

INF1005A: Programmation procedurale

Chapitre 5: Résolution de problèmes, débogage et autres fonctions utiles

Agenda

- Résolution de problèmes
- Mesurer le temps
- Erreurs
- Débogage
- Outils MATLAB

Résolution de problèmes – étapes

- La définition du problème
- L'analyse du problème
- La conception des algorithmes
- La mise au point du programme MATLAB
- La vérification du programme

Résolution de problèmes – la définition du problème

Répondre à QUI? QUE? QUOI?

- Identifier les entités à manipuler
- Identifier les traitements à effectuer
- Identifier les résultats attendus

Résolution de problèmes – **l'analyse du problème**

Répondre à **COMMENT?**

- Inventer et réfléchir à diverses solutions possibles.
- Il faut s'attendre à envisager plusieurs possibilités avant de trouver la bonne.
- En programmation, il n'y a pas de solution unique. Choisir celle avec laquelle vous êtes le plus à l'aise. Pas nécessairement la plus performante ou la plus courte ou celle du professeur.
- Il n'y a pas de fonctions MATLAB magiques. Elles ont toutes des avantages et des inconvénients.
- On a chacun notre style de programmation.

Résolution de problèmes – **l'analyse du problème**

Voici trois solutions possibles pour le problème qui consiste à déterminer les nombres premiers compris entre 50 et 100.

1. Pour chaque nombre de 50 à 100 on vérifie tous les diviseurs possibles, c.à.d. de 2 à lui-même. Si on trouve une division sans reste, ce n'est pas un nombre premier.
2. Pour chaque nombre de 50 à 100 on vérifie les diviseurs possibles, mais du moment qu'une division est sans reste, on conclut toute suite que ce nombre n'est pas premier.
3. Pour chaque nombre de 50 à 100 on vérifie les diviseurs de deux jusqu'à la racine carrée de lui-même. Cependant si une division est sans reste, on conclut toute suite que ce nombre n'est pas premier.

Résolution de problèmes – la conception des algorithmes

- Écrire dans l'ordre les opérations déduites de l'analyse.
- Faire ressortir la structure de base
 - les structures de décision
 - les structures de répétition
- Utiliser adéquatement le pseudo-code schématique.

Résolution de problèmes – la mise au point du programme

- Faire en sorte que le programme fonctionne bien.
- Trois types d'erreur

- syntaxe:

i.e. `fro`

`if a=b`

- logique:

`a = 10;`

`while a < 20`

`a = a - 1;`

`end`

- de données:

`a = 10;`

`b = a(3) + 10;`

Résolution de problèmes – la vérification du programme

Pour s'assurer que son programme fonctionne bien et découvrir les erreurs de logique et de données.

- Vérifier le programme avec des données tests
 - Données usuelles;
 - Données de cas habituellement problématique: matrice vide, matrice ligne, matrice colonne, nombres très grands ou très petits, nombres positifs, nombres négatifs.
 - Données pour vérifier les conditions

`if a>10 & isqual(b, 'allo'),`

tester les combinaisons :

`a=13 et a=3, b='allo' et b='rien'.`

Agenda



Résolution de problèmes

- Mesurer le temps
- Erreurs
- Débogage
- Outils MATLAB

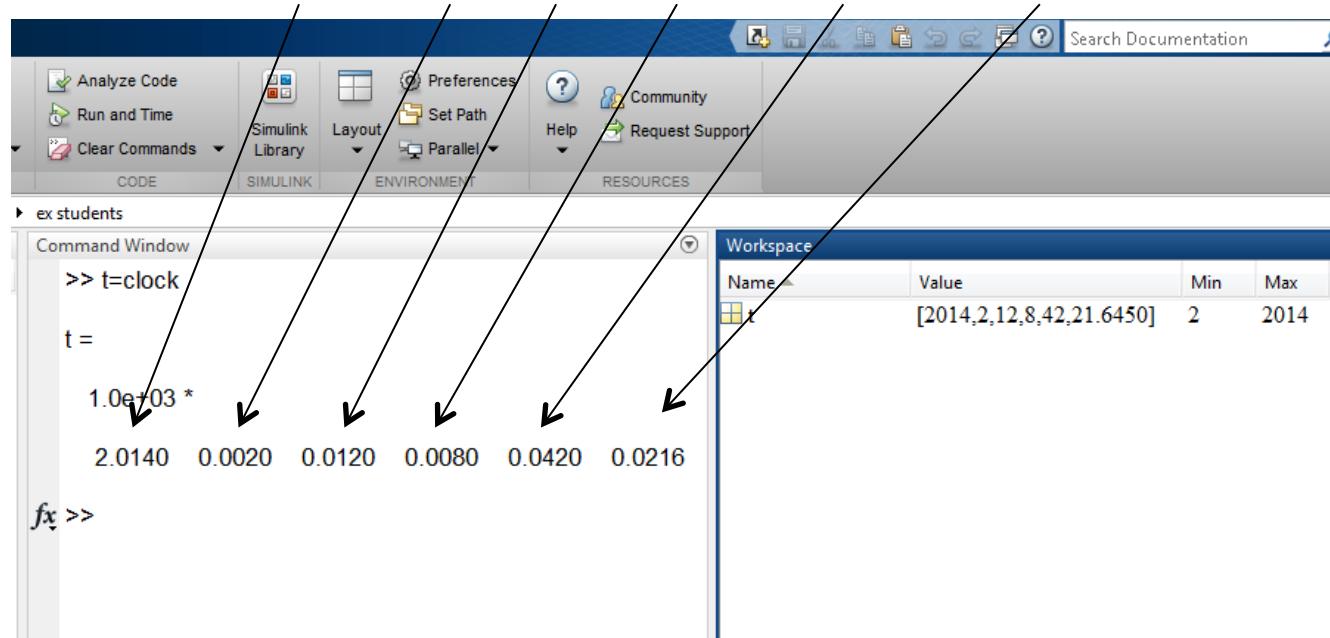
Mesurer le temps – `clock()`

MATLAB contient plusieurs fonctions permettant d'obtenir le temps.

`t = clock % ou t = clock()`

`t` : Variable contenant la matrice renvoyée par la fonction `clock`.

`t = [année mois jour heure minute seconde]`



Mesurer le temps – **etime()**

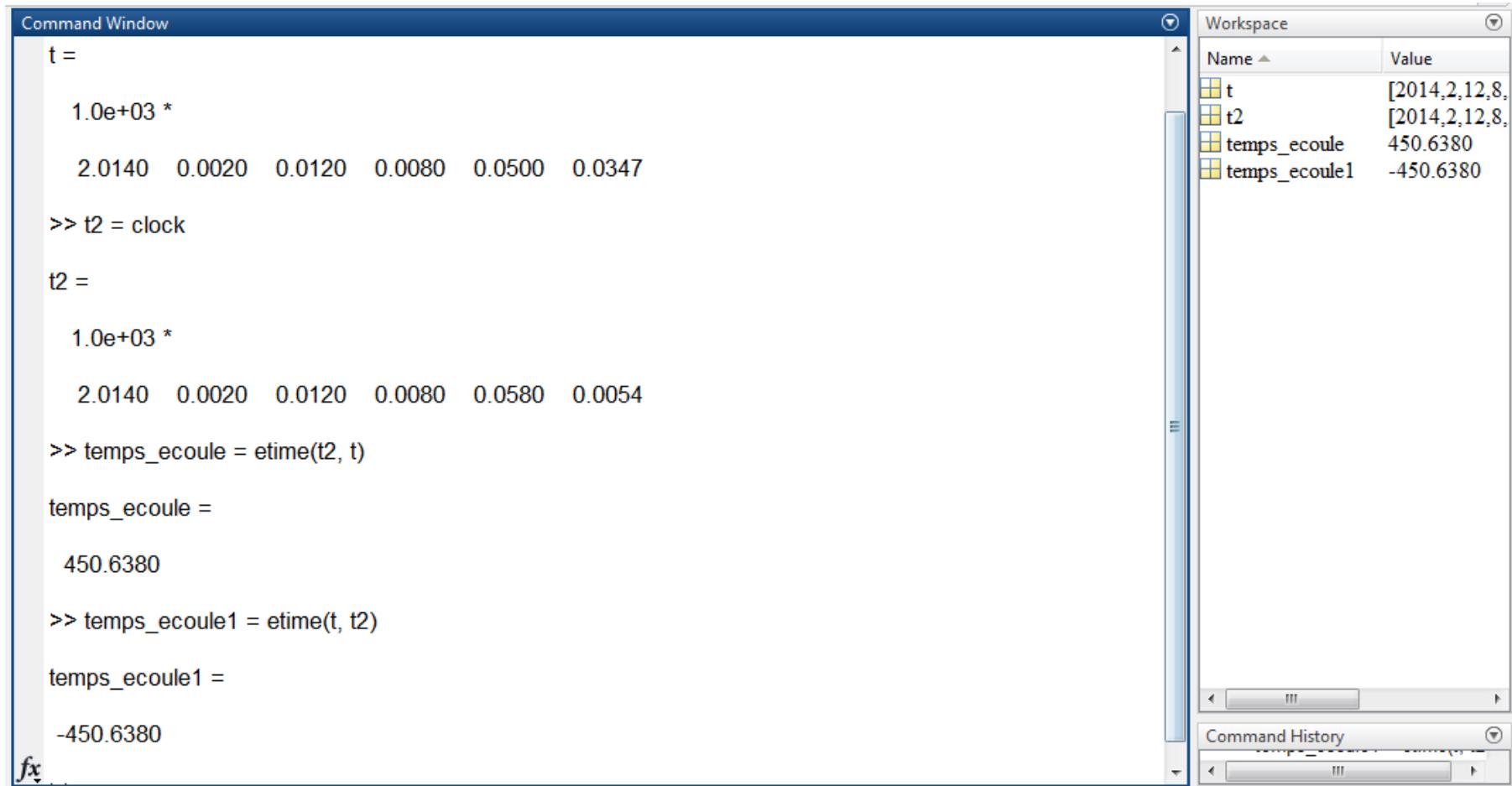
```
temps_ecoule = etime(t2, t1)
```

Fonction qui calcule le temps écoulé en secondes entre deux matrices obtenues avec `clock()`. Pour obtenir un temps écoulé positif, il faut mettre le deuxième temps mesuré en premier.

`t1, t2` : Matrices contenant les informations sur le temps.

`temps_ecoule` : Durée, en secondes, du temps écoulé entre `t1` et `t2`.

Mesurer le temps – **etime()**



The screenshot shows the MATLAB graphical user interface with the following components:

- Command Window:** Displays the following code and its execution results:

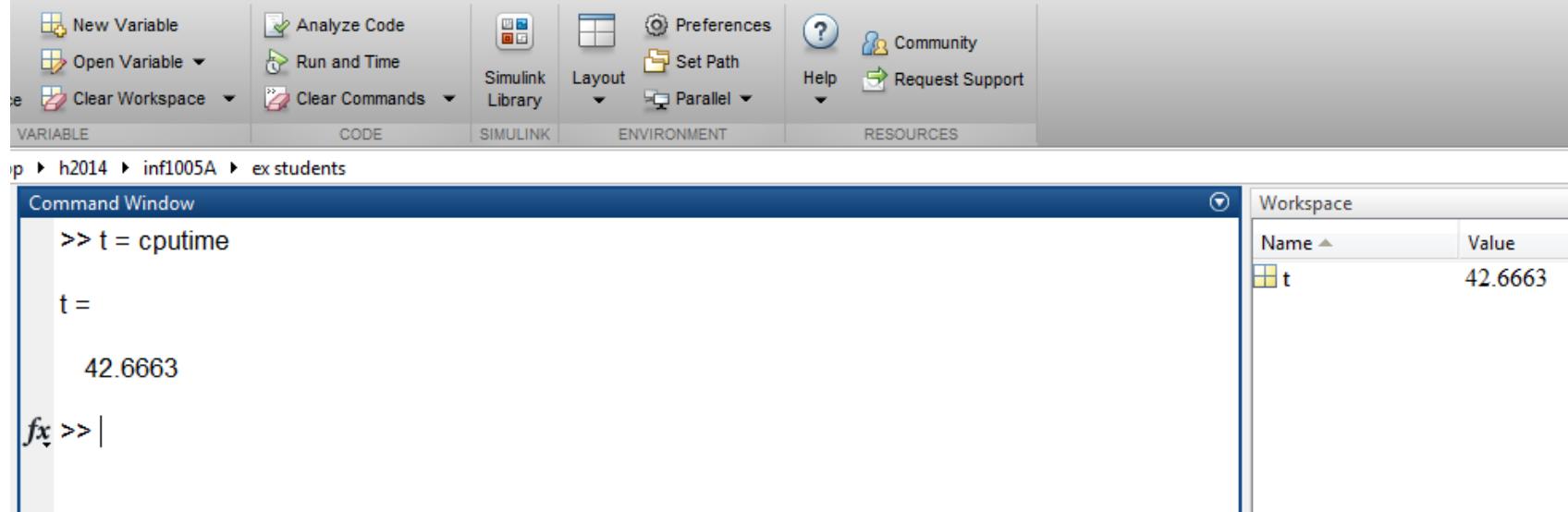
```
t =  
1.0e+03 *  
2.0140 0.0020 0.0120 0.0080 0.0500 0.0347  
  
>> t2 = clock  
  
t2 =  
1.0e+03 *  
2.0140 0.0020 0.0120 0.0080 0.0580 0.0054  
  
>> temps_ecoule = etime(t2, t)  
  
temps_ecoule =  
450.6380  
  
>> temps_ecoule1 = etime(t, t2)  
  
temps_ecoule1 =  
-450.6380
```
- Workspace:** Shows the variables and their values:

Name	Value
t	[2014,2,12,8,
t2	[2014,2,12,8,
temps_ecoule	450.6380
temps_ecoule1	-450.6380
- Command History:** Displays the command history at the bottom of the interface.

Mesurer le temps – `cputime()`

`temps = cputime % ou temps = cputime()`

temps : Variable qui contient le temps (en secondes) d'utilisation du UCT (CPU) par MATLAB depuis son démarrage.



Mesurer le temps – `tic()` / `toc()`

`tic % ou tic()`

`... % instructions`

`temps = toc % ou temps = toc()`

Fonctions qui calculent le temps écoulé entre les appels de `tic()` et `toc()`. `tic()` démarre le chronomètre et `toc()` affiche la valeur présente du chronomètre. On peut faire plusieurs `toc()` après un `tic()`.

`instructions` : instructions dont la durée totale est mesurée.

`temps` : variable qui contient la durée, en secondes, entre `tic` et `toc`.

Mesurer le temps – pause()

pause() ou % pause

Cette fonction sert à interrompre un programme jusqu'à ce que l'usager appuie sur une touche.

pause(durée_en_secondes)

Si un nombre est donné en paramètre, le programme est alors arrêté durant le nombre de secondes spécifiées.

i.e. pause(1.5);

Il est aussi possible d'activer ou de désactiver cette fonction:

pause('on'); %ou pause on

pause('off'); %ou pause off

Agenda



Résolution de problèmes



Mesurer le temps

- Erreurs

- Débogage

- Outils MATLAB

Erreurs – **error()**

error('message_d' erreur' , var₁, var₂, ...)

Fonction qui affiche un message d'erreur et arrête le programme.

Affiche le message en rouge.

message_d' erreur : message d'erreur désiré. Peut contenir des variables (comme fprintf()).

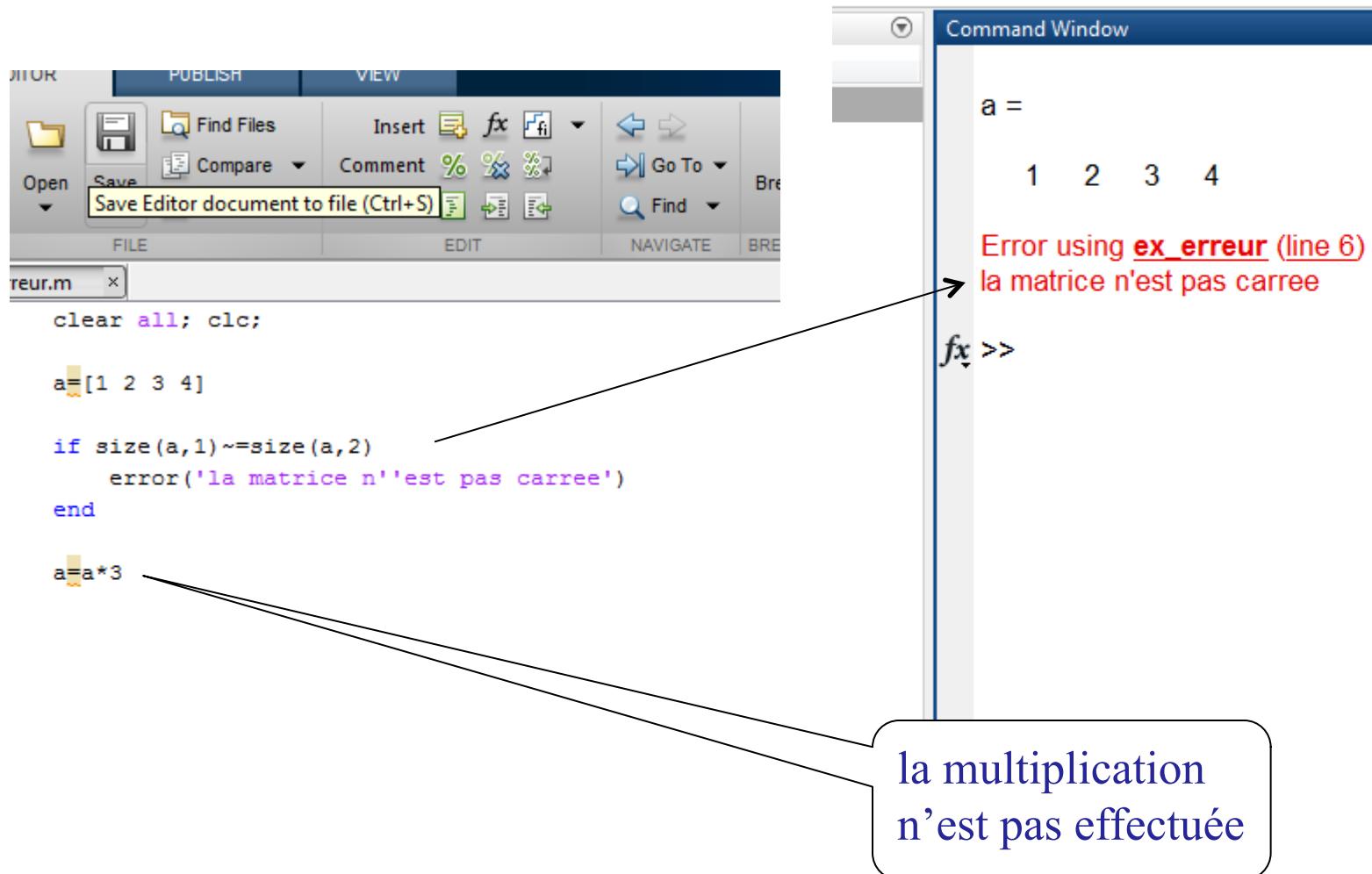
var_n : variables à afficher dans le message d'erreur.

[message, ID_message] = lasterr() %ou lasterr

message : message affiché lors de la dernière erreur, ou du dernier avertissement.

ID_message : mot significatif résumant le type d'erreur

Erreurs – `error()`



The screenshot shows the MATLAB IDE interface. On the left, the Editor window displays the script 'erreur.m' with the following code:

```

EDITOR      PUBLISH      VIEW
FILE        EDIT          NAVIGATE
Open       Save           Find Files
Save Editor document to file (Ctrl+S) Compare Comment Go To
Find
erreur.m x
clear all; clc;
a=[1 2 3 4]
if size(a,1)~=size(a,2)
    error('la matrice n''est pas carree')
end
a=a*3

```

A callout bubble points to the line `a=a*3` with the text "la multiplication n'est pas effectuée".

The Command Window on the right shows the execution of the script. It displays the variable `a` as a row vector [1 2 3 4] and then triggers an error message:

```

Command Window
a =
1 2 3 4
Error using ex_erreur (line 6)
la matrice n'est pas carree
fx >>

```

Erreurs – **error()**

```
Command Window

a =
1 2 3 4
Error using ex_erreur (line 6)
la matrice n'est pas carree

fx >>
```

la dernière erreur

```
Command Window

a =
1 2 3 4
Error using ex_erreur (line 6)
la matrice n'est pas carree
>> lasterr
ans =
Error using ex_erreur (line 6)
la matrice n'est pas carree
fx >> |
```

Erreurs – **warning()**

```
warning ('message_d'erreur', var1, var2, ...)
```

Fonction qui affiche un message d'avertissement.

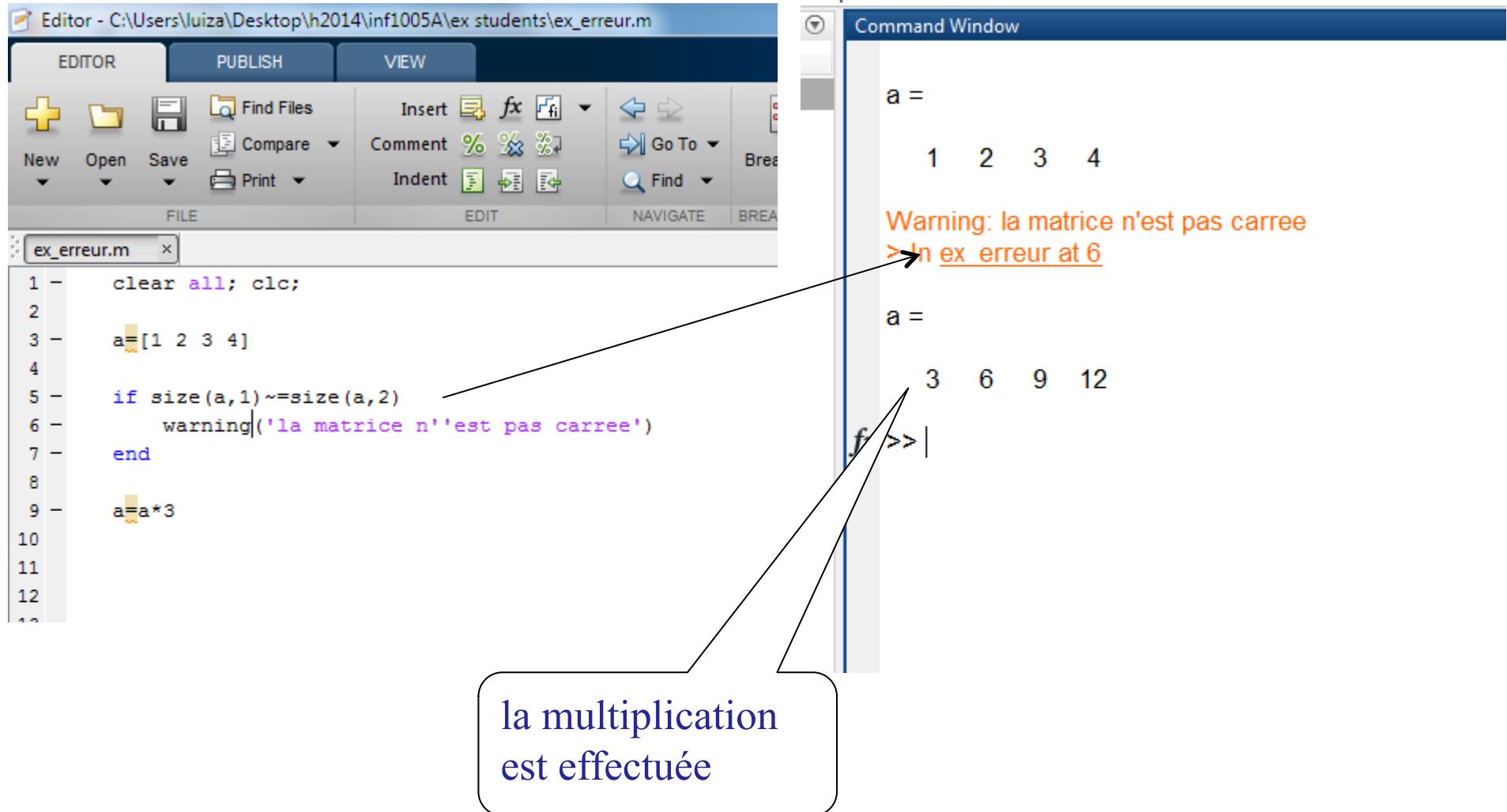
N'arrête pas le programme.

```
[message, ID_message] = lastwarn() %ou lastwarn
```

message : Message affiché lors de la ~~dernière erreur~~, ou du dernier avertissement.

ID_message : Mot significatif résumant le type d'erreur ou d'avertissement.

Erreurs – `error()`



The screenshot shows the MATLAB environment with the Editor window open and the Command Window active.

Editor - C:\Users\luiza\Desktop\h2014\inf1005A\ex students\ex_erreur.m

```

1 - clear all; clc;
2
3 - a=[1 2 3 4]
4
5 - if size(a,1)~=size(a,2)
6 -     warning('la matrice n''est pas carree')
7 - end
8
9 - a=a*3
10
11
12
13

```

Command Window

```

a =
1 2 3 4

Warning: la matrice n'est pas carree
>> In ex_erreur at 6

a =
3 6 9 12
f>>

```

A callout bubble points from the text "la multiplication est effectuée" to the line `a=a*3` in the Editor.

la multiplication
est effectuée

Agenda



Résolution de problèmes



Mesurer le temps



Erreurs

- Débogage

- Outils MATLAB

Le débogage

- Le débogage est le nom de l'action qui **consiste à trouver les erreurs dans un programme.**
- La première étape pour détecter une erreur, est de la **localiser.**
- Plusieurs méthodes sont possibles pour y arriver, l'une d'elle consiste à **enlever les points virgules à la fin de certaines lignes** afin d'observer l'affichage produit par les **différentes opérations. (Très efficace)**

Le débogage

```
%*****  
%* NOM DU FICHIER : ex_debug.m  
%* DESCRIPTION : Programme qui calcule la moyenne d'une serie de 5 nombres entiers.  
%* AUTEUR : Francois Morin  
%* DATE DE CRÉATION  
%* MISES À JOUR : programme à corriger  
%*****
```

```
%Demande des donnees a l'usager  
donnee_1 = input('Entrer un premier entier ');\ndonnees_2 = input('Entrer un deuxieme entier ');\ndonnee_3 = input('Entrer un troisieme entier ');\ndonnee_4 = input('Entrer un quatrieme entier ');\ndonnee_5 = input('Entrer un dernier entier ');\n\n%Calcul de la moyenne  
moyenne = (donnee_1 + donnee_2 + donnee_3 + ...  
    donnee_4 + donnee_5) \ 5;  
  
%Affichage des resultats  
fprintf('La moyenne des entiers est %f \n',moyenne);
```

exécution du programme

Command Window

```
>> ex_debug  
Entrer un premier entier 2  
Entrer un deuxieme entier 6  
Entrer un troisieme entier 7  
Entrer un quatrieme entier 3  
Entrer un dernier entier 7  
La moyenne des entiers est 0.200000  
>>
```

réponse erronée

Le débogage

enlever les points virgules pour voir si les nombres sont bien entrés

```
%Demande des donnees a l'usager
donnee_1 = input('Entrer un premier entier ')
donnee_2 = input('Entrer un deuxième entier ')
donnee_3 = input('Entrer un troisième entier ')
donnee_4 = input('Entrer un quatrième entier ')
donnee_5 = input('Entrer un dernier entier ')
```

les nombres sont bien mémorisés

Command Window

```
>> ex_debug
Entrer un premier entier 2

donnee_1 =
2

Entrer un deuxième entier 6

donnee_2 =
6

Entrer un troisième entier 7

donnee_3 =
7

Entrer un quatrième entier 3

donnee_4 =
3
```

Le débogage

```
%Demande des données à l'usager
donnee_1 = input('Entrer un premier entier ');
donnee_2 = input('Entrer un deuxième entier ');
donnee_3 = input('Entrer un troisième entier ');
donnee_4 = input('Entrer un quatrième entier ');
donnee_5 = input('Entrer un dernier entier ');

%Calcul de la moyenne
moyenne = (donnee_1 + donnee_2 + donnee_3 + ...
           donnee_4 + donnee_5) / 5
```

```
>> ex_debug
Entrer un premier entier 2
Entrer un deuxième entier 6
Entrer un troisième entier 7
Entrer un quatrième entier 3
Entrer un dernier entier 7
```

```
moyenne =
0.2000
```

calcul erroné

on remet les points virgules

vérification du calcul en
enlevant le point-virgule

```
%Calcul de la moyenne
moyenne = (donnee_1 + donnee_2 + donnee_3 + ...
           donnee_4 + donnee_5) / 5;
```

correction de
l'erreur

La moyenne des entiers est 0.200000

Le débogage – points d'interruption (breakpoints)

Une autre façon de trouver des erreurs est d'utiliser le débogueur de MATLAB.

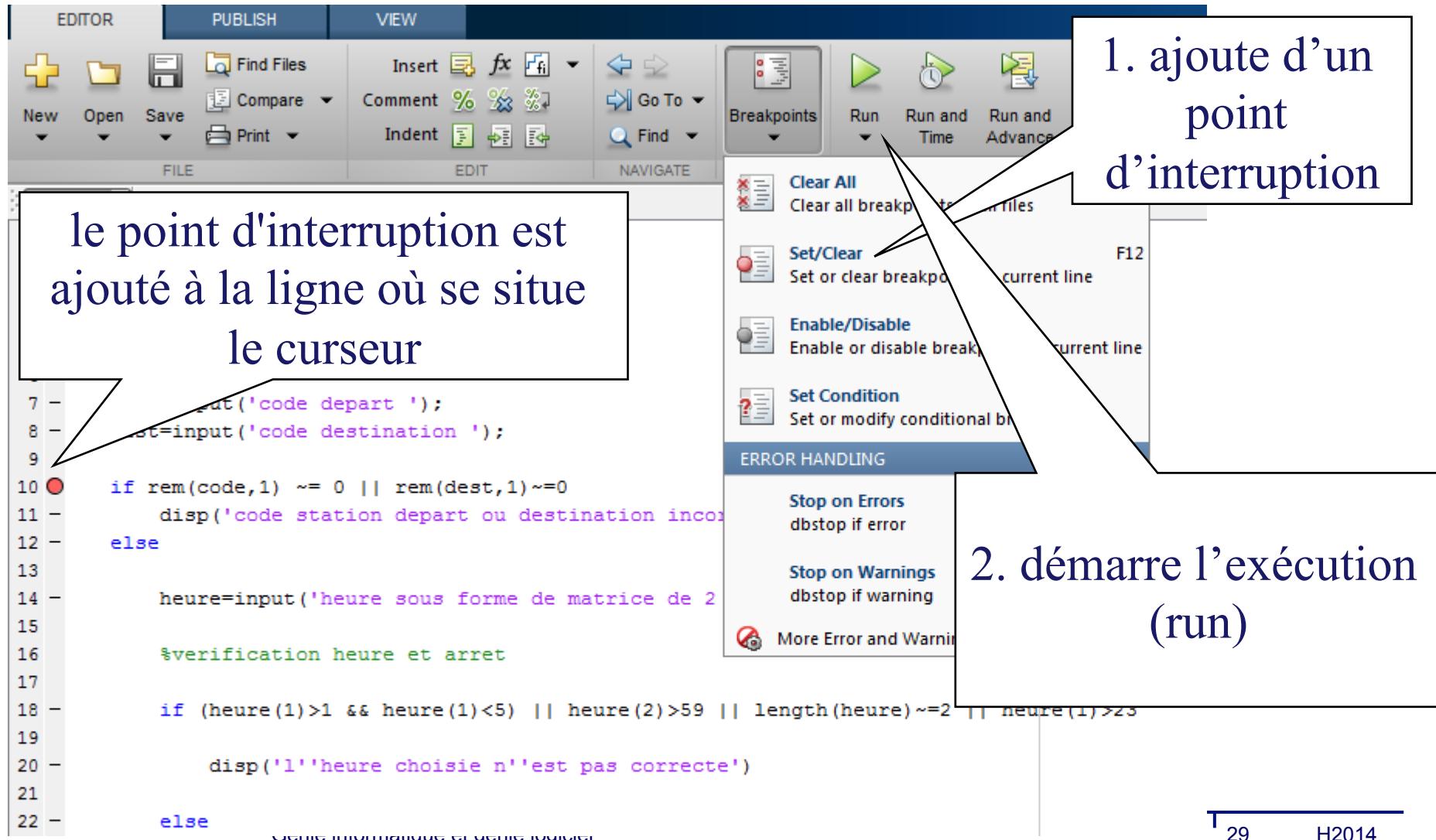
Permet d'exécuter ligne par ligne les opérations d'un programme.

Ceci permet d'accélérer la recherche d'erreurs.

Par exemple, si plusieurs lignes de code sont exécutées avant l'erreur, il est alors possible d'inclure des points d'interruption dans le programme.

Le programme s'exécutera jusqu'au point d'interruption.

Le débogage – points d'interruption (breakpoints)



The screenshot shows the MATLAB IDE interface. On the left is the code editor with the following MATLAB script:

```

1 put('code depart ');
2 st=input('code destination ');
3
4 if rem(code,1) ~= 0 || rem(dest,1)~=0
5     disp('code station depart ou destination incorrect')
6 else
7
8     heure=input('heure sous forme de matrice de 2x2');
9
10    %verification heure et arret
11
12    if (heure(1)>1 && heure(1)<5) || heure(2)>59 || length(heure) ~=2 || heure(1)>23
13        disp('l''heure choisie n''est pas correcte')
14    else
15        %code informatique et genie logiciel
16    end
17
18 end
19
20 %fin de la fonction
21
22

```

A callout box on the left side of the code editor contains the text: "le point d'interruption est ajouté à la ligne où se situe le curseur".

A context menu is open over line 10, specifically the line containing the if statement. The menu items are:

- Clear All
- Clear all breakpoints in current files
- Set/Clear** (highlighted with a red circle)
- Set or clear breakpoint on current line
- Enable/Disable
- Enable or disable breakpoint on current line
- Set Condition
- Set or modify conditional breakpoint
- ERROR HANDLING
- Stop on Errors
- dbstop if error
- Stop on Warnings
- dbstop if warning
- More Error and Warning Options

Two large callout boxes are positioned to the right of the menu. The top one, titled "1. ajoute d'un point d'interruption", points to the "Set/Clear" option. The bottom one, titled "2. démarre l'exécution (run)", points to the "Run" button in the toolbar.

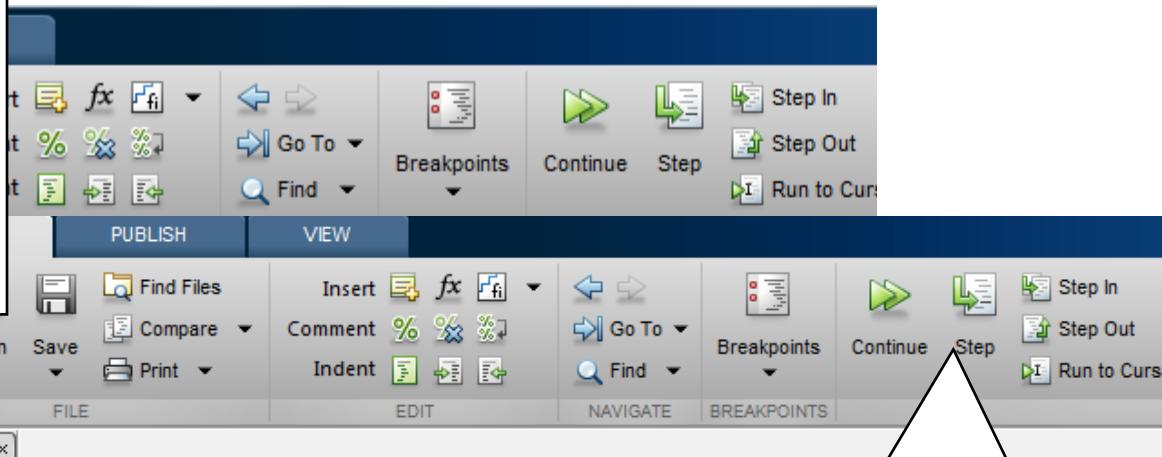
Le débogage – points d'interruption (breakpoints)

3. l'exécution qui s'arrête au point d'interruption (point rouge)

```
1 - all; clc
2
3
4 - ad bus;
5 - tm=bus;
6
7 - code=input('code ');
8 - dest=input('code ');
9
10 - if rem(code,1) ~= 0 || rem(dest,1) ~= 0
11 -     disp('code station depart ou destination incorrecte')
12 - else
13
14 -     heure=input('heure ');
15
16 -     %verification
17
18 - end
```

bus.m

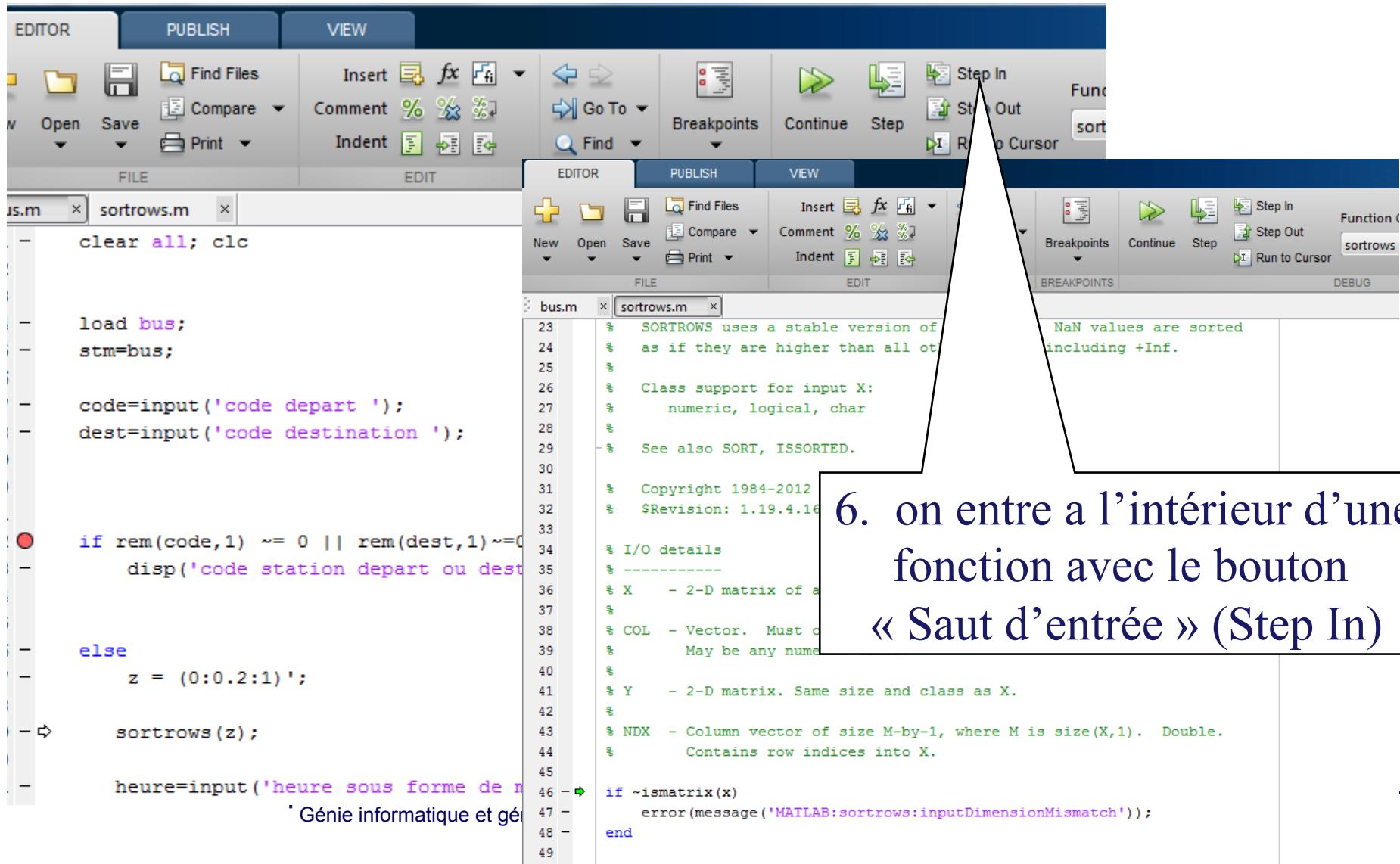
```
1 - clear all; clc
2
3
4 - load bus;
5 - stm=bus;
6
7 - code=input('code depart ');
8 - dest=input('code destination ');
9
10 - if rem(code,1) ~= 0 || rem(dest,1) ~= 0
11 -     disp('code station depart ou destination incorrecte')
12 - else
13
14 -     heure=input('heure sous forme de matrice de dimension n x n ');
15
```



4. avec le bouton
"Pas" (step)

5. on exécute ligne par ligne

Le débogage – points d'interruption (breakpoints)



The screenshot shows the MATLAB IDE interface with two files open:

- bus.m** (left pane): Contains code for calculating station coordinates based on input codes.
- sortrows.m** (right pane): Contains documentation and implementation for the `sortrows` function.

A red dot is placed on line 46 of the `sortrows.m` code, indicating a breakpoint. The code for `sortrows.m` is as follows:

```

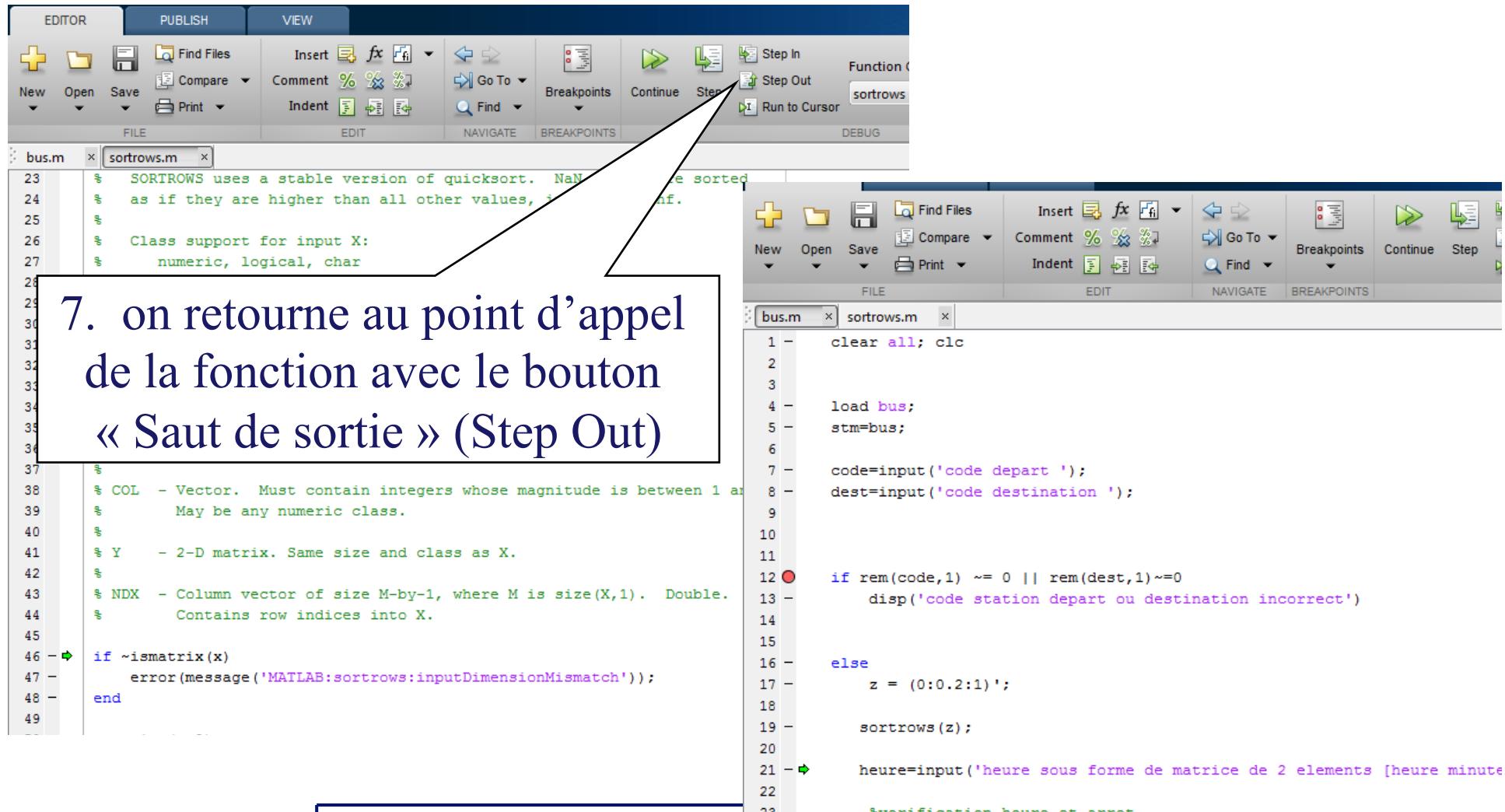
23 % SORTROWS uses a stable version of
24 % as if they are higher than all other
25 %
26 % Class support for input X:
27 % numeric, logical, char
28 %
29 % See also SORT, ISSORTED.
30 %
31 % Copyright 1984-2012
32 % $Revision: 1.19.4.16 $
33 %
34 % I/O details
35 % -----
36 % X - 2-D matrix of a
37 %
38 % COL - Vector. Must c
39 % May be any nume
40 %
41 % Y - 2-D matrix. Same size and class as X.
42 %
43 % NDX - Column vector of size M-by-1, where M is size(X,1). Double.
44 % Contains row indices into X.
45 %
46 if ~ismatrix(x)
47 error(message('MATLAB:sortrows:inputDimensionMismatch'));
48 end
49

```

A callout box with the number 6 points to the "Step In" button in the toolbar, which is highlighted in green. The text inside the box reads:

6. on entre à l'intérieur d'une fonction avec le bouton « Saut d'entrée » (Step In)

Le débogage – points d'interruption (breakpoints)



7. on retourne au point d'appel de la fonction avec le bouton « Saut de sortie » (Step Out)

```

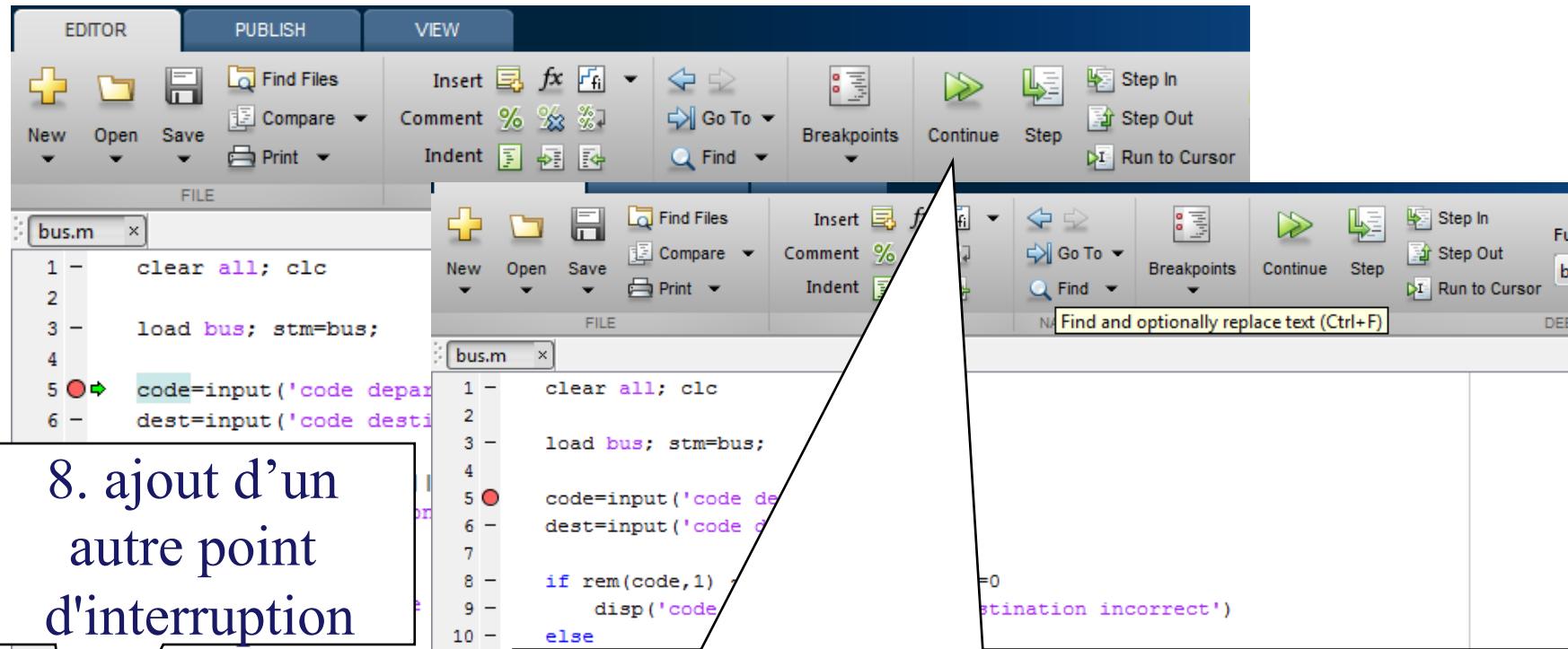
bus.m x sortrows.m x
23 % SORTROWS uses a stable version of quicksort. NaN
24 % as if they are higher than all other values, i.e., infinity.
25 %
26 % Class support for input X:
27 %     numeric, logical, char
28 %
29 %
30 %
31 %
32 %
33 %
34 %
35 %
36 %
37 %
38 % COL - Vector. Must contain integers whose magnitude is between 1 and
39 %       May be any numeric class.
40 %
41 % Y - 2-D matrix. Same size and class as X.
42 %
43 % NDX - Column vector of size M-by-1, where M is size(X,1). Double.
44 %       Contains row indices into X.
45 %
46 -> if ~ismatrix(x)
47 -     error(message('MATLAB:sortrows:inputDimensionMismatch'));
48 -end
49 
```

```

bus.m x sortrows.m x
1 - clear all; clc
2 -
3 -
4 - load bus;
5 - stm=bus;
6 -
7 - code=input('code depart ');
8 - dest=input('code destination ');
9 -
10 -
11 -
12 -> if rem(code,1) ~= 0 || rem(dest,1)~=0
13 -     disp('code station depart ou destination incorrect')
14 -
15 -
16 - else
17 -     z = (0:0.2:1)';
18 -
19 -     sortrows(z);
20 -
21 -> heure=input('heure sous forme de matrice de 2 elements [heure minute
22 -
23 -     #verification heure et arret

```

Le débogage – points d'interruption (breakpoints)



The screenshot shows the MATLAB IDE interface with three windows. The top window displays the toolbar with various tools like New, Open, Save, Insert, Comment, Breakpoints, Continue, Step, and Run to Cursor. Below the toolbar, the code editor shows a script named 'bus.m' with several lines of MATLAB code. A red circular breakpoint marker is visible on line 5. The middle window shows the same code with the cursor on line 5, indicating it is currently executing or has just executed that line. The bottom window shows the code again, but now the cursor is on line 20, indicating the program has moved past the breakpoint.

8. ajout d'un autre point d'interruption

9. le bouton « Continuer » (Continue) permet l'exécution de toutes les lignes de code jusqu'au prochaine point d'interruption

```

bus.m
1 - clear all; clc
2 -
3 - load bus; stm=bus;
4 -
5 - code=input('code depart');
6 - dest=input('code destination');
7 -
8 - if rem(code,1) ~= 0
9 -     disp('code incorrect')
10 - else
11 -     %sauvegarde dans une matrice les temps entre stations
12 -     temp=[];
13 -     for t=2:size(bus,1)
14 -         temp=t;
15 -         bus(t,:)=temp;
16 -     end
17 -     %sauvegarde dans une matrice les temps entre stations
18 -     temp=[];
19 -     for t=2:size(bus,1)
20 -         temp=t;
21 -         bus(t,:)=temp;
22 -     end
23 - end
    
```

Génie informatique et génie logiciel : size(bus) = 2 * size(bus, 1)

Agenda



Résolution de problèmes



Mesurer le temps



Erreurs



Débogage

- Outils MATLAB

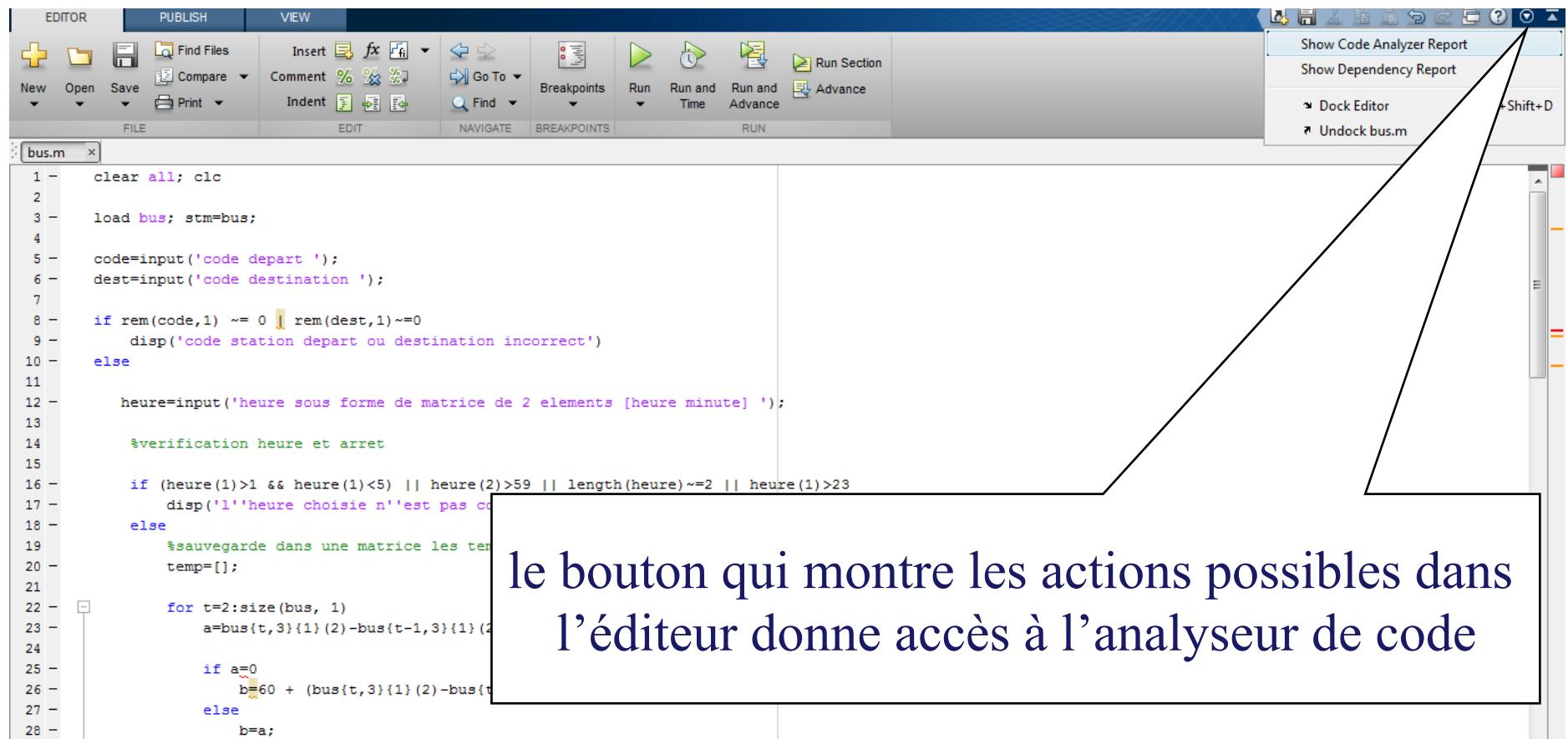
Outils MATLAB – **Code analyzer**

L'analyseur de code (Code analyzer) est un outil qui permet de trouver des erreurs ou des optimisations pour les programmes composés de fichier M.

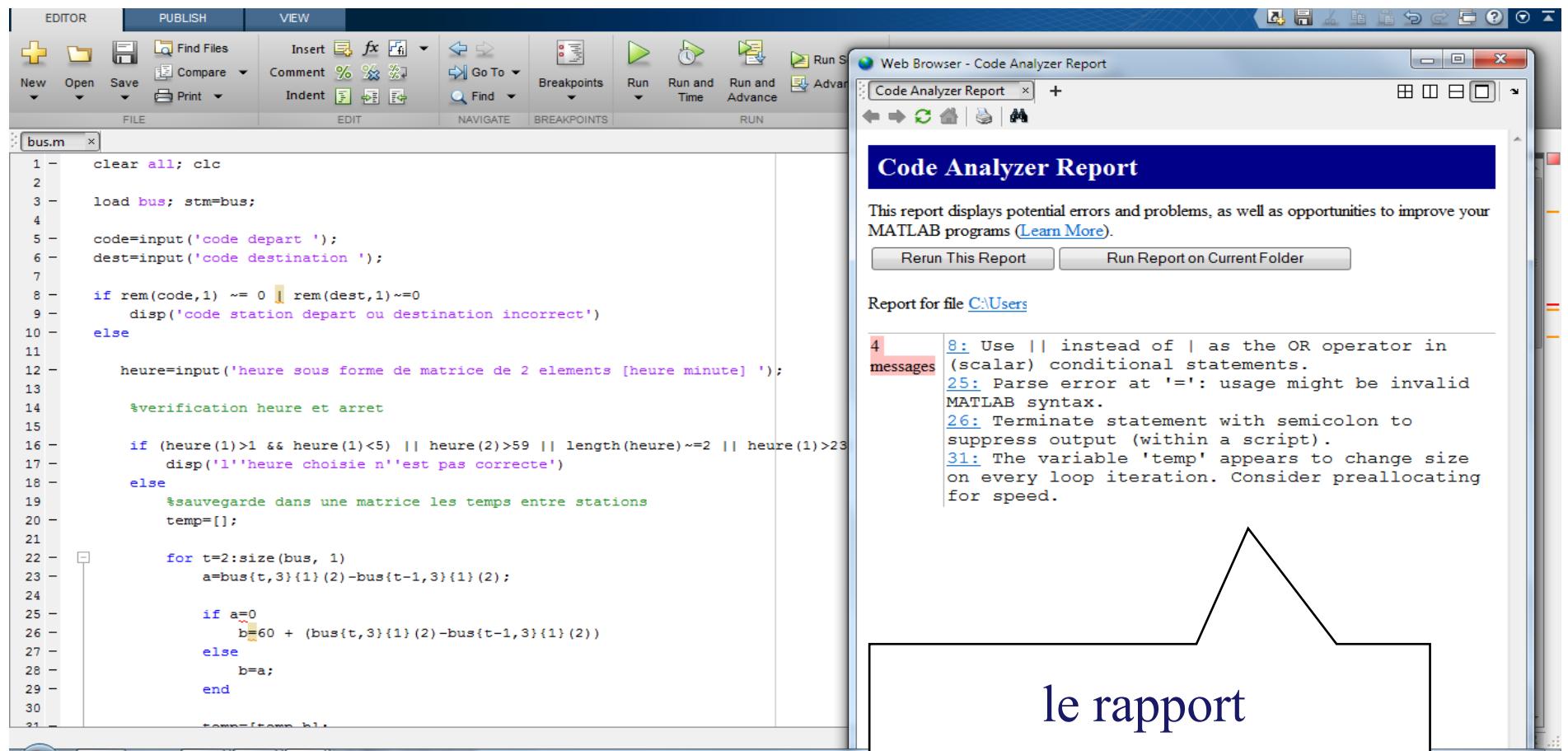
Est un outil automatisé donc il est possible que les erreurs qu'il détecte, n'en soient pas. (fonctionne un peu comme le correcteur de word et avertit des erreurs possibles).

S'il affiche une erreur qui n'en n'est pas une, il est possible de lui indiquer de ne pas en tenir compte à l'avenir en ajoutant `%#ok` à la fin de la ligne concernée dans le fichier M.

Outils MATLAB – Code analyzer



Outils MATLAB – Code analyzer



The screenshot shows the MATLAB IDE interface. On the left, the MATLAB Editor displays the script file `bus.m`. The code performs various operations such as loading data, prompting for station codes, validating times, and calculating travel times between stations. On the right, a separate window titled "Web Browser - Code Analyzer Report" shows the results of the analysis. The report header states: "This report displays potential errors and problems, as well as opportunities to improve your MATLAB programs (Learn More)". It includes buttons for "Rerun This Report" and "Run Report on Current Folder". The report details four messages:

- 4 messages**
- 8:** Use `||` instead of `|` as the OR operator in (scalar) conditional statements.
- 25:** Parse error at `'='`: usage might be invalid MATLAB syntax.
- 26:** Terminate statement with semicolon to suppress output (within a script).
- 31:** The variable 'temp' appears to change size on every loop iteration. Consider preallocating for speed.

A large, stylized arrow graphic points downwards from the report area towards the text "le rapport" at the bottom.

Agenda



Résolution de problèmes



Mesurer le temps



Erreurs



Débogage



Outils MATLAB

Sommaire

- 1 Résolution de problèmes
- 2 Mesurer le temps
- 3 Erreurs
- 4 Débogage
- 5 Outils MATLAB