Programming with Matlab

Day 5: Debugging, efficiency, advanced types of variables, logical indices, images

Debugging: keyboard (I)

Editor

```
n_elems=length(vector);
s=0;
for c_elems=1:n_elems
   s=s+vector(c_elems);
   keyboard
end
```

Program stops in the point where we inserted the keyboard command.

We get control at command window.

Command Window

```
>> vec=[1 2 3];
>> sumador(vec)
K>> c_elems
c_elems =
    1
```

We now have access to the function's workspace, in the state it is at that moment

Debugging: keyboard (II)

Editor

```
n_elems=length(vector);
s=0;
for c_elems=1:n_elems
    s=s+vector(c_elems);
    keyboard
end
```

Command Window

```
>> sumador(vec)
K>> return
K>> c_elems
c_elems =
     2
```

In our example, execution continues one more iteration, until we again encounter the keyboard command

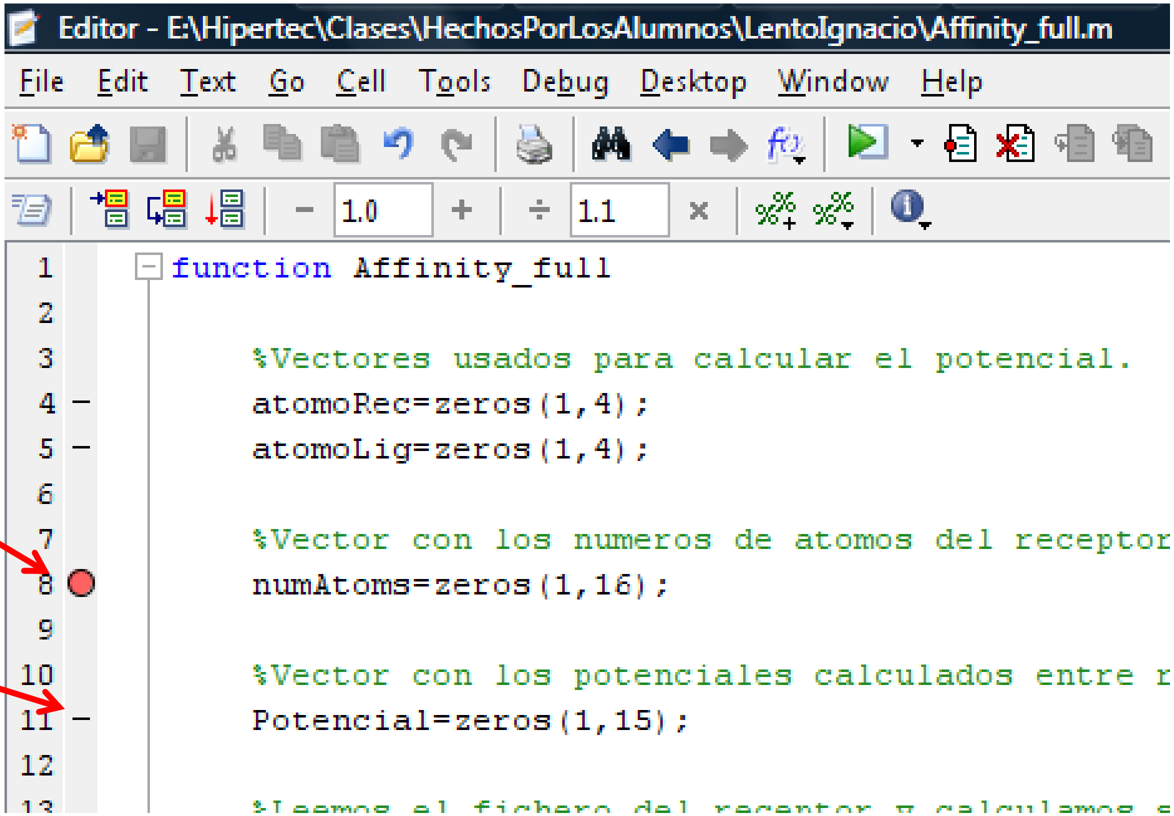
return command continues execution of the program, which runs until the next keyboard command (or until the end)

Command Window

```
>> sumador(vec)
K>> dbquit
>>
```

dbquit cancels execution of the program. We go back to our usual command window, with the base workspace.

Debugging: Red dots



A red dot will appear if you click on one of these lines.

They work exaclty the same way as keyboard

Debugging: F5, F10

- They are used when execution is stopped by a keyboard command or by a red dot.
- F5: Continues execution until the next keyboard/red dot (the same as return command)
- F10: Executes only the next line of the program.

Debugging: try...catch

Editor

```
function c=errores

c=1;
caca
```

Command Window

```
>> errores
??? Undefined function or variable 'caca'.

>>
```

Editor

```
function c=errores2

c=1;
try
   caca
catch
   disp(‘Eres idiota’)
   keyboard
end
```

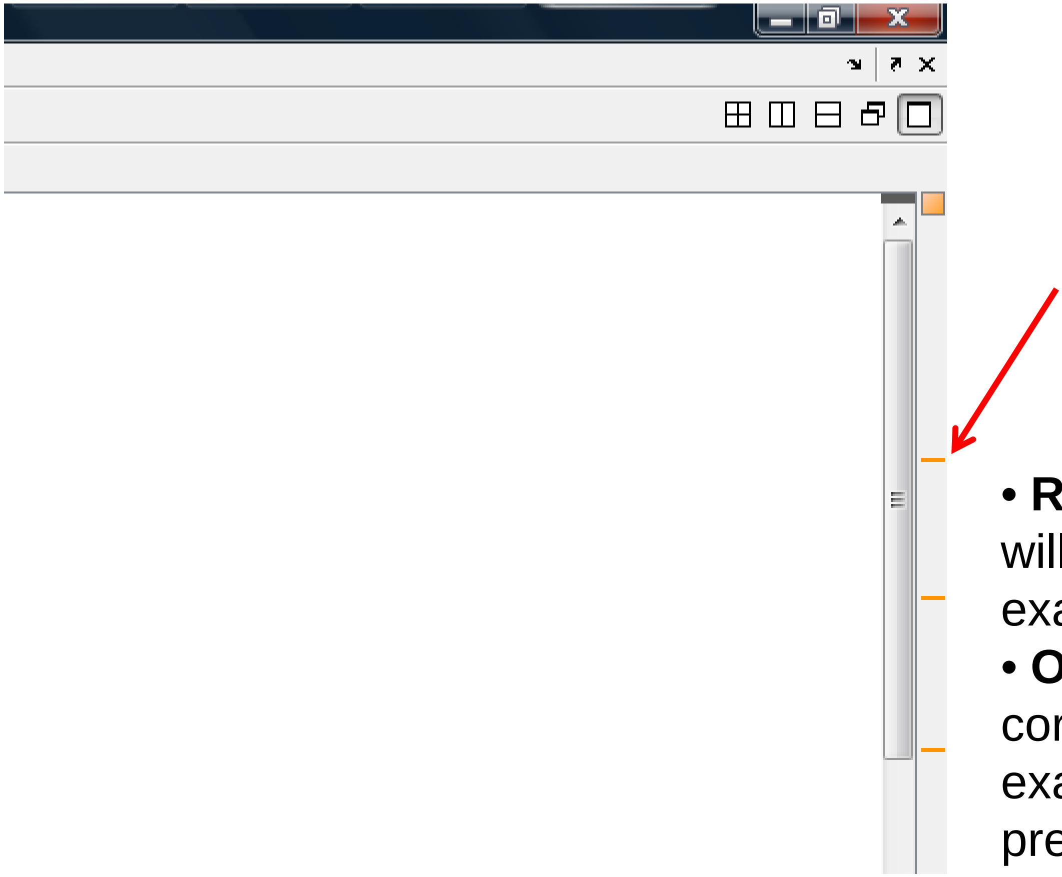
Matlab tries to execute the code in the “try” section. If the code works properly, the “catch” section does not execute.

If there is an error in the “try” section, instead of aborting execution, matlab executes the “catch” section.

Command Window

```
>> errores
Eres idiota
K>>
```

Debugging/efficiency: Those small orange and red lines



Each line indicates that Matlab found something in the code that needs improvement

- **Red lines:** Errors. The program will not execute correctly (for example, a parenthesis is missing).
- **Orange lines:** Will execute correctly, but it might be better (for example, variables that should be pre-allocated for speed).

Efficiency: tic…toc

- Timer starts, when tic command is executed.
- When toc command is executed, we get the time elapsed since the tic.

```
>> tic
a=0.1;
toc
Elapsed time is 0.005019 seconds.
```

Efficiency: Preallocate variables

```
>> tic
for c=1:50000
vec(c)=c;
end
toc
Elapsed time is 3.196955 seconds.
```

vec becomes one element longer each iteration. Therefore, Matlab must reserve an extra bit of memory each iteration. This takes A LOT of time.

```
>> tic
vec=zeros(1,50000);
for c=1:50000
vec(c)=c;
end
toc
Elapsed time is 0.060863 seconds.
```

All the memory we are going to need is reserved at the beginning.

50 times faster!

Efficiency: Matrices, matrices, matrices!

- Matlab’s loops are relatively slow
- But Matlab works very fast with matrices

```
>> x=0:.0001:10;
>> exponencial=zeros(1,100001);

>> tic
for c=1:100001
exponencial(c)=exp(x(c));
end
toc
Elapsed time is 0.233338 seconds.
```

```
>> tic
exponencial=exp(x);
toc
Elapsed time is 0.013351 seconds.
```

20 times faster!

A useful instruction: repmat

- repmat creates a matrix which is just a vector repeated several times.
- Exercise tabla5 in two lines:

```
>> mat=repmat(1:5,5,1)

mat =

     1     2     3     4     5
     1     2     3     4     5
     1     2     3     4     5
     1     2     3     4     5
     1     2     3     4     5

>> tabla5=mat .* mat'

tabla5 =

     1     2     3     4     5
     2     4     6     8    10
     3     6     9    12    15
     4     8    12    16    20
     5    10    15    20    25
```

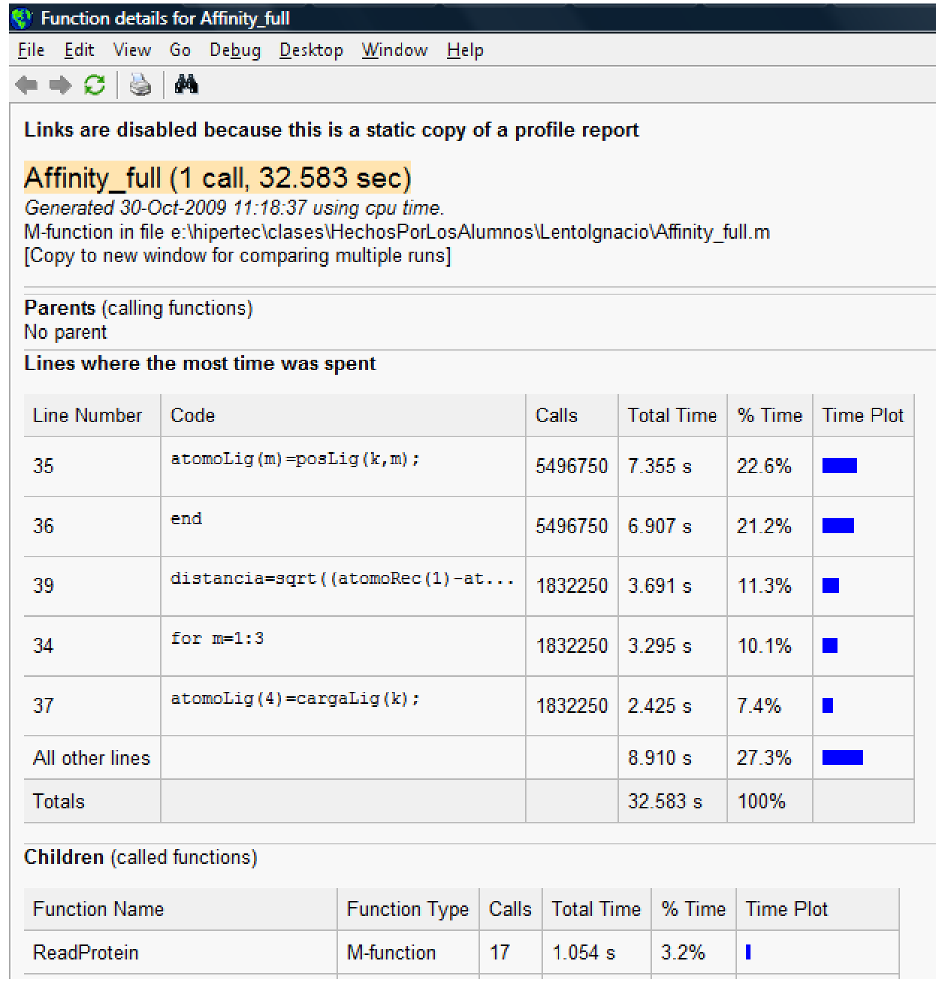
Efficiency: Profiler

>> profile on

>> Affinity_full

>> profile viewer

The profiler tells you what lines of your program are consuming more time



Function details for Affinity_full

File Edit View Go Debug Desktop Window Help

Links are disabled because this is a static copy of a profile report

Affinity_full (1 call, 32.583 sec)

Generated 30-Oct-2009 11:18:37 using cpu time.

M-function in file e:\hipertec\clases\HechosPorLosAlumnos\LentoIgnacio\Affinity_full.m

[Copy to new window for comparing multiple runs]

Parents (calling functions)

No parent

Lines where the most time was spent

Line Number	Code	Calls	Total Time	% Time	Time Plot
35	`atomoLig(m)=posLig(k,m);`	5496750	7.355 s	22.6%	
36	`end`	5496750	6.907 s	21.2%	
39	`distancia=sqrt((atomoRec(1)-at...`	1832250	3.691 s	11.3%	
34	`for m=1:3`	1832250	3.295 s	10.1%	
37	`atomoLig(4)=cargaLig(k);`	1832250	2.425 s	7.4%	
All other lines			8.910 s	27.3%	
Totals			32.583 s	100%	

Children (called functions)

Function Name	Function Type	Calls	Total Time	% Time	Time Plot
ReadProtein	M-function	17	1.054 s	3.2%	

3D matrices

```
>> matriz(1,3,2)
```

→ Just like 2D matrices, but adding a third index.

And if we come to that…

```
>> matriz(1,3,2,7,14,8)
```

Matlab matrices may have many dimensions

Cells

- Cells are “matrices of matrices”

For cells, use braces instead of parenthesis

```
>> a{1,1}='Holapis';
>> a{1,2}=7;
>> a{2,1}=1:4

a =

    'Holapis'       [7]
    [1x4 double]     []

>> a{2,1}(3)
ans =
     3
```

Each component stores a different type of variable.

Third element of the vector contained in the cell.

Element 2,1 of the cell (which was the vector with 4 components)

Structures

- Structures are sets of variables of different types

Name of the structure

Dot

Name of variable within the structure (called “field”)

```
>> casita.a=1;
>> casita.hola='saludete';
>> casita.matriz=rand(10);
>> casita

casita = 

         a: 1
      hola: 'saludete'
    matriz: [10x10 double]
```

It is possible to have matrices of structures, so that casita(1) is a complete structure, and casita(2) is another one, with the same fields, but with different values.

Logical indexing (I)

Logical variables:

>> logical([0 1 0 0 0 1]); → Transforms numeric variables in logical ones (0=false, nonzero=true)

>> vector=rand(1,10);

>> vector([1 4 6 7])

>> vector(logical([1 0 0 1 0 1 1 0 0 0]))

Logical vector with 'true' in the elements that we want

Logical indexing (II)

A natural way of creating logical variables:

```
>> vec=[2 3 3 7 3 4 1];
>> igualesatres = vec==3
igualesatres =
        0 1 1 0 1 0 0
```

True for the elements that are equal to 3

Logical indexing (III)

The most basic logical indexing: Not using find

```
>> vec=[2 3 3 7 3 4 1];
>> dat=[0.1 0.2 5.7 -2.1 5 6 9.4];
>> indices=find(vec==3);
>> datosbuenos=dat(indices)
datosbuenos =
          0.2   5.7   5
```

```
>> vec=[2 3 3 7 3 4 1];
>> dat=[0.1 0.2 5.7 -2.1 5 6 9.4];
>>buenos= vec==3;
>> datosbuenos=dat(buenos)
datosbuenos =
          0.2   5.7   5
```

```
>> vec=[2 3 3 7 3 4 1];
>> dat=[0.1 0.2 5.7 -2.1 5 6 9.4];
>> datosbuenos=dat(vec==3)
datosbuenos =
          0.2   5.7   5
```

Working with images: Load an image

```
>> mat_img=imread('filename.tif');
```

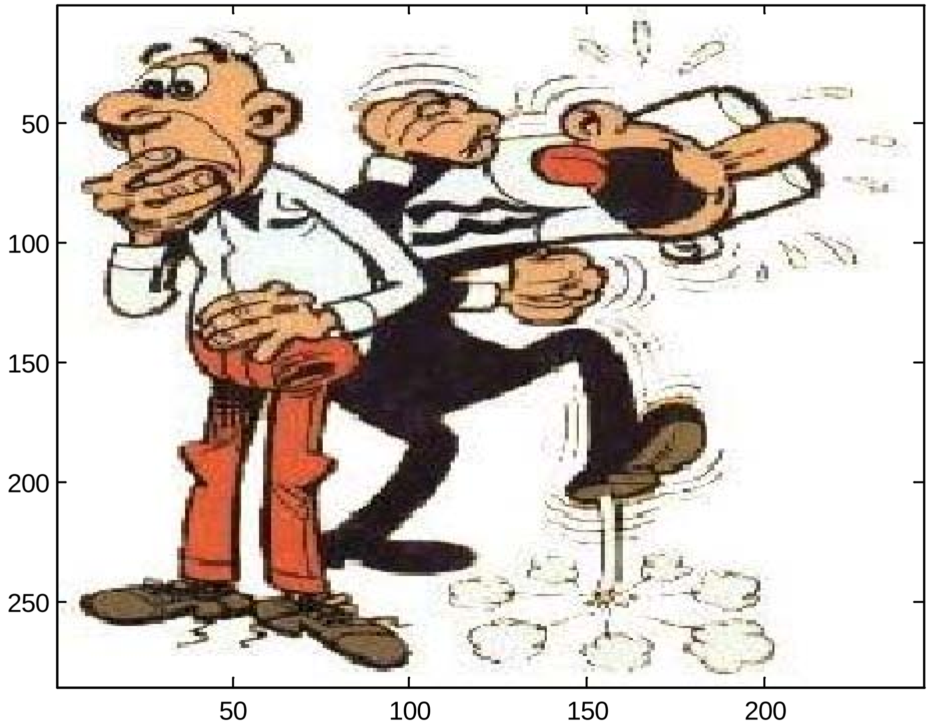
The image is stored in a matrix (2D matrix if the image is grayscale, 3D matrix if it is a color image)

Matlab supports most image formats

Each component of the matrix stores the gray level of one pixel of the image (in color images, the 3D matrix has three components in the third dimension. Each of these three matrices stores the component for the r,g,b code of each pixel)

Matlab’s representation of images

Pixel number.
Rows: First component of the matrix



Pixel number.
Columns: second component of the matrix