INTENSIVE COMPUTATION

Annalisa Massini

2020-2021

Lecture 3

INTRODUCTION TO MATLAB

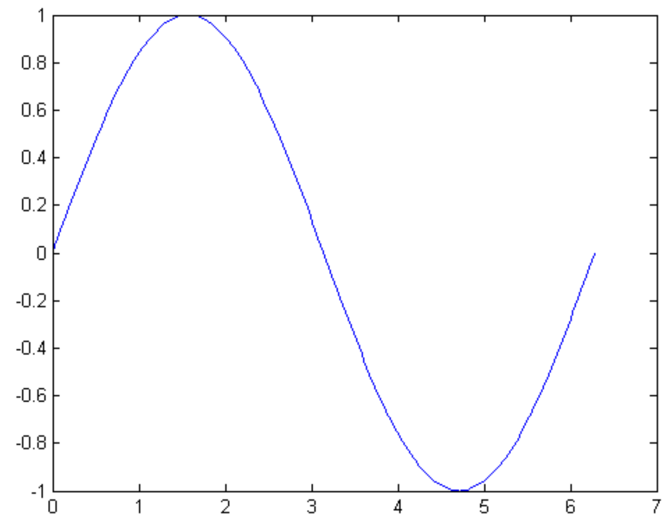
Part 2

Plotting

The function **plot** creates a 2D line plot - it can be used in different ways

- **Example**

```
» n = 31
» x = linspace(0,2*pi,n)
» y = sin(x)
» plot(x,y)
```



x is a vector of linearly spaced values between 0 and 2π

y is the vector of values of sine function evaluated at the values in x

Plotting

- Command **plot** is:
  - **plot(X,Y,options)**

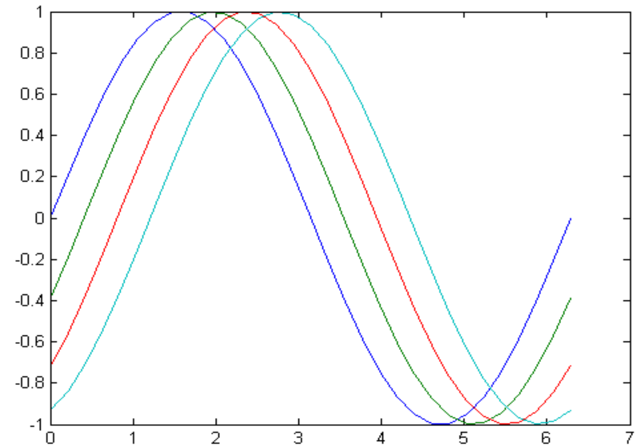
  Where **X** is for abscissas and **Y** is for ordinates

  options sets the *line style*, *marker symbol*, and *color*

- To plot ***multiple lines*** in the same windows, we can use two ways:

```
y2 = sin(x - .4);
y3 = sin(x - .8);
y4 = sin(x - 1.2);
```

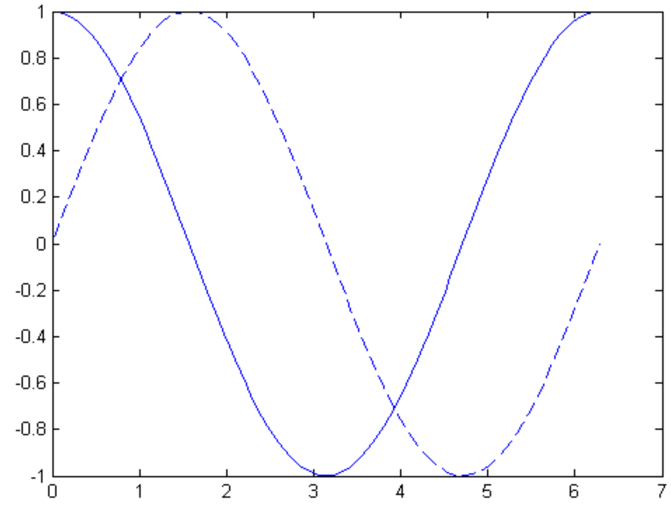
- `plot(x,y,x,y2,x,y3,x,y4)`
- `plot(x,[y;y2;y3;y4])`



Plotting

- Another way to plot **multiple line** in the same window is by using commands **`hold on`** and **`hold off`**:

```
» x = linspace(0,2*pi)
» y1 = cos(x)
» y2 = sin(x)
» plot(x,y1,'-')
» hold on
» plot(x,y2,'--')
» hold off
```



Plotting

- You can add a ***title*** and ***axis labels*** to the graph

```
» title('title of the graph')
» xlabel('x axis')
» ylabel('y axis')
```

- **axis** - axis scaling and appearance
- **legend** - graph legend
- **text** - create text object in current axes

```
» text(x(70)+0.5,r(70),'r = -2x')
```

- **grid on** add grid lines for 2D and 3D plots

Plotting

Other functions for graphs are:

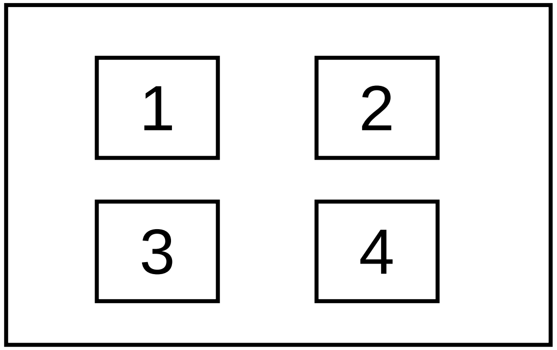
- `loglog` Log-log scale plot
- `semilogx` Semilogarithmic plot (x logarithmic, y linear)
- `semilogy` Semilogarithmic plot (x linear, y logarithmic)
- `errorbar` Plot error bars along curve
- `bar` Bar graph
- `stairs` Stairstep graph
- `scatter` Scatter plot

Plotting

`subplot` divides the current figure into ***grid***, it numbers the cells by rows

» **`subplot(m,n,p)`**

divides the current figure into an **m-by-n** grid and plots in the **grid position** specified by p



Plotting

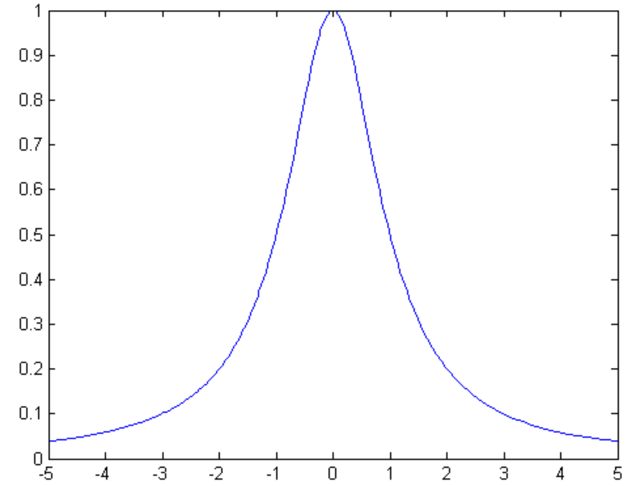
fplot(fun, lims) plots a function

- **fun**, that must be ***a string***
- between the limits specified by **lims**, specifying the ***x-axis limits*** ([xmin xmax]), or the ***x- and y-axes limits***, ([xmin xmax ymin ymax])

```
» fun='1/(1+x^2)';
» lims=[-5,5];
» fplot(fun,lims);
```

or the equivalent

```
» fplot('1/(1+x^2)', [-5,5]);
```



Plotting

- `fplot(fun,limits,LineSpec)` plots fun using the line specification ***LineSpec***

```
fplot(fun, lims, '- -')
fplot(fun, lims, 'r -')
```

- **`fplot`** can plot a vector of functions

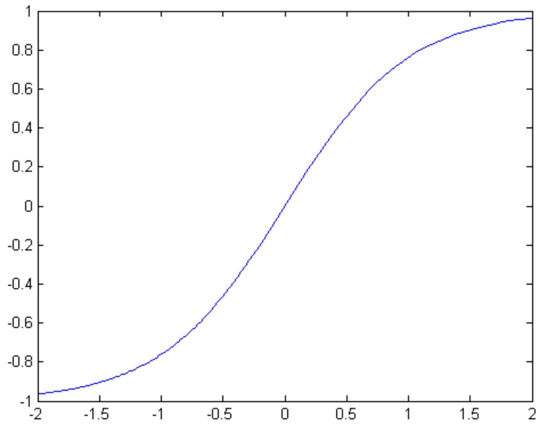
```
fplot('[sin(t), sin(t-.25), sin(t-.5)]',[0,2*pi])
```

Plotting

- **ezplot** plots the expression fun(x) over the default domain -2π < *x* < 2π, where fun(x) is an explicit function of only *x*
- **ezplot(fun,[xmin,xmax])** plots fun(x) over the domain: xmin < *x* < xmax

- Both for **fplot** and **ezplot** **fun** can be a **function handle**

```
fh = @tanh;
fplot(fh,[-2,2])
```

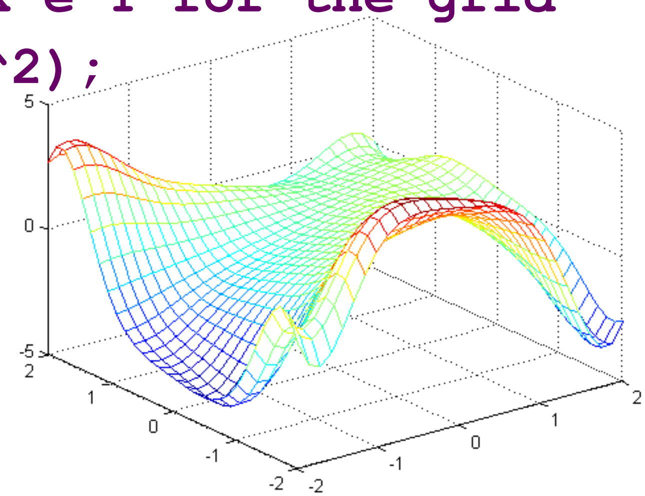


Plotting

3D plot with **mesh** and **surf**

- **mesh** and **surf** plot a surface
- **mesh** and **surf** create 3D surface plots of matrix data generated by the command **meshgrid**

```
» n=30; m=n;
» x=linspace(-2,2,n);
» y=linspace(-2,2,n);
» [X,Y]=meshgrid(x,y); % matrices X e Y for the grid
» Z=(1-Y).*cos(X.^2)+(X-1).*cos(Y.^2);
» mesh(X,Y,Z);
```



Data and file management

You can load variables from file into workspace with **load**

For example if you want analyze data coming from a program, like the following, that are in the file data.dat

```
1       0.2000        -5
2       0.2500        -9
3       0.0740       -23
4       0.0310       -53
5       0.0160      -105
6       0.0090      -185
7       0.0050      -299
8       0.0030      -453
9       0.0020      -653
10      0.0020      -905
```

Data and file management

If you load these data with the function **`load`**, a matrix is created of size 10x3

```
>> load data.dat
>> whos
Name Size Bytes Class
data 10x3 240 double array
Grand total is 30 elements using 240 bytes
```

`load filename` is the command form

`load 'filename'` is the function form

Data and file management

```
>> M = load('data.dat')
M =
1.0000  2.0000    -5.0000
2.0000  0.2500    -9.0000
3.0000  0.0740   -23.0000
4.0000  0.0310   -53.0000
5.0000  0.0160  -105.0000
6.0000  0.0090  -185.0000
7.0000  0.0050  -299.0000
8.0000  0.0030  -453.0000
9.0000  0.0020  -653.0000
10.0000 0.0020  -905.0000
```

Data and file management

`save` save workspace variables to file

- **`save (`*`filename`*`)`**
  saves all variables from the current workspace in a formatted binary file (MAT-file) called ***`filename`***
  if ***`filename`*** is not specified the file **`Matlab.mat`** is created

- **`save(filename,variables)`**
  saves only the variables or fields of a structure array specified by variables

- **`save(`*`filename,variables,fmt`*`)`**
  saves in the file format specified by ***`fmt`*** - ***`variables`*** is optional

Data and file management

Example

```
% mytable.m
n=input('Insert the number of values n:');
x=linspace(0,pi,n);
s=sin(x);
c=cos(x);
v=(1:n);
save mytable.dat v x s c -ascii
```

Data and file management

Example

To visualize the table saved in the previous example with save we can load the file and display the table

```
% viewtable.m
load mytable.dat
A=mytable;
disp('--------------------------------------');
fprintf('k\t x(k)\t sin(x(k))\t cos(x(k))\n');
disp('--------------------------------------');
fprintf('%d\t %3.2f\t %8.5f\t %8.5f\n',A);
```

Data and file management

`dir` List directory

`dir` *`directory_name`* or `dir('`*`directory_name`*`')` lists the files in a directory -- Pathnames and wildcards may be used

`dir *.m` lists all the M-files in the current directory

`D = dir('directory_name')` returns the results in an M-by-1 **structure** with the fields:

name -- filename

date -- modification date

bytes -- number of bytes allocated to the file

isdir -- 1 if name is a directory and 0 if not

datenum -- modification date as a MATLAB serial date number

Improving performance

Techniques for Improving Performance

- **Preallocating Arrays**
  - `for` and `while` loops that incrementally increase the size of a data structure each time through the loop can adversely affect performance and memory use
  - resizing arrays often requires MATLAB to spend extra time looking for larger contiguous blocks of memory, and then moving the array into those blocks
  - you can improve code execution time by **preallocating** the maximum amount of space required for the array

Improving performance

Techniques for Improving Performance

- **Preallocating a Nondouble Matrix**
  - Use the following command to create an array having **`int8`** values

```
A = zeros(100, 'int8')
```
← *EFFICIENT*

  - This command allows to save time and memory

  - **Avoid** using the following method, when you preallocate a block of memory to hold a matrix of some type other than double

```
A = int8(zeros(100))
```
← *INEFFICIENT*

  - This statement preallocates a 100-by-100 matrix of **`int8`**, first by creating a ***full matrix of double values***, and then by converts each element to **`int8`**

Improving performance

Techniques for Improving Performance

- **Vectorization**
  - MATLAB is optimized for operations involving matrices and vectors
  - The process of revising loop-based, scalar-oriented code to use MATLAB matrix and vector operations is called ***vectorization***
- Vectorizing your code is worthwhile for several reasons:
  - *Appearance*: Vectorized mathematical code appears more like the mathematical expressions, making the code easier to understand
  - *Less Error Prone*: Without loops, vectorized code is often shorter, and fewer lines of code mean fewer programming errors
  - *Performance*: Vectorized code often runs much faster

Improving performance

- **Vectorizing Code for General Computing**

  - This code computes the sine of 1,001 values ranging from 0 to 10:

```
i = 0;
for t = 0:.01:10
  i = i + 1;
  y(i) = sin(t);
end
```

  - This is a vectorized version of the same code:

```
t = 0:.01:10;
y = sin(t);
```

Improving performance

- **Vectorizing Code for Specific Tasks**

  - This code computes the cumulative sum of a vector at every fifth element:

```
x = 1:10000;
ylength = (length(x) - mod(length(x),5))/5;
y(1:ylength) = 0;
for n= 5:5:length(x)
  y(n/5) = sum(x(1:n));
end
```

  - This code shows one way to accomplish the task:

```
x = 1:10000;
xsums = cumsum(x);
y = xsums(5:5:length(x));
```

Improving performance

- **Array Operations**
  - Array operators perform the same operation for all elements in the data set
  - **Example**
    - collect the volume (V) of various cones by recording their diameter (D) and height (H)
    - The volume for that single cone: `V = 1/12*pi*(D^2)*H`
    - Consider 10,000 cones
    - The vectors D and H each contain 10,000 elements

```
for n = 1:10000
    V(n) = 1/12*pi*(D(n)^2)*H(n));
end
```

  - Vectorized Calculation

```
V = 1/12*pi*(D.^2).*H;
```

More examples

Use built-in Matlab functions

- find is a very important function
  - Returns indices of nonzero values
  - Can simplify code and help avoid loops
- Basic syntax: index=find(cond)

```
» x=rand(1,100);
» inds = find(x>0.4 & x<0.6);
```

  - **Inds will contain the indices at which x has values between 0.4 and 0.6.**
  - **This is what happens:**
    - x>0.4 returns a vector with 1 where true and 0 where false
    - x<0.6 returns a similar vector
    - The & combines the two vectors using an **and**
    - The find returns the indices of the 1's

More examples

- Given x= sin(linspace(0,10*pi,100)), how many of the entries are positive?

- Using a loop and if/else

```
count=0;
for n=1:length(x)
 if x(n)>0
     count=count+1;
 end
end
```

- Being more clever

```
count=length(find(x>0));
```

- **Avoid loops!** Built-in functions will make it faster to write and execute

PARALLEL TOOLBOX

Parallel Computing

- Parallel Computing: Using multiple computer processing units (CPUs) at the same time to solve a problem
- The compute resources might be:
  - computer with multiple processors or
  - networked computers
- The computational problem should be able to:
  - Be broken into discrete parts that can be solved simultaneously and independently
  - Be solved in less time with multiple compute resources than with a single compute resource.

Parallel Computing in Matlab

- Parallel Computing Toolbox (PCT)
  - shared memory, single node
  - parfor
- Matlab Distributed Computing Server (MDCS)
  - distributed computing across nodes
  - spmd or parfor
- Built-in multithreading
  - shared memory, single node

Parallel Computing in MATLAB

- MATLAB Parallel Computing Toolbox
  - Workers limited only by resources on the node, see parallel preferences for the default. Typically the entire node.
  - Built in functions for parallel computing
    - `parfor` loop (for running task-parallel algorithms on multiple processors)
    - `spmd` (handles large datasets and data-parallel algorithms)

Primary Parallel Commands

- `parpool`
  - `mypool = parpool(4)`
  - `… do work …`
  - `delete(mypool)`

- `parfor` (for loop)

- `spmd` (distributed computing for datasets)

parpool

- Use **`parpool`** to open a pool of ***workers*** to execute code on other compute cores
- In Matlab you can think of *workers* like *threads* or *processes*
- You can open these workers locally (on the same node) or remotely
- Local access is enabled by the Parallel Computing Toolkit, remote access is enabled via MDCS (Matlab Distributed Computing Server)

parpool

Starting a parallel pool

- `mypool = parpool ('local',4);`
- `Mypool = parpool(4);`
  - Opens 4 workers locally on the same node
  - Communication is fastest within a node
  - Make sure you submitted your Matlab job with "-n X" where X matches the number of workers you open! Use –N 1 to ensure the slots are all on the same node (i.e. local)
- Use `display(mypool)` to show information about the pool

Closing a parallel pool

- `delete(mypool)` to end parallel session
- If you didn't save the name you can use
  - `delete (gcp('nocreate'));`

Parallel for Loops (parfor)

- `parfor` loop executes a series of statements in the loop body in parallel
- A parfor-loop can provide significantly better performance than its analogous for-loop, because *several MATLAB workers* can compute simultaneously on the same loop
- Each execution of the body of a parfor-loop is an **iteration**
- The MATLAB client issues the `parfor` command and coordinates with MATLAB workers to execute the loop iterations in parallel on the workers in a parallel pool
- The client sends the necessary data on which `parfor` operates to workers, where most of the computation is executed
- The results are sent back to the client and assembled

Parallel for Loops (parfor)

- MATLAB workers evaluate *iterations in no particular order* and independently of each other
- Because each iteration is independent, there is no guarantee that the iterations are synchronized in any way, nor is there any need for this

- If the number of workers is equal to the number of loop iterations, each worker performs one iteration of the loop
- If there are more iterations than workers, some workers perform more than one loop iteration; in this case, a worker might receive multiple iterations at once to reduce communication time

Parallel for Loops (parfor)

A **parfor-loop** can be **useful** if you have a slow for-loop.

Consider parfor if you have:

- Some loop iterations that take a long time to execute.
  - In this case, the workers can execute the long iterations simultaneously.
  - Make sure that the number of iterations exceeds the number of workers. Otherwise, you will not use all workers available.
- Many loop iterations of a simple calculation (such as a Monte Carlo simulation or a parameter sweep)
  - parfor divides the loop iterations into groups so that each worker executes some portion of the total number of iterations.

Parallel for Loops (parfor)

A **parfor-loop** might **not be useful** if you have:

- Code that has vectorized out the for-loops
  - If you want to make code run faster, first try to vectorize it
  - Vectorizing code allows you to benefit from the built-in parallelism provided by the multithreaded nature of many of the underlying MATLAB libraries
  - However, if you have vectorized code and you have access only to local workers, then ***parfor-loops may run slower than for-loops***
  - Do not devectorize code to allow for parfor; in general, this solution does not work well
- Loop iterations that take a short time to execute
  - In this case, ***parallel overhead dominates your calculation***

Parfor example

Will work in parallel, loop increments are not dependent on each other

```
mypool =parpool(2)
j=zeros(100,1); %pre-allocate vector
parfor i=1:100;
    j(i,1)=i^4;
end;
delete(mypool);
```

Makes the loop run in parallel

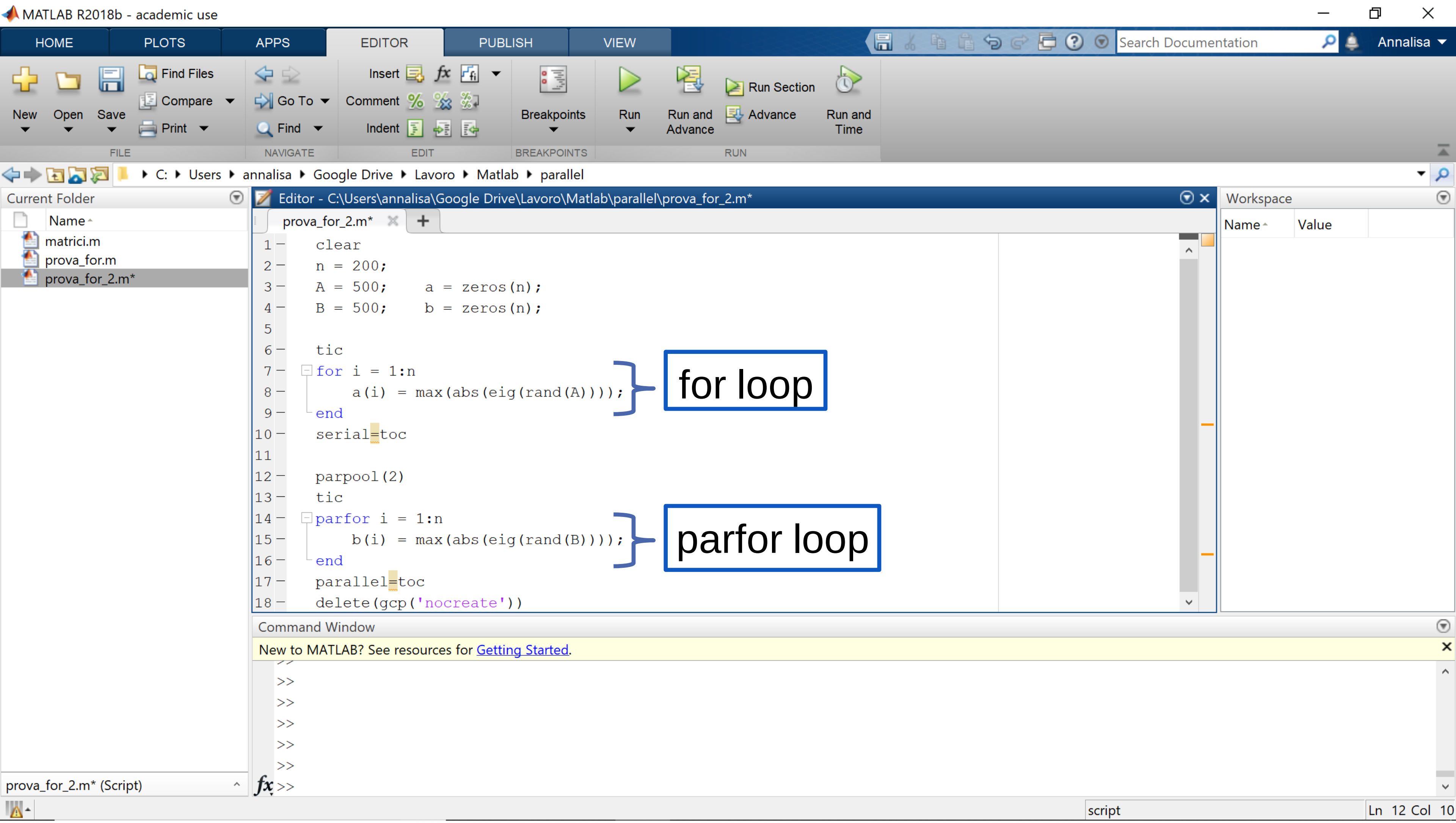
Serial Loop example

- DOES NOT work in parallel - **it's serial**:

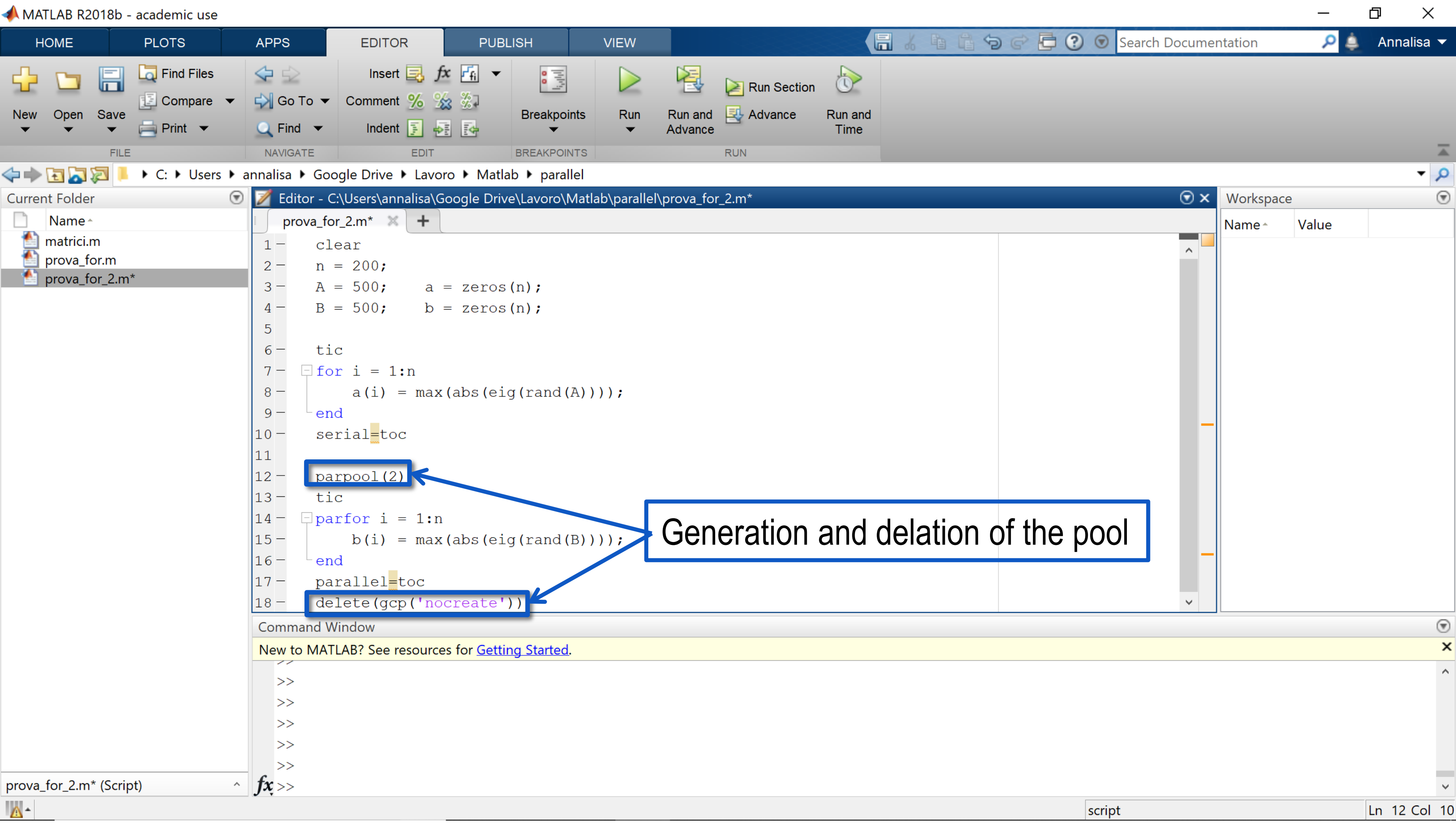
```
j=zeros(100,1); %pre-allocate vector
j(1)=5;
for i=2:100;
    j(i,1)=2*j(i-1);
end;
```

`j(i-1)` needed to calculate `j(i,1)`
→ serial!

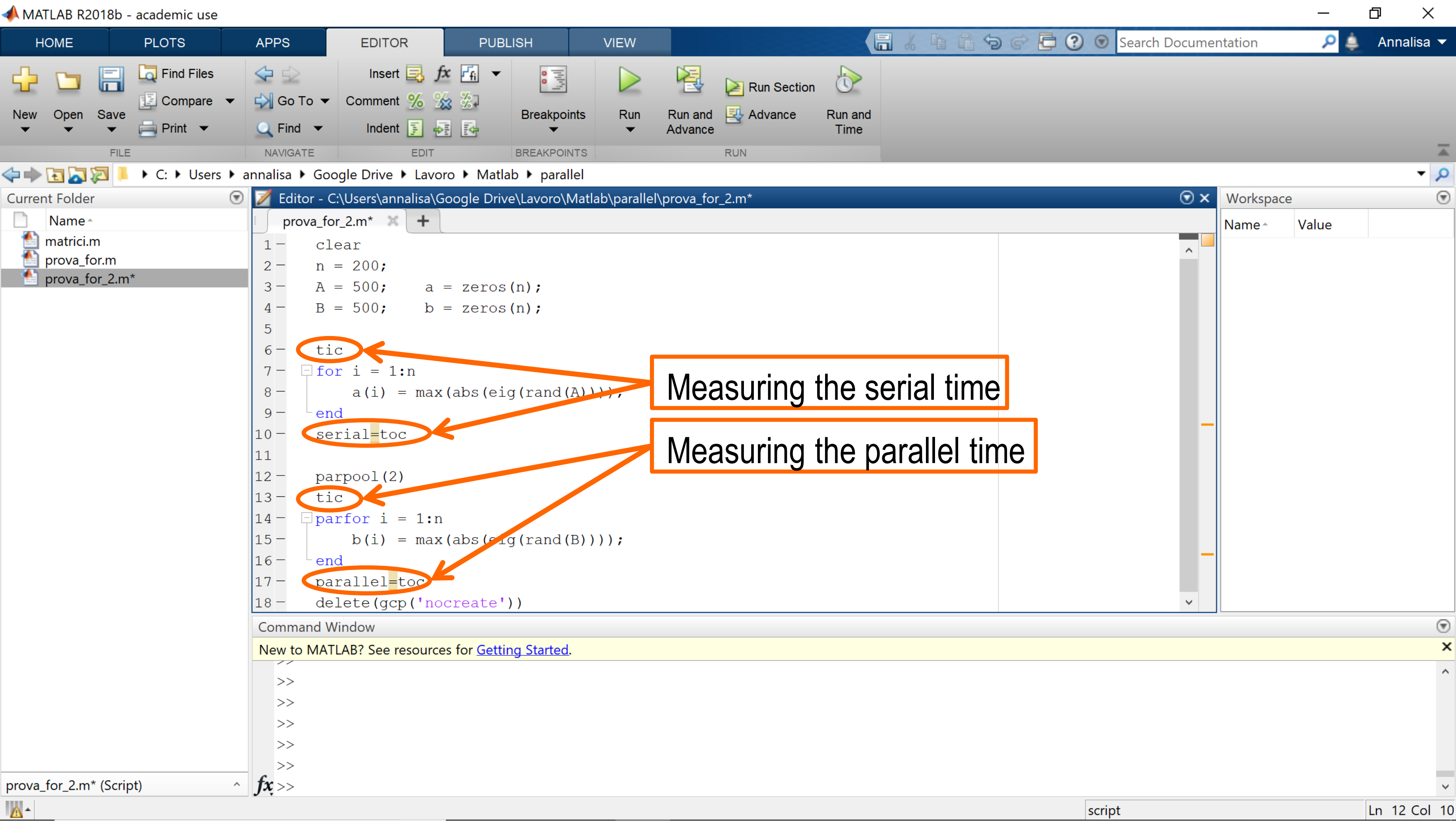
Example


MATLAB R2018b - academic use
Editor - C:\Users\annalisa\Google Drive\Lavoro\Matlab\parallel\prova_for_2.m*
clear
n = 200;
A = 500;   a = zeros(n);
B = 500;   b = zeros(n);
tic
for i = 1:n
a(i) = max(abs(eig(rand(A))));
end
serial=toc
parpool(2)
tic
parfor i = 1:n
b(i) = max(abs(eig(rand(B))));
end
parallel=toc
delete(gcp('nocreate'))
for loop
parfor loop
Current Folder
matrici.m
prova_for.m
prova_for_2.m*
Workspace
Command Window
New to MATLAB? See resources for Getting Started.

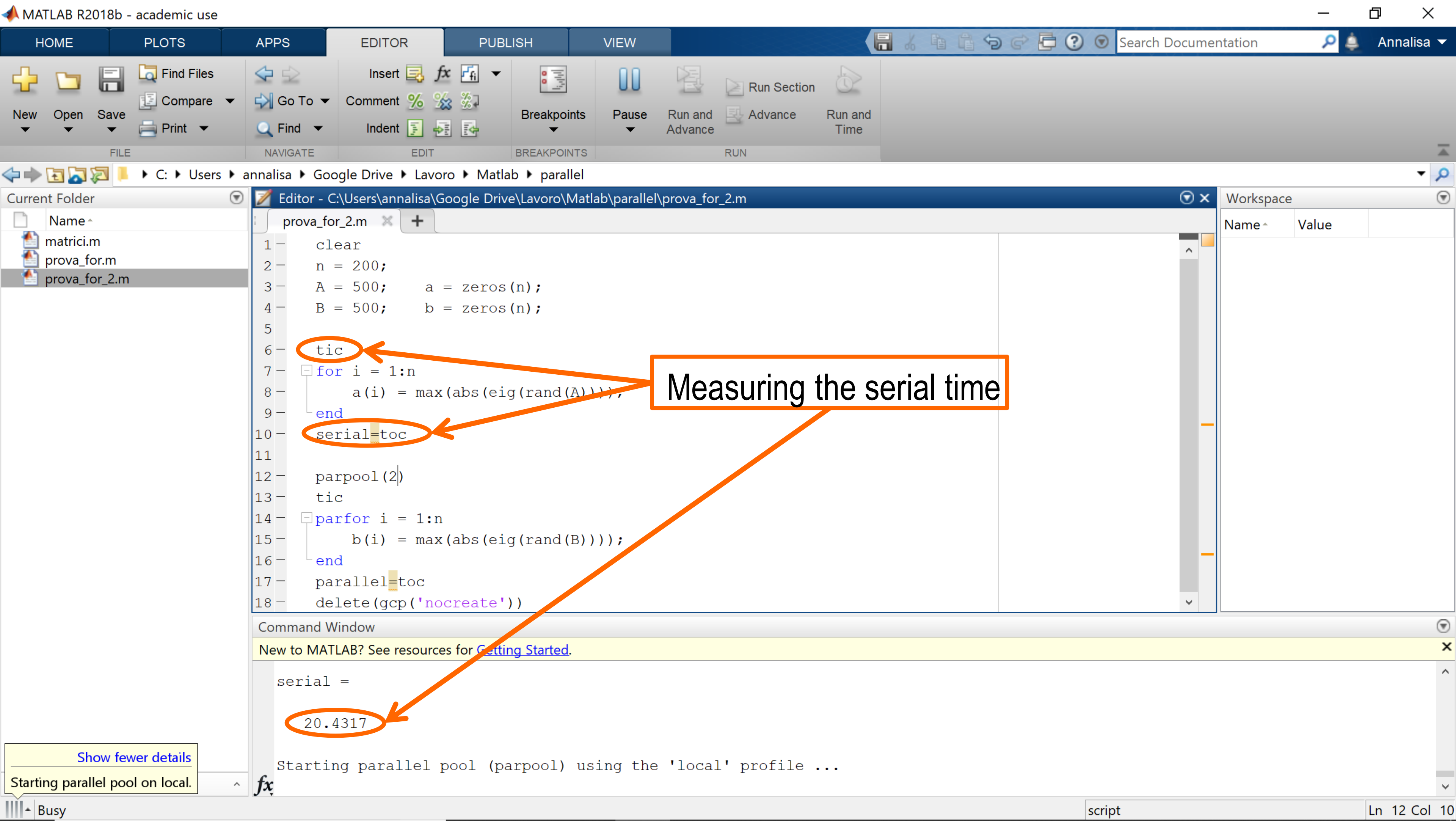
Example


MATLAB R2018b - academic use
HOME
PLOTS
APPS
EDITOR
PUBLISH
VIEW
Search Documentation
Annalisa
New
Open
Save
Find Files
Compare
Print
Go To
Find
Insert
Comment
Indent
Breakpoints
Run
Run and Advance
Run Section
Advance
Run and Time
FILE
NAVIGATE
EDIT
BREAKPOINTS
RUN
C: ▸ Users ▸ annalisa ▸ Google Drive ▸ Lavoro ▸ Matlab ▸ parallel
Current Folder
Name
matrici.m
prova_for.m
prova_for_2.m*
Editor - C:\Users\annalisa\Google Drive\Lavoro\Matlab\parallel\prova_for_2.m*
prova_for_2.m*
clear
n = 200;
A = 500; a = zeros(n);
B = 500; b = zeros(n);
tic
for i = 1:n
a(i) = max(abs(eig(rand(A))));
end
serial=toc
parpool(2)
tic
parfor i = 1:n
b(i) = max(abs(eig(rand(B))));
end
parallel=toc
delete(gcp('nocreate'))
Generation and delation of the pool
Workspace
Name
Value
Command Window
New to MATLAB? See resources for Getting Started.
>>
prova_for_2.m* (Script)
script
Ln 12 Col 10

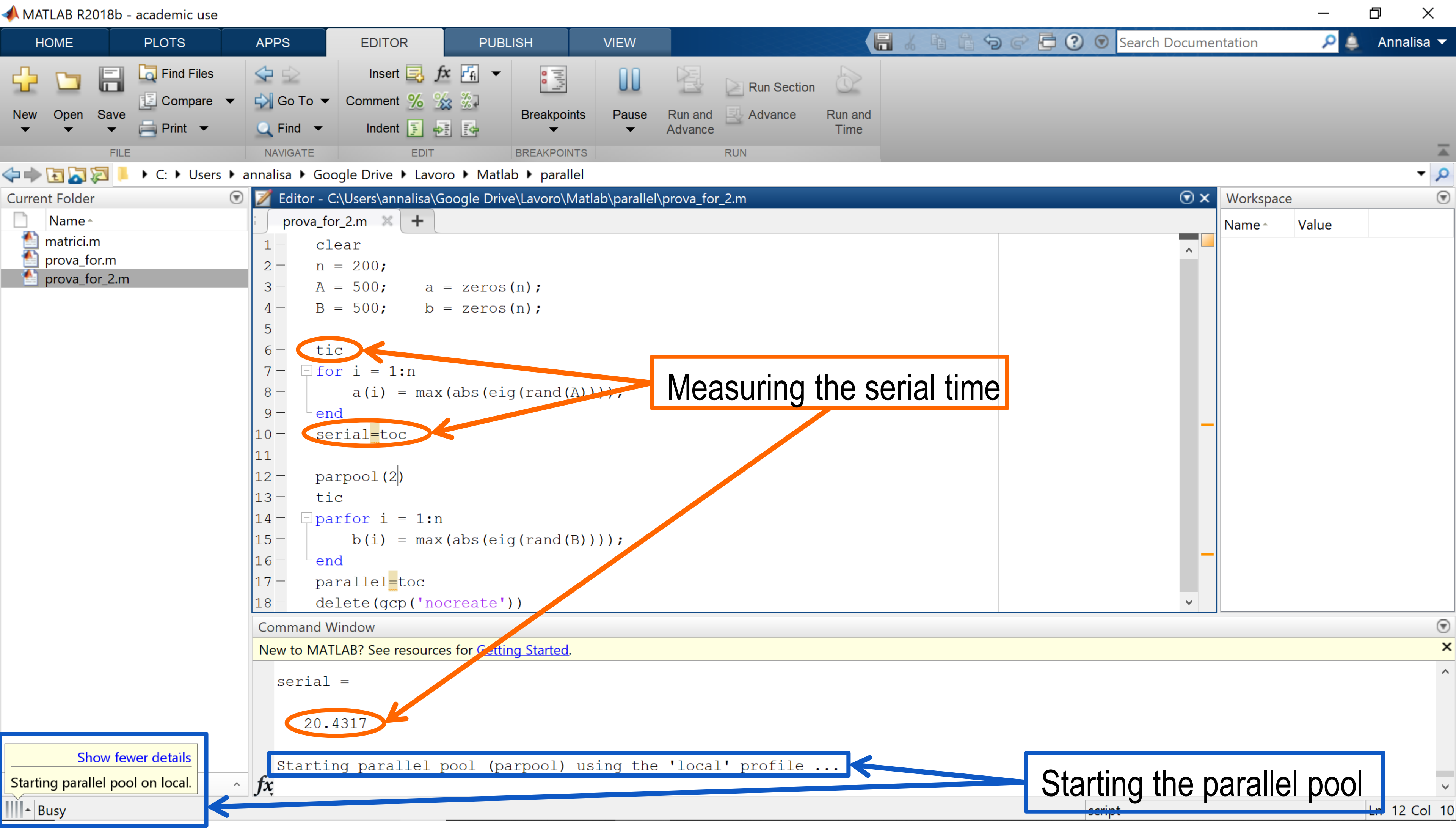
Example


MATLAB R2018b - academic use
HOME
PLOTS
APPS
EDITOR
PUBLISH
VIEW
Search Documentation
Annalisa
New
Open
Save
Find Files
Compare
Print
FILE
Go To
Find
NAVIGATE
Insert
Comment
Indent
EDIT
Breakpoints
BREAKPOINTS
Run
Run and Advance
Run Section
Advance
Run and Time
RUN
C: ▸ Users ▸ annalisa ▸ Google Drive ▸ Lavoro ▸ Matlab ▸ parallel
Current Folder
Name
matrici.m
prova_for.m
prova_for_2.m*
Editor - C:\Users\annalisa\Google Drive\Lavoro\Matlab\parallel\prova_for_2.m*
prova_for_2.m*
clear
n = 200;
A = 500;   a = zeros(n);
B = 500;   b = zeros(n);
tic
for i = 1:n
a(i) = max(abs(eig(rand(A))));
end
serial=toc
parpool(2)
tic
parfor i = 1:n
b(i) = max(abs(eig(rand(B))));
end
parallel=toc
delete(gcp('nocreate'))
Measuring the serial time
Measuring the parallel time
Workspace
Name
Value
Command Window
New to MATLAB? See resources for Getting Started.
>>
>>
>>
>>
>>
>>
prova_for_2.m* (Script)
script
Ln 12 Col 10

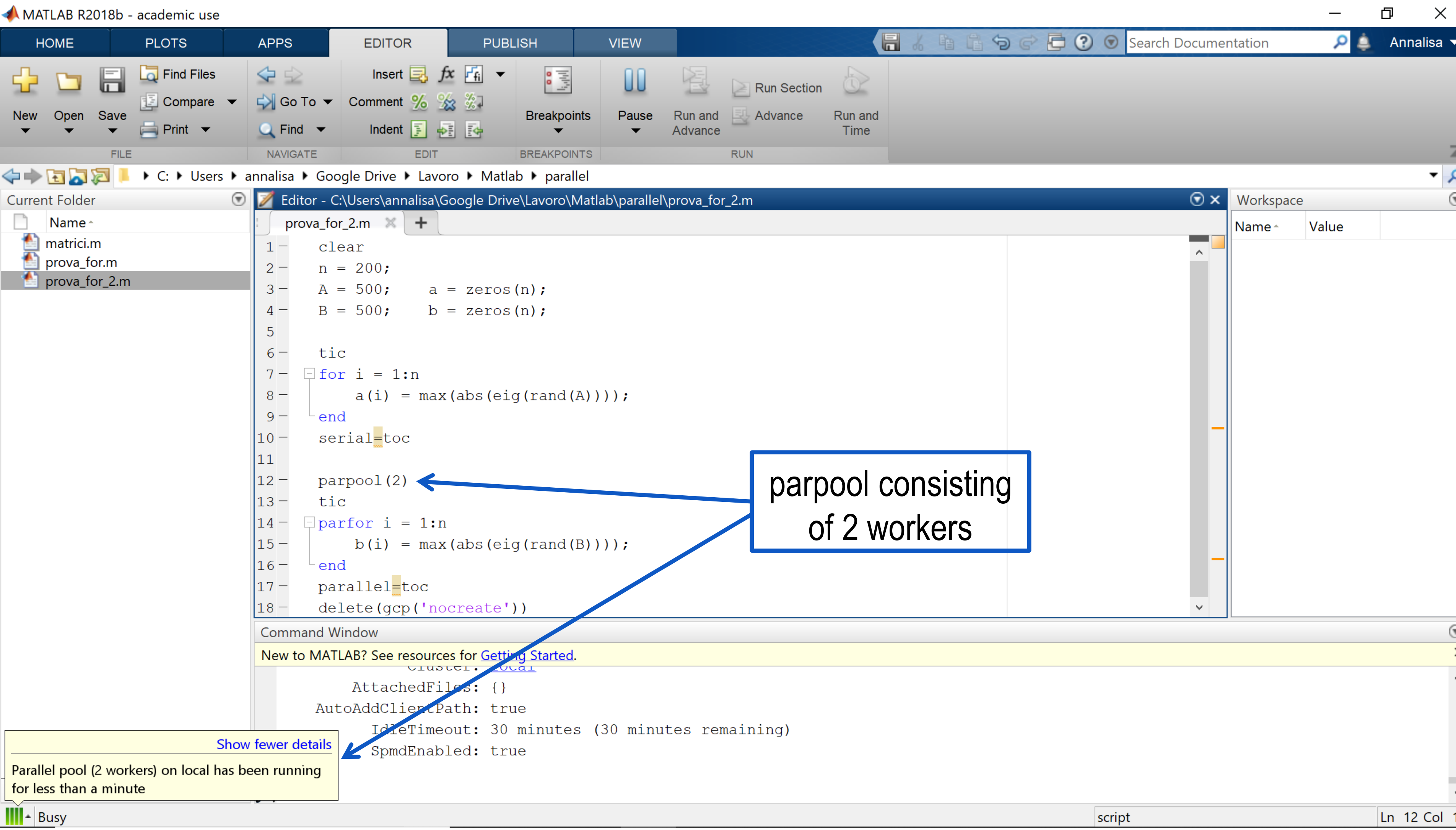
Example


MATLAB R2018b - academic use
Editor - C:\Users\annalisa\Google Drive\Lavoro\Matlab\parallel\prova_for_2.m
prova_for_2.m
clear
n = 200;
A = 500;   a = zeros(n);
B = 500;   b = zeros(n);
tic
for i = 1:n
a(i) = max(abs(eig(rand(A))));
end
serial=toc
parpool(2)
tic
parfor i = 1:n
b(i) = max(abs(eig(rand(B))));
end
parallel=toc
delete(gcp('nocreate'))
Measuring the serial time
Command Window
serial =
20.4317
Starting parallel pool (parpool) using the 'local' profile ...
Current Folder
matrici.m
prova_for.m
prova_for_2.m
Workspace
Starting parallel pool on local.
Busy

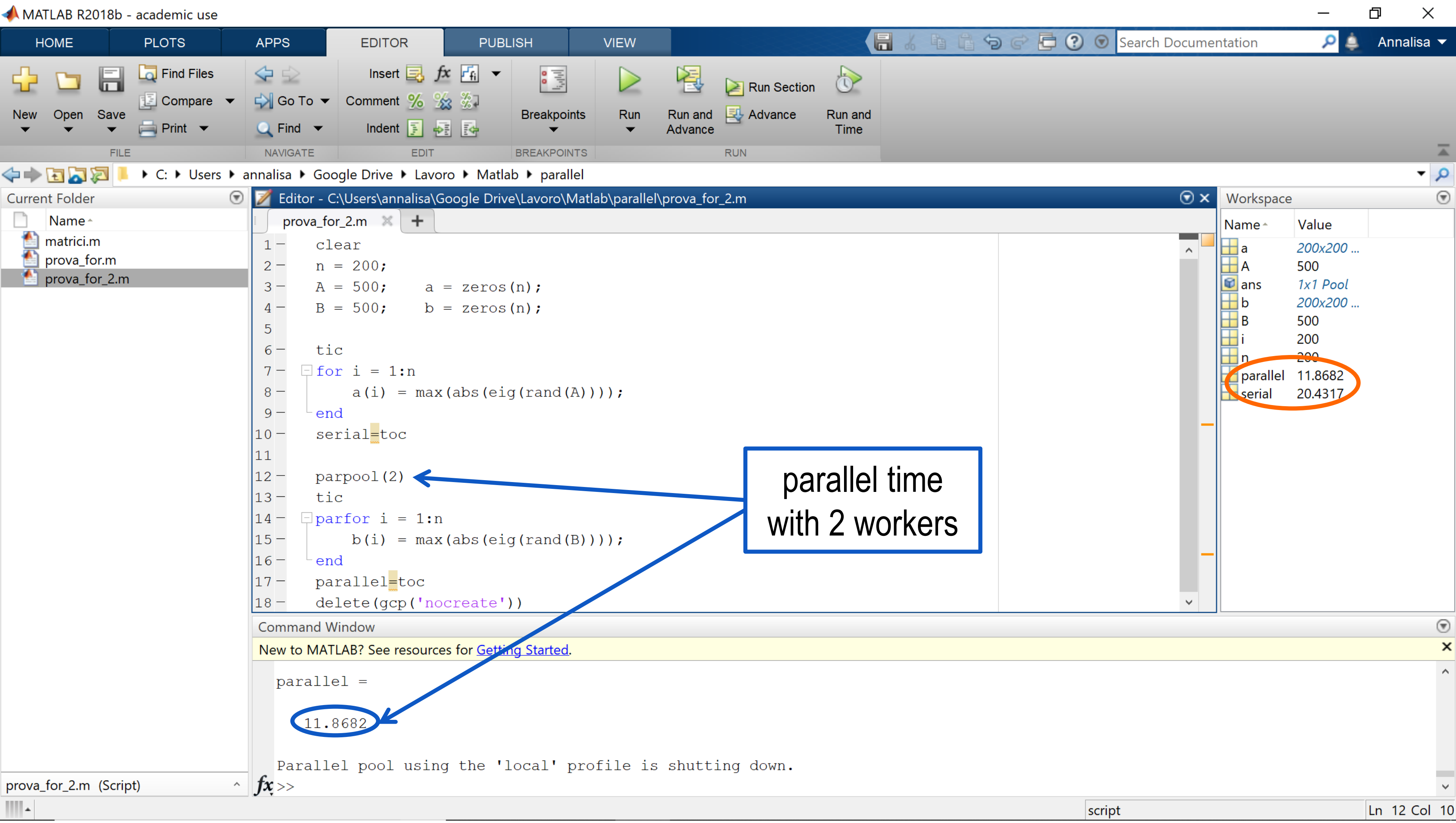
Example


MATLAB R2018b - academic use
Editor - C:\Users\annalisa\Google Drive\Lavoro\Matlab\parallel\prova_for_2.m
clear
n = 200;
A = 500; a = zeros(n);
B = 500; b = zeros(n);
tic
for i = 1:n
a(i) = max(abs(eig(rand(A))));
end
serial=toc
parpool(2)
tic
parfor i = 1:n
b(i) = max(abs(eig(rand(B))));
end
parallel=toc
delete(gcp('nocreate'))
Measuring the serial time
Command Window
serial =
20.4317
Starting parallel pool (parpool) using the 'local' profile ...
Starting the parallel pool
Show fewer details
Starting parallel pool on local.
Busy

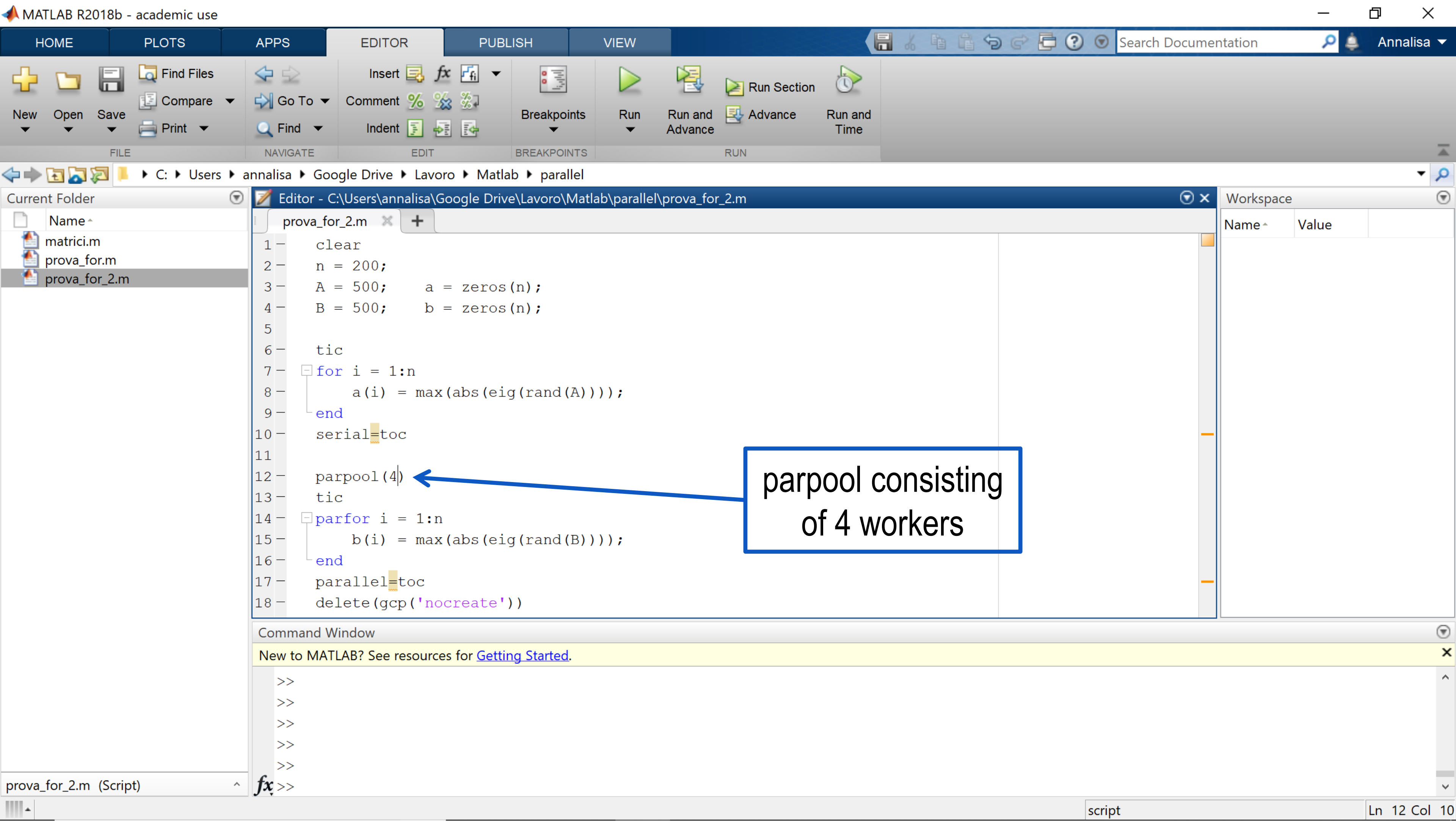
Example


MATLAB R2018b - academic use
Editor - C:\Users\annalisa\Google Drive\Lavoro\Matlab\parallel\prova_for_2.m
clear
n = 200;
A = 500;   a = zeros(n);
B = 500;   b = zeros(n);
tic
for i = 1:n
a(i) = max(abs(eig(rand(A))));
end
serial=toc
parpool(2)
tic
parfor i = 1:n
b(i) = max(abs(eig(rand(B))));
end
parallel=toc
delete(gcp('nocreate'))
parpool consisting of 2 workers
AttachedFiles: {}
AutoAddClientPath: true
IdleTimeout: 30 minutes (30 minutes remaining)
SpmdEnabled: true
Show fewer details
Parallel pool (2 workers) on local has been running for less than a minute

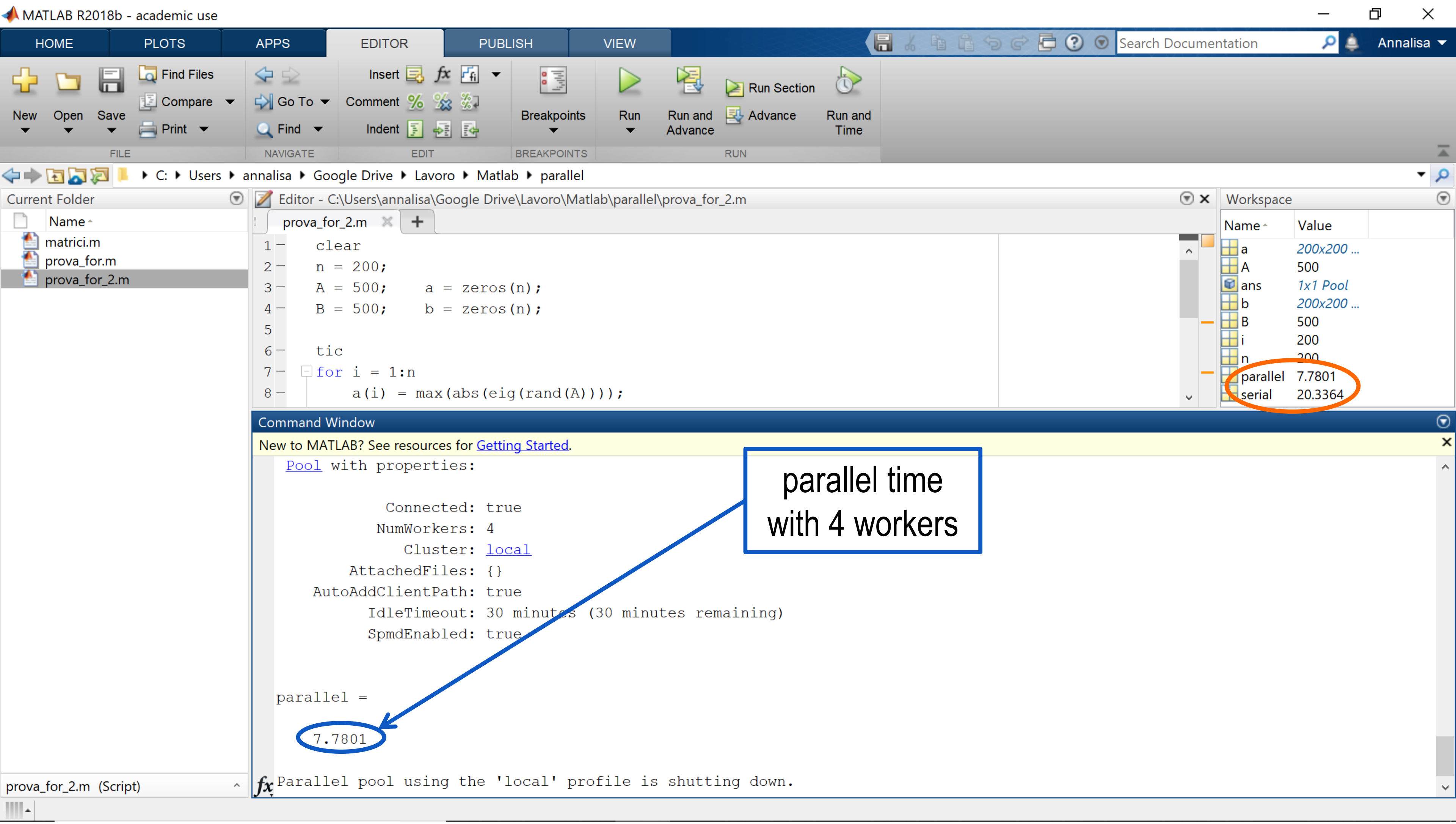
Example


MATLAB R2018b - academic use
HOME PLOTS APPS EDITOR PUBLISH VIEW
Editor - C:\Users\annalisa\Google Drive\Lavoro\Matlab\parallel\prova_for_2.m
clear
n = 200;
A = 500; a = zeros(n);
B = 500; b = zeros(n);
tic
for i = 1:n
a(i) = max(abs(eig(rand(A))));
end
serial=toc
parpool(2)
tic
parfor i = 1:n
b(i) = max(abs(eig(rand(B))));
end
parallel=toc
delete(gcp('nocreate'))
parallel time with 2 workers
Workspace
Name Value
a 200x200 ...
A 500
ans 1x1 Pool
b 200x200 ...
B 500
i 200
n 200
parallel 11.8682
serial 20.4317
Command Window
New to MATLAB? See resources for Getting Started.
parallel =
11.8682
Parallel pool using the 'local' profile is shutting down.

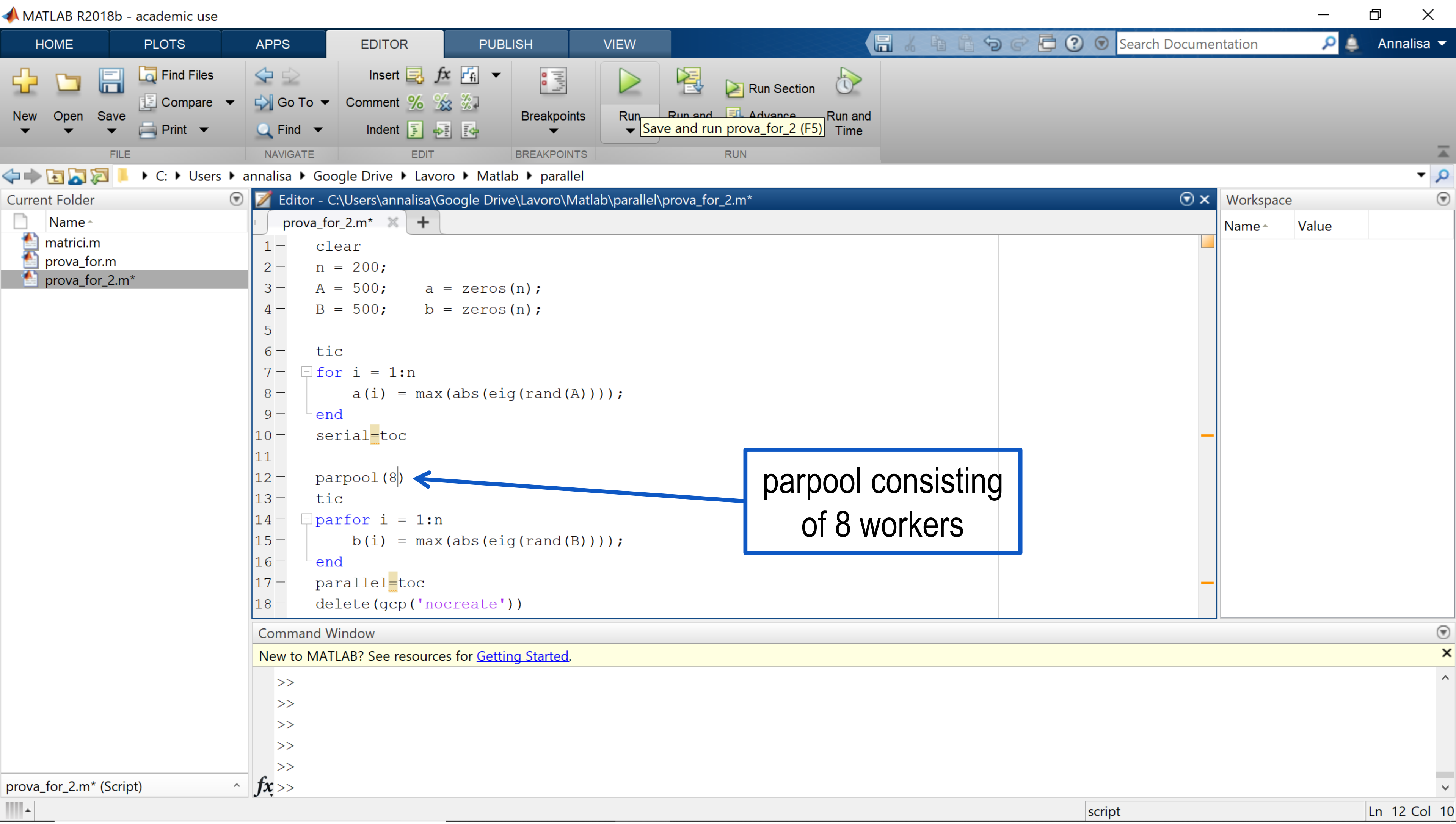
Example


MATLAB R2018b - academic use
HOME PLOTS APPS EDITOR PUBLISH VIEW
Editor - C:\Users\annalisa\Google Drive\Lavoro\Matlab\parallel\prova_for_2.m
prova_for_2.m
clear
n = 200;
A = 500;   a = zeros(n);
B = 500;   b = zeros(n);
tic
for i = 1:n
    a(i) = max(abs(eig(rand(A))));
end
serial=toc
parpool(4)
tic
parfor i = 1:n
    b(i) = max(abs(eig(rand(B))));
end
parallel=toc
delete(gcp('nocreate'))
parpool consisting of 4 workers
Current Folder
matrici.m
prova_for.m
prova_for_2.m
Workspace
Command Window
New to MATLAB? See resources for Getting Started.

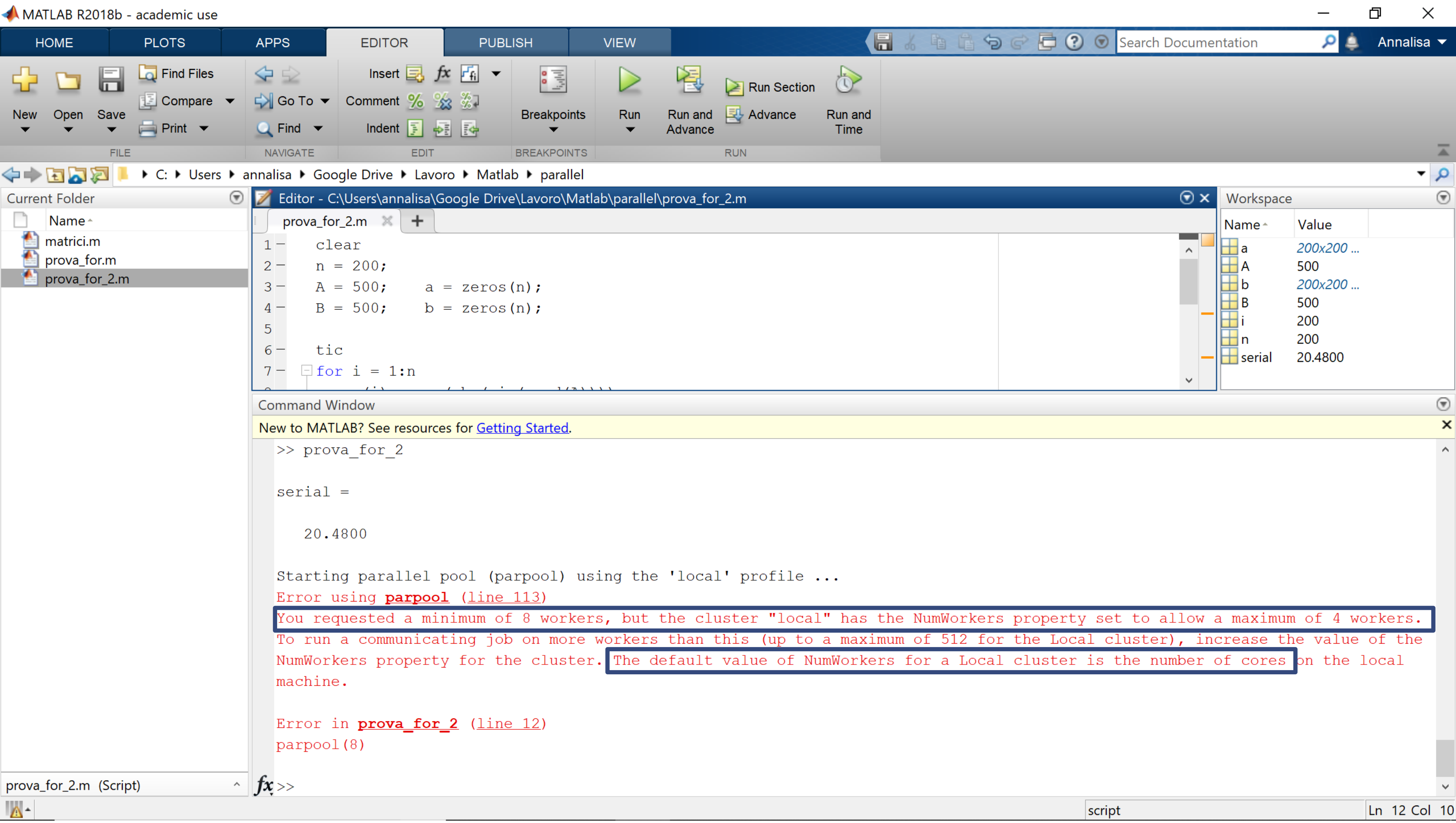
Example


MATLAB R2018b - academic use
HOME
PLOTS
APPS
EDITOR
PUBLISH
VIEW
Search Documentation
Annalisa
Editor - C:\Users\annalisa\Google Drive\Lavoro\Matlab\parallel\prova_for_2.m
prova_for_2.m
clear
n = 200;
A = 500; a = zeros(n);
B = 500; b = zeros(n);
tic
for i = 1:n
a(i) = max(abs(eig(rand(A))));
Current Folder
matrici.m
prova_for.m
prova_for_2.m
Workspace
Name Value
a 200x200 ...
A 500
ans 1x1 Pool
b 200x200 ...
B 500
i 200
n 200
parallel 7.7801
serial 20.3364
Command Window
New to MATLAB? See resources for Getting Started.
Pool with properties:
Connected: true
NumWorkers: 4
Cluster: local
AttachedFiles: {}
AutoAddClientPath: true
IdleTimeout: 30 minutes (30 minutes remaining)
SpmdEnabled: true
parallel =
7.7801
Parallel pool using the 'local' profile is shutting down.
parallel time with 4 workers

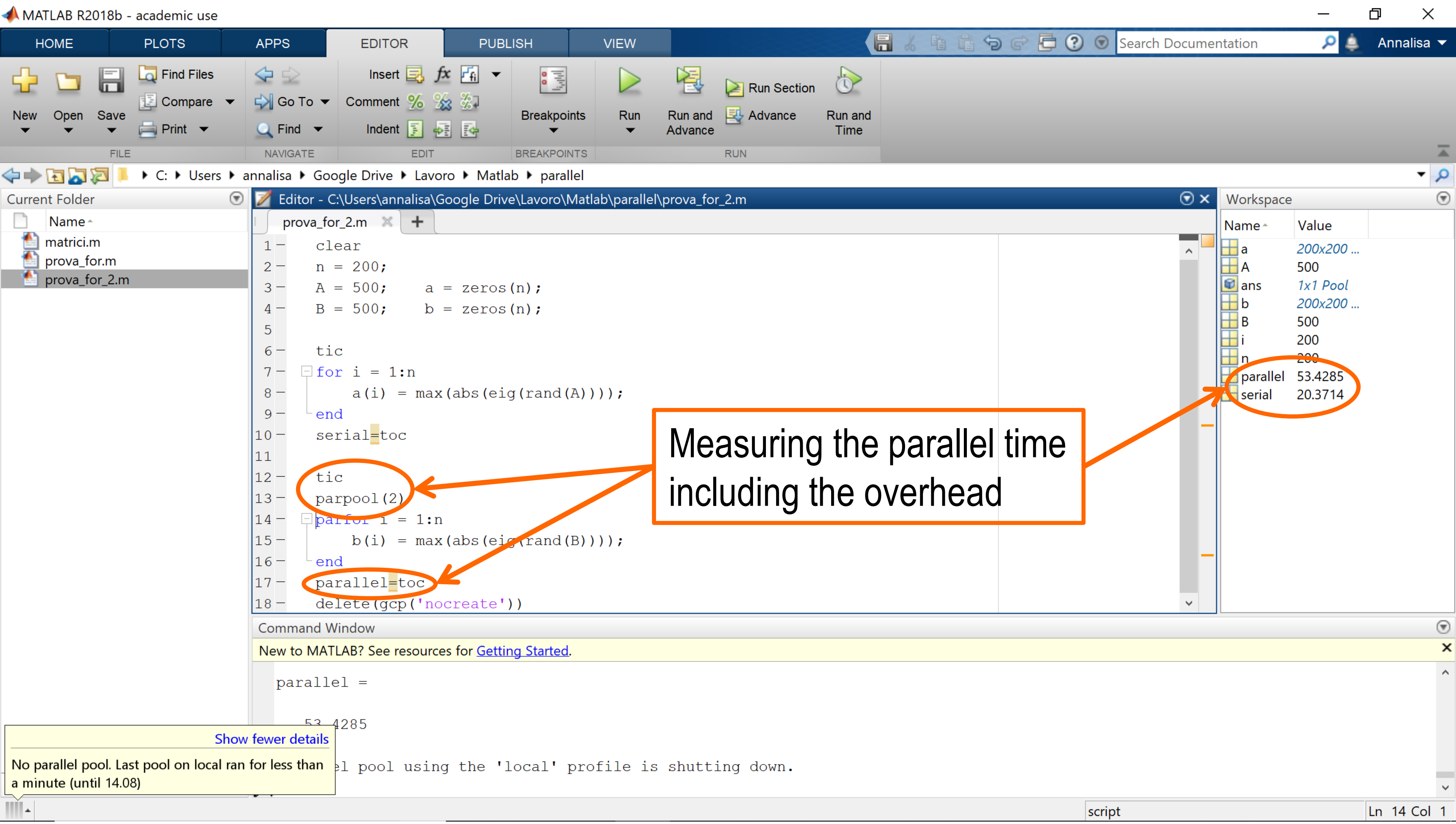
Example


MATLAB R2018b - academic use
HOME PLOTS APPS EDITOR PUBLISH VIEW
Search Documentation
Annalisa
Save and run prova_for_2 (F5)
Editor - C:\Users\annalisa\Google Drive\Lavoro\Matlab\parallel\prova_for_2.m*
prova_for_2.m*
clear
n = 200;
A = 500;   a = zeros(n);
B = 500;   b = zeros(n);
tic
for i = 1:n
a(i) = max(abs(eig(rand(A))));
end
serial=toc
parpool(8)
tic
parfor i = 1:n
b(i) = max(abs(eig(rand(B))));
end
parallel=toc
delete(gcp('nocreate'))
parpool consisting of 8 workers
Current Folder
matrici.m
prova_for.m
prova_for_2.m*
Workspace
Command Window
New to MATLAB? See resources for Getting Started.

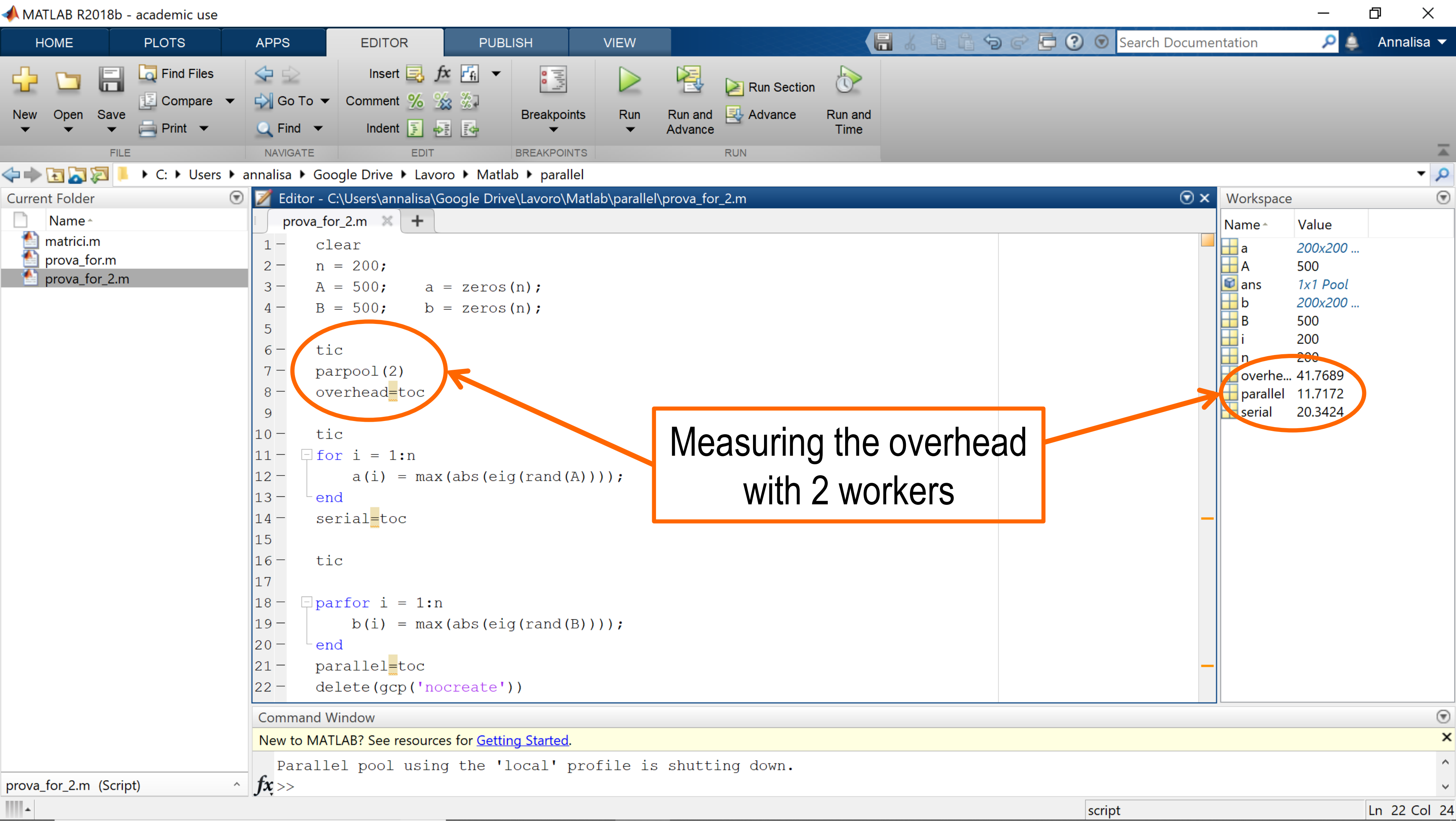
Example


MATLAB R2018b - academic use
HOME
PLOTS
APPS
EDITOR
PUBLISH
VIEW
Search Documentation
Annalisa
New
Open
Save
Find Files
Compare
Print
FILE
Go To
Find
NAVIGATE
Insert
Comment
Indent
EDIT
Breakpoints
BREAKPOINTS
Run
Run and Advance
Run Section
Advance
Run and Time
RUN
C: Users annalisa Google Drive Lavoro Matlab parallel
Current Folder
Name
matrici.m
prova_for.m
prova_for_2.m
Editor - C:\Users\annalisa\Google Drive\Lavoro\Matlab\parallel\prova_for_2.m
prova_for_2.m
clear
n = 200;
A = 500; a = zeros(n);
B = 500; b = zeros(n);
tic
for i = 1:n
Workspace
Name
Value
a 200x200 ...
A 500
b 200x200 ...
B 500
i 200
n 200
serial 20.4800
Command Window
New to MATLAB? See resources for Getting Started.
>> prova_for_2
serial =
20.4800
Starting parallel pool (parpool) using the 'local' profile ...
Error using parpool (line 113)
You requested a minimum of 8 workers, but the cluster "local" has the NumWorkers property set to allow a maximum of 4 workers.
To run a communicating job on more workers than this (up to a maximum of 512 for the Local cluster), increase the value of the
NumWorkers property for the cluster. The default value of NumWorkers for a Local cluster is the number of cores on the local
machine.
Error in prova_for_2 (line 12)
parpool(8)
prova_for_2.m (Script)
script
Ln 12 Col 10

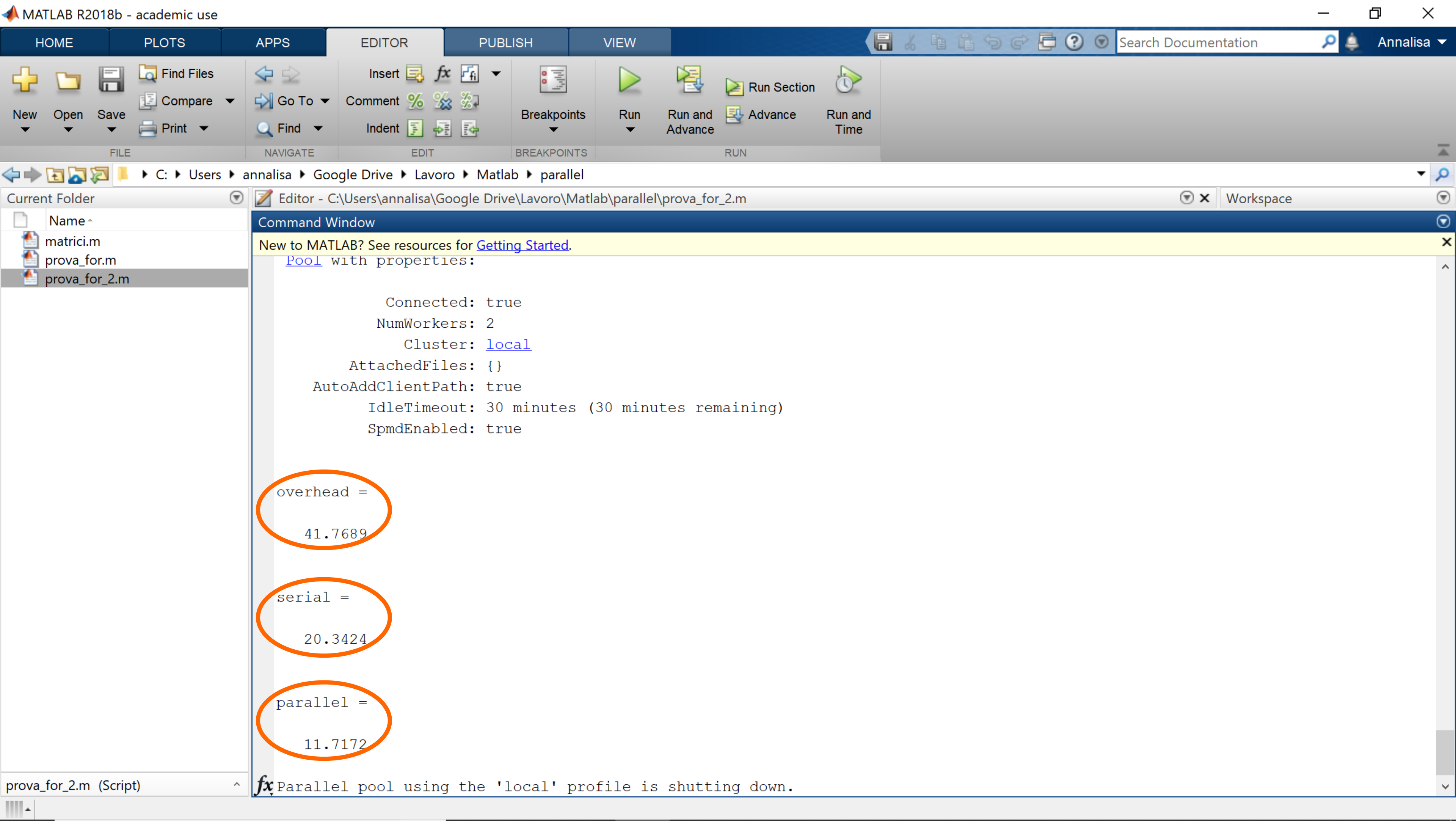
Example


MATLAB R2018b - academic use
HOME
PLOTS
APPS
EDITOR
PUBLISH
VIEW
Search Documentation
Annalisa
New
Open
Save
Find Files
Compare
Print
FILE
Go To
Find
NAVIGATE
Insert
Comment
Indent
EDIT
Breakpoints
BREAKPOINTS
Run
Run and Advance
Run Section
Advance
Run and Time
RUN
C: Users annalisa Google Drive Lavoro Matlab parallel
Current Folder
Name
matrici.m
prova_for.m
prova_for_2.m
Editor - C:\Users\annalisa\Google Drive\Lavoro\Matlab\parallel\prova_for_2.m
prova_for_2.m
clear
n = 200;
A = 500; a = zeros(n);
B = 500; b = zeros(n);
tic
for i = 1:n
a(i) = max(abs(eig(rand(A))));
end
serial=toc
tic
parpool(2)
parfor i = 1:n
b(i) = max(abs(eig(rand(B))));
end
parallel=toc
delete(gcp('nocreate'))
Measuring the parallel time including the overhead
Workspace
Name
Value
a 200x200 ...
A 500
ans 1x1 Pool
b 200x200 ...
B 500
i 200
n 200
parallel 53.4285
serial 20.3714
Command Window
New to MATLAB? See resources for Getting Started.
parallel =
53.4285
Show fewer details
No parallel pool. Last pool on local ran for less than a minute (until 14.08)
l pool using the 'local' profile is shutting down.
script
Ln 14 Col 1

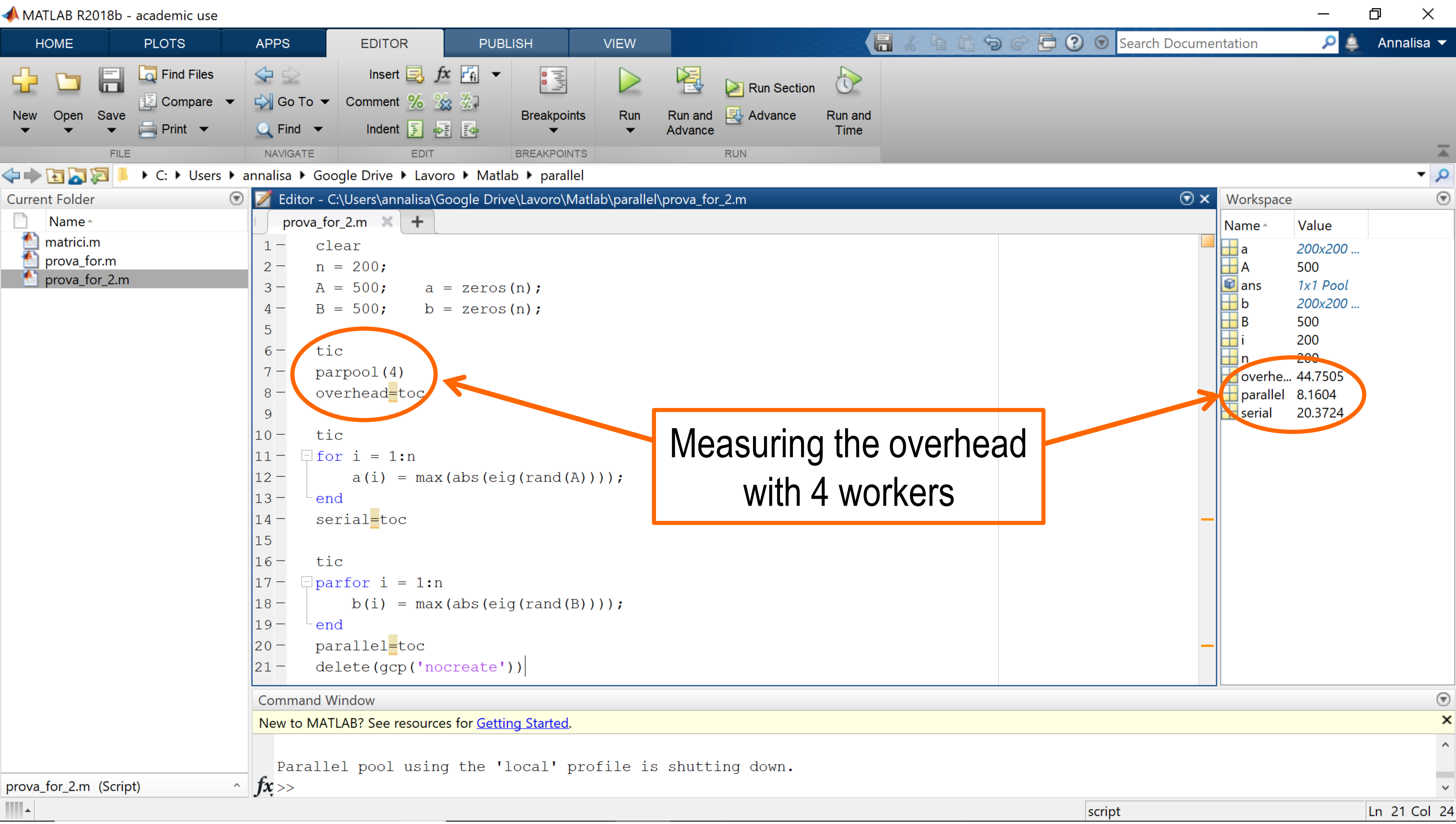
Example


MATLAB R2018b - academic use
HOME PLOTS APPS EDITOR PUBLISH VIEW
Search Documentation
Annalisa
New Open Save Find Files Compare Print
Go To Find
Insert Comment Indent
Breakpoints
Run Run and Advance Run Section Advance Run and Time
FILE NAVIGATE EDIT BREAKPOINTS RUN
C: Users annalisa Google Drive Lavoro Matlab parallel
Current Folder
Name
matrici.m
prova_for.m
prova_for_2.m
Editor - C:\Users\annalisa\Google Drive\Lavoro\Matlab\parallel\prova_for_2.m
prova_for_2.m
clear
n = 200;
A = 500; a = zeros(n);
B = 500; b = zeros(n);
tic
parpool(2)
overhead=toc
tic
for i = 1:n
a(i) = max(abs(eig(rand(A))));
end
serial=toc
tic
parfor i = 1:n
b(i) = max(abs(eig(rand(B))));
end
parallel=toc
delete(gcp('nocreate'))
Measuring the overhead with 2 workers
Workspace
Name Value
a 200x200 ...
A 500
ans 1x1 Pool
b 200x200 ...
B 500
i 200
n 200
overhe... 41.7689
parallel 11.7172
serial 20.3424
Command Window
New to MATLAB? See resources for Getting Started.
Parallel pool using the 'local' profile is shutting down.
prova_for_2.m (Script)
script
Ln 22 Col 24

Example


MATLAB R2018b - academic use
HOME PLOTS APPS EDITOR PUBLISH VIEW
Search Documentation
Annalisa
New Open Save Find Files Compare Print
Go To Find
Insert Comment Indent
Breakpoints
Run Run and Advance Run Section Advance Run and Time
FILE NAVIGATE EDIT BREAKPOINTS RUN
C: Users annalisa Google Drive Lavoro Matlab parallel
Current Folder
Name
matrici.m
prova_for.m
prova_for_2.m
prova_for_2.m (Script)
Editor - C:\Users\annalisa\Google Drive\Lavoro\Matlab\parallel\prova_for_2.m
Workspace
Command Window
New to MATLAB? See resources for Getting Started.
Pool with properties:
Connected: true
NumWorkers: 2
Cluster: local
AttachedFiles: {}
AutoAddClientPath: true
IdleTimeout: 30 minutes (30 minutes remaining)
SpmdEnabled: true
overhead =
41.7689
serial =
20.3424
parallel =
11.7172
Parallel pool using the 'local' profile is shutting down.

Example


MATLAB R2018b - academic use
HOME PLOTS APPS EDITOR PUBLISH VIEW
Editor - C:\Users\annalisa\Google Drive\Lavoro\Matlab\parallel\prova_for_2.m
prova_for_2.m
clear
n = 200;
A = 500;   a = zeros(n);
B = 500;   b = zeros(n);
tic
parpool(4)
overhead=toc
tic
for i = 1:n
a(i) = max(abs(eig(rand(A))));
end
serial=toc
tic
parfor i = 1:n
b(i) = max(abs(eig(rand(B))));
end
parallel=toc
delete(gcp('nocreate'))
Current Folder
matrici.m
prova_for.m
prova_for_2.m
Workspace
Name Value
a 200x200 ...
A 500
ans 1x1 Pool
b 200x200 ...
B 500
i 200
n 200
overhe... 44.7505
parallel 8.1604
serial 20.3724
Measuring the overhead with 4 workers
Command Window
Parallel pool using the 'local' profile is shutting down.

spmd

- **S**ingle **P**rogram **M**ultiple **D**ata model

- Used to create parallel regions of code
- Values returning from the body of an **`spmd`** statement are converted to Composite objects
- A Composite object contains references to the values stored on the remote MATLAB workers, and those values can be retrieved using cell-array indexing
- The actual data on the workers remains available on the workers for subsequent **`spmd`** execution, so long as the Composite exists on the client and the parallel pool remains open

spmd

- `spmd` distributes the array among MATLAB workers (each worker contains a part of the array) but can still operate on entire array as 1 entity
- Inside the body of the `spmd` statement, each MATLAB worker has:
  - a unique value of labindex,
  - the total number of workers numlabs executing the block in parallel
- Data automatically transferred between workers when necessary
- Within the body of the spmd statement, communication functions for communicating jobs (such as labSend and labReceive) can transfer data between the workers

Spmd Format

- Format

```
parpool (4)
spmd
  statements
end
```

- Simple Example

```
parpool(4)
spmd
  j=zeros(1e7,1);
end;
```

Spmd Examples

- Result j is a Composite with 4 parts!

j <1x4 Composite>

```
j =

     1: class = double, size = [10000000         1]
     2: class = double, size = [10000000         1]
     3: class = double, size = [10000000         1]
     4: class = double, size = [10000000         1]
```

MATLAB Composites

- A Composite is an object used for data distribution in MATLAB
- A Composite object has one entry for each worker
  - parpool(12) creates 12X1 composite
  - parpool(6) creates 6X1 composite
- You can create a composite in two ways:
  - spmd
  - c = Composite();
    - This creates a composite that does not contain any data, just placeholders for data
    - Also, one element per parpool worker is created for the composite
    - Use smpd or indexing to populate a composite created this way

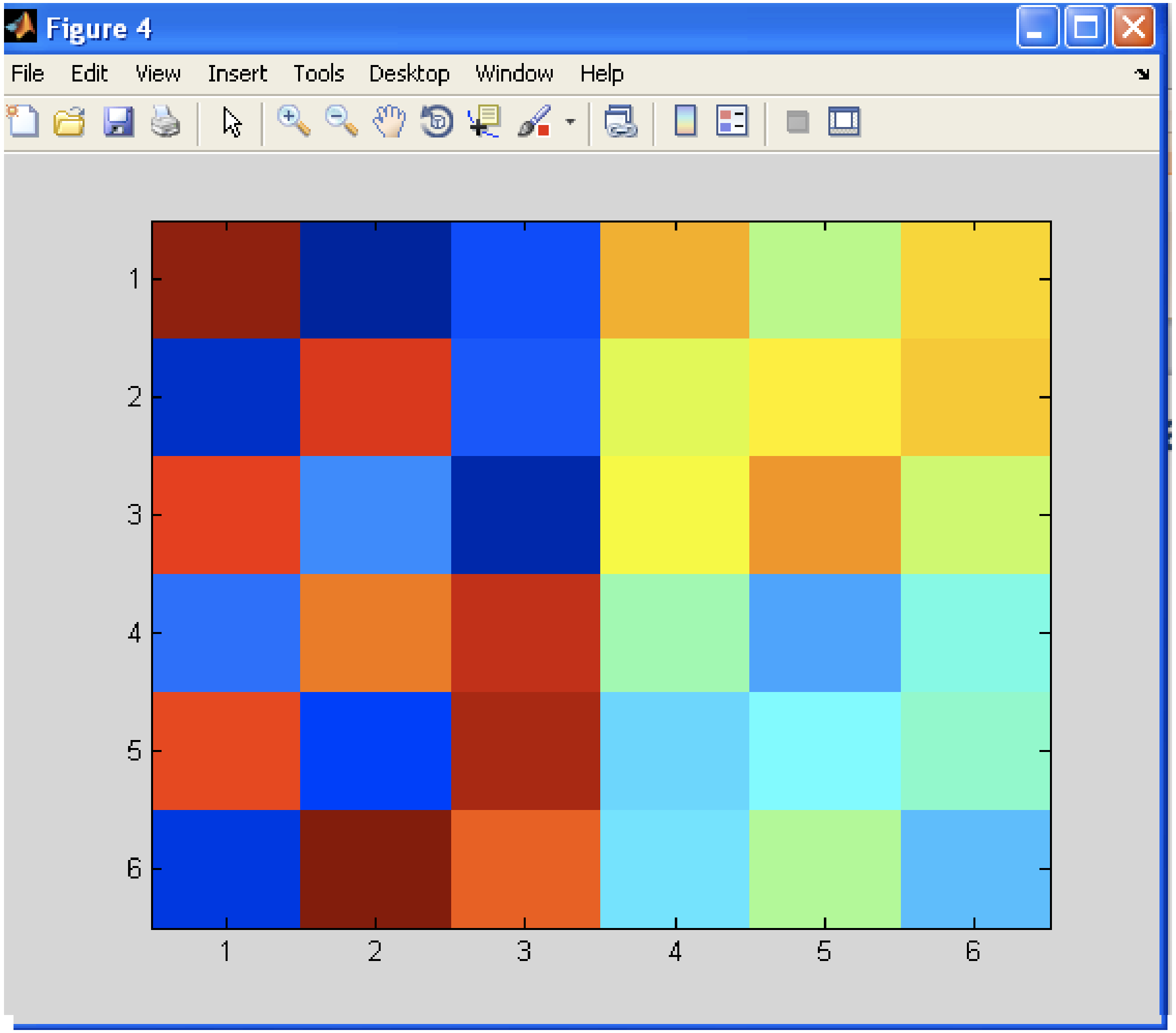
Another spmd Example - creating graphs

```
%Perform a simple calculation in parallel, and plot the
results:
 parpool(4)
 spmd
   % build magic squares in parallel
   q = magic(labindex + 2);
   % labindex - index of the lab/worker (e.g. 1)
 end

 for ii=1:length(q)
   % plot each magic square
   figure, imagesc(q{ii}); %plot a matrix as an image
 end
 delete (gcp('nocreate'));
```

Another spmd Example- creating graphs

- Results



parfor vs spmd

- parfor is simpler to use
- parfor can't control iterations
- parfor only does loops

- spmd more control over iterations
- spmd more control over data movement
- spmd is persistent
- spmd is more flexible and you can create parallel regions that do more than just loop

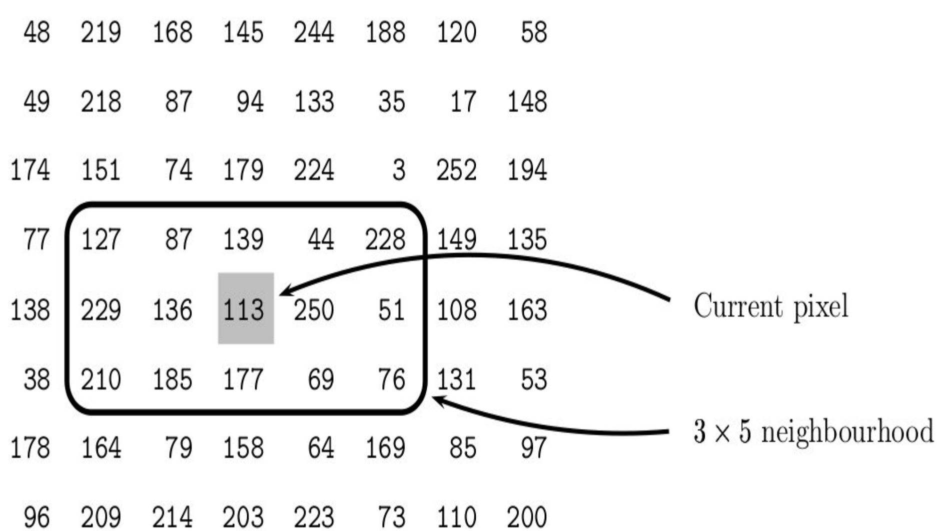
Built-in Multithreading

- Operations in the algorithm carried out by the function are easily partitioned into sections that can be executed concurrently, and with little communication or few sequential operations required
- Data size is large enough so that any advantages of concurrent execution outweigh the time required to partition the data and manage separate execution threads. For example, *most functions speed up only when the array is greater than several thousand elements*.
- Operation is not memory-bound where the processing time is dominated by memory access time. As a general rule, more complex functions speed up better than simple functions.
- *http://www.mathworks.com/matlabcentral/answers/95958-which-matlab-functions-benefit-from-multithreaded-computation*

IMAGES AND MATLAB

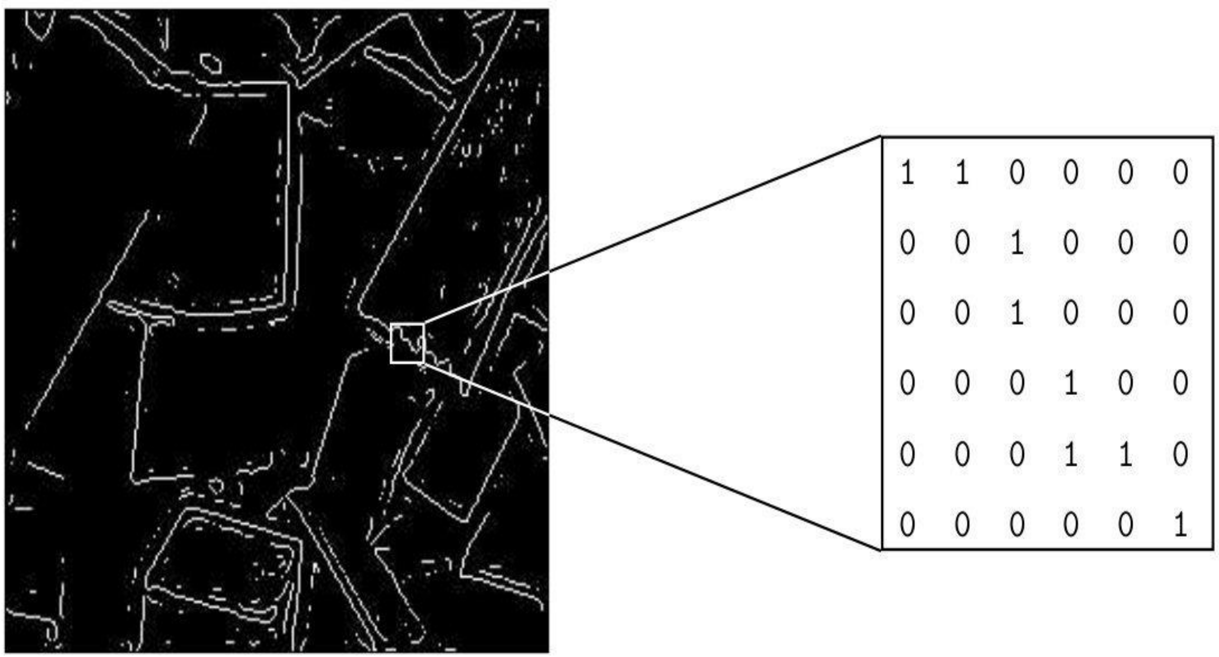
Images

- A digital image can be considered as a large array of discrete dots, each of which has a brightness associated with it
- These dots are called picture elements or more simply **pixels**
- The pixels surrounding a given pixel constitute its **neighborhood**
- A neighborhood can be characterized by its shape in the same way as a matrix: 3x3 neighborhood, 5x7 neighborhood…



Types of digital image

- **Binary:** Each pixel is just **black** or **white**. Since there are only two possible values for each pixel (0,1), we only need **one bit** per pixel



Types of digital image

- **Grayscale:** Each pixel is a shade of gray, normally from **0** (black) to **255** (white), that is each pixel can be represented exactly **one byte**
- Other greyscale ranges can be used, generally power of **2**

230	229	232	234	235	232	148
237	236	236	234	233	234	152
255	255	255	251	230	236	161
99	90	67	37	94	247	130
222	152	255	129	129	246	132
154	199	255	150	189	241	147
216	132	162	163	170	239	122

Types of digital image

- **True Color**, or **RGB**: Each pixel has a particular color, described by the amount of **red**, **green** and **blue**
- Each components has a range 0–255, for a total of **256^3** different possible colors
- **Three matrices** representing the **red**, **green** and **blue** values for each pixel

49	55	56	57	52	53
58	60	60	58	55	57
58	58	54	53	55	56
83	78	72	69	68	69
88	91	91	84	83	82
69	76	83	78	76	75
61	69	73	78	76	76

Red

64	76	82	79	78	78
93	93	91	91	86	86
88	82	88	90	88	89
125	119	113	108	111	110
137	136	132	128	126	120
105	108	114	114	118	113
96	103	112	108	111	107

Green

66	80	77	80	87	77
81	93	96	99	86	85
83	83	91	94	92	88
135	128	126	112	107	106
141	129	129	117	115	101
95	99	109	108	112	109
84	93	107	101	105	102

Blue

Image Import and Export

- Read and write images in Matlab

```
img = imread('apple.jpg');
dim = size(img);
figure;
imshow(img);
imwrite(img, 'output.bmp', 'bmp');
```

- Alternatives to `imshow`

```
imagesc(I)
imtool(I)
image(I)
```

Image and Matrices

How to build a matrix (or image)?

Intensity Image:

```
row = 256;
col = 256;
img = zeros(row, col);
img(100:105, :) = 0.5;
img(:, 100:105) = 1;
figure;
imshow(img);
```

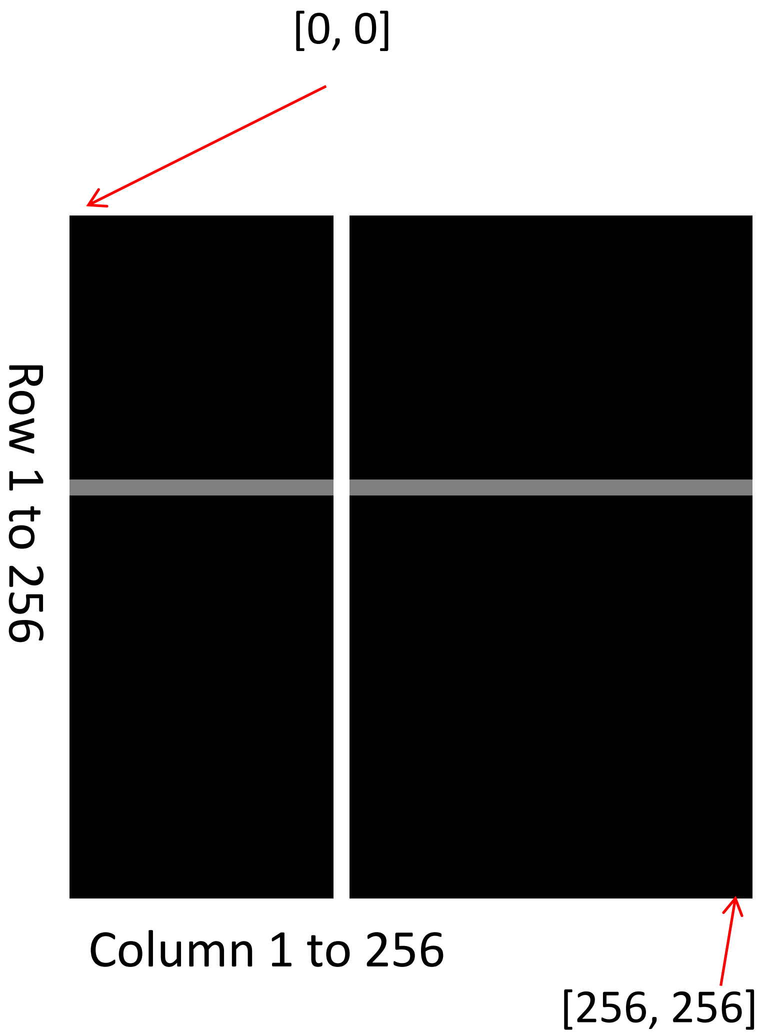


Image and Matrices

Binary Image

```
row = 256;
col = 256;
img = rand(row,
col);
img = round(img);
figure;
imshow(img);
```

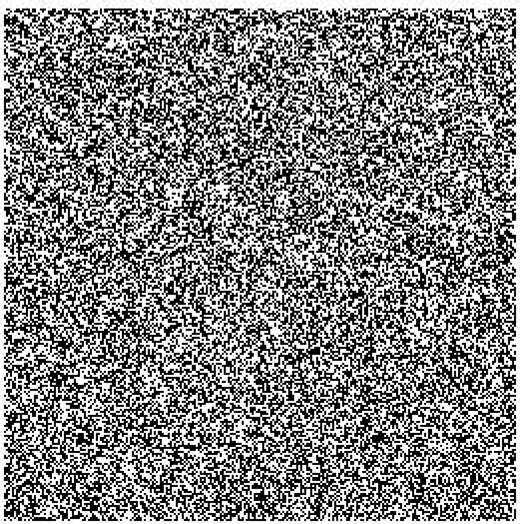


Image display

- `image` - create and display image object
- `imagesc` - scale and display as image
- `imshow` - display image
- `colorbar` - display colorbar
- `getimage` - get image data from axes
- `truesize` - adjust display size of image
- `zoom` - zoom in and zoom out of 2D plot

Image information

`iminfo` returns information about the image

`impixel(i,j)` returns the value of the pixel (i,j)

```
                 Filename: 'aster.tif'
              FileModDate: '13-Mar-2008 16:54:26'
                 FileSize: 17224424.00
                   Format: 'tif'
            FormatVersion: []
                    Width: 4100.00
                   Height: 4200.00
                 BitDepth: 8.00
                ColorType: 'grayscale'
          FormatSignature: [77.00 77.00 0 42.00]
                ByteOrder: 'big-endian'
           NewSubFileType: 0
            BitsPerSample: 8.00
              Compression: 'Uncompressed'
PhotometricInterpretation: 'BlackIsZero'
             StripOffsets: [525x1 double]
          SamplesPerPixel: 1.00
             RowsPerStrip: 8.00
          StripByteCounts: [525x1 double]
              XResolution: 1.00
              YResolution: 1.00
           ResolutionUnit: 'None'
                 Colormap: []
      PlanarConfiguration: 'Chunky'
                TileWidth: []
               TileLength: []
              TileOffsets: []
           TileByteCounts: []
              Orientation: 1.00
                FillOrder: 1.00
         GrayResponseUnit: 0.01
           MaxSampleValue: 255.00
           MinSampleValue: 0
             Thresholding: 1.00
                 Software: 'ERDAS IMAGINE '
             SampleFormat: 'Unsigned integer'
```

Image conversion

- `gray2ind` - intensity image to index image
- `im2bw` - image to binary
- `im2double` - image to double precision
- `im2uint8` - image to 8-bit unsigned integers
- `im2uint16` - image to 16-bit unsigned integers
- `ind2gray` - indexed image to intensity image
- `mat2gray` - matrix to intensity image
- `rgb2gray` - RGB image to grayscale
- `rgb2ind` - RGB image to indexed image

Point Processing: Arithmetic operations

Arithmetic operations act by applying a simple function y=f(x) to each gray value in the image

- Simple functions include **adding** or **subtract** a constant value to each pixel: y = x±C (`imadd`, `imsubtract`)
- **Multiplying** each pixel by a constant: y = C·x (`immultiply`, `imdivide`)
- **Complement**: For a grayscale image is its photographic negative.

Addition

Image: I

Image: I+50

Subtraction

Image: I

Image: I-80

Multiplication

Image: I

Image: I*3

Division

Image: I

Image: I/2

Complement

Image: I

Image: 255-I

Image filtering

- **Filtering** is used to enhance or attenuate some characteric of the image
- **Filtering** modifies the pixels in an image based on some function of a local neighborhood of each pixel

- Filtering generates a new image

- **Linear filtering** (cross-correlation, convolution) replace each pixel by a linear combination of its neighbors

Image filtering

- Linear **filtering** uses a matrix of coefficients **W**
- Imagine **F** is obtained from imagine **I** using **W**:

$$F[x,y] = \sum_{s=-a}^{a} \sum_{t=-b}^{b} W[s,t] I[x+s, y+t]$$

- Where W and the submatrix of I are:

W[-1,-1]	W[-1,0]	W[-1,1]
W[0,-1]	W[0,0]	W[0,1]
W[1,-1]	W[1,0]	W[1,1]

I[x-1,y-1]	I[x-1,y]	I[x-1,y+1]
I[x,y-1]	I[x,y]	I[x,y+1]
I[x+1,y-1]	I[x+1,y]	I[x+1,y+1]

Image filtering

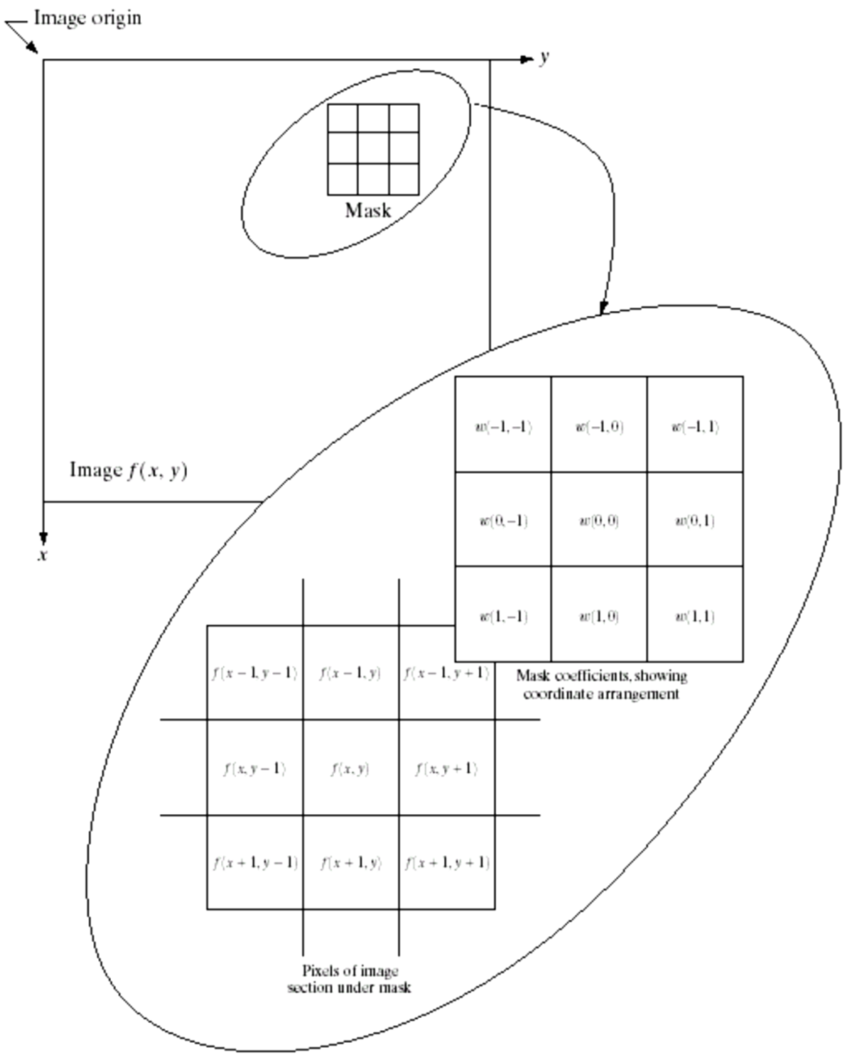
- **Convolution** Same as cross-correlation, except that the kernel is *flipped* (horizontally and vertically)

$$F[x,y]=\sum_{s=-a}^{a}\sum_{t=-b}^{b}W[s,t]I[x-s,y-t]$$

- The prescription for the linear combination - W - is called the **kernel** (or **mask**, or **filter**) of the cross-correlation/convolution

Image filtering

- **Smoothing filters**: mean filter, gaussian filter, median filter
- **Sharpening filters**



Smoothing filter

- **Mean filter**

$$W_{medio} = \frac{1}{a \cdot b} \begin{bmatrix} 1 & 1 & \cdots & 1 \\ 1 & 1 & \cdots & 1 \\ \cdots & \cdots & \cdots & \cdots \\ 1 & 1 & \cdots & 1 \end{bmatrix}$$

Smoothing filter

- **Gaussian filter**: weights of filter follow a gaussian distribution

$$G_\sigma(x,y) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

- Example

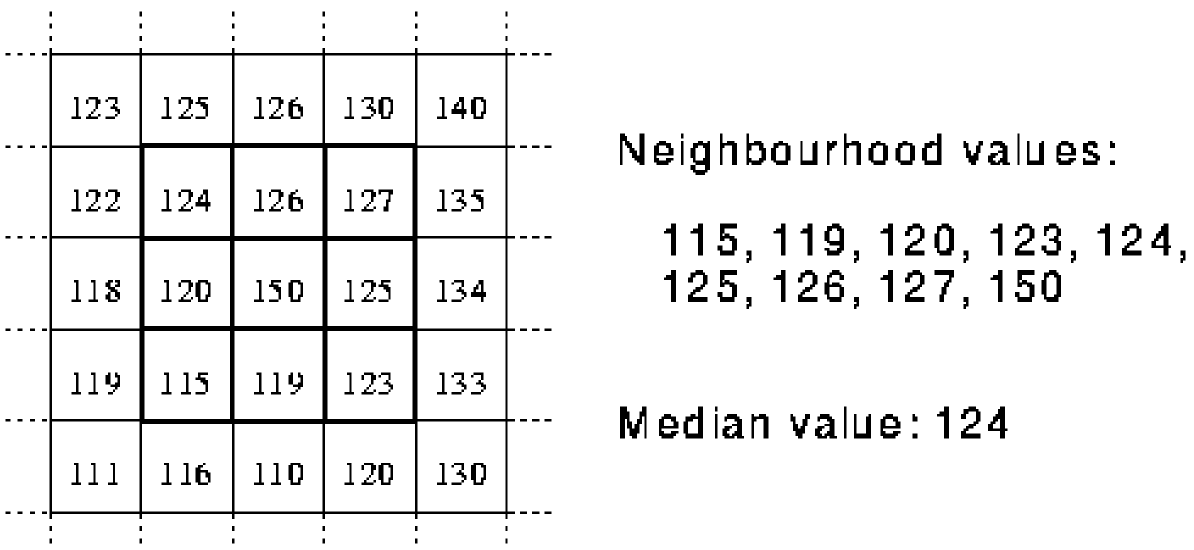
$$G_\sigma = \frac{1}{273}\begin{bmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{bmatrix}$$

Gaussian filter

- Removes high-frequency components from the image (low-pass filter)

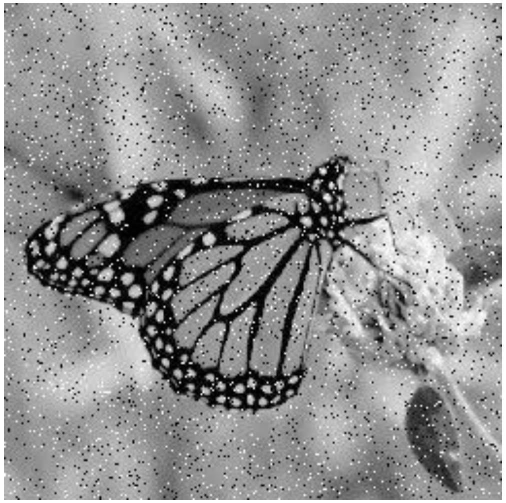
Median filter

The median filter selects a sample from the window, does not average



Median filter

Best suited for *salt and pepper* noise



Sharpening filter

- **Sharpening** filters emphasize fine details in the image, exactly the opposite of the low-pass filter such as Gaussian filter → it just uses a different convolution kernel
- A **high-pass filter** can be used to make an image appear sharper.
- Usually the **central pixel is positive**, whereas adjacent pixels are negative

−1	−1	−1
−1	8	−1
−1	−1	−1

Sharpening filter

- First, I is modified by using a gaussian filter
- Then I_s cis obtained as a linaera combination among image I and the Gauss filtered image, with a suitable value of k usually equal to 1

$$\bar{I}[x,y] = I[x,y] - (G_\sigma * I)[x,y]$$

$$I_s[x,y] = I[x,y] + k\bar{I}[x,y]$$