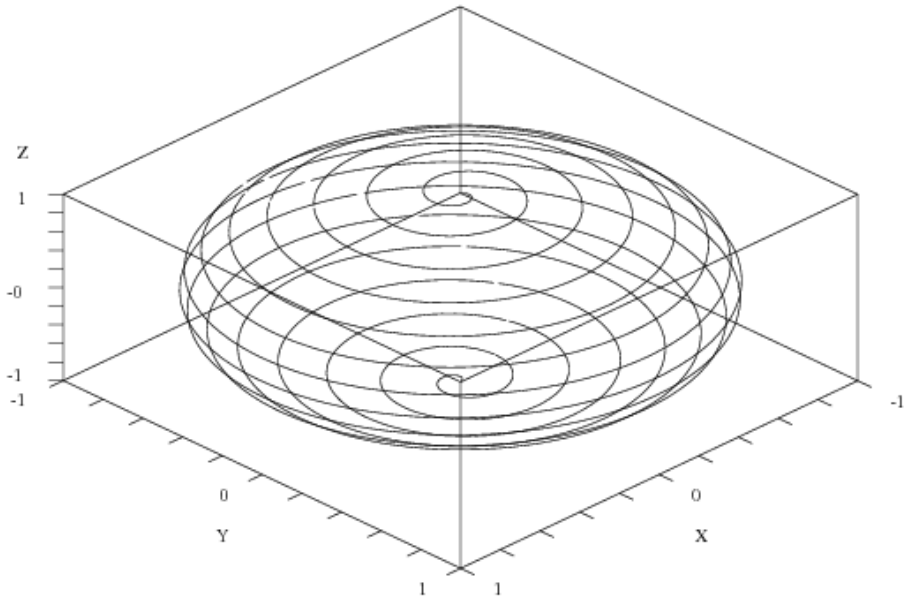
UNIVERSITÉ
PARIS DESCARTES
UFR DE MATHÉMATIQUES ET INFORMATIQUE

DÉMARRER EN SCILAB



B. Ycart, G. Koepfler

Introduction

Scilab (contraction de *Scientific Laboratory*) est un logiciel distribué librement, maintenu et développé par le *Consortium Scilab* depuis la création de cette structure en 2003. Il est téléchargeable gratuitement depuis le site officiel `http://www.scilab.org`. C'est un environnement numérique qui permet d'effectuer rapidement les calculs et représentations graphiques couramment rencontrées en mathématiques. Il ne faut pas attendre de ce cours d'initiation une documentation complète de Scilab, pas plus qu'un manuel de programmation ou un cours d'utilisation avancée. Le but de ce qui suit est d'aider le débutant en introduisant quelques unes des fonctionnalités de base.

Il est conseillé de lire ce document après avoir lancé Scilab, en exécutant les commandes proposées une par une pour en observer l'effet. Les exemples ont été testés sur la version 5.3.0 pour Linux. À l'exception peut-être des exemples utilisant les outils graphiques (qui ont beaucoup évolué), ils devraient fonctionner sur tous supports et toute version du logiciel.

Ce document se trouve aussi sur les pages *Maths en Ligne* de Bernard Ycart à l'*Université Joseph Fourier*, Grenoble : `http://ljk.imag.fr/membres/Bernard.Ycart/mel/`.

Table des matières

1 Cours **1**
- 1.1 Vecteurs et matrices . . . 2
- 1.2 Graphiques . . . 16
- 1.3 Programmation . . . 27
- 1.4 Calcul numérique . . . 31

Index **38**

2 Entraînement **42**
- 2.1 Vrai ou faux . . . 42
- 2.2 Exercices . . . 49
- 2.3 QCM . . . 56
- 2.4 Devoir . . . 59
- 2.5 Corrigé du devoir . . . 61

3 Compléments **66**
- 3.1 Un voyage à Anticythère . . . 66
- 3.2 La pascaline . . . 67
- 3.3 Les machines à calculs . . . 68
- 3.4 La manufacture à logarithmes . . . 69
- 3.5 Que calcule la calculette ? . . . 72

1 Cours

À savoir pour commencer

Une fois Scilab chargé, une première invite de commande, indiquée par `-->`, apparaît dans la fenêtre. Tapez `1+1`, puis Entrée. Le résultat est donné : `ans = 2`.

Pour démarrer, et pour une utilisation ´ légère ˇ, vous rentrerez ainsi des commandes ligne par ligne. Un ´ retour-chariot ˇ Entrée exécute la ligne, sauf dans deux cas :

- si la ligne se termine par deux points, la séquence de commandes se prolonge sur la ligne suivante ;
- si la commande définit une matrice, les lignes de cette matrice peuvent être séparées par des retour-chariot. Ceci sert essentiellement à importer de grandes matrices depuis des fichiers.

Dans une ligne de commande, tout ce qui suit `//` est ignoré, ce qui est utile pour les commentaires. Les commandes que nous proposons sur des lignes successives sont supposées être séparées par des retour-chariot.

```
A=[1,2,3;4,5,6;7,8,9]  // definit une matrice 3X3
A=[1,2,3;4,            // message d'erreur
A=[1,2,3;4,..          // attend la suite de la commande
5,6;7,8,9]             // la meme matrice est definie
A=[1,2,3;              // premiere ligne
4,5,6;                 // deuxieme ligne
7,8,9]                 // fin de la matrice
```

Ajouter un point virgule en fin de ligne supprime l'affichage du résultat (le calcul est quand même effectué). Ceci évite les longs défilements à l'écran, et s'avère vite indispensable.

```
x=ones(1,100);         // rien n'apparait
x                      // le vecteur x a bien ete defini
```

Souvent des commandes doivent être répétées, ou légèrement modifiées. On peut naviguer dans les commandes précédentes avec les touches ↑ et ↓. Une fois rappelée une commande, elle peut être modifiée en se déplaçant dans la ligne : ← ou →. Les résultats sont affectés par défaut à la variable `ans` (´ answer ˇ), qui contient donc le résultat du dernier calcul *non affecté*.

Dans les *noms de variables*, les majuscules sont distinctes des minuscules. Toutes les variables d'une session sont globales et conservées en mémoire. Des erreurs proviennent souvent de confusions avec des noms de variables déjà affectés. Il faut penser à ne pas toujours utiliser les mêmes noms ou à libérer les variables par `clear`. Vos variables sont listées par `who_user`.

```
a=[1,2]; A=[1,2;3,4];  // affecte a et A
```

```
1+1                    // affecte ans
who_user               // vos variables
clear a
who_user               // a disparait
clear
who_user               // a, A et ans disparaissent
```

Faites attention à ne pas utiliser des noms de fonctions prédéfinies.

```
gamma(1)    // appel de la fonction gamma
gamma=3     // warning
gamma(1)    // 1er element de la variable gamma
gamma(4)    // erreur : pas de 4e element dans gamma
clear       // efface les variables de l'utilisateur
gamma(4)    // la fonction gamma est de nouveau disponible
```

L'aide en ligne est appelée par `help`. La commande `apropos` permet de retrouver les rubriques d'aide quand on ignore le nom exact d'une fonction. On peut aussi accéder à l'aide par la touche F1.

```
help help
help apropos
apropos matrix         // rubriques dont le titre contient "matrix"
help matrix            // aide de la fonction "matrix"
```

1.1 Vecteurs et matrices

Tout calcul, programmation ou tracé graphique peut se faire à partir de matrices rectangulaires. En Scilab, tout est matrice : les scalaires sont des matrices 1×1, les éléments de $\mathbb{R}^n$ sont des *vecteurs colonnes*, c'est-à-dire des matrices $n \times 1$ et les *vecteurs lignes* sont des matrices $1 \times n$.

Construire des matrices

On peut saisir manuellement des (petites) matrices. Les coefficients d'une même ligne sont séparés par des blancs ou des virgules (préférable). Les lignes sont séparées par des points-virgules. La transposée est notée par une apostrophe. Elle permet en particulier de changer un vecteur ligne en un vecteur colonne.

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

On a souvent besoin de connaître les dimensions d'une matrice (par exemple pour vérifier si un vecteur est une ligne ou une colonne). On utilise pour cela la fonction `size`. Les options `"r"` ou `"c"` désignent les lignes (rows) ou les colonnes (columns).

La fonction size	
`size(A)`	nombre de lignes et de colonnes
`size(A,"r")`ou `size(A,1)`	nombre de lignes
`size(A,"c")`ou `size(A,2)`	nombre de colonnes
`size(A,"*")`	nombre total d'éléments

```
help size
A=[1,2,3;4,5,6]
size(A)
size(A')
size(A,1)
size(A,"c")
size(A,"*")
```

Les commandes d'itération permettent de construire des vecteurs de nombres séparés par des pas positifs ou négatifs. La syntaxe de base, `[deb:pas:fin]`, retourne un vecteur ligne de valeurs allant de `deb` à `fin` par valeurs séparées par des multiples de `pas`. Les crochets sont facultatifs. Par défaut, `pas` vaut 1. Selon que `fin-deb` est ou non un multiple entier de `pas`, le vecteur se terminera ou non par `fin`.

Si l'on souhaite un vecteur commençant par `deb` et se terminant par `fin` avec `n` coordonnées régulièrement espacées, il est préférable d'utiliser `linspace(deb,fin,n)`.

```
v=0:10
v=0.10
v=[0:10]
v=[0;10]
v=[0,10]              // attention aux confusions
v=[0:0.1:1]
size(v)
v=[0:0.1:0.99]
size(v)
v=[0:0.15:1]
v=[1:-0.15:0]
v=linspace(0,1,10)
v=linspace(0,1,11)
v=linspace(0,1)       // attention, n=100 par defaut
v=logspace(1,2,11)    // puissances de 10
v-10^(linspace(1,2,11))
```

L'élément de la `i`-ième ligne, `j`-ième colonne de la matrice `A` est `A(i,j)`. Si `v` et `w` sont deux vecteurs d'entiers, `A(v,w)` désigne la sous-matrice extraite de `A` en conservant les éléments dont l'indice de ligne est dans `v` et l'indice de colonne dans `w`. `A(v,:)` (respectivement : `A(:,w)`) désigne la sous-matrice formée des lignes (resp. : des colonnes) indicées par les coordonnées de `v` (resp. : `w`). Les coordonnées sont numérotées à partir de 1. Le dernier indice peut être désigné par `$`.

```
A=[1,2,3;4,5,6]
A(2,2)
A(:,2)
A(:,[1,3])
A(1,$)
A($,2)
```

On peut modifier un élément, une ligne, une colonne, pourvu que le résultat reste une matrice. Pour supprimer des éléments, on les affecte à la matrice vide : `[]`

```
A=[1,2,3;4,5,6]
A(1,3)=30
A(:,[1,3])=[10,20]        // erreur
A(:,[1,3])=[10,30;40,60]
A(:,[1,3])=0
A(:,2)=[]
```

Il suffit d'un indice pour repérer les éléments d'un vecteur ligne ou colonne.

```
v=[1:6]
v(3)
v(4)=40
v([2:4])=[]
w=[1:6]'
w([2,4,6])=0
```

Faire appel à un élément dont un des indices dépasse le nombre de lignes ou de colonnes de la matrice provoque un message d'erreur. Cependant on peut étendre une matrice en affectant de nouvelles coordonnées, au-delà de sa taille. Les coordonnées intermédiaires sont nulles par défaut.

```
A=[1,2,3;4,5,6]
A(0,0)           // erreur
A(3,2)           // erreur
A(3,2)=1
v=[1:6]
v(8)             // erreur
v(8) = 1
```

Vecteurs	
`x:y`	nombres de `x` à `y` par pas de 1
`x:p:y`	nombres de `x` à `y` par pas de `p`
`linspace(x,y)`	100 nombres entre `x` et `y`
`linspace(x,y,n)`	`n` nombres entre `x` et `y`
`v(i)`	`i`-ième coordonnée de `v`
`v($)`	dernière coordonnée de `v`
`v(i1:i2)`	coordonnées `i1` à `i2` de `v`
`v(i1:i2)=[]`	supprimer les coordonnées `i1` à `i2` de `v`

Matrices	
`A(i,j)`	coefficient d'ordre `i,j` de `A`
`A(i1:i2,:)`	lignes `i1` à `i2` de `A`
`A($,:)`	dernière ligne de `A`
`A(i1:i2,:)=[]`	supprimer les lignes `i1` à `i2` de `A`
`A(:,j1:j2)`	colonnes `j1` à `j2` de `A`
`A(:,$)`	dernière colonne de `A`
`A(:,j1:j2)=[]`	supprimer les colonnes `j1` à `j2` de `A`
`diag(A)`	coefficients diagonaux de `A`

Si `A`, `B`, `C`, `D` sont 4 matrices, les commandes `[A,B]`, `[A;B]`, `[A,B;C,D]` retourneront des matrices construites par blocs, pourvu que les dimensions coïncident. En particulier, si `v` et `w` sont deux vecteurs lignes, `[v,w]` les concaténera, tandis que `[v;w]` les empilera (s'ils sont de même longueur).

```
A=[1,2,3;4,5,6]
M=[A,A]
M=[A;A]
M=[A,A']                    // erreur
M=[A,[10;20];[7,8,9],30]
M=[A,[10,20];[7,8,9],30]  // erreur
```

Des fonctions prédéfinies permettent de construire certaines matrices particulières.

Matrices particulières	
`zeros(m,n)`	matrice nulle de taille `m,n`
`ones(m,n)`	matrice de taille `m,n` dont les coefficients valent 1
`eye(m,n)`	matrice identité de taille min(`m,n`), complétée par des zéros
`rand(m,n)`	matrice de taille `m,n` à coefficients aléatoires uniformes sur $]0,1[$
`diag(v)`	matrice diagonale dont la diagonale est le vecteur `v`
`diag(v,k)`	matrice dont la k-ième diagonale est le vecteur `v`
`diag(A)`	extrait la diagonale de la matrice `A`
`diag(A,k)`	extrait la k-ième diagonale de la matrice `A`
`triu(A)`	annule les coefficients au-dessous de la diagonale
`tril(A)`	annule les coefficients au-dessus de la diagonale
`toeplitz`	matrices à diagonales constantes

```
d=[1:6]
D=diag(d)
diag(d,1)
A=[1,2,3;4,5,6]
d=diag(A)
B=[A,zeros(2,3);rand(2,2),ones(2,4)]
diag(B,1)
```

```
diag(B,-1)
diag(B,0)
help toeplitz
M=toeplitz([1:4],[1:5])
triu(M)
tril(M)
```

Les commandes `zeros`, `ones`, `eye`, `rand` retournent des matrices dont la taille peut être spécifiée soit par un nombre de lignes et de colonnes, soit par une autre matrice. Si un seul nombre est donné comme argument, ce nombre est compris comme une matrice de taille 1×1.

```
A=ones(5,5)       // 5 lignes et 5 colonnes
rand(A)           // idem
eye(A)            // idem
eye(5)            // 1 ligne et 1 colonne
ones(5)           // idem
rand(size(A))     // 1 ligne et 2 colonnes
```

On dispose de deux moyens pour réordonner des coefficients en une nouvelle matrice. La commande `A(:)` concatène toutes les colonnes de `A` en un seul vecteur colonne (on peut jouer avec la transposée pour concaténer les vecteurs lignes). La fonction `matrix` crée une matrice de dimensions spécifiées, pourvu qu'elles soient compatibles avec l'entrée. On peut même créer avec `matrix` des tableaux à plus de deux entrées (hypermatrices), ce qui ne sert que rarement.

```
v=[1:6]
help matrix
A=matrix(v,2,2)    // erreur
A=matrix(v,3,3)    // erreur
A=matrix(v,2,3)
A=matrix(v,3,2)'
w=A(:)
A=A'; w=A(:)'
H=matrix([1:24],[2,3,4])
```

Un peu d'algèbre permet souvent de construire des matrices particulières à peu de frais, en utilisant le produit, noté `*`, et la transposée. Le produit de Kronecker, `kron` ou `.*.`, peut également être utile.

```
A=ones(4,1)*[1:5]
A=[1:4]'*ones(1,5)
B=[1,2;3,4]
kron(B,B)
kron(B,eye(B))
```

```
kron(eye(B),B)
kron(B,ones(2,3))
kron(ones(2,3),B)
```

Un générateur de nombres pseudo-aléatoires plus performant que `rand` est fourni par la commande `grand`. Dans sa syntaxe, il faut indiquer les dimensions du tableau, la loi de distribution et les paramètres nécessaires :

```
grand(2,2,'bin',100,0.5)
grand(1,10,'uin',-100,100)
help grand
```

Opérations

Les opérations numériques s'effectuent en suivant les ordres de priorité classiques (puissance avant multiplication, et multiplication avant addition). Pour éviter les doutes il est toujours prudent de mettre des parenthèses.

```
2+3*4
(2+3)*4
2^3*4
2^(3*4)
2^3^4
(2^3)^4
```

Toutes les opérations habituelles `+, -, *, ^, /` sont matricielles. Tenter une opération entre matrices de tailles non compatibles retournera en général un message d'erreur, sauf si une des matrices est un scalaire. Dans ce cas, l'opération (addition, multiplication, puissance) s'appliquera terme à terme.

```
A=[1,2,3;4,5,6]
A+ones(1,3)     // erreur
A+ones(A)
A+10
A*10
A*ones(A)       // erreur
A*ones(A')
A'*ones(A)
```

Il n'y a pas d'ambiguïté sur l'addition, mais il est souvent utile de pouvoir appliquer des multiplications terme à terme sans qu'elles soient interprétées comme des produits matriciels.

Dans ce cas on considère que les matrices sont des *tableaux* qui doivent avoir les *mêmes dimensions* afin de garantir un sens aux opérations *terme à terme*. Le symbole d'opérateur, `*` ou `^`, doit alors être précédé d'un point, `.*` ou `.^`

```
A=[1,2,3;4,5,6]
A*A              // erreur
A*A'
A.*A'            // erreur
A.*A
A=[1,2;3,4]
A^3
A.^3             // attention
2.^A
2^A              // attention
A.^A
```

La division est un cas particulier dangereux. Par défaut, la division à droite, `A/B`, calcule une solution d'un système linéaire, `X*B=A`.

```
A=[1,2;3,4]
B=[1,2;2,2]
D=A./B
X=A/B
X*B
```

Si `v` est un vecteur colonne, `1/v` retourne un vecteur ligne `w` tel que `w*v=1`. Ceci engendre des erreurs fréquentes, d'autant que `1./v`, compris comme `(1.0)/v`, retourne la même chose. Il y a plusieurs solutions à ce problème, dont la plus simple consiste à mettre des parenthèses : `(1)./v`

```
v=[1,2,3]'
w=1/v
w*v
1./v
(1)./v
ones(v)./v
v.^(-1)
v=v'
w=1/v
1./v
```

Opérations matricielles	
`+ -`	addition, soustraction
`* ^`	multiplication, puissance (matricielles)
`.* .^`	multiplication, puissance terme à terme
`A\b`	solution de `A*x=b`
`b/A`	solution de `x*A=b`
`./`	division terme à terme

Types de données

Certaines constantes sont prédéfinies, et leur valeur ne peut être modifiée.

Constantes prédéfinies	
`%pi`	3.1415927
`%e`	2.7182818
`%i`	$\sqrt{-1}$
`%eps`	précision machine
`%inf`	infini
`%t`	vrai
`%f`	faux
`%s`	variable de polynôme

Scilab est un langage faiblement typé. Les variables ne sont pas déclarées, et la même lettre peut désigner un réel, un entier, un polynôme, ou une fonction. Les types existent cependant, et peuvent être obtenus par `typeof`. Par défaut, les nombres sont traités comme des réels en double précision. Les calculs ne sont donc pas ´ exacts ˇ. La précision machine est donnée par la constante `%eps`, de l'ordre de 10^{-16}. Cela ne signifie pas que les nombres inférieurs à `%eps` ou supérieurs à `1/%eps` sont inaccessibles, mais Scilab ne peut distinguer deux réels que à `%eps` près.

```
help typeof
%eps
1/%eps
a=1+%eps
typeof(a)
a-1
b=(1+a)/2
b-a
b-1
1/%inf
1/0.00000000000000000000000000000000000003
1/30000000000000000000000000000000000000
A=[1,2,3;4,5,6;7,8,8]
A/A
A*(1/A)
2^[1:20]
2^2^[1:20]
```

La commande `format` permet de modifier l'affichage des réels.

```
a=1/3
format("v",15); a
format("e",5);  a
format("e",15); a
```

```
format("v",25); a
%eps
100000/3
tan(atan(1))
v=[-1:0.1:1];
v(11)
w=linspace(-1,1,21);
w(11)
format('v',10)   // retour aux valeurs par defaut
```

À part les réels, les entrées d'une matrice peuvent être des booléens, des complexes, des polynômes ou des chaînes de caractères.

Les booléens sont `%t` (true) et `%f` (false) pour les saisies, `T` et `F` pour les affichages. L'égalité logique est notée `==` pour la distinguer de l'affectation.

```
a=2==3
b=2<3
c=%t
a|(b&c)
```

Opérateurs logiques

`==`	égalité	`~`	non
`<`	$<$	`>`	$>$
`<=`	$\leqslant$	`>=`	$\geqslant$
`&`	et	`|`	ou

Les matrices booléennes sont le résultat d'opérations logiques matricielles (toujours terme à terme). Par exemple, si `A` et `B` sont deux matrices (réelles) de même taille, `A<B` est la matrice des booléens `A(i,j)<B(i,j)`. Les opérateurs `and` et `or` peuvent s'appliquer à l'ensemble des éléments d'une matrice booléenne, à ses lignes ou à ses colonnes selon l'option choisie. La commande `find(v)` retourne la liste des indices des coordonnées ´ vraies ˇ du vecteur booléen `v`. Si `v` est une matrice, ses colonnes sont préalablement concaténées. Si `A` est une matrice et `B` est une matrice de booléens, la commande `A(B)` extrait de `A` les coordonnées correspondant aux indices pour lesquels `B` est à vrai. La fonction `bool2s` transforme des booléens en 0 ou 1.

```
x=rand(2,10)
b=x<0.5
bool2s(b)
and(b)
and(b,"c")
and(b,"r")
or(b)
or(b,"r")
b1=b(1,:); b2=b(2,:);
```

```
b1 & b2
b1 | b2
b1
find(b1)
y=[1:10]
y(b1)
A=[1,2,3;4,5,6;7,8,9]
x=[%t,%f,%t]
A(x,x)
A(~x,x)
B=rand(3,3)<0.5
A(B)
```

Les complexes sont définis à l'aide de la constante `%i` ($\sqrt{-1}$) ou bien en affectant `sqrt(-1)` à une nouvelle variable. A noter que par défaut `A'` est la transposée *conjuguée* de `A`. La transposée non conjuguée est `A.'`. Les fonctions classiques appliquées à un complexe donnent toujours un résultat unique, même si mathématiquement elles ne sont pas définies de façon unique (racine carrée, logarithme, puissances).

```
A=[1,2;3,4]+%i*[0,1;2,3]
real(A)
imag(A)
conj(A)
A'
A.'
abs(A)
phasemag(A)
i=sqrt(-1)
%i*i
A*i
%e^(%i*%pi)
x=log(%i)
exp(x)
x=%i^(1/3)
x^3
```

Complexes	
`real`	partie réelle
`imag`	partie imaginaire
`conj`	conjugué
`abs`	module
`phasemag`	argument (en degrés)

Les polynômes et les fractions rationnelles constituent un type de données particulier. On peut construire un polynôme comme résultat d'opérations sur d'autres polynômes. Il existe par défaut un polynôme élémentaire, `%s`. On peut construire un polynôme d'une variable quelconque à l'aide de la fonction `poly`, en spécifiant soit ses racines (par défaut), soit ses coefficients. Les fractions rationnelles sont des quotients de polynômes. Par défaut, Scilab effectue automatiquement les simplifications qu'il reconnaît. Pour évaluer la fonction polynôme associée à un polynôme on utilise la fonction `horner`. Ceci permet une diminution du nombre d'opérations et évite la manipulation de nombres très grands.

```
apropos poly
help poly
v=[1,2,3]
p1=poly(v,"x")
roots(p1)
p2=poly(v,varn(p1),"c") // meme variable que p1
coeff(p2)
horner(p2,0)            // evaluation en x=0
horner(p2,1)            // evaluation en x=1
horner(p2,2)            // evaluation en x=2
p1+p2
p3=1+2*%s-3*%s^2
p1+p3                   // erreur : variables differentes
p4=(%s-1)*(%s-2)*(%s-3)
p3/p4
// Racines complexes : racines n-ième de l'unité
n = 6
v = [-1, zeros(1,n-1), 1];
pn = poly(v,"x","c")
R = roots(pn)        // observer les couples de racines conjuguées
phasemag(R)*%pi/180  // arguments des racines, convertis en radians
factors(pn)          // regroupement des racines conjuguées
```

Polynômes	
`poly(v,"x")`	polynôme unitaire dont les racines sont les éléments de `v`
`poly(v,"x","c")`	polynôme dont les coefficients sont les éléments de `v`
`inv_coeff(v)`	idem
`coeff(P)`	coefficients du polynôme `P`
`roots(P)`	racines du polynôme `P`
`factors`	facteurs irréductibles réels d'un polynôme
`horner`	évaluation d'un polynôme
`varn(P)`	indéterminée du polynôme `P`

Les chaînes de caractères, encadrées par des apostrophes, en anglais *quotes*, (`"..."` ou `'...'`), permettent de définir des expressions mathématiques, interprétables ensuite

comme des commandes à exécuter ou des fonctions à définir. Elles servent aussi d'intermédiaire pour des échanges de données avec des fichiers. On peut transformer et formater des nombres en chaînes de caractères (voir `apropos string`).

```
expression=["x+y","x-y"]      // matrice de 2 chaines de caracteres
length(expression)            // longueurs des 2 chaines
size(expression)              // taille de la matrice
part(expression,2)            // deuxieme caractere de chaque chaine
expression(1)+expression(2)   // concatenation des deux chaines
x=1; y=2;
evstr(expression)
x=1; instruction="y=(x+1)^2"
execstr(instruction)
y
deff("p=plus(x,y)","p=x+y")
plus(1,2)
plus(2,3)
```

Chaînes de caractères	
`evstr`	évaluer une expression
`deff`	définir une fonction
`execstr`	exécuter une instruction
`length`	longueur
`part`	extraire
`+`	concaténer
`string`	transformer en chaîne

Fonctions

Scilab propose beaucoup de fonctions dans des domaines très variés. On peut en obtenir la description par `help`. On peut retrouver une fonction avec `apropos`, qui retourne les rubriques d'aide dont le titre contient une chaîne de caractères donnée. De très nombreux exemples sont disponibles en démonstration, on y accède par le menu ? $\longrightarrow$ Démonstrations

Les fonctions numériques s'appliquent en général à chaque terme d'un vecteur ou d'une matrice. Il ne faut pas oublier que la multiplication, la puissance, la division doivent être précédées par un point pour s'appliquer terme à terme.

```
x=1:10
y=sin(x)
y=x*sin(x)       // erreur
y=x.*sin(x)
y=1./x.*sin(x)   // erreur
```

```
y=(1)./x.*sin(x)
y=sin(x)./x
A=[ 4, 4; 4, 4]
sqrt(A)
A.^(1/2)          // operation terme a terme
B=A^(1/2)         // operation matricielle
B*B
```

Fonctions mathématiques			
sqrt	exp	log	round
sin	cos	tan	floor
asin	acos	atan	ceil

Les fonctions vectorielles s'appliquent à l'ensemble d'un vecteur ou d'une matrice, et retournent un scalaire. Pour appliquer une telle fonction colonne par colonne ou ligne par ligne, il faut rajouter l'option `"r"` ou `"c"`. Il n'est pas toujours évident de décider laquelle des deux options choisir. Il faut se souvenir qu'avec l'option `"r"`, la fonction retournera un vecteur ligne (row), et donc s'appliquera aux colonnes. Par exemple, `sum(A,"r")` retourne un vecteur ligne, qui est formé des sommes des coefficients dans chaque colonne.

Fonctions vectorielles	
max	maximum
min	minimum
sort	tri par ordre décroissant
gsort	tri, ordres particuliers
sum	somme
prod	produit
cumsum	sommes cumulées
cumprod	produits cumulés
mean	moyenne
median	médiane
stdev	écart-type

```
A=[1,2,3;4,5,6]
sum(A)
sum(A,"r")
sum(A,"c")
cumsum(A,"r")
cumsum(A,"c")
cumsum(A)
x=rand(1,5)
mean(x)
```

```
stdev(x)
median(x)
sort(x)
gsort(x,"c","i")
```

L'utilisation de Scilab consiste en général à étendre le langage par de nouvelles fonctions, définies par des séquences d'instructions. Nous avons vu l'utilisation de `deff`, qu'il convient de réserver à des définitions courtes. L'alternative à `deff`, est de définir une fonction sur plusieurs lignes dans la fenêtre Scilab. La première ligne est nécessairement du type `function y=f(x)`, la dernière est `endfunction`. Si la fonction à écrire est un tant soit peu compliquée il vaut mieux écrire un fichier de fonction externe, comme nous le verrons plus loin. Cela permet une programmation beaucoup plus claire et un débuggage plus efficace.

Il est important de choisir des noms différents pour les nouvelles fonctions, sans quoi les définitions se superposeront, y compris à celles des fonctions prédéfinies.

```
deff("y=sin(x)","y=2*x")      // message d'avertissement
sin(2)
clear
sin(2)
```

Les nouvelles fonctions sont traitées comme des variables, à la différence des fonctions prédéfinies (primitives) : une fonction définie par `deff` peut être utilisée comme argument dans `fplot2d` mais une primitive ne peut pas.

```
deff("z=pente_secante(f,x,y)","z=(f(y)-f(x))/(y-x)")
x=%pi; y=%pi+0.01;
pente_secante(cos,x,y)        // correct
deff("y=f(x)","y=cos(x)")
pente_secante(f,x,y)
z=[0:0.01:%pi];
fplot2d(z,cos)                // erreur
fplot2d(z,f)
```

Quand on définit une nouvelle fonction numérique, on a toujours intérêt à faire en sorte qu'elle puisse s'appliquer correctement à une matrice, ce qui impose de veiller aux multiplications terme à terme. On peut aussi utiliser `feval`, qui distribue l'évaluation d'une fonction sur l'ensemble des éléments d'un vecteur.

```
deff("y=f(x)","y=x*sin(x)")
f(1)
f([1:5])                      // erreur
help feval
feval([1:5],f)
deff("y=g(x)","y=x.*sin(x)")
g(1)
g([1:5])
```

Certaines fonctions peuvent retourner plus d'un argument. Par exemple les fonctions de tri `sort` et `gsort` retournent par défaut le vecteur trié, mais peuvent aussi donner la permutation des coordonnées qui a été effectuée. Les fonctions `max` et `min` donnent aussi la position des maxima et minima dans une matrice.

```
v=rand(1,5)
max(v)
[m,i]=max(v)
sort(v)
[vtrie,perm]=sort(v)
```

1.2 Graphiques

Le principe général des représentations graphiques est de se ramener à des calculs sur des matrices ou des vecteurs. Ainsi la représentation d'une fonction de $\mathbb{R}$ dans $\mathbb{R}$ commencera par la création d'un vecteur d'abscisses, en général régulièrement espacées, auxquelles on applique la fonction pour créer le vecteur des ordonnées. Pour la représentation d'une surface, il faudra créer la matrice des valeurs de la fonction sur une grille rectangulaire dans $\mathbb{R}^2$.

Il est impossible de décrire ici l'ensemble des fonctions graphiques et leurs multiples options. Certaines de ces options, comme la numérotation des couleurs, sont globales et peuvent être fixées par `set` (voir `help set`). Les démonstrations donnent une bonne idée des possibilités graphiques de Scilab. On obtient en général un exemple d'utilisation d'une fonction graphique en appelant cette fonction à vide. L'interface graphique permet de *zoomer* sur une partie d'un graphe 2D et d'effectuer des *rotations* sur les images 3D à l'aide de la souris.

On ne présentera ici que les fonctions graphiques propres à Scilab. Pour la compatibilité avec Matlab©, voir `help plot`.

Par défaut, les graphiques successifs sont superposés sur la même fenêtre. On efface la fenêtre courante par `clf()` On ouvre la fenêtre numéro `i` par `scf(i)`. On l'efface avec `clf(i)`.

```
help Graphics
plot2d1()
clf()
histplot()
clf()
plot3d()
clf()
hist3d()
clf()
param3d()
```

Avec `scf()` on crée une nouvelle fenêtre graphique et `xdel(i)` supprime la fenêtre numéro `i`.

```
clf()
plot2d1()
scf()
histplot()
scf()
plot3d()
clf(1)            // efface la figure 1
scf(1)            // la figure 1 est figure courante
hist3d()
xdel(winsid())    // supprime toutes les fenetres graphiques ouvertes
```

La commande `winsid()` renvoie la liste des fenêtres graphiques ouvertes.

Représenter des fonctions

Le plus simple pour commencer est de tracer le graphe d'une fonction de $\mathbb{R}$ dans $\mathbb{R}$ à l'aide de `plot2d`. On crée pour cela un vecteur `x` d'abscisses, et on prend l'image de ce vecteur par la fonction pour créer un vecteur `y` d'ordonnées. La commande `plot2d(x,y)` représente les points de coordonnées `(x(i),y(i))` en les joignant par des traits noirs (par défaut), ou selon un autre style, si le style de base a été changé. La qualité de la représentation dépend donc du nombre de points.

```
x=linspace(0,3*%pi,10); y=x.*sin(x);
plot2d(x,y)
clf()                                  // sinon superposition
x=linspace(0,3*%pi,100); y=x.*sin(x);
plot2d(x,y)
```

On obtient le même résultat par `fplot2d`, mais il faut pour cela prédéfinir la fonction à représenter. Les tracés successifs se superposent.

```
deff("y=f(x)","y=x.*sin(x)")
x=linspace(0,3*%pi,10);
fplot2d(x,f)
x=linspace(0,3*%pi,100);
fplot2d(x,f)                       // superposition des deux graphiques
```

Quand on veut superposer plusieurs courbes avec les mêmes échelles de représentation, il est préférable d'utiliser `plot2d`, qui autorise des styles différents pour chaque courbe. La syntaxe générale est la suivante.

```
plot2d(abscisses,ordonnees [,clef1=valeurs1 [,clef2=valeurs2]] )
```

Après les deux premiers, les arguments sont facultatifs, les ´ clefs ˇ indiquent les paramètres du graphique que l'on désire fixer à ´ valeurs ˇ, voir la suite.

Signification des arguments :

• `abscisses, ordonnees` : Si ce sont des vecteurs (une seule courbe à tracer), ils peuvent être ligne ou colonne. Si plusieurs courbes doivent être tracées, soit `abscisses` est un vecteur colonne et `ordonnees` une matrice ayant le même nombre de lignes (*i.e.* nombre de points) et un nombre de colonnes donné (*i.e.* le nombre de courbes) ; soit ce sont des matrices de mêmes dimensions : à chaque vecteur colonne ´ abscisse ˇ correspond un vecteur colonne ´ ordonnée ˇ de même taille.
Par défaut les points seront reliés par des segments. À chaque courbe correspond une couleur de la palette : il y en a 32 ; `getcolor` affiche la palette.

```
x=linspace(0,3.5*%pi,30);
y=x.*sin(x);
plot2d(x,y)
clf()
y2=2*y;
plot2d([x,x],[y,y2])   // incorrect : deux courbes concatenees
clf()
plot2d([x;x],[y;y2])   // incorrect : trace 30 segments
clf()
plot2d([x;x]',[y;y2]') // correct
clf()
plot2d(x',[y;y2]')     // correct
clf()
X=x'*ones(1,20);
Y=y'*[1:20];
plot2d(X,Y)
```

Les clefs et une partie des valeurs possibles sont présentées ci-dessous, sous la forme `clef=valeur`. Pour plus de détails se référer à l'aide en ligne de `plot2d`.

• `style=vect_style` : `vect_style` est un vecteur ligne dont la dimension est le nombre de courbes à tracer, c'est-à-dire le nombre de colonnes des matrices `abscisses` et `ordonnees`. Les coordonnées sont positives ou négatives. Si le style est positif, les points sont joints par des segments. Si le style est nul, les points sont affichés comme des pixels noirs. Si le style est négatif, des marques de formes particulières sont affichées.

```
x=linspace(0,3*%pi,30); X=x'*ones(1,10);
y=x.*sin(x); Y=y'*[1:10];
couleurs=matrix([2;5]*ones(1,5),1,10)
clf()
plot2d(X,Y,style=couleurs)
marques=-[0:9]
clf()
plot2d(X,Y,style=marques)
```

• `rect=vect_rect` : `vect_rect` est le rectangle de représentation, décrit par les deux coordonnées du coin inférieur gauche, suivies des deux coordonnées du coin supérieur droit : `[xmin,ymin,xmax,ymax]`.

```
x=linspace(0,3*%pi,30);
y=x.*sin(x);
clf()
plot2d(x,y)
scf()
plot2d(x,y,rect=[0,-10,9.5,10])
```

• `frameflag=int_ff` : `int_ff` est un entier entre 0 et 8 qui code le calcul des échelles en abscisse et ordonnée.
Pour `frameflag=0` on utilise les échelles du graphique précédent ;
pour `frameflag=1` on calcule les échelles grâce à `vect_rect` ;
pour `frameflag=2` on détermine les échelles optimales à partir des données `abscisses`, `ordonnees`. pour `frameflag=6` on détermine les échelles optimales à partir des données `abscisses`, `ordonnees` mais de façon à avoir de ´ jolies ˇ graduations.

• `axesflag=int_ax` : `int_ax` est un entier entre 0 et 5 qui code le tracé des axes et du cadre.
Pour `axesflag=0` rien n'est tracé ;
pour `axesflag=1` les axes sont tracés avec les ordonnées à gauche ;
Pour `axesflag=2` un cadre est tracé autour du graphe.

```
x=linspace(0,3*%pi,30);
y=x.*sin(x);
clf()
plot2d(x,y)
y2=2*y;
plot2d(x,y2,style=2)          //  echelles differentes adaptees
scf()
plot2d(x,y)
plot2d(x,y2,style=2,axesflag=0, frameflag=0)
                              // on utilise les axes precedents
scf()
plot2d(x,y2,style=2)
plot2d(x,y,axesflag=0, frameflag=0)
                              // on utilise les axes precedents
```

• `leg=legendes` : c'est une chaîne de caractères contenant les différentes légendes, séparées par `@`.

```
x=linspace(0,3*%pi,30); X=x'*ones(1,5);
y=x.*sin(x); Y=y'*[1:5];
```

```
vect_styles=[-2:2]
legendes="x sin(x)@2 x sin(x)@3 x sin(x)@4 x sin(x)@5 x sin(x)"
clf()
plot2d(X,Y,style=vect_styles,leg=legendes)
scf() // meme affichage
plot2d(X,Y,style=vect_styles, frameflag=6, axesflag=1,leg=legendes)
```

Les commandes `captions` et `legend` sont plus flexibles, donc préférables.

• `nax=graduations` : `graduations` est un vecteur de quatre entiers qui permet de préciser le nombre des graduations et sous-graduations en abscisse et ordonnée. Par exemple, avec `[4,11,4,5]`, l'intervalle des abscisses sera divisé en 10 intervalles, *i.e.* 11 graduations, chacun des 10 intervalles étant subdivisé par 4 sous-graduations, *i.e.* 5 sous-intervalles. Pour les ordonnées il y aura 4 intervalles, chacun subdivisé en 5 sous-intervalles.

```
x=linspace(0,3*%pi,30);
y=x.*sin(x);
clf()
plot2d(x,y, rect=[0,-10,10,10])
scf()
plot2d(x,y, rect=[0,-10,10,10],nax=[4,11,4,5])
```

• `logflag=ch_log` : `ch_log` est une chaîne de deux caractères parmi : `"nn"` (défaut), `"nl"`, `"ln"` et `"ll"`, où `n` indique l'échelle habituelle, tandis que `l` indique que l'on utilise une échelle logarithmique. Ainsi `logflag="nl"` utilise en abscisse l'échelle normale et en ordonnée l'échelle logarithmique.

```
N=[2:100000]';
C1=N.*log(N);
C2=N.^2;
clf()
plot2d(N,[ C1 C2],style=[2,3],leg="N ln(N) @ N^2" )
scf()
plot2d(N,[ C1 C2],style=[2,3], logflag="ll",leg="N ln(N) @ N^2")
```

Graphes de fonctions	
`plot2d`	plusieurs courbes avec styles différents
`plot2d1`	idem, avec plus d'options
`plot2d2`	représentation en escalier
`plot2d3`	barres verticales
`plot2d4`	flèches
`fplot2d`	représenter des fonctions

Les échelles de représentation, choisies automatiquement, ne sont pas les mêmes en abscisse et en ordonnée. On peut corriger ceci grâce à l'option `frameflag` de `plot2d` ou à l'aide des commandes `isoview` et `square`.

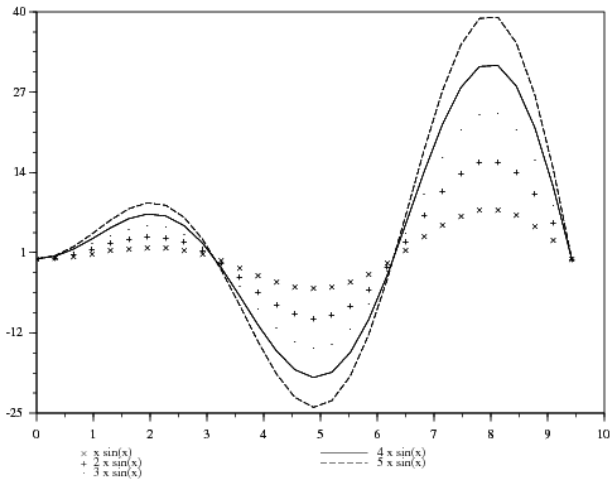


FIG. 1 – Représentation de plusieurs courbes.

Graphiques composés

Si une même représentation graphique comporte plusieurs tracés, on spécifiera d'abord le rectangle de représentation et les échelles des axes lors du premier tracé, pour ensuite superposer les tracés suivants.

```
clf();
x=linspace(-%pi,%pi,50); y=sin(x);
plot2d(x,y,rect=[-4,-1,4,1],nax=[2,9,5,9],axesflag=2);
                                          // trace une courbe
xtitle( '$\mbox{Titre pour }y=\sin(x)$', '$x$', '$y$')
                                          // titre et legendes en LaTeX
x=linspace(-%pi/2,%pi/2,5); y=sin(x);
plot2d(x,y,style=-3, frameflag=0);       // affiche 5 marques
x=linspace(-%pi,%pi,20); y=sin(x)/2;
xfpoly(x,y,color(200,200,200));
                        // surface remplie en gris, couleur (200,200,200)
```

Il est fréquent qu'un graphique contienne non seulement une ou plusieurs représentations de fonctions, mais aussi des chaînes de caractères, des rectangles, ellipses ou autres ajouts graphiques. Les coordonnées de ces ajouts sont relatives à la fenêtre courante.

```
clf()
x=linspace(-1,2,100); y=x.*x;
plot2d(x,y,rect=[-1,0,2,4],nax=[10,4,5,5],style=2)
                                       // represente la courbe
xtitle( 'Parabole', '$x$', '$f(x)$')  // titre et legendes en LaTeX
```

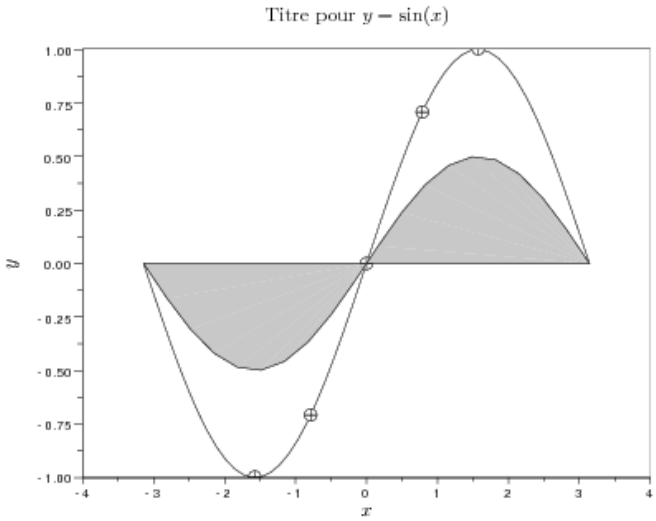


FIG. 2 – Figure composée.

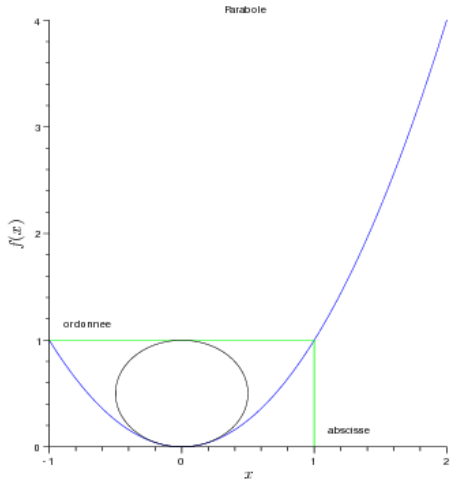


FIG. 3 – Figure composée.

```
plot2d([1,1,-1],[0,1,1],style=3) // trace deux segments
help xstring
xstring(1.1,0.1,"abscisse")      // chaine de caracteres
xstring(-0.9,1.1,"ordonnee")     // autre chaine
help xarc
xarc(-0.5,1,1,1,0,360*64)        // trace un cercle
```

Ajouts sur graphique	
`xarc`	arc d'ellipse
`xfarc`	arc d'ellipse plein
`xarrows`	flèches
`xnumb`	nombres
`xpoly`	polygone
`xfpoly`	polygone plein
`xfpolys`	polygones plein
`xrpoly`	polygone régulier
`xrect`	rectangle
`xfrect`	rectangle plein
`xstring`	chaîne de caractères (à partir d'un point)
`xstringb`	chaîne de caractères (dans un rectangle)
`xtitle`	titre du graphique et des axes

Des fonctions prédéfinies permettent d'effectuer des représentations planes particulières, comme des histogrammes, des projections de surfaces par courbes de niveau ou niveaux de gris, ou des champs de vecteurs. Les exemples qui suivent concernent la fonction de $\mathbb{R}^2$ dans $\mathbb{R}$ qui à (x, y) associe $\sin(xy)$ (voir figures 4, 5 et 6).

```
//
// Courbes de niveau
//
x=linspace(-%pi,%pi,50);      // vecteur d'abscisses
y=x;                          // vecteur d'ordonnees
z=sin(x'*y);                  // matrice des valeurs de la fonction
help contour2d
clf()
contour2d(x,y,z,4);           // trace 4 courbes de niveau
//
// Surface par niveaux de couleurs
//
f=scf()
grayplot(x,y,z)               // pas vraiment gray le plot
f.color_map = graycolormap(32);
//
// Champ de vecteurs tangents
//
clf()
x=linspace(-%pi,%pi,10);      // vecteur d'abscisses
y=x;                          // vecteur d'ordonnees
fx=cos(x'*y)*diag(y);         // matrice des abscisses de vecteurs
fy=diag(x)*cos(x'*y);         // matrice des ordonnees de vecteurs
champ(x,y,fx,fy)              // champ des vecteurs
```

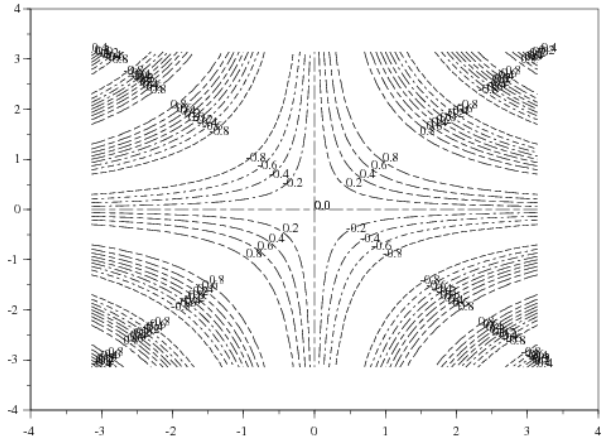


FIG. 4 – Représentation par courbes de niveau.

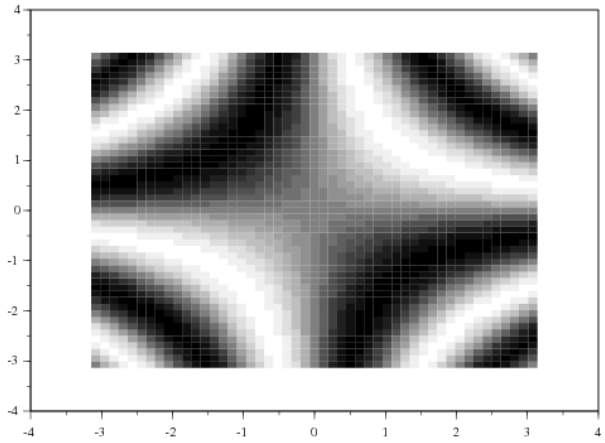


FIG. 5 – Représentation par niveaux de gris.

Représentations planes particulières	
`histplot`	histogramme
`champ`	champ de vecteurs
`fchamp`	idem, définition par une fonction
`grayplot`	surface par rectangles de couleurs
`fgrayplot`	idem, définition par une fonction
`contour2d`	courbes de niveaux projetées
`fcontour2d`	idem, définition par une fonction

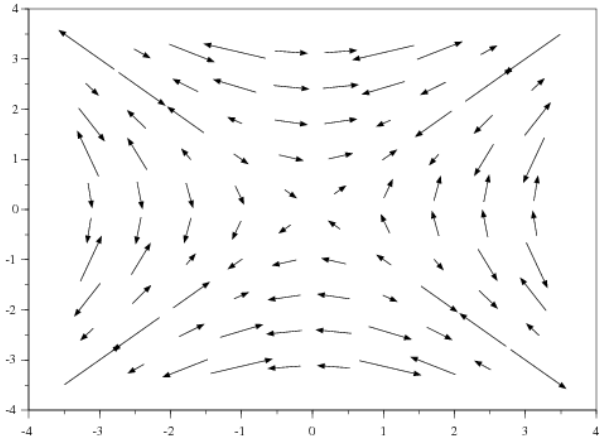


FIG. 6 – Représentation d'un champ de vecteurs.

Dimension 3

Le tracé d'une courbe en dimension 3 se fait par la fonction `param3d`, selon les mêmes principes qu'en dimension 2.

```
clf()
t=linspace(0,2*%pi,50);
x=sin(t); y=sin(2*t); z=sin(3*t);
param3d(x,y,z)                // courbe de Lissajous
scf()                         // nouvelle fenetre
t=linspace(-%pi/2,%pi/2,1000);
x=cos(t*50).*cos(t);
y=sin(t*50).*cos(t);
z=sin(t);
param3d(x,y,z)                // helice spherique
```

Pour représenter une famille de courbes en dimension 3, il faut utiliser `param3d1`. Les arguments sont trois matrices de coordonnées pour lesquelles les différentes courbes sont en colonne.

```
clf()
t=linspace(0,2*%pi,100);
a=linspace(-%pi,%pi,10);
X=cos(t')*cos(a);             // matrice des abscisses
Y=sin(t')*cos(a);             // matrice des ordonnees
Z=ones(t')*sin(a);            // matrice des cotes
param3d1(X,Y,Z)               // paralleles d'une sphere
```

La représentation des surfaces se fait par `plot3d` ou `plot3d1`. Nous reprenons comme exemple la fonction de $\mathbb{R}^2$ dans $\mathbb{R}$ qui à (x, y) associe $\sin(xy)$ (figure 7).

```
x=linspace(-%pi,%pi,50);      // vecteur d'abscisses
y=x;                          // vecteur d'ordonnees
z=sin(x'*y);                  // matrice des valeurs de la fonction
help plot3d
clf()
plot3d(x,y,z)                 // representation monochrome
f=scf()
plot3d1(x,y,z)                // representation coloree
f.color_map = coolcolormap(32);
f.color_map = rainbowcolormap(32);
f.color_map = autumncolormap(32);
```

Pour représenter une surface définie par deux paramètres, il faut la définir comme une fonction, puis utiliser `eval3dp` qui prend comme argument cette fonction et deux vecteurs de paramètres, et retourne les arguments nécessaires pour la représentation par `plot3d`. Voici par exemple la représentation d'une sphère. Pour obtenir des couleurs variables, il faut parfois changer le sens d'un des deux vecteurs de paramètres.

```
deff("[x,y,z]=sphere(u,v)",.. // definition de la fonction
["x=cos(u).*cos(v);..         // abscisses
y=sin(u).*cos(v);..           // ordonnees
z=sin(v)"])                   // cotes

u = linspace(-%pi,%pi,50);
v = linspace(-%pi/2,%pi/2,25);// parametres
[x,y,z] = eval3dp(sphere,u,v);// calcul de la surface
plot3d1(x,y,z);               // representation monochrome

u = linspace(%pi,-%pi,50);    // changement de sens
[x,y,z] = eval3dp(sphere,u,v);// nouveau calcul
scf()
plot3d1(x,y,z)                // les couleurs dependent de z
```

Dimension 3	
`param3d`	courbes paramétriques
`param3d1`	plusieurs courbes ou points
`plot3d`	surface en dimension 3
`fplot3d`	idem, définition par une fonction
`plot3d1`	surface par niveaux de couleurs
`fplot3d1`	idem, définition par une fonction
`eval3dp`	surface paramétrée
`hist3d`	histogrammes

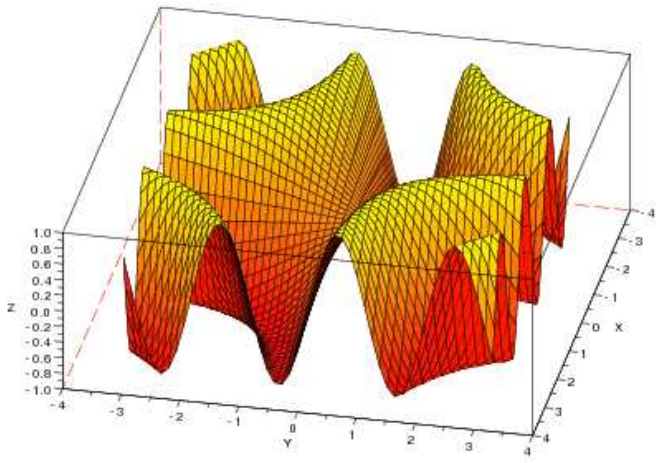


FIG. 7 – Représentation par `plot3d1` et avec rotation.

1.3 Programmation

Types de fichiers

Scilab travaille à partir d'un répertoire de base, qui est donné par la commande `pwd`. C'est là qu'il va chercher par défaut les fichiers à charger ou à exécuter. On peut le changer par la commande `chdir`. À défaut, il faut saisir le chemin d'accès complet du fichier que l'on souhaite charger ou sauvegarder. Le plus facile est d'utiliser le menu de l'interface Fichier → Changer le répertoire courant

Il convient de distinguer trois sortes de fichiers.

- *Les fichiers de sauvegarde.*
  Ce sont des fichiers binaires, créés par la commande `save` et rappelés par `load`. Ceci permet de reprendre un calcul en conservant les valeurs déjà affectées. On peut aussi enregistrer des variables dans un fichier texte par `write` et les rappeler par `read`.
- *Les fichiers de commandes.*
  Ce sont des fichiers texte pour lesquels nous recommandons l'extension `.sce`. Ils contiennent des suites d'instructions Scilab, qui sont exécutées successivement par `exec`. Enregistrez dans le répertoire courant les trois lignes suivantes sous le nom `losange.sce`. Attention, la dernière ligne du fichier doit obligatoirement se terminer par un retour-chariot pour être prise en compte.

  ```
  x=[0,-1,0,1;-1,0,1,0]
  y=[-1,0,1,0;0,1,0,-1]
  plot2d(x,y)
  ```

  La commande `exec("losange.sce")` affichera `x`, puis `y`, puis tracera un losange.

  On peut écrire une matrice dans un fichier texte à exécuter (par exemple pour

importer des données issues d'un tableur). Dans ce cas, les lignes du fichier, si elles correspondent aux lignes de la matrice, ne doivent pas se terminer par deux points. Par exemple le fichier `saisie.sce` peut contenir les trois lignes (terminées par un retour-chariot) :

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

La commande `exec("saisie.sce")` affectera la matrice `A`.

– *Les fichiers de fonctions.* Comme les fichiers de commandes, ce sont des fichiers texte, pour lesquels nous recommandons l'extension standard `.sci`. Ils contiennent la définition d'une ou plusieurs fonctions. Ces fonctions sont disponibles pour Scilab après les avoir chargées et compilées par `exec`.
La définition d'une fonction commence obligatoirement par une ligne qui déclare le nom de la fonction, les variables d'entrée `x1,...,xm` et le vecteur des variables de sortie `[y1,...,yn]`.

```
function [y1,...,yn] = nom_de_la_fonction(x1,...,xm)
      ⋮
 instructions
      ⋮
endfunction
```

Là encore, il ne faut pas oublier de terminer la dernière ligne par un retour-chariot. Enregistrez par exemple dans le fichier `cloche.sci` les lignes suivantes.

```
function d = cloche(x)
//
//     densite de la loi normale N(0,1)
//
coef=1/sqrt(2*%pi)
d = coef*exp(-x.^2/2);
endfunction
```

Si ce fichier est placé dans le répertoire courant, `exec("cloche.sci")` charge et compile la nouvelle fonction.

```
clear
exec("cloche.sci")
t=[-3:0.1:3]; y=cloche(t);
d                          // erreurs : d variable interne
x                          //           x variable interne
coef                       //        coef variable interne
plot2d(t,y)
fplot2d(t,cloche)
intg(-5,1.96,cloche)
```

Il n'y a pas de raison structurelle pour distinguer les fichiers de commande (extension

`.sce`) des fichiers de fonctions (extension `.sci`). Vous pourriez insérer des définitions de fonctions dans des suites de commandes, mais nous recommandons néanmoins de séparer les fonctions des commandes, de manière à mieux structurer votre programmation et à constituer vos propres librairies de fonctions.

Dès que les calculs à effectuer requièrent plus de quelques lignes de commande, on a intérêt à saisir ces lignes dans un fichier exécutable externe. Dans l'interface de Scilab, les seules commandes qui apparaîtront seront les exécutions ou les chargements répétés de fichiers externes. Il est conseillé de maintenir ouvertes deux fenêtres : la fenêtre Scilab, et une fenêtre d'édition, de préfénce l'éditeur de Scilab : *SciNotes* accessible par Applications $\longrightarrow$ Scinotes. Les lignes écrites dans Scinotes seront chargées et compilées par le menu Exécuter de SciNotes, on peut charger (Load) ou évaluer (Evaluate) le contenu. Le menu Fichier $\longrightarrow$ Exécuter... de la console Scilab permet d'exécuter le contenu d'un fichier sans passer par l'éditeur.

Autant pour les fichiers de commande que pour les fichiers de fonctions, une présentation claire et abondamment commentée est vivement recommandée. Pour une utilisation courante ou de mise au point, les fichiers de commandes permettent de suivre le contenu de toutes les variables. Pour une programmation plus avancée, il est préférable de définir des fonctions, car leurs variables internes restent locales. Un même fichier `.sci` peut contenir plusieurs fonctions. Les fonctions du langage sont regroupées dans des librairies qui contiennent leur code Scilab (fichiers texte `.sci`), et leur code compilé (fichiers `.bin`). On peut transformer un ensemble de fichiers de fonctions en librairie, en sauvant les versions compilées et en rajoutant les fichiers d'aide.

Style de programmation

La philosophie de Scilab est celle d'un langage fonctionnel. Au lieu de créer un logiciel avec programme principal et procédures, on étend le langage par les fonctions dont on a besoin. Le rôle du programme principal est joué par un fichier de commandes contenant essentiellement des appels de fonctions.

Certaines erreurs difficiles à trouver proviennent de confusions entre noms de variables ou de fonctions. Scilab garde en mémoire tous les noms introduits tant qu'ils n'ont pas été libérés par `clear`. Il est donc prudent de donner des noms assez explicites aux variables. Les variables introduites dans la session ou dans les fichiers de commandes sont globales. Par défaut, toutes les variables introduites à l'intérieur d'une fonction sont locales. C'est une des raisons pour lesquelles on a intérêt à définir de nouvelles fonctions plutôt que d'utiliser des fichiers de commande exécutables.

Pour comparer l'efficacité des algorithmes, on dispose de la fonction `timer` qui permet de compter le temps CPU écoulé entre deux appels consécutifs. Les fonctions `tic` et `toc` fonctionnent comme `timer` mais affichent le temps en secondes.

```
A=rand(200,200);
b=rand(200,1);
timer(); x=A\b; timer()      // resout le systeme lineaire
timer(); x=inv(A)*b; timer() // inverse la matrice : plus lent
```

```
A=rand(1000,100);
B=rand(1000,100);
format("v",25)
tic(); C=A.*B; toc()
clear C
tic(); for i=1:100,for j=1:100, ..
 C(i,j)=A(i,j)*B(i,j); end, end; toc()
format("v",10)
```

Scilab propose les commandes des langages de programmation classiques.

Commandes principales				
Pour	`for`	`x=vecteur,`	`instruction;`	`end`
Tant que	`while`	`booleen,`	`instruction;`	`end`
Si	`if`	`booleen then,`	`instruction;`	`end`
Sinon	`else`		`instruction;`	`end`
Sinon si	`elseif`	`booleen then,`	`instruction;`	`end`
Selon	`select x`	`case 1 ...`	`then ...`	`end`

La boucle `for` peut aussi s'appliquer à une matrice. Dans :

```
for x=A, instruction, end
```

l'instruction sera exécutée pour `x` prenant successivement comme valeurs les colonnes de la matrice `A`.

Scilab est un outil de calcul. Pour un problème compliqué, on aura tendance à utiliser Scilab pour réaliser des maquettes de logiciels et tester des algorithmes, quitte à lancer ensuite les gros calculs dans un langage compilé comme C. Cela ne dispense pas de chercher à optimiser la programmation en Scilab, en utilisant au mieux la logique du calcul matriciel. Voici un exemple illustrant cette logique. Si $v = (v_i)_{i=1\ldots n}$ et $w = (w_j)_{j=1\ldots m}$ sont deux vecteurs, on souhaite définir la matrice $A = (a_{i,j})$, où $a_{i,j} = v(i)^{w(j)}$. Il y a plusieurs solutions. Dans les commandes qui suivent, `v` est un vecteur colonne et `w` est un vecteur ligne.

```
for i=1:n, for j=1:m, A(i,j)=v(i)^w(j); end; end // a eviter
A=v^w(1); for j=2:m, A=[A,v^w(j)]; end           // mieux
A=v(1)^w; for i=2:n, A=[A;v(i)^w]; end           // equivalent
A=(v*ones(w)).^(ones(v)*w)                       // preferable
```

Si on doit appeler plusieurs fois ce calcul, on aura intérêt à en faire une fonction.

Scilab offre toutes les facilités pour programmer correctement : protection des saisies, utilitaires de débuggage...

Protection et débuggage	
`disp`	affichage de variables
`type`	type des variables
`typeof`	idem
`argn`	nombre de variables d'entrée
`break`	sortie de boucle
`pause`	attente clavier
`return`	sortie de fonction
`resume`	idem
`error`	message d'erreur
`warning`	message d'avertissement

1.4 Calcul numérique

Algèbre linéaire

Tout traitement mathématique est vu par Scilab comme une manipulation de matrices. Pour l'algèbre linéaire, les fonctionnalités sont donc particulièrement développées. Dans une utilisation un peu avancée, on aura intérêt à tirer parti de la structure de matrice creuse (`sparse`) que nous n'aborderons pas.

Il est important de garder à l'esprit que tous les calculs numériques dépendent de la représentation des nombres réels en machine, et que donc les résultats ´ exacts ˇ n'existent pas. La détection de singularités dépend d'une précision prédéfinie et n'est donc pas infaillible.

```
rank([1,1;1,1])
rank([1.00000000000001,1;1,1])
rank([1.000000000000001,1;1,1])
inv([1,1;1,1])
A=[1,2,3;4,5,6;7,8,9]
det(A)
det(A')
rank(A)
inv(A)     // le calcul est effectue
inv(A')    // message d'erreur
A*inv(A)   // ce n'est pas l'identite
```

Si `A` et `B` sont deux matrices, alors `A\B` retourne une matrice `X` telle que `A*X=B`, et `B/A` une matrice `X` telle que `X*A=B` pourvu que les dimensions de `A` et `B` soient compatibles. Si `A` n'est pas une matrice carrée inversible, ces commandes retourneront un résultat, qu'il y ait une infinité de solutions ou qu'il y en ait aucune.

```
A=[1,2,3;4,5,6;7,8,9]
b=[1;1;1]
x=A\b       // le systeme a une infinite de solutions
```

```
A*x
b=[1,1,1]
x=b/A       // le systeme a une infinite de solutions
x*A
b=[1;1;2]
x=A\b       // le systeme n'a pas de solution
A*x
b=[1,1,2]
x=b/A       // le systeme n'a pas de solution
x*A
```

Si `A` est carrée et régulière `x=A\b` est unique et équivalent mathématiquement à `x=inv(A)*b` mais plus rapide.

```
b=[1:500]';
A=diag(b);
timer(); A\b ; timer()
timer(); inv(A)*b ; timer()
```

Pour résoudre un système linéaire quand on craint une singularité, il vaut mieux utiliser `linsolve`, qui détecte les impossibilités et peut retourner une base du noyau de la matrice en cas de solutions multiples. Attention, `linsolve(A,b)` résout `A*x=-b`.

```
A=[1,2,3;4,5,6;7,8,9]
b=[1;1;1]
[x,k]=linsolve(A,b)
A*x
A*k
b=[1;1;2]
[x,k]=linsolve(A,b)  // erreur
```

La commande `[D,U]=bdiag(A)` retourne (en théorie) une matrice diagonale `D` et une matrice inversible `U` telles que `U*D*inv(U)=A`.

```
A=diag([1,2,3]); P=rand(3,3); A=P*A*inv(P)
spec(A)
[D,U]=bdiag(A)       // matrice diagonalisable
inv(U)*A*U
U*D*inv(U)-A
A=[1,0,0;0,1,1;0,0,1]; P=rand(3,3); A=P*A*inv(P)
spec(A)
[D,U]=bdiag(A)       // matrice non diagonalisable
inv(U)*A*U
U*D*inv(U)-A
[X,diagevals]=spec(A)
```

La décomposition LU d'une matrice, c'est-à-dire la décomposition en produit d'une matrice triangulaire inférieure L et d'une matrice triangulaire supérieure U, à des permutations près, s'obtient grâce à la commande `lu`.

```
A= [ 1, 2, 3; 4, 5, 6 ; 7, 8, 9  ]
[L,U,E]=lu(A)
E*A-L*U
det(U)
det(A)
```

Opérations matricielles	
`A'`	transposée (conjuguée) de `A`
`rank`	rang
`lu`	décomposition LU
`inv`	inverse
`expm`	exponentielle matricielle
`det`	déterminant
`trace`	trace
`poly(A,"x")`	polynôme caractéristique de `A`
`spec`	valeurs propres de `A`
`bdiag`	diagonalisation
`svd`	décomposition en valeurs singulières
`A\b`	solution de `A*x = b`
`b/A`	solution de `x*A = b`
`linsolve(A,b)`	solution de `A*x = -b`

Intégration

Des fonctions d'intégration numérique sont disponibles pour les fonctions réelles et complexes, à une, deux et trois variables. Les fonctions `integrate`, `intg`, `int2d`, `int3d`, `intc` et `intl` prennent en entrée une fonction externe, ou définie par une chaîne de caractères. Les fonctions `inttrap` et `intsplin` prennent en entrée des vecteurs d'abscisses et d'ordonnées.

Calculs d'intégrales	
`integrate`	fonction définie par une chaîne de caractères
`intg`	fonction externe
`inttrap`	méthode des trapèzes
`intsplin`	approximation par splines
`int2d`	fonction de deux variables
`int3d`	fonction de trois variables
`intc`	fonction complexe le long d'un segment
`intl`	fonction complexe le long d'un arc de cercle

Dans l'exemple ci-dessous, on calcule avec les différentes fonctions d'intégration disponibles, la valeur de :

$$\int_{-10}^{0} \mathrm{e}^{x}\,\mathrm{d}x \;=\; 1-\mathrm{e}^{-10} \;\simeq\; 0.9999546\,.$$

```
x = [-10:0.1:0];
y=exp(x);
1-1/%e^10
inttrap(x,y)
intsplin(x,y)
integrate("exp(x)","x",-10,0)
deff("y=f(x)","y=exp(x)")
intg(-10,0,f)
```

Les algorithmes de calcul numérique des transformées classiques sont disponibles. Le tableau ci-dessous en donne quelques-unes, voir `apropos transform` pour les autres.

Transformées	
`dft`	transformée de Fourier discrète
`fft`	transformée de Fourier rapide
`convol`	produit de convolution

Résolutions et optimisation

La fonction `fsolve` résout numériquement un système d'équations, mis sous la forme $f(x) = 0$. Comme toutes les résolutions numériques, celle-ci part d'une valeur initiale x_0, et itère une suite censée converger vers une solution. Le résultat dépend évidemment de la valeur initiale.

```
deff("y=f(x)","y=sin(%pi*x)")
fsolve(0.2,f)
fsolve(0.4,f)
fsolve(0.45,f)
fsolve(0.5,f)
fsolve([0.45:0.01:0.5],f)
help fsolve
[x,v,info]=fsolve(0.5,f)
[x,v,info]=fsolve([0.45:0.01:0.5],f)
```

Résolutions	
`fsolve`	systèmes d'équations
`roots`	racines d'un polynôme
`factors`	facteurs irréductibles réels d'un polynôme
`linsolve`	systèmes linéaires

La fonction `optim` recherche un minimum (local) d'une fonction dont on connaît le gradient. La définition de la fonction en entrée est un peu particulière. Le choix de trois algorithmes différents est offert en option.

```
help optim
deff("[y,yprim,ind]=f(x,ind)","y=sin(%pi*x),yprim=%pi*cos(%pi*x)")
[x,v]=optim(f,0.2)
[v,xopt]=optim(f,0.50000000000000001)
[v,xopt]=optim(f,0.5000000000000001)
```

Optimisation	
`optim`	optimisation
`linpro`	programmation linéaire
`quapro`	programmation quadratique

Equations différentielles

La fonction `ode` est en fait un environnement qui donne accès à la plupart des méthodes numériques classiques pour la résolution des équations différentielles ordinaires (voir `help ode`).

À titre d'exemple nous commençons par le problème de Cauchy en dimension 1 suivant :

$$\begin{cases} y'(t) &= y(t)\cos(t)\ , \\ y(0) &= 1\ . \end{cases}$$

La solution explicite est $y(t) = \exp(\sin(t))$. La résolution numérique par `ode` est très stable.

```
deff("yprim=f(t,y)","yprim=y*cos(t)")
t0=0; y0=1; t=[0:0.01:10];
sol=ode(y0,t0,t,f);
max(abs(sol-exp(sin(t))))     // l'erreur est tres faible
t0=0; y0=1; t=[0:0.1:100];
sol=ode(y0,t0,t,f);
max(abs(sol-exp(sin(t))))     // l'erreur reste tres faible
```

La situation n'est pas toujours aussi favorable. Considérons par exemple $y(t) = \exp(t)$, solution du problème de Cauchy suivant.

$$\begin{cases} y'(t) &= y(t)\ , \\ y(0) &= 1\ . \end{cases}$$

```
deff("yprim=f(t,y)","yprim=y")
t0=0; y0=1; t=[0:0.01:10];
sol=ode(y0,t0,t,f);
max(abs(sol-exp(t)))          // l'erreur est raisonnable
```

```
t0=0; y0=1; t=[0:0.1:100];
sol=ode(y0,t0,t,f);
max(abs(sol-exp(t)))          // l'erreur est enorme
```

Voici en dimension 2 la résolution d'un système différentiel de Lotka-Volterra, avec superposition de la trajectoire calculée et du champ de vecteurs défini par le système (figure 8).

```
//
// definition du systeme
//
deff("yprim=f(t,y)",..
    ["yprim1=y(1)-y(1)*y(2)";..
     "yprim2=-2*y(2)+2*y(1)*y(2)";..
     "yprim=[yprim1;yprim2]"])
//
// champ de vecteurs
//
xmin=0; xmax=3; ymin=0; ymax=4;
fx=xmin:0.3:xmax; fy=ymin:0.3:ymax;
clf()
fchamp(f,1,fx,fy)
//
//resolution du systeme
//
t0=0; tmax=8; pas=0.1
t=t0:pas:tmax;
y0=[2; 2];
sol=ode(y0,t0,t,f);
//
// trace de la trajectoire et point initial
//
plot2d(y0(1),y0(2),style=-3)
plot2d(sol(1,:),sol(2,:),style=5)
```

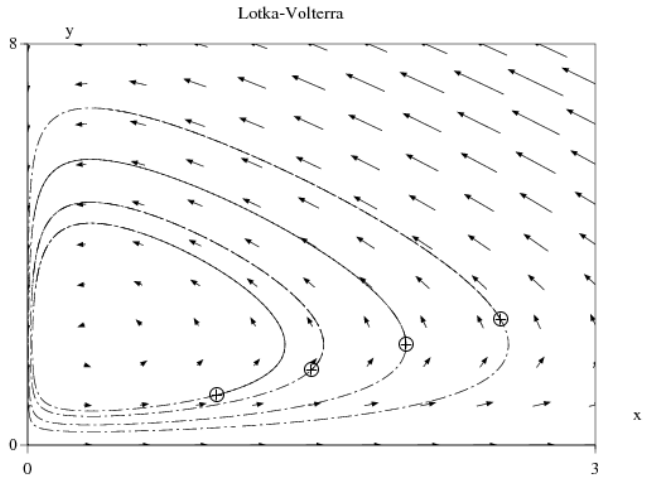


FIG. 8 – Résolution d'un système de Lotka-Volterra par `ode`, représentation de plusieurs solutions et du champ de vecteurs.

Index

`abs`, 11
`acos`, 14
addition, 7, 8
affichage
    suppression de l', 1
aide en ligne, 2
`and`, 10
`ans`, 1
`apropos`, 2, 13
arcosinus, 14
arcsinus, 14
arctangente, 14
`argn`, 31
argument, 11
arrondi, 14
`asin`, 14
`atan`, 14

`bdiag`, 33
`bool2s`, 10
booléens, 10
`break`, 31

`case`, 30
`ceil`, 14
chaînes de caractères, 13
`champ`, 24
champ de vecteurs, 23, 25, 36
`chdir`, 27
`clear`, 1, 29
`clf`, 16
`coeff`, 12
commandes d'itération, 3
commentaire, 1
complexes, 11
concaténer, 5, 13
`conj`, 11
conjuguée, 11
constantes prédéfinies, 9
`contour2d`, 24
`convol`, 34
`cos`, 14
cosinus, 14
courbe
    en dimension 2, 17
    en dimension 3, 25
courbes de niveau, 23, 24
`cumprod`, 14
`cumsum`, 14

`deff`, 13
définition d'une fonction, 13, 15, 28
`det`, 33
`dft`, 34
`diag`, 5
diagonalisation, 33
`disp`, 31
division
    terme à terme, 8

`%e`, 9
écart-type, 14
`else`, 30
`elseif`, 30
empiler, 5
`end`, 30
`%eps`, 9
équations
    différentielles, 35
    non linéaires, 34
`error`, 31
`eval3dp`, 26
évaluation
    d'une expression, 13
`evstr`, 13
`exec`, 27
`execstr`, 13
`exp`, 14
`expm`, 33
exponentielle, 14
    matricielle, 33
extraction, 10
extraire, 13
`eye`, 5

`%f`, 9
`factors`, 12, 35
false, 10
`fchamp`, 24
`fcontour2d`, 24
`fft`, 34
`fgrayplot`, 24
fichiers
    de commandes, 27
    de fonctions, 28
    de sauvegarde, 27
`find`, 10
`floor`, 14
fonctions, 13
    mathématiques, 14
    vectorielles, 14
`for`, 30
`format`, 9
`fplot2d`, 17, 20
`fplot3d`, 26
`fplot3d1`, 26
fractions rationnelles, 12
`fsolve`, 34, 35

`getcolor`, 18
graduations, 20
`grand`, 7
graphiques, 16
`grayplot`, 24
`gsort`, 14, 16

`help`, 2, 13
`hist3d`, 26
histogramme, 24
    en dimension trois, 26
`histplot`, 24
`horner`, 12

`%i`, 9
`if`, 30
`imag`, 11
`%inf`, 9
infini, 9
`int2d`, 34
`int3d`, 34
`intc`, 34
`integrate`, 34
`intg`, 34
`intl`, 34
`inttrap`, 34
`intsplin`, 34
`inv`, 33
`inv_coeff`, 12

`kron`, 6

légendes, 19
`length`, 13
`linpro`, 35
`linsolve`, 33, 35
`linspace`, 3
`log`, 14
logarithme, 11, 14
`logspace`, 3
`lu`, 33

matrice, 2
    aléatoire, 5
    coefficients d'une, 3
    coefficients diagonaux, 5
    constante, 5
    décomposition LU d'une, 33
    définition d'une, 2
    déterminant d'une, 33
    de zéros, 5
    diagonale, 5
    diagonales constantes, 5
    diagonalisation d'une, 33
    dimensions d'une, 2
    exponentielle d'une, 33
    extraire des coefficients, 10
    identité, 5
    inverse d'une, 33
    par blocs, 5
    polynôme caractéristique d'une, 33
    rang d'une, 33
    redimensionner une, 6
    trace d'une, 33
    triangulaire, 5
    valeurs propres d'une, 33

vide, 4
matrix, 6
max, 14, 16
maximum, 14
mean, 14
median, 14
médiane, 14
min, 14, 16
minimum, 14
module, 11
moyenne, 14

niveaux de gris, 23, 24

ode, 35
ones, 1, 5
opérations
matricielles, 7, 8
numériques, 7
ordre des, 7
optim, 35
or, 10

param3d, 26
param3d1, 26
part, 13
partie
entière, 14
imaginaire, 11
réelle, 11
pause, 31
phasemag, 11
%pi, 9
plot, 17
plot2d, 17, 20
plot2d1, 20
plot2d2, 20
plot2d3, 20
plot2d4, 20
plot3d, 26
plot3d1, 26
poly, 12, 33
polynôme, 12
caractéristique, 33
coefficients d'un, 12
racines d'un, 12
précision machine, 9
prod, 14
produit, 14
cumulé, 14
de Kronecker, 6
matriciel, 6–8
terme à terme, 7, 8
programmation, 29
puissance, 11
matricielle, 7, 8
terme à terme, 7, 8
pwd, 27

quapro, 35

racines
carrées, 11, 14
d'un polynôme, 12
rand, 5
rank, 33
real, 11
redimensionner, 6
répertoire, 27
resume, 31
return, 31
roots, 12, 35
round, 14

%s, 9
scalaires, 2
scf, 16
select, 30
set, 16
sin, 14
sinus, 14
size, 2
scf, 16
somme, 14
cumulée, 14
sort, 14, 16
spec, 33
sqrt, 14
stdev, 14
string, 13

style, 18
`sum`, 14
surface
    équation d'une, 26
    paramétrée, 26
`svd`, 33
système
    linéaire, 32, 33
    non linéaire, 35

`%t`, 9
`tan`, 14
tangente, 14
`then`, 30
`tic`, 29
`timer`, 29
`toc`, 29
`toeplitz`, 5
`trace`, 33
transformées, 34
transposée, 2, 6, 33
tri, 14
`tril`, 5
`triu`, 5
true, 10
`type`, 31
`typeof`, 31
types de données, 9

valeurs
    propres, 33
    singulières, 33
variables
    globales, 29
    locales, 29
`varn`, 12
vecteur, 2
    coefficients d'un, 4

`warning`, 31
`while`, 30
`who_user`, 1
`winsid`, 17

`xarc`, 23
`xarrows`, 23
`xdel`, 16
`xfarc`, 23
`xfpoly`, 23
`xfpolys`, 23
`xfrect`, 23
`xnumb`, 23
`xpoly`, 23
`xrect`, 23
`xrpoly`, 23
`xstring`, 23
`xstringb`, 23
`xtitle`, 23

`zeros`, 5

2 Entraînement

2.1 Vrai ou faux

Vrai-Faux 1. On pose `v=[1:5]'`. Quelles affirmations sont correctes, lesquelles ne le sont pas et pourquoi ?

1. ☐ `v*v'==v'*v` donne 1
2. ☐ `v*v'==v'*v` donne `T`
3. ☒ `v*v'` est une matrice
4. ☐ `v*v` est une matrice
5. ☒ `v'*v-norm(v).^2` donne 0

Vrai-Faux 2. On pose `A=[ 1:3 ; 2,2,2 ; 3:-1:1]` et `v=[3:-1:1]'`, quelles affirmations sont correctes, lesquelles ne le sont pas et pourquoi ?

1. ☒ `v'*v` vaut 14
2. ☐ `v.*v` vaut 14
3. ☒ `v'*A*v` est un réel
4. ☐ `v==v(3:-1:1)` donne T
5. ☒ `v==v(3:-1:1)` donne `[ F T F ]'`
6. ☒ `v(3:-1:1)==A(:,1)` donne `[T T T]'`;
7. ☐ `v(3:-1:1)==A(1,:)` donne `[T T T]`;

Vrai-Faux 3. Parmi les lignes de commande suivantes, lesquelles affichent le vecteur ligne `v=[1,2,3,4,5]`, lesquelles ne l'affichent pas et pourquoi ?

1. ☐ `v=[1;2;3;4;5]`
2. ☐ `v=[1,2,3,4,5];`
3. ☐ `[1:5]; v`
4. ☒ `v=[1:5]`
5. ☐ `v=[1:5];`
6. ☒ `[1:5]; v=ans`
7. ☐ `v=[1;5]`
8. ☐ `v=[1:10]; v(1,5)`
9. ☒ `v=[1:10]; v(1,[1:5])`
10. ☐ `v=linspace(1,5)`
11. ☒ `v=linspace(1,5,5)`
12. ☒ `v=cumsum(ones(1,5))`
13. ☐ `v=cumsum(ones(5,1))`
14. ☒ `v=find(ones(5,1)==1)`

15. ☐ `v=find(rand(1,5)<0)`
16. ☒ `v=find(rand(1,5)<1)`
17. ☒ `v=find(zeros(1,5)<=0)`
18. ☐ `v=sort([1,3,2,5,4])`
19. ☐ `v=sort([1;3;2;5;4]); v([5:-1:1])`
20. ☒ `v=sort([1,3,2,5,4]); v([5:-1:1])`
21. ☐ `v=gsort([1,3,2,5,4],"r","i")`
22. ☒ `v=gsort([1,3,2,5,4],"c","i")`
23. ☒ `v=gsort([1;3;2;5;4],"r","i"); v=v'`
24. ☐ `for i=1:5, v=i; end`
25. ☐ `for i=1:5, v=[v,i]; end; v`
26. ☒ `v=[]; for i=1:5, v=[v,i]; end; v`
27. ☐ `v=1; for i=1:5, v=[v,v($)+1]; end; v`
28. ☒ `v=1; for i=1:4, v=[v,v($)+1]; end; v`
29. ☐ `i=0; v=i; while i<=5, i=i+1; v=[v,i]; end; v`
30. ☐ `i=1; v=i; while i<=5, i=i+1; v=[v,i]; end; v`
31. ☒ `i=1; v=i; while i<5, i=i+1; v=[v,i]; end; v`

Vrai-Faux 4. Parmi les lignes de commande suivantes, lesquelles affichent le vecteur ligne `v=[1,0.5,0.25,0.125,0.0625]`, lesquelles ne l'affichent pas et pourquoi ?

1. ☒ `2^(-[0:4])`
2. ☐ `1/2^[0:4]`
3. ☐ `(1)/2^[0:4]`
4. ☒ `(1)./2^[0:4]`
5. ☐ `v=cumprod(ones(1,5)/2)`
6. ☒ `v=[1,cumprod(ones(1,4)/2)]`
7. ☒ `v=cumprod([1,0.5*ones(1,4)])`
8. ☐ `for i=0:4, v=2^(-i); end;`
9. ☒ `v=[]; for i=0:4, v=[v,2^(-i)]; end; v`
10. ☐ `v=[1]; for i=1:4, v=[v,v/2]; end; v`
11. ☒ `x=1; v=[x]; for i=1:4, x=x/2; v=[v,x]; end; v`
12. ☐ `x=1; v=[x]; for i=1:4, v=[v,x/2]; end; v`
13. ☐ `x=1; v=[x]; while x>0.06, x=x/2; v=[v,x]; end; v`
14. ☒ `x=1; v=[x]; while x>0.1, x=x/2; v=[v,x]; end; v`

Vrai-Faux 5. Parmi les lignes de commande suivantes, lesquelles affichent la matrice carrée à deux lignes et deux colonnes `A`, dont la première ligne est le vecteur `[0,1]` et la deuxième ligne est le vecteur `[1,0]`, lesquelles ne l'affichent pas et pourquoi ?

1. ☒ `A=[0,1;1,0]`
2. ☐ `A=[0,1]; A=[A,[1,0]]`
3. ☒ `A=[0,1]; A=[A;[1,0]]`
4. ☐ `A=[0;1]; A=[A;[1;0]]`
5. ☒ `A=[0;1]; A=[A,[1;0]]`
6. ☐ `A=[0,1]; A=A+A'`
7. ☐ `A=[0,1]; A=[A;A]`
8. ☒ `A=[0,1]; A=[A;A([2,1])]`
9. ☐ `A=[0,1]; A=[A;A(2,1)]`
10. ☒ `A(1,2)=1; A(2,1)=1`
11. ☒ `A=ones(2,2); A(1,1)=0; A(2,2)=0`
12. ☐ `A=eye(2,2)`
13. ☒ `A=ones(2,2)-eye(2,2)`
14. ☒ `A=ones(2,2)-diag(ones(1,2))`
15. ☐ `A=matrix([0,1,0,1],2,2)`
16. ☒ `A=matrix([0,1,1,0],2,2)`
17. ☒ `A=matrix([0;1;1;0],2,2)`
18. ☐ `A=[1,0]*[0;1]+[0,1]*[1;0]`
19. ☒ `A=[1;0]*[0,1]+[0;1]*[1,0]`
20. ☐ `U=ones(2,2); U(1,2)=-1; A=U*diag([1,-1])*U'`
21. ☒ `U=ones(2,2); U(1,2)=-1; A=U*diag([1,-1])*U'/2`
22. ☐ `A=toeplitz(0,1)`
23. ☒ `A=toeplitz([0,1])`
24. ☐ `A=bool2s(eye(2,2)==2);`
25. ☒ `A=bool2s([1,2;2,3]==2)`

Vrai-Faux 6. Parmi les lignes de commande suivantes, lesquelles affichent la matrice carrée à dix lignes et dix colonnes `A`, dont les coefficients d'ordre $(i, i+1)$ valent 1 pour $i = 1, \ldots, 9$, tous les autres coefficients étant nuls, lesquelles ne l'affichent pas et pourquoi ?

1. ☐ `A=eye(10,10)`
2. ☒ `A=[zeros(9,1),eye(9,9);zeros(1,10)]`
3. ☐ `for i=1:9, A(i,i+1)=1; end; A`
4. ☒ `A=zeros(10,10); for i=1:9, A(i,i+1)=1; end; A`
5. ☐
```
for i=1:10, for j=1:10, if j==i+1 then A(i,j)=1;..
                                  else A(i,j)=0; end; end; A
```

6. ☒ `for i=1:10, for j=1:10, if j==i+1 then A(i,j)=1;.. else A(i,j)=0; end; end; end; A`
7. ☐ `A=toeplitz([0,1,zeros(1,8)])`
8. ☒ `A=toeplitz(zeros(10,1),[0,1,zeros(1,8)])`

Vrai-Faux 7. Parmi les lignes de commande suivantes, lesquelles affichent la matrice carrée à dix lignes et dix colonnes `A`, dont les coefficients d'ordre $(i, i+1)$ et $(i+1, i)$ valent 1 pour $i = 1, \ldots, 9$, tous les autres coefficients étant nuls, lesquelles ne l'affichent pas et pourquoi ?

1. ☐ `A=eye(10,10)`
2. ☒ `A=[zeros(9,1),eye(9,9);zeros(1,10)]; A=A+A'`
3. ☒ `A=zeros(10,10); for i=1:9, A(i,i+1)=1; A(i+1,i)=1; end; A`
4. ☐ `for i=1:10, for j=1:10, if or(j==i+1,j==i-1) then A(i,j)=1;.. else A(i,j)=0; end; end; A`
5. ☒ `for i=1:10, for j=1:10, if or([j==i+1,j==i-1]) then A(i,j)=1;.. else A(i,j)=0; end; end; end; A`
6. ☒ `for i=1:10, for j=1:10, if j==i+1|j==i-1 then A(i,j)=1;.. else A(i,j)=0; end; end; end; A`
7. ☐ `A=toeplitz([1,zeros(1,9)])`
8. ☒ `A=toeplitz([0,1,zeros(1,8)])`

Vrai-Faux 8. Soit n un entier. Parmi les lignes de commande suivantes, lesquelles affichent le vecteur de taille $2n$ dont la coordonnée d'ordre $2i$ vaut i pour $i = 1, \ldots, n$, les autres coordonnées étant nulles, lesquelles ne l'affichent pas et pourquoi ?

1. ☒ `v=zeros(1,2*n); v(2*[1:n])=[1:n]`
2. ☐ `v=zeros(1,2*n); v([1:n])=2*[1:n]`
3. ☒ `v=[zeros(1,n);[1:n]]; v=matrix(v,1,2*n)`
4. ☐ `v=[zeros(n,1);[1:n]']; v=matrix(v,1,2*n)`
5. ☐ `v=[zeros(1,n);[1:n]]; v=v([1:2*n])`
6. ☒ `v=[zeros(1,n);[1:n]]'; v=v([1:2*n])`
7. ☒ `v=[]; for i=1:n, v=[v,0,i]; end; v`
8. ☐ `v=[]; for i=1:n, v=[v,[0;i]]; end; v`
9. ☐ `v=kron([1:n],[0;1])`
10. ☒ `v=kron([1:n],[0,1])`

Vrai-Faux 9. Soit n un entier. Parmi les lignes de commande suivantes, lesquelles affichent le vecteur de taille n dont la coordonnée d'ordre i vaut $\frac{i(i-1)}{2}$ pour $i = 1, \ldots, n$, lesquelles ne l'affichent pas et pourquoi ?

1. ☐ `v=[1:n]*[0:n-1]/2`

2. ☒ `v=[1:n].*[0:n-1]/2`
3. ☒ `v=[1:n]; v=v.*(v-1)/2`
4. ☐ `v=[1:n]; v=v.^2-v/2`
5. ☒ `v=[1:n]; v=(v.^2-v)/2`
6. ☐ `v=cumsum([1:n])`
7. ☒ `v=cumsum([0:n-1])`
8. ☐ `for i=1:n v(i)=i*(i-1)/2; end;`
9. ☒ `for i=1:n, v(i)=(i^2-i)/2; end; v`
10. ☒ `v=[]; for i=1:n, v=[v,i*(i-1)/2]; end;`

Vrai-Faux 10. Soit n un entier et $v = (a_1, \ldots, a_n)$ un vecteur ligne d'entiers tous compris entre 0 et 9. Parmi les lignes de commande suivantes, lesquelles affichent le réel x, compris entre 0 et 1 dont les n décimales sont $(a_1, \ldots, a_n)$, lesquelles ne l'affichent pas et pourquoi ?

1. ☐ `x=sum(v*10^(-[1:length(v)]))`
2. ☒ `x=sum(v.*10^(-[1:length(v)]))`
3. ☐ `x=sum(v*10^(-[1:size(v)]))`
4. ☒ `x=sum(v.*10^(-[1:size(v,2)]))`
5. ☒ `x=sum(v.*10^(-[1:size(v,"*")]))`
6. ☐ `s="0"+"."+sum(string(v)); x=evstr(s)`
7. ☒ `s="0"+"."+strcat(string(v)); x=evstr(s)`
8. ☒ `x=0; for i=1:length(v), x=x+v(i)*10^(-i); end; x`
9. ☐ `x=0; for i=v, x=x+i*10^(-i); end; x`
10. ☐ `x=0; d=0.1; for i=v, d=d/10; x=x+i*d; end; x`
11. ☒ `x=0; d=0.1; for i=v, x=x+i*d; d=d/10; end; x`

Vrai-Faux 11. Soit x un réel compris entre 0 et 1 et n un entier. Parmi les lignes de commande suivantes, lesquelles affichent le vecteur ligne formé des n premières décimales de x, lesquelles ne l'affichent pas et pourquoi ?

1. ☐ `v=floor(x*10^[1:n])`
2. ☒ `v=floor(x*10^[1:n]); v=v-[0,v([1:n-1])]*10`
3. ☐ `v=[]; for i=1:n, a=int(x*10^i); v=[v,a]; end;`
4. ☒ `v=[]; for i=1:n, a=int(x*10); v=[v,a]; x=x*10-a; end; v`
5. ☐ `s=string(x); v=[]; for i=1:n, v=[v,part(s,i+2)]; end; v`
6. ☒ `s=string(x); v=[]; for i=1:n, v=[v,part(s,i+2)]; end; v=evstr(v)`

Vrai-Faux 12. Soit $x = (x_i)_{i=1,\dots,n}$ et $y = (y_i)_{i=1,\dots,n}$ deux vecteurs de réels de même taille. Parmi les lignes de commande suivantes lesquelles affichent le calcul de l'intégrale par la méthode des rectangles à droite :

$$I = \sum_{i=1}^{n-1} f(x_{i+1})(x_{i+1} - x_i) \ ,$$

lesquelles ne l'affichent pas et pourquoi

1. ☐ `n=size(x,"*"); I = sum((x([2:n])-x([1:n-1]))*y([2:n]))`
2. ☒ `n=size(x,"*"); I = sum((x([2:n])-x([1:n-1])).*y([2:n]))`
3. ☐ `n=size(x,"*"); I = sum((x([2:n])-x([1:n-1]))*y([1:n-1]))`
4. ☐ `I=0; for i=1:length(x), I = I+(x(i+1)-x(i))*y(i+1); end; I`
5. ☒ `I=0; for i=2:length(x), I = I+(x(i)-x(i-1))*y(i); end; I`
6. ☒ `I=0; for i=1:length(x)-1, I = I+(x(i+1)-x(i))*y(i+1); end; I`

Vrai-Faux 13. Soit z un nombre complexe non nul. Parmi les lignes de commande suivantes, lesquelles affichent l'argument de z (réel t dans $[0, 2\pi[$ tel que $z = |z|e^{it}$), lesquelles ne l'affichent pas et pourquoi ?

1. ☐ `t=argn(z)`
2. ☐ `t=phasemag(z)`
3. ☒ `t=%phasemag(z)/180*%pi`
4. ☐ `t=log(z/abs(z))/%i`
5. ☐ `t=imag(log(z/abs(z)))`
6. ☒ `t=imag(log(z/abs(z))); if imag(z)<0 then t=t+2*%pi; end; t`
7. ☐ `t=acos(real(z)/abs(z))`
8. ☒ `t=acos(real(z)/abs(z)); if imag(z)<0 then t=-t+2*%pi; end; t`
9. ☐ `t=asin(imag(z)/abs(z))`
10. ☐ `t=asin(imag(z)/abs(z)); if imag(z)<0 then t=-t+2%pi; end; t`
11. ☐ `t=atan(imag(z)/real(z))`

Vrai-Faux 14. Soit $P(X) = a_0 + a_1X + \cdots + a_nX^n$ un polynôme. Parmi les lignes de commande suivantes, lesquelles affichent le polynôme réciproque $Q(X) = a_n + a_{n-1}X + \cdots + a_0X^n$, lesquelles ne l'affichent pas et pourquoi ?

1. ☐ `Q=X^degree(P)*horner(P,1/X)`
2. ☒ `X=poly([0,1],"X","coeff"); Q=X^degree(P)*horner(P,1/X)`
3. ☐ `c=coeff(P); c=c([length(c):-1:1]); Q=poly(c,"X");`
4. ☒ `c=coeff(P); c=c([length(c):-1:1]); Q=poly(c,"X","coeff")`
5. ☐ `r=roots(P); Q=poly(1/r,"X")`
6. ☒

```
r=roots(P); c=coeff(P);
Q=c($)*poly((1)./r,"X")*prod(r)*(-1)^degree(P)
```

Vrai-Faux 15. Parmi les lignes de commande suivantes, lesquelles affichent un segment de droite d'une seule couleur, lesquelles ne l'affichent pas et pourquoi ?

1. ☐ `plot2d([0,1],[0,1],style=-5)`
2. ☒ `plot2d([0,1],[0,1],style=5)`
3. ☐ `x=[0:0.1:1]; plot2d(x,x,style=0)`
4. ☐ `x=[0:0.1:1]; plot2d2(x,x)`
5. ☐ `x=[0:0.1:1]; plot2d3(x,x)`
6. ☐ `x=[0,0.5;0.5,1]; plot2d(x,x)`
7. ☒ `x=[0,0.5;0.5,1]; plot2d(x,x,style=[5,5])`

Vrai-Faux 16. Parmi les lignes de commande suivantes, lesquelles affichent un losange, lesquelles ne l'affichent pas et pourquoi ?

1. ☒ `x=[-1;0;1]; y=[0;1;0]; plot2d([x,x],[y,-y],style=[1,1])`
2. ☐ `x=[-1,0,1]; y=[0,1,0]; plot2d([x;x],[y;-y],style=[1,1])`
3. ☐ `x=[0,1,0,-1]; y=[-1,0,1,0]; xpoly(x,y)`
4. ☒ `plot2d(0,0,rect=[-2,-2,2,2])`
   `x=[0,1,0,-1]; y=[-1,0,1,0]; xpoly(x,y,"lines",1)`

Vrai-Faux 17. On suppose que les échelles en abscisse et ordonnée ont été fixées par la commande
`isoview(-1,1,-1,1)`. Parmi les lignes de commande suivantes, lesquelles affichent un cercle, lesquelles ne l'affichent pas et pourquoi ?

1. ☒ `x=linspace(-1,1,200)'; y=sqrt(1-x.^2); plot2d([x,x],[y,-y],style=[5,5])`
2. ☐ `x=[-1:1]'; y=sqrt(1-x.^2); plot2d([x,x],[y,-y],style=[5,5])`
3. ☐ `x=linspace(-1,1,200)';y=sqrt(1-x.^2);plot2d([x;x],[y;-y],style=[5,5])`
4. ☒ `t=linspace(0,2*%pi); x=cos(t); y=sin(t); plot2d(x,y)`
5. ☐ `xarc(-1,1,2,2,0,2*%pi)`
6. ☒ `plot2d(0,0); xarc(-1,1,2,2,0,360*64)`

Vrai-Faux 18. Parmi les lignes de commande suivantes, lesquelles affichent une représentation graphique satisfaisante de la fonction $f(x) = \frac{1}{\sin(x)}$, pour $x \in [0, 2\pi]$, lesquelles ne l'affichent pas et pourquoi ?

1. ☒ `e=0.1; x=linspace(e,%pi-e,200); y=(1)./sin(x);`
   `plot2d(x,y,style=5,rect=[0,-10,2*%pi,10]);`
   `e=0.1; x=linspace(%pi+e,2*%pi-e,200); y=(1)./sin(x);`
   `plot2d(x,y,style=5,frameflag=0);`
2. ☐ `e=0.001; x=linspace(e,%pi-e,200); y=(1)./sin(x); plot2d(x,y,style=5);`
   `e=0.001; x=linspace(%pi+e,2*%pi-e,200); y=(1)./sin(x); plot2d(x,y,style=5);`

3. ☒ `e=0.1; x=linspace(e,%pi-e,200); x=[x',x'+%pi]; y=(1)./sin(x); plot2d(x,y,style=[5,5]);`
4. ☐ `e=0.1; x=[e:e:%pi-e]; x=[x ,x+%pi]; y=(1)./sin(x); plot2d(x,y,style=[5,5]);`

Vrai-Faux 19. On souhaite définir une fonction f qui prenne en entrée une matrice quelconque `x`, et qui retourne la matrice des images des coefficients de `x` par la fonction $f(x) = \frac{1}{e^x \sin(x)}$. Parmi les lignes de commande suivantes, lesquelles sont correctes, lesquelles ne le sont pas et pourquoi ?

1. ☐ `deff("y=f(x)","y=1/(exp(x)*sin(x))")`
2. ☒ `deff("y=f(x)","y=(1)./(exp(x).*sin(x))")`
3. ☐ `deff("y=f(x)","y=1./(%e^x.*sin(x))")`
4. ☒ `deff("y=f(x)","y=(1)./(%e^x.*sin(x))")`
5. ☒ `deff("y=f(x)","y=exp(-x)./sin(x)")`
6. ☐ `function y=f(x) y=(1)./(%e^x.*sin(x)) endfunction`
7. ☒
```
function y=f(x)
y=(1)./(%e^x.*sin(x))
endfunction
```

2.2 Exercices

Il y a souvent plusieurs manières d'obtenir le même résultat en Scilab. On s'efforcera de choisir les solutions les plus compactes, c'est-à-dire celles qui utilisent au mieux le langage matriciel.

Exercice 1. Écrire (sans utiliser de boucle) les vecteurs suivants :

1. Nombres de 1 à 3 par pas de 0.1.
2. Nombres de 3 à 1 par pas de -0.1.
3. Carrés des 10 premiers entiers.
4. Nombres de la forme $(-1)^n n^2$ pour $n = 1, \ldots, 10$.
5. 10 "0" suivis de 10 "1".
6. 3 "0" suivis de 3 "1", suivis de 3 "2",..., suivis de 3 "9".
7. "1", suivi de 1 "0", suivi de "2", suivi de 2 "0",..., suivi de "8", suivi de 8 zéros, suivi de "9".
8. 1 "1" suivi de 2 "2", suivis de 3 "3",..., suivis de 9 "9".

Exercice 2. Écrire (sans utiliser de boucle) les matrices carrées d'ordre 6 suivantes :

1. Matrice diagonale, dont la diagonale contient les entiers de 1 à 6.
2. Matrice contenant les entiers de 1 à 36, rangés par lignes.
3. Matrice dont toutes les lignes sont égales au vecteur des entiers de 1 à 6.

4. Matrice diagonale par blocs, contenant un bloc d'ordre 2 et un d'ordre 4. Les 4 coefficients du premier bloc sont égaux à 2. Le deuxième bloc contient les entiers de 1 à 16 rangés sur 4 colonnes.
5. Matrice $A = \left((-1)^{i+j}\right)$, $i, j = 1, \ldots, 6$.
6. Matrice contenant des "1" sur la diagonale, des "2" au-dessus et au-dessous, puis des "3", jusqu'aux coefficients d'ordre $(1, 6)$ et $(6, 1)$ qui valent 6.

Exercice 3. Écrire la matrice $A = (a_{i,j})$ d'ordre 12 contenant les entiers de 1 à 144, rangés par lignes. Extraire de cette matrice les matrices B suivantes.

1. Coefficients $a_{i,j}$ pour $i = 1, \ldots, 6$ et $j = 7, \ldots, 12$.
2. Coefficients $a_{i,j}$ pour $i + j$ pair.
3. Coefficients $a_{i,j}$ pour $i, j = 1, 2, 5, 6, 9, 10$.

Exercice 4.

1. Écrire les polynômes de degré 6 suivants.
   (a) polynôme dont les racines sont les entiers de 1 à 6.
   (b) polynôme dont les racines sont 0 (racine triple), 1 (racine double) et 2 (racine simple).
   (c) polynôme $(x^2 - 1)^3$.
   (d) polynôme $x^6 - 1$.

   Pour chacun de ces polynômes :
2. Écrire la matrice compagnon A associée à ce polynôme : la matrice compagnon associée au polynôme :

$$P = x^d - a_{d-1}x^{d-1} - \cdots - a_1 x - a_0 \ ,$$

est :

$$A = \begin{pmatrix} 0 & 1 & 0 & \ldots & & 0 \\ \vdots & \ddots & \ddots & \ddots & & \vdots \\ & & & & & \\ \vdots & & & \ddots & \ddots & 0 \\ 0 & \ldots & & \ldots & 0 & 1 \\ a_0 & a_1 & & \ldots & & a_{d-1} \end{pmatrix} .$$

3. Calculer les valeurs propres de la matrice A.
4. Calculer le polynôme caractéristique de A.

Exercice 5.

1. Écrire la matrice carrée N d'ordre 6, telle que $n_{i,j} = 1$ si $j = i + 1$, 0 sinon.
2. Calculer N^k, pour $k = 1, \ldots, 6$.
3. Écrire la matrice $A = xI + N$, où x est une variable de polynôme.

4. Calculer A^k, pour $k = 1, \ldots, 6$.
5. Pour $x = -2$, et $t = 0, 1, 2$, calculer $\exp(At)$.

Exercice 6. Etant donnée une série numérique convergente $\sum_{n=0}^{+\infty} u_n$, de somme s, le reste à l'ordre n est défini comme la différence entre la somme et la somme partielle à l'ordre n.

$$r_n = s - \sum_{k=0}^{n} u_k = \sum_{k=n+1}^{+\infty} u_k \ .$$

Le but de l'exercice est de déterminer la valeur de n à partir de laquelle le reste r_n est inférieur à 10^{-3}, pour les séries suivantes.

1. $\displaystyle\sum_{n=0}^{+\infty} \frac{1}{(n+1)(n+2)} = 1 \ .$
2. $\displaystyle\sum_{n=0}^{+\infty} 0.9^n = 10 \ , \quad \sum_{n=0}^{+\infty} 0.99^n = 100 \ , \quad \sum_{n=0}^{+\infty} 0.999^n = 1000 \ .$
3. $\displaystyle\sum_{n=0}^{+\infty} \frac{(-1)^n}{n!} = \exp(-1) \ , \quad \sum_{n=0}^{+\infty} \frac{1}{n!} = \exp(1) \ , \quad \sum_{n=0}^{+\infty} \frac{(10)^n}{n!} = \exp(10) \ .$
4. $\displaystyle\sum_{n=0}^{+\infty} \frac{(-1)^n \pi^{2n}}{(2n)!} = -1 \ , \quad \sum_{n=0}^{+\infty} \frac{(-1)^n (\frac{\pi}{2})^{2n}}{(2n)!} = 0 \ , \quad \sum_{n=0}^{+\infty} \frac{(-1)^n (\frac{\pi}{4})^{2n}}{(2n)!} = \frac{\sqrt{2}}{2} \ .$

Exercice 7. Représenter les fonctions f suivantes, en choisissant l'intervalle des abscisses et des ordonnées ainsi que le pas de discrétisation des abscisses, de façon à obtenir la représentation la plus informative possible.

1. $f(x) = 1/x$.
2. $f(x) = \mathrm{e}^x$.
3. $f(x) = 1/\sin(x)$.
4. $f(x) = x/\sin(x)$.
5. $f(x) = \sin(x)/x$.

Exercice 8. On considère la fonction $f(x) = 3x^2 + 1 + \frac{1}{\pi^4}\log((\pi - x)^2)$.

1. Vérifier que cette fonction prend des valeurs négatives sur $\mathbb{R}^+$. Représenter la fonction sur les abscisses `x=[0:0.0001:5]`.
2. Choisir deux réels positifs (petits), `eps1` et `eps2`. Représenter la fonction sur `[%pi-eps1:eps2:%pi+eps1]`. Quelles valeurs de `eps1` et `eps2` donnent une représentation correcte de la fonction f au voisinage de π ?

Exercice 9.

1. Représenter la fonction $\exp(x)$ sur l'intervalle $[-1, 1]$. Sur le même graphique, superposer les représentations des polynômes de Taylor de cette fonction en $x = 0$, aux ordres $1, 2, 3, 4$.

2. Représenter la fonction $\exp(x)$ sur l'intervalle $[1, 2]$. Sur le même graphique, superposer les représentations des polynômes de Taylor de cette fonction en $x = 1$, aux ordres $1, 2, 3, 4$.
3. Représenter la fonction $\sin(x)$ sur l'intervalle $[-\pi, \pi]$. Sur le même graphique, superposer les représentations des polynômes de Taylor de cette fonction en $x = 0$, aux ordres $1, 3, 5$.

Exercice 10. Superposer les représentations suivantes sur le même graphique, allant de 0 à 1 en abscisse et en ordonnée.

1. La première bissectrice ($y = x$).
2. La fonction $y = f(x) = 1/6 + x/3 + x^2/2$.
3. La tangente à la fonction f au point $x = 1$.
4. Un segment vertical allant de l'axe des x au point d'intersection de la fonction f et de la première bissectrice, et un segment horizontal allant de ce point d'intersection à l'axe des y.
5. Les indications ´ point fixe ˇ et ´ tangente ˇ, positionnées sur le graphique comme chaînes de caractères.

Exercice 11. Le but de l'exercice est de représenter sur un même graphique des familles de fonctions. On choisira le nombre de courbes, l'intervalle de représentation, les échelles en x et y ainsi que le pas de discrétisation des abscisses, de façon à obtenir la représentation la plus informative possible.

1. Fonctions $f_a(x) = x^a \mathrm{e}^{-x}$, pour a allant de -1 à 1.
2. Fonctions $f_a(x) = 1/(x - a)^2$, pour a allant de -1 à 1.
3. Fonctions $f_a(x) = \sin(a * x)$, pour a allant de 0 à 2.

Exercice 12. Pour chacune des courbes paramétrées suivantes, on choisira un intervalle de valeurs du paramètre et un pas de discrétisation assurant une représentation complète et suffisamment lisse.

1. $$\begin{cases} x(t) &= \sin(t) \\ y(t) &= \cos^3(t) \end{cases}$$
2. $$\begin{cases} x(t) &= \sin(4\,t) \\ y(t) &= \cos^3(6\,t) \end{cases}$$
3. $$\begin{cases} x(t) &= \sin(132\,t) \\ y(t) &= \cos^3(126\,t) \end{cases}$$

Exercice 13. Le but de l'exercice est de visualiser de différentes manières la surface définie par $z = f(x, y) = x\,y^2$.

1. Choisir un domaine de représentation et les pas de discrétisation, de manière à optimiser la représentation par `fsurf`.

2. Même question en utilisant `plot3d`.
3. Modifier l'échelle des couleurs pour obtenir une représentation en dégradé de gris, pour laquelle l'intensité lumineuse croît avec z.
4. Choisir un vecteur de valeurs de x. Pour chaque valeur de ce vecteur, on considère la courbe définie par $z = f(x, y)$. Représenter ces courbes sur la même fenêtre graphique.
5. Choisir un vecteur de valeurs de y. Pour chaque valeur de ce vecteur, on considère la courbe définie par $z = f(x, y)$. Représenter ces courbes sur la même fenêtre graphique.

Exercice 14. Le but de l'exercice est de visualiser un cône de différentes manières.

1. Représenter la surface d'équation $z = 1 - \sqrt{x^2 + y^2}$.
2. Représenter la surface paramétrée définie par :

$$\begin{cases} x(u,v) &= u\,\cos(v) \\ y(u,v) &= u\,\sin(v) \\ z(u,v) &= 1-u\ . \end{cases}$$

3. Représenter la courbe paramétrée définie par :

$$\begin{cases} x(t) &= t\,\cos(at) \\ y(t) &= t\,\sin(at) \\ z(t) &= 1-t\ . \end{cases}$$

   (On choisira une valeur de a suffisamment grande).
4. Représenter la famille de courbes paramétrées définies par :

$$\begin{cases} x(t) &= a\,\cos(t) \\ y(t) &= a\,\sin(t) \\ z(t) &= 1-a\ . \end{cases}$$

Exercice 15. Écrire les fonctions suivantes, sans utiliser de boucle. Toutes prennent en entrée un vecteur colonne $v = (v_i)$, un vecteur ligne $w = (w_j)$ et retournent en sortie une matrice $A = (a_{i,j})$ qui a autant de lignes que v et autant de colonnes que w. Seules les expressions des coefficients $a_{i,j}$ diffèrent.

1. `produit` : $a_{i,j} = v_i * w_j$.
2. `somme` : $a_{i,j} = v_i + w_j$.
3. `quotient` : $a_{i,j} = v_i / w_j$.
4. `echiquier` : $a_{i,j} = v_i$ si $i + j$ est pair, w_j sinon.

Exercice 16. Écrire les fonctions suivantes, sans utiliser de boucle.

1. `insere_zeros` : Elle prend en entrée une matrice quelconque A. Elle insère une colonne de zéros après chaque colonne de `A`, et retourne en sortie la matrice modifiée (même nombre de lignes, deux fois le nombre de colonnes).

2. `alterne2_colonnes` : Elle prend en entrée deux matrices quelconques A et B, supposées de mêmes dimensions. Elle retourne la matrice formée en alternant les colonnes de A et B.
3. `alterne3_colonnes` : Même chose pour trois matrices A, B et C de mêmes dimensions.

Exercice 17. Écrire les fonctions suivantes, sans utiliser de boucle.

1. La fonction f prend en entrée un entier n et deux réels a, b et retourne la matrice A dont les termes diagonaux valent a, tous les autres termes étant égaux à b.
2. La fonction g prend en entrée un entier n et trois réels a, b, c et retourne la matrice $A = (a_{i,j})_{i,j=1,\ldots,n}$ dont les termes diagonaux sont égaux à a, les termes $a_{i,i+1}$ égaux à b et termes $a_{i+1,i}$ égaux à c, pour $i = 1, \ldots, n-1$.
3. La fonction h prend en entrée un vecteur $x = (x_i)_{i=1,\ldots,n}$ et retourne en sortie la matrice $A = (a_{i,j})_{i,j=1,\ldots,n}$ définie par $a_{i,j} = x_j^{i-1}$ (matrice de Vandermonde).

Exercice 18. Le but de l'exercice est d'étudier la méthode de Horner pour l'évaluation algorithmique des polynômes. Soit un polynôme P défini par :

$$P(X) = a_n X^n + a_{n-1} X^{n-1} + \cdots + a_1 X + a_0 \ .$$

1. On considère la suite de polynômes définie par :

$$\left\{ \begin{array}{rcl} Q_0(X) & = & a_n \\ Q_i(X) & = & XQ_{i-1}(X) + a_{n-i} \ . \end{array} \right.$$

   Vérifier que $Q_n = P$. Etant donné un réel x, calculer en fonction de n le nombre d'opérations (multiplications ou additions) nécessaires pour le calcul de $Q_0(x), \ldots, Q_n(x)$. Comparer avec le nombre d'opérations nécessaires pour le calcul de $P(x)$.
2. Écrire une fonction `Horner_Poly`, qui prend en entrée un polynôme P et un vecteur $x = (x_i)$, et qui retourne en sortie le vecteur des $P(x_i)$, en utilisant l'algorithme de Horner.
3. On considère le polynôme $P(X) = (X-2)^{15}$, le vecteur `x=[1.6:0.0001:2.4]` et les quatre vecteurs des images de `x` par P, calculés comme suit.
   (a) `y1=(x-2).^15`
   (b) `c=coeff((%s-2)^15); y2=zeros(x); for i=1:16, y2=y2+c(i)*x.^(i-1); end;`
   (c) `y3=Horner_Poly(P,x)`
   (d) `y4=horner(P,x)`

   Comparer les temps de calcul des quatre vecteurs. Calculer la différence maximale entre ces vecteurs pris deux à deux.

Exercice 19. Écrire les fonctions suivantes. Toutes prennent en entrée une fonction externe f (de $\mathbb{R}$ dans $\mathbb{R}$), et trois valeurs x_{min}, x_0 et x_{max} (supposées telles que $x_{min} \leq x_0 \leq x_{max}$).

1. `derive` : Elle calcule numériquement et représente graphiquement la dérivée de f sur l'intervalle $[x_{min}, x_{max}]$. Elle retourne la valeur approchée de $f'(x_0)$.
2. `tangente` : Elle représente la fonction f sur l'intervalle $[x_{min}, x_{max}]$, elle superpose sur le même graphique la tangente à f au point x_0, et retourne l'équation de cette tangente comme un polynôme du premier degré.
3. `araignee` : Elle représente la fonction f sur l'intervalle $[x_{min}, x_{max}]$, ainsi que la droite d'équation $y = x$ (première bissectrice). Elle calcule et retourne les 10 premiers itérés de f en x_0 ($x_1 = f(x_0), x_2 = f \circ f(x_0), \ldots$). Elle représente la suite de segments, alternativement verticaux et horizontaux, permettant de visualiser les itérations : segments joignant $(x_0, 0)$, (x_0, x_1), (x_1, x_1), (x_1, x_2), (x_2, x_2), ...
4. `newton` : Elle représente la fonction f sur l'intervalle $[x_{min}, x_{max}]$. Elle calcule et retourne les dix premiers itérés de la suite définie à partir de x_0 par la méthode de Newton : $x_1 = x_0 - f(x_0)/f'(x_0)$, $x_2 = x_1 - f(x_1)/f'(x_1)$ ... Les valeurs de la dérivée sont approchées. La fonction représente sur le même graphique les segments permettant de visualiser les itérations : segments joignant $(x_0, 0)$, $(x_0, f(x_0))$, $(x_1, 0)$, $(x_1, f(x_1))$, $(x_2, 0)$, $(x_2, f(x_2))$,...

Exercice 20. Soit n un entier, $x = (x_i)_{i=1,\ldots,n+1}$ et $y = (y_i)_{i=1,\ldots,n+1}$ deux vecteurs de réels. On appelle ´ polynôme interpolateur de Lagrange ˇ des y_i aux abscisses x_i, le polynôme $P(X)$ défini par :

$$P(X) = \sum_{i=1}^{n+1} y_i \prod_{\substack{1 \le j \le n+1 \\ j \neq i}} \frac{X - x_j}{x_i - x_j} \ .$$

1. Écrire une fonction `poly_Lagrange`, qui prend en entrée deux vecteurs x et y de même taille. Elle retourne le polynôme interpolateur de Lagrange $P(X)$. Elle represente sur le même graphique les points de coordonnées (x_i, y_i) et le graphe du polynome $P(X)$.
2. Tester votre fonction avec : `x=[-5:5]; y=x.^2`, puis `y=x.^3-2*x.^2+4*x+1;` (verifier que `y-horner(P,x)` est proche de 0).
   Tester votre fonction avec plusieurs realisations de `x=rand(1,10); y=rand(1,10);`

Exercice 21.

1. Écrire une fonction `integre_rectangles` qui prend en entrée un vecteur $x = (x_i)_{i=1,\ldots,n}$ d'abscisses et un vecteur $y = (y_i)_{i=1,\ldots,n} = (f(x_i))_{i=1,\ldots,n}$ d'ordonnées et qui calcule l'intégrale approchée de la fonction f par la méthode des rectangles à gauche :
   $$I = \sum_{i=1}^{n-1} f(x_i)(x_{i+1} - x_i) \ .$$
   Comparer les résultats de cette fonction sur plusieurs intégrales, calculées explicitement et avec les fonctions `intrap` et `intsplin`.

2. Même question avec la méthode des trapèzes :

$$I = \sum_{i=1}^{n-1} \frac{1}{2}(f(x_i) + f(x_{i+1}))(x_{i+1} - x_i) \ .$$

Exercice 22.

1. L'intégrale de 0 à $+\infty$ de $\frac{\sin(t)}{t}$ vaut $\frac{\pi}{2}$. Donner une representation graphique convaincante de la fonction. Écrire une suite de commandes qui permette de calculer l'intégrale avec une précision de 10^{-3}, avec chacune des fonctions `inttrap`, `intsplin`, `integ` et `intg`.
2. Même question pour l'intégrale sur $\mathbb{R}$ de $\exp(-t^2)$ qui vaut $\sqrt{\pi}$.

2.3 QCM

Donnez-vous une heure pour répondre à ce questionnaire. Les 10 questions sont indépendantes. Pour chaque question 5 affirmations sont proposées, parmi lesquelles 2 sont vraies et 3 sont fausses. Pour chaque question, cochez les 2 affirmations que vous pensez vraies. Chaque question pour laquelle les 2 affirmations vraies sont cochées rapporte 2 points.

Question 1. La ligne de commande proposée définit et affiche le vecteur ligne `v` formé de tous les entiers de 1 à 100.

A `v=linspace(1,100,100)`
B `v=[1:0.1:100]`
C `v=[]; for i=1:100, v=[v,i]; end; v`
D `v=1; for i=1:100, v=[v,i+1]; end; v`
E `v=[1;1;100]`

Question 2. On définit `A=[1:4;4:-1:1;0:3]` et `v=[2;2;3]`.

A `v'*A*v` retourne un réel.
B `v*v'` retourne une matrice à trois lignes et trois colonnes.
C `v(3:-1:1)'==A(3,:)` retourne `[T T T]'`
D `v(3:-1:1)'==A(:,3)` retourne `[T T T]'`
E `B=A'; B(:,3)==v` retourne `[T T T]`

Question 3. On définit `A=[0:3;3:-1:0]+%i*[1:4;4:-1:1]`. La ligne de commande proposée affiche une matrice réelle à deux lignes et quatre colonnes.

A `A.'`
B `A*conj(A)`
C `real(A)`
D `abs(A)`
E `real(A)<imag(A)`

Question 4. On définit `R=[-2,-2,2,2]`. La ligne de commande proposée affiche un rectangle.

A `x=[-1,1,1,-1,-1];y=[-1,-1,1,1,-1]; plot2d(x,y,rect=R);`

B `x=[-1,-1,1,1];y=[-1,1,-1,1]; plot2d(x,y,rect=R);`

C `x=[-1,-1,1]';y=[-1,1,1]'; plot2d([x,y],[y,x],style=[1,1],rect=R)`

D `x=[-1,-1,1];y=[-1,1,1]; plot2d([x,y],[y,x],style=[1,1],rect=R)`

E
```
x=[-1,-1,1,1;-1,1,1,-1]; y=[-1,1,1,-1;1,1,-1,-1];
plot2d(x,y,style=[1,1,1],rect=R);
```

Question 5. Les lignes de commande suivantes affichent une représentation graphique correcte de la fonction $x \mapsto \frac{1}{\sin(x)}$, pour $x \in]-\pi, +\pi[$.

A
```
e=0.1; x=linspace(e,%pi-e,200); x=[x'-%pi,x'];
y=(1)./sin(x); plot2d(x,y,style=[5,5]);
```

B
```
x=[0:0.001:%pi]; x=[x'-%pi,x'];
y=(1)./sin(x); plot2d(x,y,style=[5,5]);
```

C `x=[-%pi:%pi]; y=(1)./sin(x); plot(x,y);`

D `x=[-%pi+0.1:0.1:%pi-0.1]; y=1/sin(x); plot(x,y);`

E
```
x=[0.1:0.01:%pi-0.1]'; y=(1)./sin(x);
plot2d([x-%pi,x],[-y,y],style=[5,5]);
```

Question 6. Les lignes de commande suivantes :

```
x=linspace(-%pi,%pi,50)'; y=[];
for i=-1:1, y=[y,sin(x.*i)]; end;
plot2d([x,x,x],y);
```

A tracent les courbes d'équation $y = \sin(x)$, $y = 0$ et $y = -\sin(x)$ en noir.

B renvoient un message d'erreur

C tracent les courbes d'équation $y = \sin(x)$, $y = 0$ et $y = -\sin(x)$ avec des couleurs différentes.

D donnent le même résultat que :
```
x=linspace(-%pi,%pi,50)';
y=[sin(-x),zeros(50,1),sin(x)];
plot2d([x,x,x],y,style=[1,2,3]);
```

E tracent seulement la courbe d'équation $y = \sin(x)$.

Question 7. Soit `A` une matrice. La ligne de commande proposée définit la matrice `B` dont le coefficient d'ordre (i, j) est le même que celui de `A` si i est pair, nul sinon.

A `B=kron(A,[0;1])`

B `B=A; for i=2:n, B(i,:)=0`

C `B=zeros(A); for i=1:2:size(A,"r"), B(i,:)=A(i,:); end;`

D `B=A; n=size(A,"r"); B(find((-1)^[1:n]==-1,:)=0`

E `B=(A+A*ones(-A))/2`

Question 8. La ou les lignes proposées ont été sauvegardées dans le fichier `f.sci`, dans le répertoire courant. La commande `getf("f.sci")` permet de définir la fonction `f` qui à deux matrices `A` et `B` de même taille associe la matrice `f(A,B)` dont le coefficient d'ordre (i, j) est le plus petit des coefficients de même ordre de `A` et `B`.

A
```
f(A,B)=min(A,B)
```
B
```
function C=f(A,B)
   C=min(A,B);
endfunction
```
C
```
deff("f(A,B)","min(A,B)")
```
D
```
function C=f(A,B)
   for i=1:size(A), C(i)=min(A(i),B(i));
endfunction
```
E
```
function C=f(A,B)
   C=B; C(A<B)=A(A<B);
endfunction
```

Question 9. Les opérations suivantes définissent la fonction `moins` qui retourne la différence de deux matrices de même taille

A Sauver dans le fichier `f.sci` les lignes
```
function C=f(A,B)
   C=A-B;
endfunction
```
puis charger le fichier par `getf("f.sci")`.

B Sauver dans le fichier `f.sci` la ligne
```
deff("C=moins(A,B)","C=A-B")
```
puis charger le fichier par `getf("f.sci")`.

C Sauver dans le fichier `f.sci` les lignes
```
function C=f(A,B)
   for i=1:n, for j=1:n, C(i,j)=A(i,j)-B(i,j); end; end;
endfunction
```
puis charger le fichier par `getf("f.sci")`.

D Sauver dans le fichier `f.sci` les lignes
```
function C=f(A,B)
   C=A-B
endfunction
```
puis charger le fichier par `getf("f.sci")`.

E Sauver dans le fichier `f.sce` la ligne
```
deff("C=moins(A,B)","C=A-B")
```
puis charger le fichier par `exec("f.sce")`.

Question 10. La ligne de commande proposée retourne une valeur numérique proche de $\int_0^{10} \sqrt{t}\,\mathrm{d}t$.

A
```
integrate("sqrt(x)","x",0,10)
```
B
```
t=[0,10]; inttrap(t,sqrt(t))
```
C
```
t=[0:0.001:10]; inttrap(t,sqrt(t))
```

D `t=[0:0.001:10]; cumsum(sqrt(t))`
E `t=[0:0.001:10]; sum(sqrt(t)/0.001)`

Réponses : 1–AC 2–BD 3–CD 4–AD 5–AE 6–CD 7–CD 8–BE 9–DE 10–AC

2.4 Devoir

Essayez de bien rédiger vos réponses, sans vous reporter ni au cours, ni au corrigé. Si vous souhaitez vous évaluer, donnez-vous deux heures ; puis comparez vos réponses avec le corrigé et comptez un point pour chaque question à laquelle vous aurez correctement répondu.

Questions de cours : On désigne par f la fonction qui à $x \in \mathbb{R}$ associe $x\sin(1/x)$ (prolongée par continuité en 0). Le but de l'exercice est d'en obtenir une représentation graphique sur un intervalle contenant 0, qui puisse être agrandie par zoom au voisinage de 0.

1. Expliquer les raisons pour lesquelles les lignes de commandes suivantes ne conviennent pas, quelle que soit la valeur de `n`.

   ```
   x=linspace(-1,1,n); y=x*sin(1/x);
   plot2d(x,y)
   ```

2. Soit h un paramètre réel strictement positif. Définir le vecteur `x` de valeurs allant de $\pi\lfloor 1/(h\pi)\rfloor + \pi/10$ à $\pi\lfloor 1000/(h\pi)\rfloor + \pi$ par pas de $\pi/10$. Définir le vecteur `X` des inverses des valeurs de `x`, puis le vecteur `Y`, produit des valeurs de X par les sinus de celles de `x`. Expliquer en quoi la commande `plot2d(X,Y)` constitue une représentation graphique satisfaisante de f, et préciser l'intervalle de représentation.
   On note $\lfloor x\rfloor$ la partie entière¨re du nombre réel x.
3. Comment en déduire une représentation satisfaisante sur $[-h, 0[$?
4. On souhaite utiliser ce qui précède pour obtenir une représentation sur $[-h, h]$. Donner les limites du cadre graphique.
5. Écrire une fonction qui prend en entrée un paramètre h strictement positif, qui retourne en sortie $f(h)$ et représente la fonction f sur l'intervalle $[-h, h]$.

Exercice 1 : Soient $x = (x(i))_{i=1,\dots,n}$ et $y = (y(i))_{i=1,\dots,n}$ deux vecteurs. On appelle suite des différences divisées de x par y la suite de vecteurs $d = (d_i)_{i=1,\dots,n}$, définie par $d_1 = y$, et pour $i = 1, \dots, n-1$,

$$\forall j = 1, \dots n-i\,,\ d_{i+1}(j) = \frac{d_i(j+1) - d_i(j)}{x(j+i) - x(j)}\ .$$

1. Définir la fonction `differences_divisees`, qui prend en entrée deux vecteurs lignes `x` et `y` de même taille, et qui retourne en sortie la matrice D, triangulaire inférieure dont les lignes sont les vecteurs de différences d_i, complétés par des zéros.

2. Soit P un polynôme de degré $k \leqslant n-2$. On pose $y = (P(x(i)))_{i=1,\ldots,n}$. Vérifiez expérimentalement que le vecteur d_i est constant pour $i = k+1$, et nul pour $i = k+2, \ldots, n$.
3. Définir la fonction `Newton`, qui prend en entrée deux vecteurs lignes `x` et `y` de même taille, et qui retourne en sortie le polynôme P, de degré $n-1$, défini par :
$$P(X) = d_1(1) + d_2(1)(X - x(1)) + \cdots + d_n(1)(X - x(1)) \cdots (X - x(n-1)) .$$
4. Vérifiez expérimentalement que P est le *polynôme interpolateur de Lagrange* des ordonnées $y(i)$ aux abscisses $x(i)$:
$$\forall i = 1, \ldots, n , \quad P(x(i)) = y(i) .$$
5. Le polynôme interpolateur de Lagrange P peut se calculer aussi par la formule suivante :
$$P(X) = \sum_{i=1}^{n} y_i \prod_{j \neq i} \frac{X - x_j}{x_i - x_j} .$$
Définir la fonction `Lagrange`, analogue à la fonction `Newton`, mais utilisant la nouvelle formule.
6. Vérifiez expérimentalement que `Lagrange` est à la fois plus lente et plus instable numériquement que `Newton`.

Exercice 2 : Étant donnée une matrice symétrique définie positive $A = (a_{i,j})_{i,j=1,\ldots,n}$, sa *décomposition de Cholesky* consiste à écrire $A = L\,{}^tL$, où L est une matrice triangulaire inférieure. Pour obtenir la matrice $L = (l_{i,j})$, on définit d'abord sa première colonne :
$$l_{1,1} = \sqrt{a_{1,1}} , \quad \forall i = 2, \ldots, n , \; l_{i,1} = \frac{a_{i,1}}{l_{1,1}} .$$
Pour $j = 2, \ldots, n$, la j-ième colonne est définie à partir des précédentes :
$$l_{i,j} = \begin{cases} 0 & \forall i = 1, \ldots, j-1 \\ \sqrt{a_{j,j} - \displaystyle\sum_{k=1}^{j-1} l_{j,k}^2} & \text{pour } i = j \\ \dfrac{a_{i,j} - \displaystyle\sum_{k=1}^{j-1} l_{i,k} l_{j,k}}{l_{i,i}} & \forall i = j+1, \ldots, n . \end{cases}$$
1. Définir une fonction `Lt` qui prend en entrée un entier strictement positif n et retourne en sortie la matrice `L` de taille $n \times n$ dont le coefficient d'ordre (i, j) est nul si $i < j$, égal à $i - j + 1$ sinon.
2. Définir une fonction `Lr` qui prend en entrée un entier strictement positif n et retourne en sortie la matrice `L` de taille $n \times n$ dont le coefficient d'ordre (i, j) est nul si $i < j$, égal à un nombre au hasard entre 0 et 1 sinon.

3. Pour `L=Lt(n)` et `L=Lr(n)`, calculer la matrice `A` égale au produit de `L` par sa transposée, puis la norme matricielle de la différence entre `L'` et `chol(A)`. Noter le temps d'exécution de la commande `chol(A)`. Effectuez des essais pour différentes valeurs de `n` et notez vos observations.
4. Définir la fonction `Cholesky3`, qui prend en entrée une matrice symétrique `A`, et qui retourne en sortie la matrice triangulaire inférieure `L`, en utilisant 3 boucles emboîtées : la première pour l'indice de colonne, la seconde pour l'indice de ligne, la troisième pour calculer la somme définissant le numérateur de $l_{i,j}$.
5. Répéter les expériences de la question 3 en remplaçant la fonction Scilab `chol` par `Cholesky3`.
6. Modifier la fonction `Cholesky3` en `Cholesky2` et remplacer la troisième boucle par un calcul utilisant `sum`.
7. Répéter les expériences de la question 3 en renplaçant `chol` par `Cholesky2`.
8. Modifier la fonction `Cholesky2` en `Cholesky1` et remplacer la deuxième boucle par un calcul direct de la j-ième colonne.
9. Répéter les expériences de la question 3 en renplaçant `chol` par `Cholesky1`.

2.5 Corrigé du devoir

Questions de cours :

1. Il y a deux erreurs dans le calcul de `y`. D'une part, la commande correcte est `y=x.*sin((1)./x)`. D'autre part, si `n` est impair, le vecteur `x` contient 0 et le calcul de `y` renvoie un message de division par 0. En outre, les variations de f sont beaucoup plus importantes au voisinage de 0. Définir un vecteur d'abscisses régulièrement espacées, même grand, ne permet pas de zoomer en conservant la même qualité de représentation au voisinage de 0.
2. ```
   h=0.1; uh = 1/h;
   x = %pi*[floor(uh/%pi):0.1:ceil(uh*1000/%pi)]; x(1)=[];
   X = (1)./x;
   Y = X.*sin(x);
   clf();
   plot2d(X,Y);
   ```
   Le vecteur des abscisses `X` a pour valeurs extrêmes des valeurs proches de h et $h/1000$ (si h est petit). Les valeurs sont de plus en plus concentrées au voisinage de 0, de plus les abscisses de la forme $2/(n\pi)$ figurent dans le vecteur, donnant une représentation qui respecte les maxima et minima locaux de f.
3. Du fait de la parité de la fonction f, il suffit de représenter les abscisses opposées avec les mêmes ordonnés.
4. La représentation peut se faire sur le carré $[-h\,;\,+h]^2$.
5. ```
   function y=xsin1(h)
   //
   ```

```
//   Represente le graphe de la fonction y=x*sin(1/x)
//   entre -h et h. Retourne la valeur en h.
//
uh = 1/h;
y = h*sin(uh);
x = %pi*[floor(uh/%pi):0.1:ceil(uh*1000/%pi)]; x(1)=[];
X = (1)./x;
Y = X.*sin(x);
clf();
plot2d(-X,Y,style=5,rect=[-h,-h,h,h],nax=[1,10,1,10]);
plot2d([-X($),0,X($)],[Y($),0,Y($)],style=5,frameflag=0);  // relie parties pa
plot2d(X,Y,style=5,frameflag=0);
endfunction
```

Exercice 1 :

1.
```
function D=differences_divisees(x,y)
//
//     Calcule la matrice des differences divisees
//     du vecteur y par le vecteur x (vecteurs lignes).
//
d = y; D=[d];                // initialisation
n=size(y,"*");               // taille du vecteur
for i=1:n-1,                 // construction des lignes
   dy = d(2:n-i+1)-d(1:n-i); // differences d'ordonnees
   dx = x(1+i:n)-x(1:n-i);   // differences d'abscisses
   d = dy./dx;               // differences divisees
   D=[D;d,zeros(1,i)];       // stocker le resultat
end;
endfunction
```

2.
```
x=gsort(rand(1,10),"c","i"); y=x.^5+3*x^4-2*x^2+x-1;
D=differences_divisees(x,y)
```

3.
```
function P = Newton(x,y)
//
//     Retourne le polynome d'interpolation
//     des valeurs y aux points x
//     (x et y sont deux vecteurs lignes de meme taille).
//     Le calcul utilise les differences divisees.
//
n = size(x,"*");               // nombre de points
d = y;                         // differences divisees
s = n;                         // longueur
P = y(1);                      // initialisation du polynome
f = 1;                         // produit de facteurs
for k=1:n-1,
```

```
    d = d([2:s])-d([1:s-1]);    // nouveau vecteur de differences
    a = x([n-s+2:n])-x([1:s-1]);// differences d'abscisses
    d = d./a;                   // diviser par les differences
    s = s-1;                    // la taille diminue
    f = f*(%s-x(k));            // nouveau produit
    P = P+d(1)*f;               // ajouter au polynome
end;
endfunction
```

4. `x=gsort(rand(1,5),"c","i"); y=x.^5+3*x^4-2*x^2+x-1;`
   `P=Newton(x,y); horner(P,x)-y`
5.
```
function P = Lagrange(x,y)
//
//     Retourne le polynome d'interpolation
//     des valeurs y aux points x
//     (x et y sont deux vecteurs lignes de meme taille).
//     Le calcul utilise les polynomes cardinaux.
//
n = size(x,"*");              // nombre de points
P = 0;                        // initialisation du polynome
for k=1:n                     // parcourir les abscisses
   z = x;                     // recopier le vecteur d'abscisses
   z(k)=[];                   // supprimer la k-ieme
   L = prod((%s-z)./(x(k)-z)); // k-ieme polynome cardinal
   P = P+y(k)*L;              // multiplier par la valeur et ajouter
end;
endfunction
```
6.
```
x=gsort(rand(1,1000),"c","i"); y=rand(1,1000);
timer(); Newton(x,y); tN=timer()
timer(); Lagrange(x,y); tL=timer()

x=gsort(rand(1,10),"c","i"); c=rand(1,10);
P=poly(c,"X","coeff"); y=horner(P,x);
PN=Newton(x,y); norm(coeff(PN)-c)
PL=Lagrange(x,y); norm(coeff(PL)-c)
```

Exercice 2 :

1. `deff("L=Lt(n)","L=toeplitz([1:n],[1,zeros(1,n-1)])")`
2. `deff("L=Lr(n)","L=tril(rand(n,n))")`
3. `n=100; L=Lt(n); A=L*L';`
   `timer(); d=norm(L-chol(A)'); t=timer(); [d,t]`
   Avec `Lt`, les matrices `L` et `A` sont à valeurs entières. La décomposition de Cholesky est numériquement stable : la norme de la différence est nulle, même pour de grandes valeurs de n. Le temps d'exécution observé (dépendant de la machine) est de l'ordre de 0.03 s pour $n = 100$, 1.8 s pour $n = 1000$.

Avec `Lr`, la décomposition de Cholesky est numériquement instable : la norme de la différence peut être très variable. Un message d'erreur sur le fait que la matrice `A` n'est pas définie positive peut apparaître dès $n = 50$.

4.
```
function L = Cholesky3(A)
//
//      Pour une matrice symetrique A, la matrice L
//      est triangulaire inferieure et A=L*L'
//      3 boucles emboitees.
//
L=zeros(A);             // initialisation : matrice nulle
n = size(A,"r");        // nombre de lignes
dia = sqrt(A(1,1));     // coefficient diagonal
L(:,1) = A(:,1)/dia;    // premiere colonne

for j=2:(n-1),          // colonnes suivantes
   dia = A(j,j);
   for l=1:(j-1),
      dia = dia - L(j,l)^2;
   end;
   dia = sqrt(dia);
   L(j,j) = dia;        // coefficient diagonal
   for i=(j+1):n,
       aux = A(i,j);
       for l=1:(j-1),
          aux = aux - L(i,l)*L(j,l);
       end;
       aux = aux/dia;
       L(i,j) = aux;
   end;
end;

dia = A(n,n);
for l=1:(n-1),
   dia = dia -L(n,l)^2;
end;
dia = sqrt(dia);
L(n,n) = dia;           // dernier coefficient
endfunction
```

5. Les mêmes propriétés de stabilité ou instabilité numérique sont observées, le temps d'exécution est de 0.5 s pour $n = 100$, 277 s pour $n = 1000$.

6.
```
function L = Cholesky2(A)
//
//      Pour une matrice symetrique A, la matrice L
//      est triangulaire inferieure et A=L*L'
```

```
//      2 boucles emboitees.
//
L=zeros(A);             // initialisation : matrice nulle
n = size(A,"r");        // nombre de lignes
dia = sqrt(A(1,1));     // coefficient diagonal
L(:,1) = A(:,1)/dia;    // premiere colonne

for j=2:n-1,            // colonnes suivantes
   dia = sqrt(A(j,j)-sum(L(j,[1:j-1]).^2));
   L(j,j) = dia;        // coefficient diagonal
   for i=j+1:n,
       L(i,j) = (A(i,j) - sum(L(i,[1:j-1]).*L(j,[1:j-1])))/dia;
   end;
end;

dia = sqrt(A(n,n)-sum(L(n,[1:n-1]).^2));
L(n,n) = dia;           // dernier coefficient
endfunction
```

7. Le temps d'exécution est de 11 s pour $n = 1000$.

8.
```
function L = Cholesky1(A)
//
//      Pour une matrice symetrique A, la matrice L
//      est triangulaire inferieure et A=L*L'
//      1 seule boucle.
//
L=zeros(A);             // initialisation : matrice nulle
n = size(A,"r");        // nombre de lignes
dia = sqrt(A(1,1));     // coefficient diagonal
L(:,1) = A(:,1)/dia;    // premiere colonne

for j=2:n-1,            // colonnes suivantes
   dia = sqrt(A(j,j)-sum(L(j,[1:j-1]).^2));
   L(j,j) = dia;        // coefficient diagonal
   L([j+1:n],j) = (A([j+1:n],j) - ..
     sum(L([j+1:n],[1:j-1]).*(ones(n-j,1)*L(j,[1:j-1])),"c"))/dia;
end;

dia = sqrt(A(n,n)-sum(L(n,[1:n-1]).^2));
L(n,n) = dia;           // dernier coefficient
endfunction
```

9. Le temps d'exécution est de 6.75 s pour $n = 1000$.

3 Compléments

3.1 Un voyage à Anticythère

Nous sommes en octobre 1900, et le mauvais temps sévit en Méditerranée. De retour d'Afrique, le capitaine Dimitrios Kondos décide de laisser passer la tempête sur l'île d'Anticythère, entre Cythère et la Crète. Bien équipés pour la plongée avec leurs costumes de toile et leurs casques de cuivre, ses hommes passent le temps en pêchant des éponges, quand l'un d'eux tombe sur une épave. Datant du premier siècle av. J.C., celle-ci révèle une cargaison riche et assez hétéroclite, probablement le butin d'un général romain de retour d'Athènes. La découverte la plus étonnante fut faite vers la fin de la fouille par les archéologues : un mécanisme comprenant trois cadrans et des dizaines d'engrenages. Plusieurs théories se sont affrontées depuis sur l'usage de cette machine, mais les reconstitutions les plus récentes ont permis d'établir qu'elle effectuait toutes sortes de calculs astronomiques, entre autres la prédiction des éclipses, en tenant compte des théories les plus sophistiquées de l'époque[1]. On savait par les écrits de Cicéron, qu'Archimède avait construit des dispositifs mécaniques visualisant le mouvement apparent du soleil, de la lune et des cinq planètes connues à l'époque. Mais ces machines n'avaient pas été conservées et personne n'imaginait que les dispositifs mécaniques de calcul astronomique aient pu atteindre il y a si longtemps la sophistication et la complexité du mécanisme d'Anticythère.

Des horloges et des automates élaborés ont donc été construits dès l'antiquité. Ils furent portés à un très haut degré de perfection par les savants arabes à partir du IX^{e} siècle. Parmi ces inventeurs, la palme du courage revient à Abbas ibn Firnas (810–887), qui vivait en Andalousie. Voici ce qu'en dit Al Maqqari, historien marocain du XVII^{e} siècle.

> [Il] fut le premier à fabriquer du verre à partir du sable et en établit des fabriques en Andalousie. Il passe aussi pour être le premier à avoir introduit dans ce pays le fameux traité de prosodie de Khalil, et à y avoir enseigné la science de la musique. Il inventa un instrument appelé *al-minkálah*, par lequel on marquait le temps en musique sans avoir recours à des notes ou des figures. Entre autres expériences très curieuses qu'il fit, il essaya de voler. Il se couvrit de plumes à cet effet, attacha une paire d'ailes à son corps, et montant sur une éminence, il se lança dans l'air. Selon le témoignage de plusieurs écrivains dignes de confiance témoins de l'exploit, il vola sur une distance considérable, comme s'il avait été un oiseau ; mais au moment de se poser à l'endroit d'où il était parti, son dos fut grièvement blessé, car il ne savait pas que les oiseaux quand ils se posent arrivent sur leur queue, et il avait oublié de s'en procurer une. Múmen Ibn Saíd a dit, dans un vers où il évoque cet homme extraordinaire, ´ Il surpassa en vitesse le vol de l'autruche, mais il négligea de munir son corps de la force du vautour ˇ.
>
> Le même poète a aussi fait allusion à une représentation des cieux que

[1] T. Freeth et al. : Calendars with Olympiad display and eclipse prediction on the Antikythera Mechanism. *Nature, 454 p.614–617 (2008)*

> cet ibn Firnas, qui était un astronome consommé, avait fabriquée dans sa maison, et où les spectateurs avaient l'impression qu'ils voyaient les nuages, les étoiles, les éclairs, et qu'ils entendaient le bruit terrible de la foudre. ´ Les cieux de Abú-l-kásim Ábbás, le savant, feront fortement impression sur ton esprit par l'étendue de leur perfection et de leur beauté. Tu entendras le tonnerre gronder, des éclairs croiseront ta vue, et par Allah, le firmament même temblera sur ses fondations. Mais ne vas pas au sous-sol, sauf si tu y tiens, car je l'ai fait et en voyant la tromperie j'ai craché au visage de son créateur ˇ.
>
> Le vers qui suit est de la main même de ibn Firnas, qui l'avait adressé à l'Émir Mohammed. ´ J'ai vu le Prince des croyants, Mohammed, et l'étoile florissante de la bienveillance brillait sur son front ˇ. Ce à quoi Múmen répondit, quant il en eut connaissance, ´ Oui, tu as raison, mais elle s'évanouit au moment même où tu t'en es approché ; tu as fait de la face du Calife un champ où les étoiles fleurissent ; hélas tu en as fait aussi un tas d'excréments, car les plantes ne poussent pas sans fumier ˇ.

Difficile d'en déduire ce qui était le plus dangereux dans l'Andalousie de l'époque : se jeter dans le vide, tromper ses visiteurs, ou déplaire au Calife ?

3.2 La pascaline

Le dispositif de calcul mécanique le plus ancien est le boulier, qui est la traduction portable des abaques ; il est encore en usage de nos jours. Il permet d'effectuer assez facilement des additions et des soustractions, mais devient plus difficile à manipuler quand il s'agit de multiplier et de diviser. Néanmoins, quand les premières calculatrices électroniques sont apparues, des compétitions de vitesse ont été organisées avec des spécialistes du boulier : l'électronique ne gagnait pas encore à tous les coups ! L'inventeur des logarithmes, Napier, avait aussi proposé un dispositif de réglettes sur lesquelles étaient disposés les résultats des multiplications de chiffres entre eux, qu'il suffisait de déplacer les uns par rapport aux autres pour transformer les multiplications en suites d'additions. Ces dispositifs furent bientôt perfectionnés en règles et cercles à calcul, grâce aux logarithmes. Même si les premières horloges mécaniques européennes sont apparues dès le XIII^{e}, les premières machines à engrenages capables d'additionner ne furent inventées qu'au XVII^{e} avec Wilhelm Schikard (1623) et Blaise Pascal (1642). Le 22 mai 1649 Louis XIV (ou plutôt sa mère la régente, car le roi n'avait que 10 ans) accorde à Blaise Pascal un ´ privilège royal ˇ pour la production de ses machines.

> Notre cher et bien aimé le Sr Pascal nous a fait remontrer qu'à l'invitation du Sr Pascal, son père, nostre Conseiller en nos conseils, et président en notre Cour des Aydes d'Auvergne, il auroit eu, dès ses plus jeunes années, une inclination particulière aux sciences Mathématiques, dans lesquelles par ses études et ses observations, il a inventé plusieurs choses, et particulièrement une machine, par le moyen de laquelle on peut faire toutes sortes de supputations, Additions, Soustractions, Multiplications, Divisions, et toutes les autres Règles d'Arithmétique, tant en nombre entier que rompu, sans

> se servir de plume ni jettons, par une méthode beaucoup plus simple, plus facile à apprendre, plus prompte à l'exécution, et moins pénible à l'esprit que toutes les autres façons de calculer, qui ont été en usage jusqu'à présent ; et qui outre ces avantages, a encore celuy d'estre hors de tout danger d'erreur, qui est la condition la plus importante de toutes dans les calculs. De laquelle machine il avoit fait plus de cinquante modèles, tous differens, les uns composez de verges ou lamines droites, d'autres de courbes, d'autres avec des chaisnes les uns avec des rouages concentriques, d'autres avec des excentriques, les uns mouvans en ligne droite, d'autres circulairement, les uns en cones, les autres en cylindres, et d'autres tous différens de ceux-là, soit pour la matière, soit pour la figure, soit pour le mouvement : de toutes lesquelles manières différentes l'invention principale et le mouvement essentiel consistent en ce que chaque roue ou verge d'un ordre faisant un mouvement de dix figures arithmétiques, fait mouvoir sa prochaine d'une figure seulement. Après tous lesquels essais auxquels il a employé beaucoup de temps et de frais, il seroit enfin arrivé à la construction d'un modèle achevé qui a été reconnu infaillible par les plus doctes mathématiciens de ce temps, qui l'ont universellement honoré de leur approbation et estimé très utile au public.

´ ...chaque roue ou verge d'un ordre faisant un mouvement de dix figures arithmétiques, fait mouvoir sa prochaine d'une figure seulement ˇ : ce principe de base était encore utilisé récemment pour les compteurs kilométriques.

Reprenant le mécanisme de Pascal, et lui ajoutant l'invention fondamentale de cylindres cannelés à dents progressives, Leibniz (1646–1716) conçoit en 1671 une machine élaborée, à laquelle il travaillera tout au long de sa vie. Les difficultés mécaniques étaient telles que des machines à calculer véritablement opérationnelles ne seront commercialisées qu'au XIX^{e} siècle. Une fois de plus, Leibniz s'était montré remarquablement visionnaire : ´ Il est indigne d'hommes éminents de perdre des heures comme esclaves dans le travail de calcul, qui pourrait sûrement être confié à n'importe qui, si des machines étaient utilisées ˇ.

3.3 Les machines à calculs

À partir du XIX^{e} siècle, toutes sortes d'opérations mathématiques furent mécanisées. Les *planimètres*, conçus dès 1814, étaient formés de tiges articulées reliées à des cadrans ; ils servaient à évaluer des aires planes, c'est-à-dire calculer des intégrales. En 1836 Gaspard Coriolis imagine une machine pour calculer des solutions de l'équation différentielle $y'(x) = f(x, y)$, à l'aide d'´ une surface exécutée en relief ayant pour ordonnée $y/f(x, y)$ ˇ. Il la fait réaliser pour $f(x, y) = ay$ (calcul de l'exponentielle)[2].

> Si l'on conçoit qu'un fil tendu s'enroule sur un cylindre, et que le frottement y soit assez fort pour empêcher ce fil de glisser le long de la surface contre

[2]G. Coriolis : Note sur un moyen de tracer des courbes données par des équations différentielles. *Journal de Mathématiques Pures et appliquées. serie I(1) p. 5–9 (1836)*

> laquelle il s'est enroulé, la courbe formée par le fil sur la surface du cylindre, développée ensuite sur un plan, jouira de la propriété que la direction de sa tangente sera toujours celle de la partie du fil tendue en ligne droite avant qu'elle s'enroule.
>
> Si donc on peut donner au fil, dans cette partie, une direction qui résulte de l'équation différentielle d'une courbe, celle-ci se trouvera tracée sur le cylindre en prenant pour abscisse les arcs comptés sur la base du cylindre. Cette considération conduit à un tracé assez simple de plusieurs courbes.
>
> [...]
>
> J'ai fait construire, d'après cette remarque, une machine au moyen de laquelle un fil tendu par un léger poids s'enroule ou se déroule autour d'un cylindre en passant par un petit trou percé dans une plaque mobile qu'on approche ou qu'on écarte à volonté du cylindre. Une aiguille et un cadran indiquent les tours et les fractions de tours dont on a tourné le cylindre. Ce sont ces quantités qui représentent les exposants.

Le principe de toutes les machines à calculs est remarquablement résumé par Leonardo Torres y Quevedo (1852–1936)[3], ingénieur espagnol inventeur entre autres d'une machine qui calculait les racines des polynômes, et d'une autre qui résolvait des fins de partie d'échecs.

> Une machine est un intrument qui relie plusieurs mobiles et qui impose mécaniquement certaines relations entre les valeurs simultanées de leurs déplacements. Ces relations se traduisent ordinairement en une ou plusieurs équations et on peut dire de manière appropriée qu'en construisant la machine, on construit les équations établies entre les valeurs des déplacements considérés. Il suffit de prendre en compte cette analogie pour comprendre la possibilité d'obtenir des machines qui exécutent certains calculs algébriques.

Jusqu'à la seconde guerre mondiale, les machines à calcul ont connu un développement impressionnant. L'´ intégrateur à eau ˇ, construit en Union Soviétique en 1936 résolvait des équations différentielles. Il occupait toute une salle, pleine de tuyaux et de pompes. Le niveau d'eau dans les divers réservoirs, mesuré avec une grande précision, représentait des enregistrements de nombres, et les flux entre ces réservoirs traduisaient les opérations mathématiques. La machine électrique de Mallock, contruite aux États-Unis en 1933 résolvait des systèmes différentiels comprenant jusqu'à 10 équations couplées.

3.4 La manufacture à logarithmes

Exercice : évaluez l'âge canonique de l'auteur sachant que, comme tous ses camarades à l'époque, il a commencé par se munir d'une règle à calcul et de tables de logarithmes avant d'entamer ses études de mathématiques. Comptez pendant combien de siècles mathématiciens, ingénieurs et astronomes n'ont eu à leur disposition que des

[3] L. Torres y Quevedo : Memoria sobre las máquinas algébricas. *Revista de Obras Públicas, (26–33) (1895)*

tables de valeurs numériques pour les aider dans leurs calculs, sachant que les premières tables trigonométriques sont celles d'Hipparque de Nicée (IIe siècle av. J.C.).

La puissance des logarithmes, inventés par John Napier en 1614, est de transformer les produits en sommes : $ab = \exp(\log(a) + \log(b))$. Mais cela n'a d'intérêt que si on peut calculer simplement $\log(a)$ et $\log(b)$ puis l'exponentielle de la somme : d'où la nécessité de tables précises permettant de lire les valeurs des logarithmes sans calcul supplémentaire. Tout au long des XVIIe et XVIIIe siècles, des tables de plus en plus perfectionnées ont été calculées et publiées.

Comment ? Du temps de Napier, on utilisait simplement la définition : le logarithme en base b d'un nombre x est le nombre l tel que $x^l = b$. Prenez par exemple le nombre $x = 1.001$ et multipliez-le par lui-même de façon répétée. Ce n'est pas difficile, pour multiplier un nombre a par x, il suffit d'ajouter a avec son décalage de 3 décimales, en plaçant bien la virgule. Arrêtez-vous dès que le produit dépasse $b = 10$.

Déjà fini ? C'est bien ! Vous avez fait 2304 multiplications. Vous êtes sûr que vous n'avez pas triché ?

```
x=1.001; p=x; P=[p]; n=1; while p<10, p=p*x; P=[P,p]; n=n+1; end;
```

Maintenant divisez tous les entiers de 1 à 2304 par 2304 : $k/2304$ est une approximation du logarithme en base 10 de x^k ; pas trop mauvaise d'ailleurs.

```
norm([1:n]/n-log10(P))
```

Pour pas beaucoup plus cher, améliorez sérieusement le résultat avec une interpolation linéaire, qui fournit un diviseur mieux placé entre 2303 et 2304 :

```
d=n-(P(n)-10)/(P(n)-P(n-1))
norm([1:n]/d-log10(P))
```

Vous avez maintenant une approximation des logarithmes décimaux de 2304 nombres entre 1 et 10. Des interpolations linéaires entre ces valeurs peuvent vous en fournir d'autres. Tel est le principe des premières tables de Napier, perfectionnées par Briggs en 1617 (1000 logarithmes à 14 décimales) puis 1624 (30000 logarithmes à 14 décimales). Le courageux Briggs produira aussi entre autres des tables de logarithmes de sinus et tangentes, précises au 100e de degré.

Les techniques de calcul changent avec la publication, le 5 juillet 1687 des ´ Philosophiæ Naturalis Principia Mathematica ˇ d'Isaac Newton (1642–1727). Le lemme V du livre III, pages 695 et 696 décrit une méthode pour ´ trouver une ligne courbe de genre parabolique qui passe par un nombre quelconque de points ˇ. C'est la méthode des différences divisées que nous vous avons proposée en devoir. Elle permet de ramener le calcul d'une série de valeurs d'une fonction polynomiale à une succession d'additions. Il suffit alors d'approcher une fonction quelconque à tabuler par un polynôme assurant la précision désirée, puis de calculer les valeurs de ce polynôme par différences successives.

Parmi les œuvres de la révolution, l'établissement du système métrique fut une entreprise d'une ambition difficilement imaginable. Outre la définition des nouvelles mesures, le passage systématique au système décimal impliquait entre autres que l'on

recalcule les tables trigonométriques : l'angle droit, jusque-là divisé en 90 degrés, 90×60 minutes et $90 \times 60 \times 60$ secondes, se composait désormais de 100 grades, subdivisés en puissances de 10. La mission de calculer de nouvelles tables fut confiée au directeur du bureau du cadastre, Gaspard-Clair-François-Marie Riche, Baron de Prony, devenu plus simplement le citoyen Prony. Il dit avoir été inspiré par les travaux d'Adam Smith sur la division du travail en tâches homogènes répétées, et en particulier par son exemple de la fabrique d'épingles (1776).

> One man draws out the wire, another straights it, a third cuts it, a fourth points it, a fifth grinds it at the top for receiving the head : to make the head requires two or three distinct operations : to put it on is a particular business, to whiten the pins is another...and the important business of making a pin is, in this manner, divided into about eighteen distinct operations, which in some manufactories are all performed by distinct hands, though in others the same man will sometime perform two or three of them.

Peut-être Prony et Smith avaient-ils lu l'article ´ Épingle ˇ de l'Encyclopédie de Diderot et d'Alembert, qui décrit dans les moindres détails les 18 opérations nécessaires.

> Cet article est de M. Delaire, qui décrivoit la fabrication de l'épingle dans les atteliers même des ouvriers, sur nos desseins, tandis qu'il faisoit imprimer à Paris son analyse de la philosophie sublime & profonde du chancelier Bacon ; ouvrage qui joint à la description précédente, prouvera qu'un bon esprit peut quelquefois, avec le même succès, & s'élever aux contemplations les plus hautes de la Philosophie, & descendre aux détails de la méchanique la plus minutieuse. Au reste ceux qui connoîtront un peu les vûes que le philosophe anglois avoit en composant ses ouvrages, ne seront pas étonnés de voir son disciple passer sans dédain de la recherche des lois générales de la nature, à l'emploi le moins important de ses productions.

´ Je ferai mes calculs comme on fait des épingles ˇ décide Prony. Ainsi naquit la ´ Manufacture à Logarithmes[4] ˇ. Prony organisa le travail en trois groupes. Le premier était celui des mathématiciens (parmi lesquels Legendre), chargés d'établir les formules d'approximation polynomiale garantissant la précision souhaitée, de calculer les tables de différences, bref d'établir les algorithmes de calcul. Le deuxième groupe organisait les calculs pour le troisième, en préparant des formulaires destinés à recevoir les résultats intermédiaires. Il compilait ensuite les valeurs obtenues et réalisait le manuscrit définitif. Le troisième groupe était celui des calculateurs, qui passaient leurs journées à effectuer les additions prescrites et à en noter les résultats. Il est difficile de savoir exactement combien il y eut de ces calculateurs, probablement plusieurs dizaines. Ils étaient répartis en deux groupes, produisant deux manuscrits originaux dont les résultats furent ensuite confrontés pour détecter les erreurs. La Manufacture à Logarithmes fit merveille : à raison de plusieurs centaines de valeurs calculées par jour, avec des précisions allant jusqu'à 24 chiffres, plusieurs milliers de pages furent produites. Le Comité de Salut Public avait décrété le 11 mai 1794 que 10000 exemplaires des tables

[4] D. Roegel : The great logarithmic and trigonometric tables of the French Cadastre : a preliminary investigation `http://locomat.loria.fr/cadastre/analysis.pdf` *(2011)*

devraient être imprimés aux frais de la République. Ils ne sont jamais sortis : les problèmes politiques, les coûts et les difficultés techniques de l'entreprise s'accumulant, les tables ne dépassèrent pas le stade des deux manuscrits qui nous sont parvenus.

Même si elle ne déboucha sur aucune utilisation pratique, cette extraordinaire entreprise eut néanmoins une influence indirecte sur l'histoire de l'informatique. L'anglais Charles Babbage (1791–1871), lui même auteur de tables de logarithmes, en avait eu connaissance. Il avait conçu l'idée de remplacer les calculateurs de Prony par les rouages d'une machine : sa ´ machine à différences ˇ devait supprimer toute intervention humaine source d'erreurs, et réaliser l'ensemble des opérations jusqu'à composer automatiquement les matrices des pages pour l'impression des tables. Cette machine ne vit jamais le jour du vivant de son inventeur, pas plus que la ´ machine analytique ˇ, beaucoup plus ambitieuse, qu'il conçut ensuite : dommage, elle aurait été le premier ordinateur.

3.5 Que calcule la calculette ?

Vous êtes-vous demandé comment fait votre calculette pour retourner instantanément avec 10 chiffres de précision n'importe quelle valeur d'une fonction usuelle ? À la lecture de ce qui précède, vous êtes peut-être tenté de croire à une approximation polynomiale, calculée par la méthode des différences. Non : votre calculette utilise probablement l'algorithme CORDIC (pour Coordinate Rotation Digital Computer), inventé par J. Volder en 1959, et généralisé par J. Walther en 1971[5].

Voici le principe, pour les fonctions trigonométriques. Soit à calculer $\sin(\theta)$ et $\cos(\theta)$, pour un certain angle θ. L'idée consiste à approcher θ par la suite $(\theta_n)_{n\in\mathbb{N}}$, définie par $\theta_0 = 0$ et pour tout $n \geqslant 0$:

$$\theta_{n+1} = \begin{cases} \theta_n & \text{si} \quad \theta = \theta_n \\ \theta_n + \arctan(2^{-n}) & \text{si} \quad \theta > \theta_n \\ \theta_n - \arctan(2^{-n}) & \text{si} \quad \theta < \theta_n \end{cases}$$

La série de terme général $\arctan(2^{-n})$ est convergente. De plus, pour tout $n \in \mathbb{N}$:

$$\arctan(2^{-n}) < \sum_{i=n+1}^{+\infty} \arctan(2^{-i}) \; .$$

Démontrez-le, et déduisez-en que la suite (θ_n) converge vers θ, pour tout θ compris entre deux valeurs extrêmes :

$$-\sum_{n=0}^{+\infty} \arctan(2^{-n}) \leqslant \theta \leqslant +\sum_{n=0}^{+\infty} \arctan(2^{-n}) \; .$$

Ce n'est pas une limitation gênante : comme vous le savez, on peut toujours se ramener à un angle compris entre 0 et $\pi/2$, quitte à changer ensuite le signe du résultat.

[5] J. Walther : A unified algorithm for elementary functions, *Joint Computer Conference Proceedings, 38 p. 379–385 (1971)*

```
n=[1:100]; sum(atan(2^(-n)))/%pi
```

Les angles $\arctan(2^{-n})$ sont fixes : ils peuvent être tabulés et mémorisés. Peu de valeurs sont nécessaires, car pour n assez grand, 2^{-n} est une bonne approximation de $\arctan(2^{-n})$.

```
d=2^(-[0:10]); atan(d)-d
```

Notons α_n l'angle $\theta_{n+1} - \theta_n = \pm\arctan(2^{-n})$. Les valeurs approchant le cosinus et le sinus de θ sont $x_n = \cos(\theta_n)$, $y_n = \sin(\theta_n)$. Le vecteur de coordonnées (x_{n+1}, y_{n+1}) se déduit du précédent par une rotation d'angle α_n (d'où le nom de l'algorithme) :

$$\begin{pmatrix} x_{n+1} \\ y_{n+1} \end{pmatrix} = \begin{pmatrix} \cos(\alpha_n) & -\sin(\alpha_n) \\ \sin(\alpha_n) & \cos(\alpha_n) \end{pmatrix} \begin{pmatrix} x_n \\ y_n \end{pmatrix} = \cos(\alpha_n) \begin{pmatrix} 1 & -\tan(\alpha_n) \\ \tan(\alpha_n) & 1 \end{pmatrix} \begin{pmatrix} x_n \\ y_n \end{pmatrix}$$

Pour tout $n \in \mathbb{N}$, posons

$$K_n = \prod_{i=0}^{n-1} \cos(\alpha_i) = \prod_{i=0}^{n-1} \cos(\arctan(2^{-i})) = \prod_{i=0}^{n-1} \frac{1}{\sqrt{1+2^{-2i}}} \, .$$

Posons $(X_0, Y_0) = (x_0, y_0) = (1, 0)$, et pour tout $n \geqslant 1$, $(X_n, Y_n) = K_n^{-1}(x_n, y_n)$. Il vient :

$$\begin{pmatrix} X_{n+1} \\ Y_{n+1} \end{pmatrix} = \begin{pmatrix} 1 & -\tan(\alpha_n) \\ \tan(\alpha_n) & 1 \end{pmatrix} \begin{pmatrix} X_n \\ Y_n \end{pmatrix}$$

La puissance de l'algorithme réside dans cette formule de récurrence : elle ne comporte que des additions, et des multiplications par $\tan(\alpha_n)$. Or par définition de α_n, $\tan(\alpha_n) = \pm 2^{-n}$. Multiplier par $\tan(\alpha_n)$ en arithmétique binaire, c'est décaler la virgule de n places, et éventuellement inverser le bit de signe. Le calcul de (X_n, Y_n) est donc peu coûteux. Pour revenir à (x_n, y_n), il faut multiplier par K_n : les valeurs de K_n sont fixes et peuvent être mémorisées. Peu de valeurs sont nécessaires car le produit converge vite.

```
cumprod(cos(atan(2^(-[0:15]))))
```

Voici en Scilab le calcul de sin(1) et cos(1) (non optimisé bien sûr).

```
theta=1; tn=0; v=[1;0]; V=[]; n=40;
for i=0:n,
    s=sign(theta-tn); tn=tn+s*atan(2^(-i));
    b=s*2^(-i); M=[1,-b;b,1]; v=M*v; V=[V,v];
end;
K=cumprod(cos(atan(2^(-[0:n])))); sc=V.*[K;K]
sc-[cos(theta);sin(theta)]*ones(1,n+1)
```

Pour les autres fonctions usuelles, il faut définir une autre notion de ´ rotation ˇ, mais le principe reste le même : une récurrence linéaire peu coûteuse, éventuellement suivie d'une multiplication par des valeurs mémorisées.