MATLAB à la carte

Module 3:

Programmation avancée

Eric ANTERRIEU

Observatoire Midi-Pyrénées
Laboratoire d'Astrophysique de Toulouse-Tarbes

CNRS — UMR5572

Eric.Anterrieu@ast.obs-mip.fr

MATLAB à la carte

1. Fonctions
2. Variables
3. Optimisation du code
4. Lien vers d'autres langages

❶ Fonctions

Nom d'une fonction

La fonction `mfilename` retourne le nom de la fonction (sans l'extension .m du fichier) en cours d'exécution.

☺ ***C'est un moyen pratique et simple pour nommer des fichiers log:***

```
diary([mfilename,'.log']);
```


❶ Fonctions

Récursivité

Les fonctions dans MATLAB peuvent être récursives (elles peuvent s'appeler elles-mêmes).

La récursivité est un outil de programmation très pratique mais qui ne conduit pas toujours aux codes les plus performants…

❶ Fonctions

function

MATLAB

```
                    < M A T L A B >
        Copyright 1984-2005 The MathWorks, Inc.
        Version 7.0.4.352 (R14) Service Pack 2

To get started, type
For product informati

» factorial(0)
ans =
     1
» factorial(1)
ans =
     1
» factorial(2)
ans =
     2
» factorial(3)
ans =
     6
» factorial(10)
ans =
   3628800
»
```

factorial.m

```
function m=factorial(n)
%FACTORIAL  Factorial function.
%  FACTORIAL(N) is N!=PROD(1:N)
%
%  See also: GAMMA, PROD.

%  Written by: E. ANTERRIEU

if n == 0
  m = 1;
else
  m = n*factorial(n-1);
end
```


❶ Fonctions

MATLAB reconnait plusieurs types de fonctions:

- les sous-fonctions;
- les fonctions imbriquées;
- les fonctions privées;
- les pointeurs de fonctions;
- les fonctions en ligne;
- les fonctions anonymes.

❶ Fonctions

Sous-fonctions

Un fichier ***fun.m*** peut contenir la définition d'autres fonctions que la fonction principale ***fun***.

Elles peuvent être définies après le corps* de la fonction principale, ce sont alors des sous-fonctions.

* **function** ↔ **end** (facultatif)

fun.m

```
function y=fun(x)

end
function y1=fun1(x1)

end
function y2=fun2(x2)

end
function y3=fun3(x3)

end
```


❶ Fonctions

Sous-fonctions

Une sous-fonction est visible:

- depuis la fonction principale
- depuis une autre sous-fonction

fun.m

```
function y=fun(x)

end
function y1=fun1(x1)

end
function y2=fun2(x2)

end
function y3=fun3(x3)

end
```

❶ Fonctions

Sous-fonctions

Portée des variables:
chaque fonction a son propre espace de travail

<u>donc:</u>

elles n'ont pas accès aux mêmes variables (sauf si **`global`**).

fun.m

```
function y=fun(x)

end

function y1=fun1(x1)

end

function y2=fun2(x2)

end

function y3=fun3(x3)

end
```


❶ Fonctions

Fonctions imbriquées

Un fichier ***`fun.m`*** peut contenir la définition d'autres fonctions que la fonction principale ***`fun`***.

Elles peuvent être définies <u>dans</u> le corps* de la fonction principale, ce sont alors des fonctions imbriquées.

* **`function`** ↔ **`end`** (<u>obligatoire</u>)

fun.m

```
function y=fun(x)

    function y1=fun1(x1)

    end

    function y2=fun2(x2)

        function y3=fun3(x3)

        end

    end

end
```

❶ Fonctions

Fonctions imbriquées

Une fonction imbriquée est visible:

- depuis sa fonction mère (niveau supérieur);
- depuis une fonction sœur (imbriquée au même niveau);
- depuis une fonction fille (imbriquée dans elle-même).

fun.m

```
function y=fun(x)
    function y1=fun1(x1)
    end
    function y2=fun2(x2)
        function y3=fun3(x3)
        end
    end
end
```


❶ Fonctions

Fonctions imbriquées

Portée des variables:
chaque fonction a son propre espace de travail

<u>donc:</u>

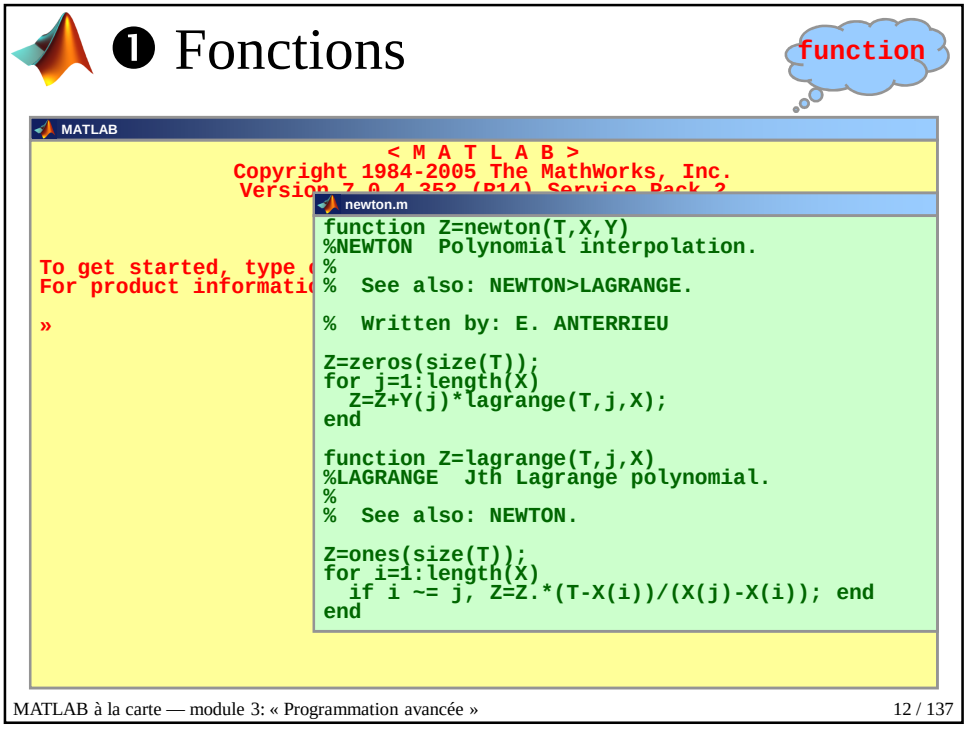
elles n'ont pas accès aux mêmes variables (sauf si **`global`**).

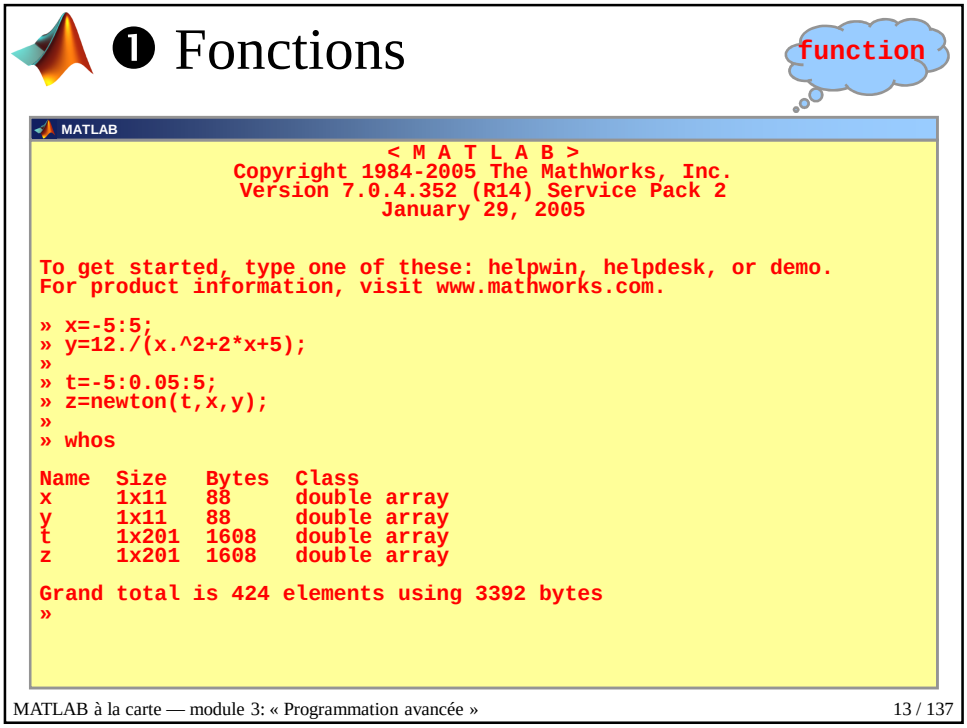
<u>mais:</u>

elles ont accès aux variables de leurs ancêtres.

fun.m

```
function y=fun(x)
    function y1=fun1(x1)
    end
    function y2=fun2(x2)
        function y3=fun3(x3)
        end
    end
end
```


❶ Fonctions
function
MATLAB
< M A T L A B >
Copyright 1984-2005 The MathWorks, Inc.
To get started, type
For product informati
»
newton.m
function Z=newton(T,X,Y)
%NEWTON Polynomial interpolation.
%
% See also: NEWTON>LAGRANGE.
% Written by: E. ANTERRIEU
Z=zeros(size(T));
for j=1:length(X)
Z=Z+Y(j)*lagrange(T,j,X);
end
function Z=lagrange(T,j,X)
%LAGRANGE Jth Lagrange polynomial.
%
% See also: NEWTON.
Z=ones(size(T));
for i=1:length(X)
if i ~= j, Z=Z.*(T-X(i))/(X(j)-X(i)); end
end


❶ Fonctions
function
MATLAB
< M A T L A B >
Copyright 1984-2005 The MathWorks, Inc.
Version 7.0.4.352 (R14) Service Pack 2
January 29, 2005
To get started, type one of these: helpwin, helpdesk, or demo.
For product information, visit www.mathworks.com.
» x=-5:5;
» y=12./(x.^2+2*x+5);
»
» t=-5:0.05:5;
» z=newton(t,x,y);
»
» whos
Name Size Bytes Class
x 1x11 88 double array
y 1x11 88 double array
t 1x201 1608 double array
z 1x201 1608 double array
Grand total is 424 elements using 3392 bytes
»

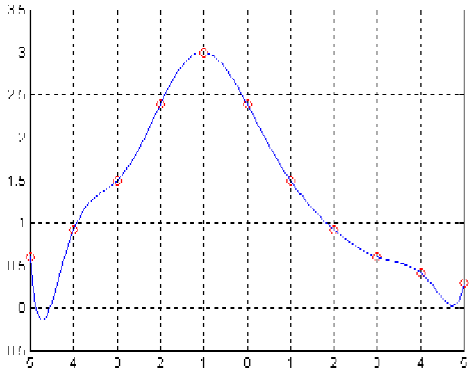
❶ Fonctions

function

MATLAB

```
                    < M A T L A B >
          Copyright 1984-2005 The MathWorks, Inc.
          Version 7.0.4.352 (R14) Service Pack 2
                      January 29, 2005

To get started, type one o
For product information, v

» x=-5:5;
» y=12./(x.^2+2*x+5);
»
» t=-5:0.05:5;
» z=newton(t,x,y);
»
» whos

Name  Size    Bytes  Class
x     1x11    88     double
y     1x11    88     double
t     1x201   1608   double
z     1x201   1608   double

Grand total is 424 element
» figure(1);
» plot(x,y,'ro',t,z,'b-');
»
```

Figure No. 1



❶ Fonctions

Sous-fonctions

Comme MATLAB recherche toujours une sous-fonction avant de rechercher une fonction principale, les sous-fonctions peuvent porter le nom de fonctions principales se trouvant dans d'autres répertoires mais ce nom doit être unique au sein d'un même fichier.

❶ Fonctions

Sous-fonctions

L'aide pour une sous-fonction est afficher à l'aide de la commande **help** en spécifiant le nom de la fonction principale suivi d'un **>** puis du nom de la sous-fonction.

❶ Fonctions

function

MATLAB

```
» help newton
NEWTON  Polynomial interpolation.

  See also NEWTON/LAGR
»
» help newton>lagrange
LAGRANGE  Jth Lagrange

  See also NEWTON.
»
»
```

newton.m

```
function Z=newton(T,X,Y)
%NEWTON  Polynomial interpolation.
%
%  See also: NEWTON/LAGRANGE.

%  Written by: E. ANTERRIEU

Z=zeros(size(T));
for j=1:length(X)
  Z=Z+Y(j)*lagrange(T,j,X);
end

function Z=lagrange(T,j,X)
%LAGRANGE  Jth Lagrange polynomial.
%
%  See also: NEWTON.

Z=ones(size(T));
for i=1:length(X)
  if i ~= j, Z=Z.*(T-X(i))/(X(j)-X(i)); end
end
```

❶ Fonctions

Fonctions privées

Les fonctions privées sont des fonctions qui résident dans des sous-répertoires avec le nom **`private`**.

Les fonctions privées ne sont visibles que des fonctions du répertoire parent.

❶ Fonctions

Fonctions privées

Comme MATLAB recherche toujours une fonction privée avant de rechercher une fonction principale, les fonctions privées peuvent porter le nom de fonctions principales se trouvant dans d'autres répertoires mais ce nom doit être unique au sein d'un même répertoire.

☺ ***C'est un moyen pratique et simple pour créer une version spéciale d'une fonction tout en conservant l'originale dans un autre répertoire.***

❶ Fonctions

Fonctions en ligne

La fonction **`inline`** permet de définir directement une fonction sans avoir à créer un fichier portant le nom de cette fonction:

```
>> h = inline('expression');
>> h = inline('expression','args');
```

😐 L'expression de la fonction doit être assez simple et le nombre d'arguments pas trop élevé!

❶ Fonctions

inline

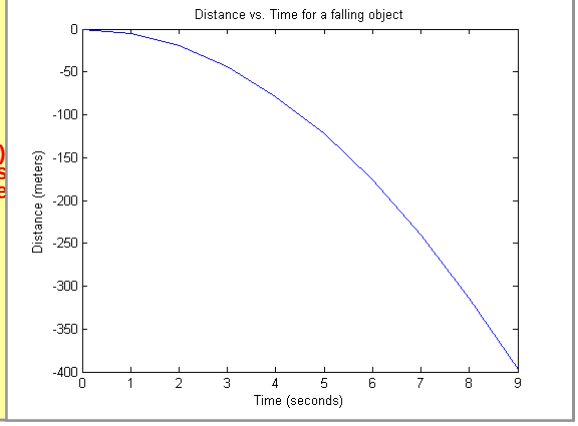
MATLAB

```
» f = inline('sin(pi*x)/(pi*x)','x')
f =  Inline function:
     f(x) = sin(pi*x)/(pi*x)

» argnames(f)
ans = 'x'

» formula(f)
g = sin(pi*x)/(pi*x)

» whos
  Name  Size  Bytes  Class   Attributes

  f     1x1   848    inline
  g     1x16  32     char

» f(0.1)
ans = 0.9836

» f = vectorize(f)
f =  Inline function:
     f(x) = sin(pi.*x)./(pi.*x)

» f(0.1:0.2:1.5)
ans = 0.9836  0.8584  0.6366  0.3679  0.1093 -0.0894 -0.1981 -0.2122

»
```

❶ Fonctions

inline

```
» f = inline('-0.5*g*t.^2','t','g')
f =  Inline function:
     f(t,g) = -0.5*g*t.^2

» argnames(f)
ans = 't'
      'g'

» formula(f)
ans = -0.5*g*t.^2

» t=0:0.01:9;
» figure(1);
» plot(t,f(t,9.81),'b-')
» xlabel('Time (seconds)')
» xlabel('Distance (meters
» title('Distance vs. Time
»
```



❶ Fonctions

Fonctions en ligne

La fonction **`inline`** permet de définir directement une fonction sans avoir à créer un fichier portant le nom de cette fonction:

```
>> h = inline('expression');
>> h = inline('expression','args');
```

☹ Il n'est pas possible d'utiliser une variable locale, ni globale, dans l'expression de la fonction.

❶ Fonctions

MATLAB

```
» f = inline('-0.5*g*t.^2','t')
f =  Inline function:
     f(t) = -0.5*g*t.^2

» argnames(f)
ans = 't'

» formula(f)
ans = -0.5*g*t.^2

» g = 9.81;
» t=0:0.01:9;
» f(t)
??? Error using ==> eval
Undefined function or variable 'g'.

» global g
» f(t)
??? Error using ==> eval
Undefined function or variable 'g'.

»
```


❶ Fonctions

Pointeurs de fonctions

Le caractère @ est un opérateur utilisé pour créer un pointeur (*handle*) vers une fonction contenu dans un fichier:

```
>> h = @fun;
```

ou à partir d'une expression:

```
>> h = @(args) expression;
```

😐 L'expression de la fonction doit être assez simple et le nombre d'arguments pas trop élevé!

❶ Fonctions

MATLAB

```
» f = @sinc
f = @sinc

» whos
  Name   Size   Bytes  Class           Attributes

  f      1x1    16     function_handle

» f(0.1)
ans = 0.9836

» f(0.1:0.2:1.5)
ans = 0.9836  0.8584  0.6366  0.3679  0.1093 -0.0894 -0.1981 -0.2122

»
```

sinc.m

```
function y=sinc(x)
% SINC Cardinale sine of argument.
%
% See also: SIN.

% Written by: E. ANTERRIEU

y = ones(size(x));
k = find(x);
y(k) = sin(pi*x(k))./(pi*x(k));
```


❶ Fonctions

MATLAB

```
» f = @(x) sin(pi*x)/(pi*x)
f = @(x)sin(pi*x)/(pi*x)

» whos
  Name   Size   Bytes  Class           Attributes

  f      1x1    16     function_handle

» f(0.1)
ans = 0.9836

» f = @(x) sin(pi*x)./(pi*x)
f = @(x)sin(pi*x)./(pi*x)

» f(0.1:0.2:1.5)
ans = 0.9836  0.8584  0.6366  0.3679  0.1093 -0.0894 -0.1981 -0.2122

»
```

❶ Fonctions

Pointeurs de fonctions

Le caractère @ est un opérateur utilisé pour créer un pointeur (*handle*) vers une fonction contenu dans un fichier:

```
>> h = @fun;
```

ou à partir d'une expression:

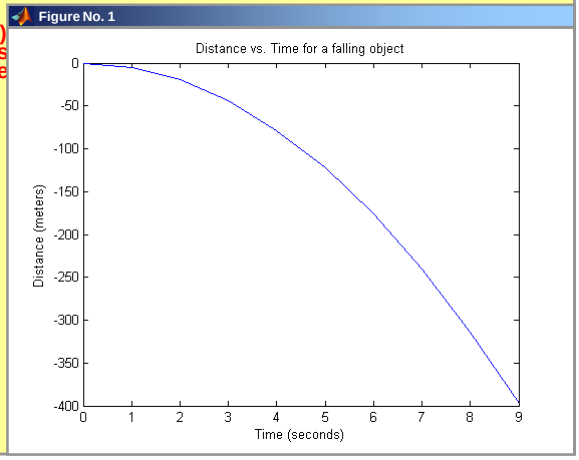
```
>> h = @(args) expression;
```

☺ Il est possible d'utiliser des variables (locales ou globales) dans l'expression de la fonction.

❶ Fonctions

MATLAB

```
» f = @(t,g) -0.5*g*t.^2
f = @(t,g)-0.5*g*t.^2

» t=0:0.01:9;
» figure(1);
» plot(t,f(t,9.81),'b-')
» xlabel('Time (seconds)')
» xlabel('Distance (meters
» title('Distance vs. Time
»
```



❶ Fonctions

MATLAB

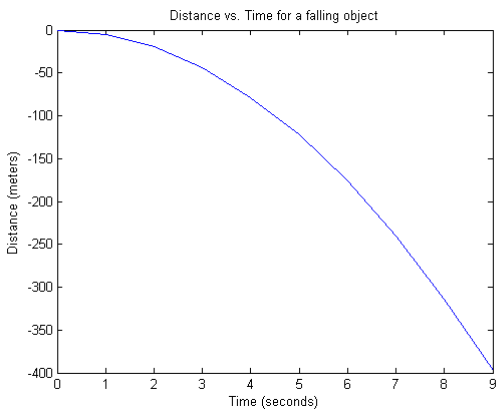
```
» f = @(t) -0.5*g*t.^2
f = @(t)-0.5*g*t.^2

» t=0:0.01:9;
» g=9.81;
» figure(1);
» plot(t,f(t),'b-')
??? Undefined function or variable 'g'.
Error in ==> @(t)-0.5*g*t.^2

»
```


❶ Fonctions

MATLAB

```
» g=9.81;
» f = @(t) -0.5*g*t.^2
f = @(t)-0.5*g*t.^2

» t=0:0.01:9;
» figure(1);
» plot(t,f(t),'b-')
» xlabel('Time (seconds)')
» xlabel('Distance (meters
» title('Distance vs. Time
»
```

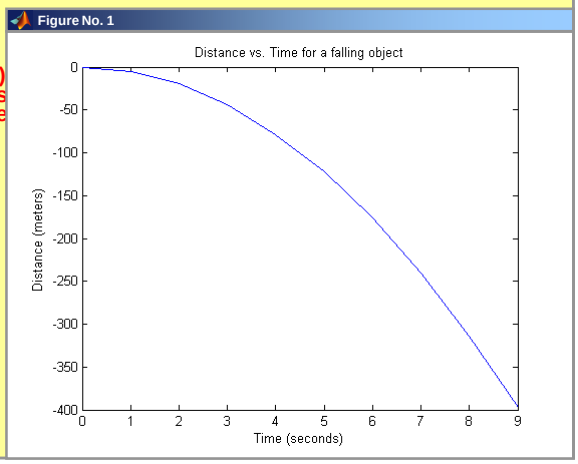
Figure No. 1



❶ Fonctions

MATLAB

```
» g=9.81;
» f = @(t) -0.5*g*t.^2
f = @(t)-0.5*g*t.^2

» clear g
» t=0:0.01:9;
» figure(1);
» plot(t,f(t),'b-')
» xlabel('Time (seconds)')
» xlabel('Distance (meters
» title('Distance vs. Time
»
```

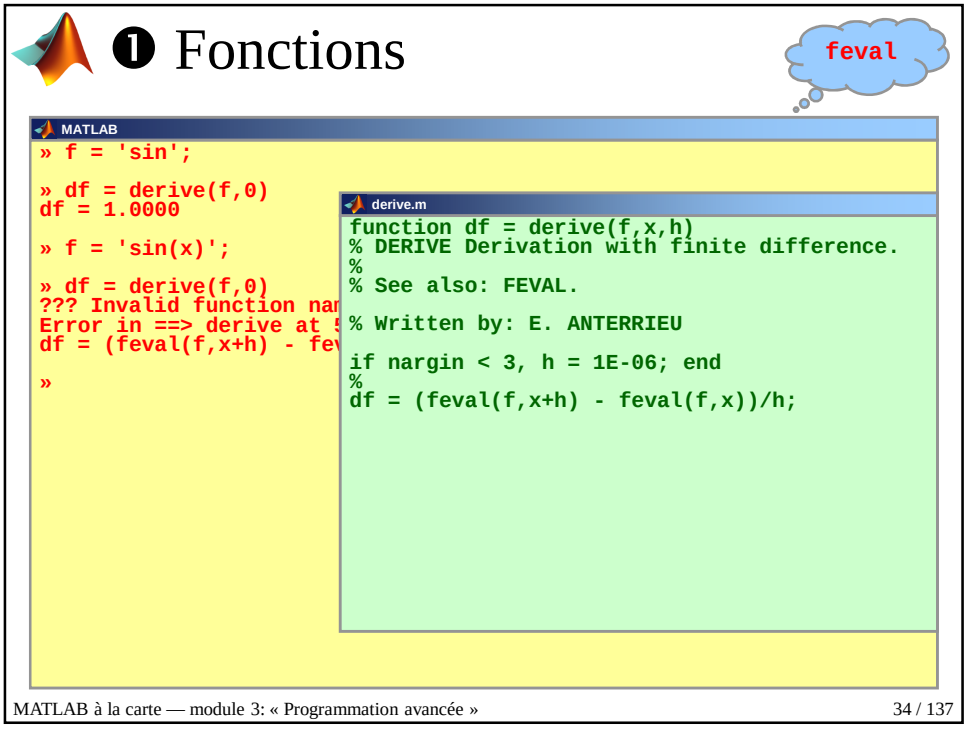


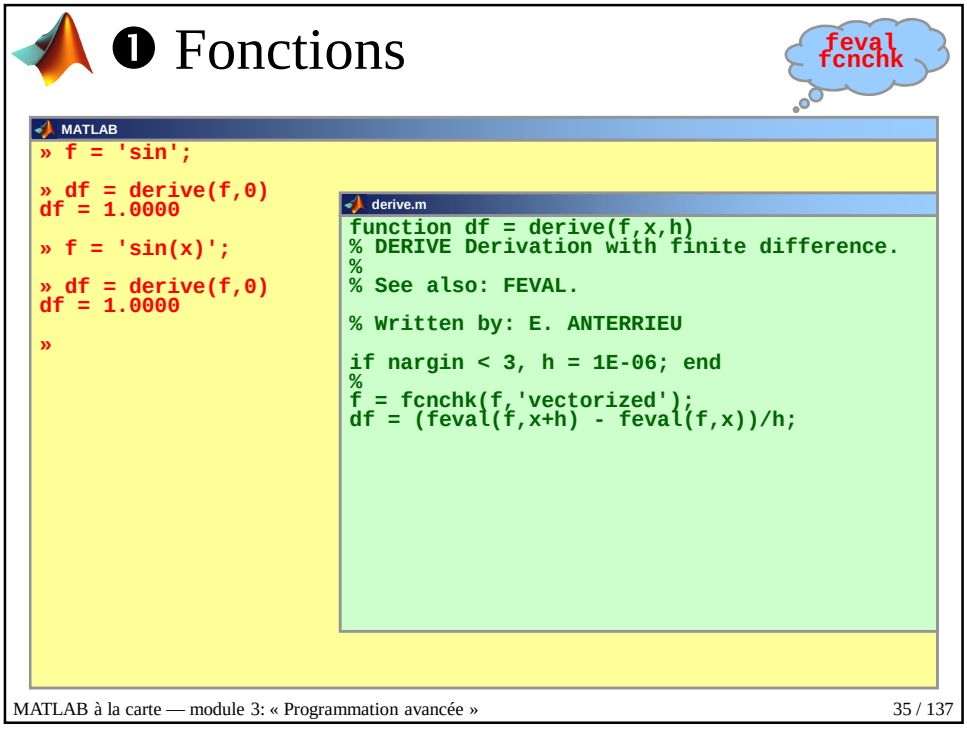
❶ Fonctions

Passer une fonction en argument

Il est possible de passer une fonction en argument à une autre fonction de plusieurs manières:

- à l'aide d'une chaine de caractères;
- à l'aide d'une fonction en ligne;
- à l'aide d'un pointeur de fonction.


❶ Fonctions
feval
MATLAB
» f = 'sin';
» df = derive(f,0)
df = 1.0000
» f = 'sin(x)';
» df = derive(f,0)
??? Invalid function nar
Error in ==> derive at 5
df = (feval(f,x+h) - fev
»
derive.m
function df = derive(f,x,h)
% DERIVE Derivation with finite difference.
%
% See also: FEVAL.
% Written by: E. ANTERRIEU
if nargin < 3, h = 1E-06; end
%
df = (feval(f,x+h) - feval(f,x))/h;


❶ Fonctions
feval
fcnchk
MATLAB
» f = 'sin';
» df = derive(f,0)
df = 1.0000
» f = 'sin(x)';
» df = derive(f,0)
df = 1.0000
»
derive.m
function df = derive(f,x,h)
% DERIVE Derivation with finite difference.
%
% See also: FEVAL.
% Written by: E. ANTERRIEU
if nargin < 3, h = 1E-06; end
%
f = fcnchk(f,'vectorized');
df = (feval(f,x+h) - feval(f,x))/h;

❶ Fonctions

feval

MATLAB

```
» f = inline('sin(x)','x')
f =  Inline function:
     f(x) = sin(x)

» df = derive(f,0)
df = 1.0000

»
```

derive.m

```
function df = derive(f,x,h)
% DERIVE Derivation with finite difference.
%
% See also: FEVAL.

% Written by: E. ANTERRIEU

if nargin < 3, h = 1E-06; end
%
df = (feval(f,x+h) - feval(f,x))/h;
```


❶ Fonctions

~~feval~~

MATLAB

```
» f = inline('sin(x)','x')
f =  Inline function:
     f(x) = sin(x)

» df = derive(f,0)
df = 1.0000

»
```

derive.m

```
function df = derive(f,x,h)
% DERIVE Derivation with finite difference.
%
% See also: FEVAL.

% Written by: E. ANTERRIEU

if nargin < 3, h = 1E-06; end
%
df = (f(x+h) - f(x))/h;
```

☺ *Uniquement MATLAB ≥ R7*

❶ Fonctions

feval

MATLAB

```
» f = @sin
f = @sin

» df = derive(f,0)
df = 1.0000

»
```

derive.m

```
function df = derive(f,x,h)
% DERIVE Derivation with finite difference.
%
% See also: FEVAL.

% Written by: E. ANTERRIEU

if nargin < 3, h = 1E-06; end
%
df = (feval(f,x+h) - feval(f,x))/h;
```


❶ Fonctions

~~feval~~

MATLAB

```
» f = @sin
f = @sin

» df = derive(f,0)
df = 1.0000

»
```

derive.m

```
function df = derive(f,x,h)
% DERIVE Derivation with finite difference.
%
% See also: FEVAL.

% Written by: E. ANTERRIEU

if nargin < 3, h = 1E-06; end
%
df = (f(x+h) - f(x))/h;
```

☺ *Uniquement MATLAB ≥ R7*

❶ Fonctions

MATLAB

```
» f = @(x) sin(x)
f = @(x)sin(x)

» df = derive(f,0)
df = 1.0000

»
```

derive.m

```
function df = derive(f,x,h)
% DERIVE Derivation with finite difference.
%
% See also: FEVAL.

% Written by: E. ANTERRIEU

if nargin < 3, h = 1E-06; end
%
df = (f(x+h) - f(x))/h;
```

☺ *Uniquement MATLAB ≥ R7*

❷ Variables

Nom d'une variable

Depuis une fonction, la fonction

`>> inputname(argno)`

retourne le nom de la variable d'appel correspondant à l'argument numéro **`argno`**.

S'il n'y a pas de variable d'appel pour cet argument (par exemple parce que l'appel correspond à un nombre ou au résultat d'une opération), alors **`inputname`** retourne la chaîne vide **`[]`**.

❷ Variables

Nom d'une variable

La fonction logique

```
>> isvarname('var')
```

retourne **1** si ***var*** est un nom de variable valide, sinon elle retourne **0**.

Les mots clefs du langage MATLAB ne peuvent pas être utilisés comme nom de variable. La fonction

```
>> iskeyword('var')
```

retourne **1** si ***var*** est un mot clef de MATLAB, sinon elle retourne **0**.

❷ Variables

Variables locales

Les variables locales sont les variables définies dans le corps d'une fonction:

- elles sont connues uniquement à l'intérieure de la fonction;
- elles résident dans l'espace de travail de la fonction (différent de l'espace de travail global);
- l'espace mémoire qu'elles occupent est libéré au retour de la fonction (sauf si **persistent**).

❷ Variables

Variables locales

On peut considérer trois familles de variables locales:

• les paramètres d'entrée:

ce sont des variables locales initialisées;

• les paramètres de sortie:

ce sont des variables locales à calculer;

• les variables locales proprement dites.

❷ Variables

Variables globales

Si plusieurs fonction, et possiblement l'espace de travail global, doivent partager une même copie d'une variable sans passer par la transmission en entrée et en sortie des dites fonctions, cette variable doit être déclarée globale à l'aide de l'instruction **`global`** partout où elle doit être connue (elle sera inconnue ailleurs).

☹ La valeur initiale d'une variable globale est **`[]`**, penser à utiliser la fonction **`isempty`**!

❷ Variables

Variables globales

La fonction logique

```
>> isglobal(var)
```

retourne **1** si la variable ***var*** a été déclaré globale, sinon elle retourne **0**.

La commande

```
>> clear global
```

efface toutes les variables globales.

❷ Variables

global

MATLAB

```
» clear all

» constants

» whos

» whos global
  Name    Size   Bytes  Cl

  CST_C   1x1        8  do
  CST_G   1x1        8  do
  CST_H   1x1        8  do
  CST_K   1x1        8  do

» clear global

» whos

» whos global

»
```

constants.m

```
function constants
%CONSTANTS Values of some usefull constants.
%
% See also: GLOBAL.

% Written by: E. ANTERRIEU

global CST_C
CST_C = 3.00E+08; % m/s

global CST_G
CST_G = 9.81;     % m/s²

global CST_H
CST_H = 6.62E-34; % J/s

global CST_K
CST_K = 1.38E-23; % J/K
```

❷ Variables

global

MATLAB

```
» clear all

» t=0:0.1:9;

» Z=Zfalling(t,10,0);
??? Error using ==> Zfa
CST_G not defined.

» constants

» Z=Zfalling(t,10,5);

»

»
```

Zfalling.m

```
function Z = Zfalling(t,Vo,Zo)
% ZFALLING Position vs. time of a
% free-falling object.
%
% See also: GLOBAL.

% Written by: E. ANTERRIEU

global CST_G
if isempty(CST_G)
  error('CST_G not defined.');
end
if nargin == 1, Vo=0; Zo=0; end
if nargin == 2; Zo=0; end
%
Z = -1/2*CST_G*t.^2 + Vo*t + Zo;
```


❷ Variables

global

MATLAB

```
» clear all

» t=0:0.1:9;

» Z=Zfalling(t,10,0);
??? Error using ==> Zfa
CST_G not defined.

» constants

» Z=Zfalling(t,10,5);

» V=Vfalling(t,10);

»
```

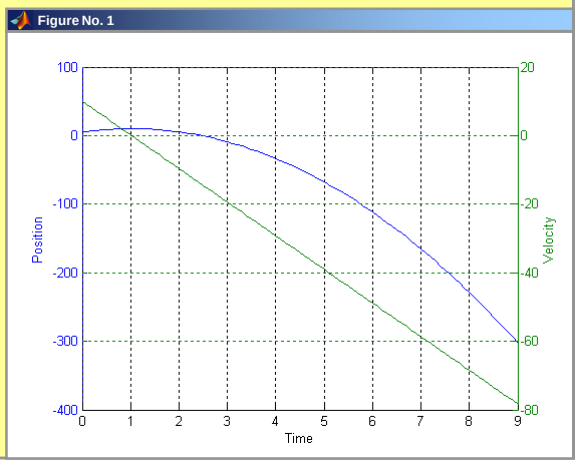
Vfalling.m

```
function V = Vfalling(t,Vo)
% VFALLING Velocity vs. time of a
% free-falling object.
%
% See also: GLOBAL.

% Written by: E. ANTERRIEU

global CST_G
if isempty(CST_G)
  error('CST_G not defined.');
end
if nargin == 1, Vo=0; end
%
V = -CST_G*t + Vo;
```

❷ Variables

global

```
» [ax,h1,h2]=plotyy(t,Z,t,V);
» set(get(ax(1),'Ylabel'),'String','Position');
» set(get(ax(2),'Ylabel'),'String','Velocity');
» xlabel('Time');
» grid('on');
»
```

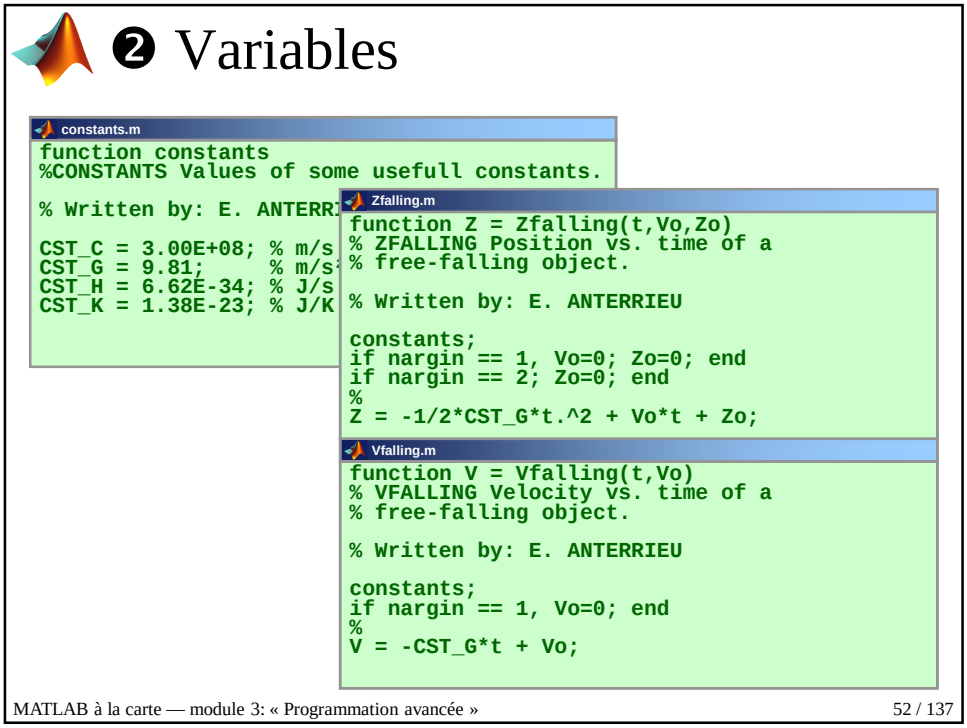


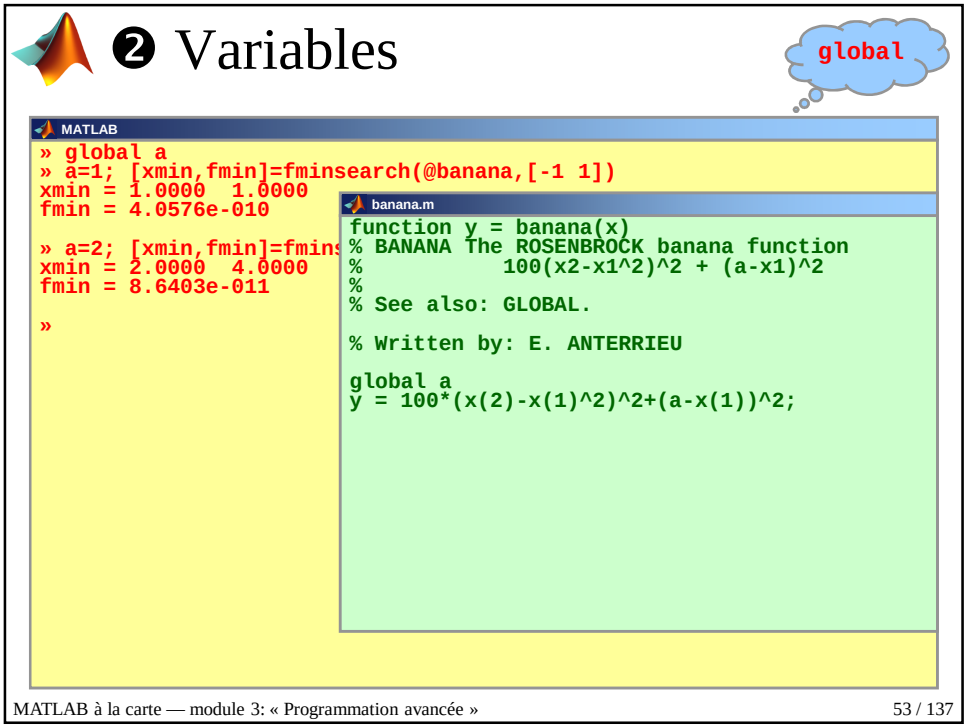
❷ Variables

Variables globales

Le respect des bonnes règles de programmation recommande de ne pas utiliser de variables globales…

…sauf cas extrême, après avoir épuisé toutes les alternatives possibles!


❷ Variables
constants.m
function constants
%CONSTANTS Values of some usefull constants.
% Written by: E. ANTERR
CST_C = 3.00E+08; % m/s
CST_G = 9.81; % m/s
CST_H = 6.62E-34; % J/s
CST_K = 1.38E-23; % J/K
Zfalling.m
function Z = Zfalling(t,Vo,Zo)
% ZFALLING Position vs. time of a
% free-falling object.
% Written by: E. ANTERRIEU
constants;
if nargin == 1, Vo=0; Zo=0; end
if nargin == 2; Zo=0; end
%
Z = -1/2*CST_G*t.^2 + Vo*t + Zo;
Vfalling.m
function V = Vfalling(t,Vo)
% VFALLING Velocity vs. time of a
% free-falling object.
% Written by: E. ANTERRIEU
constants;
if nargin == 1, Vo=0; end
%
V = -CST_G*t + Vo;


❷ Variables
global
MATLAB
» global a
» a=1; [xmin,fmin]=fminsearch(@banana,[-1 1])
xmin = 1.0000 1.0000
fmin = 4.0576e-010
» a=2; [xmin,fmin]=fmins
xmin = 2.0000 4.0000
fmin = 8.6403e-011
»
banana.m
function y = banana(x)
% BANANA The ROSENBROCK banana function
% 100(x2-x1^2)^2 + (a-x1)^2
%
% See also: GLOBAL.
% Written by: E. ANTERRIEU
global a
y = 100*(x(2)-x(1)^2)^2+(a-x(1))^2;

❷ Variables

MATLAB

```
» a=1; [xmin,fmin]=fminsearch(@banana,[-1 1],[],a)
xmin = 1.0000  1.0000
fmin = 4.0576e-010

» a=2; [xmin,fmin]=fmins
xmin = 2.0000  4.0000
fmin = 8.6403e-011

»
```

banana.m

```
function y = banana(x,a)
% BANANA The ROSENBROCK banana function
%           100(x2-x1^2)^2 + (a-x1)^2

% Written by: E. ANTERRIEU

y = 100*(x(2)-x(1)^2)^2+(a-x(1))^2;
```


❷ Variables

Variables persistantes

Si la valeur d'une variable locale doit persister d'un appel à l'autre de la fonction dans laquelle elle se trouve, cette variable doit être déclarée persistante à l'aide de l'instruction **`persistent`**.

☹ La valeur initiale d'une variable persistante est **`[]`**, penser à utiliser la fonction **`isempty`**!

❷ Variables

Variables persistantes

Si une variable a été déclarée **`persistent`** dans une fonction ***`fun`***, sa valeur persiste jusqu'à la commande:

```
>> clear fun
```

qui efface ***`fun.m`*** de l'espace de travail global, ou

```
>> clear all
```

qui efface toutes les variables et toutes les fonctions!

❷ Variables

persistent

MATLAB

```
» clear all

» neval=banana
neval = 0

» a=1; [xmin,fmin]=fmins
xmin = 1.0000  1.0000
fmin = 4.0576e-010
» neval=banana
neval = 187

»
```

banana.m

```
function y = banana(x,a)
% BANANA The ROSENBROCK banana function
%           100(x2-x1^2)^2 + (a-x1)^2

% Written by: E. ANTERRIEU

persistent n;
if isempty(n)
  n = 0;
end

if nargin == 2
  y = 100*(x(2)-x(1)^2)^2+(a-x(1))^2;
  n = n+1;
end
if nargin == 0
  y = n;
end
```

❷ Variables

persistent

MATLAB

```
» clear all

» neval=banana
neval = 0

» a=1; [xmin,fmin]=fmins
xmin = 1.0000  1.0000
fmin = 4.0576e-010
» neval=banana
neval = 187

» a=1; [xmin,fmin]=fmins
xmin = 1.0000  1.0000
fmin = 4.0576e-010
» neval=banana
neval = 374

» clear banana

» a=1; [xmin,fmin]=fmins
xmin = 1.0000  1.0000
fmin = 4.0576e-010
» neval=banana
neval = 187

»
```

banana.m

```
function y = banana(x,a)
% BANANA The ROSENBROCK banana function
%           100(x2-x1^2)^2 + (a-x1)^2

% Written by: E. ANTERRIEU

persistent n;
if isempty(n)
  n = 0;
end

if nargin == 2
  y = 100*(x(2)-x(1)^2)^2+(a-x(1))^2;
  n = n+1;
end
if nargin == 0
  y = n;
end
```


❷ Variables

Variables persistantes

On peut prévenir l'effacement d'une fonction, et donc de ses variables persistantes, lors d'une commande **`clear`** en bloquant cette fonction à l'aide de la fonction **`mlock`**.

La fonction **`munlock`** permet de la débloquer de sorte que les prochains appels à **`clear`** pourront à nouveau l'effacer de l'espace de travail global.

La fonction **`mislocked`** permet de savoir si une fonction est bloquée ou non.

❷ Variables

persistent

MATLAB

```
» clear all
» neval=banana
neval = 0

» a=1; [xmin,fmin]=fmins
xmin = 1.0000  1.0000
fmin = 4.0576e-010
» neval=banana
neval = 187

» clear all
» neval=banana
neval = 187

» mislocked('banana')
ans = 1
» munlock('banana');
» mislocked('banana')
ans = 0

» clear all
» neval=banana
neval = 0

»
```

banana.m

```
function y = banana(x,a)
% BANANA The ROSENBROCK banana function
%           100(x2-x1^2)^2 + (a-x1)^2

% Written by: E. ANTERRIEU

persistent n;
if isempty(n)
  n = 0;
end

if nargin == 2
  y = 100*(x(2)-x(1)^2)^2+(a-x(1))^2;
  n = n+1;
end
if nargin == 0
  y = n;
end

mlock
```


❷ Variables

Nombre d'arguments variable

Un des aspects particulier de MATLAB, par rapport à d'autres langages de programmation, réside dans la possibilité d'utiliser et de concevoir des fonctions à nombre d'arguments variable.

❷ Variables

Nombre d'arguments variable

Dans le corps d'une fonction, les deux commandes **`nargin`** et **`nargout`** retournent les nombres d'arguments en entrée et en sortie utilisés lors d'une utilisation de cette fonction.

☺ Ces nombres peuvent varier d'une utilisation à une autre d'une même fonction.

❷ Variables

Nombre d'arguments variable

En dehors du corps d'une fonction, **`nargin`** et **`nargout`** indiquent les nombres d'arguments en entrée et en sortie déclarés pour une fonction:

```
>> narg=nargin('fun');
>> narg=nargout('fun');
```

`narg` est négatif si la fonction ***`fun`*** est déclarée comme ayant un nombre d'arguments variable.

❷ Variables

MATLAB

```
» nargin('sqrt
ans = 3

» nargout('sqr
ans = 2

»
```

sqrtn.m

```
function [y,iter]=sqrtn(x,tol,maxit)
%SQRTN Square root by Newton's method.
%
% See also: SQRT, NARGIN, NARGOUT.

% Written by: E. ANTERRIEU

if nargin  < 1, error('Too few input arguments.'); end
if nargin == 1, tol=eps; maxit=20; end;
if nargin == 2, maxit=20; end

n=1; y=abs(x); dy=Inf;
while dy > tol
  yo = y;
  y = (y+abs(x)/y)/2;
  dy = abs((y-yo)/y);
  if n == maxit
    error(['Not converged after ' int2str(n) '
iterations.']);
  end
  n = n+1;
end
if x < 0, y = 0+j*y; end

if nargout == 2, iter=n; end
```


❷ Variables

MATLAB

```
» format long

» sqrtn(-2)
ans = 0 + 1.414213562373095i

» sqrtn(2)
ans = 1.414213562373095

» sqrtn(2,0.01)
ans = 1.414215686274510

» sqrtn(2,1E-05,6)
ans = 1.414213562374690

» sqrtn(2,1E-12,6)
??? Error using ==> sqrtn at 23
Not converged after 6 iterations.

» z=sqrtn(2,1E-05,6)
z = 1.414213562374690

» [z,k]=sqrtn(2,1E-05,6)
z = 1.414213562374690
k = 5

»
```

❷ Variables

nargin
nargout

```
MATLAB
» format long

» sqrtn
??? Error using ==> sqrtn
Too few input arguments.

» sqrtn(2,1E-05,6,0)
??? Error using ==> sqrtn
Too many input arguments.

» [z,k,e]=sqrtn(2,1E-05,6)
??? Error using ==> sqrtn
Too many output arguments.

»
```

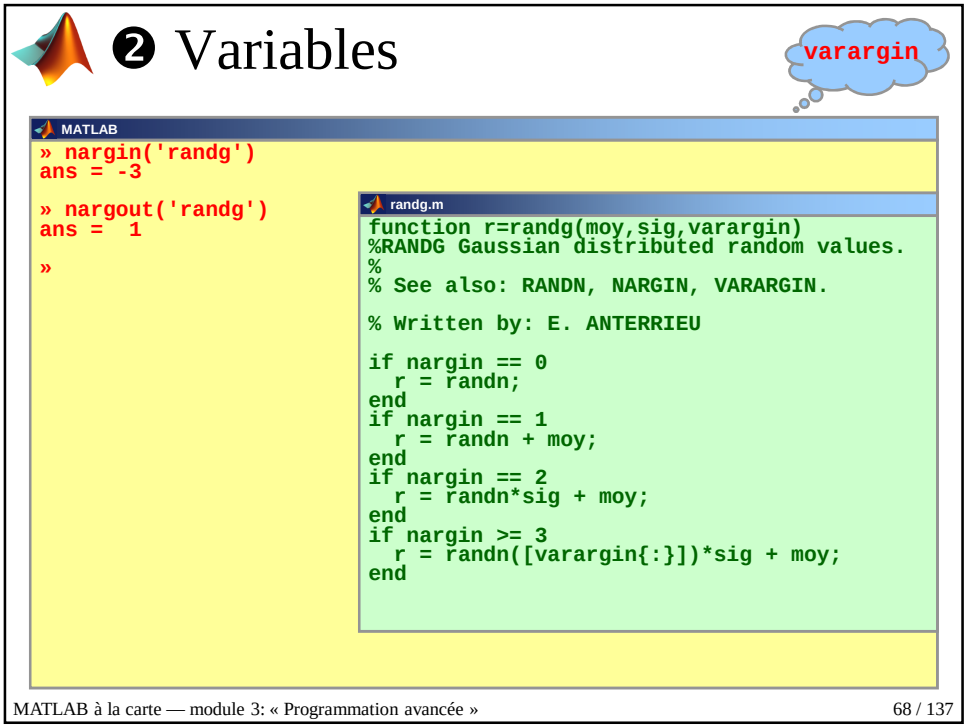

❷ Variables

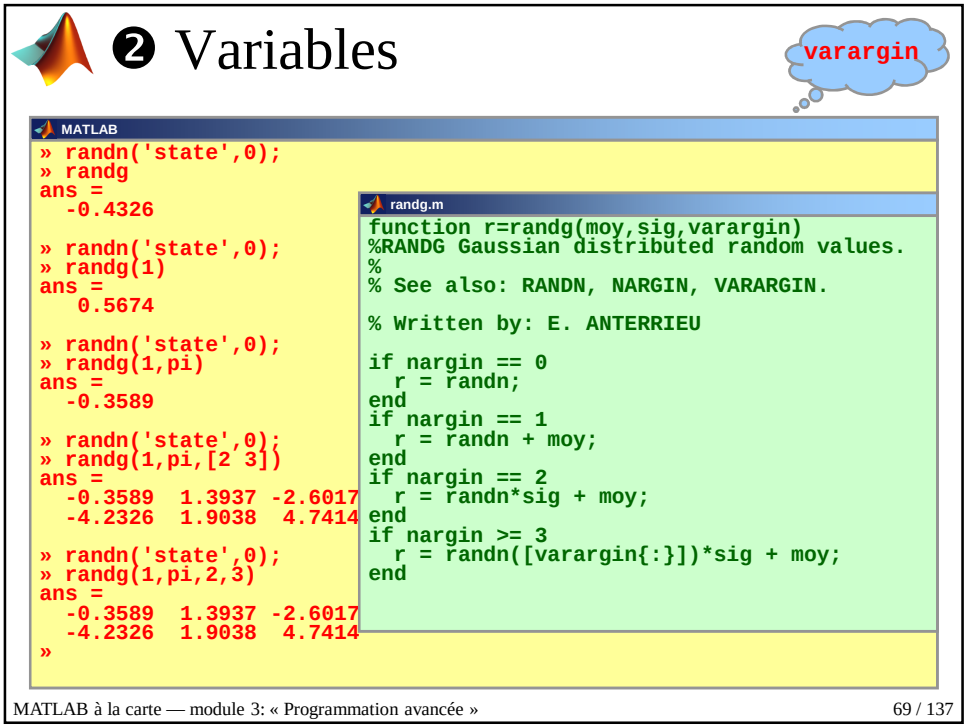
Nombre d'arguments variable

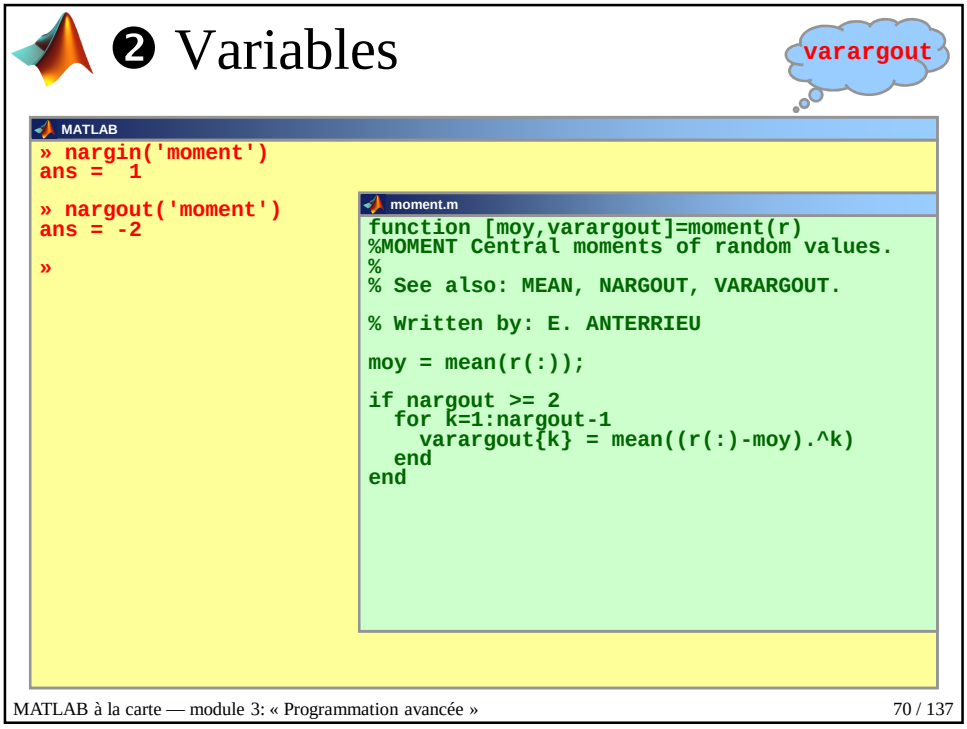
Les instructions **`varargin`** et **`varargout`** sont utilisées, uniquement dans le corps d'une fonction, pour passer et retourner des arguments en nombres variables:

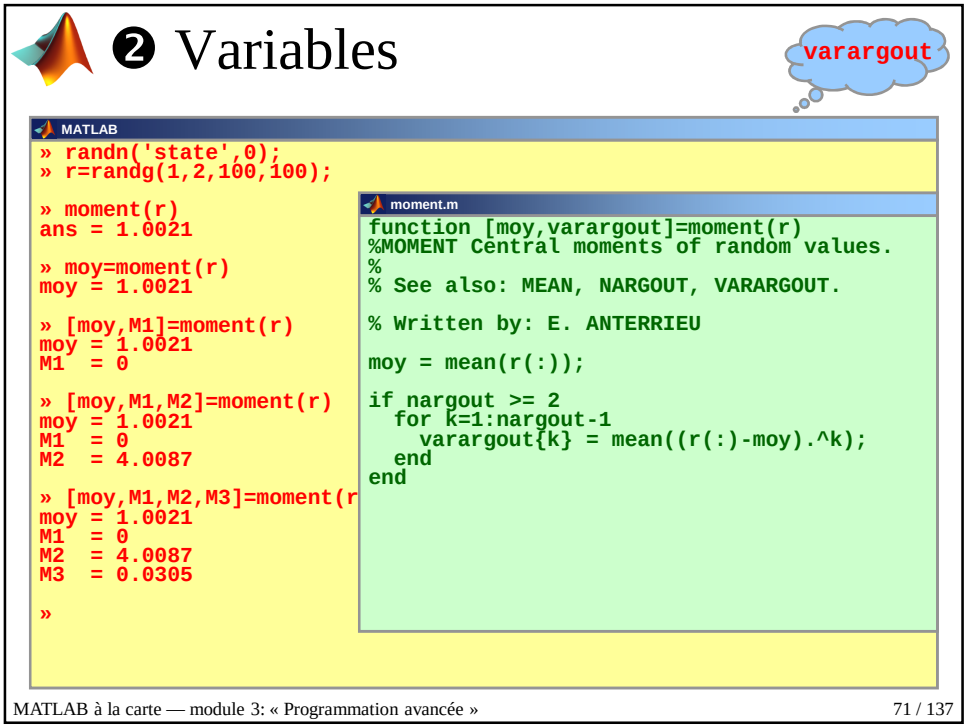
```
function y=fun(x,varargin)
function [y,varargout]=fun(x)
```

Dans la déclaration, chaque instruction doit être écrite en minuscules et doit être la dernière variable de la liste des arguments.


❷ Variables
varargin
MATLAB
» nargin('randg')
ans = -3
» nargout('randg')
ans = 1
»
randg.m
function r=randg(moy,sig,varargin)
%RANDG Gaussian distributed random values.
%
% See also: RANDN, NARGIN, VARARGIN.
% Written by: E. ANTERRIEU
if nargin == 0
r = randn;
end
if nargin == 1
r = randn + moy;
end
if nargin == 2
r = randn*sig + moy;
end
if nargin >= 3
r = randn([varargin{:}])*sig + moy;
end


❷ Variables
varargin
MATLAB
» randn('state',0);
» randg
ans =
-0.4326
» randn('state',0);
» randg(1)
ans =
0.5674
» randn('state',0);
» randg(1,pi)
ans =
-0.3589
» randn('state',0);
» randg(1,pi,[2 3])
ans =
-0.3589 1.3937 -2.6017
-4.2326 1.9038 4.7414
» randn('state',0);
» randg(1,pi,2,3)
ans =
-0.3589 1.3937 -2.6017
-4.2326 1.9038 4.7414
»
randg.m
function r=randg(moy,sig,varargin)
%RANDG Gaussian distributed random values.
%
% See also: RANDN, NARGIN, VARARGIN.
% Written by: E. ANTERRIEU
if nargin == 0
r = randn;
end
if nargin == 1
r = randn + moy;
end
if nargin == 2
r = randn*sig + moy;
end
if nargin >= 3
r = randn([varargin{:}])*sig + moy;
end


❷ Variables
varargout
MATLAB
» nargin('moment')
ans = 1
» nargout('moment')
ans = -2
»
moment.m
function [moy,varargout]=moment(r)
%MOMENT Central moments of random values.
%
% See also: MEAN, NARGOUT, VARARGOUT.
% Written by: E. ANTERRIEU
moy = mean(r(:));
if nargout >= 2
for k=1:nargout-1
varargout{k} = mean((r(:)-moy).^k)
end
end
MATLAB à la carte — module 3: « Programmation avancée »
70 / 137


❷ Variables
varargout
MATLAB
» randn('state',0);
» r=randg(1,2,100,100);
» moment(r)
ans = 1.0021
» moy=moment(r)
moy = 1.0021
» [moy,M1]=moment(r)
moy = 1.0021
M1 = 0
» [moy,M1,M2]=moment(r)
moy = 1.0021
M1 = 0
M2 = 4.0087
» [moy,M1,M2,M3]=moment(r
moy = 1.0021
M1 = 0
M2 = 4.0087
M3 = 0.0305
»
moment.m
function [moy,varargout]=moment(r)
%MOMENT Central moments of random values.
%
% See also: MEAN, NARGOUT, VARARGOUT.
% Written by: E. ANTERRIEU
moy = mean(r(:));
if nargout >= 2
for k=1:nargout-1
varargout{k} = mean((r(:)-moy).^k);
end
end
MATLAB à la carte — module 3: « Programmation avancée »
71 / 137

❸ Optimisation du code

Vectorisation

Dès que cela est possible, il est impératif d'éliminer, sinon de réduire autant que possible, l'utilisation des boucles: c'est la vectorisation.

☺ vectoriser = remplacer les boucles par des opérations vectorielles (ou matricielles) équivalentes.

☹ Facile à dire, pas toujours facile à faire…

😐 Attention au JIT-Accelerator et autres effets de caches dans la mesure des temps d'exécution…

❸ Optimisation du code

MATLAB

```
» clear all
» tic; test; toc
Elapsed time is 0.330069 seconds.

» tic; test; toc
Elapsed time is 0.001234 sec

» tic; test; toc
Elapsed time is 0.001235 sec

» tic; test; toc
Elapsed time is 0.001235 sec

» clear all
» tic; test; toc
Elapsed time is 0.326593 sec

» clear all
» tic; test; toc
Elapsed time is 0.327169 sec

» clear all
» tic; test; toc
Elapsed time is 0.324425 sec

»
```

test.m

```
k=1;
for t=0:0.01:100
   y(k) = sin(pi*t);
   k = k+1;
end
```

❸ Optimisation du code

MATLAB

```
» clear all
» tic; test; toc
Elapsed time is 0.002848 seconds.

» tic; test; toc
Elapsed time is 0.001168 sec

» tic; test; toc
Elapsed time is 0.001136 sec

» tic; test; toc
Elapsed time is 0.001210 sec

» clear all
» tic; test; toc
Elapsed time is 0.002850 sec

» clear all
» tic; test; toc
Elapsed time is 0.002708 sec

» clear all
» tic; test; toc
Elapsed time is 0.002695 sec

»
```

test.m

```
t = 0:0.01:100;
y = sin(pi*t);
```


❸ Optimisation du code

Vectorisation

En présence de boucles imbriquées, vérifier que l'adressage des tableaux est fait correctement (l'indice qui varie le plus vite doit être celui de la boucle la plus interne).

❸ Optimisation du code

MATLAB

```
» Z=rand(128,128,128);
» whos
  Name  Size          Bytes      Class
  Z     128x128x128   16777216   double

» tic; s=test(Z), toc
s = 6.9876e+005
Elapsed time is 0.219548 sec

» tic; s=test(Z), toc
s = 6.9876e+005
Elapsed time is 0.217521 sec

» tic; s=test(Z), toc
s = 6.9876e+005
Elapsed time is 0.217149 sec

» tic; s=test(Z), toc
s = 6.9876e+005
Elapsed time is 0.217740 sec

» tic; s=test(Z), toc
s = 6.9876e+005
Elapsed time is 0.218039 sec

»
```

test.m

```
function s=test(Z)
% written by Eric Anterrieu
s=0;
for k=1:size(Z,1)
for l=1:size(Z,2)
for m=1:size(Z,3)
   s=s+Z(k,l,m)*Z(k,l,m);
end
end
end
```


❸ Optimisation du code

MATLAB

```
» Z=rand(128,128,128);
» whos
  Name  Size          Bytes      Class
  Z     128x128x128   16777216   double

» tic; s=test(Z), toc
s = 6.9847e+005
Elapsed time is 0.041658 sec

» tic; s=test(Z), toc
s = 6.9847e+005
Elapsed time is 0.041293 sec

» tic; s=test(Z), toc
s = 6.9847e+005
Elapsed time is 0.041212 sec

» tic; s=test(Z), toc
s = 6.9847e+005
Elapsed time is 0.041292 sec

» tic; s=test(Z), toc
s = 6.9847e+005
Elapsed time is 0.041508 sec

»
```

test.m

```
function s=test(Z)
% written by Eric Anterrieu
s=0;
for m=1:size(Z,3)
for l=1:size(Z,2)
for k=1:size(Z,1)
   s=s+Z(k,l,m)*Z(k,l,m);
end
end
end
```

❸ Optimisation du code

Vectorisation

Utiliser dès que possible la possibilité d'adresser les éléments des tableaux multi-dimensionnels à l'aide d'un seul indice.

❸ Optimisation du code

MATLAB

```
» Z=rand(128,128,128);
» whos
  Name  Size          Bytes     Class
  Z     128x128x128   16777216  double

» tic; s=test(Z), toc
s = 6.9847e+005
Elapsed time is 0.041658 sec

» tic; s=test(Z), toc
s = 6.9847e+005
Elapsed time is 0.041293 sec

» tic; s=test(Z), toc
s = 6.9847e+005
Elapsed time is 0.041212 sec

» tic; s=test(Z), toc
s = 6.9847e+005
Elapsed time is 0.041292 sec

» tic; s=test(Z), toc
s = 6.9847e+005
Elapsed time is 0.041508 sec

»
```

test.m

```
function s=test(Z)
% written by Eric Anterrieu
s=0;
for m=1:size(Z,3)
for l=1:size(Z,2)
for k=1:size(Z,1)
   s=s+Z(k,l,m)*Z(k,l,m);
end
end
end
```

❸ Optimisation du code

MATLAB

```
» Z=rand(128,128,128);
» whos
  Name  Size          Bytes     Class
  Z     128x128x128   16777216  double

» tic; s=test(Z), toc
s = 6.9871e+005
Elapsed time is 0.040056 sec

» tic; s=test(Z), toc
s = 6.9871e+005
Elapsed time is 0.040043 sec

» tic; s=test(Z), toc
s = 6.9871e+005
Elapsed time is 0.040104 sec

» tic; s=test(Z), toc
s = 6.9871e+005
Elapsed time is 0.040135 sec

» tic; s=test(Z), toc
s = 6.9871e+005
Elapsed time is 0.040271 sec

»
```

test.m

```
function s=test(Z)
% written by Eric Anterrieu
s=0;
for k=1:prod(size(Z))
   s=s+Z(k)*Z(k);
end
```


❸ Optimisation du code

Vectorisation

Vérifier qu'une fonction (vectorisée, voir *built-in*!) de MATLAB ne remplisse pas déjà la fonction souhaitée, ou puisse être détournée pour la remplir.

❸ Optimisation du code

MATLAB

```
» Z=rand(128,128,128);
» whos
  Name  Size          Bytes      Class
  Z     128x128x128   16777216   double

» tic; s=test(Z), toc
s = 6.9922e+005
Elapsed time is 0.024217 sec

» tic; s=test(Z), toc
s = 6.9922e+005
Elapsed time is 0.023984 sec

» tic; s=test(Z), toc
s = 6.9922e+005
Elapsed time is 0.025095 sec

» tic; s=test(Z), toc
s = 6.9922e+005
Elapsed time is 0.025063 sec

» tic; s=test(Z), toc
s = 6.9922e+005
Elapsed time is 0.025010 sec

»
```

test.m

```
function s=test(Z)
% written by Eric Anterrieu
s=sum(Z(:).*Z(:));
```


❸ Optimisation du code

MATLAB

```
» Z=rand(128,128,128);
» whos
  Name  Size          Bytes      Class
  Z     128x128x128   16777216   double

» tic; s=test(Z), toc
s = 6.9927e+005
Elapsed time is 0.005695 sec

» tic; s=test(Z), toc
s = 6.9927e+005
Elapsed time is 0.005588 sec

» tic; s=test(Z), toc
s = 6.9927e+005
Elapsed time is 0.005642 sec

» tic; s=test(Z), toc
s = 6.9927e+005
Elapsed time is 0.005857 sec

» tic; s=test(Z), toc
s = 6.9927e+005
Elapsed time is 0.005791 sec

»
```

test.m

```
function s=test(Z)
% written by Eric Anterrieu
s=norm(Z(:))^2;
```

❸ Optimisation du code

Pré-allocation mémoire

L'allocation de mémoire pour les tableaux n'est pas nécessaire, mais vivement recommandée car elle contribue à l'optimisation du code:

- diminution du temps de calcul:
  - pas d'allocations partielles;
- réduction de la fragmentation de la mémoire:
  - moins de sauts de page;
  - meilleure utilisation des caches mémoire.

❸ Optimisation du code

MATLAB

```
» clear all
» tic; test; toc
Elapsed time is 0.302116 seconds.

» tic; test; toc
Elapsed time is 0.001141 sec

» tic; test; toc
Elapsed time is 0.001080 sec

» tic; test; toc
Elapsed time is 0.001067 sec

» clear all
» tic; test; toc
Elapsed time is 0.310490 sec

» clear all
» tic; test; toc
Elapsed time is 0.319505 sec

» clear all
» tic; test; toc
Elapsed time is 0.314024 sec

»
```

test.m

```
% suite de Fibonacci
n = 10000;
x(1) = 1;
x(2) = 1;
for k=3:n
   x(k) = x(k-1)+x(k-2);
end
```

❸ Optimisation du code

MATLAB

```
» clear all
» tic; test; toc
Elapsed time is 0.003123 seconds.

» tic; test; toc
Elapsed time is 0.001181 sec

» tic; test; toc
Elapsed time is 0.001173 sec

» tic; test; toc
Elapsed time is 0.001183 sec

» clear all
» tic; test; toc
Elapsed time is 0.003127 sec

» clear all
» tic; test; toc
Elapsed time is 0.003102 sec

» clear all
» tic; test; toc
Elapsed time is 0.003014 sec

»
```

test.m

```
% suite de Fibonacci
n = 10000;
x = ones(n,1);
for k=3:n
   x(k) = x(k-1)+x(k-2);
end
```


❸ Optimisation du code

Profiling

Lorsque l'optimisation du code n'est pas aussi simple, MATLAB dispose d'un outil de profilage, **`profile`**, qui retourne de nombreuses informations utiles pour l'optimisation du code (notamment, où chercher?).

❸ Optimisation du code

profile

MATLAB

```
» profile on
» fractal
» profview
» profile off
»
```

fractal.m

```
N = 75;
x = linspace(-2,2,400);
y = linspace(-2,2,400);
%
for k=1:length(x)
   for l=1:length(y)
      z = x(k)+j*y(l);
      for it=1:N
         z = z-(z^3-1)/(3*z^2);
      end
      Z(l,k) = angle(z);
   end
end
%
figure(1); colormap('prism');
imagesc(x,y,Z);
axis('xy'); axis('square');
hold('on');
plot(cos([0 2 4]*pi/3),
     sin([0 2 4]*pi/3),'k+');
```

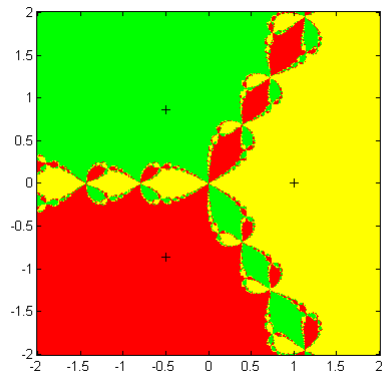

❸ Optimisation du code

profile

MATLAB

```
» profile on
» fractal
» profview
» profile off
»
```

Figure No. 1



fractal.m

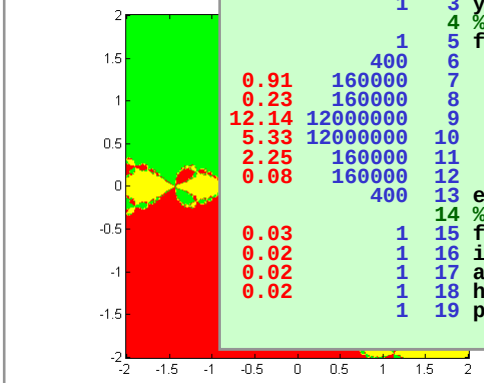
```
N = 75;
,2,400);
,2,400);

x)
th(y)
j*y(l);
N
(z^3-1)/(3*z^2);

angle(z);



');

s('square');

]*pi/3),
]*pi/3),'k+');
```

❸ Optimisation du code

profile

MATLAB
```
» profile on
» fractal
» profview
» profile off
»
```

Figure No. 1



PROFILER

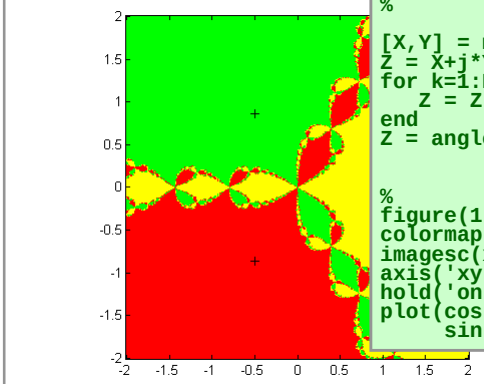
time	calls	line	
	1	1	N = 75;
	1	2	x = linspace(-2,2,400);
	1	3	y = linspace(-2,2,400);
		4	%
	1	5	for k=1:length(x)
	400	6	for l=1:length(y)
0.91	160000	7	z = x(k)+j*y(l);
0.23	160000	8	for it=1:N
12.14	12000000	9	z = z-(z^3-1)/(3*z^2);
5.33	12000000	10	end
2.25	160000	11	Z(l,k) = angle(z);
0.08	160000	12	end
	400	13	end
		14	%
0.03	1	15	figure(1); colormap('prism');
0.02	1	16	imagesc(x,y,Z);
0.02	1	17	axis('xy'); axis('square');
0.02	1	18	hold('on');
	1	19	plot(cos([0 2 4]*pi/3), sin([0 2 4]*pi/3),'k+');

❸ Optimisation du code

profile

MATLAB
```
» profile on
» fractal
» profview
» profile off
»
```

Figure No. 1



fractal.m
```
N = 75;
x = linspace(-2,2,400);
y = linspace(-2,2,400);
%
[X,Y] = meshgrid(x,y);
Z = X+j*Y;
for k=1:N
   Z = Z - (Z/3-1./(3*Z.^2));
end
Z = angle(Z);


%
figure(1);
colormap('prism');
imagesc(x,y,Z);
axis('xy'); axis('square');
hold('on');
plot(cos([0 2 4]*pi/3),
     sin([0 2 4]*pi/3),'k+');
```

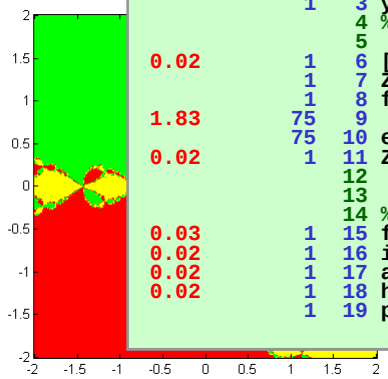
❸ Optimisation du code

profile

MATLAB

```
» profile on
» fractal
» profview
» profile off
»
```

Figure No. 1



PROFILER

```
time      calls  line
                1   1 N = 75;
                1   2 x = linspace(-2,2,400);
                1   3 y = linspace(-2,2,400);
                    4 %
                    5
 0.02           1   6 [X,Y] = meshgrid(x,y);
                1   7 Z = X+j*Y;
                1   8 for it=1:N
 1.83          75   9    Z = Z - (Z/3-1./(3*Z.^2));
               75  10 end
 0.02           1  11 Z = angle(z);
                   12
                   13
                   14 %
 0.03           1  15 figure(1); colormap('prism');
 0.02           1  16 imagesc(x,y,Z);
 0.02           1  17 axis('xy'); axis('square');
 0.02           1  18 hold('on');
                1  19 plot(cos([0 2 4]*pi/3),
                           sin([0 2 4]*pi/3),'k+');
```


❸ Optimisation du code

profile

MATLAB

```
» x=rand(1000);
» y=rand(1000);
» tmp=zeros(1000);
» profile on
» test
» profview
» profile off
»
```

test.m

```
tmp=x+y;
tmp=x-y;
tmp=x.*y;
tmp=x./y;
tmp=x.^y;
tmp=sqrt(x);
tmp=exp(x);
tmp=log(x);
tmp=sin(x);
tmp=tan(x);
```

❸ Optimisation du code

profile

MATLAB

```
» x=rand(1000);
» y=rand(1000);
» tmp=zeros(1000);
» profile on
» test
» profview
» profile off
»
```

PROFILER

```
time   calls  line
0.14       1   1 tmp=x+y;
           1   2 tmp=x-y;
0.14       1   3 tmp=x.*y;
0.14       1   4 tmp=x./y;
3.03       1   5 tmp=x.^y;
1.15       1   6 tmp=sqrt(x);
0.86       1   7 tmp=exp(x);
0.86       1   8 tmp=log(x);
0.58       1   9 tmp=sin(x);
0.72       1  10 tmp=tan(x);
```


❸ Optimisation du code

Pré-parsing

Lors de l'exécution d'un script ou d'une fonction, le m-code contenu dans le fichier .m est converti à la volée par MATLAB en p-code (ou pré-mâché, donc) plus rapide à interpréter. Ce p-code est stocké en mémoire (voir en cache!) jusqu'à son effacement par la commande **`clear`** (ou faute de place).

☺ Ceci explique pourquoi la seconde exécution et les suivantes sont plus rapides que la première.

❸ Optimisation du code

Pré-parsing

MATLAB dispose d'une commande **pcode** pour convertir avant l'exécution le m-code contenu dans un fichier .m en p-code stocké dans un fichier .p.

Puisque le p-code est plus rapide à interpréter que le m-code, l'exécution d'un script ou d'une fonction contenu dans un fichier .p sera plus rapide que celle du même script ou de la même fonction contenu dans le fichier .m.

☹ Vrai seulement pour la première exécution!

❸ Optimisation du code

Pré-parsing

Avec les améliorations apportées régulièrement à l'environnement, MATLAB fabrique le p-code à la volée et le stocke en mémoire très rapidement, de sorte qu'il n'est pas nécessaire d'invoquer la fonction **pcode** pour gagner du temps!

Un seul intérêt à la fabrication de p-code dans des fichiers .p: diffusion de programmes lorsqu'on veux cacher le code source contenus dans les fichiers .m.

☺ Reverse- engineering plus lent sur p-code!

❸ Optimisation du code

pcode

MATLAB

```
»
» clear all
» tic; fractal; toc
Elapsed time is 20.235623 se

» clear all
» tic; fractal; toc
Elapsed time is 20.106776 se

» clear all
» tic; fractal; toc
Elapsed time is 20.221359 se

» clear all
» tic; fractal; toc
Elapsed time is 20.138243 se

» clear all
» tic; fractal; toc
Elapsed time is 20.204121 se

»
```

fractal.m

```
N = 75;
x = linspace(-2,2,400);
y = linspace(-2,2,400);
%
for k=1:length(x)
   for l=1:length(y)
      z = x(k)+j*y(l);
      for it=1:N
         z = z-(z^3-1)/(3*z^2);
      end
      Z(l,k) = angle(z);
   end
end
%
figure(1); colormap('prism');
imagesc(x,y,Z);
axis('xy'); axis('square');
hold('on');
plot(cos([0 2 4]*pi/3),
     sin([0 2 4]*pi/3),'k+');
```


❸ Optimisation du code

pcode

MATLAB

```
» pcode fractal
» clear all
» tic; fractal; toc
Elapsed time is 20.131405 se

» clear all
» tic; fractal; toc
Elapsed time is 20.117084 se

» clear all
» tic; fractal; toc
Elapsed time is 20.153935 se

» clear all
» tic; fractal; toc
Elapsed time is 20.137558 se

» clear all
» tic; fractal; toc
Elapsed time is 20.168137 se

»
```

fractal.p

```
^@P-file 3.1^@^@^@^@^@^@;^@^@^@^@^@^@^@T^@
ky§½wZuè|<ìAþÈù>ç¼%Ús¹Ñv?W}ÒË &ïì-BûhÙÞG
¤E°Ò`1Ve 3Ï‘vÿn=’Ë é.×¶ÕÊ#±•&/)mbÛAy~'B¥
 s ‘¶ÿý½Òãén× Õ£pÁæ uè|<ìAþÈù>ç¼%ÚcrÁ¦ïC
»(ÙÞG6sãL(ø ÝÂl^( ²=,•L”i•x©.ÒË&ïì-Bû7,5
ÓÃ aÆO@Mâ›Ïy~§w¥«óµÑ×?žý·hÙÞG¤E°Ò`1Ve3lí
B»èky§½wZuè|<ìAþÈù>ç¼%Ús¹Ñv?W}ÒË&ïì-BûhÙ
ÞG¤E°Ò`1Ve3Ï‘vÿn=’Ë é.×¶ÕÊ#±•&/)mbÛAy~'B
¥s‘¶ÿý½Òãén× Õ£pÁæ líB»èÙ^ûïàäL(ø ÝÂl^(
²=,•L”i•x©7,5ÓÃ aÆO@Mâ›Ïy~§w¥«žý·ky§½wZu
è|<ìAþÈù>ç¼%Ús¹Ñv?W}ÒË&ïì-BûhÙÞG¤E°Ò`1Ve
3Ï‘vÿn=’Ë é.×¶ÕÊ#±•&/)mbÛAy~'B¥s‘¶ÿý½Òãé
n× Õ£pÁæ líB»èÙ^ky§½wZuè|<ìAþÈù>ç¼%ÚcrÁ¦
ïC»(ÙÞG6s¹Ñv?W}ûïàäL(øÝÂl^(²=,•L”i•x©.ÒË
&ïì-Bû7,5ÓÃaÆO@Mâ›Ïy~§w¥«óµÑ×?žý·hÙÞG¤E°
Ò`1Vevÿn=’Ë é.×¶ÕÊ#±•&/)mbÛAy~'B¥s‘¶ÿý½Ò
ãé£pÁæ líB»èky§½wZuè|<ìAþÈù>ç¼%Ús¹Ñv?W}Ò
Ë&ïì-BûhÙÞG¤E°Ò`1Ve3Ï‘vÿn=’Ë é.×¶ÕÊ#±•&/
)mbÛAy~'B¥s‘¶ÿý½Òã¥«óµÑ×?žý·.¥«óµÑ×?žý·k
y§½wZuè|<ìAþÈù>ç¼%Ús¹Ñv?W}ÒË&ïì-BûhÙÞG¤E
°Ò`1Ve3Ï‘vÿn=’Ë é.×¶ÕÊ#±•&/)mbÛAy~'B¥s^G
```

❸ Optimisation du code

Vérifier les dépendances

La commande **`depfun`** retourne la liste des fichiers .m ou .p, et éventuellement des fonctions de la bibliothèque, qui sont nécessaires à l'utilisation d'une fonction:

```
>> [Mfiles,Builts]=depfun('mfile');
```


❸ Optimisation du code

Vérifier les dépendances

La commande **`inmem`** retourne la liste des M-files, et éventuellement des MEX-files, qui ont été interprétés et chargés en mémoire récemment (depuis le dernier **`clear all`**):

```
>> clear all;
>> mfile;
>> [Mfiles, MEXfiles]=inmem;
```

❹ Lien vers d'autres langages

MATLAB propose des interfaces qui permettent:

- d'appeler un autre langage de programmation depuis un script ou une fonction MATLAB;
- d'invoquer le moteur de calcul de MATLAB depuis un autre langage de programmation;
- d'accéder aux fichiers de données de MATLAB .mat depuis un autre langage de programmation.

❹ Lien vers d'autres langages

Appeler le C ou le FORTRAN depuis MATLAB

Bien que l'environnement de travail de MATLAB soit autonome pour la programmation et la manipulation des données, il peut parfois s'avérer utile d'interagir avec des données et des programmes extérieur à cet environnement.

De ce point de vue, MATLAB propose une interface vers des programmes extérieurs écrits, par exemple, en C ou en FORTRAN.

❹ Lien vers d'autres langages

Appeler le C ou le FORTRAN depuis MATLAB

La librairie *libmex.lib* qui réalise cette interface est une collection de routines qui permettent d'appeler depuis MATLAB des fonctions écrites en C ou en FORTRAN de la même manière que cela se fait avec les fonctions internes de MATLAB.

Ces fonctions MEX (pour Matlab EXecutable) sont liées dynamiquement par MATLAB au moment de leur exécution.

Les routines de la librairie ont le préfixe **`mex`**.

❹ Lien vers d'autres langages

Appeler le C ou le FORTRAN depuis MATLAB

La librairie *libmx.lib* est une collection de routines qui permettent de créer/manipuler/détruire des tableaux MATLAB de type **`mxArray`** dans des fonctions MEX écrites en C ou en FORTRAN.

Les routines de la librairie ont le préfixe **`mx`**.

❹ Lien vers d'autres langages

Appeler le C ou le FORTRAN depuis MATLAB

Alors que les fichiers écrit dans le langage MATLAB ont une extension .m indépendante de la plate-forme, les fonctions MEX écrites dans un autre langage ont une extension spécifique à la plate-forme qu'il est possible de connaitre à l'aide de la fonction **`mexext`**.

L'aide en ligne d'une fonction MEX sera lue depuis le fichier .m correspondant (s'il existe).

❹ Lien vers d'autres langages

MATLAB

```
» list=mexext('all');
» for k=1:length(list)
  disp(sprintf('Arch: %s  Ext: %s',list(k).arch,list(k).ext));
  end

Arch: glnx86   Ext: mexglx
Arch: glnxa64  Ext: mexa64
Arch: sol2     Ext: mexsol
Arch: sol64    Ext: mexs64
Arch: win32    Ext: mexw32
Arch: win64    Ext: mexw64
Arch: mac      Ext: mexmac
Arch: maci     Ext: mexmaci

»
```

❹ Lien vers d'autres langages

MATLAB

```
» mex -setup
Please choose your compiler for building MEX files:

Would you like mex to locate installed compilers [y]/n? y

Select a compiler:
[1] Lcc-win32 C 2.4.1 in C:\PROGRA~1\MATLAB\R2007a\sys\lcc

[0] None

Compiler: 1

Please verify your choices:

Compiler: Lcc-win32 C 2.4.1
Location: C:\PROGRA~1\MATLAB\R2007a\sys\lcc

Are these correct?([y]/n): y

Trying to update options file mexopts.bat
From template C:\PROGRA~1\MATLAB\R2007a\bin\win32\mexopts\lccopts.bat

Done ...

»
```


❹ Lien vers d'autres langages

C:\PROGRA~1\MATLAB\R2007\bin\win32\mexopts\lccopts.bat

```
@echo off
rem
rem Options file for building MEX programs using LCC compiler
rem
rem *** General parameters ***************************************************
set MATLAB=%MATLAB%
set PATH=%MATLAB%\sys\lcc\bin;%PATH%
rem *** Compiler parameters **************************************************
set COMPILER=lcc
set COMPFLAGS=-c -Zp8 -I"%MATLAB%\sys\lcc\include" -DMATLAB_MEX_FILE -noregistrylookup
set OPTIMFLAGS=-DNDEBUG
set DEBUGFLAGS=-g4
set NAME_OBJECT=-Fo
set MW_TARGET_ARCH=win32
rem *** Library creation command ********************************************
set PRELINK_CMDS1=lcc %COMPFLAGS% "%MATLAB%\sys\lcc\mex\lccstub.c" -Fo%LIB_NAME%2.obj
rem *** Linker parameters ****************************************************
set LIBLOC=%MATLAB%\extern\lib\win32\lcc
set LINKER=lcclnk
set LINKFLAGS= -tmpdir "%OUTDIR%." -dll "%MATLAB%\extern\lib\win32\lcc\%ENTRYPOINT%.def" -
L"%MATLAB%\sys\lcc\lib" -libpath "%LIBLOC%" %LIB_NAME%2.obj
set LINKFLAGSPOST=libmx.lib libmex.lib libmat.lib
set LINKOPTIMFLAGS=-s
set LINKDEBUGFLAGS=
set LINK_FILE=
set LINK_LIB=
set NAME_OUTPUT=-o "%OUTDIR%%MEX_NAME%%MEX_EXT%"
set RSP_FILE_INDICATOR=@
rem *** Resource compiler parameters ****************************************
set RC_COMPILER=lrc -I"%MATLAB%\sys\lcc\include" -noregistrylookup -fo"%OUTDIR%mexversion.res"
set RC_LINKER=

set POSTLINK_CMDS1=del %LIB_NAME%2.obj
set POSTLINK_CMDS2=del "%OUTDIR%%MEX_NAME%.exp"
set POSTLINK_CMDS3=del "%OUTDIR%%MEX_NAME%.lib«
```

❹ Lien vers d'autres langages

Appeler le C ou le FORTRAN depuis MATLAB

Le code source d'une fonction MEX comporte deux parties distinctes:

- une fonction de calcul qui contient le code pour effectuer les calculs;
- une fonction passerelle qui réalise l'interface entre MATLAB et la fonction de calcul.

❹ Lien vers d'autres langages

Appeler le C ou le FORTRAN depuis MATLAB

Le nom de la fonction passerelle est **`mexFunction`** .

Cette fonction doit apparaitre dans le code source de toutes les fonctions MEX: c'est le point d'entrée obligatoire d'une fonction MEX (ce n'est pas la fonction **`main`** en C, ni **`program`** en FORTRAN).

❹ Lien vers d'autres langages

Appeler le FORTRAN depuis MATLAB

```
#include "fintrf.h"
subroutine mexFunction(nlhs,plhs,nrhs,prhs)
integer nlhs, plhs(*), nrhs, prhs(*)
...
return
end
```

nlhs : nombre d'arguments en sortie
plhs : pointeur vers les arguments de sortie
nrhs : nombre d'arguments en entrée
prhs : pointeur vers les arguments d'entrée

❹ Lien vers d'autres langages

Appeler le FORTRAN depuis MATLAB

MATLAB

Lors de l'appel à la fonction MEX myfunct:
`» [C,D]=myfunct(A,B);`
MATLAB passe A et B à la fonction MEX:
A est assigné à **prhs(1)**
B est assigné à **prhs(2)**.

```
integer A
A = prhs(1)
integer B
B = prhs(2)
```

myfunct.c

```
#include "fintrf.h"
subroutine mexFunction(nlhs,plhs,nrhs,prhs)
integer nlrh, plhs(*), nrhs, prhs(*)
...
call myfunct(a,b,c,...);
...
return
end

subroutine myfunct(x,y,z,...)
type x,type y,type z,...
...
return
end
```

MATLAB

Au retour de la fonction MEX myfunct:
`» [C,D]=myfunct(A,B);`
plhs(1) est assigné à **C**
plhs(2) est assigné à **D**.

```
integer C
C = plhs(1)
integer D
D = plhs(2)
```

❹ Lien vers d'autres langages

Appeler le C depuis MATLAB

```
#include "mex.h"
void mexFunction(int nlhs, mxArray *plhs[],
                 int nrhs, const mxArray *prhs[])
{
...
}
```

nlhs : nombre d'arguments en sortie
plhs : tableau de pointeurs vers les arguments de sortie
nrhs : nombre d'arguments en entrée
prhs : tableau de pointeurs vers les arguments d'entrée

❹ Lien vers d'autres langages

Appeler le C depuis MATLAB

MATLAB

Lors de l'appel à la fonction MEX myfunct:
`» [C,D]=myfunct(A,B);`
MATLAB passe A et B à la fonction MEX:
A est assigné à **prhs[0]**
B est assigné à **prhs[1]**.

```
const mxArray *A
A = prhs[0]
```

```
const mxArray *B
B = prhs[1]
```

myfunct.c

```
#include "mex.h"
void mexFunction(int nlhs, mxArray *plhs[],
                 int nrhs, const mxArray *prhs[])
{
...
myfunct(a,b,c,...);
...
return;
}
type myfunct(type x,type y,type z,...)
{
...
}
```

MATLAB

Au retour de la fonction MEX myfunct:
`» [C,D]=myfunct(A,B);`
plhs[0] est assigné à **C**
plhs[1] est assigné à **D**.

```
mxArray *C
C = plhs[0]
```

```
mxArray *D
D = plhs[1]
```

❹ Lien vers d'autres langages

mexfunct1.c

```
#include "mex.h"

void mexfunct1(double *B, double alpha, double *x, double *y, double *A, int m, int n)
{
int i, j , k;

/* rank 1 operation alpha*x*y' + A as performed by BLAS level 2 routine ESGER */
for (i=0, k=0; i<n; i++)
for (j=0; j<m; j++, k++) B[k] = alpha*x[j]*y[i] + A[k];
}

void mexFunction(int nlhs, mxArray *plhs[],
                 int nrhs, const mxArray *prhs[])
{
int    m, n;
double alpha, *x, *y, *A, *B;

if (nrhs != 4) mexErrMsgTxt("Four inputs are required.");
if (nlhs != 1) mexErrMsgTxt("One output is required.");

if (!mxIsDouble(prhs[0]) || mxGetN(prhs[0])*mxGetM(prhs[0]) != 1)
    mexErrMsgTxt("First input must be a real-valued scalar.");
if (!mxIsDouble(prhs[1]) || mxGetN(prhs[1]) != 1)
    mexErrMsgTxt("Second input must be a real-valued column vector.");
m = mxGetM(prhs[1]);
if (!mxIsDouble(prhs[2]) || mxGetN(prhs[2]) != 1)
    mexErrMsgTxt("Third input must be a real-valued column vector.");
n = mxGetM(prhs[2]);
if (!mxIsDouble(prhs[3]) || mxGetM(prhs[3]) != m || mxGetN(prhs[3]) != n)
    mexErrMsgTxt("Fourth input must be a real-valued matrix.");

alpha = mxGetScalar(prhs[0]);
x = mxGetPr(prhs[1]);
y = mxGetPr(prhs[2]);
A = mxGetPr(prhs[3]);

plhs[0] = mxCreateDoubleMatrix(m,n,mxREAL);
B = mxGetPr(plhs[0]);

mexfunct1(B,alpha,x,y,A,m,n);
}
```


❹ Lien vers d'autres langages

MATLAB

```
» mex -f lccopts.bat mexfunct1.c
» rand('seed',0)
» n=3; m=4; x=rand(n,1); y=rand(m,1); A=rand(n,m); alpha=2;
» B=mexfunct1(alpha,x,y,A)

B =

    1.1284    0.9390    0.5514    0.9142
    0.0985    0.7591    0.1029    0.6378
    0.9758    1.2768    0.9382    1.6357

» alpha*x*y'+A

ans =

    1.1284    0.9390    0.5514    0.9142
    0.0985    0.7591    0.1029    0.6378
    0.9758    1.2768    0.9382    1.6357

»
```

❹ Lien vers d'autres langages

Appeler MATLAB depuis le C ou le FORTRAN

La librairie du moteur MATLAB *libeng.lib* est une collection de fonctions qui permettent d'appeler le cœur de MATLAB depuis n'importe quel programme écrit en C ou en FORTRAN, faisant ainsi de MATLAB un moteur de calcul.

Les programmes qui utilisent ainsi le moteur de calcul de MATLAB créent un processus MATLAB et communiquent avec lui, via des pipes sous UNIX, à travers une interface COM sous WINDOWS®.

❹ Lien vers d'autres langages

Appeler MATLAB depuis le C ou le FORTRAN

Les fonctions de la librairie *libeng.lib* permettent:

- de démarrer/arrêter le processus du moteur de MATLAB;
- d'échanger des données vers/depuis le moteur de MATLAB;
- d'envoyer des commandes au moteur de MATLAB pour qu'il les interprète et récupérer le résultat.

Les fonctions de la librairie ont le préfixe **eng**.

❹ Lien vers d'autres langages

Appeler MATLAB depuis le C ou le FORTRAN

Que peut-on faire avec le moteur de MATLAB utilisé depuis un programme écrit en C ou en FORTRAN?

- appeler une fonction de haut-niveau pour **`inv`**erser une matrice, par exemple, ou bien utiliser la **`fft`**,… ;
- construire un environnement d'analyse de données avec l'interface graphique *(front-end)* en C et l'analyse *(back-end)* en MATLAB.

MATLAB est alors utilisée comme une librairie mathématique.

❹ Lien vers d'autres langages

Appeler MATLAB depuis le C ou le FORTRAN

Le moteur de MATLAB est un processus à part qui s'exécute en arrière-plan du programme écrit en C ou en FORTRAN. Ceci procure plusieurs avantages:

- sous UNIX, le processus du moteur MATLAB peut s'exécuter sur la même machine ou sur n'importe quelle machine du réseau local (y compris d'architecture différente);
- au lieu de lier le programme en C/FORTRAN avec tout MATLAB, seule un petit code de communication est inclus dans l'exécutable.

❹ Lien vers d'autres langages

Appeler MATLAB depuis le C ou le FORTRAN

La librairie *libeng.lib* contient les fonctions suivantes:

engOpen	démarre le processus MATLAB
engClose	termine le processus MATLAB
engGetVariable	reçoit un tableau de MATLAB
engPutVariable	envoie un tableau à MATLAB
engEvalString	exécute une commande dans MATLAB
engOutputBuffer	créer un *buffer* pour récupérer tout ce qui normalement apparaît dans la fenêtre de commande

❹ Lien vers d'autres langages

C:\PROGRA~1\MATLAB\R2007\bin\win32\mexopts\lccengmatopts.bat

```
@echo off
rem
rem Options file for building standalone ENG/MAT programs using LCC compiler
rem
rem *** General parameters ************************************************
set MATLAB=%MATLAB%
set PATH=%MATLAB%\sys\lcc\bin;%PATH%
set LCCMEX=%MATLAB%\sys\lcc\mex
set PERL="%MATLAB%\sys\perl\win32\bin\perl.exe"
set MW_TARGET_ARCH=win32
rem *** Compiler parameters ************************************************
set COMPILER=lcc
set OPTIMFLAGS=-DNDEBUG
set DEBUGFLAGS=-g4
set COMPFLAGS=-c -Zp8 -I"%MATLAB%\sys\lcc\include" -noregistrylookup
set NAME_OBJECT=-Fo
rem *** Linker parameters **************************************************
set LIBLOC=%MATLAB%\extern\lib\win32\lcc
set LINKER=lcclnk
set LINKFLAGS=-tmpdir "%OUTDIR%." -L"%MATLAB%\sys\lcc\lib" -libpath "%LIBLOC%"
set LINKFLAGSPOST=libmx.lib libmat.lib libeng.lib
set LINKOPTIMFLAGS=
set LINKDEBUGFLAGS=
set LINK_FILE=
set LINK_LIB=
set NAME_OUTPUT=-o "%OUTDIR%%MEX_NAME%.exe"
set RSP_FILE_INDICATOR=@
rem *** Resource compiler parameters ***************************************
set RC_COMPILER=
set RC_LINKER=
```

❹ Lien vers d'autres langages

engine1.c

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include "engine.h"

int main()
{
Engine *eng;
mxArray *D = NULL, *T = NULL;
int k;
double *d, *t, data[10] = {0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0};

if (!(eng = engOpen("\0"))) {printf("Can't start MATLAB engine\n"); return(-1);}
printf("Starting MATLAB engine...\n\n");

T = mxCreateDoubleMatrix(1,10,mxREAL);
t = mxGetPr(T);
for (k=0; k<10; k++) t[k] = data[k];
engPutVariable(eng,"T",T);

engEvalString(eng,"G=9.81;");
engEvalString(eng,"D = -0.5*G*T.^2;");
engEvalString(eng,"plot(T,D);");
engEvalString(eng,"title('Distance vs. Time for a falling object');");
engEvalString(eng,"xlabel('Time (seconds)');");
engEvalString(eng,"ylabel('Distance (meters)');");

D = engGetVariable(eng,"D");
if ((D != NULL) && (mxGetNumberOfElements(D) == 10))
  {
  d = mxGetPr(D);
  for (k=0; k<10; k++) printf("%f %f\n",t[k],d[k]);
  }

printf("Hit return to continue...\n\n");
fgetc(stdin);

engEvalString(eng,"close;");
mxDestroyArray(D);
mxDestroyArray(T);
engClose(eng);

printf("Done\n");
return(0);
}
```


❹ Lien vers d'autres langages

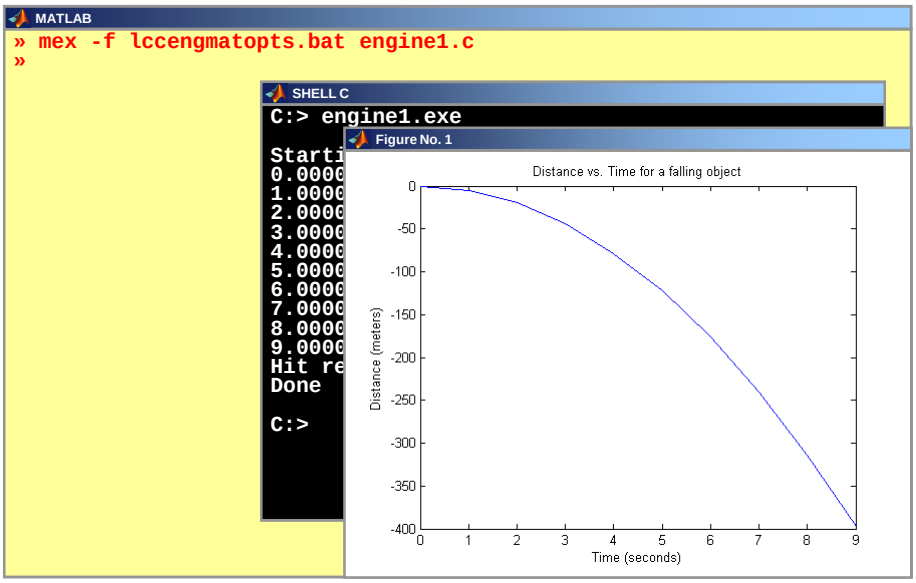
MATLAB

```
» mex -f lccengmatopts.bat engine1.c
»
```

SHELL C

```
C:> engine1.exe

Starting MATLAB engine...
0.000000 0.000000
1.000000 -4.905000
2.000000 -19.620000
3.000000 -44.145000
4.000000 -78.480000
5.000000 -122.625000
6.000000 -176.580000
7.000000 -240.345000
8.000000 -313.920000
9.000000 -397.305000
Hit return to continue...
Done

C:>
```

❹ Lien vers d'autres langages



❹ Lien vers d'autres langages

Accéder aux fichiers .mat depuis le C ou le FORTRAN

La librairie *libmat.lib* est une interface avec les fichiers .mat. C'est une collection de fonctions qui permettent d'accéder aux fichiers de données au format .mat de MATLAB depuis un programme écrit en C ou en FORTRAN.

Le format (propriétaire) .mat de MATLAB peut ainsi devenir un format d'échange de données avec d'autres applications.

❹ Lien vers d'autres langages

Accéder aux fichiers .mat depuis le C ou le FORTRAN

Les fonctions de la librairie *libmat.lib* permettent:

- d'ouvrir/fermer un fichier;
- de lire/écrire des données dans un fichier;
- de rajouter/supprimer des données dans un fichier.

Les fonctions de la librairie ont le préfixe **`mat`**.

❹ Lien vers d'autres langages

Accéder aux fichiers .mat depuis le C ou le FORTRAN

La librairie *libmat.lib* contient les fonctions suivantes:

`matOpen`	ouvre le fichier
`matClose`	ferme le fichier
`matGetDir`	liste les variables du fichier
`matGetNextVariableInfo`	lit l'entête de la prochaine variable
`matGetNextVariable`	lit la prochaine variable
`matGetVariableInfo`	lit l'entête d'une variable du fichier
`matGetVariable`	lit une variable depuis le fichier
`matPutVariable`	écrit une variable dans le fichier
`matPutVariableAsGlobal`	écrit une variable dans le fichier
`matDeleteVariable`	supprime une variable du fichier

❹ Lien vers d'autres langages

C:\PROGRA~1\MATLAB\R2007\bin\win32\mexopts\lccengmatopts.bat

```
@echo off
rem
rem Options file for building standalone ENG/MAT programs using LCC compiler
rem
rem *** General parameters ***************************************************
set MATLAB=%MATLAB%
set PATH=%MATLAB%\sys\lcc\bin;%PATH%
set LCCMEX=%MATLAB%\sys\lcc\mex
set PERL="%MATLAB%\sys\perl\win32\bin\perl.exe"
set MW_TARGET_ARCH=win32
rem *** Compiler parameters **************************************************
set COMPILER=lcc
set OPTIMFLAGS=-DNDEBUG
set DEBUGFLAGS=-g4
set COMPFLAGS=-c -Zp8 -I"%MATLAB%\sys\lcc\include" -noregistrylookup
set NAME_OBJECT=-Fo
rem *** Linker parameters ****************************************************
set LIBLOC=%MATLAB%\extern\lib\win32\lcc
set LINKER=lcclnk
set LINKFLAGS=-tmpdir "%OUTDIR%." -L"%MATLAB%\sys\lcc\lib" -libpath "%LIBLOC%"
set LINKFLAGSPOST=libmx.lib libmat.lib libeng.lib
set LINKOPTIMFLAGS=
set LINKDEBUGFLAGS=
set LINK_FILE=
set LINK_LIB=
set NAME_OUTPUT=-o "%OUTDIR%%MEX_NAME%.exe"
set RSP_FILE_INDICATOR=@
rem *** Resource compiler parameters *****************************************
set RC_COMPILER=
set RC_LINKER=
```


❹ Lien vers d'autres langages

matfile1.c

```
#include <stdio.h>
#include <stdlib.h>
#include "mat.h"

int main(int argc, char **argv)
{
MATFile *fp;
mxArray *mxPtr1 = NULL, *mxPtr2 = NULL, *mxPtr3 = NULL;
int     k, status;
double  *cPtr1,  *cPtr2;
double  data[9] = {1.0, 4.0, 7.0, 2.0, 5.0, 8.0, 3.0, 6.0, 9.0};

printf("Creating mat-file %s...\n\n",argv[1]);

fp = matOpen(argv[1],"w");
if (fp == NULL) {printf("Error creating mat-file %s\n",argv[1]); return(-1);}

mxPtr1 = mxCreateDoubleMatrix(1,1,mxREAL);
if (mxPtr1 == NULL) {printf("Unable to create mxArray\n"); return(-1);}
cPtr1 = mxGetPr(mxPtr1);
*cPtr1 = 3.14;

mxPtr2 = mxCreateDoubleMatrix(3,3,mxREAL);
if (mxPtr2 == NULL) {printf("Unable to create mxArray\n"); return(-1);}
cPtr2 = mxGetPr(mxPtr2);
for (k=0; k<9; k++) cPtr2[k] = data[k];

mxPtr3 = mxCreateString("MATLAB: the language of technical computing");
if (mxPtr3 == NULL) {printf("Unable to create string mxArray\n"); return(-1);}

status = matPutVariable(fp,"x",mxPtr1);
if (status != 0) {printf("Error using matPutVariable\n"); return(-1);}

status = matPutVariableAsGlobal(fp,"A",mxPtr2);
if (status != 0) {printf("Error using matPutVariableAsGlobal\n"); return(-1);}

status = matPutVariable(fp,"s",mxPtr3);
if (status != 0) {printf("Error using matPutVariable\n"); return(-1);}

mxDestroyArray(mxPtr1);
mxDestroyArray(mxPtr2);
mxDestroyArray(mxPtr3);

if (matClose(fp) != 0) {printf("Error closing mat-file %s\n",argv[1]); return(-1);}

printf("Done\n");
return(0);
}
```

❹ Lien vers d'autres langages

MATLAB

```
» mex -f lccengmatopts.bat matfile1.c
»
```

SHELL C

```
C:> matfile1.exe test.mat

Creating mat-file test.mat...
Done

C:>
```


❹ Lien vers d'autres langages

MATLAB

```
» mex -f lccengmatopts.bat matfile1.c
» whos -file test.mat
  Name      Size            Bytes  Class     Attributes

  A         3x3                72  double    global
  s         1x43               86  char
  x         1x1                 8  double

» load test.mat
» x
x =
   3.1400

» s
s =
   MATLAB: the language of technical computing

» A
A =
   1   2   3
   4   5   6
   7   8   9

»
```

❹ Lien vers d'autres langages

matfile2.c

```
#include <stdio.h>
#include <stdlib.h>
#include "mat.h"

int main(int argc, char **argv)
{
MATFile *fp;
mxArray *mxPtr1 = NULL, *mxPtr2 = NULL, *mxPtr3 = NULL;
char    str[256];
int     k, status;
double  *cPtr1,  *cPtr2;

printf("Reading mat-file %s...\n\n",argv[1]);

fp = matOpen("test.mat","r");
if (fp == NULL) {printf("Error reading mat-file %s\n",argv[1]); return(-1);}

mxPtr1 = matGetVariable(fp,"x");
if (mxPtr1 == NULL) {printf("Error using matGetVariable\n"); return(-1);}
cPtr1 = mxGetPr(mxPtr1);
printf("x = %f\n",*cPtr1);

mxPtr2 = matGetVariable(fp,"A");
if (mxPtr2 == NULL) {printf("Error using matGetVariable\n"); return(-1);}
cPtr2 = mxGetPr(mxPtr2);
printf("A =");
for (k=0; k<9; k++) printf(" %g",cPtr2[k]);
printf("\n");

mxPtr3 = matGetVariable(fp,"s");
if (mxPtr3 == NULL) {printf("Error using matGetVariable\n"); return(-1);}
status = mxGetString(mxPtr3, str, sizeof(str));
if(status != 0) {printf("Unable to create string\n"); return(-1);}
printf("s = %s\n",str);

mxDestroyArray(mxPtr1);
mxDestroyArray(mxPtr2);
mxDestroyArray(mxPtr3);

if (matClose(fp) != 0) {printf("Error closing mat-file %s\n",argv[1]); return(-1);}

printf("Done\n");
return(0);
}
```


❹ Lien vers d'autres langages

MATLAB

```
» mex -f lccengmatopts.bat matfile2.c
»
```

SHELL C

```
C:> matfile2.exe test.mat

Reading mat-file test.mat...
x = 3.140000
A = 1 4 7 2 5 8 3 6 9
s = MATLAB: the language of technical computing
Done

C:>
```

❹ Lien vers d'autres langages

matfile3.c

```
#include <stdio.h>
#include <stdlib.h>
#include "mat.h"

int main(int argc, char **argv)
{
MATFile *fp;
mxArray *mxPtr = NULL;
char    *name, **dir;
int     k, l, ndim, ndir, *dim;

printf("Reading mat-file %s...\n\n",argv[1]);

fp = matOpen(argv[1], "r");
if (fp == NULL) {printf("Error opening file %s\n",argv[1]); return(-1);}

dir = (char **)matGetDir(fp,&ndir);
if (dir == NULL) {printf("Error reading directory of file %s\n",argv[1]); return(-1);}
printf("Directory of %s:\n",argv[1]);
for (k=0; k<ndir; k++) printf("%s\n",dir[k]);
mxFree(dir);

if (matClose(fp) != 0) {printf("Error closing file %s\n",argv[1]); return(-1);}
fp = matOpen(argv[1], "r");
if (fp == NULL) {printf("Error opening file %s\n",argv[1]); return(-1);}

printf("Information from headers:\n");
for (k=0; k<ndir; k++)
  {
  mxPtr = matGetNextVariableInfo(fp,&name);
  if (mxPtr == NULL) {printf("Error reading in file %s\n",argv[1]); return(-1);}
  printf("%s is a ",name);
  ndim=mxGetNumberOfDimensions(mxPtr);
  dim =mxGetDimensions(mxPtr);
  printf("%d",dim[0]);
  for (l=1; l<ndim; l++) printf("x%d",dim[l]);
  printf(" array of ");
  printf("%s and ",mxGetClassName(mxPtr));
  if (mxIsFromGlobalWS(mxPtr)) printf("was a global variable when saved\n");
  else                         printf("was a local variable when saved\n");
  mxDestroyArray(mxPtr);
  }

if (matClose(fp) != 0) {printf("Error closing file %s\n",argv[1]); return(-1);}
printf("Done\n");
return(0);
}
```


❹ Lien vers d'autres langages

MATLAB

```
» mex -f lccengmatopts.bat matfile3.c
»
```

SHELL C

```
C:> matfile3.exe test.mat

Reading mat-file test.mat...
Directory of test.mat:
x
A
s
Information from headers:
x is a 1x1 array of double
  and was a local variable when saved
A is a 3x3 array of double
  and was a global variable when saved
s is a 1x43 array of char
  and was a local variable when saved
Done

C:>
```