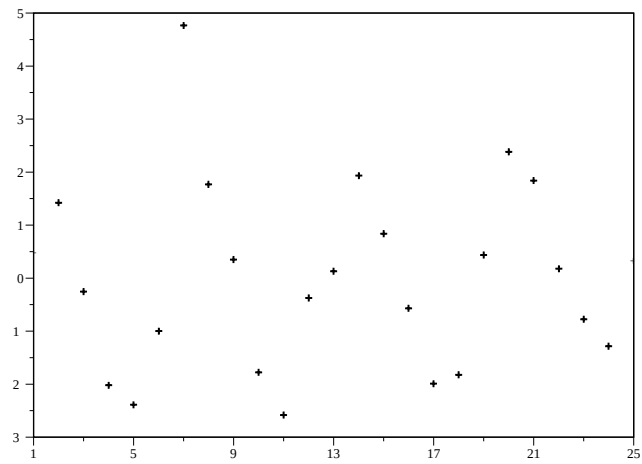


FIG. 16.2 – Les résidus : une mesure aberrante

On constate qu'il y a un point excentré. On considère qu'il s'agit d'une anomalie. On l'élimine et l'on recommence l'estimation des paramètres

```
-->z(7)=[ ];t(7)=[ ];

-->A1=[ones(t),t];
-->beta2=A1\z
    beta2 =

    3.5113551
    0.5557537

-->subplot(2,1,1)

-->plot2d(t,z,-1)

-->plot2d(t,beta1(1)+beta1(2)*t)

-->subplot(2,1,2)

-->r=z-beta1(1)-beta1(2)*t;

-->plot2d(t,r)
```

On dessine en même temps l'approximation et les résidus

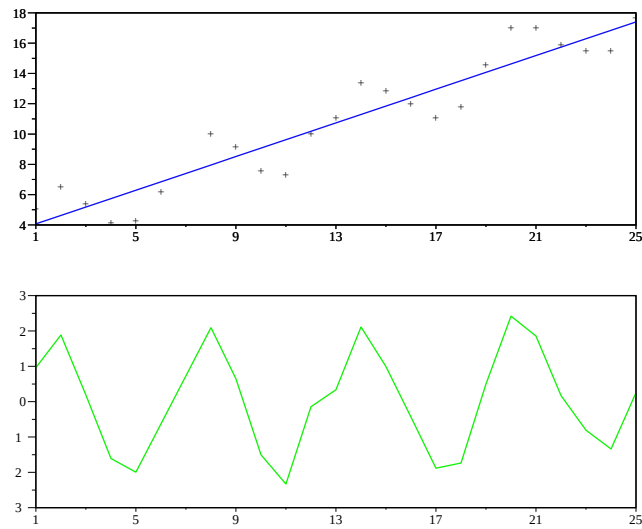


FIG. 16.3 – deuxième approximation, sans la mesure aberrante

On constate que les résidus ont une forme périodique. On va essayer le modèle

$$y(t) \approx \beta_1 + \beta_2 t + \beta_3 \sin(t)$$

On identifie les paramètres (toujours sans le point aberrant)

```
A2=[ones(t),t,sin(t)];
```

```
-->beta3=A2\z
```

```
beta3 =
```

```
! 3.1646828 !
```

```
! 0.586349 !
```

```
! 1.9705571 !
```

On va maintenant rajouter le point aberrant puis tracer les courbes (on discrétise un peu plus l'intervalle pour la courbe du modèle) :

```
-->z(7)=val;t(7)=7;
```

```
-xbasc()
```

```
-T=1:0.01:25;
```

```
-->plot2d(t,z,-1)
```

```
-->plot2d(T,beta3(1)+beta3(2)*T+beta3(3)*sin(T))
```

On obtient

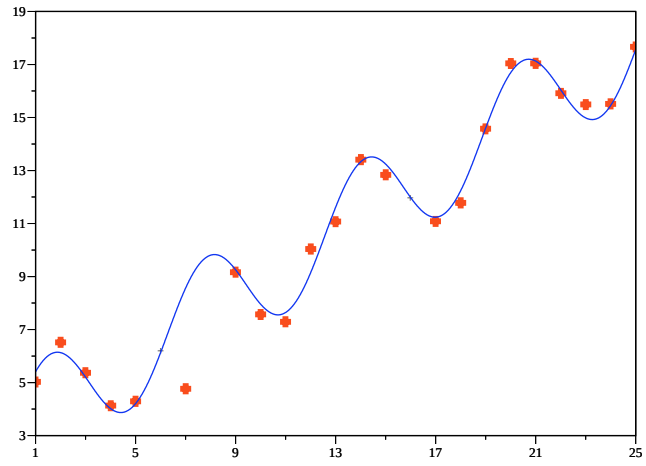


FIG. 16.4 – Modèle final

Chapitre 17

Gestion de fichiers

Nous venons de voir un exemple académique d'analyse de données. Cet exemple ne traitait pas un grand nombre de données. Dans la réalité les données peuvent être nombreuses. Dans ce cas pour les analyser avec Scilab il est hors de question de les rentrer à la main. En effet ce serait trop long et aussi source d'erreurs de saisie. Les données se trouvent en général dans des fichiers. Une partie du travail va être de les transformer en des éléments utilisables par Scilab, donc en faire des matrices réelles ou complexes.

Les deux types de fichiers que vous rencontrerez le plus souvent sont les fichiers Excel et les fichiers ASCII. Les fichiers binaires ne sont lisibles que par Scilab. Un fichier binaire MATLAB ne pourra être lu par Scilab. Il faudra le transformer en fichier ASCII.

17.1 Chaînes de caractères

Les chaînes de caractères (strings) Scilab sont délimitées par les caractères apostrophe ' ou guillemets anglo-saxons " (qui sont équivalents). Pour insérer une apostrophe ou des guillemets, il faut les faire précéder d'un délimiteur (à nouveau ' ou "). Les opérations de base sur les chaînes de caractères sont la concaténation, notée par l'opérateur + , et la fonction `length` qui renvoie le nombre de caractères contenus dans une chaîne.

Syntaxe 17.1.1 :

La commande `strcat` (v) concatène la vecteur de string v en une seule chaîne.

17.2 Récupérer un fichier ASCII

On utilise la commande `mgetl`

Syntaxe 17.2.1 :

$A = mgetl('nomdefichier', [m])$

m est un entier optionnel. Scilab lit exactement *m* lignes du fichier ASCII. Si *m* est omis, Scilab lit toutes les lignes. La matrice *A* est une matrice de « string ».

Une fois que l'on a un fichier de string et que les caractères sont tous des chiffres on peut le transformer en un fichier de nombre réels.

Syntaxe 17.2.2 :

`A=evstr(B)` retourne l'évaluation de la matrice de strings

```
-->A=["a","b"; "1","2"]
```

```
A =
```

```
!a  b  !
```

```
!      !
```

```
!1  2  !
```

```
-->a=3;b=2;
```

```
-->M=evstr(A)
```

```
M =
```

```
!   3.   2.  !
```

```
!   1.   2.  !
```

Attention s'il reste des caractères autres que des chiffres vous obtiendrez un message d'erreur.

17.3 Un exemple

Dans le fichier courant on a deux fichiers textes (ASCII) :

`pop.dat` et `census.dat` qui correspondent respectivement aux dates et aux valeurs des recensement de la population des Etats-Unis. On sait que on dispose de 21 données.

```
-->cdate=mgetl('pop.dat');
```

```
-->pop=mgetl('census.dat',21);
```

```
-->typeof(pop)
```

```
ans =
```

```
string
```

```
--> size(pop)
```

```

ans =

! 21  1!

-->A=evstr(pop);

-->B=evstr(cdate);

```

La commande `typeof` vous indique quelle est la nature de la variable. C'est à dire si on a affaire soit à une matrice de nombre, auquel cas Scilab répond `constant`, soit à une matrice de « string », auquel cas Scilab répond `string`, soit une matrice de Booléen (réponse `boolean`) soit à une matrice polynômiale (réponse `polynomial`)

17.4 Fichiers excel

On a créé un fichier excel de nombres, sauvegardés sous format `.txt`, ce qui signifie que l'on utilise le séparateur `tabulation`. Attention Il faut remplacer les `,` par des points dans le fichier Excel

```

--A=excel2sci('exceltest.txt','txt');

-A=evstr(A)                                ;;
A =

!  1.    55.    !
!  2.    23.    !
!  3.    44.    !
!  4.   122.    !
!  5.   29.666667 !
!  6.   24.166667 !
!  7.   18.666667 !
!  8.    72.5    !
!  9.   13.166667 !
! 10.    7.666667 !
! 11.    2.166667 !
! 12.    23.    !

-->typeof(A)
ans =

constant

```

Et voilà! on a une matrice 12×2 . On aurait pu ne pas mettre le deuxième terme de la commande `'txt'`

On peut aussi récupérer des fichiers avec le séparateur ; qui se nomme `csv`

```
A=excel2sci('exceltest.csv','csv'); ;
```

```
-->A=evstr(A)
```

```
A =
```

!	1.	!
!	55.	!
!	2.	!
!	23.	!
!	3.	!
!	44.	!
!	4.	!
!	122.	!
!	5.	!
!	29.666667	!
!	6.	!
!	24.166667	!
!	7.	!
!	18.666667	!
!	8.	!
!	72.5	!
!	9.	!
!	13.166667	!
!	10.	!
!	7.666667	!
!	11.	!
!	2.166667	!
!	12.	!

```
[More (y or n ) ?]
```

On voit qu'ici les données sont alternées. C'est dû au séparateur. Les changements de colonnes ne sont pas observés avec ce choix de séparateur.

Chapitre 18

Etude d'un cas réel

18.1 Phénomènes périodiques

Beaucoup de phénomènes peuvent être modélisés par des sommes de fonctions trigonométriques

$$y(t) = \sum_{k=1}^n A_k \sin\left(\frac{2k\pi}{T}(t + \phi_k)\right)$$

Les nombres A_k , T et ϕ_k s'appellent respectivement l'amplitude, la période et la phase de la k -ème harmonique. Si l'on connaît la période T , le problème de déterminer les paramètres A_k et ϕ_k devient un problème linéaire. En effet, à l'aide des formules trigonométriques on a

$$A_k \sin\left(\frac{2k\pi}{T}(t + \phi_k)\right) = a_k \cos\left(\frac{2k\pi}{T}t\right) + b_k \sin\left(\frac{2k\pi}{T}t\right)$$

Et les coefficients a_k et b_k sont donnés par

$$a_k = A_k \sin\left(\frac{2\pi k}{T}\phi_k\right)$$

et

$$b_k = A_k \cos\left(\frac{2\pi k}{T}\phi_k\right)$$

Quand on a estimé a_k et b_k on obtient

$$A_k = \sqrt{a_k^2 + b_k^2}$$

et

$$\phi_k = \frac{T}{2\pi k} \arcsin\left(\frac{a_k}{A_k}\right)$$

On va traiter un exemple tiré de la réalité. Les concentrations atmosphérique en CO₂ sont mesurées à l'observatoire de Mauna Loa (Hawaï) depuis 1958, mois par

mois. On peut les trouver sur les sites consacrés à l'observation atmosphérique. Les valeurs sont exprimées en "ppm" et reportées dans le 2003A SIO, échelle de fraction manométrique molaire. Les valeurs sont ajustées au 15 de chaque mois. Les valeurs manquantes sont notées -99.99. La moyenne annuelle est la moyenne arithmétique des 12 mois de l'année. Les années où une ou deux valeurs sont manquantes, les valeurs annuelles sont calculées en substituant une valeur d'ajustement pour ce mois.

18.2 Mise en forme des données

On récupère sur le net un fichier ASCII. Si on ouvre avec un éditeur de texte le fichier se présente au début comme

```
*****
*** Atmospheric CO2 concentrations (ppmv) derived from in situ ***
*** air samples collected at Mauna Loa Observatory, Hawaii ***
***
*** Source: C.D. Keeling ***
*** T.P. Whorf, and the Carbon Dioxide Research Group ***
*** Scripps Institution of Oceanography (SIO) ***
*** University of California ***
*** La Jolla, California USA 92093-0444 ***
***
*** June 2004 ***
***
*****
```

suivi d'une ligne de caractères suivi de lignes de chiffres, dont le début est

```
Year Jan. Feb. March April May June July Aug. Sept. Oct. Nov. Dec. Annual Annual-Fit
1958 -99.99 -99.99 315.71 317.45 317.50 -99.99 315.86 314.93 313.19 -99.99 313.34 314.67 -99.99 -99.99
1959 315.58 316.47 316.65 317.71 318.29 318.16 316.55 314.80 313.84 313.34 314.81 315.59 315.98 316.00
1960 316.43 316.97 317.58 319.03 320.03 319.59 318.18 315.91 314.16 313.83 315.00 316.19 316.91 316.91
1961 316.89 317.70 318.54 319.48 320.58 319.78 318.58 316.79 314.99 315.31 316.10 317.01 317.65 317.63
1962 317.94 318.56 319.69 320.58 321.01 320.61 319.61 317.40 316.26 315.42 316.69 317.69 318.45 318.46
1963 318.74 319.08 319.86 321.39 322.24 321.47 319.74 317.77 316.21 315.99 317.07 318.36 318.99 319.02
1964 319.57 -99.99 -99.99 -99.99 322.23 321.89 320.44 318.70 316.70 316.87 317.68 318.71 -99.99 319.52
1965 319.44 320.44 320.89 322.13 322.16 321.87 321.21 318.87 317.81 317.30 318.87 319.42 320.03 320.09
```

Enfin le fichier se termine par un commentaire.

Il faut transformer ce fichier pour que Scilab puisse l'interpréter. Le plus simple est avec l'éditeur de supprimer ce qui n'est pas pertinent, autrement dit ce qui n'est pas chiffres. A savoir les 14 première lignes. Cela peut être fait avec un éditeur de texte qui lit les fichiers ASCII. Mais on peut aussi le faire avec Scilab. L'avantage est que l'on maîtrise ce que Scilab va récupérer. On n'introduira pas de caractères invisibles ou parasites.

A titre d'exercice on va le faire avec Scilab.

On va chercher le fichier "maunaloa.dat"

```
-->chdir('Documents/Scilab2')
ans =
```

0.

```
-->A=mgetl('maunaloa.dat');
```

On va essayer d'analyser ce fichier qui est une matrice de strings.

```
-->size(A)
```

```
ans =
```

```
! 66.    1. !
```

```
-->M=length(A);
```

```
-->M'
```

```
ans =
```

```
column 1 to 10
```

```
! 67.    67.    67.    67.    67.    67.    67.    67.    67.    67. !
```

```
column 11 to 19
```

```
! 67.    67.    67.   106.   102.   102.   102.   102.   102. !
```

```
column 20 to 27
```

```
! 102.   102.   102.   102.   102.   102.   102.   102. !
```

```
column 28 to 35
```

```
! 102.   102.   102.   102.   102.   102.   102.   102. !
```

```
column 36 to 43
```

```
! 102.   102.   102.   102.   102.   102.   102.   102. !
```

```
column 44 to 51
```

```
[More (y or n ) ?]
```

```
! 102.   102.   102.   102.   102.   102.   102.   102. !
```

```
column 52 to 59
```

```
! 102.   102.   102.   102.   102.   102.   102.   102. !
```

```
column 60 to 66
```

On a utilisé ici la commande `length` que l'on a déjà vu. Mais avec un résultat différent. Explication :

Syntaxe 18.2.1 :

Pour une matrice M de nombres ou une matrice polynomiale, la réponse est le produit du nombre de lignes par celui du nombre de colonnes, autrement dit le nombre d'éléments de la matrice. Pour une matrice de string, `length(M)` donne une matrice de même taille que M dont les éléments sont les longueurs en caractère des éléments correspondant de M.

En revanche `size(M)` pour une matrice de string sera inchangé, ce sera le nombre de lignes et de colonnes.

Avec cette précision on constate que l'on observe que les 13 premières lignes ont 67 caractères, qu'il y a un changement en longueur à partir de 14. On regarde ce qu'il en est :

```
-->A(1:13)
ans =

!*****!
!
!*** Atmospheric CO2 concentrations (ppmv) derived from in situ ***!
!
!*** air samples collected at Mauna Loa Observatory, Hawaii ***!
!
!*** ***!
!
!*** Source: C.D. Keeling ***!
!
!*** T.P. Whorf, and the Carbon Dioxide Research Group ***!
!
!*** Scripps Institution of Oceanography (SIO) ***!
!
!*** University of California ***!
!
!*** La Jolla, California USA 92093-0444 ***!
!
!*** ***!
!
!*** June 2004 ***!
!
!*** ***!
!
!*****!
```

La ligne 14

```
-->A(14)
ans =

Year Jan. Feb. March April May June July Aug. Sept. Oct.
Nov. Dec. Annual Annual-Fit
```

On voit qu'il s'agit de la première ligne d'un tableau qui donne la signification des colonnes.

On supprime ces lignes dont on n'a pas besoin

```
-->A(1:14)=[ ];
-->M=length(A)';
```

Les lignes à partir de 48, pour la nouvelle matrice, sont différentes

```
-->A(48:$)
ans =

!Monthly values are expressed in parts per million (ppm) and reported in th!
!   e 2003A SIO manometric mole                                           !
!                                                                           !
!fraction scale. The monthly values have been adjusted to the 15th of each!
!   month. Missing values are                                           !
!                                                                           !
!denoted by -99.99. The "annual" average is the arithmetic mean of the twel!
!   ve monthly values. In years                                         !
!                                                                           !
!with one or two missing monthly values, annual values were calculated by s!
!   ubstituting a fit value                                             !
!                                                                           !
!(4-harmonics with gain factor and spline) for that month and then averagin!
!   g the twelve monthly values.
```

On les supprime et on regarde la première ligne de la matrice obtenue

```
-->A(48:$)=[ ];

-->A(1)
ans =

1958 -99.99 -99.99 315.71 317.45 317.50 -99.99 315.86 314.93 313.19 -99.99
      313.34 314.67 -99.99 -99.99
```

On peut maintenant transformer la matrice de string en un matrice de réels

```
-->co2=evstr(A);
```

On supprime la première colonne qui est l'année, la dernière et l'avant dernière qui sont les moyennes annuelles et moyennes annuelles ajustées.

```
-->size(co2)
ans =

!   46.    15. !

-->co2(:, [1,$-1,$])=[ ];

-->size(co2)
ans =

!   46.    12. !
```

On a bien 46 années (de 1958 à 2003) sur 12 années. On va mettre ces données en colonne chronologique. On transpose pour avoir les mois colonne par colonne de 1958 à 2003, puis on transforme en un grand vecteur colonne de 552 données :

```
-->co2=co2';
-->co2=co2(:);
-->size(co2)
ans =
!    552.    1. !
```

Il reste à tenir compte des données manquantes : les -99.99. On va les remplacer par une interpolation.

18.3 Interpolation des données manquantes

On cherche l'indice des données manquantes

```
-->find(co2==-99.99)
ans =
!    1.    2.    6.   10.   74.   75.   76. !
```

On peut donc supprimer les deux premières mesures, plutôt que de les interpoler.

```
co2(1:2)=[ ] ;
```

On va interpoler les données manquantes. On identifie à nouveau les indices

```
-->I=find(co2==-99.99)
I =
!    4.    8.   72.   73.   74. !
```

Pour l'interpolation linéaire on va utiliser une fonction toute faite de Scilab.

18.4 Interpolation linéaire

Syntaxe 18.4.1 :

La commande `y=interpnl([X;Y],x)` où X est un vecteur **ligne** donnant les abscisses,

Y un vecteur **ligne** de même dimension donnant les ordonnées connues, enfin x une matrice des abscisses où l'on veut interpoler linéairement à partir des points connus. y est le vecteur ligne des interpolations calculées au points x_i

Application : on connaît *co2* aux indices avant et après les inconnues, i.e.

```
-->X=[3 5 7 9 71 75]
-->y=interp1n([X;co2(X)'], I);
--> co2(I)=y';
```

On va avoir une idée du nuage de points en dessinant le graphe.

```
-->xbasc()
--> plot2d((1:550)',co2,style=-1)
```

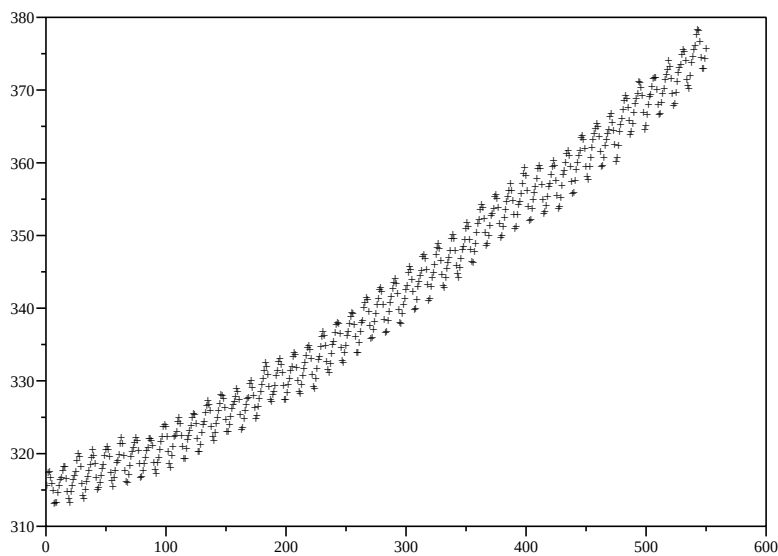


FIG. 18.1 – les points de mesure du taux de CO2 en ppm

On constate une tendance. On va essayer d'approximer cette tendance par une parabole $a + bt + ct^2$.

18.5 Tendance générale

Les mesures vont de mars 1958 à décembre 2003. Le vecteur T des temps (unité de temps le mois) varie de 1 à 550.

```
-->T=(1:550)';

-->A=[ones(550,1),T,T.^2];

-->sol=A\co2
sol =

!    314.14762 !
!    0.0682202 !
!    0.0000815 !
```

Traçons la tendance sur le graphe des points en rouge et faisons en zoom sur les 151 dernières mesures

```
-->plot2d(T,sol(1)+sol(2)*T+sol(3)*T.^2,style=5)

-->xset("window",1) \\ nouvelle fenêtre

-->plot2d(T(300:$),sol(1)+sol(2)*T(300:$)+sol(3)*T(300:$).^2,style=5)

-->plot2d(T(300:$)',co2(300:$),style=-1)
```

On obtient

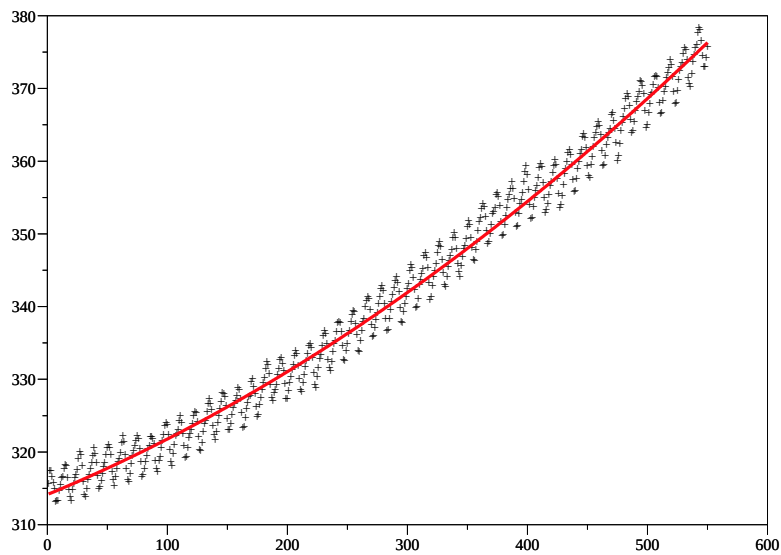


FIG. 18.2 – Tendance générale en rouge

Le zoom

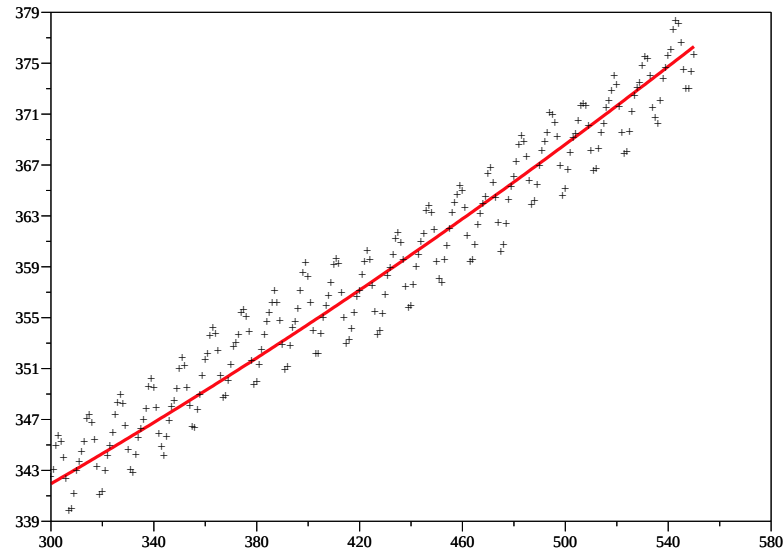


FIG. 18.3 – Zoom sur les dernières valeurs

On dessine les résidus

```
-->xset("window",2)
```

```
-->plot2d(T,co2-(sol(1)+sol(2)*T+sol(3)*T.^2),style=-1)
```

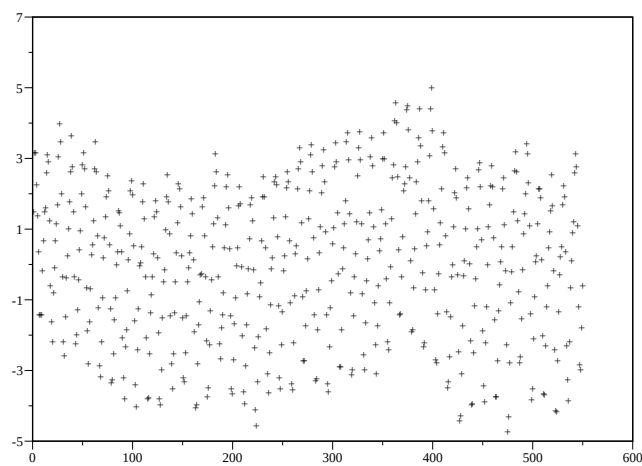


FIG. 18.4 – Résidus

Dessignons les points de mesures (reliés par des droites sur les 100 dernières mesures

```
--xset("window",3)

-plot2d(T(450:$)',co2(450:$))
```

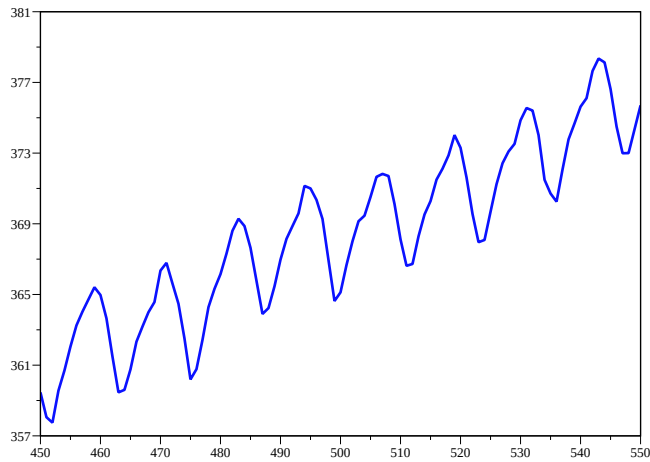


FIG. 18.5 – les 100 dernières mesures

18.6 Ajustement périodique

18.6.1 Première harmonique

On estime à $P=12$ la période, on va essayer la première harmonique. Autrement dit on va estimer les mesures, avec les notations du (18.1), par

$$y(t) = a + b t + c t^2 + a_1 \cos\left(\frac{2 \pi k t}{T}\right) + b_1 \sin\left(\frac{2 \pi k t}{T}\right)$$

On pose

```
-->A=[ones(550,1), T, T.^2, cos(2*%pi*T/12), sin(2*%pi*T/12)] ;
```

```
-->-->A=[ones(550,1),T,T.^2,cos(2*%pi*T/12),sin(2*%pi*T/12)];
```

```
-->sol1=A\co2
sol1 =
```

```
! 314.10414 !
```

```
! 0.0683391 !
! 0.0000816 !
! 1.0543311 !
! 2.5766248 !
```

Il reste à voir ce que cela donne

```
-->xbasc()
-plot2d(T(450:$)',co2(450:$),style=-1) ;
-->x=T(450:$)';
--> y= sol1(1)+sol1(2)*x+sol1(3)*x.^2+sol1(4)*cos(2*%pi*x/12)+sol1(5)*sin(2*%pi*x/12);
-->plot2d(x,y,style=5)
```

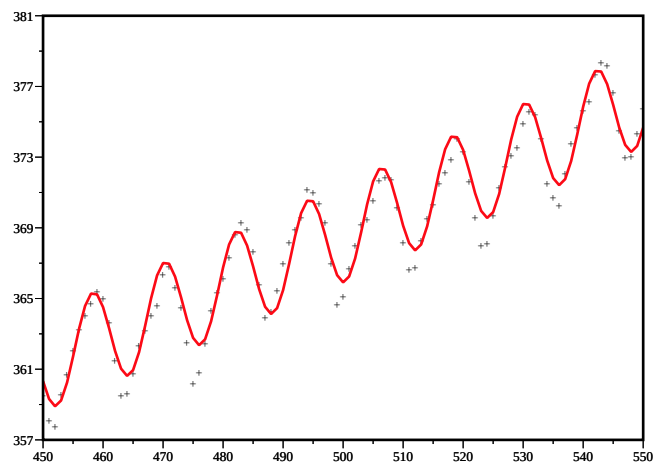


FIG. 18.6 – première harmonique, 100 dernières mesures

Regardons les résidus

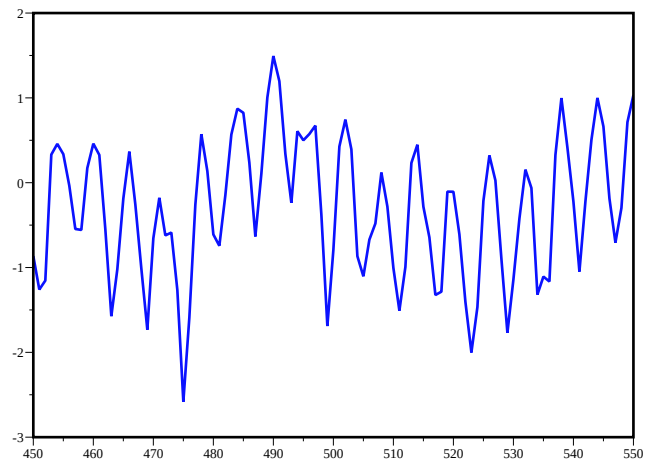


FIG. 18.7 – première harmonique, résidus 100 derrières mesures

18.6.2 Deuxième harmonique

il reste encore un peu de périodicité. On essaie $n = 2$

```
-->y=sol1(1)+sol1(2)*x+sol1(3)*x.^2+sol1(4)*cos(2*pi*x/12)+sol1(5)*sin(2*pi*x/12);
-->y=y+sol1(6)*cos(4*pi*x/12)+sol1(7)*sin(4*pi*x/12);
-->xbasc()
-->plot2d(x,co2(450:$),style=-1)
-->plot2d(x,y,style=5)
```

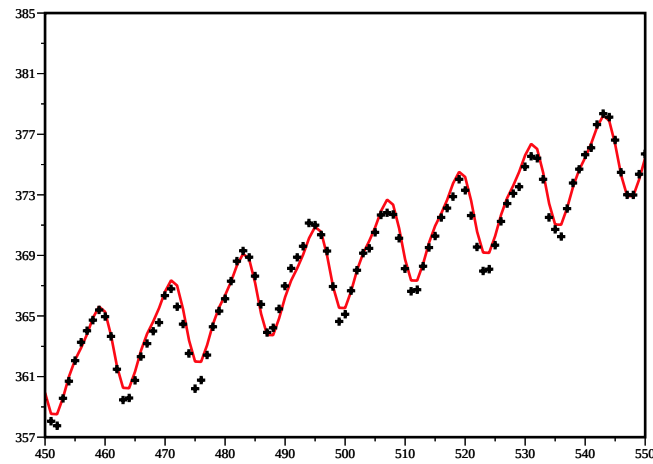


FIG. 18.8 – deuxième harmonique, 100 dernières mesures

Les résidus

```
-->xbasc()
```

```
-->plot2d(x,co2(450:$)-y,style=2)
```

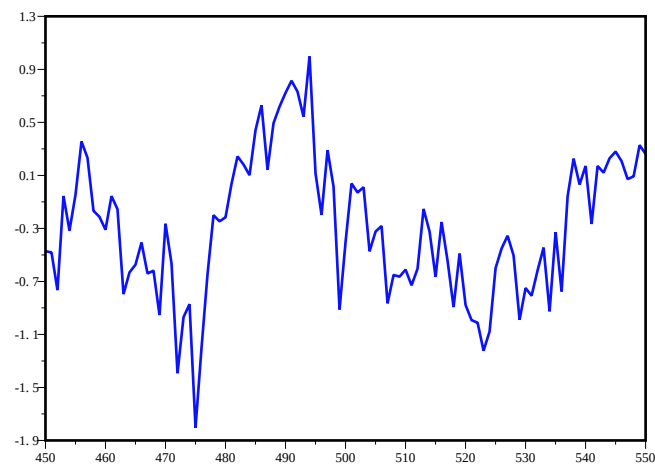


FIG. 18.9 – deuxième harmonique, résidus

On pourrait vérifier que pour $n = 3, 4 \dots$ cela ne change plus. Il faudrait trouver des harmoniques plus élevées. On verra cela plus loin avec la transformée de Fourier.

Chapitre 19

Graphiques 2D

On a utilisé précédemment des graphiques, on a vu leur utilité. Ce chapitre va donner les syntaxes systématiques.

19.1 Les fenêtres graphiques

19.1.1 Créer une fenêtre

Syntaxe 19.1.1 :

```
xset("window",window-number)
```

Choisit la fenêtre `window-number` comme fenêtre graphique par défaut (fenêtre courante) et crée cette fenêtre si elle n'existe pas.

19.1.2 Effacer une fenêtre

Par défaut Scilab superpose chaque dessin dans une fenêtre.

Il est donc nécessaire d'effacer un dessin avant d'en dessiner un autre si on ne veut pas les deux dessins simultanément.

Syntaxe 19.1.2 :

```
xbasc() ou xbasc(window-number)
```

Sans argument, cette fonction efface la fenêtre graphique courante et les graphiques enregistrés. Sinon elle efface les fenêtres graphiques dont les numéros sont donnés dans le vecteur

`window-number` , et efface les graphiques enregistrés.

Par exemple `xbasc(1:3)` efface les fenêtres 1, 2 et 3.

Si une des fenêtres n'existe pas, elle est automatiquement créée.

Si l'on veut que Scilab efface automatiquement chaque dessin avant d'en dessiner un nouveau, il faut régler le paramètre de la fenêtre

Syntaxe 19.1.3 :

```
xset("auto clear","on"|"off")
```

Met 'on' ou 'off'

le mode d'effacement automatique des graphiques.

Quand le mode est 'on', les dessins successifs ne sont pas superposés, i.e. la commande `xbasc()` est exécutée avant chaque commande graphique de haut niveau (`plot2d` par exemple).

La valeur par défaut est 'off'.

19.1.3 Sélectionner une fenêtre**Syntaxe 19.1.4 :**

```
xselect()
```

`xselect ( )` met la fenêtre graphique courante au premier plan. Une fenêtre est créée si aucune fenêtre n'existait auparavant. Si la fenêtre graphique est iconifiée, `xselect` n'a aucun effet.

19.2 Diviser la fenêtre**Syntaxe 19.2.1 :**

La commande `subplot(m,n,p)` permet de subdiviser la fenêtre graphique en $m \times n$ sous-fenêtres. p le le numéro de sous-fenêtre. Le compte se fait ligne par ligne (contrairement à la numérotation matricielle Scilab)

`subplot(2,3, 4)` est la sous-fenêtre (2,1) i.e. deuxième ligne, première colonne.

Voir un exemple en (7.3.2)

19.3 Résumé

TAB. 19.1 – Gestion de fenêtres

<code>xset('window', i)</code>	la fenêtre <code>i</code> est courante ou crée la fenêtre <code>i</code>
<code>xbasc()</code>	Efface la fenêtre courante
<code>xbasc(i)</code>	Efface la fenêtre <code>i</code>
<code>xselect ()</code>	Met en avant la fenêtre courante
<code>subplot(m,n,p)</code>	sous-fenêtre <code>p</code> de la fenêtre

19.4 la commande simple plot2d

Syntaxe 19.4.1 :

`plot2d([x],y)` ou `plot2d([x],y,<opt_args>)`

Les paramètres sont x un vecteur colonne ou une matrice et y est un vecteur colonne ou une matrice de même dimension que x . On trouve ensuite des arguments optionnels qui permettent d'adapter le graphe aux besoins.

19.4.1 Utilisation simple

Si x et y sont des vecteurs alors Scilab trace les points de coordonnées (x_i, y_i) . Les deux vecteurs doivent avoir la même dimension. Il n'est pas obligé que ce soit des vecteurs colonnes, mais il vaut mieux en prendre l'habitude, pour le cas général.

Si x et y sont des matrices, elles doivent avoir les mêmes dimensions. Scilab trace les courbes correspondant à chaque colonne de x et y . La i -ème courbe correspond aux points définis par la i -ème colonne de x en abscisse et la i -ème colonne de y en ordonnées.

Si x est un vecteur (colonne) et y une matrice alors Scilab trace les courbes correspondant à chaque colonne de y pour ordonnées et x pour vecteur des abscisses. Cela impose bien que x est un vecteur colonne.

19.4.2 Exemple

Une courbe

```
-->x=0:0.01:2*pi;  
-->x=x';  
-->subplot(1,2,1)  
-->plot2d(x,sin(x))  
-->subplot(2,2,1)  
-->plot2d(x,[sin(x),cos(x),cos(2*x)])
```

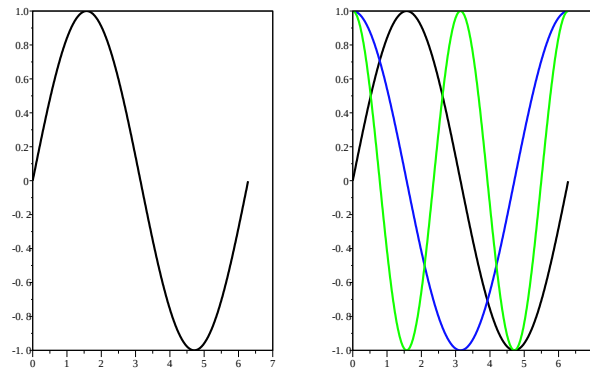


FIG. 19.1 – une courbe, puis trois courbes

19.5 Personnaliser un graphique

On dispose de plusieurs entrées optionnelles qui sont utilisées de la façon suivante `entrée1=valeur1, entrée2=valeur2`

Les options sont

1. `style`
2. `rect`
3. `logflag`
4. `frameflag`
5. `axesflag`
6. `nax`
7. `leg`

19.5.1 style

Par défaut Scilab colore les courbes et trace un trait continu entre points consécutif. Sinon la commande est `style=x` où `x` est un vecteur de dimension égal au nombre de courbes, i.e. le nombre de colonnes de `y`. Scilab utilisera le style `x(i)` pour la i -ème courbe. La valeur `x(i)` est un entier relatif. Si c'est strictement positif le tracé est continu. Si la valeur est négative seul les points sont dessinés avec un marqueur particulier.

Voici le code des couleurs

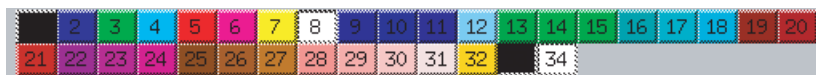


FIG. 19.2 – style=code

Pour ceux qui n'ont pas ce document en couleur, voici la table

TAB. 19.2 – Les codes de style

1	noir
2	bleu foncé
3	vert clair
4	bleu clair
5	vermillon
6	un rose
7	jaune
25	brun

Les styles négatifs tracent les points, sans les relier par des traits, en utilisant un marqueur. Voici 10 lignes tracée horizontalement, de la côte 0 à 9 en utilisant les marqueurs de 0 à -9

```
-->x=0:9;

-->y=x(ones(1,10),:);

-->x=x';

-->plot2d(x,y,style=0:-1:-9,rect=[-1,-1,10,10])
```

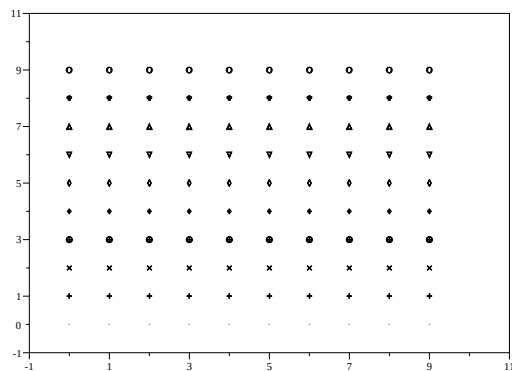


FIG. 19.3 – style=-i

lignes pointillées Il existe 6 styles de lignes pointillées. On ne peut le régler directement à partir de `plot2d`. Il faut le faire en utilisant le réglages de fenêtre.

Ce réglage, s'il n'est pas modifié est valable pour tout les dessins faits dans la fenêtre. Autrement dit toutes les lignes tracées seront pointillée. Il faut changer le réglage pour obtenir autre chose. Voici 6 lignes tracées avec un réglage de 1 à 6

```
x=1:10;x=x(:);
for i=1:6,xset("line style",i); plot2d(x,i*ones(x),rect=[0,0,10,7],style=i), end
```

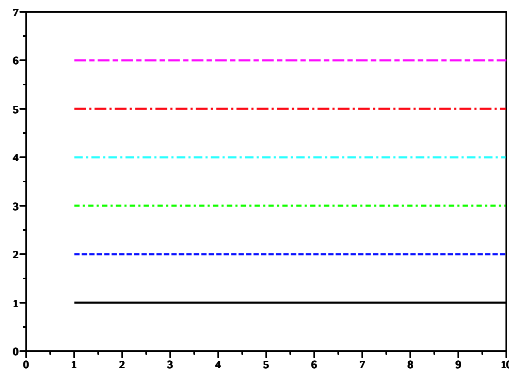


FIG. 19.4 – pointillés

Epaisseur des lignes C'est le même principe que pour le "linestyle". On règle par la commande

```
xset (thickness", i)"
```

L'indice i peut valoir de 1 à 20

Par exemple, voici un fichier script que l'on exécute, composé des commandes suivantes

```
x=1:10;x=x(:);
for i=1:6,xset("line style",i);
xset("thickness",i)
plot2d(x,i*ones(x),rect=[0,0,10,7],style=i),
end
```

ce qui donne

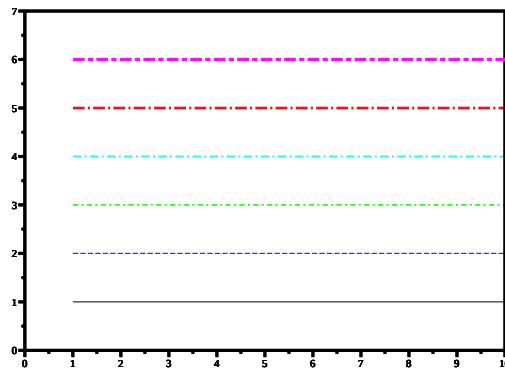


FIG. 19.5 – Différentes épaisseurs

19.5.2 rect

Syntaxe 19.5.1 :

La syntaxe est `rect=[xmin,ymin, xmax,ymax]`

Autrement dit on définit le coin en bas à gauche et en haut à droite du rectangle.

Voir un exemple d'utilisation dans l'affichage des marqueurs.

19.5.3 logflag

Syntaxe 19.5.2 :

on écrit `logflag = 'xy'` où `x` et `y` prennent la valeur `n` ou `l`

- `logflag='nn'` graphe normal
- `logflag='nl'` graphe semi-log, l'axe des `y` est en échelle logarithmique
- `logflag='ll'` graphe log-log

exemple :

```
-->subplot(1,2,1)

-->plot2d(t',exp(t'),logflag="nl")

-->xtitle('exponentielle en echelle semi-log')

subplot(1,2,2)

plot2d(t',t'.^3,logflag="ll")

-->xtitle('puissance en echelle log-log')
```

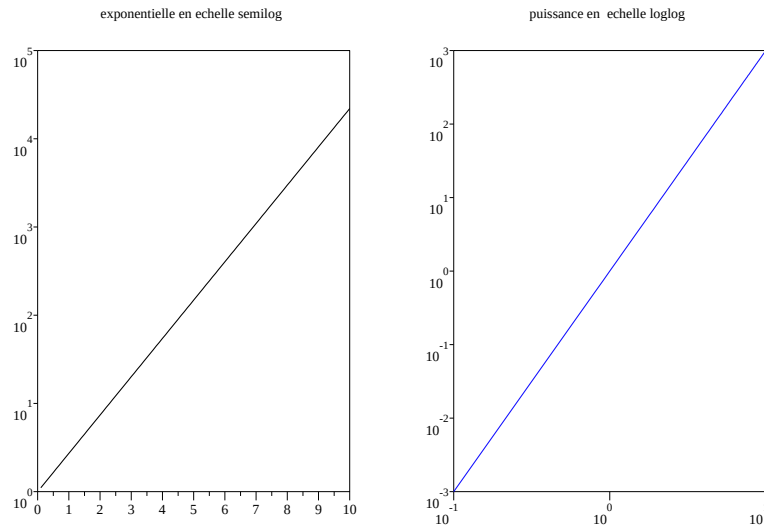


FIG. 19.6 – Echelles logarithmiques

19.5.4 frameflag

:
 frameflag gère la façon dont sont tracés les différents dessins, en fonction des coordonnées.

1. **frameflag=0** aucun calcul, le tracé utilise l'échelle précédente, où une échelle par défaut.
2. **frameflag=1** Le cadre du tracé est donné par l'option **rect**
3. **frameflag=2** Le cadre est calculé à partir des valeurs extrémales de **x** et **y**
4. **frameflag=3** Le cadre est donné par **rect** et agrandi pour donner une échelle isométrique
5. **frameflag=4** Le cadre est calculé à partir des valeurs extrémales de **x** et **y** et agrandi pour donner une échelle isométrique
6. **frameflag=5** Le cadre est donné par **rect** et agrandi pour donner une jolie numérotation des axes
7. **frameflag=6** Le cadre est calculé à partir des valeurs extrémales de **x** et **y** et agrandi pour donner une jolie numérotation des axes
8. **frameflag=7** Comme **frameflag=1**, mais les dessins précédents sont re-tracés pour utiliser la nouvelle échelle. Utilisé pour ajouter le graphe courant à un graphe précédent.

9. `frameflag=8` Comme `frameflag=2`, mais les dessins précédents sont re-tracés pour utiliser la nouvelle échelle. Utilisé pour ajouter le graphe courant à un graphe précédent.

Voici un exemple d'utilisation obtenu en exécutant le script suivant

```
x=0:0.01:2*pi;x=x(:);

xbasc()

subplot(3,3,1)

plot2d(x,ones(x))

plot2d(x,sin(x),frameflag=0,style=2)

xtitle('frameflag=0')

subplot(3,3,2)

plot2d(x,sin(x),frameflag=1,style=2,rect=[0,0,6.3,1])

xtitle('frameflag=1')

subplot(3,3,3)

plot2d(x,sin(x),frameflag=2,style=2)

xtitle('frameflag=2')

subplot(3,3,4)

plot2d(x,sin(x),frameflag=3,style=2,rect=[0,0,6.3,1])

xtitle('frameflag=3')

subplot(3,3,5)

plot2d(x,sin(x),frameflag=4,style=2)

xtitle('frameflag=4')

subplot(3,3,6)

plot2d(x,sin(x),frameflag=5,style=2,rect=[0,0,6.3,1])

xtitle('frameflag=5')
```

```

subplot(3,3,7)

plot2d(x,sin(x),frameflag=6, style=5)

xlabel('frameflag=6')

subplot(3,3,8)

plot2d(x,ones(x))

plot2d(x,sin(x),frameflag=7,style=5)

xlabel('frameflag=7')

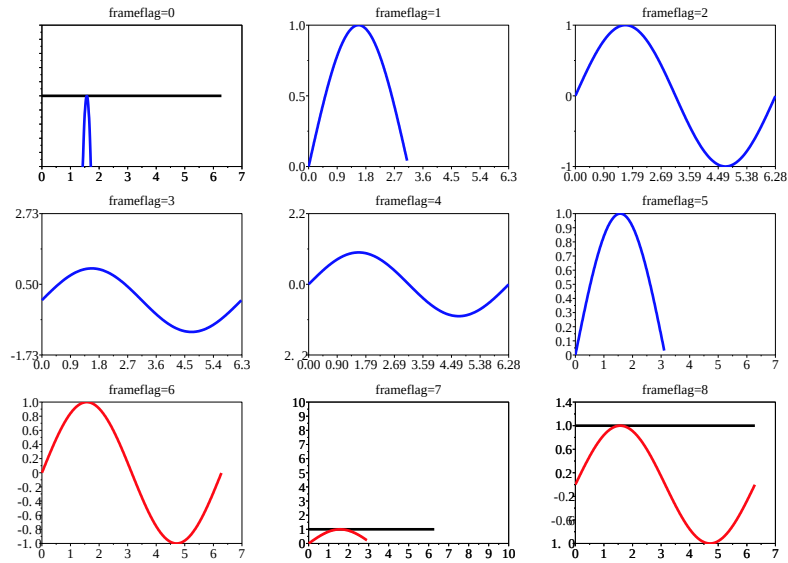
subplot(3,3,9)

plot2d(x,ones(x))

plot2d(x,sin(x),frameflag=8,style=5,rect=[0,-1,6.5,1])

xlabel('frameflag=8')

```

FIG. 19.7 – Exemples d'utilisation de `frameflag`

19.5.5 axesflag

Cette option spécifie comment les axes et les encadrements sont tracés. Elle prend 6 valeurs

1. `axesflag=0` pas de « boîte » autour
2. `axesflag=1` Des axes, l'axe des y est à gauche
3. `axesflag=2` Une boîte non graduée
4. `axesflag=3` Des axes, l'axe des y est à droite
5. `axesflag=4` Des axes centrés au milieu du dessin.
6. `axesflag=5` Des axes intersection au point $(0,0)$, si ce point est dans le cadre. Sinon pas d'axes.

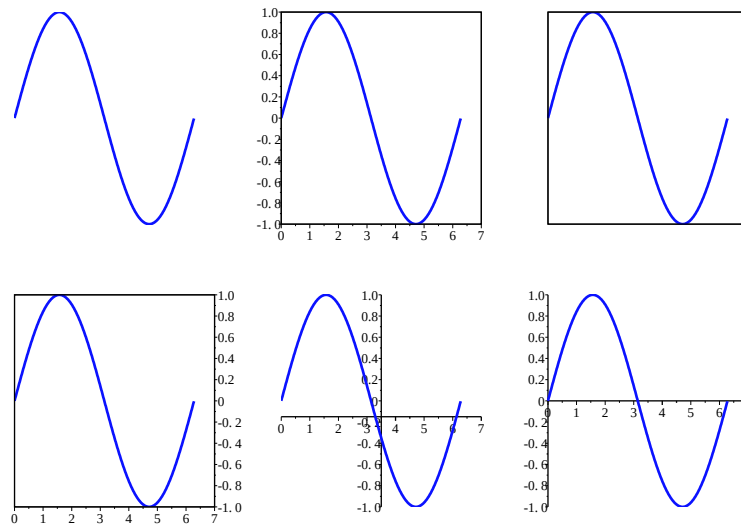


FIG. 19.8 – Exemples d'utilisation de `axesflag`

19.5.6 nax

Cette option pour spécifier la graduation des axes

19.5.7 leg

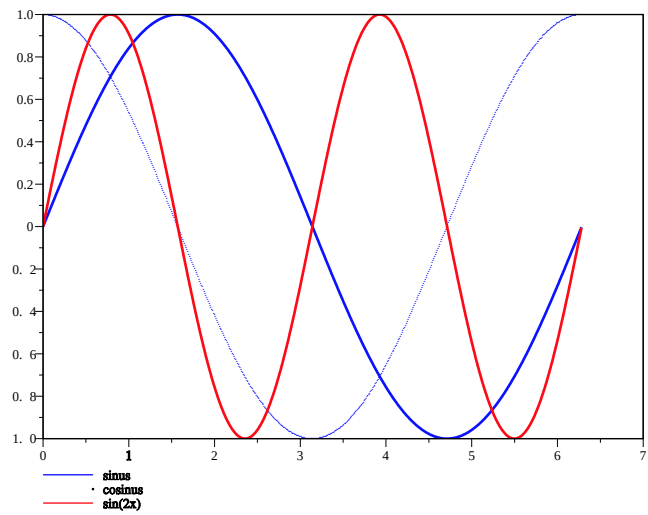
Pour mettre une légende sur chaque courbe

Syntaxe 19.5.3 :

`leg=legende1@legende2@ ...@legendek " "`

où *legende_i* est la chaîne de caractère que l'on veut associer à la courbe *i*

```
-->plot2d(x,[sin(x),cos(x),sin(2*x)],style=[2,-0,5],leg="sinus@cosinu @sin(2x)')
```

FIG. 19.9 – Exemples d'utilisation de `leg`

Chapitre 20

Transformée de Fourier discrète

Dans les chapitres précédents on a utilisé la puissance de Scilab, calcul, programmation et graphique, pour ajuster des données. On a utilisé la méthode des moindres carrés, en linéaire.

Dans le cas de signaux périodiques on a vu que si l'on connaissait la période le problème se ramenait à un problème linéaire. La question est précisément comment déterminer cette période. La transformée de Fourier est un outil puissant pour ce type de problème. On se donne un vecteur x de longueur n . En fait il faut penser que $x(j)$ sont les n mesures d'un signal échantillonné. Alors on associe à x un vecteur X dont la k -ème composante est définie par

$$X(k) = \sum_{j=1}^n x(j) e^{-2\pi i \frac{(j-1)(k-1)}{n}} \quad (20.1)$$

Cette formule diffère de la formule classique, car on a sommé de 1 à n au lieu de 0 à $n-1$. Ceci pour tenir compte de Scilab qui commence les indices des vecteurs à 1

Si on connaît X on peut récupérer le signal d'origine x par la formule

$$x(j) = \frac{1}{n} \sum_{k=1}^n X(k) e^{2\pi i \frac{(j-1)(k-1)}{n}} \quad (20.2)$$

Si le signal est réel alors x peut s'écrire

$$x(j) = \frac{1}{n} \left[\sum_{k=1}^n a(k) \cos \left(2\pi \frac{(k-1)(j-1)}{n} \right) + b(k) \sin \left(2\pi \frac{(k-1)(j-1)}{n} \right) \right] \quad (20.3)$$

Avec

$$a(k) = \Re(X(k))$$

et

$$b(k) = \Im(X(k))$$

Le signal est écrit comme une somme de signaux périodiques de différentes périodes.

Les périodes sont $2\pi \frac{k-1}{n}$.

Avec nos notations il y a une phase de $\frac{-2(k-1)\pi}{n}$. C'est dû au $j-1$.

20.1 FFT

Il existe un algorithme pour calculer rapidement les transformées de Fourier discrète : FFT

F.F.T. : Fast Fourier Transform

Cet algorithme est implémenté en Scilab

Syntaxe 20.1.1 :

`X=fft(x,-1)`

Le signe -1 est optionnel, il fait référence au signe $-$ dans l'exponentielle dans la formule (20.1)

20.2 FFT inverse

Syntaxe 20.2.1 (transformée de Fourier inverse) :

`x=fft(X,1)`

Le 1 fait référence au signe $+$ dans l'exponentielle dans la formule (20.2)

20.3 Définitions

On considère un signal réel de longueur n . Autrement dit `n=length(x)`

Soit X sa transformée de Fourier discrète. La valeur de $X(1)$ vérifie

$$X(1) = \sum_{j=1}^n x(j)$$

On peut donc supprimer la première composante de X

`-->X(1)=[ ];`

On note F_e la fréquence d'échantillonnage. Autrement dit le nombre de mesures par unité de temps.

La discrétisation, i.e. l'échelle des temps, est donc $\mathbf{t}=(1:n-1)/F_e$.

Le pas de temps est $dt = \frac{1}{F_e}$, c'est l'intervalle de temps entre deux mesures.

Définition 20.3.1 :

Le vecteur des modules des composante de la transformée de Fourier discrète s'appelle l'amplitude Ampdu signal

$$Amp(k) = \sqrt{(a(k))^2 + b(k)^2}$$

Le carré de l'amplitude $Amp.^2$ s'appelle le vecteur puissance du signal.

Le vecteur des fréquences est défini comme

$$\mathbf{f}=(1:n-1)*(F_e/n)$$

Le vecteur des périodes est défini comme l'inverse des périodes $\mathbf{p}=1 ./ \mathbf{f}$

Définition 20.3.2 :

On appelle fréquence de Nyquist la quantité $(n/2)(F_e/n)=F_e/2$*

On appelle périodogramme le tracé de la puissance ou de l'amplitude en fonction de la fréquence. On a simplement besoin de tracer jusqu'à la moitié car le périodogramme est symétrique par rapport à la fréquence de Nyquist.

On va donc redéfinir le vecteur des fréquences :

$$\mathbf{f}=(1:n/2)*(F_e/n)$$

20.4 Application : le taux de CO₂ à Maunaloa

On a vu le taux de CO₂ relevé à Maunaloa. dans le fichier. On avait éliminé une tendance générale en approximant au moindre carrés par un polynôme du second degré (parabole)

Rappelons la forme du signal :

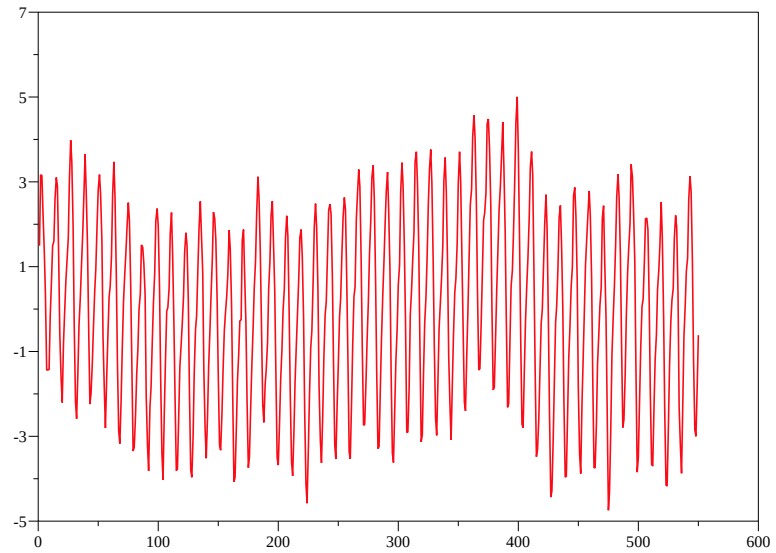


FIG. 20.1 – Signal périodique CO2

On va appliquer ce qui précède. On note x le signal du taux de CO_2 moins la tendance générale.

```
-->x=res1;

-->Fe=12;

-->n=length(x);

-->X=fft(x,-1);

-->X(1)=[ ] ;

-->pow=abs(X).^2;

-->f=(0:n-1)*(Fe/n);

-->Ny=(n/2)*(Fe/n)
Ny =
```

6.

```
-->xbasc()
```

```
-->plot2d(f,pow)
```

```
-->xbasc()
```

On obtient le périodogramme complet

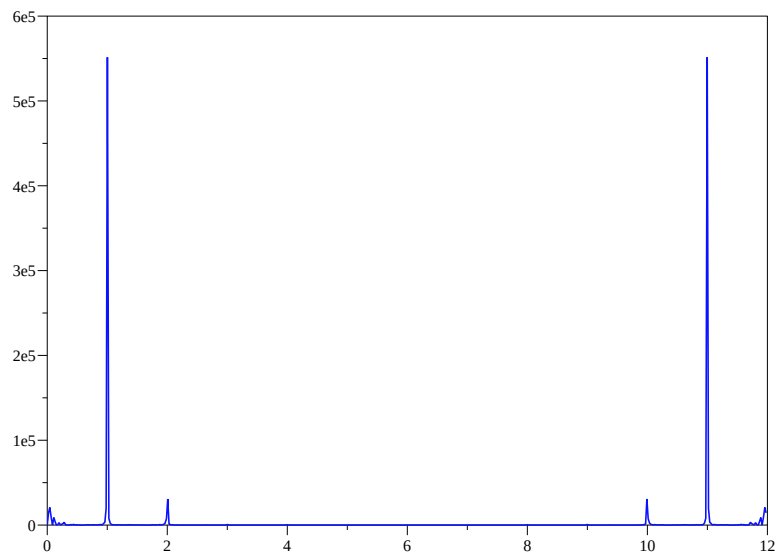


FIG. 20.2 – périodogramme complet CO₂

On constate, comme annoncé, qu'il est symétrique par rapport à la fréquence de Nyquist $Ny = 6$.

On va donc redéfinir comme prévu le vecteur des fréquences et retracer le périodogramme

```
-->f=(1: n/2)*(Fe/n);    \\ nouveaux vecteurs des fréquences
```

```
--> pow=pow(1:n/2);
```

```
-->plot2d(f,pow)
```

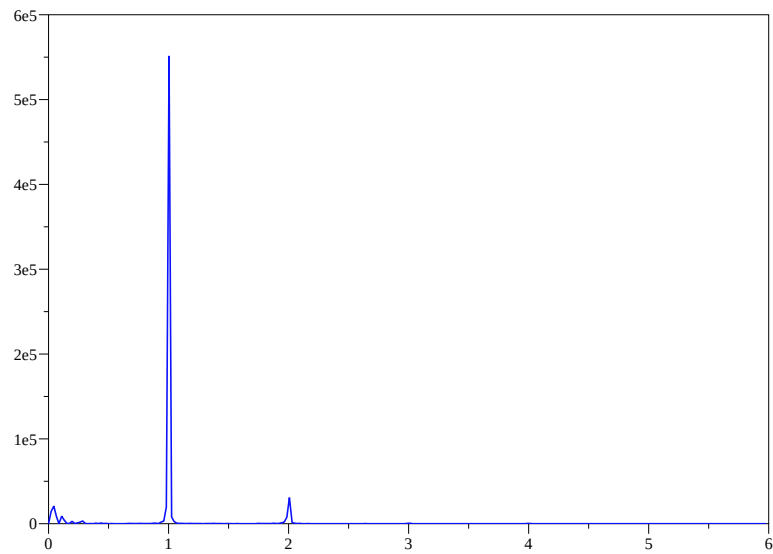


FIG. 20.3 – périodogramme CO2, jusqu'à la fréquence de Nyquist

On observe 3 pics qui correspondent à des fréquences particulières. On peut vouloir tracer la puissance par rapport aux périodes. Il suffit de calculer le vecteur des périodes, en prenant l'inverse. Mais le vecteur des fréquences commence à 0, il faut décaler et prendre de $(1:n/2)$

```
--> period=1 ./ ( (1:n/2)*(Fe/n)) :
--> xset("window",2)
--> plot2d(period,power(1:n/2),style=2)
```

Ce qui donne une figure différente

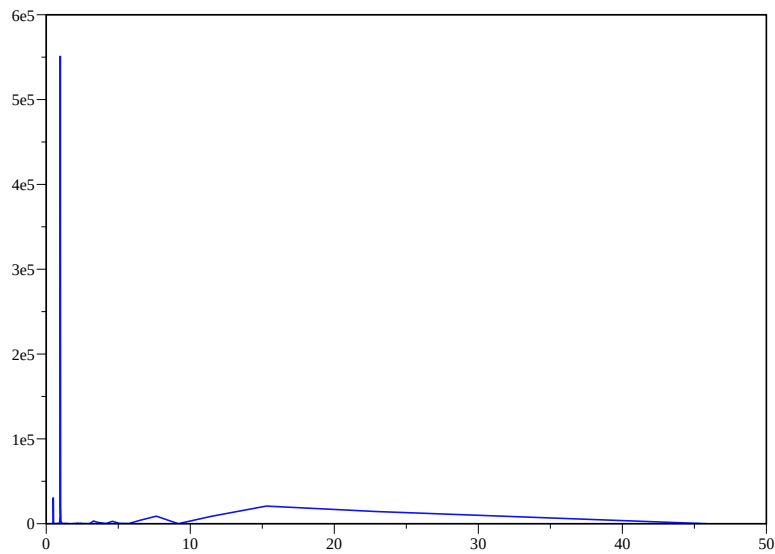


FIG. 20.4 – périodes en années

Si on cherche les pics

```
-->power=pow(1:n/2);

-->I1=find(power==max(power))
I1 =

    47.

-->period(47)
ans =

    0.9751773
```

On a donc une période de 0.97 année, soit 11.70, pratiquement un an à 3% près. Ce qui explique les bons résultats obtenus dans le deuxième chapitre. En effet la puissance pour la période 0.97 est `pow(I1)=551546`. On voit que ce pic est beaucoup plus important que les autres. Cela signifie que les amplitudes sont grandes par rapport aux autres pics. On va le vérifier.

```
-->power(I1)=-%inf;

-->I2=find(power==max(power))
I2 =
```

```

93.

-->period(I2)
ans =

0.4928315

-->power(I2)=-%inf;

-->I3=find(power==max(power))
I3 =

3.

-->period(3)
ans =

15.277778

```

On a trouvé les 2 autres petits pics, à 0.49 soit 6 mois et un autre à 15 ans. On a utilisé `-%inf` qui est dans Scila $-\infty$. Les valeurs des 3 pics sont

```

-->pow([I1,I2,I3])
ans =

! 551546.93 !
! 30731.325 !
! 20666.97  !

```

Cela confirme bien ce que l'on a vu sur le périodogramme.

20.5 FFT, Scilab et les touches du téléphone

Vous devez trouver sur votre machine un dossier `TPScilab` qui doit contenir deux fichiers Scilab, `elninosci` et `signal1` ainsi qu'un fichier texte `SUNSPOT.DAT`. La base du signal émis par une touche d'un téléphone est un signal à deux fréquences. Le pavé des touches d'un téléphone est une matrice 4×3 . Associée à chaque ligne sont associées les fréquences correspondantes :

```
f1=[ 697  770  852  941 ] ;
```

Associée à chaque colonne sont associées les fréquences correspondantes :

```
fc=[ 1209  1336  1477 ] ;
```

Cela correspond à la figure 20.5

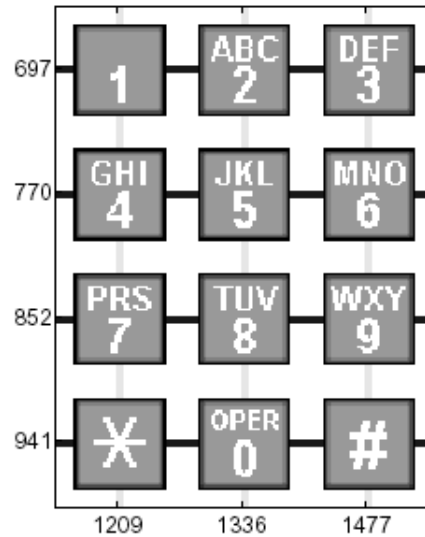


FIG. 20.5 – Touches et fréquences correspondantes

Par exemple la touche # en position (4,3) correspond aux fréquences

$[f_1(4), f_c(3)] = [941, 1477]$

La tonalité engendrée par la touche en position (i, j) est donné par le signal y qui est la superposition des deux sons fondamentaux :

```
y1 = sin(2*pi*f1(i)*t);
y2=sin(2*pi*f_c(j)*t);
y=(y1+y2)/2;
```

Le fichier `signal1` est un fichier scilab qui contient un signal bruité de la tonalité d'une touche.

20.5.1 Identifier la touche ?

Quelle est cette touche ?

20.5.2 Procédure

On va identifier les deux périodes qui composent le signal par une analyse de Fourier comme indiqué au paragraphe (20.

```
-->load('signal1')
-->x=signal1;
-->plot2d(x)
```

On trace le signal

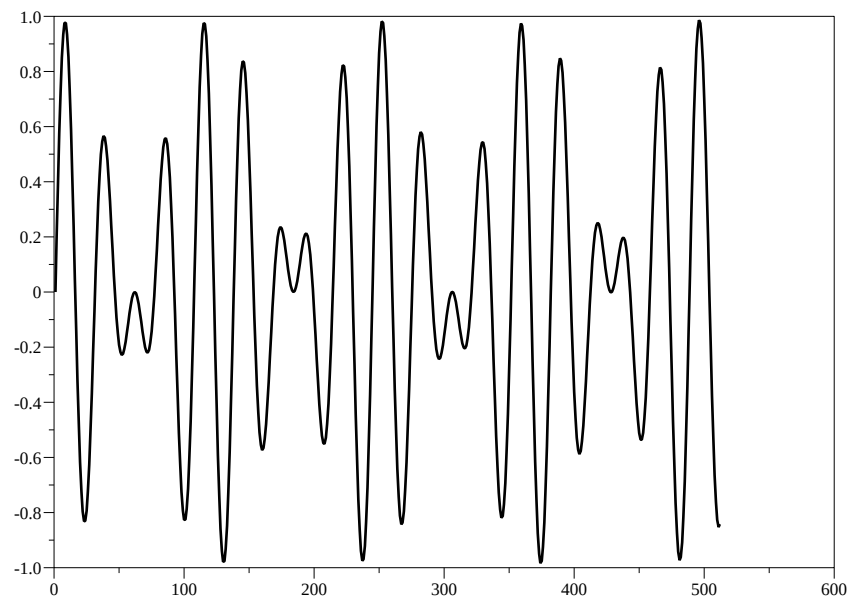


FIG. 20.6 – Le signal à analyser

```
-->n=length(x);
-->Fe=32768;  %%frequence d'échantillonnage
-->X=fft(x,-1);
-->pxx=abs(X);
-->f=(0:n/2)*Fe/n;
-->Nyquist=Fe/2
ans =
    16384.
--> find(f==16384)

ans =

    257.

-->plot2d(f,x(1:257))
```

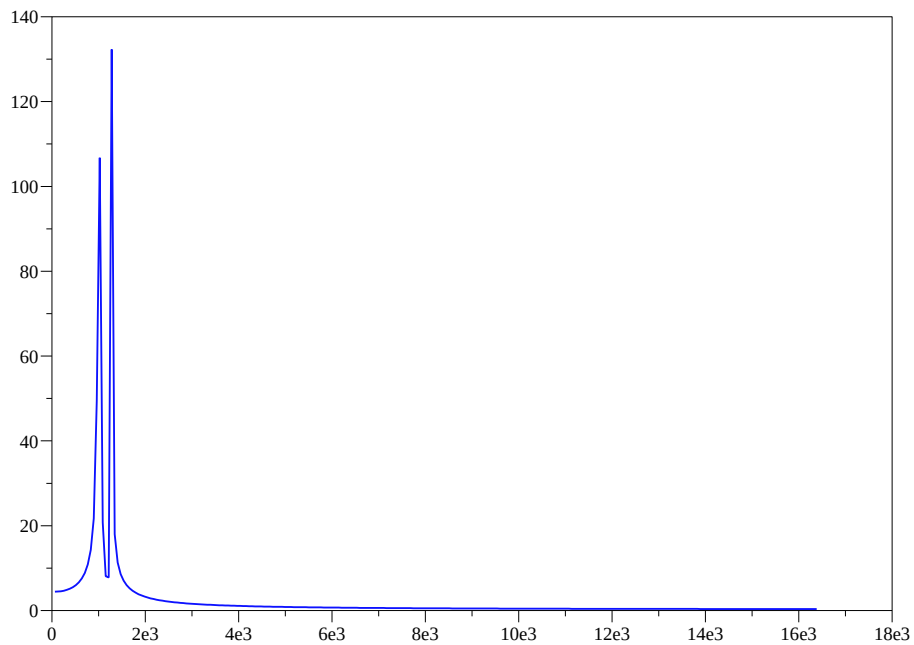


FIG. 20.7 – Périodogramme

```
-->px=pxx(0:n/2-1);
-->I1=find(px==max(px))
I1 =

    20.

-->f(20)
ans =

    1216.

-->px(I1)=-%inf;

-->I2=find(px==max(px))
I2 =

    16.

-->f(I2)
ans =
```

980.

Les fréquences sont 1216 et 980. On regarde la figure (20.5). Les plus proches sont (1209, 941). Il s'agit de la touche `*`.

20.6 El Niño

Le phénomène climatologique El Niño est le résultat de changements dans la pression atmosphérique de l'océan pacifique sud. L'indice d'oscillation du Sud est la différence entre la pression en Irlande Est et Darwin en Australie, mesurée au niveau de la mer au même moment.

Le fichier `elninosci` est un fichier scilab qui contient les valeurs de cet indice, mesurées chaque mois sur une période de 14 ans entre 1962 et 1975. On peut obtenir un fichier ASCII de ce tableau à : <http://www.bom.gov.au/index.shtml>. L'unité de temps est l'année, la période d'échantillonnage est de 12. On introduit le vecteur `x` qui correspond aux instants de mesure

```
--> load('elninosci')
--> y=elninosci;
--> n=length(y);
--> x=1:n;
--> x=x(:);
```

20.6.1 Identifier la tendance générale

Soit le polynôme $P(T)=aT+b$ de degré 1 qui approxime ces données au sens des moindres carrés, on cherche `a` et `b`.

```
--> A=[ones(x), x];

--> A\y
ans =

!    10.139635 !
!    0.0059412 !

--> b =10.139635
--> a= 0.0059412
```

20.6.2 Signal moins la tendance

On considère le signal $yy=y-P(x)$, c'est à dire le signal débarrassé des tendances saisonnières. On cherche à identifier 3 cycles dans le phénomène El Niño, dont on donnera les périodes en années. On fait comme à la section (20)

```

-->yy=y-(sol(1)*ones(x)+sol(2)*x);

-->xset("window",1)

-->subplot(1,2,1)

-->plot2d(x,y,style=2)

-->plot2d(x,sol(1)*ones(x)+sol(2)*x,style=5)

-->subplot(1,2,2)

-->plot2d(x,yy,style=3)

```

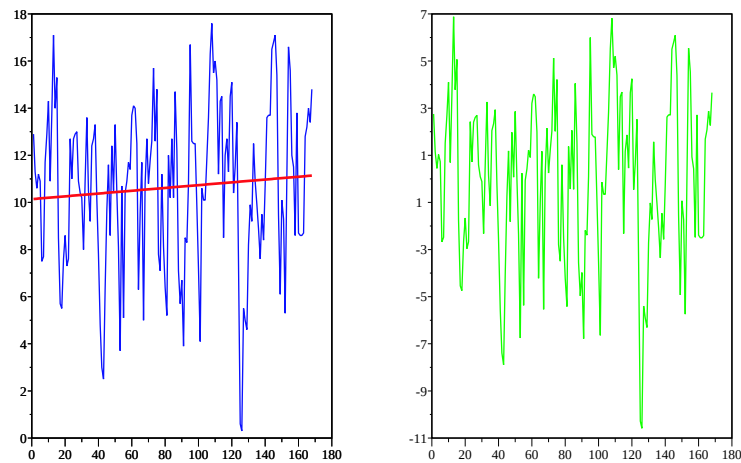


FIG. 20.8 – El niño

```

-->Fe=12;

-->Y=fft(yy,-1);

-->pow=abs(Y).^2;

-->f=(0:n/2)*(Fe/n);

-->Nyquist=Fe/2
Nyquist =

```

6.

```
-->find(f==Nyquist)
ans =
```

85.

```
-->power=pow(1:85);
-->xbasc()

-->plot2d(f,power)
```

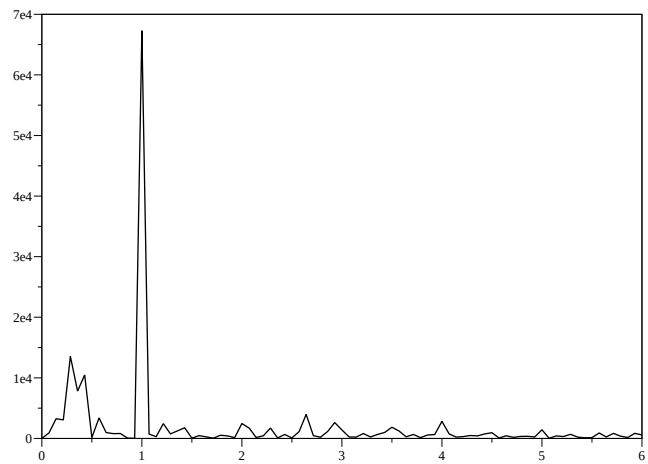


FIG. 20.9 – FFT El niño

On identifie les pics comme précédemment. Les périodes sont les inverses des fréquences.

20.7 Taches solaires

Pendant des siècles il a été observé que la face solaire est ni constante ni uniforme. Des régions plus sombres apparaissent au hasard de façon cyclique. Cette activité a été corrélée avec le climat et d'autres phénomènes terrestres importants économiquement. En 1848 Rudolf Wolfer a proposé une règle qui en combinant la taille et le nombre de ces taches solaires permettait d'obtenir un seul indice. En utilisant les relevés d'archives les astronomes ont appliqué la méthode de Wolfer pour déterminer cet indice depuis 1700. Actuellement cet indice est

mesuré et les résultats sont coordonnés par le centre des influences solaires à l'observatoire royal de Belgique.

<http://sidc.oma.be/sunspot-data/>

Importer le fichier **yearly sunspot number** sur le site précédent. Importer ce fichier en Scilab puis faire une analyse de ces données.

Index

- \$, 30
- abs, 46
- ajustement de données, 57, 73
- analyse de données : fonctions de base, 45
- chdir, 11
- clear, 25
- conjuguée d'une matrice, 39
- dernier indice, 30
- deux points, 27, 30, 34
- diag, 47
- e, 16
- eps, 66
- évaluation d'un polynôme, 62
- evstr, 80
- extraction, 29
- FFT, 110
- fichiers
 - lire un fichier
 - ASCII, 79
 - Excel, 79, 81
- find, 46
- gestion des fenêtres, 98
- graphique
 - fenêtres
 - créer, 97
 - effacer, 97
 - mettre en avant, 98
 - sous-fenêtres, 98
- help, 13
- horner, 62
- $i(\sqrt{-1})$, 16, 20
- ieee, 19, 59, 65, 67
- indexation
 - à deux indices, 29
 - à un seul indice, 30
- interpfn, 88
- interpolation, 88
- kron, 51
- length, 23, 86
- lignes longues, 13
- lister, 13
- ls, 13
- matrice
 - aléatoire, 33
 - de uns, 32
 - de Vandermonde, 54
 - identité, 33
 - nulle, 32
 - Toeplitz, 48
 - vide, 32
- matrices particulières, 34
- matrix, 31
- max, 46
- mgetl, 79
- min, 30
- moindres carrés, 71
- nan, 67
- ne pas afficher, 17
- nom de variable, 16
- nombres flottants, 65
- opérations
 - sur les matrices, 40
 - sur les tableaux, 43

- pi, 16
- plot2d, 99
 - paramètres, 100
- point-virgule
 - fin de ligne d'une matrice, 21
 - pour ne pas afficher, 17
- poly, 61
- polynômes, 60
- produit de Kronecker, 51
- pwd, 10

- racines d'un polynôme, 61
- rand, 33
- rappeler une ligne de commande passée,
13
- redimensionner, 31
- roots, 61

- save, 26
- size, 23
- solution aux moindres carrés, 71
- sous-matrice, 29
- string, 79
- supprimer une ligne, 32

- toeplitz, 48
- Tony's trick, 53
- transformée de Fourier discrète, 109
- transformée de Fourier inverse, 110
- transposée d'une matrice, 38
- tril, 48
- triu, 48

- vecteurs réguliers, 27, 28
- vectorisation, 43

- who, 25