Le logiciel de simulation MATLAB
Partie 5
Programmation en function

Professeur Ali Tahri
Université des sciences et de la technologie d'Oran Mohamed Boudiaf

1. Les fonctions

1.1 Les fonctions simples

Matlab permet de définir des fonctions. Une fonction est un ensemble d'instructions regroupées de manière à ne pas devoir les répéter régulièrement.

Une fonction peut prendre des arguments et renvoyer des valeurs. (Par exemple, si l'on fournit à une fonction une date de naissance, cette fonction peut renvoyer l'âge de la personne).

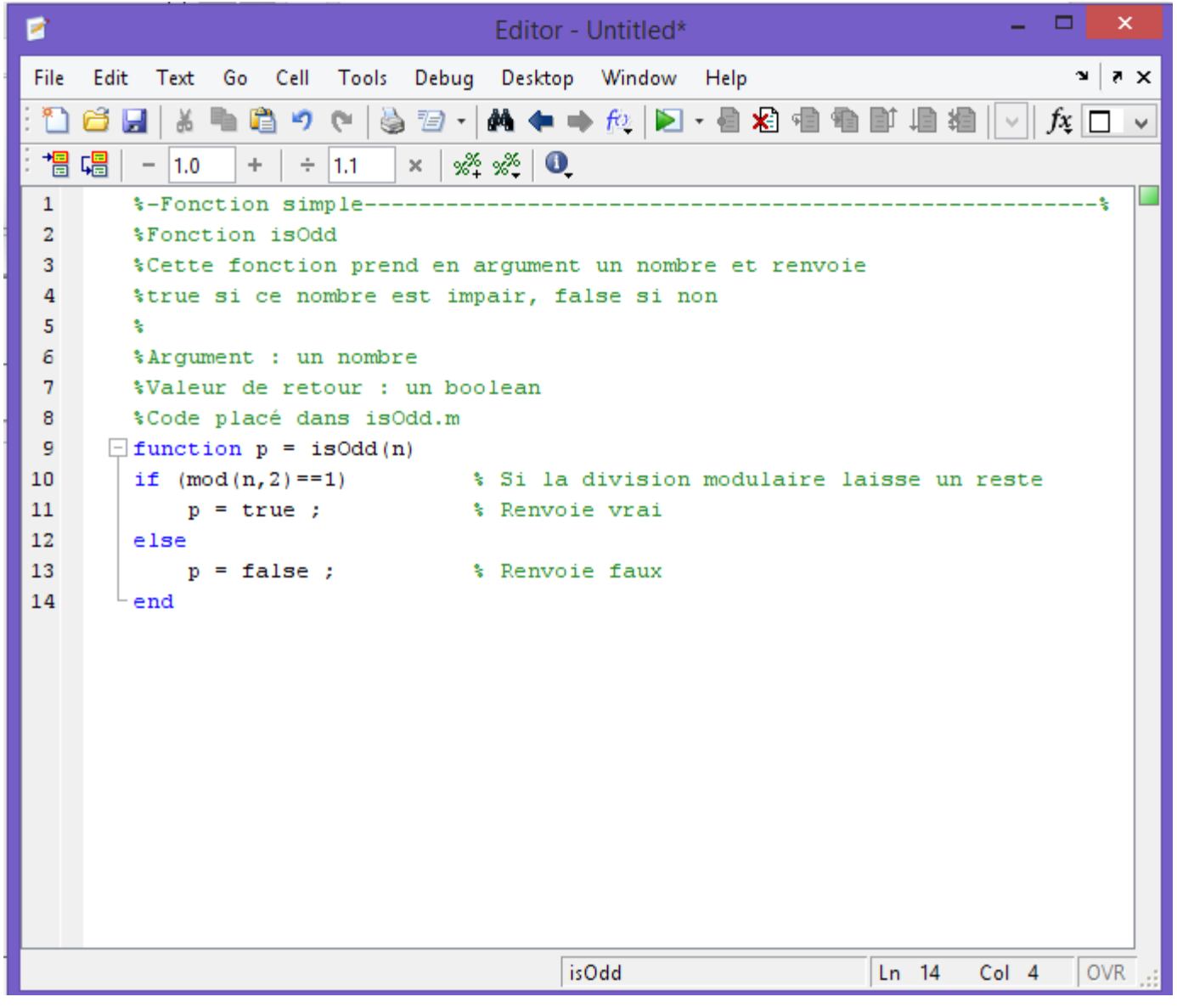
La définition d'une fonction se fait de la manière suivante :

function *arguments de sortie* = nomDeMaFonction (*arguments dentrée*).

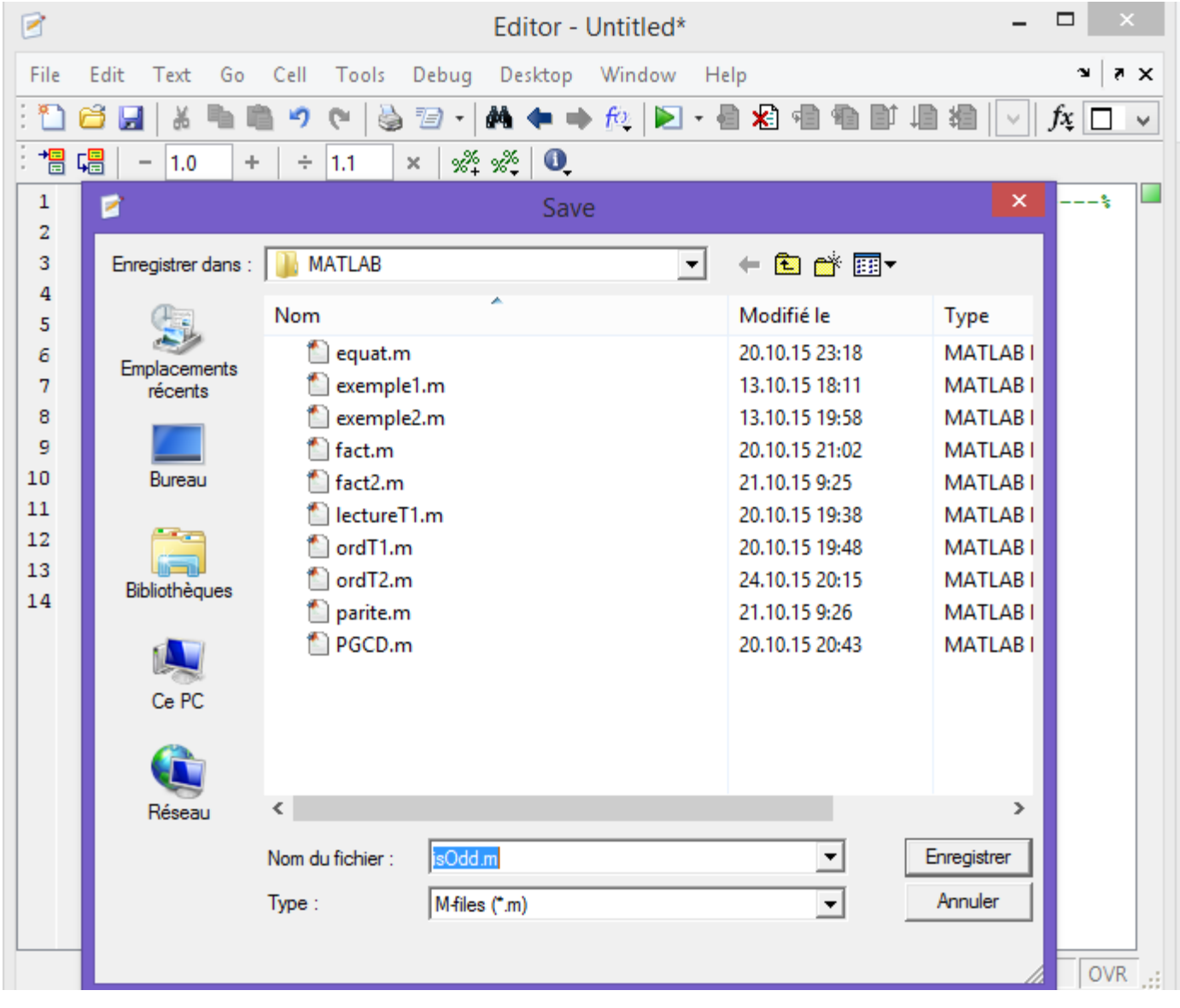
Attention, pour que Matlab reconnaisse une fonction et sache l'utiliser à partir de l'espace de travail, il est obligatoire que le nom du M-file soit identique au nom donné à la fonction

```
% — Fonction simple ——————————————————————————- %

% Fonction isOdd
% Cette fonction prend en argument un nombre et renvoie
% true si ce nombre est impair, false si non.
%
% Argument : un nombre
% Valeur de retour : un boolean
% Code placé dans isOdd.m

function p = isOdd(n)

if (mod(n,2) == 1)        % Si la division modulaire laisse un reste
  p = true;               % Renvoie vrai
else
  p = false;              % renvoie faux
end
```

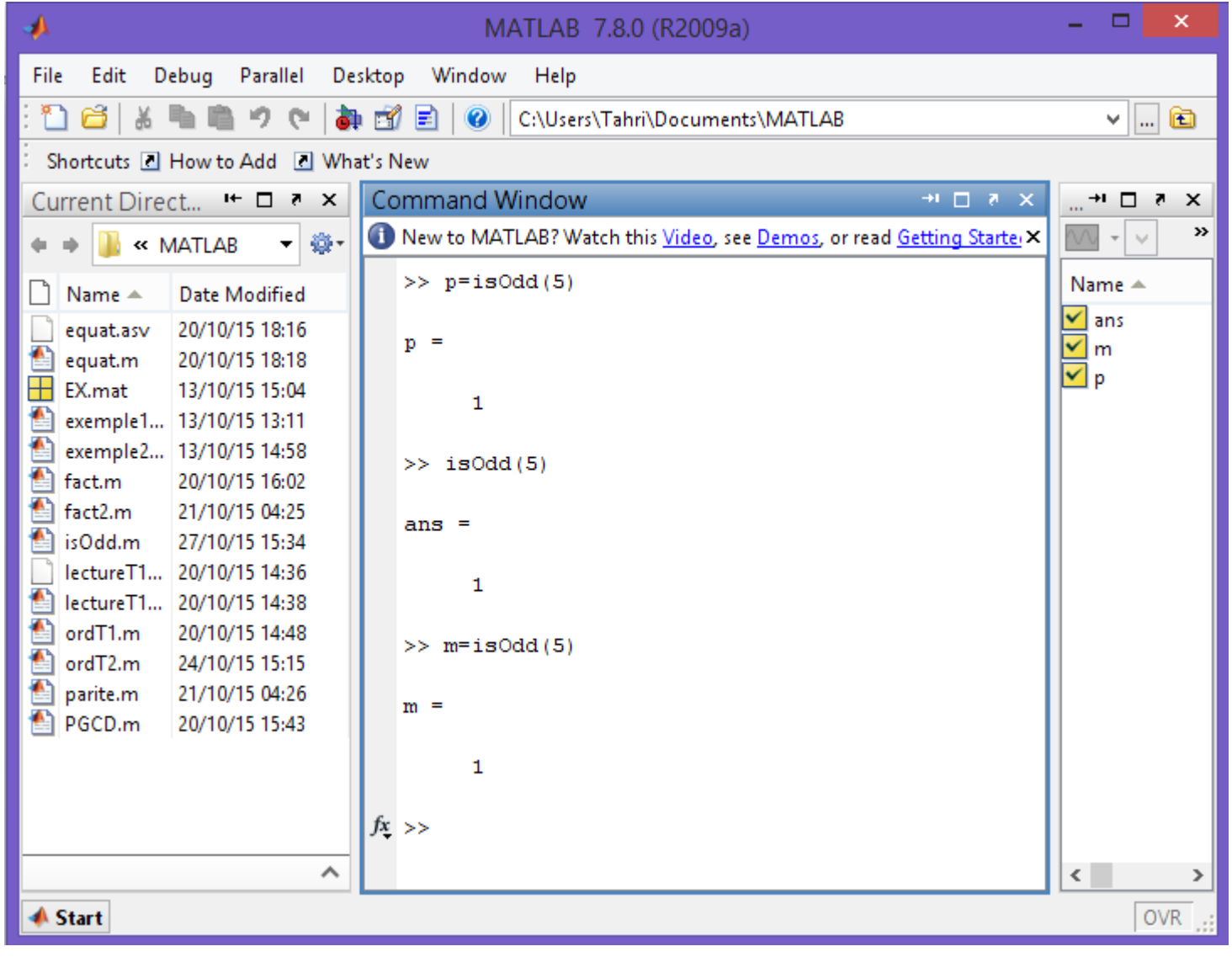
Cette fonction est écrite dans l'éditeur des M-files



Remarquez, le Matlab automatiquement donne au fichier m (M-file) le même nom que celui de la fonction.



L'appel de la fonction peut se faire dans le workspace.



1.2 Les fonctions récursives

Une fonction est dite récursive si elle s'appelle elle-même. La fonction suivante est récursive et permet de calculer la factoriel d'un nombre.

```
% — Fonction récursive ———————————————————————————— %

% Fonction fact
% Cette fonction prend en argument un entier positif
% et renvoie la factoriel correspondante.
%
% Argument : un entier positif
% Valeur de retour : un entier positif
%

function y = fact(n)

if (n <= 1)              % Si l'on est dans le cas de base
  y = 1;
else
  y = n * fact(n-1);     % Si non, appel récursif
end
```

```
>> f=fact(4)

f =

    24
```

1.3 Les fonctions primaires et sous fonctions

Un fichier de fonction peut contenir une fonction primaire qui apparait en premier et n'importe quel nombre d'autres fonctions optionnelles qu'on appellera des sous fonctions qui viennent après la fonction primaire et qui sont utilisées par cette dernière.

Les fonctions primaires peuvent être appeler de l'extérieur du fichier où elles sont définis (comme du workspace), mais les sous fonctions ne peuvent pas être appeler de l'extérieur du fichier de la fonction.
Les sous fonctions ne sont visibles qu'à l'intérieur du fichier de la fonction primaire.

```
function [x1,x2] = quadratic(a,b,c)
%this function returns the roots of
% a quadratic equation.
% It takes 3 input arguments
% which are the co-efficients of x2, x and the
%constant term
% It returns the roots
d = disc(a,b,c);
x1 = (-b + d) / (2*a);
x2 = (-b - d) / (2*a);
end % end of quadratic

function dis = disc(a,b,c)
%function calculates the discriminant
dis = sqrt(b^2 - 4*a*c);
end % end of sub-function
```

1.4 Les fonctions imbriquées

Une fonction imbriquée (nested function en anglais) est une fonction contenue à l'intérieur d'une autre (mère). Elle offre la possibilité de manipuler les variables déclarées dans la fonction mère, et d'appeler les mêmes fonctions qu'elle. On peut avoir autant de niveaux d'imbrication que l'on souhaite.

•Il faut veiller à rajouter le mot-clé **end** marquant la fin de toutes les fonctions contenues dans le fichier m.

•La définition d'une fonction imbriquée ne peut pas figurer au sein d'une instruction de contrôle. Ceci inclut les blocs **if/elseif/else**, **switch/case**, **for**,**while**, et **try/catch**.

•Une fonction imbriquée doit être appelée directement par son nom (pas d'utilisation de **feval**) ou par son handle obtenu grâce à l'**opérateur @** (pas d'utilisation de **str2func**).

•Toutes les variables d'une fonction imbriquée ainsi que celles dans laquelle elle est imbriquée doivent être définies explicitement. Aucun script ou fonction (ex. **load** appelée sans argument de sortie ou **assignin**) créant des variables qui n'existent pas déjà dans le workspace des fonctions ne peut être appelé.

Une fonction imbriquée peut être appelée depuis

• la fonction du niveau directement supérieur dans laquelle elle se trouve ;
• une autre fonction imbriquée du même niveau ;
• n'importe quelle fonction imbriquée à l'intérieur de celle-ci.

```
function [x1,x2] = quadratic2(a,b,c)
function disc  % nested function
d = sqrt(b^2 - 4*a*c);
end % end of function disc
disc;
x1 = (-b + d) / (2*a);
x2 = (-b - d) / (2*a);
end % end of function quadratic2
```

1.5 Les fonctions privées (private functions)

Une fonction privée est une fonction primaire qui est visible seulement à un groupe limité d'autres fonctions. Vous pouvez créer des fonctions privées dans un sous dossier avec un nom spécial **private,** ainsi ces fonctions ne sont visibles que par le fichier parent.

Exemple
Ecrivant une fonction *quadratic*. A ce moment la fonction *disc* qui calcule le déterminant, va être une fonction privée (private function).
Créant un sous dossier nommé private dans le dossier ou répertoire de travail. Sauvegardant la fonction suivante dans le fichier *disc.m*

```
function dis = disc(a,b,c)
%function calculates the discriminant
dis = sqrt(b^2 - 4*a*c);
end % end of sub-function
```

Creant maintenant la fonction quadratic3.m dans le répertoire de travail

```
function [x1,x2] = quadratic3(a,b,c)
%this function returns the roots of
% a quadratic equation.
% It takes 3 input arguments
% which are the co-efficient of x2, x and the
%constant term
% It returns the roots
d = disc(a,b,c);
x1 = (-b + d) / (2*a);
x2 = (-b - d) / (2*a);
end % end of quadratic3
```

Merci

pour

votre Attention