Analyse Numérique : Introduction à Matlab

Ecole des Mines de Nancy

Xavier ANTOINE [1] [2]

1. Ecole Nationale Supérieure des Mines de Nancy, Département de Génie Industriel, Parc de Saurupt, CS 14 234, 54042 Nancy cedex, France.

2. Institut Elie Cartan Nancy (IECN), Université Henri Poincaré Nancy 1, INRIA, Bureau 307, B.P. 239, F-54506 Vandoeuvre-lès-Nancy Cedex, France.

Table des matières

0.1 Preface

Ces notes sont constituées en grande partie d'une traduction du livre "Getting Started with MATLAB" disponible dans la documentation de Matlab ou comme fichier pdf.

0.2 Introduction

0.2.1 Qu'est ce que MATLAB ?

MATLAB est un langage hautes performances pour le calcul scientifique et technique. Il intègre la possibilité de calculs, de visualisation et de programmation dans un environnement très simple d'emploi. Les résultats sont exprimés sous une forme mathématique standard. L'utilisation typique est

- Calcul scientifique
- Développement d'algorithmes
- Acquisition de données
- Modélisation et simulation

– Analyse de données, exploration et visualisation
– Graphisme scientifique
– Développement d'applications, interface graphique (gui)

MATLAB est un système interactif dont la brique de base est un tableau dont la taille n'est pas nécessairement connue. Ceci permet de résoudre des problèmes, en particulier ceux qui ont une formulation matricielle, en un minimum de temps (contrairement aux langages de bas niveau comme le C ou le fortran). Le nom de MATLAB est un résumé de "Matrix Laboratory". MATLAB a été à l'origine développé pour avoir un accès simple et rapide aux projets EISPACK et LINPACK. Aujourd'hui, MATLAB intègre les librairies LAPACK et BLAS, incorporant ainsi les dernières techniques pour le calcul matriciel.

Dans l'enseignement universitaire, MATLAB s'est imposé comme un standard pour l'apprentissage de l'algorithmique scientifique. Dans l'industrie, il est l'outil de choix pour une productivité accrue en recherche et développement.

MATLAB peut aussi être enrichi à l'aide de Toolbox (boites à outils) pour des problèmes spécifiques.

0.2.2 Le système MATLAB

Le système MATLAB consiste en cinq parties majeures :

Environnement de développement . C'est un ensemble d'outils pour l'utilisation des fonctions MATLAB et des fichiers. La plupart de ces outils sont des interfaces graphiques. Ils incluent le bureau MATLAB et la fenêtre de commande, un historique des commandes, un éditeur et un débuggeur, et un navigateur pour voir l'aide, l'espace de travail (workspace), les fichiers, et le chemin.

La librairie de fonctions MATLAB C'est une grande collection d'algorithmes de calcul allant de fonctions élémentaires comme les sommes, les sinus et cosinus et l'arithmétique complexe, jusqu'aux fonctions plus sophistiquées comme l'inverse de matrices, le calcul de valeurs propres, les fonctions de Bessel et la transformée de Fourier.

Le langage MATLAB C'est un langage de haut niveau sous forme matrice/vecteur. Il comporte des structures de contrôles, des fonctions, des structures de données, des fonctions d'entrées-sorties et une programmation orientée objet. Il permet le développement de petites applications avec un code simple ou de grandes application industrielles.

Graphisme MATLAB possède un grand choix de fonctions pour faire afficher les vecteurs et les matrices comme graphes. Il permet aussi de compléter les graphiques avec des légendes. Il inclue des fonctions de visualisation 2D et 3D de haut niveau. Il est même possible de fabriquer de petites IHM (interfaces hommes machines)

L'API MATLAB L'API (Application Program Interface) est une librairie qui permet de développer ses propres programmes optimisés en C et en Fortran et de les faire interagir avec MATLAB. Elle inclut des fonctions pour l'appel de routines à partir de MATLAB (liaison dynamique) ou l'appel de MATLAB comme moteur de calcul.

0.2.3 La Documentation MATLAB

MATLAB fournit une documentation détaillée à la fois en ligne et sous forme de fichiers pdf. L'aide en ligne est la plus simple à manipuler. Il suffit de sélectionner l'aide MATLAB dans le menu d'aide (Help menu) directement dans MATLAB. L'aide MATLAB est organisée en

Environnement de développement fourni l'aide sur le bureau MATLAB

Mathématique décrit comment utiliser les capacités mathématiques et statistique de MATLAB

Programmation et types de données Décrit la manière de créer dans scripts et des fonctions en utilisant le langage Matlab

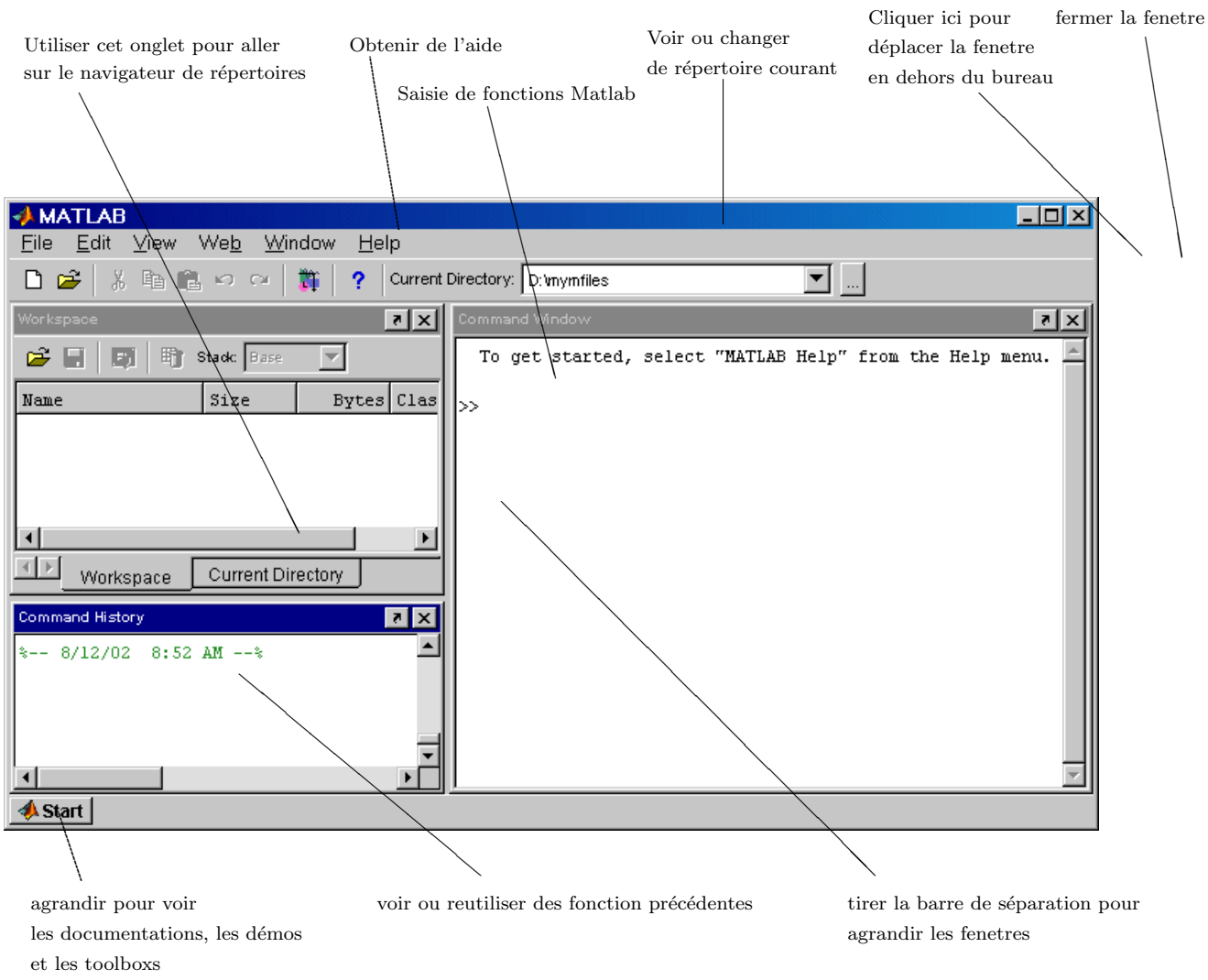
Graphisme Décrit comment tracer des données

Visualisation Introduit comment on utilise les vues, les lumières, la transparences pour réaliser des effets graphiques complets

GUI Création d'interfaces IHM

Interfaces externes/API interface avec des programmes C et Fortran, classes Java, COM objets, fichiers de données, liaison séries

En plus de la documentation ci-dessus, la documentation MATLAB inclut des références aux fonctions, à un navigateur pour la manipulation des propriétés graphiques et des exemples.



Il est donc possible de changer le look du bureau en ouvrant, fermant, déplaçant et réduisant/augmentant la taille des fenêtres. Utiliser le menu "View" pour ouvrir ou fermer des outils. Il est aussi possible de bouger des outils en dehors du bureau ou les ramener (propriété de docking). Tous les outils ont des propriétés communes tels que les menus contextuels et les raccourcis claviers.

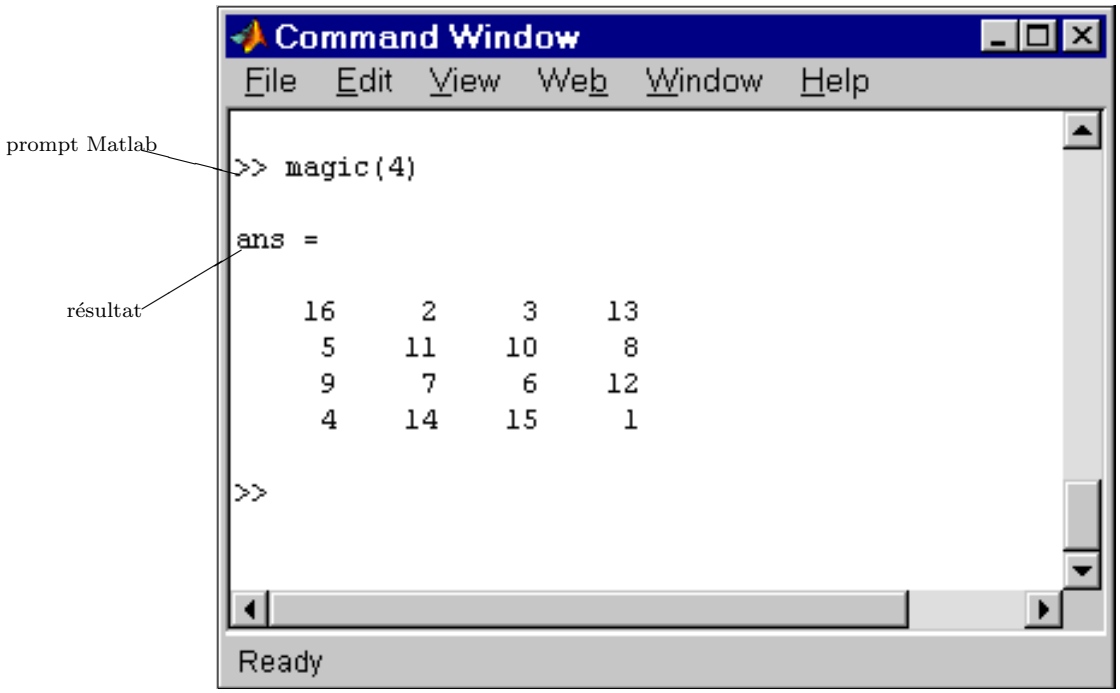
Il est possible de changer quelques caractéristiques pour les outils en sélectionnant "Preferences" dans le menu "File". Par exemple, on peut changer la caractéristique "Font" pour le texte de la fenêtre

de commande ("Command Window").

0.2.4 Desktop Tools

Fenêtre de commande

Il faut utiliser la fenêtre de commande pour entre des variables et lancer des fonctions et des M-fichiers.



Il est possible d'exécuter des programmes externes à partir de la fenêtre de commandes. Il suffit d'utiliser comme premier caractère le point d'exclamation `!`. Ceci est utile pour lancer des utilitaires tel que l'éditeur Emacs par exemple.

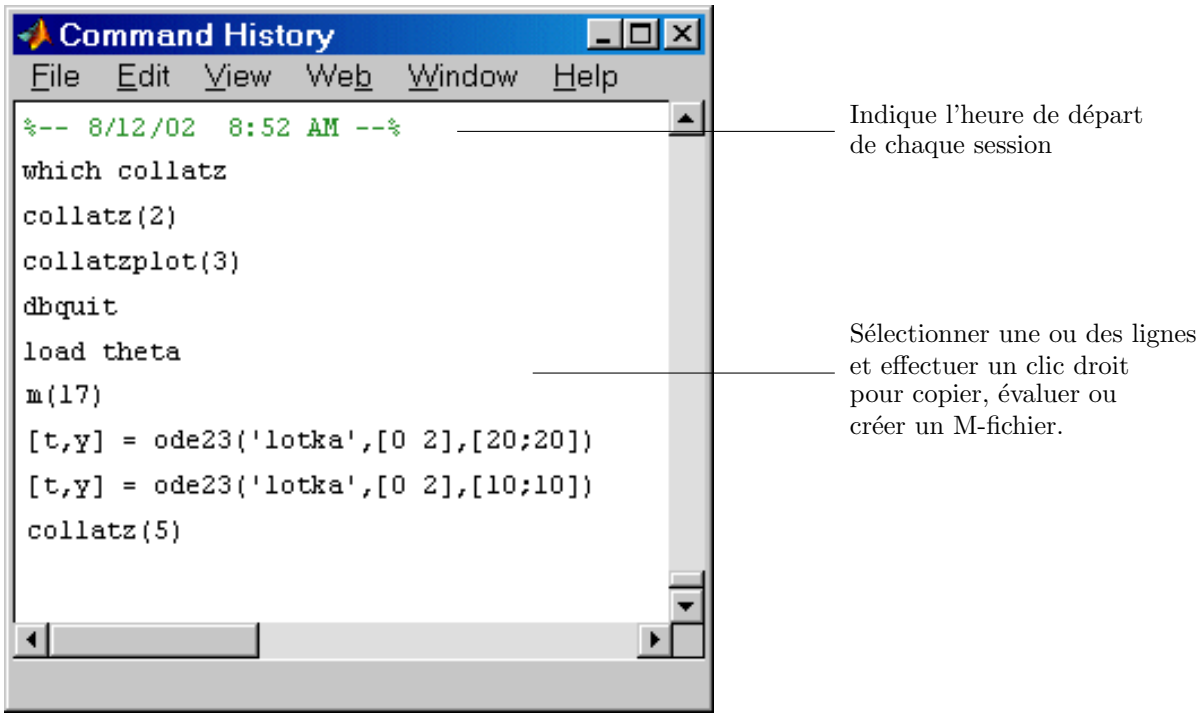
```
!emacs magik.m
```

Une fois le programme terminé, la main est rendue à Matlab.

0.2.5 Historique de commandes

Les lignes tapées dans la fenêtre de commande sont automatiquement sauvegardées dans la fenêtre "Command History". On y voit donc les lignes précédemment tapées et il est possible de les copier ou

d'en sélectionner un groupe afin de l'exécuter.



Pour sauver les entrées/sorties d'un session Matlab dans un fichier, utiliser la fonction `diary`.

0.2.6 Le bouton Start et le Launch Pad

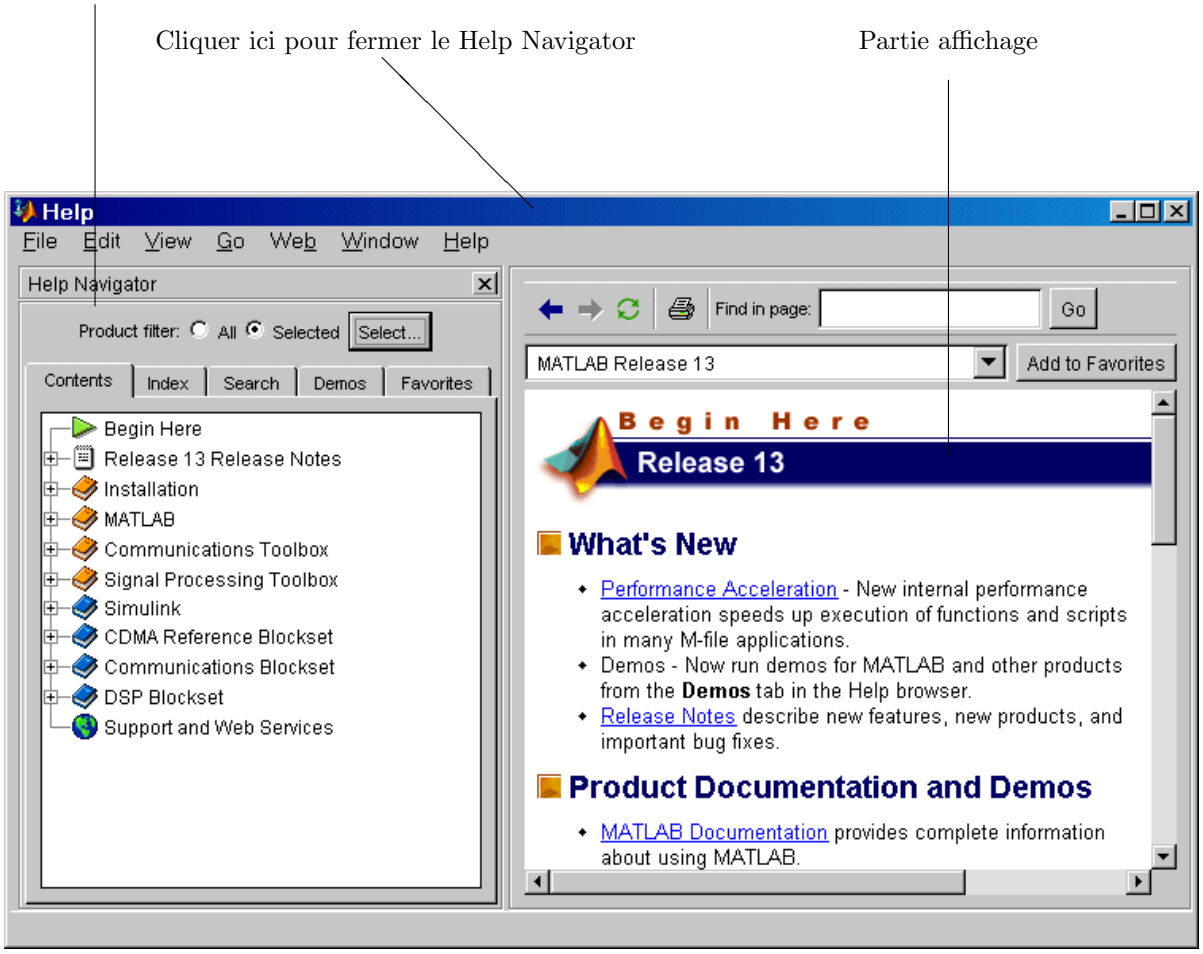
Le bouton Start fournit un accès simplifier aux outils, démonstration et à la documentation. Il suffit juste de cliquer dessus pour voir les options. Le "Launch Pad" donne un accès similaire dans la vue arborescente.

0.2.7 Le navigateur d'aide

On utilise le navigateur d'aide pour chercher et voir la documentation et les démonstrations pour tous les produits Matlab. Le navigateur d'aide est un navigateur internet intégré au bureau Matlab qui affiche des documents HTML.

Pour ouvrir le navigateur d'aide, il suffit de cliquer sur le bouton d'aide dans la barre d'outils, ou de taper `helpbrowser` ou encore `helpdesk` dans la fenêtre de commande.

Les onglets permettent de trouver de la documentation de différentes manières



Le navigateur d'aide consiste en deux parties, le "Help Navigator" (partie gauche), qui sert à la recherche de l'information, et la partie affichage où l'on voit l'information.

"Help Navigator"

On utilise l'Help Navigator pour trouver l'information. Il inclut

Product Filter On position ce filtre uniquement pour voir la documentation sur un produit spécifique.

Contents Voir les titres et la table des matières de la documentation (pour le ou les produits choisis).

Index Trouver des entrées par mots clés dans la documentation.

Demos Voir et exécuter des démonstrations.

Search Recherche d'un mot ou d'une phrase spécifique dans la documentation. Pour obtenir de l'aide sur une fonction particulière, positionner le type de recherche à "Function Name".

Favorites Voir un liste des liens vers des documents désignés précédemment comme favoris.

Partie affichage

Pendant qu'on lit la documentation, on peut

Naviguer vers d'autres pages Utiliser les flèches en haut et en base des pages de documentation pour se déplacer dans le document, ou utiliser les boutons "back" et "forward" dans la barre d'outils pour revenir aux pages précédentes.

Liens Cliquer sur le bouton "Add to Favorites" dans la barre d'outils.

Impression Cliquer sur le bouton "print".

Find in page Taper un terme dans le champ "Find in page" et cliquer sur "Go".

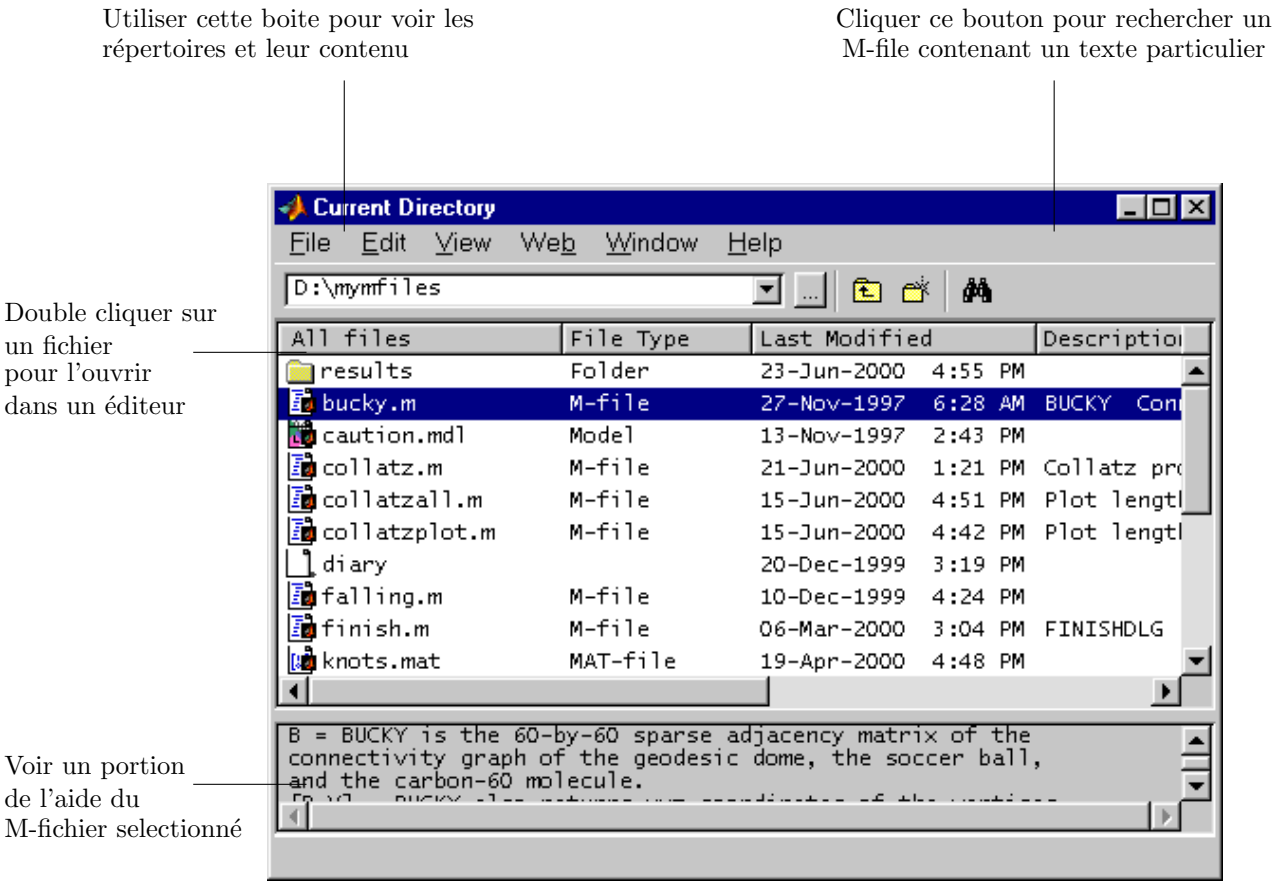
Les autres fonctions disponibles dans la partie affichage sont la copie, l'évaluation d'un sélection et l'affichage de pages Web.

0.2.8 Pour plus d'aide

Il est possible d'obtenir directement de l'aide sur les fonctions en utilisant la commande `doc`. Par exemple, la commande `doc format` va afficher la documentation pour la fonction format dans le navigateur d'aide. Si l'on désire une information réduite, on peut utiliser la fonction `help`. Dans ce cas, l'aide apparat dans la fenêtre de commande.

0.2.9 Le répertoire courant

Les opérations sur les fichiers utilisent le répertoire courant et les chemins d'accès ("search path") comme points de références. Tout fichier que l'on veut exécuter doit impérativement se trouver dans le répertoire courant ou bien dans le "search path". Une manière rapide de voir ou de changer le répertoire courant est d'utiliser le champ "Current Directory" dans la barre d'outils du bureau comme ci-dessous. Pour voire, ouvrir et faire des changements dans les répertoires ou sur des fichiers, utiliser le navigateur "Current Directory". On peut autrement utiliser les fonctions `dir`, `cd` et `delete`



0.2.10 Search Path

Matlab utilise un chemin de recherche (”search path”) pour trouver les M-fichiers et les autres fichiers reliées, qui sont organisés dans des répertoires. Tout fichier que l'on peut pouvoir exécuter en Matlab doit résider dans le répertoire courant ou dans un répertoire qui est dans le ”search path”. Il faut donc ajouter les répertoire qui contiennent les fichiers qu'on veut pouvoir exécuter au ”search path”. Par défaut, les fichiers fournit par Matlab sont inclus dans de ”search path”.

Pour voir ou éditer les répertoires contenus dans le ”search path”, sélectionner ”Set Path” dans le menu ”File” du bureau, et utiliser la boite de dialogue. On peut aussi utiliser les fonctions `path` pour

voir le "search path", `addpath` pour ajouter des répertoires au chemin d'accès et `rmpath` pour ôter des répertoires.

0.2.11 Le navigateur Workspace

Ce navigateur consiste en l'ensemble des variables (nommés *arrays*) utilisées durant une session Matlab et stockées dans la mémoire. On ajoute des variables dans le *workspace* (espace de travail) en utilisant des fonctions, en exécutant des M-fichiers, et en chargeant des *workspaces* préalablement sauvés.

Pour voir le *workspace* et des informations sur chaque variables, utiliser le navigateur *workspace*, ou utiliser les fonctions `who` et `whos`. Pour effacer des variables de l'espace de travail, sélectionner la variable et choisir *Delete* dans le menu d'édition. On peut aussi utiliser la commande `clear`.

Le *workspace* est effacé à la fin d'un session Matlab. Pour sauver son état courant et ainsi pouvoir repartir directement en l'état après un redémarrage de Matlab, il faut utiliser soit "Save Workspace" du menu "File", soit la commande `save`. Ceci sauve toutes les variables dans un fichier binaire appelé un fichier MAT, qui a une extension `.mat`. Pour relire ce type de fichier, utiliser soit "Import Data" du menu "File", soit la fonction `load`

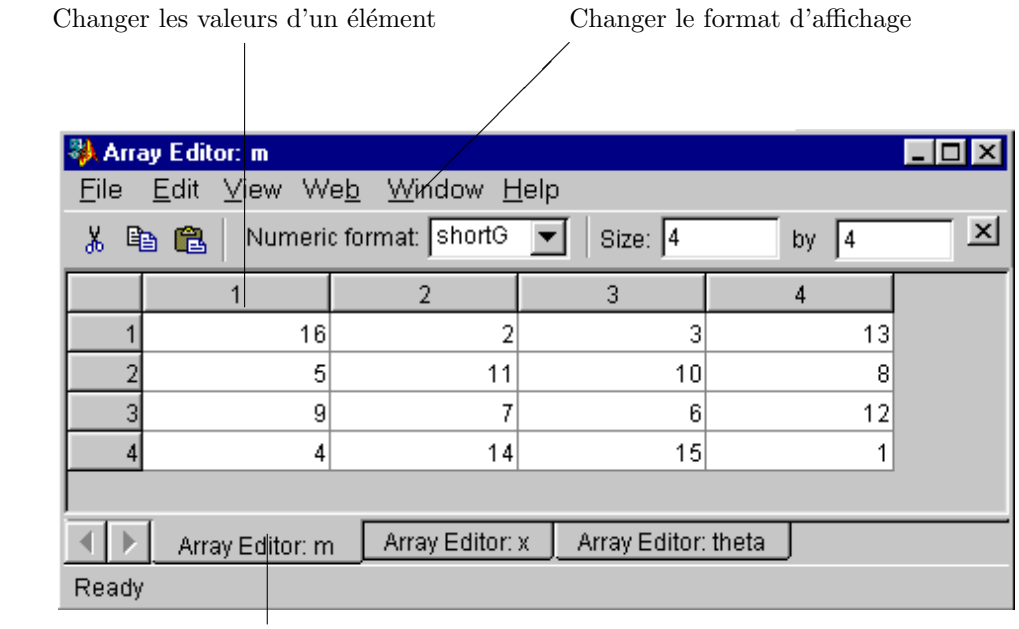
Exemple :

```
>> s1 = sin(pi/4);
>> c1 = cos(pi/4); c2 = cos(pi/2);
>> str = 'hello world';            % C'est une chaine
>> save                            % sauve toutes les variables
                                   % dans un fichier binaire nomm\'e matlab.mat
>> save data                       % sauve toutes les variables
                                   % dans un fichier binaire nomm\'e data.mat
>> save numdata s1, c1             % sauve les variables s1 et c1 dans numdata.mat
>> save strdata str                % sauve la variable str dans strdata.mat
>> save allcos.dat c* -ascii       % sauve c1,c2 dans un format ascii dans allcos.dat
>> load                            % charge toutes les variables du fichier matlab.mat
>> load data s1 c1                 % charge seulement les variables s1 et c1 de data.mat
>> load allcos.dat                 % charge toutes les donn\'ees de allcos.dat dans
                                   % la variable allcos
```

0.2.12 Array Editor

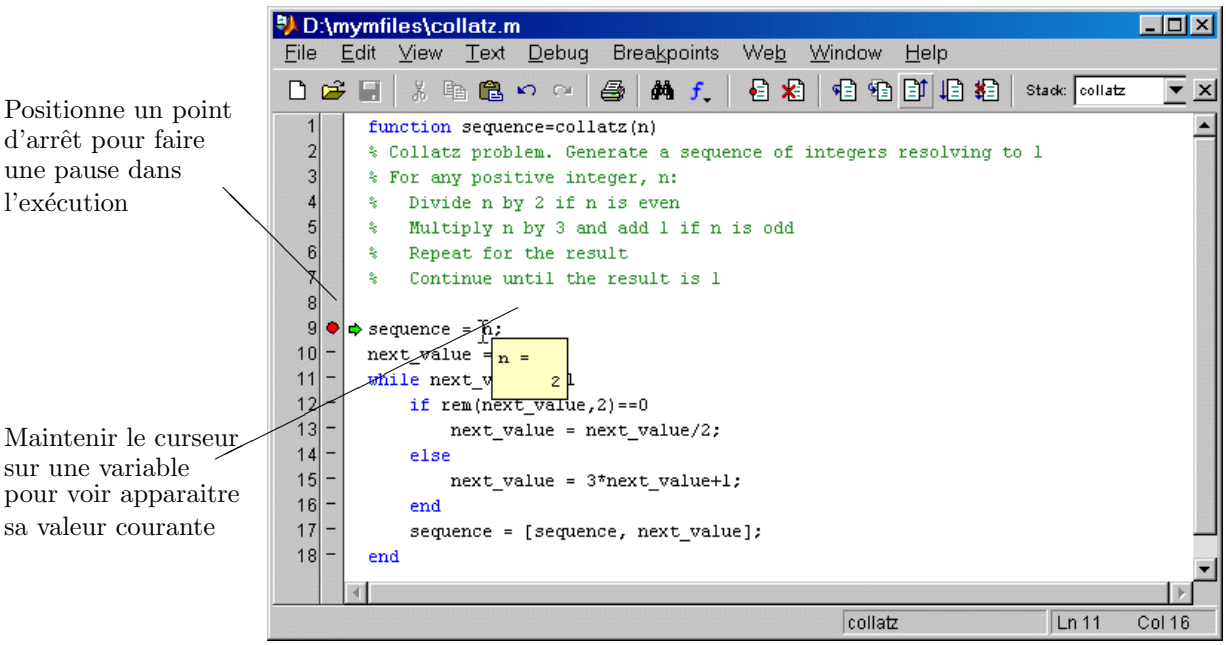
Double-cliquer une variable dans le navigateur *workspace* pour la voir dans l'éditeur de variables "Array Editor". Il est alors possible de l'éditer sous une représentation visuelle d'un tableau uni ou

bidimensionnel, d'une chane ou d'un tableau de cellules de chanes.



0.2.13 Editeur/Debuggeur

Utiliser l'Editeur/Debuggeur pour créer ou débugguer un M-fichier ou des fonctions. L'Editeur/Débuggeur fournit une interface graphique pour des éditions de textes basiques mais aussi un outil de débuggage.



Il n'est nullement nécessaire d'utiliser cet outil pour développer des fonctions Matlab. On peut utiliser aussi des éditeurs de fichiers puissant comme Emacs par exemple. Pour cela, on utilise le menu "preferences" du menu "File" pour spécifier l'éditeur par défaut.

Si l'on désire juste l'affichage d'une fonction ou d'un M-fichier sans nécessité d'édition, on peut simplement utiliser la commande `type` qui affiche le contenu d'un fichier dans la fenêtre de commandes.

0.3 Manipulons des Matrices

Dans Matlab, une matrice est un tableau rectangulaire de nombres. Des significations particulières sont liées aux matrices 1×1, qui sont des scalaires, et aux matrices avec seulement une ligne ou un colonne, qui sont vecteurs. Ainsi, toutes variables, à quelques exceptions près, sont des matrices. Matlab possède d'autres méthodes de représentation de l'information, à la fois numérique et non numérique, mais pour commencer, il est préférable de penser que tout est une matrice. Les opérations en Matlab sont définie pour être les plus naturelles possibles. Avec Matlab, il est possible de manipuler et de travailler

simplement avec des matrices complètes, sans être obligé de coder les produits matrices-matrices ou matrices-vecteurs par exemple comme c'est le cas en Fortran ou en C.

Nous utiliserons dans cette partie comme exemple de matrices les matrices carrées magiques ou matrice de Drer.

0.3.1 Saisie de matrices

La meilleure manière de débuter avec Matlab est d'apprendre comment manipuler les matrices. Il est possible de saisir des matrices de différentes manières

- entrer une liste explicite d'arguments
- charger une matrice depuis un fichier externe
- générer des matrices avec des fonctions Matlab
- créer une matrices avec des M-fichiers

Nous commençons par saisir la matrice de Drer comme une liste de ses éléments. Il faut suivre les règles suivantes

- séparer les éléments d'une même ligne par des espaces ou des virgules
- utiliser le point virgule pour indiquer la fin d'une ligne
- encadrer toute la liste des éléments par des crochets [et].

En suivant les principes précédents, la saisie de la matrice de Drer se fait par

```
A = [16 3 2 13; 5 10 11 8; 9 6 7 12; 4 15 14 1]
```

Matlab affiche alors la matrice

```
A =

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

Une fois que la matrice a été saisie, elle est automatiquement stockée dans l'espace de travail ("workspace") Matlab sous forme de la variable `A`. On peut donc maintenant se référer à cette matrice par `A`. Maintenant, pourquoi cette matrice est magique ?

0.3.2 Somme, transposée et diagonale

Si l'on prend la somme des lignes ou la somme des colonnes, ou bien des deux diagonales principales, on obtient toujours le même nombre. Vérifions cela avec Matlab.

```
sum(A)
```

Matlab répond

```
ans =

    34    34    34    34
```

Quand on ne spécifie par de variable pour le résultat, Matlab utilise la variable par défaut `ans`. On vient de calculer un vecteur ligne qui contient la somme des colonnes de A. Pour faire la même opération sur les lignes, il suffit de transposer la matrice.

```
A'
```

produit

```
ans =

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

et

```
sum(A')'
```

produit un vecteur colonne contenant la somme des lignes

```
ans =

    34
    34
    34
    34
```

La somme des éléments de la diagonale principale est obtenue avec les commandes `sum` et `diag`.

```
>> diag(A)

ans =
```

```
    16
    10
     7
     1
```

et

```
>> sum(diag(A))

ans =

    34
```

L'autre diagonale principale, aussi appelée anti-diagonale, n'est pas très importante mathématiquement et donc Matlab n'a pas de fonctions prédéfinie pour l'obtenir. On va utiliser la fonction `fliplr` qui effectue une symétrie de la matrice de droite à gauche.

```
>> sum(diag(fliplr(A)))

ans =

    34
```

0.3.3 Les indices

L'élément en ligne i et en colonne j de A est désigné par `A(i,j)`. Par exemple, `A(4,2)` est le nombre en quatrièmes ligne et deuxième colonne. Pour la matrice magique précédente, `A(4,2)` est `15`. Il est donc possible de calculer la somme des éléments dans la quatrième colonne par la commande

```
>> A(1,4) + A(2,4) + A(3,4) + A(4,4)

ans =

    34
```

mais cette méthode est moins élégante.

Il est aussi possible de désigner des éléments d'une matrices à l'aide d'un seul indice `A(k)`. Dans le cas d'un vecteur, cette manière de noter n'a rien de choquant. Dans le cas d'une matrice, elle peut paratre plus étrange. Dans ce cas, la matrice est vu comme un long vecteur colonne formé à partir des

autres colonnes de la matrice originale. Ainsi, `A(8)` est une autre manière de référencer la valeur `15` stockée dans `A(4,2)`.

Si l'on tente d'utiliser un élément en dehors de la matrice, on obtient une erreur

```
>> t = A(4,5)
??? Index exceeds matrix dimensions.
```

D'un autre coté, si l'on stocke un élément en dehors de la matrice, sa taille est automatiquement augmentée pour accommoder la nouvelle matrice

```
>> X = A;
   X(4,5) = 17

X =

    16     3     2    13     0
     5    10    11     8     0
     9     6     7    12     0
     4    15    14     1    17
```

0.3.4 L'opérateur :

L'opérateur deux-points `:` est l'un des plus important en Matlab. Il est utiliser sous différentes formes. L'expression `1:10` est un vecteur ligne contenant les entiers entre 1 et 10

```
>> 1:10

ans =

     1     2     3     4     5     6     7     8     9    10
```

Pour obtenir un espacement autre que un, il suffit de spécifier l'incrément. Par exemple,

```
>> 100:-7:50

ans =

   100    93    86    79    72    65    58    51
```

et

```
>> 0:pi/4:pi

ans =

         0    0.7854    1.5708    2.3562    3.1416
```

Les indices peuvent aussi utiliser l'opérateur `:`. Il devient alors très simples de désigner des portions de matrices. Par exemple, `A(1:k,j)` désigne les k premiers éléments de la jème colonne de A. Ainsi, `sum(A(1:4,4))` calcule la somme de la 4ème colonne.

Mais, il y a encore mieux. Les `:` par eux mêmes désignent tous les éléments d'une ligne ou d'un colonne d'une matrice. Le mot clé `end` désigne la dernière ligne ou dernière colonne. Par exemple, `sum(A(:,end))` calcule la somme des éléments dans la dernière colonne de A.

Si les entiers de 1 à 16 sont rangés dans quatre groupes de sommes égales, alors, cette somme doit être

```
>> sum(1:16)/4

ans =

    34
```

0.3.5 La fonction *magic*

Matlab a en fait une fonction pour créer des matrices magiques de taille quelconque.

```
>> B=magic(4)

B =

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

Cette matrice est à peu près la même que celle de Drer. La seule différence est que les deux colonnes centrales ont été permutée. Pour transformer B en A, il suffit donc de permuter ces deux colonnes.

```
>> A = B(:,[1 3 2 4])
```

```
A =

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

Pour chaque ligne de B, on réordonne les éléments dans l'ordre 1,3,2,4.

0.4 Expressions

0.4.1 Les variables

Il n'est pas nécessaire de déclarer les variables ou bien encore leurs dimensions. Quand Matlab rencontre un nouveau nom de variable, il créer automatiquement cette variable et alloue la place mémoire nécessaire à sons stockage. Si la variable existe déjà, Matlab change son contenu et si nécessaire ré alloue une nouvelle place pour le stockage. Par exemple,

```
num_students = 25
```

crée une matrice 1×1 de nom `num_students` et stocke la valeur 25 dans cet élément simple.

Les noms de variables consistent en une lettre, suivie par un nombre quelconque de lettres, de chiffres ou d'underscores. Matlab n'utilise que les 31 premiers caractères pour identifier les variables. Attention, Matlab fait la distinction entre majuscules et minuscules. Ainsi, `A` et `a` ne désigne pas la même variable. Pour voir le contenu d'une variable, il suffit simplement de taper son nom suivi d'un retour chariot.

0.4.2 Les nombres

Matlab utilise par convention le point `.` comme notation décimale. On peut définir aussi les nombres à l'aide de la notation scientifique. Elle utilise la lettre `e` pour spécifier le facteur de puissance de 10. Voici quelques exemples de nombres

```
3           -99           0.0001
9.6397238   1.60210e-20   6.02252e23
1i          -3.14159j     3e5i
```

Tous les nombre sont stockés de manière interne en utilisant le format long défini par la norme flottante IEEE . La précision est donc de 16 chiffres après la virgule et un intervalle de nombre entre 10^{-308} et 10^{308}.

Exercice : choisir deux nombres complexes, par exemple `-3 + 2i` et `5 - 7i`. Ajouter, soustraire, multiplier et diviser ces deux nombres.

0.4.3 Les opérateurs

Les expressions utilisent les opérateurs arithmétiques classiques ainsi que leurs règles usuelles

`+`	addition
`-`	soustraction
`*`	multiplication
`/`	division
`\`	division à gauche (pour les matrices)
`^`	puissance
`'`	transposition et conjugaison complexe
`( )`	spécifie l'ordre d'évaluation

0.4.4 Les fonctions

Matlab fournit un grand nombre de fonctions mathématiques standards, incluant par exemple `abs`, `sqrt`, `sin` et `exp`. La racine carrée ou le logarithme d'un nombre négatif ne donne pas d'erreur ; Le résultat approprié en complexe est renvoyé. Matlab met aussi à disposition des fonctions mathématiques plus avancées incluant les fonctions de Bessel ou gamma. La plupart de ces fonctions acceptent les arguments complexes. Pour une liste des fonctions mathématiques élémentaires, taper

```
help elfun
```

Pour une liste de fonctions mathématiques plus avancées ou sur les matrices, taper

```
help specfun
help elmat
```

Quelques fonctions, comme `sqrt` et `sin` sont des fonctions "built-in". Ceci signifie qu'elles font parties intégrantes du noyau Matlab et qu'elles sont ainsi très efficaces, mais les règles et les détails de calcul ne sont pas accessible en lecture. D'autres fonctions, comme `gamma` et `sinh`, par contre, sont mises en oeuvre à l'aide de M-fichiers. On peut donc voir leur code et même les modifier si l'on veut.

Plusieurs fonctions spéciales donnent les valeurs de constantes usuelles

pi	3.14159265...
i	nombre imaginaire, $\sqrt{-1}$
j	comme i
eps	precision flottante relative, 2^{-52}
realmin	plus petit nombre flottant, 2^{-1022}
realmax	plus grand nombre flottant, $(2-\varepsilon)2^{1023}$
Inf	Infini
NaN	Not-a-number

L'infini est généré en divisant un valeur quelconque par 0. Not-a-numbre est généré en tentant d'évaluer des expressions comme 0/0 ou Inf-Inf.

Les noms de fonctions ne sont pas réservés. Il est possible de les écraser avec une nouvelle variable. Attention, ceci est une source d'erreur. Si l'on utilise `i` comme indice d'une boucle par exemple, il n'aura plus le sens d'un nombre complexe. Pour le ramener à son état initial, il faut utiliser `clear i`.

0.4.5 Exemples d'expressions

Voici quelques exemples supplémentaires d'expressions et leurs résultats.

```
>> rho = (1+sqrt(5))/2

rho =

    1.6180

>> a = abs(3+4i)

a =

     5

>> z = sqrt(besselk(4/3,rho-i))

z =

   0.3730 + 0.3214i

>> huge = exp(log(realmax))
```

```
huge =

  1.7977e+308

>> toobig = pi*huge

toobig =

   Inf
```

0.4.6 Résumé

1×1). Voici un résumé des opérations possibles.

Notation mathématique	Commande Matlab
$a + b$	`a+b`
$a - b$	`a-b`
ab	`a*b`
a/b	`a/b` ou `b\backslash a`
x^b	`x^b`
$\sqrt{x}$	`sqrt(x)` ou `x^0.5`
$\|x\|$	`abs(x)`
π	`pi`
4.10^3	`4e3` ou `4*10^3`
$3 - 4i$	`3-4*i` ou `3-4*j`
e, e^x	`exp(1)` ou `exp(x)`
$\ln x$, $\log x$	`log(x)`, `log10(x)`
$\sin x$, $\arctan x$, ...	`sin(x)`, `atan(x)` ...

0.5 Travaillons avec les matrices

Matlab fournit quatre fonctions pour générer les matrices de bases

`zeros`	matrices de zeros
`ones`	matrices uns
`rand`	éléments aléatoire uniformément distribués
`randn`	éléments aléatoire normalement distribués

Voici quelques exemples

```
>> Z = zeros(2,4)

Z =

     0     0     0     0
     0     0     0     0

>> F = 5*ones(3,3)

F =

     5     5     5
     5     5     5
     5     5     5

>> N = fix(10*rand(1,10))

N =

     7     4     0     8     4     6     7     9     7     1

>> R = randn(4,4)

R =

   -0.4326   -1.1465    0.3273   -0.5883
   -1.6656    1.1909    0.1746    2.1832
    0.1253    1.1892   -0.1867   -0.1364
    0.2877   -0.0376    0.7258    0.1139
```

Il est aussi possible de charger une matrice à partir d'un fichier binaire contenant une matrice préalablement générer par Matlab. Il suffit pour cela d'utiliser la fonction `load`.

On peut aussi évidement utiliser des M-fichiers comme la fonction `magic.m`.

La concaténation est un autre moyen de création de matrices. Elle permet de joindre plusieurs petites matrices pour en former une plus importante. Les crochets [et] sont en fait les opérateurs de

concaténation.

```
>> B = [A A+32; A+48 A+16]

B =

    16     3     2    13    48    35    34    45
     5    10    11     8    37    42    43    40
     9     6     7    12    41    38    39    44
     4    15    14     1    36    47    46    33
    64    51    50    61    32    19    18    29
    53    58    59    56    21    26    27    24
    57    54    55    60    25    22    23    28
    52    63    62    49    20    31    30    17
```

Il est aussi possible d'effacer des lignes et des colonnes très simplement. Commençons par copier la matrice A dans X par la commande `X=A;` Nous notons pour la première fois l'apparition du point virgule. Il sert à éviter l'affichage du résultat par Matlab. Alors, pour effacer la deuxième colonne de X, on utilise

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

X =

    16     2    13
     5    11     8
     9     7    12
     4    14     1
```

Si on efface simplement un élément, le résultat n'est alors plus une matrice. Ainsi, l'expression `X(1,2)=[]` provoque une erreur.

On peut par contre effacer des sections d'une matrice

```
>> B=A;

>> B(1:2,2:3)=0

B =
```

```
    16     0     0    13
     5     0     0     8
     9     6     7    12
     4    15    14     1
```

0.6 Algèbre linéaire

Nous avons déjà vu que la transposition s'obtient à l'aide de l'apostrophe `'`. Ajouter une matrice à sa transposée donne une matrice symétrique

```
>> A+A'

ans =

    32     8    11    17
     8    20    17    23
    11    17    14    26
    17    23    26     2
```

La multiplication s'obtient simplement par le symbole `*`. Elle met en jeu des produit scalaire entre lignes et colonnes. La multiplication d'une transposée avec la matrice elle même donne aussi une matrice symétrique.

```
>> A'*A

ans =

   378   212   206   360
   212   370   368   206
   206   368   370   212
   360   206   212   378
```

Le déterminant de cette matrice est 0.

```
>> det(ans)

ans =
```

```
     0
```

Il en est de même pour A. Comme A est de déterminant nul, elle n'est pas inversible.

```
>> X=inv(A)
Warning: Matrix is close to singular or badly scaled.
         Results may be inaccurate. RCOND = 9.796086e-018.
```

Matlab utilise le calcul flottant pour calculer les inverses des matrices. Ainsi, il est capable d'inverser la matrice mais indique que le résultat risque d'être erroné car le conditionnement réciproque et proche de la précision machine.

Les valeurs propres de cette matrice magique sont intéressantes

```
>> e=eig(A)

e =

   34.0000
    8.0000
    0.0000
   -8.0000
```

Une des valeurs propres est 0 ce qui confirme que la matrice n'est pas inversible. La plus grande valeur propre est 34 : la somme magique. En fait, le vecteur composé que de 1 est un vecteur propre.

```
>> v=ones(4,1)

v =

     1
     1
     1
     1

>> A*v

ans =

    34
```

```
    34
    34
    34
```

Quand on divise la matrice magique par sa somme magique, le résultat est une matrice doublement stochastique dont la somme des lignes et des colonnes est 1.

```
>> P=A/34

P =

    0.4706    0.0882    0.0588    0.3824
    0.1471    0.2941    0.3235    0.2353
    0.2647    0.1765    0.2059    0.3529
    0.1176    0.4412    0.4118    0.0294
```

Quand on prend des puissances P^k de la matrice P, on remarque que tous les éléments de P^k tendent vers $1/4$.

```
>> P^6

ans =

    0.2501    0.2500    0.2499    0.2499
    0.2499    0.2500    0.2501    0.2500
    0.2500    0.2501    0.2500    0.2499
    0.2499    0.2499    0.2500    0.2501
```

Finalement, il est possible de calculer le polynôme caractéristique de A.

```
>> poly(A)

ans =

  1.0e+003 *

    0.0010   -0.0340   -0.0640    2.1760   -0.0000
```

Ceci indique que le polynôme caractéristique $det(A - \lambda I)$ est $\lambda^4 - 34\lambda^3 - 64\lambda^2 + 2176\lambda$.

Lorsque l'on soustrait un scalaire à une matrice, ce scalaire est en fait soustrait de tous les éléments de la matrice. La moyenne des éléments de notre matrice A est 8.5. Ainsi,

```
>> B=A-8.5

B =

    7.5000   -5.5000   -6.5000    4.5000
   -3.5000    1.5000    2.5000   -0.5000
    0.5000   -2.5000   -1.5000    3.5000
   -4.5000    6.5000    5.5000   -7.5000

>> sum(B)

ans =

     0     0     0     0
```

Ainsi, la somme des colonnes est nulle.

0.7 Les vecteurs

L'addition et la soustraction des vecteurs est la même que pour les matrices. Seules changent les significations des produits et puissances. La somme et la différence d'un vecteur signifie la somme (ou la différence) des composantes de ce vecteur. Une opération similaire existe en Matlab pour la multiplication.

`+`	Addition
`-`	Soustraction
`.*`	Multiplication élément par élément
`./`	division élément par élément
`.\`	division à gauche élément par élément
`.^`	power élément par élément
`.'`	Transposition sans conjugaison

On ne peut utiliser le produit `*` sur des vecteurs qu'à la seule condition que l'un soit un vecteur ligne (resp colonne) et l'autre un vecteur colonne (resp ligne).

```
>> u=ones(3,1),v=rand(1,3)

u =
```

```
     1
     1
     1


v =

    0.4103    0.8936    0.0579

>> u*v

ans =

    0.4103    0.8936    0.0579
    0.4103    0.8936    0.0579
    0.4103    0.8936    0.0579

>> v*u

ans =

    1.3618
```

On peut utiliser les vecteurs pour créer des tables par exemples

```
>> n=(0:9)';
>> pows=[n n.^2 2.^n]

pows =

     0     0     1
     1     1     2
     2     4     4
     3     9     8
     4    16    16
     5    25    32
```

```
     6    36    64
     7    49   128
     8    64   256
     9    81   512

>> format short g
>> x=(1:0.1:2)';
>> logs=[x log10(x)]

logs =

           1           0
         1.1    0.041393
         1.2    0.079181
         1.3     0.11394
         1.4     0.14613
         1.5     0.17609
         1.6     0.20412
         1.7     0.23045
         1.8     0.25527
         1.9     0.27875
           2     0.30103
```

Un vecteur peut être converti en matrice à l'aide de l'instruction `reshape`

```
A =
    1    4    7    10
    2    5    8    11
    3    6    9    12

B = reshape(A,2,6)

B =
    1    3    5    7    9   11
    2    4    6    8   10   12
B = reshape(A,2,[])
```

```
B =
    1    3    5    7    9   11
    2    4    6    8   10   12
```

Exercice :

- Créer un vecteur consistant en les nombres pairs compris entre 21 et 47.
- Soit `x = [4 5 9 6]`.
  - Soustraire 3 de chaque élément.
  - ajouter 11 aux éléments d'indices impairs
  - calculer la racine carré de chaque élément
  - élever à la puissance 3 chaque élément.
- créer un vecteur `x` d'éléments
  - 2, 4, 6, 8, ...
  - 9, 7, 5, 3, 1, -1, -3, -5
  - 1, 1/2, 1/3, 1/4, 1/5, ...
  - 0, 1/2, 2/3, 3/4, 4/5, ...
- Créer un vecteur `x` d'éléments $x_n = \frac{(-1)^n}{2n-1}$ pour n = 1, 2, 3, ... Trouver la somme des 100 premiers éléments
- Soit un vecteur `t`. crire les expressions Matlab qui permettent le calcul de
  - $ln(2 + t + t2)$
  - $cos(t)^2 - sin(t)^2$
  - $e^t(1 + cos(3t))$
  - $tan^{-1}(t)$

  Tester les pour `t = 1 : 0.2 : 2`.
- Soit `x = [2 1 3 7 9 4 6]`. Expliquer ce que font les commandes suivantes.

`x(3)`	`x(6:-2:1)`
`x(1:7)`	`x(end-2:-3:2)`
`x(1:end)`	`sum(x)`
`x(1:end-1)`	`mean(x)`
`x(2:2:6)`	`min(x)`

- Soit `x = [1 + 3i, 2 - 2i]` un vecteur complexe. Que font les expressions suivantes.
  - `x'`
  - `x.'`
  - `x*x'`
  - `x*x.'`

0.8 Résumé des manipulation matricielle

Commande	Résultat
`A(i,j)`	A_{ij}
`A(:,j)`	jème colonne de A
`A(i,:)`	ième ligne de A
`A(k:l,m:n)`	sous matrice de A composée des lignes k à l et des colonnes m à n.
`v(i:j)'`	Partie du vecteur v
`n=rank(A)`	n devient le rang de A
`x=det(A)`	x devient le déterminant de A
`x=size(A)`	x devient un vecteur ligne contenant les dimensions de A
`x=trace(A)`	Calcul de la trace de A
`x=norm(v)`	Calcul de la norme euclidienne de v
`C=A+B`	somme de deux matrices
`C=A-B`	différence de deux matrices
`C=A*B`	produit de deux matrices
`C=A.*B`	matrice dont les termes sont les produits termes à termes des matrices A et B
`C=A^k`	puissance de matrices
`C=A.^k`	matrice dont les termes sont les puissances des termes de A
`C=A'`	transposée de A
`C=A./B`	matrice dont les termes sont les divisions termes à termes des matrices A et B
`X=A\backslash B`	trouve la solution du système $AX = B$
`X=B/A`	trouve la solution de $XA = B$
`C=inv(A)`	calcul de l'inverse de A
`C=null(A)`	calcul d'une base orthonormée engendrant le noyau de A
`C=orth(A)`	calcul d'une base orthonormée engendrant l'image de A
`L=eig(A)`	calcul des valeurs propres de A
`[X,D]=eig(A)`	produit une matrice diagonale D formée des valeurs propres de A et d'une matrice dont les colonnes sont les vecteurs propres associés.
`A=eye(n)`	matrice identité de taille n
`A=zeros(m,n)`	matrice rectangulaire formée de zéros
`A=ones(m,n)`	matrice rectangulaire formée de uns
`A=diag(v)`	matrice diagonale dont les éléments diagonaux sont les éléments du vecteur v
`X=trill(A)`	partie triangulaire inférieure de A
`X=triu(A)`	partie triangulaire supérieure de A
`A=rand(m,n)`	matrice rectangulaire aléatoire
`v=max(A)`	vecteur contenant les valeurs maximales de chaque colonne de A
`v=min(A)`	idem avec le minimum
`v=sum(A)`	idem avec la somme

0.9 Contrôles de l'affichage

0.9.1 La fonction *format*

La fonction *format* contrôle le format numérique des valeurs affichées. Cette fonction modifie seulement la manière dont ces valeurs sont affichées, mais pas leur valeur intrinsèque.

```
>> x=[4/3 1.2345e-6]

x =

       1.3333  1.2345e-006

>> format short;x

x =

    1.3333    0.0000

>> format short e;x

x =

  1.3333e+000  1.2345e-006

>> format short g;x

x =

       1.3333  1.2345e-006

>> format long;x

x =

   1.33333333333333   0.00000123450000
```

```
>> format long e;x

x =

    1.333333333333333e+000    1.234500000000000e-006

>> format long g;x

x =

          1.33333333333333               1.2345e-006

>> format bank;x

x =

          1.33          0.00

>> format rat;x

x =

     4/3          1/810045

>> format hex;x

x =

   3ff5555555555555   3eb4b6231abfd271

>> format compact;x; % Supprime les lignes blanches
x =
   3ff5555555555555   3eb4b6231abfd271
```

0.9.2 Suppression de l'affichage des résultats

Il suffit simplement de rajouter un point-virgule à la fin de l'instruction.

0.9.3 Saisie d'une instruction longue

Si une instruction ne tient pas sur une ligne, il suffit de taper trois points `...` suivi d'un retour chariot.

```
s = 1 -1/2 + 1/3 -1/4 + 1/5 - 1/6 + 1/7 ...
      - 1/8 + 1/9 - 1/10 + 1/11 - 1/12;
```

0.10 Programmation avec Matlab

0.10.1 Structures de contrôles

Opérateurs logiques et relationnels

Pour obtenir de l'aide sur les opérateurs relationnels, taper `help relop`. Les opérateurs `<`, `<=`, `>`, `>=`, `==` et `~=` sont utilisés pour comparer. Les opérateurs logiques `&`, `|` et `~` permettent les combinaisons logiques, les négations ou les relations. En plus existent trois fonctions supplémentaires : `xor`, `any` et `all`.

Exemple

Commande	Résultat
`a = (b > c)`	a est 1 si b est plus grand que c. Identique avec `<`, `>` et `<=`
`a = (b == c)`	a est 1 si b est égal à c
`a = (b ~= c)`	a est 1 si b n'est pas égal à c
`a = ~b`	complément logique : a est 1 si b est 0
`a = (b & c)`	ET logique : a est 1 si b est vrai et c est vrai
`a = (b | c)`	OU logique : a est A si b est vrai ou c est vrai

if

La commande `if` évalue une expression logique et exécute une groupe d'instructions quand l'expression est vraie. Les mots clés optionnels `elseif` et `else` donne la possibilité d'exécuter d'autres groupes d'instructions si l'expression est fausse. Il faut rajouter à la fin le mot clé `end` pour indiquer la fin du dernier groupe d'instructions.

Par exemple, la fonction Matlab qui crée une matrice magique utilise trois algorithmes différents suivant les cas : quand n est impair, quand n est pair mais pas divisible par 4 ou quand n est divisible par 4.

```
if rem(n,2) ~= 0
   M = odd_magic(n)
elseif rem(n,4) ~= 0
   M = single_even_magic(n)
else
   M = double_even_magic(n)
end
```

Attention, il est important de comprendre le fonctionnement de `if`. En effet, le test

```
if A == B, ...
```

est légal en Matlab. Si A et B sont des scalaires, ceci est le test d'égalité entre A et B. Par contre, si A et B sont des matrices, `A==B` ne teste pas si A et B sont égales, mais où elles le sont ! Le résultat est une autre matrice de 0 et de 1. En fait, si A et B ne sont pas de même taille, alors `A==B` est une erreur. La façon propre de teste l'égalité de deux matrices est`if isequal(A,B), ....`

Voici quelques fonctions pour comparer les matrices.

```
isequal
isempty
all
any
```

switch et case

La commande `switch` exécute un groupe d'instructions suivant la valeur d'une variable a ou d'une expression. Les mots clés `case` et `otherwise` délimite le groupe. Seulement le premier cas vrai est exécuté. Il doit obligatoirement y avoir un `end` à la fin. La partie logique pour l'algorithme de la matrice magique pourrait être

```
switch (rem(n,4)==0) + (rem(n,2)==0)
   case 0
      M = odd_magic(n)
   case 1
      M = single_even_magic(n)
   case 2
```

```
      M = double_even_magic(n)
   otherwise
      error('This is impossible')
end
```

for

La commande de boucle `for` répète un groupe d'instructions un nombre prédéterminé de fois. Un `end` délimite la fin.

```
for n = 3:32
   r(n) = rank(magic(n));
end
r
```

while

La boucle `while` répète un groupe d'instructions un nombre de fois indéterminé sous le contrôle d'une expression logique. Un `end` termine l'instruction.

Voici un programme complet qui illustre `while`, `if` et `else`. C'est l'algorithme de la bissection pour la recherche de zéros d'une fonctions.

```
a = 0; fa = -Inf;
b = 3; fb = Inf;
while b-a > eps*b
   x = (a+b)/2;
   fx = x^3-2*x-5;
   if sign(fx) == sign(fa)
      a = x; fa = fx;
   else
      b = x; fb = fx;
   end
end
x
```

Le résultat est une racine du polynôme $x^3 - 2x - 5$.

```
x =
   2.09455148154233
```

continue

L'instruction `continue` stoppe l'itération en cours dans une boucle dans laquelle elle apparat et passe à l'itération suivante. Dans des boules imbriqués, elle stoppe l'itération en cours et donne le contrôle à l'itération externe.

L'exemple suivant montre l'utilisation de `continue` dans le décompte du nombre de lignes de codes du fichier `magic.m`, sans compter les lignes blanches et les commentaires.

```
fid = fopen('magic.m','r');
count = 0;
while ~feof(fid)
    line = fgetl(fid);
    if isempty(line) | strncmp(line,'%',1)
        continue
    end
    count = count + 1;
end
disp(sprintf('%d lines',count));
```

break

La commande `break` permet de quitter une boucle avant la fin de celle ci. Dans des boucles imbriquées, `break` ne fait que sortir de la boucle interne.

Voici le même exemple que pour la commande `while`

```
a = 0; fa = -Inf;
b = 3; fb = Inf;
while b-a > eps*b
   x = (a+b)/2;
   fx = x^3-2*x-5;
   if fx == 0
      break
   elseif sign(fx) == sign(fa)
      a = x; fa = fx;
   else
      b = x; fb = fx;
   end
end
x
```

0.10.2 Les autres structures de données

Les tableaux multidimensionnels

Les tableaux multidimensionnels sont des tableaux qui ont plus de deux indices. Ils peuvent être créées à l'aide des fonctions `zeros`, `ones`, `rand` et `randn` avec plus de deux arguments. Par exemple,

```
R = randn(3,4,5);
```

crée un tableau $3 \times 4 \times 5$. Un tableau multidimensionnel peut représenter un tableau 3D de données physiques, mais aussi une suite de matrices. Dans la suite, le (i, j)ème élément d'une $k$ème matrice sera noté $A(i, j, k)$.

Les matrices magiques d'ordre 4 peuvent être modifiée en échangeant deux colonnes. La commande `p=perm(1:4)` génére les 24 permutations de `1:4`. La $k$ème permutation est le vecteur ligne `p(k,:)`. Alors,

```
A = magic(4);
M = zeros(4,4,24);
for k = 1:24
   M(:,:,k) = A(:,p(k,:));
end
```

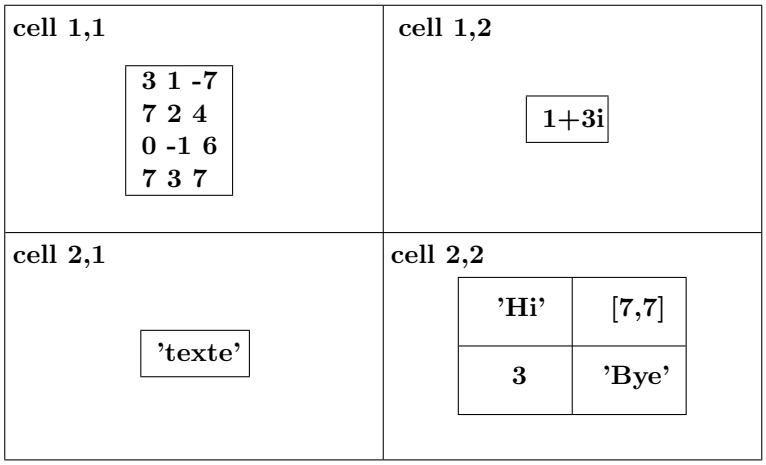
stocke la suite des 24 matrices magiques dans un tableau 3D, M. La taille de M est

```
>> size(M)

ans =
     4 4 24
```

Alors, il est évident que la troisième matrice dans la suite est

```
>> M(:,:,3)

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

Les tableaux de cellules



Les tableaux de cellules sont des tableaux multidimensionnel dont les éléments sont des copies d'autres tableaux. Un tableau de cellules peut être créé avec la commande `cell`. Plus souvent, les tableaux de cellules sont créés tout simplement avec des accolades. Elles sont aussi utilisées pour lire une cellule. Par exemple,

```
>> C = {A sum(A) prod(prod(A))}

C =

    [4x4 double]    [1x4 double]    [2.0923e+013]
```

Cette commande produit un tableau de cellules 1×3. Il contient la matrice magique, le vecteur ligne des sommes des colonnes et le produit de ses éléments. La commande `C{1}` donne la première cellule et `C{3}` la troisième. En fait, on utilise un tableau de cellules pour stocker des objets de types différents.

Un tableau 3D peut être utilisé pour stocker une suite de matrices de même taille. Les tableaux de cellules peuvent eux être utilisés pour stocker des matrices de taille différente. Par exemple,

```
>> M = cell(8,1);
for n = 1:8
M{n} = magic(n);
end
M

M =
```

```
[         1]
[2x2 double]
[3x3 double]
[4x4 double]
[5x5 double]
[6x6 double]
[7x7 double]
[8x8 double]
```

produit une suite de matrices magiques de taille différentes.

Le texte et les caractères

On saisit un texte en Matlab à l'aide d'un apostrophe `'`. Par exemple,

```
s = 'Hello'
```

Le résultat n'est d'aucun type vu jusqu'à présent. C'est un tableau de caractères de taille 1×5. De manière interne, les caractères sont stockés sous forme de nombres, mais de pas de flottants. L'instruction `a=double(s)` convertit le tableau de caractères en une matrice numérique de flottants qui contient les codes Ascii pour chaque caractère.

```
a =
    72 101 108 108 111
```

L'instruction `s=char(a)` produit l'opération réciproque.

La concaténation de chanes se fait encore avec les crochets. L'instruction `h=[s,'world']` joint les chanes horizontalement et produit

```
h =
   Hello world
```

La commande `v=[s;'world']` joint les chanes verticalement

```
v =
   Hello
   world
```

Donc, h est de taille 1×11 et v de taille 2×5.

Les structures

Les structures sont des tableaux multidimensionnels dont les éléments sont accédés avec des noms de champs. Par exemple,

```
S.name = 'Ed Plum';
S.score = 83;
S.grade = 'B+'
```

crée un structure scalaire avec trois champs

```
S =
     name: 'Ed Plum'
    score: 83
    grade: 'B+'
```

Comme tous les objets Matlab, les structures sont des tableaux, et on peut donc insérer des éléments additionnels. Dans ce cas, chaque élément du tableau est une structure avec plusieurs champs. Les champs peuvent ainsi être ajoutés un par un

```
S(2).name = 'Toni Miller';
S(2).score = 91;
S(2).grade = 'A-';
```

ou par structure entière

```
S(3) = struct('name','Jerry Garcia',...
              'score',70,'grade','C')
```

Maintenant, la structure est trop large pour permettre un affichage complet et seul un résumé est affiché.

```
S =
1x3 struct array with fields:
    name
    score
    grade
```

Pour accéder aux éléments d'une structure, on utilise simplement le point.

```
S.score
```

est similaire a

```
S(1).score, S(2).score, S(3).score
```

Seul le dernier résultat se retrouve dans `ans`. On stocke donc les résultats dans un vecteur ou dans un tableau de cellules.

```
[S.score]
{S.score}
```

0.10.3 Les scripts et les fonctions

Scripts

Un script n'est rien d'autre qu'un fichier contenant une suite d'instructions. Lorsque l'on demande à Matlab d'exécuter un script, il se contente de lancer les différentes commandes du fichier. Les scripts peuvent manipuler les données contenu dans l'espace de travail. Bien que les scripts n'aient pas d'arguments de retour, toutes les variables créées pendant leur exécution sont directement accessibles dans le workspace, et peuvent donc être utilisées pour d'autres taches.

Un script est un M-fichier. Son nom est donc composé d'une racine suivi de l'extension `.m`. Si ce fichier s'appelle `magic.m`, alors son exécution se fera simplement par l'appel `magic`.

Fonctions

Les fonctions sont de M-fichiers qui peuvent accepter des arguments d'entrées et des arguments de retour. Le nom de ces M-fichiers et de la fonction doivent être rigoureusement identiques. Les fonctions opèrent sur des variables qui leur sont locales. Elles sont dans un espace de travail propre complètement séparé du workspace principal.

Un bon exemple est fourni par la fonction `rank`. Elle se trouve dans le M-fichier `rank.m` dans le répertoire `toolbox/matlab/matfun`. Il est possible de voir la fonction à l'aide de la commande `type rank`

```
>> type rank

function r = rank(A,tol)
%RANK   Matrix rank.
%   RANK(A) provides an estimate of the number of linearly
%   independent rows or columns of a matrix A.
%   RANK(A,tol) is the number of singular values of A
%   that are larger than tol.
%   RANK(A) uses the default tol = max(size(A)) * norm(A) * eps.
```

```
%   Copyright 1984-2002 The MathWorks, Inc.
%   $Revision: 5.11 $  $Date: 2002/04/08 23:51:52 $

s = svd(A);
if nargin==1
   tol = max(size(A)') * max(s) * eps;
end
r = sum(s > tol);
```

La première ligne d'un M-fichier contenant une fonction commence par le mot clé `function`. Il donne le nom de la fonction et l'ordre des arguments. Dans ce cas, il y a deux arguments d'entrée et un de sortie. Les ligne suivantes contiennent le texte d'aide de la fonction (accessible par `help`). La première ligne de cette aide est l'information fournie par l'instruction `lookfor`. Le reste du fichier contient la partie fonctionnelle. La variable s introduite dans le corps de la fonctions, comme les variables de la première ligne r, A et tol, sont **toutes locales** à la fonction.

Cet exemple montre aussi un des aspects spécifiques à Matlab. Cette fonction utilise un nombre variable d'arguments. La fonction `rank` peut être utilisée de différente manière

```
rank(A)
r = rank(A)
r = rank(A,1.e-6)
```

De nombreuses fonctions fonctionnent de la sorte. Si aucun arguments de sortie n'est spécifié, le résultat est stocké dans `ans`. Si le second argument n'est pas fourni, on en utilise un par défaut. La variable qui contrôle le nombre d'arguments saisis est `nargin`.

Un des défaut des fonctions est qu'elles ne partage leur espace de travail avec le workspace principal de Matlab. Pour permettre des échanges entre ces deux espaces de travail, il faut utiliser des variables globales. On déclare des variables globales à l'aide du mot clé `global`. Par exemple

```
function h = falling(t)
global GRAVITY
h = 1/2*GRAVITY*t.^2;
```

Alors, lors de l'exécution, on tape

```
global GRAVITY
GRAVITY = 32;
y = falling((0:.1:5)');
```

Amélioration des performances

Vectorisation Pour obtenir la plus grande vitesse d'exécution en Matlab, il est important de vectoriser les algorithmes. Il faut essayer de bannir le plus possible les structures de contrôles (tests, boucles ...). Matlab préfère travailler directement avec des opérations vectoriels. Voici un exemple qui crée un table de logarithmes.

```
x = .01;
for k = 1:1001
   y(k) = log10(x);
   x = x + .01;
end
```

La version vectorisée de ce code est

```
x = .01:.01:10;
y = log10(x);
```

La vectorisation n'est pas toujours facile à mettre en oeuvre, mais il faut toujours essayer de l'utiliser.

Pré-allocation S(il n'est pas possible de vectoriser une partie d'un code, pour rendre les boucles `for` plus rapide, il faut pré allouer les vecteurs. Par exemple,

```
r = zeros(32,1);
for n = 1:32
   r(n) = rank(magic(n));
end
```

Sans une pré-allocation, l'interpréteur Matlab agrandit le vecteur `r` d'un élément à chaque itération. Ceci nécessite une ré-allocation à chaque étape et est donc très coteux.

0.11 Graphisme

0.11.1 Tracés de courbes basiques

La fonction `plot` a différentes formes qui dépendent des arguments d'entrée. Si y est un vecteur, `plot(y)` produit une graphe affine par morceaux des éléments de y par rapport à leur indice. Si l'on spécifie deux vecteurs comme arguments, `plot(x,y)` produit une graphe de y où x est l'abscisse.

```
x = 0:pi/100:2*pi;
y = sin(x);
plot(x,y)
```

Il est alors possible de rajouter des titres pour les axes et la figure principale.

```
xlabel('x = 0:2\pi')
ylabel('Sine of x')
title('Graphe de la fonction Sinus','FontSize',12)
```

0.11.2 Plusieurs données sur un même graphe

Plusieurs couples x-y crée plusieurs graphes avec l'appel à un seul `plot`.

```
y2 = sin(x-.25);
y3 = sin(x-.5);
plot(x,y,x,y2,x,y3)
legend('sin(x)','sin(x-.25)','sin(x-.5)')
```

0.11.3 Options

On utilise les options pour changer les couleurs des courbes ou bien les marqueurs. La syntaxe est

```
plot(x,y,'color_style_marker')
```

`color_style_marker` est une chane qui contient de 1 à 4 caractères construite avec les spécifications de couleurs, de style de lignes et de type de marqueurs.

- couleurs : 'c', 'm', 'y', 'r', 'g', 'b', 'w', and 'k'. Ceci correspond à cyan, magenta, jaune, rouge, vert, bleu, blanc, et noir.
- Style de ligne : '-' pour solide, '–' pour des pointillés sous forme de traits, ' :' pour des pointillés, '-.' pour des pointillés alternés. Il suffit de ne pas mettre de style de ligne pour ne pas tracer de lignes.
- Style de marqueurs : `'+'`, `'o'`, `'*'`, et `'x'`, `'s'` pour des carrés, `'d'` pour des losanges, `'^'` pour des triangles, `'v'` pour des triangles vers le bas, `'>'` pour des triangles vers la droite, `'<'` pour des triangles vers la gauche, `'p'` pour des pentagrammes, `'h'` pour des hexagrammes, et rien pour ne pas avoir de marqueurs.

```
x1 = 0:pi/100:2*pi;
x2 = 0:pi/10:2*pi;
plot(x1,sin(x1),'r:',x2,sin(x2),'r+')
```

Pour ajouter des graphiques à un graphe déjà existant, il suffit de taper la commande `hold on`. Pour suspendre son utilisation, taper simplement `hold off`.

Pour effacer un graphique, utiliser la commande `clf`.

```
[x,y,z] = peaks;
contour(x,y,z,20,'k')
hold on
pcolor(x,y,z)
shading interp
hold off
```

Si l'on désire des fenêtres supplémentaires pour effectuer des tracés, il suffit de taper `figure(n)` où `n` est le numéro de la figure.

0.11.4 Surface

Matlab définit une surface par la coordonnées z des points sur une grille dans le plan x-y. Les commandes `mesh` et `surf` affichent des surfaces en 3 dimensions. `mesh` produit un affichage en lignes tandis que `surf` colorie en plus les facettes.

Pour afficher une fonction de deux variables $z = f(x, y)$, il faut générer la liste des points x et y (qu'on appelle maillage) et utiliser x et y pour évaluer la fonction f. La fonction `meshgrid` transforme le domaine spécifié par un simple vecteur ou par deux vecteurs en maillages.

```
[X,Y] = meshgrid(-8:.5:8);
R = sqrt(X.^2 + Y.^2) + eps;
Z = sin(R)./R;
mesh(X,Y,Z,'EdgeColor','black')
```

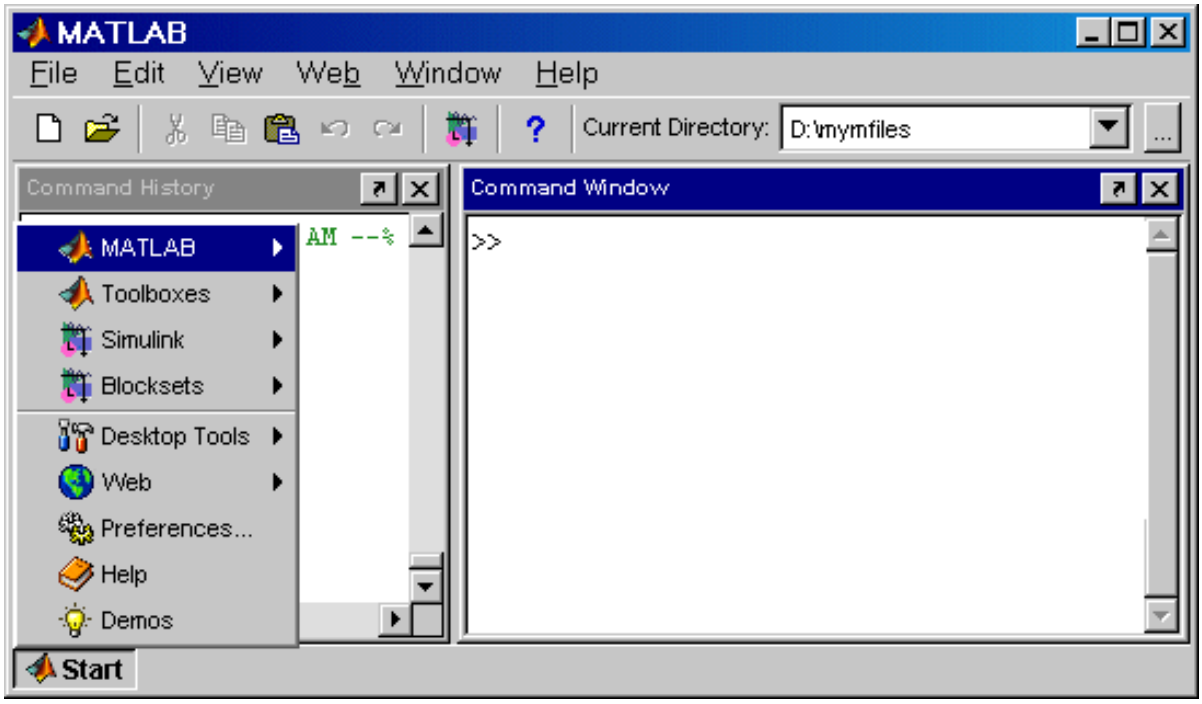
Il est possible de créer une surface transparente en utilisant la commande `hidden off`.

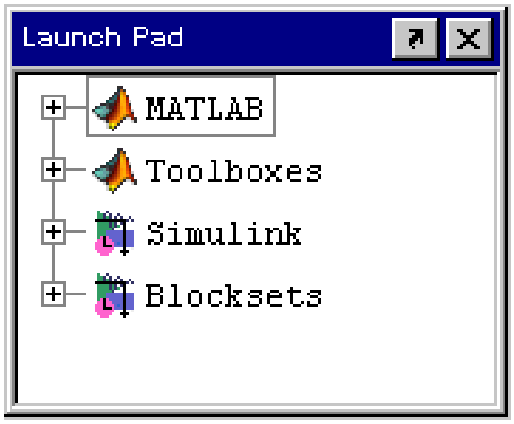
Voici le même exemple avec la commande `surf`.

```
surf(X,Y,Z)
colormap hsv
colorbar
```

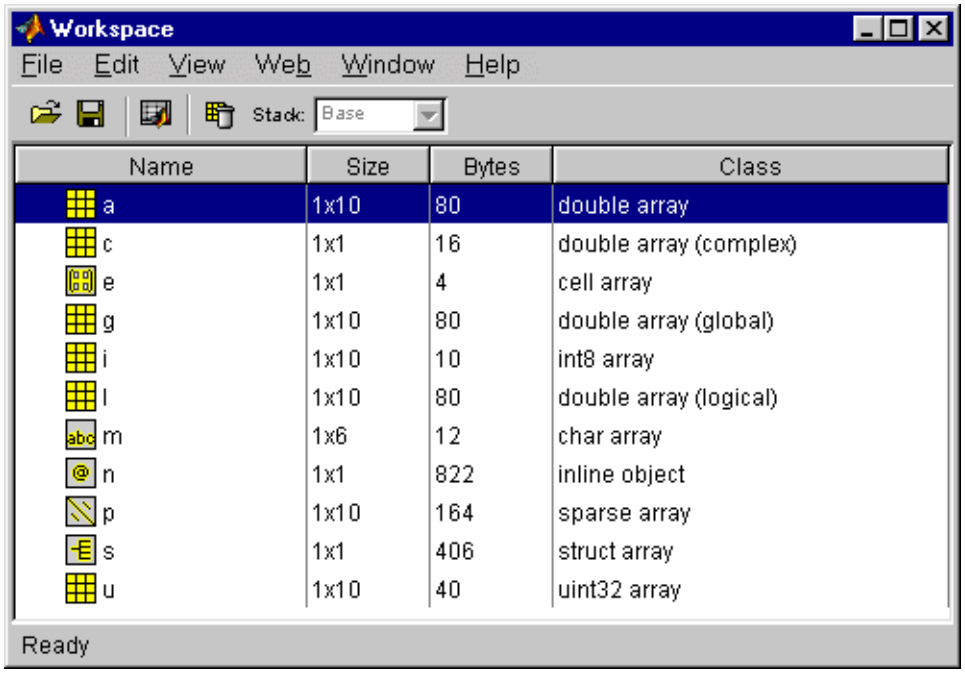
Si l'on désire une figure avec éclairage et fondu, on utilise la commande

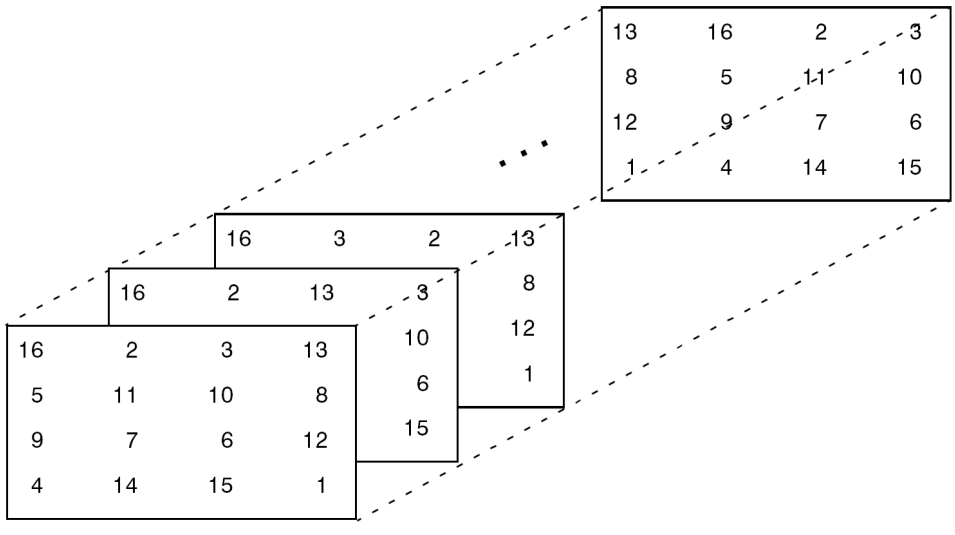
```
surf(X,Y,Z,'FaceColor','red','EdgeColor','none');
camlight left; lighting phong
view(-15,65)
```

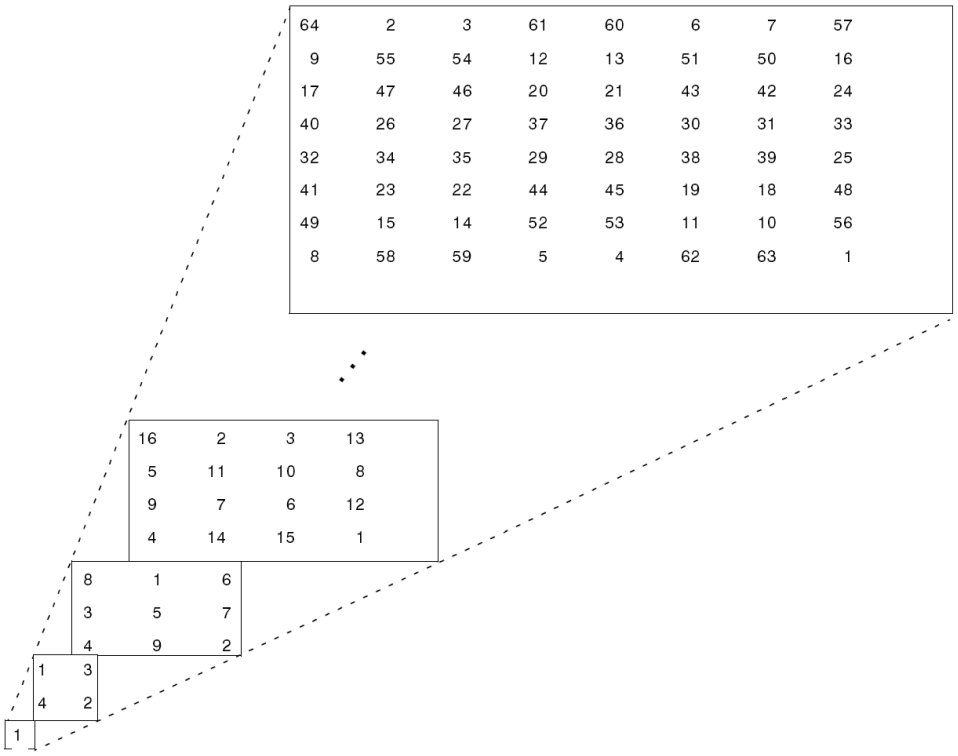

MATLAB
File Edit View Web Window Help
Current Directory: D:\mymfiles
Command History
MATLAB
Toolboxes
Simulink
Blocksets
Desktop Tools
Web
Preferences...
Help
Demos
Start
AM --%
Command Window
>>

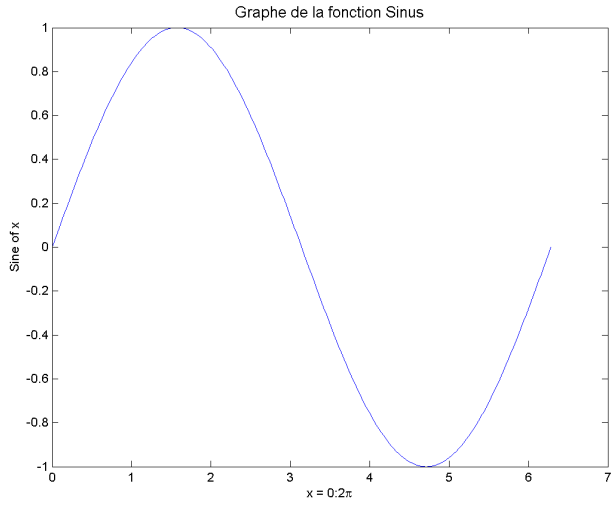

Launch Pad
MATLAB
Toolboxes
Simulink
Blocksets

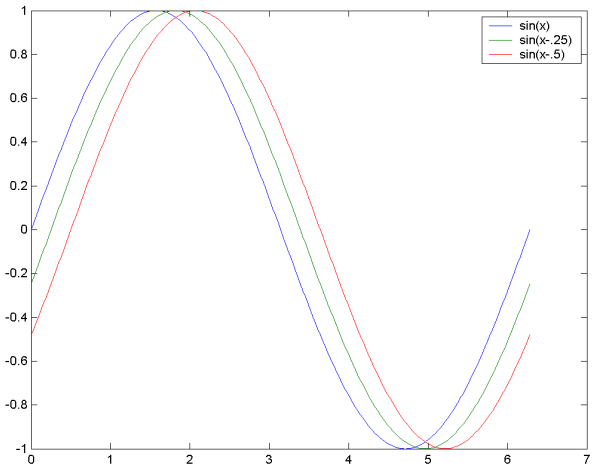

Current Directory: D:\mymfiles

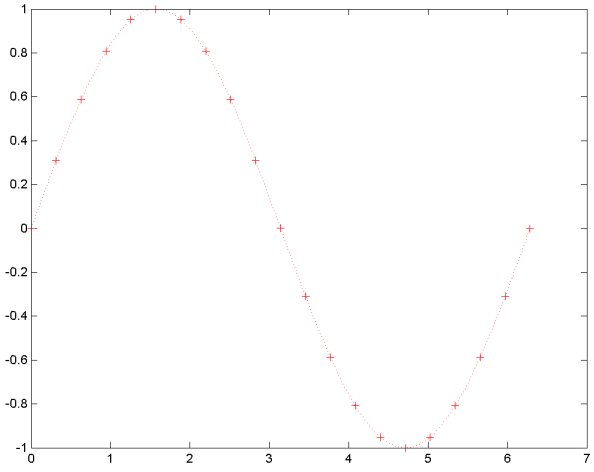

Workspace
File Edit View Web Window Help
Stack: Base
Name Size Bytes Class
a 1x10 80 double array
c 1x1 16 double array (complex)
e 1x1 4 cell array
g 1x10 80 double array (global)
i 1x10 10 int8 array
l 1x10 80 double array (logical)
m 1x6 12 char array
n 1x1 822 inline object
p 1x10 164 sparse array
s 1x1 406 struct array
u 1x10 40 uint32 array
Ready

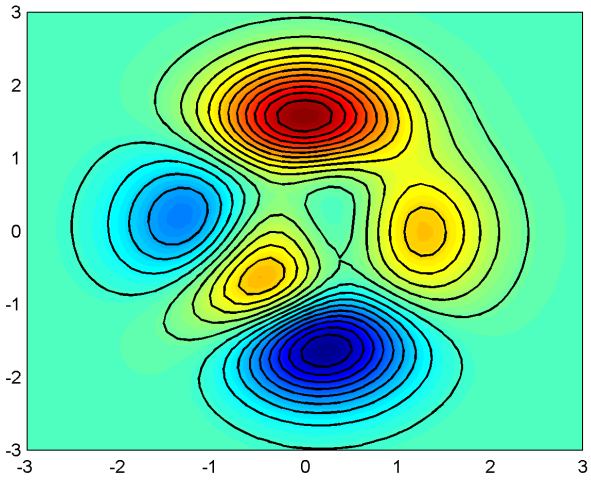

13 16 2 3
8 5 11 10
12 9 7 6
1 4 14 15
16 3 2 13
8
12
1
16 2 13 3
10
6
15
16 2 3 13
5 11 10 8
9 7 6 12
4 14 15 1

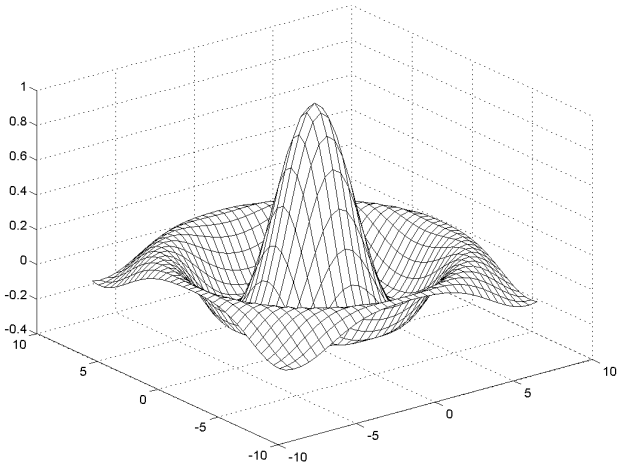

64 2 3 61 60 6 7 57
9 55 54 12 13 51 50 16
17 47 46 20 21 43 42 24
40 26 27 37 36 30 31 33
32 34 35 29 28 38 39 25
41 23 22 44 45 19 18 48
49 15 14 52 53 11 10 56
8 58 59 5 4 62 63 1
16 2 3 13
5 11 10 8
9 7 6 12
4 14 15 1
8 1 6
3 5 7
4 9 2
1 3
4 2
1

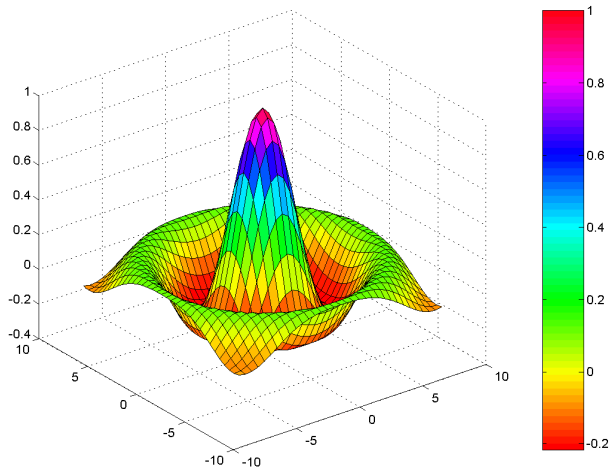

Graphe de la fonction Sinus
Sine of x
x = 0:2π
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
0
1
2
3
4
5
6
7

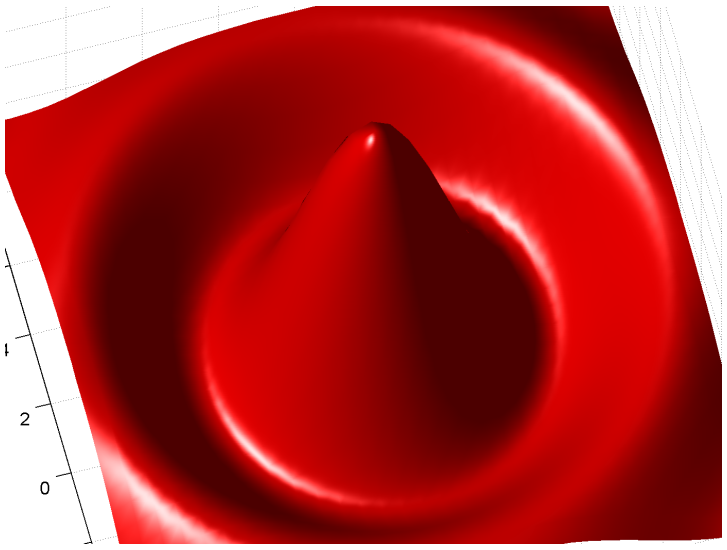

sin(x)
sin(x-.25)
sin(x-.5)
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
0
1
2
3
4
5
6
7


1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
0
1
2
3
4
5
6
7


3
2
1
0
-1
-2
-3
-3
-2
-1
0
1
2
3


1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
10
5
0
-5
-10
-10
-5
0
5
10


1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
10
5
0
-5
-10
-10
-5
0
5
10
1
0.8
0.6
0.4
0.2
0
-0.2


2
0