

# Workshop: Parallel Computing with MATLAB and Scaling to HPCC

**Raymond Norris**  
**MathWorks**

# Outline

- Parallelizing Your MATLAB Code
- Tips for Programming with a Parallel for Loop
- Computing to a GPU
- Scaling to a Cluster
- Debugging and Troubleshooting

# What's Not Being Covered Today?

- Data Parallel
- MapReduce
- MPI
- Simulink

# Let's Define Some Terms

**cli·ent noun \kli-ənt\**

1 : MATLAB session that submits the job

**com·mu·ni·cate job *adjective*  
\kə-'myü-nə-,kāt\ \jäb\**

1 : a job composed of tasks that communicate with each other, running at the same time

**in·de·pen·dent job**

**adjective \ in-də-'pen-  
dənt\ \jäb\**

1 : a job composed of independent tasks, with no communication, which do not need to run at the same time

**lab noun \lab\**

1 : see worker

## ...a Few More Terms

**MAT·LAB pool noun \mat-lab\ \'pül\**

1 : a collection of workers

**MDCS abbreviation**

1 : MATLAB Distributed Computing Server

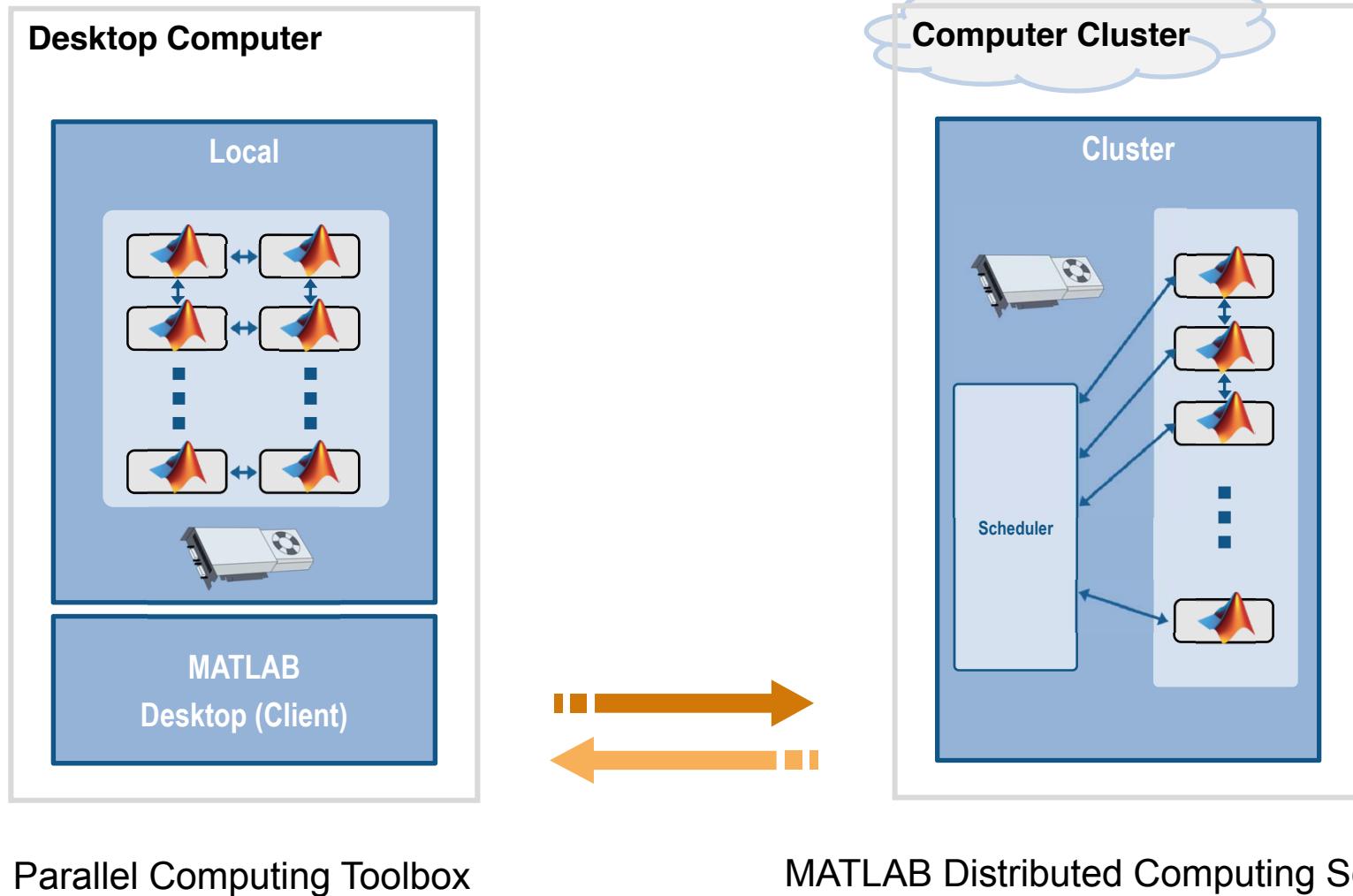
**SPMD abbreviation**

1 : Single Program Multiple Data

**worker noun \'wər-kər\**

1 : headless MATLAB session that performs tasks

# MATLAB Parallel Computing Solution



# Typical Parallel Applications

- Massive for loops (`parfor`)
    - Parameter sweep
      - Many iterations
      - Long iterations
    - Monte-Carlo simulations
    - Test suites
  - One-Off Batch Jobs
- Task Parallel Applications
- 
- Partition Large Data Sets (`spmd`)
- Data Parallel Applications

# Outline

- Parallelizing Your MATLAB Code
- Tips for Programming with a Parallel for Loop
- Computing to a GPU
- Scaling to a Cluster
- Debugging and Troubleshooting

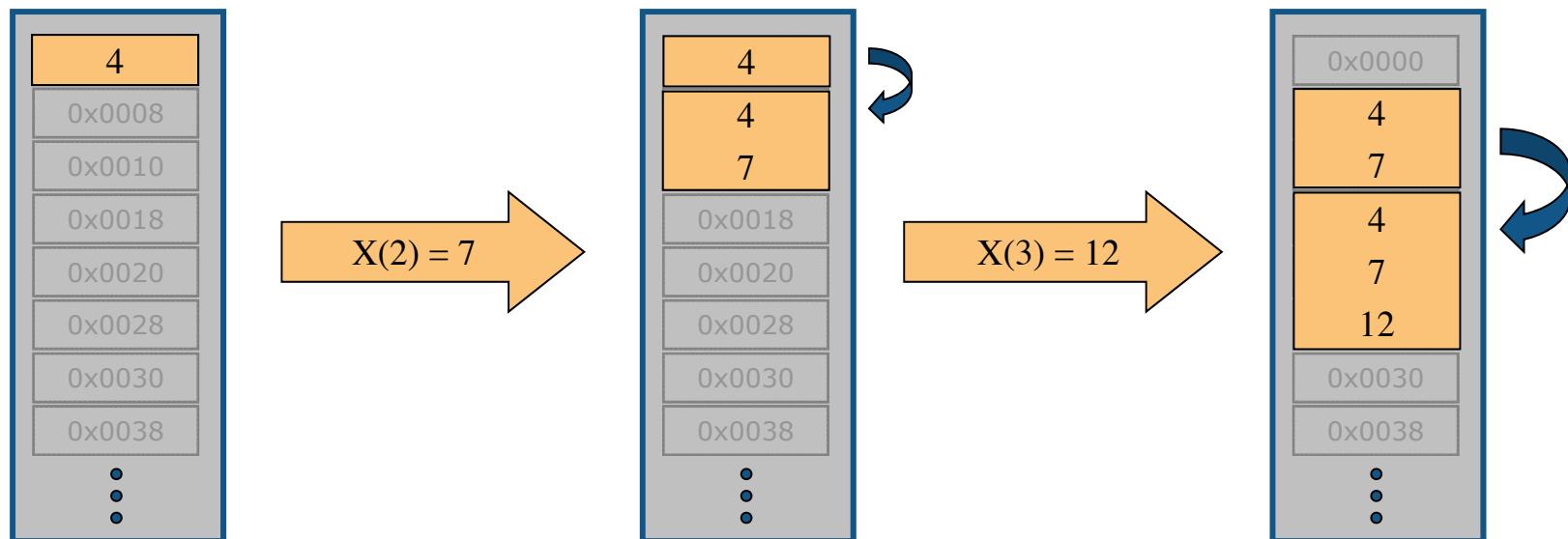
## But Before We Get Started...

- Do you preallocate your matrices?

# Effect of Not Preallocating Memory

```
>> x = 4;  
>> x(2) = 7;  
>> x(3) = 12;
```

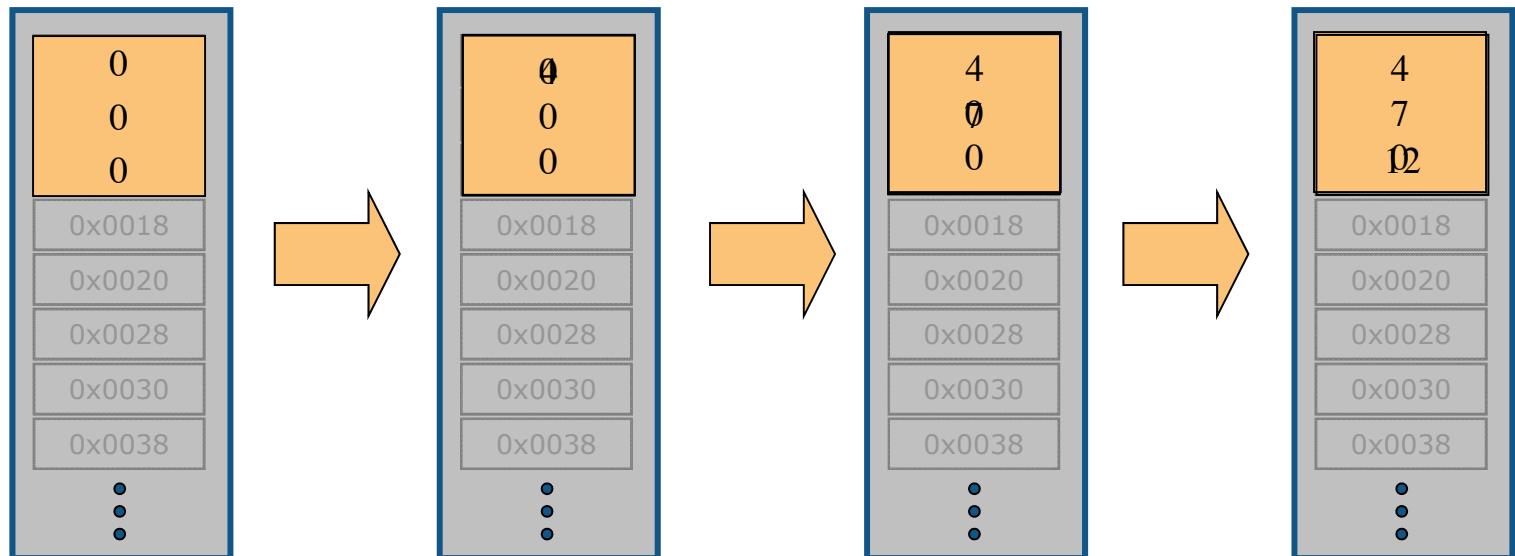
Resizing  
Arrays is  
Expensive



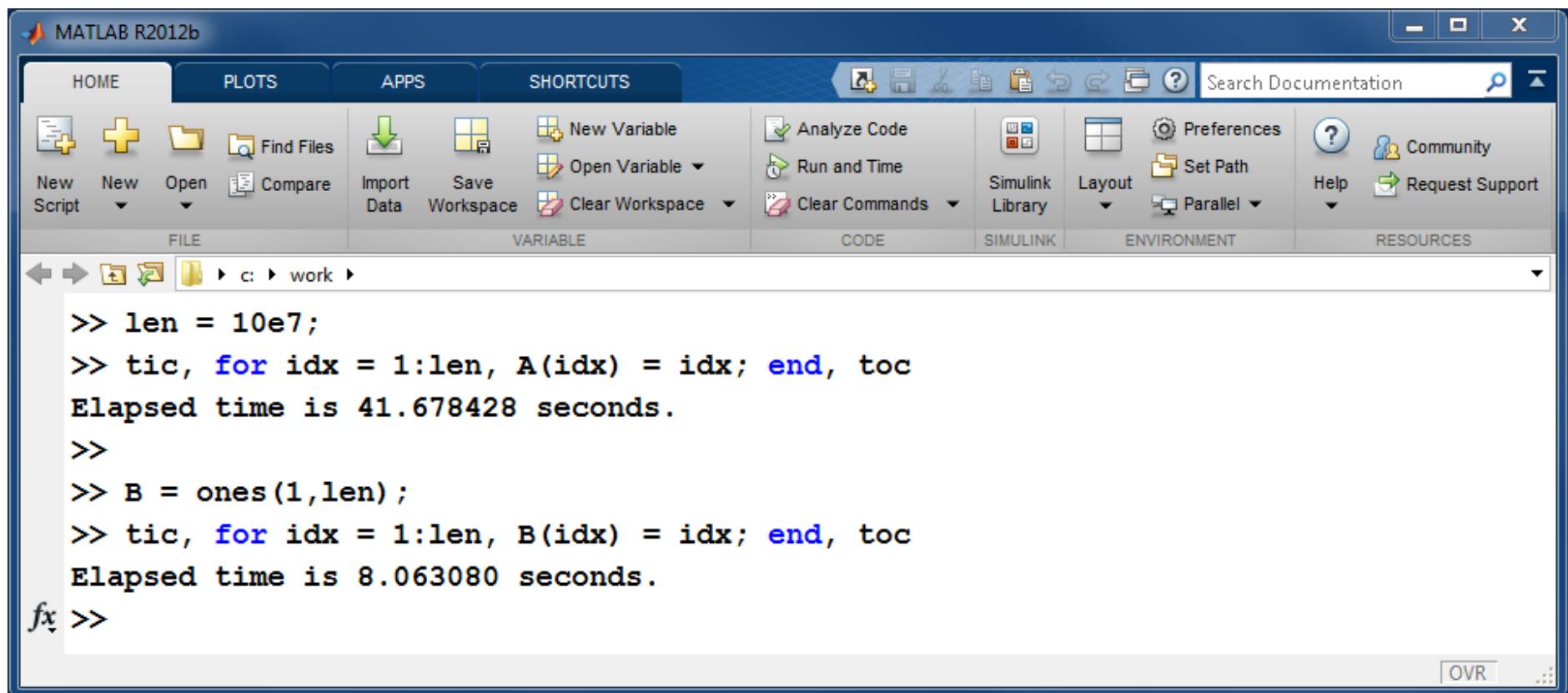
# Benefit of Preallocation

```
>> x = zeros(3,1);
>> x(1) = 4;
>> x(2) = 7;
>> x(3) = 12;
```

Reduced  
Memory  
Operations



# Let's Try It...

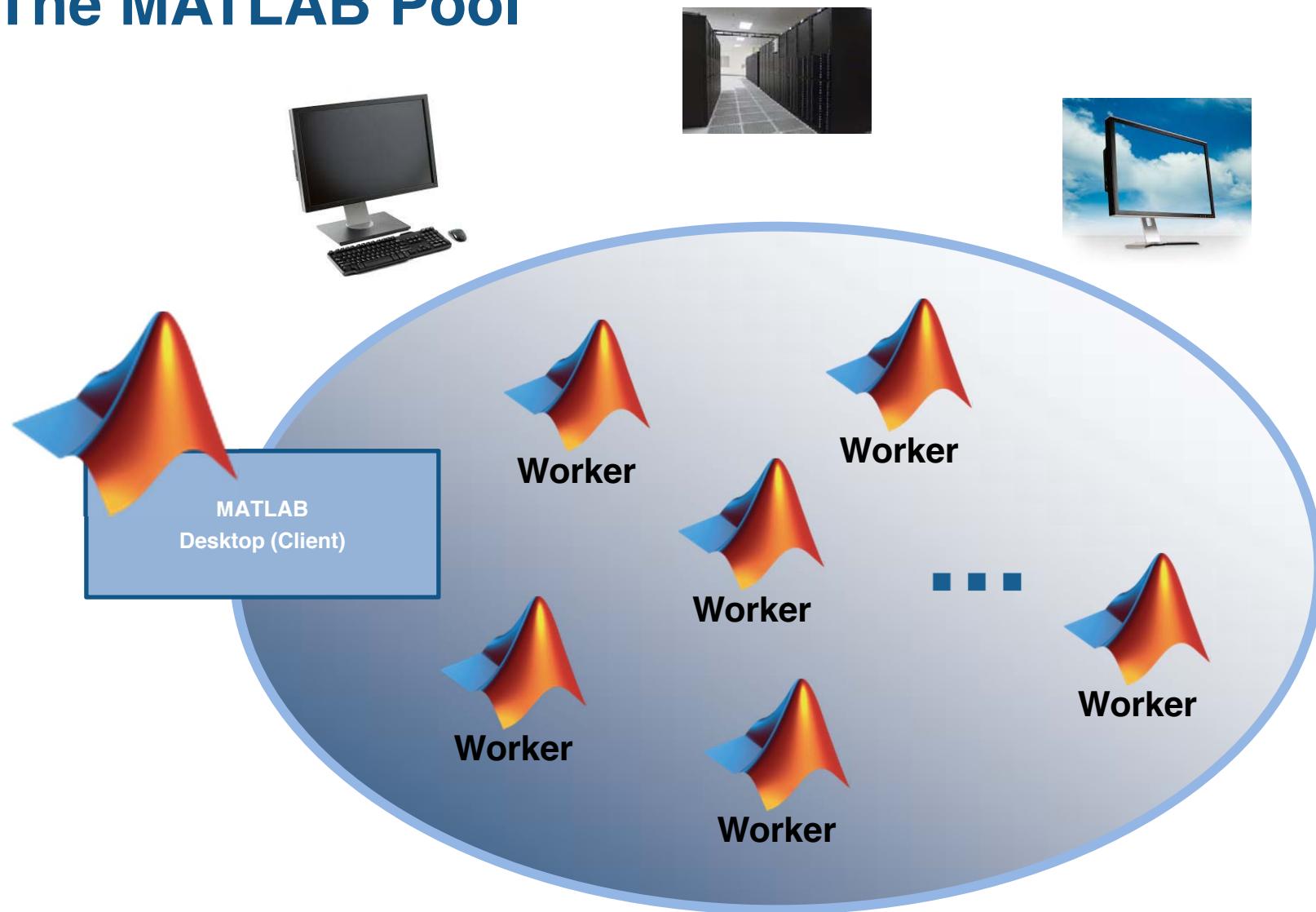


The screenshot shows the MATLAB R2012b interface. The command window displays the following code and its execution results:

```
>> len = 10e7;
>> tic, for idx = 1:len, A(idx) = idx; end, toc
Elapsed time is 41.678428 seconds.
>>
>> B = ones(1,len);
>> tic, for idx = 1:len, B(idx) = idx; end, toc
Elapsed time is 8.063080 seconds.
fx >>
```

# Getting Started With the MATLAB Pool

# The MATLAB Pool



# Connecting to HPCC to Run MATLAB

```
ssh -X USERNAME@hpc-login1.usc.edu
```

```
## For bash users
% cp ~matlab/setup_matlab.sh ~/
% source setup_matlab.sh
```

```
## For tcsh users
% cp ~matlab/setup_matlab.csh ~/
% source setup_matlab.csh
```

```
% matlab_local ## or matlab_cluster
```

```
ssh -X COMPUTE-NODE
. /usr/usc/matlab/2013a/setup.[c]sh
% matlab &
```

Only for today's  
seminar

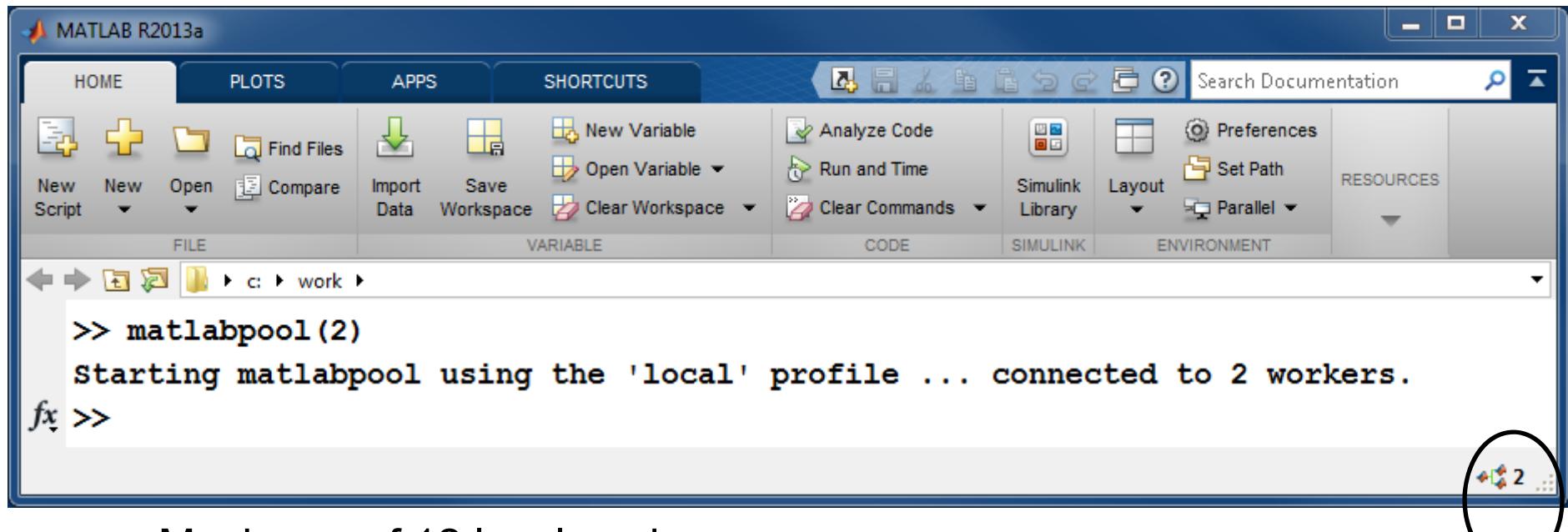
To be updated on  
the Wiki

# Starting a MATLAB Pool...

Bring up the Windows Task Manager or Linux *top*

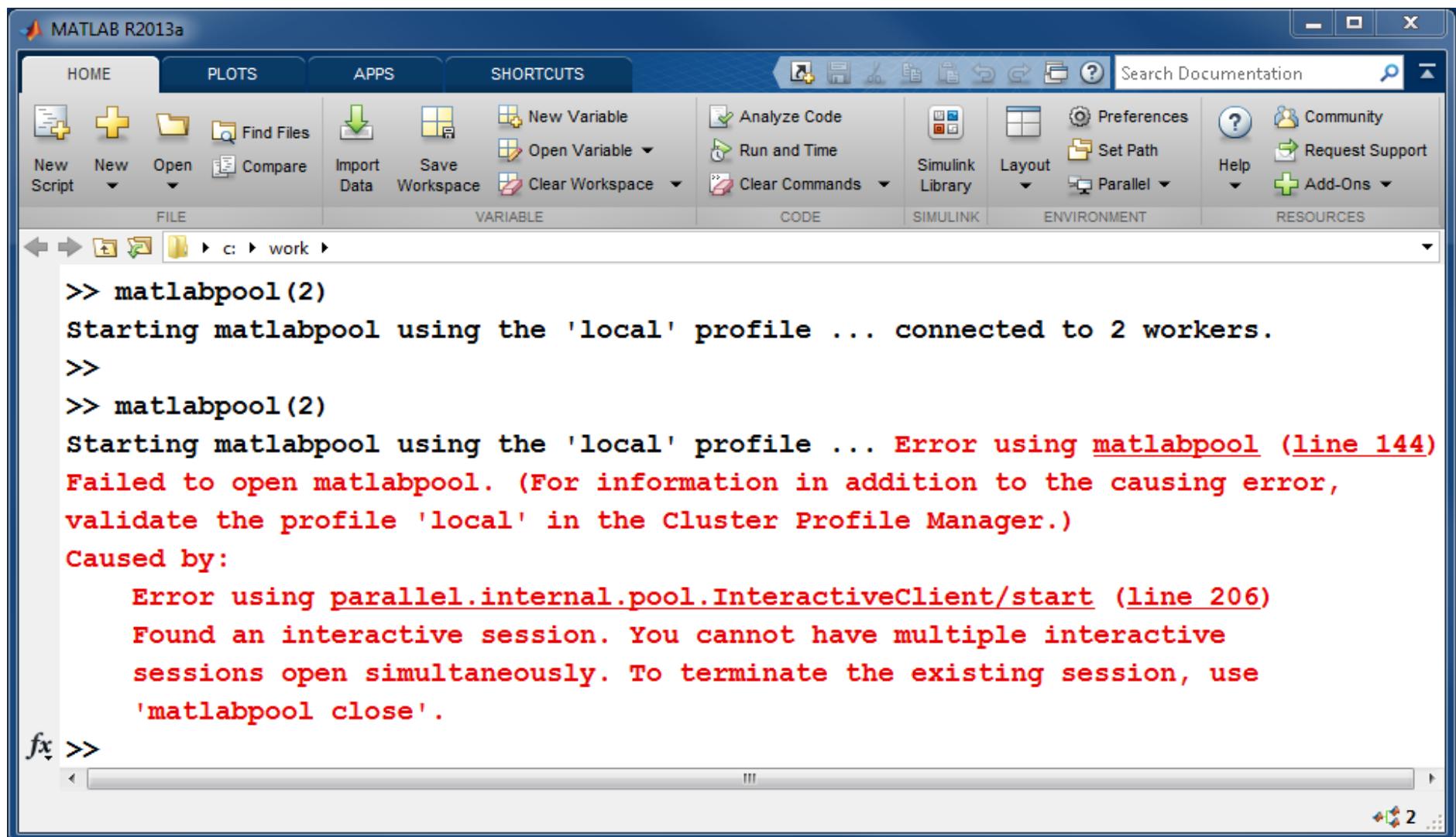
Start MATLAB

Open a MATLAB pool with two workers using the local profile



Maximum of 12 local workers

# One MATLAB Pool at a Time

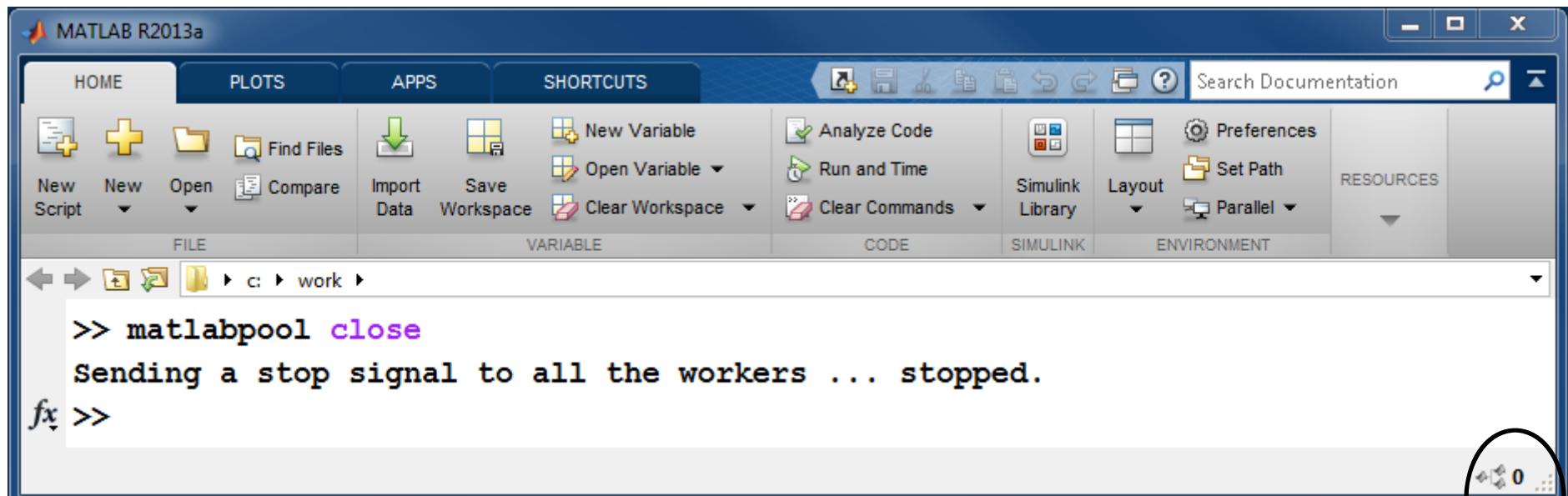


The screenshot shows the MATLAB R2013a interface with the command window open. The command window displays the following text:

```
>> matlabpool(2)
Starting matlabpool using the 'local' profile ... connected to 2 workers.
>>
>> matlabpool(2)
Starting matlabpool using the 'local' profile ... Error using matlabpool (line 144)
Failed to open matlabpool. (For information in addition to the causing error,
validate the profile 'local' in the Cluster Profile Manager.)
Caused by:
    Error using parallel.internal.pool.InteractiveClient/start (line 206)
    Found an interactive session. You cannot have multiple interactive
    sessions open simultaneously. To terminate the existing session, use
    'matlabpool close'.
fx >>
```

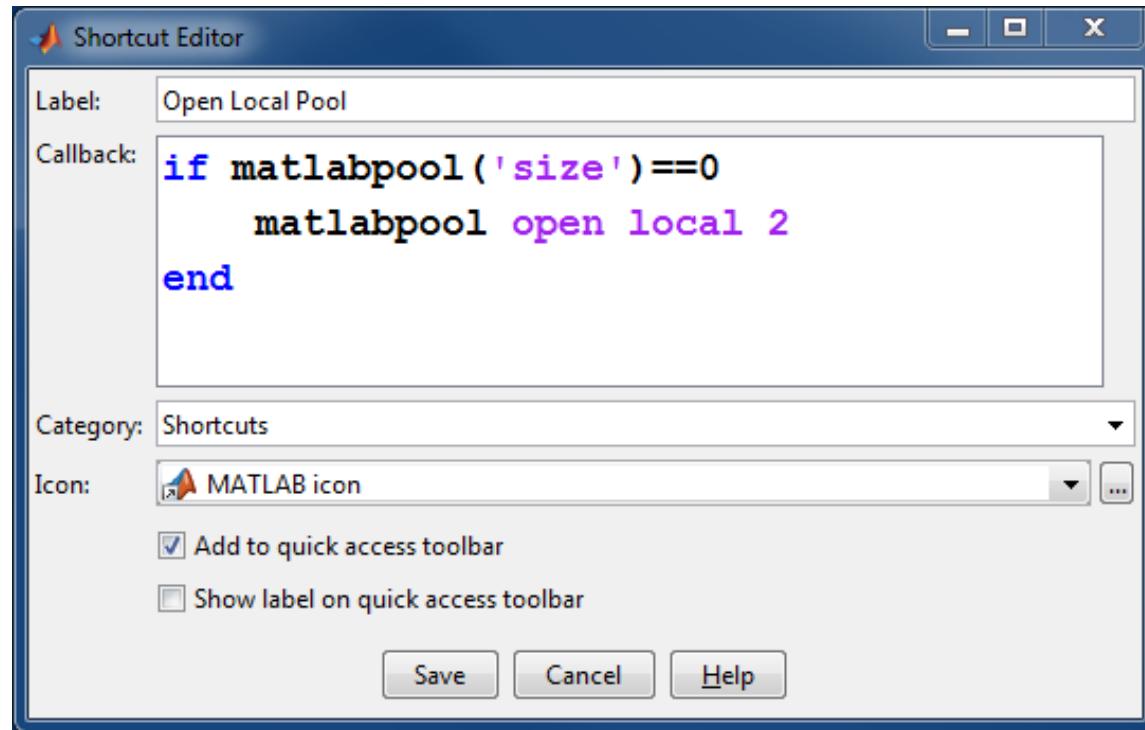
Even if you have not exceeded the maximum number of workers,  
you can only open one MATLAB pool at a time

# Stopping a MATLAB Pool



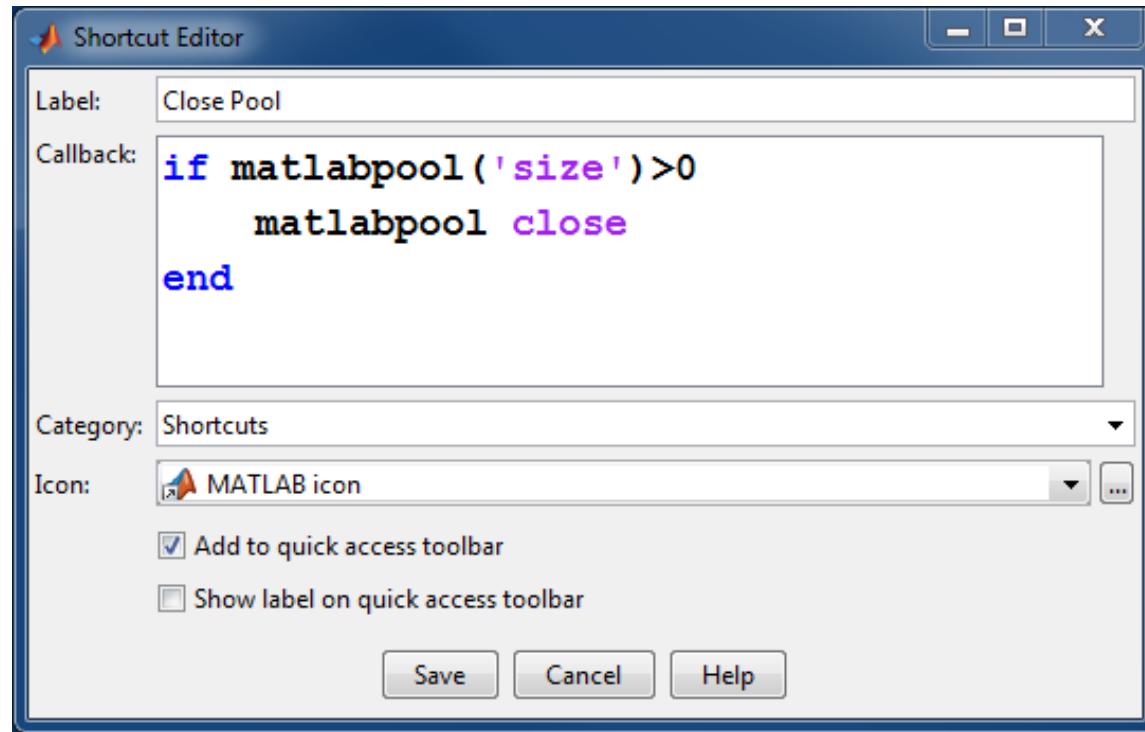


# Add Shortcut for Starting the MATLAB Pool

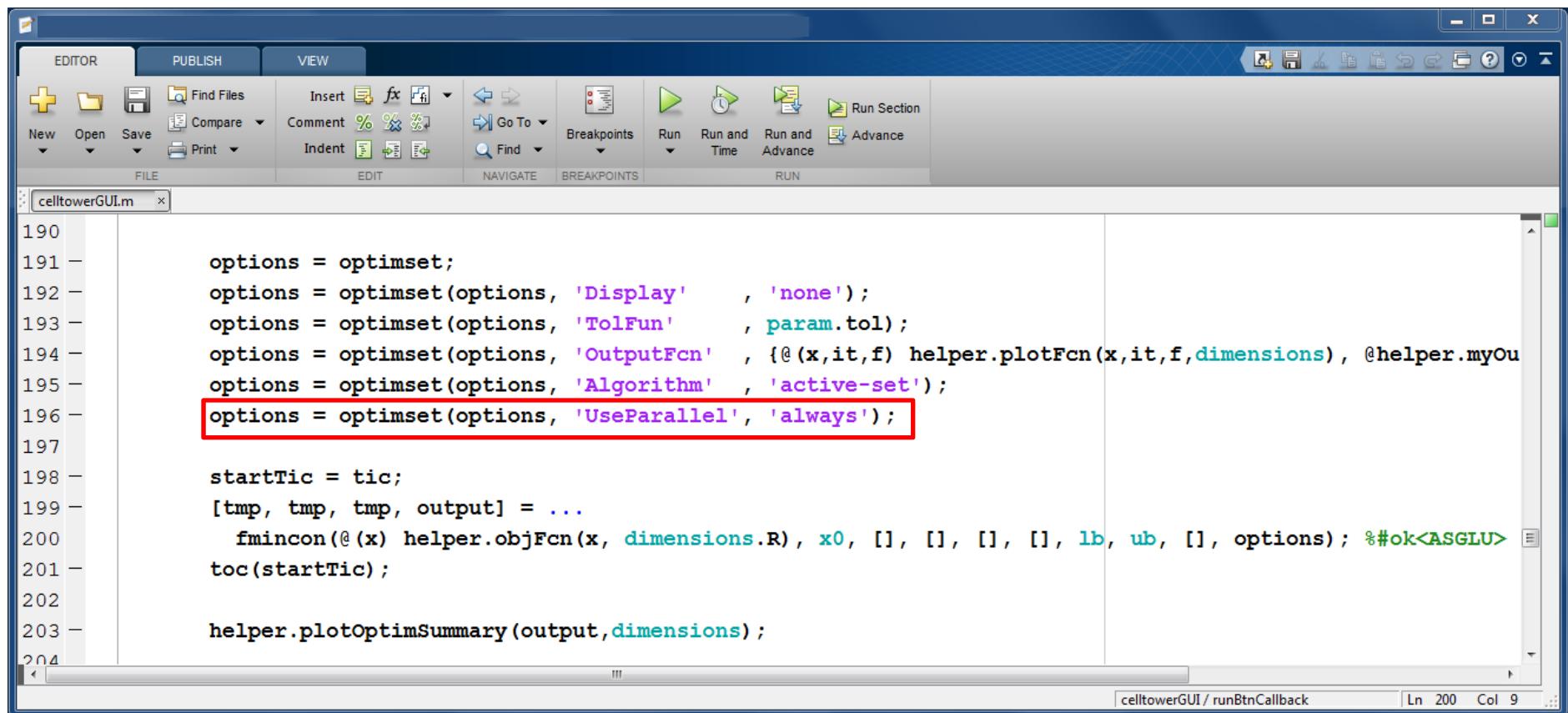




# Add Shortcut for Stopping the MATLAB Pool



# Toolbox Support for Parallel Computing



The screenshot shows the MATLAB Editor window with the file `celltowerGUI.m` open. The code in the editor is as follows:

```

190
191 options = optimset;
192 options = optimset(options, 'Display', 'none');
193 options = optimset(options, 'TolFun', param.tol);
194 options = optimset(options, 'OutputFcn', {@(x,it,f) helper.plotFcn(x,it,f,dimensions), @helper.myOut});
195 options = optimset(options, 'Algorithm', 'active-set');
196 options = optimset(options, 'UseParallel', 'always');
197
198 startTic = tic;
199 [tmp, tmp, tmp, output] = ...
200 fmincon(@(x) helper.objFcn(x, dimensions.R), x0, [], [], [], lb, ub, [], options); %#ok<ASGLU>
201 toc(startTic);
202
203 helper.plotOptimSummary(output, dimensions);
204

```

The line `options = optimset(options, 'UseParallel', 'always');` is highlighted with a red rectangular box.

# Products That Support PCT

- Bioinformatics Toolbox
- Communications System Toolbox
- Embedded Coder
- Global Optimization Toolbox
- Image Processing Toolbox
- Model-Based Calibration Toolbox
- Neural Network Toolbox
- Optimization Toolbox
- Phased Array System Toolbox
- Robust Control Toolbox
- Signal Processing Toolbox
- Simulink
- Simulink Coder
- Simulink Control Design
- Simulink Design Optimization
- Statistics Toolbox
- SystemTest

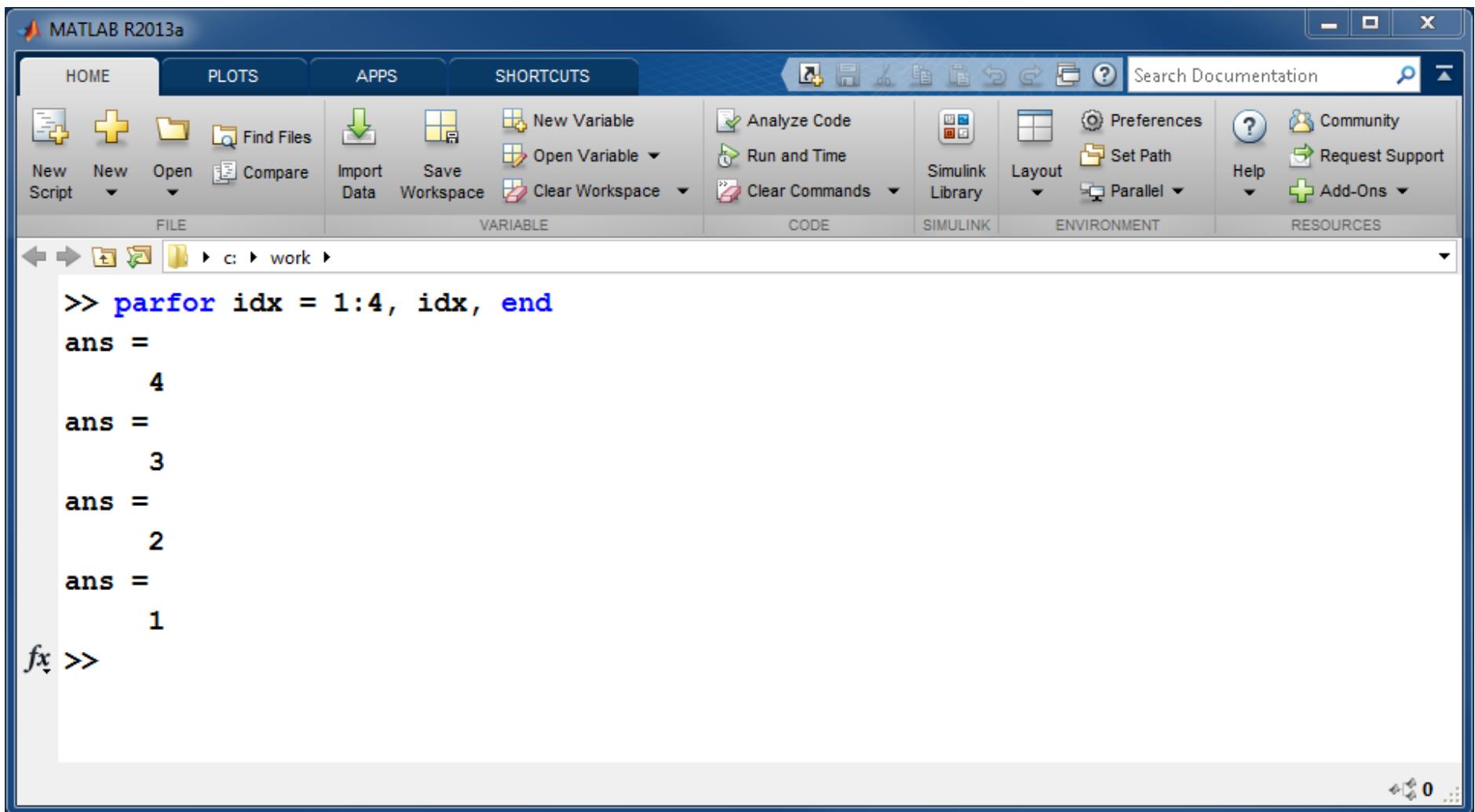
<http://www.mathworks.com/products/parallel-computing/builtin-parallel-support.html>

# parfor: The Parallel for Loop

# Using the `parfor` Construct

- In order to convert a `for` loop to a `parfor` loop, the `for` loop must at least be:
  - Task independent
  - Order independent

# Order Independent?

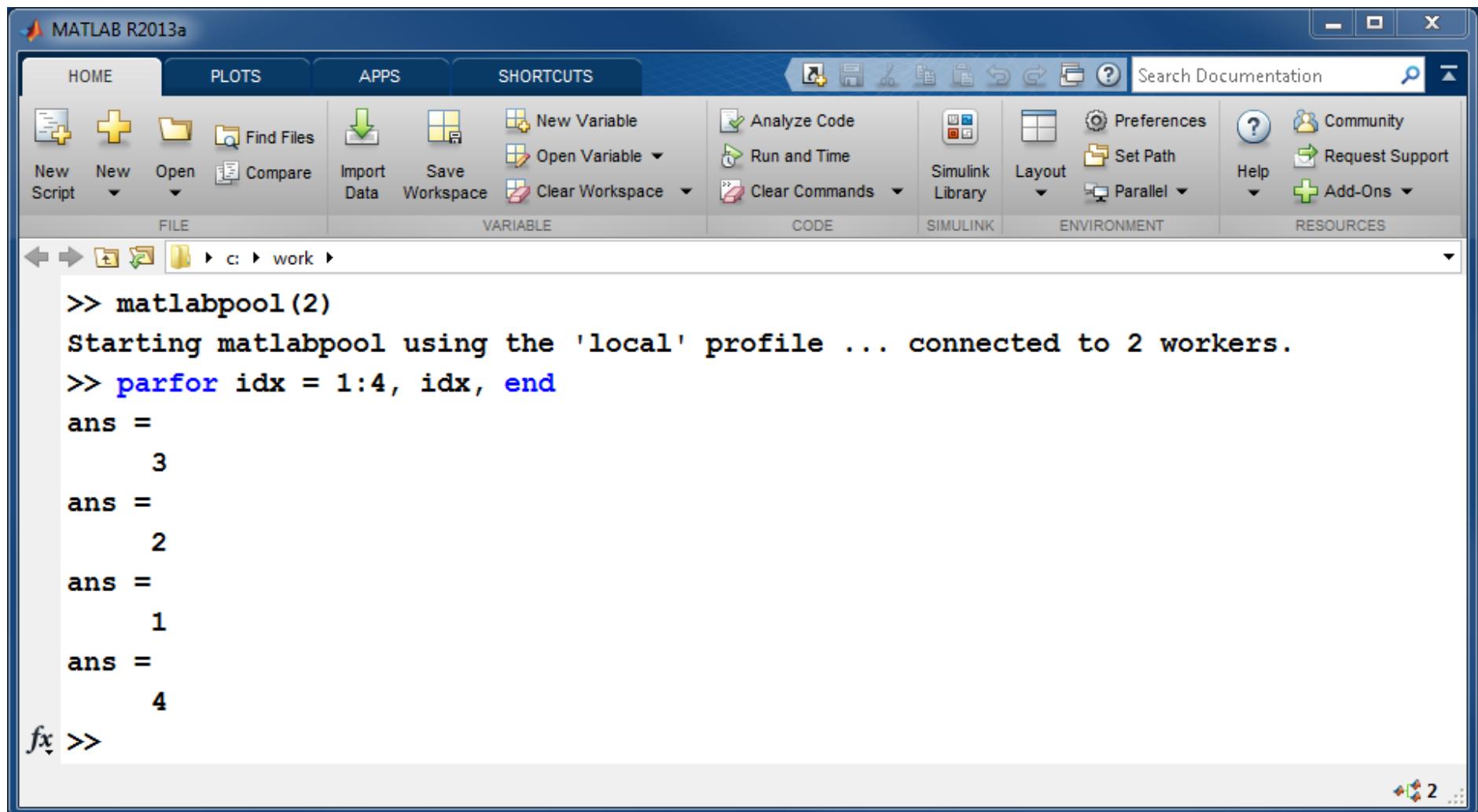


The screenshot shows the MATLAB R2013a interface with the following details:

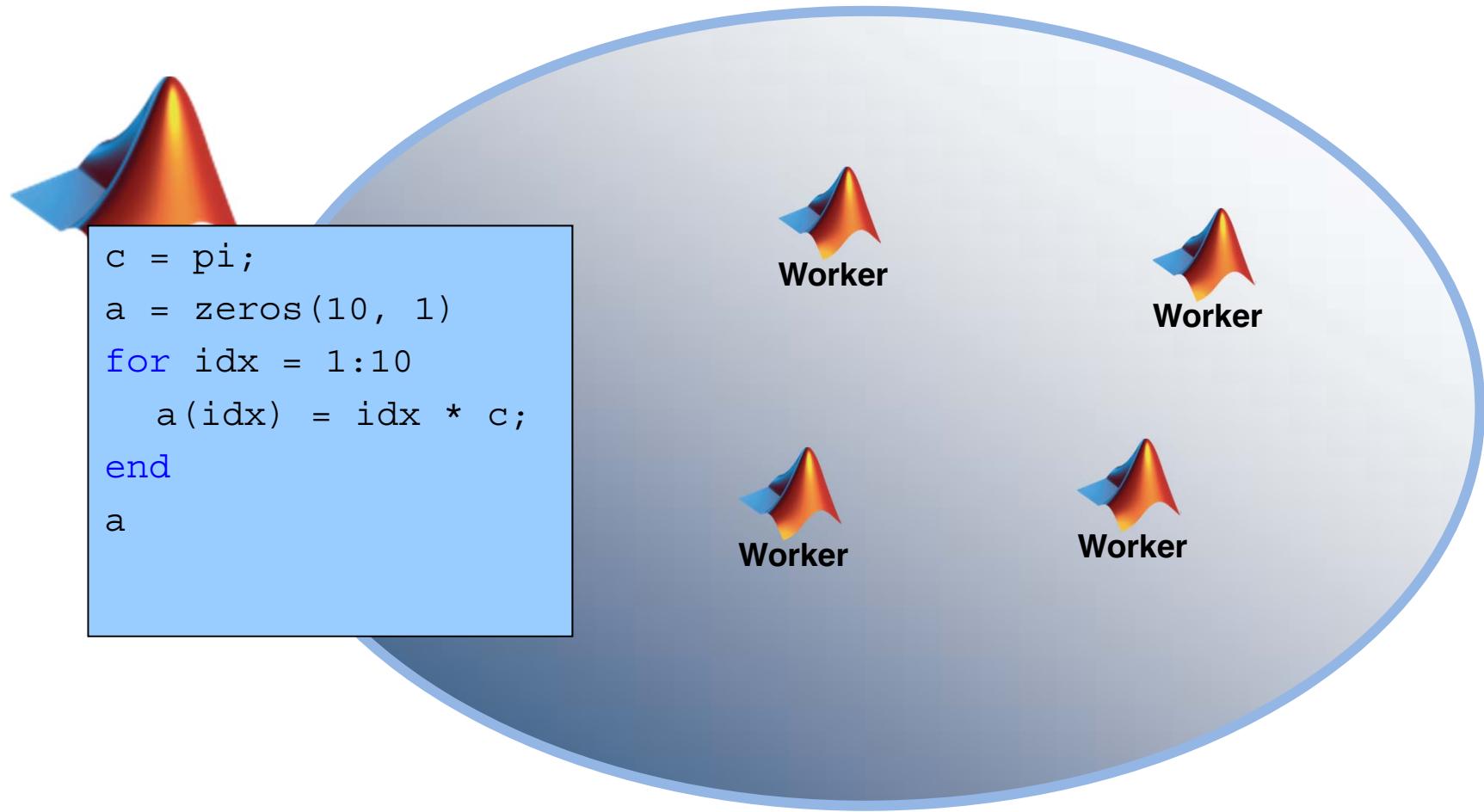
- Toolbar:** HOME, PLOTS, APPS, SHORTCUTS.
- File Explorer:** Shows the current directory as `c:\work`.
- Command Window:** Displays the following MATLAB code and output:

```
>> parfor idx = 1:4, idx, end
ans =
    4
ans =
    3
ans =
    2
ans =
    1
fx >>
```

# What If a MATLAB Pool Is Running?

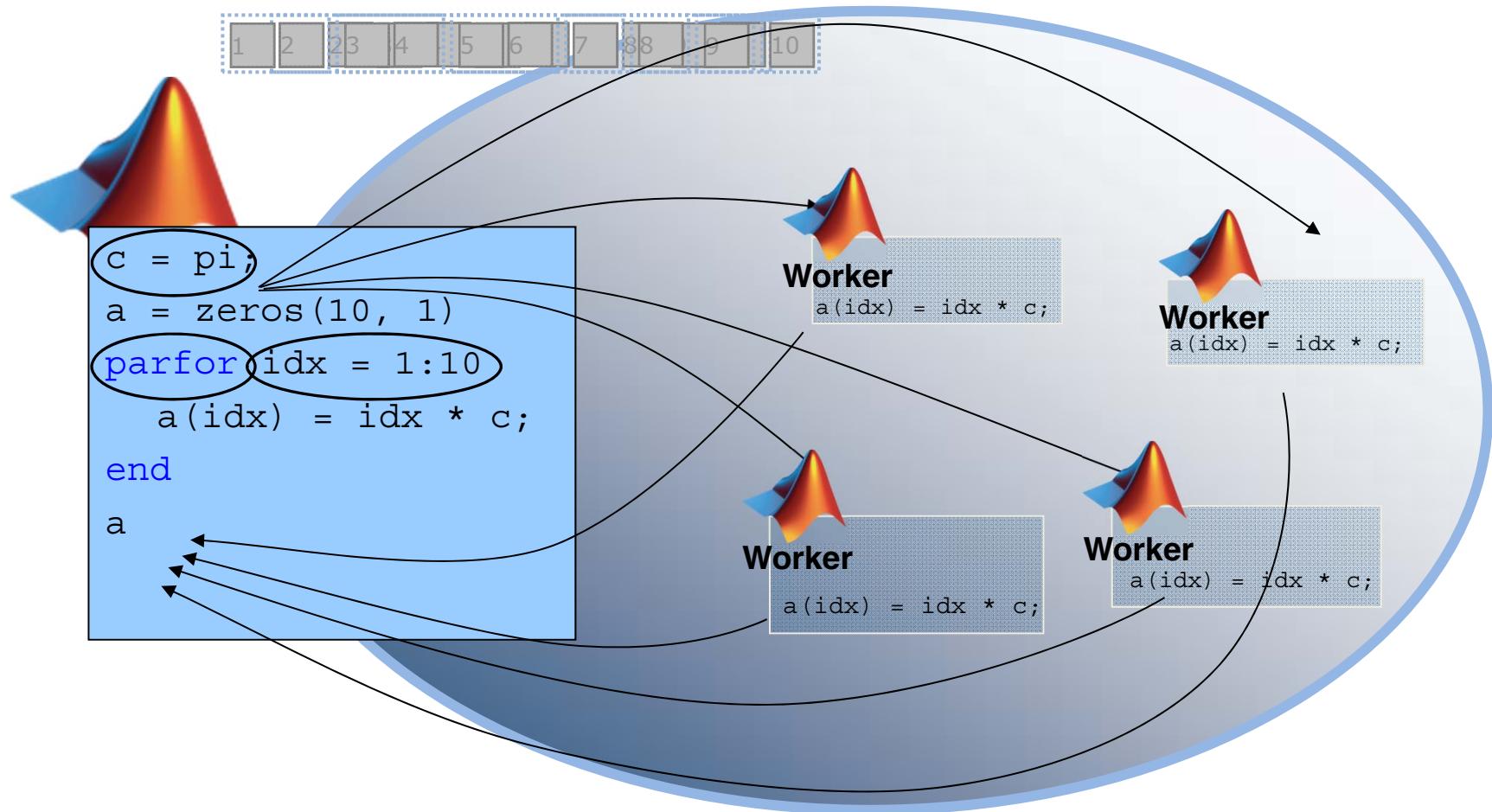


# The Mechanics of `parfor` Blocks



Pool of MATLAB Workers

# The Mechanics of `parfor` Blocks

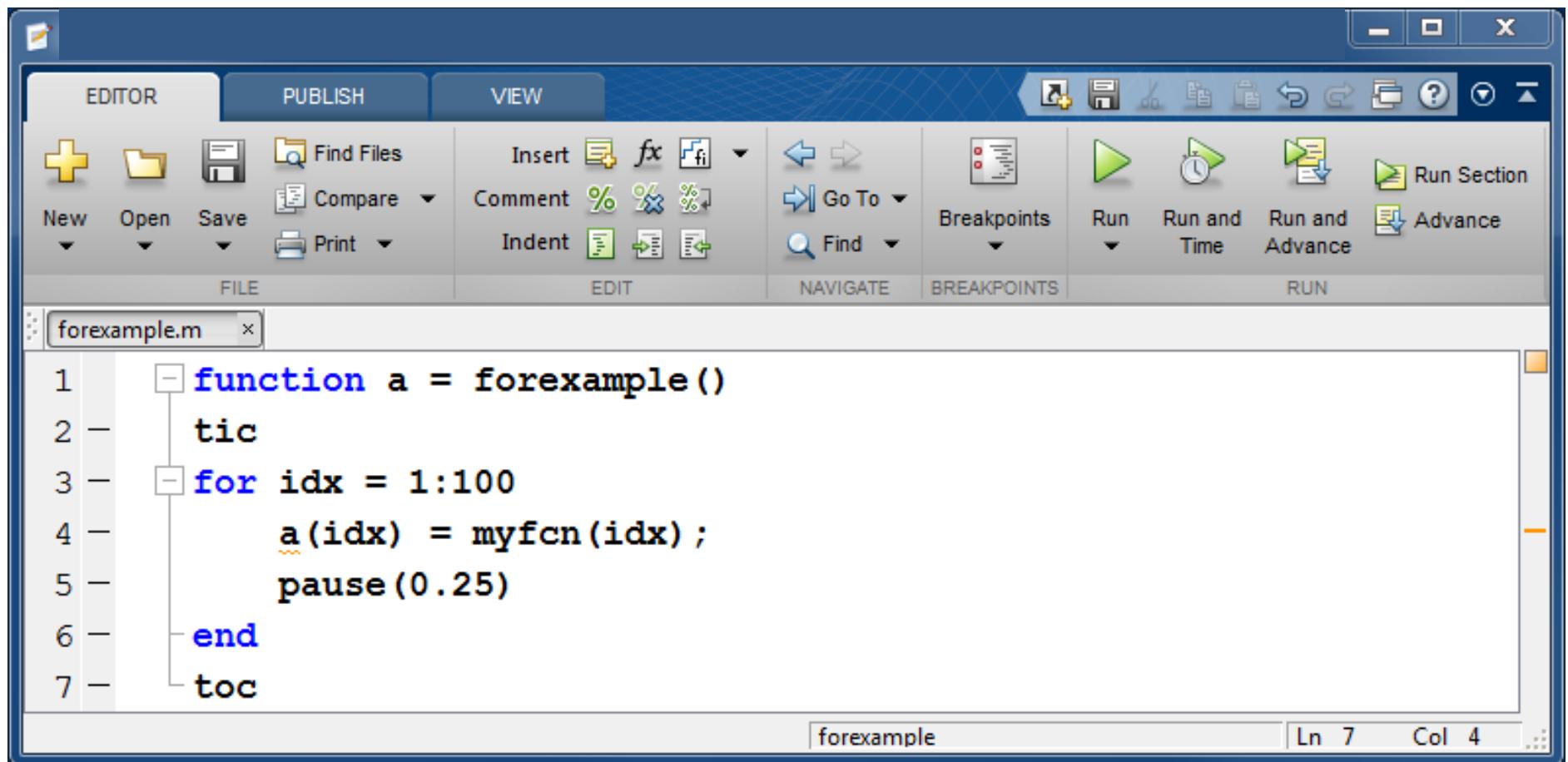


Auto-load balancing

Pool of MATLAB Workers

# Example: Hello, World!

1. Code the example below. Save it as **forexample.m**



The screenshot shows the MATLAB Editor window with the following details:

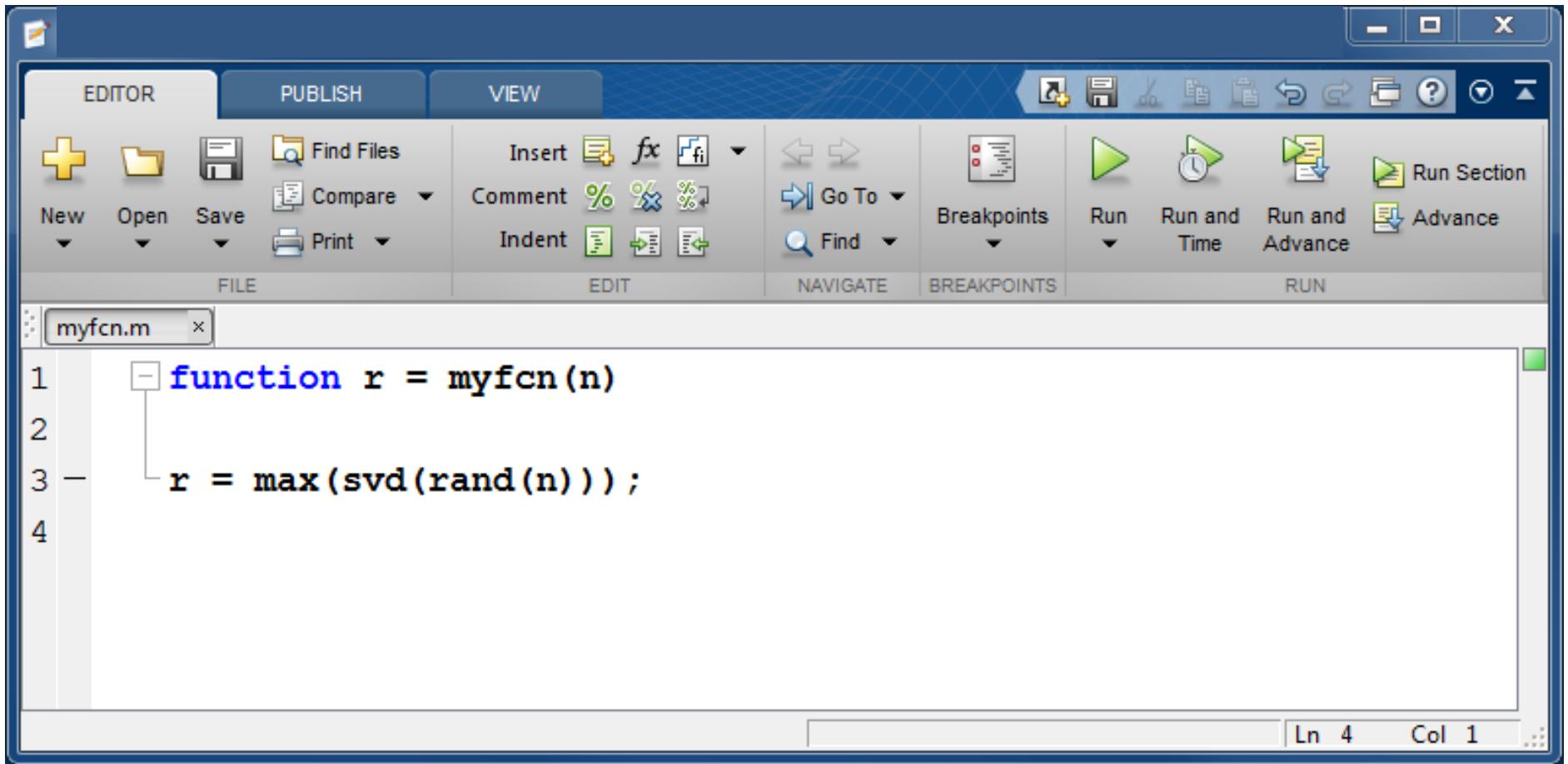
- Toolbar:** Includes buttons for New, Open, Save, Find Files, Compare, Print, Insert, Comment, Indent, Go To, Breakpoints, Find, Run, Run and Time, Run and Advance, and Advance.
- MenuBar:** Shows EDITOR, PUBLISH, and VIEW tabs.
- Code Area:** Displays the MATLAB code for `forexample.m`.

```
function a = forexample()
tic
for idx = 1:100
    a(idx) = myfcn(idx);
    pause(0.25)
end
toc
```
- Status Bar:** Shows the file name `forexample`, line number `Ln 7`, and column number `Col 4`.

```
>> forexample
```

## Example: Hello, World! (2)

2. Code the helper function. Save it as **myfcn.m**. Time and run it.



The screenshot shows the MATLAB Editor window with the following details:

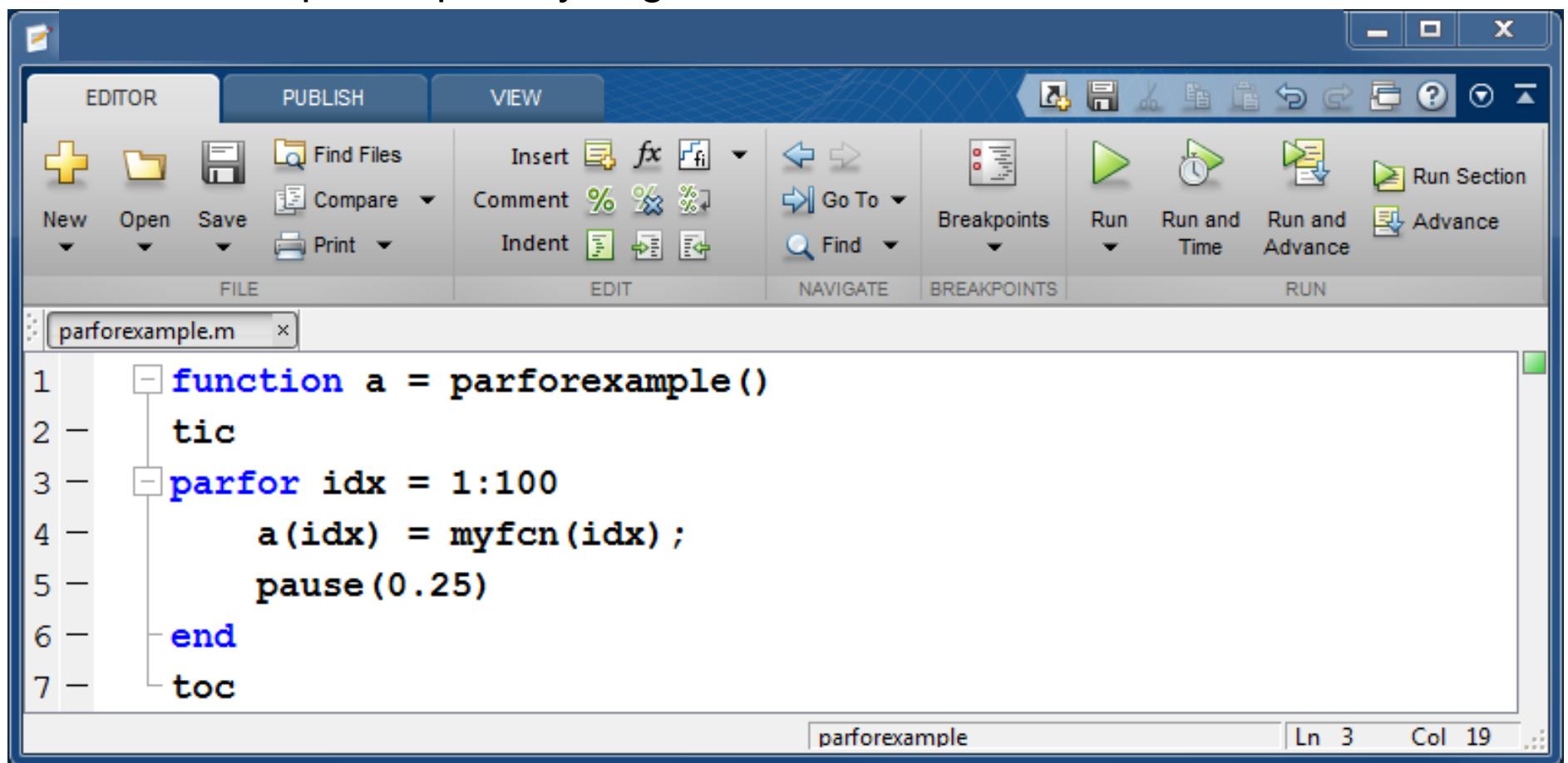
- Toolbar:** Includes buttons for New, Open, Save, Find Files, Compare, Print, Insert, Comment, Indent, Go To, Breakpoints, Find, Run, Run and Time, Run and Advance, and Advance.
- MenuBar:** Shows tabs for EDITOR, PUBLISH, and VIEW.
- Code Area:** The file `myfcn.m` is open, containing the following code:

```
1 function r = myfcn(n)
2
3 -   r = max(svd(rand(n)));
4
```
- Status Bar:** Displays "Ln 4 Col 1".

>> myfcn

## Example: Hello, World! (3)

3. Parallelize the `for` loop and save it as **parforexample.m**
4. Start a MATLAB pool and run it. Change the size of the Pool. What speed ups do you get?



The screenshot shows the MATLAB Editor window with the following details:

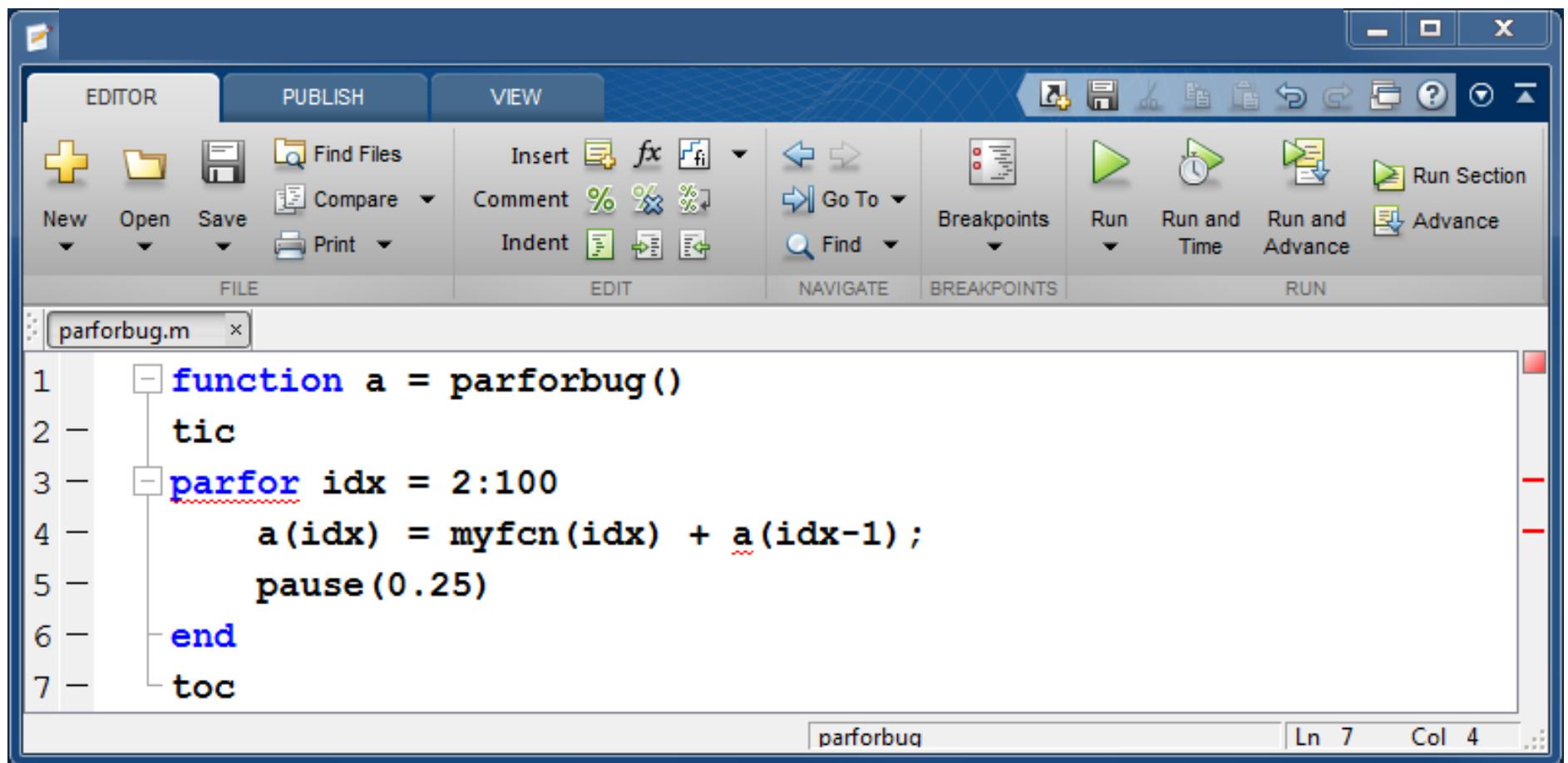
- Toolbar:** Includes buttons for New, Open, Save, Find Files, Compare, Print, Insert, Comment, Indent, Go To, Breakpoints, Find, Run, Run and Time, Run and Advance, and Advance.
- Menu Bar:** Shows EDITOR, PUBLISH, and VIEW tabs.
- File List:** Shows a file named "parforexample.m".
- Code Area:** Displays the following MATLAB script:

```
1 function a = parforexample()
2 tic
3 parfor idx = 1:100
4     a(idx) = myfcn(idx);
5     pause(0.25)
6 end
7 toc
```
- Status Bar:** Shows the file name "parforexample", line number "Ln 3", and column number "Col 19".

```
>> parforexample
```

## Example: Break It (1)

5. Add a dependency to the `parfor` loop. Look at the code analyzer messages.



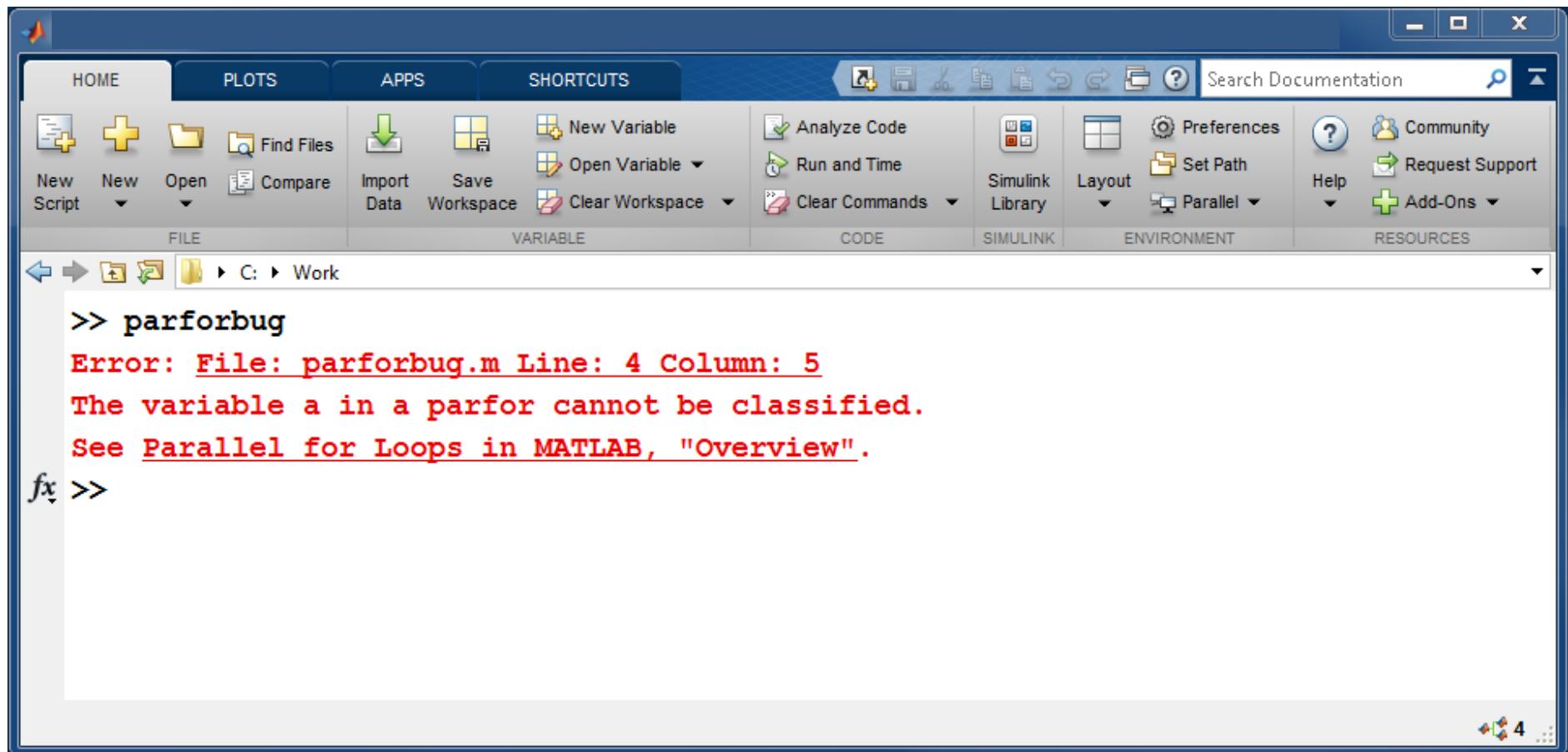
The screenshot shows the MATLAB Editor window with the file `parforbug.m` open. The code contains a `parfor` loop that lacks a dependency expression. The MATLAB IDE highlights this error with a red dotted underline under the word `parfor`. The code is as follows:

```
1 function a = parforbug()
2 tic
3 parfor idx = 2:100
4     a(idx) = myfcn(idx) + a(idx-1);
5     pause(0.25)
6 end
7 toc
```

The status bar at the bottom right indicates the current line is 7 and column is 4. The command line at the bottom shows the command `>> parforbug`.

>> parforbug

## Example: Break It (2)



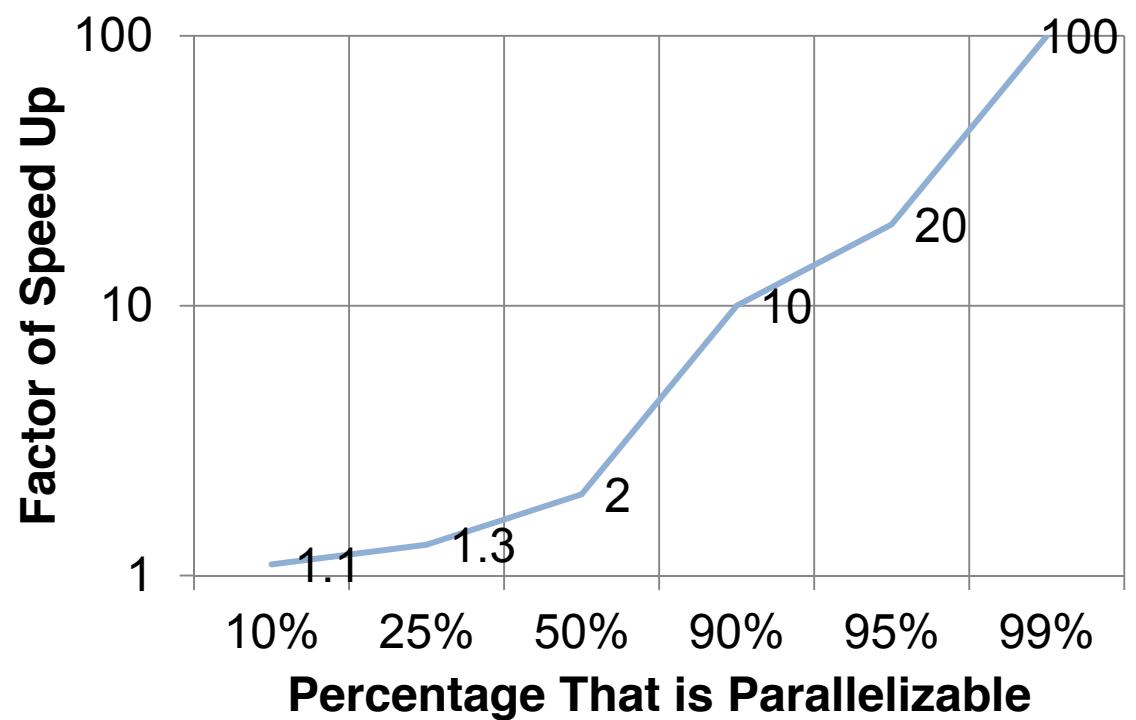
The variable a cannot be properly classified

# Constraints

- The loop variable cannot be used to index with other variables
- No inter-process communication. Therefore, a `parfor` loop cannot contain:
  - break and return statements
  - global and persistent variables
  - nested functions
  - changes to handle classes
- Transparency
  - Cannot “introduce” variables (e.g. `eval`, `load`, `global`, etc.)
  - Unambiguous Variables Names
- No nested `parfor` loops or `spmd` statement

# This is Great! Should I Get Linear Improvement?

- Not exactly
  - Too little work, too much data
- Are you calling BLAS or LAPACK routines?
- What are you timing?
  - MATLAB Profiler
- Amdahl's Law
  - $SU(N) = \frac{1}{(1-P)+\frac{P}{N}}$



# Optimizing a `parfor` Loop

- Should I pre-allocate a matrix?
  - There is no significant speedup, if any, in pre-allocating the matrix
- Should I pre-assign large matrices before the `parfor`?
  - Yes, if they're going to be referenced after the `for` loop (to be explained why later)
  - Otherwise, do all the large creation on the workers
  - So if I have a `for` loop with 100 iterations and 10 workers, are each of the matrices created 10 times? Or 100 times?
    - 100 times. See later for minimizing this.

# parfor Variable Classification

- All variables referenced at the top level of the parfor must be resolved and classified

Classification	Description
Loop	Serves as a loop index for arrays
Sliced	An array whose segments are operated on by different iterations of the loop
Broadcast	A variable defined before the loop whose value is used inside the loop, but never assigned inside the loop
Reduction	Accumulates a value across iterations of the loop, regardless of iteration order
Temporary	Variable created inside the loop, but unlike sliced or reduction variables, not available outside the loop

# Variable Classification Example

The screenshot shows the MATLAB Editor window with the script file `classification_example.m`. The code uses a `parfor` loop to calculate a variable `a` based on index `idx`.

```

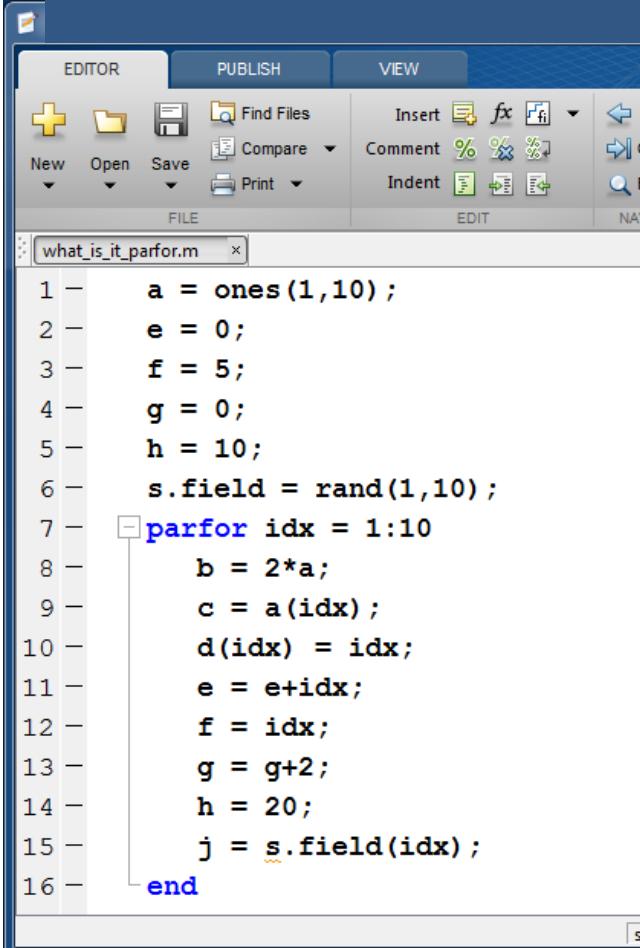
1 - a = 0;
2 - c = pi;
3 - z = 0;
4 - r = rand(1,10);
5 - parfor idx = 1:10
6 -     a = idx;           1
7 -     z = z+idx;         2
8 -     b(idx) = r(idx);  3
9 -     if idx<=c        4
10 -        d = 2*a;       5
11 -    end               6
12 - end

```

Annotations with numbered circles explain variable classifications:

- 1 Loop (around `parfor`)
- 2 Temporary (around `a`)
- 3 Reduction (around `z`)
- 4 Sliced Output (around `b(idx)`)
- 5 Sliced Input (around `r(idx)`)
- 6 Broadcast (around `d`)

# After the `for` loop, what is the type and the value of each variable?



The screenshot shows the MATLAB Editor with a script named `what_is_it_parfor.m`. The code uses a `parfor` loop to perform calculations on a broadcast variable `a`. The variables and their properties are summarized in the following table:

Variable	Type	Value
<code>a</code>	broadcast	<code>ones(1:10)</code>
<code>b</code>	temp	undefined
<code>c</code>	temp	undefined
<code>d</code>	sliced	<code>1:10</code>
<code>e</code>	reduction	55
<code>f</code>	temp	5
<code>g</code>	reduction	20
<code>h</code>	temp	10
<code>j</code>	temp	<code>0.0000 + 1.0000i</code>
<code>s</code>	broadcast	<code>rand(1,10)</code>
<code>idx</code>	loop	undefined

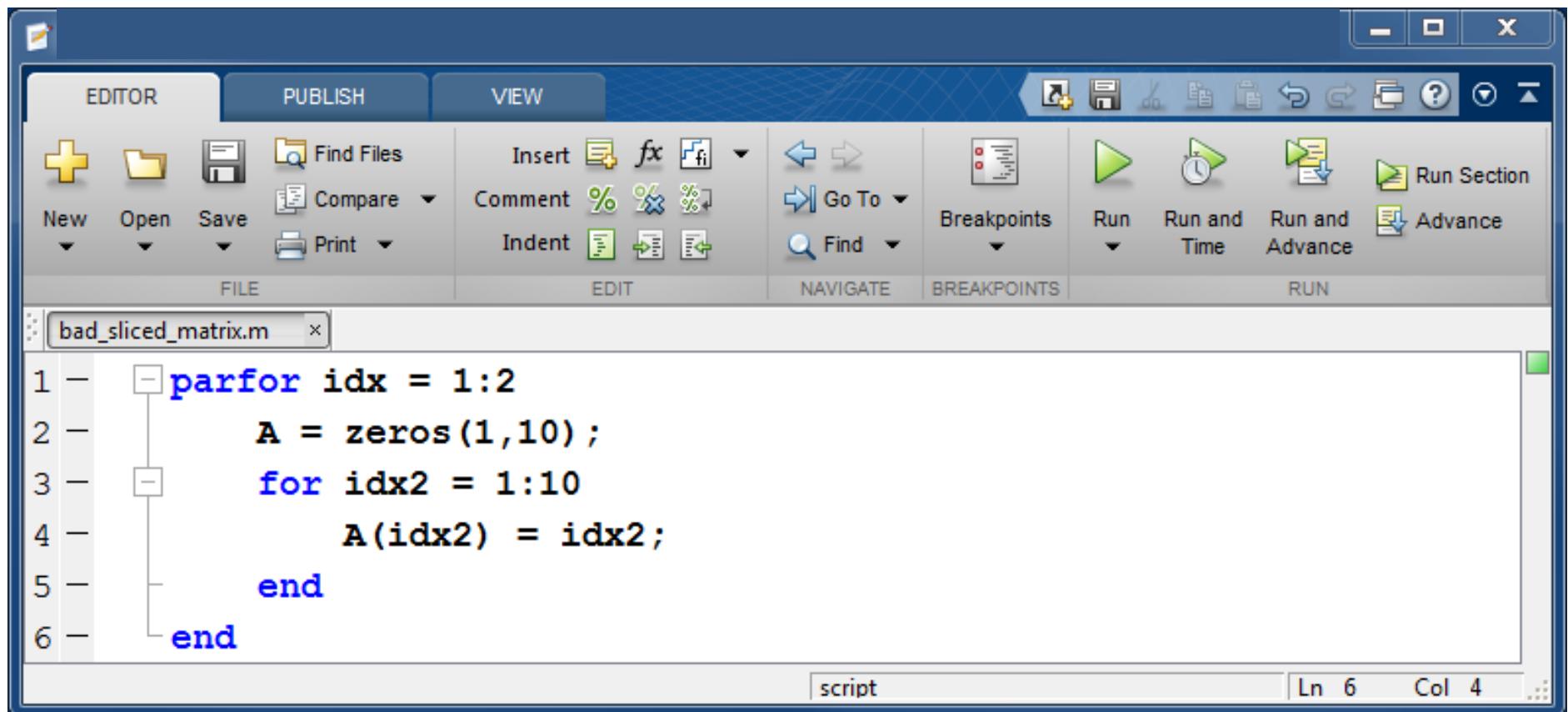
```
>> what_is_it_parfor
```

# Sliced Variables

- An indexed variables, parceled out to each worker
  - Indexing at the first level only and for () or {}
  - Within the list of indices for a sliced variable, one of these indices is of the form  $i$ ,  $i+k$ ,  $i-k$ ,  $k+i$ , or  $k-i$ , where  $i$  is the loop variable and  $k$  is a constant or a simple (non-indexed) broadcast variable; and every other index is a constant, a simple broadcast variable, colon, or end

Not Valid	Valid
$A(i+f(k),j,:,3)$	$A(i+k,j,:,3)$
$A(i,20:30,end)$	$A(i,:,end)$
$A(i,:,:s.field1)$	$A(i,:,:k)$

# Implications of Sliced Variables



The screenshot shows the MATLAB Editor window with the following details:

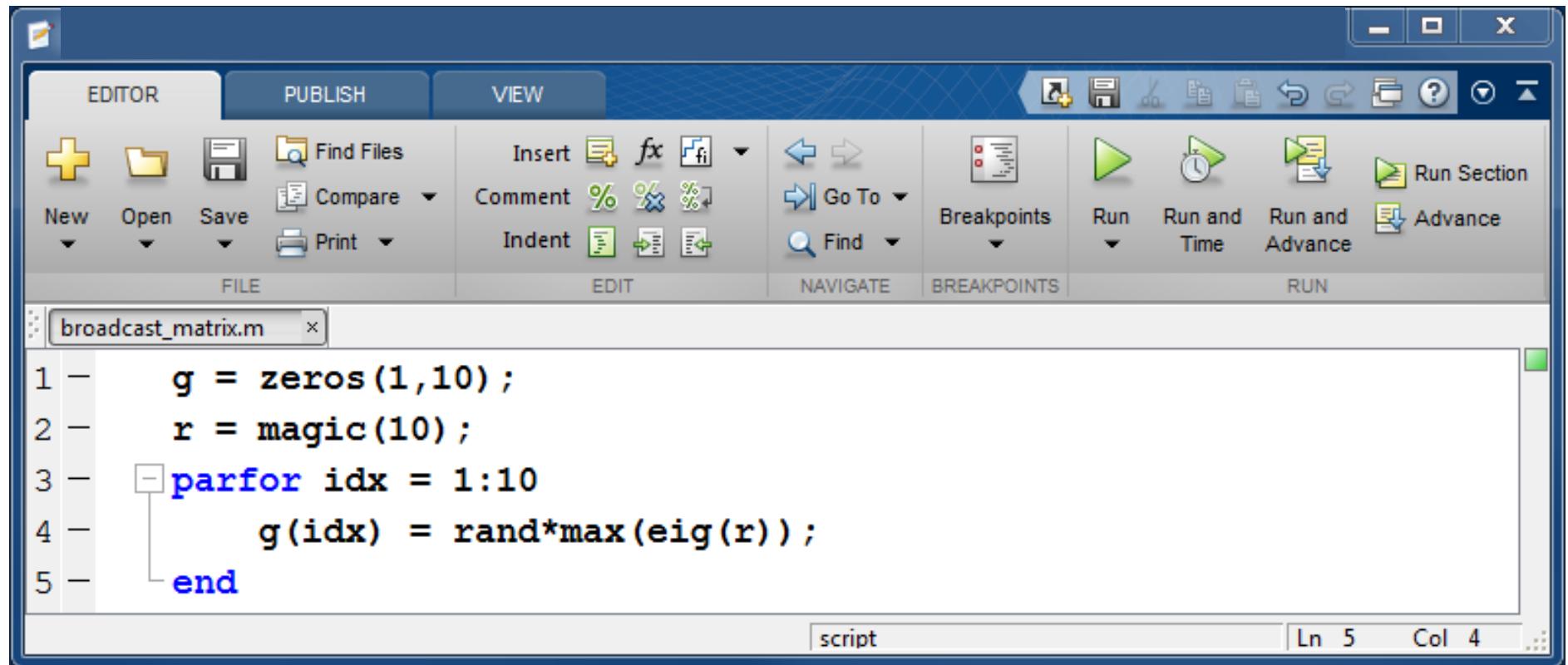
- Toolbar:** Includes buttons for New, Open, Save, Find Files, Compare, Print, Insert, Comment, Indent, Go To, Breakpoints, Find, Run, Run and Time, Run and Advance, and Advance.
- Menu Bar:** Shows tabs for EDITOR, PUBLISH, and VIEW.
- Code Area:** Displays the script file `bad_sliced_matrix.m`. The code is:

```
1 - parfor idx = 1:2
2 -     A = zeros(1,10);
3 -     for idx2 = 1:10
4 -         A(idx2) = idx2;
5 -     end
6 - end
```
- Status Bar:** Shows "script" at the bottom left, "Ln 6 Col 4" at the bottom right, and a zoom control icon on the far right.

What is the value of A?

```
>> bad_sliced_matrix
```

# Implications of Broadcast Variables



The screenshot shows the MATLAB Editor window with the script file `broadcast_matrix.m` open. The code uses a `parfor` loop to broadcast the matrix `r` to each worker. The code is as follows:

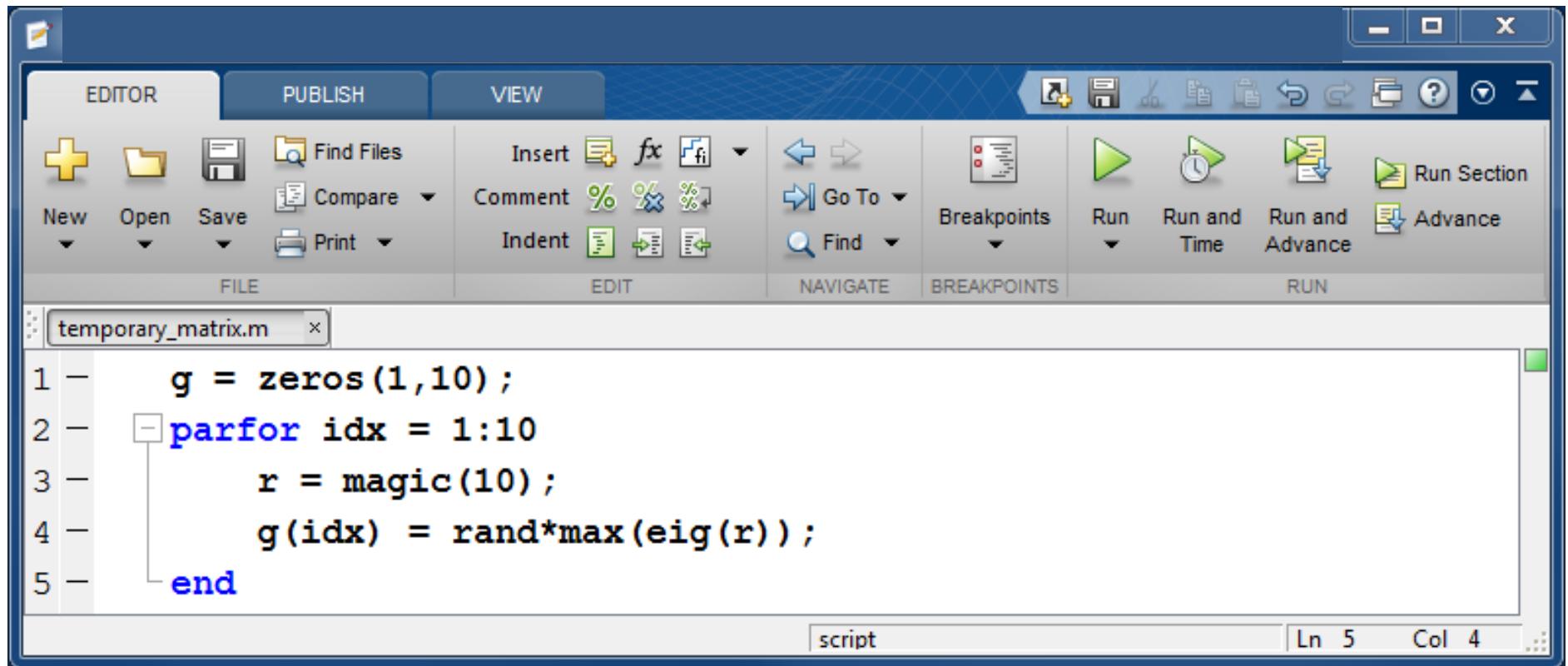
```
1 - g = zeros(1,10);
2 - r = magic(10);
3 - parfor idx = 1:10
4 -     g(idx) = rand*max(eig(r));
5 - end
```

The MATLAB interface includes a toolbar with various file operations like New, Open, Save, and Publish, along with editing and navigation tools. The status bar at the bottom right shows "script" and coordinates "Ln 5 Col 4".

The entire data set `r` is broadcast to each worker

```
>> broadcast_matrix
```

# Implications of Broadcast Variables

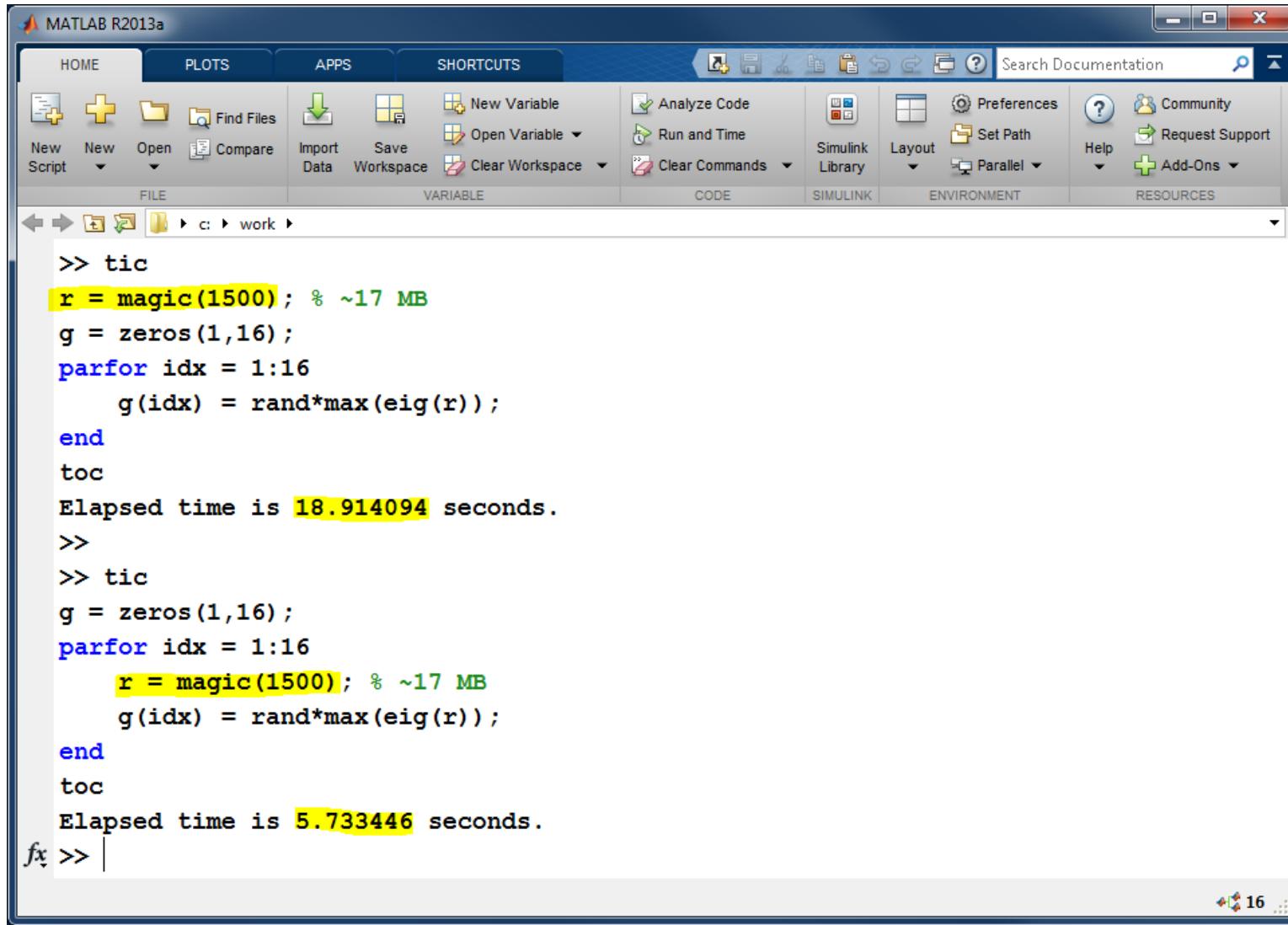


```
temporary_matrix.m
1 - g = zeros(1,10);
2 - parfor idx = 1:10
3 -     r = magic(10);
4 -     g(idx) = rand*max(eig(r));
5 - end
```

Could you create `r` on the workers instead?

```
>> temporary_matrix
```

# Implications of Broadcast Variables



The screenshot shows the MATLAB R2013a interface with the code editor window open. The code demonstrates two runs of a script. In the first run, a broadcast variable is used, resulting in a longer execution time (18.914094 seconds). In the second run, the broadcast variable is removed, resulting in a much faster execution time (5.733446 seconds).

```
>> tic
r = magic(1500); % ~17 MB
g = zeros(1,16);
parfor idx = 1:16
    g(idx) = rand*max(eig(r));
end
toc
Elapsed time is 18.914094 seconds.
>>
>> tic
g = zeros(1,16);
parfor idx = 1:16
    r = magic(1500); % ~17 MB
    g(idx) = rand*max(eig(r));
end
toc
Elapsed time is 5.733446 seconds.
fx >> |
```

# Implications of Reduction Variables

- Variable appears on both sides of assignment
- Same operation must be performed on variable for all iterations
- Reduction function must be associative and commutative

# Implications of Reduction Variables

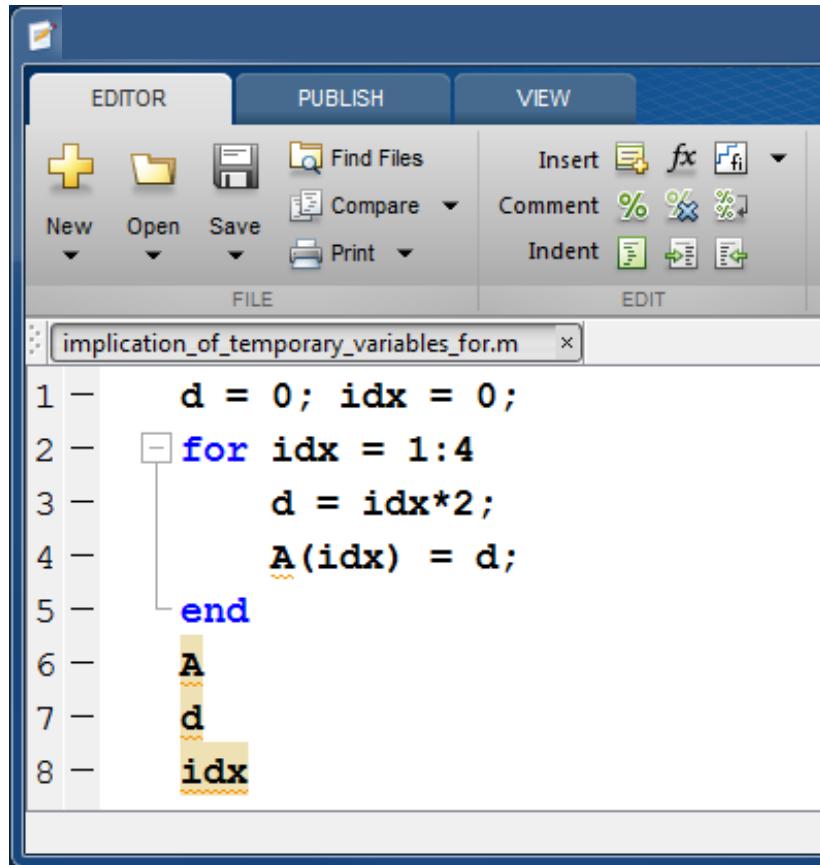
```
>> x = 0;  
parfor idx = 1:10  
    x = x+idx;  
end  
x
```

```
>> x2 = [];  
parfor idx = 1:10  
    x2 = [x2 idx];  
end  
x2
```

```
>> x3 = 0;  
parfor idx = 1:32  
    if idx<16  
        x3 = x3*idx;  
    else  
        x3 = x3+idx;  
    end  
end  
x3
```

Error: Different reduction functions are used for the same variable x3.  
See [Parallel for Loops in MATLAB, "Basic Rules for Reduction Variables"](#).

# Implications of Temporary Variables



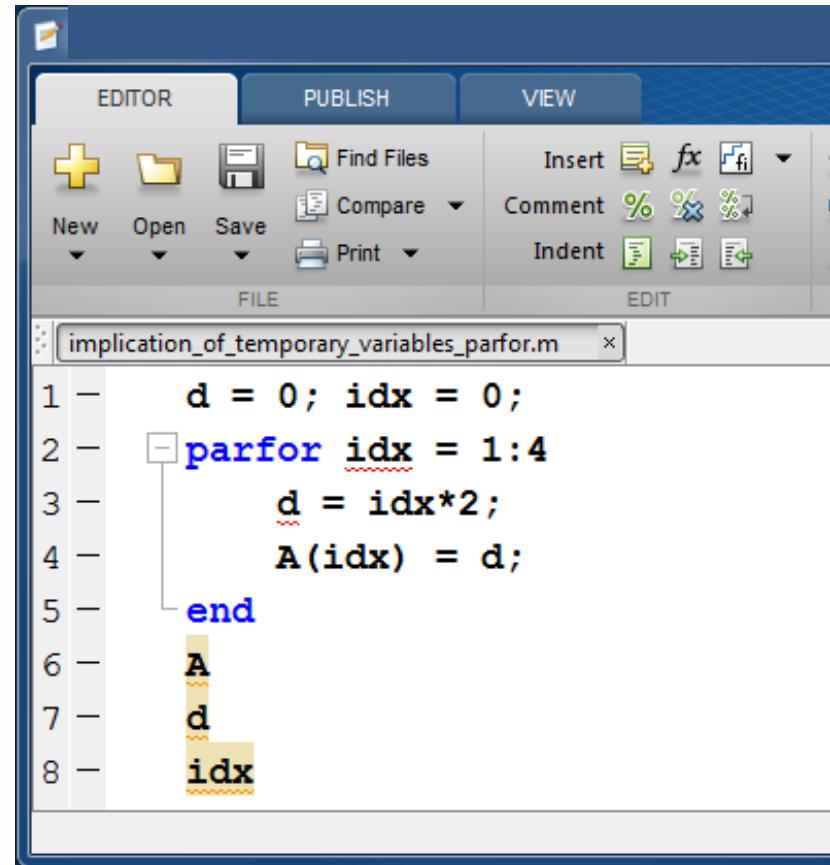
The screenshot shows the MATLAB Editor interface with the tab bar set to 'EDITOR'. The window title is 'implication\_of\_temporary\_variables\_for.m'. The code in the editor is:

```

1 -      d = 0; idx = 0;
2 -      for idx = 1:4
3 -          d = idx*2;
4 -          A(idx) = d;
5 -      end
6 -      A
7 -      d
8 -      idx

```

Temporary variable names 'd' and 'idx' are highlighted in orange.



The screenshot shows the MATLAB Editor interface with the tab bar set to 'EDITOR'. The window title is 'implication\_of\_temporary\_variables\_parfor.m'. The code in the editor is:

```

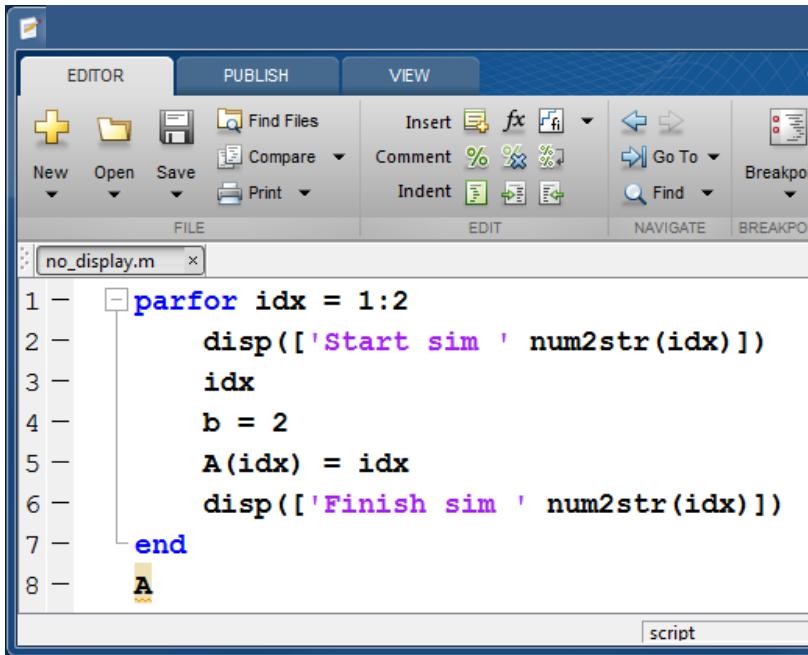
1 -      d = 0; idx = 0;
2 -      parfor idx = 1:4
3 -          d = idx*2;
4 -          A(idx) = d;
5 -      end
6 -      A
7 -      d
8 -      idx

```

Temporary variable names 'd' and 'idx' are highlighted in orange.

What is the value of A? d? idx?

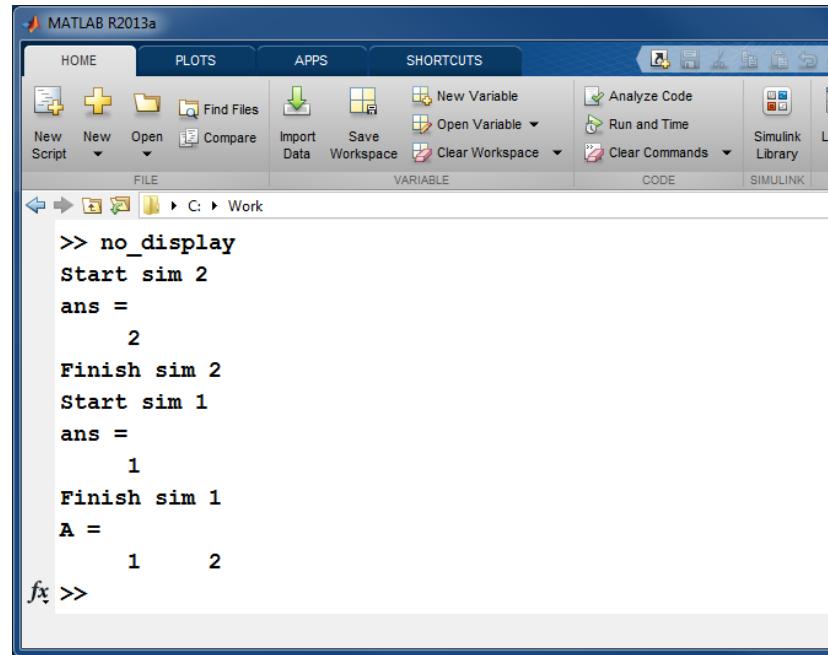
# Variable Assignments Are Not Displayed When Running a `parfor`



```

no_display.m
1 - parfor idx = 1:2
2 -     disp(['Start sim ' num2str(idx)])
3 -     idx
4 -     b = 2
5 -     A(idx) = idx
6 -     disp(['Finish sim ' num2str(idx)])
7 - end
8 - A

```



```

>> no_display
Start sim 2
ans =
    2
Finish sim 2
Start sim 1
ans =
    1
Finish sim 1
A =
    1    2
fx >>

```

>> no\_display

## rand in parfor Loops (1)

- MATLAB has a repeatable sequence of random numbers
- When workers are started up, rather than using this same sequence of random numbers, the `labindex` is used to seed the RNG

## rand in parfor Loops (2)

MATLAB R2013a

```
>> rand('twister',5489)
>> for idx = 1:8, rand, end
ans =
    0.8147
ans =
    0.9058
ans =
    0.1270
ans =
    0.9134
ans =
    0.6324
ans =
    0.0975
ans =
    0.2785
ans =
    0.5469
fx >>
```

MATLAB R2013a

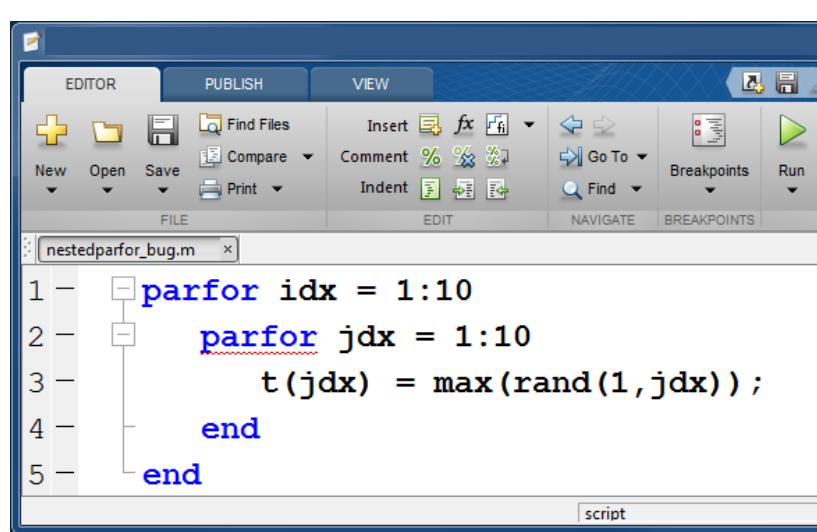
```
>> matlabpool(4)
Starting matlabpool using the 'local' method
>> parfor idx = 1:8, rand, end
ans =
    0.3246
ans =
    0.2646
ans =
    0.0968
ans =
    0.8847
ans =
    0.8939
ans =
    0.2502
ans =
    0.5052
ans =
    0.9993
fx >>
```

# Outline

- Parallelizing Your MATLAB Code
- Tips for Programming with a Parallel for Loop
- Computing to a GPU
- Scaling to a Cluster
- Debugging and Troubleshooting

# What If My `parfor` Has a `parfor` In It?

- MATLAB runs a static analyzer on the immediate `parfor` and will error out nested `parfor` loops.  
However, functions called from within the `parfor` that include `parfor` loops are treated as regular `for` loops

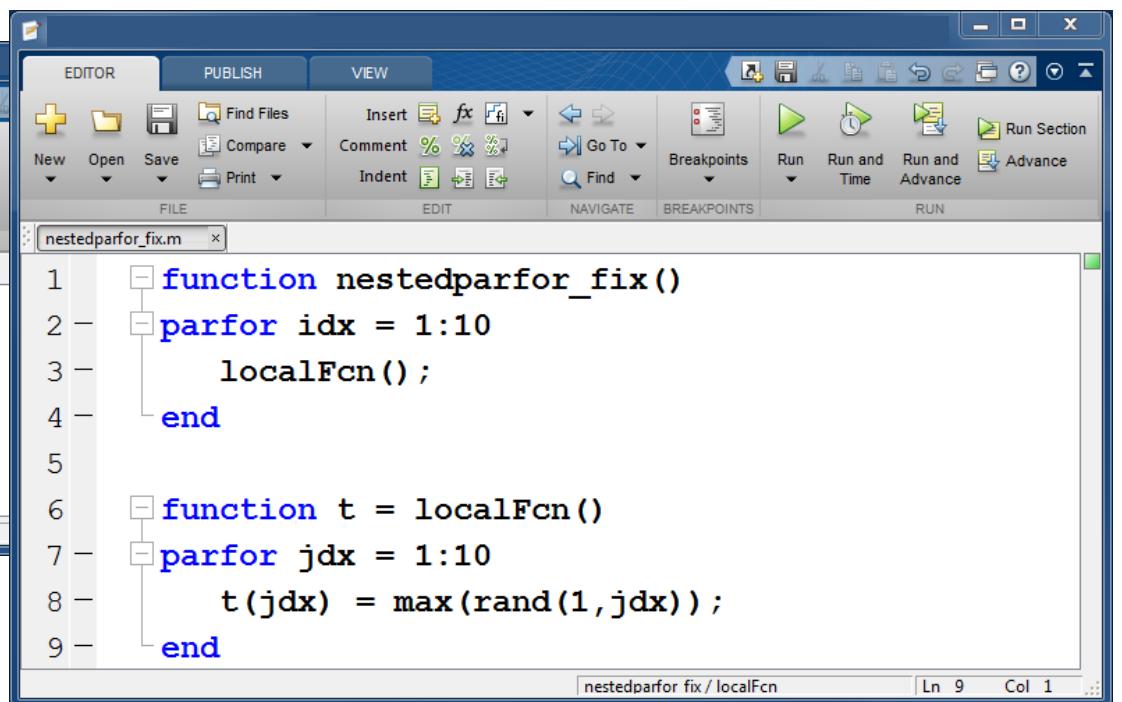


The screenshot shows the MATLAB Editor interface with a script named `nestedparfor_bug.m`. The code contains two nested `parfor` loops:

```

nestedparfor_bug.m
1 - parfor idx = 1:10
2 -   parfor jdx = 1:10
3 -     t(jdx) = max(rand(1,jdx));
4 -   end
5 - end

```



The screenshot shows the MATLAB Editor interface with a function named `nestedparfor_fix.m`. The code has been refactored to use a local function `localFcn()` to encapsulate the inner loop, thus avoiding a static analysis error:

```

nestedparfor_fix.m
1 function nestedparfor_fix()
2 parfor idx = 1:10
3   localFcn();
4 end
5
6 function t = localFcn()
7 parfor jdx = 1:10
8   t(jdx) = max(rand(1,jdx));
9 end

```

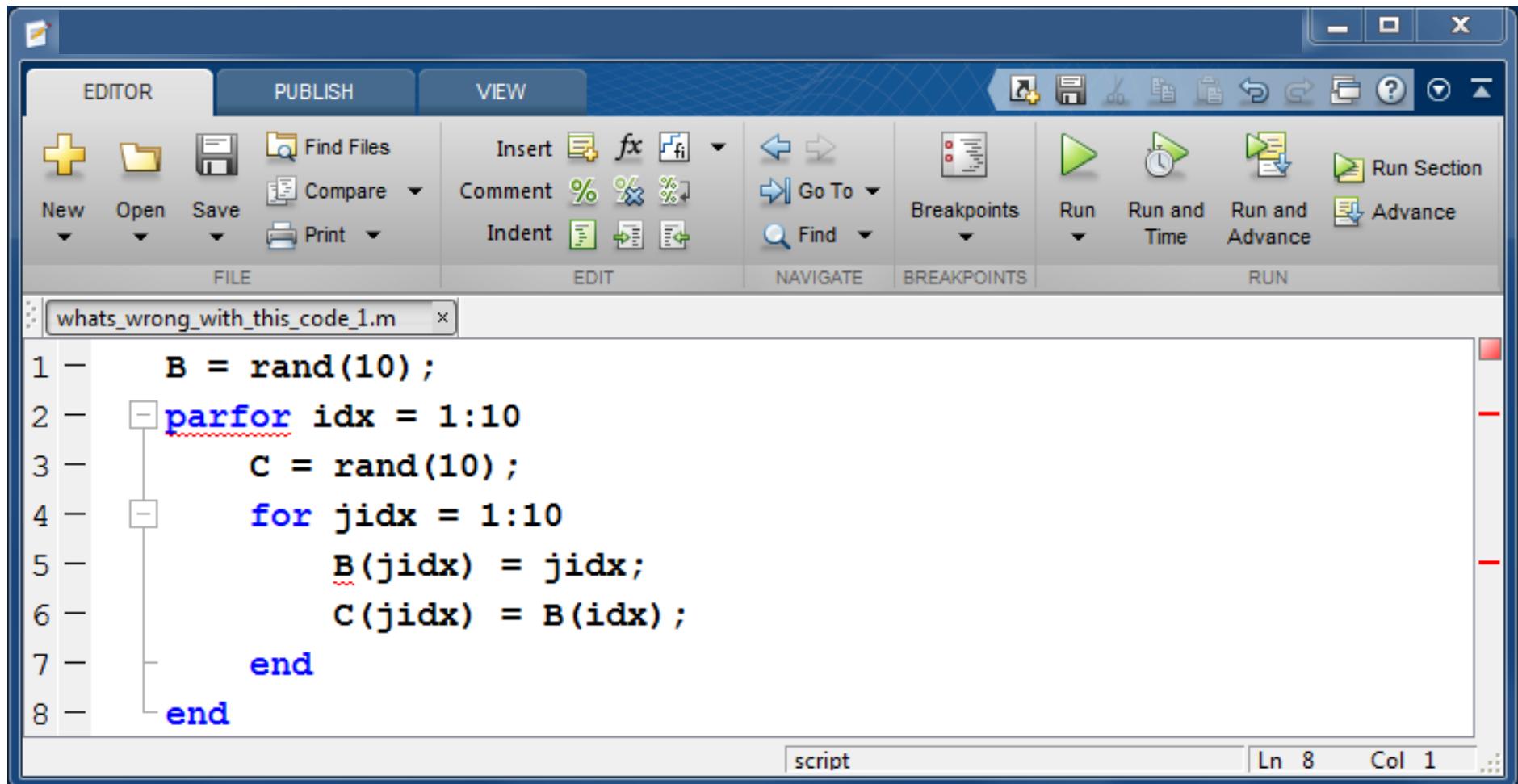
```

>> nestedparfor_bug
>> nestedparfor_fix

```

# What's Wrong With This Code?

Why can we index into C with `jidx`, but not `B`?



The screenshot shows the MATLAB Editor window with the following code:

```
1 - B = rand(10);
2 - parfor idx = 1:10
3 -     C = rand(10);
4 -     for jidx = 1:10
5 -         B(jidx) = jidx;
6 -         C(jidx) = B(idx);
7 -     end
8 - end
```

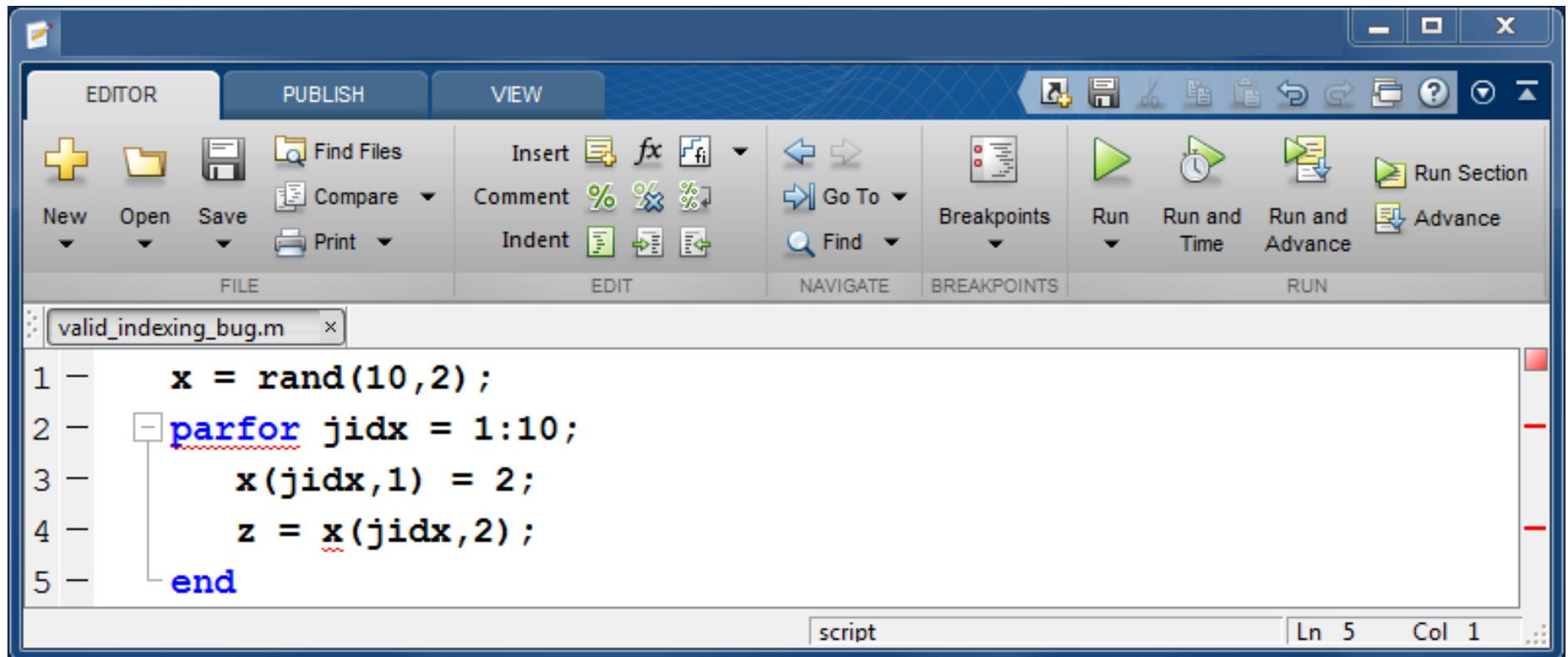
The code contains several syntax errors highlighted by red underlines and squiggly lines:

- `parfor` is underlined in red.
- `B(jidx)` is underlined in red.
- `C(jidx)` is underlined in red.
- `B(idx)` is underlined in red.

The MATLAB interface includes a toolbar with various icons for file operations, editing, and running code. The status bar at the bottom right shows "Ln 8 Col 1".

```
>> whats_wrong_with_this_code
```

# parfor issue: Indexing With Different Expressions



The screenshot shows the MATLAB Editor window with the following code:

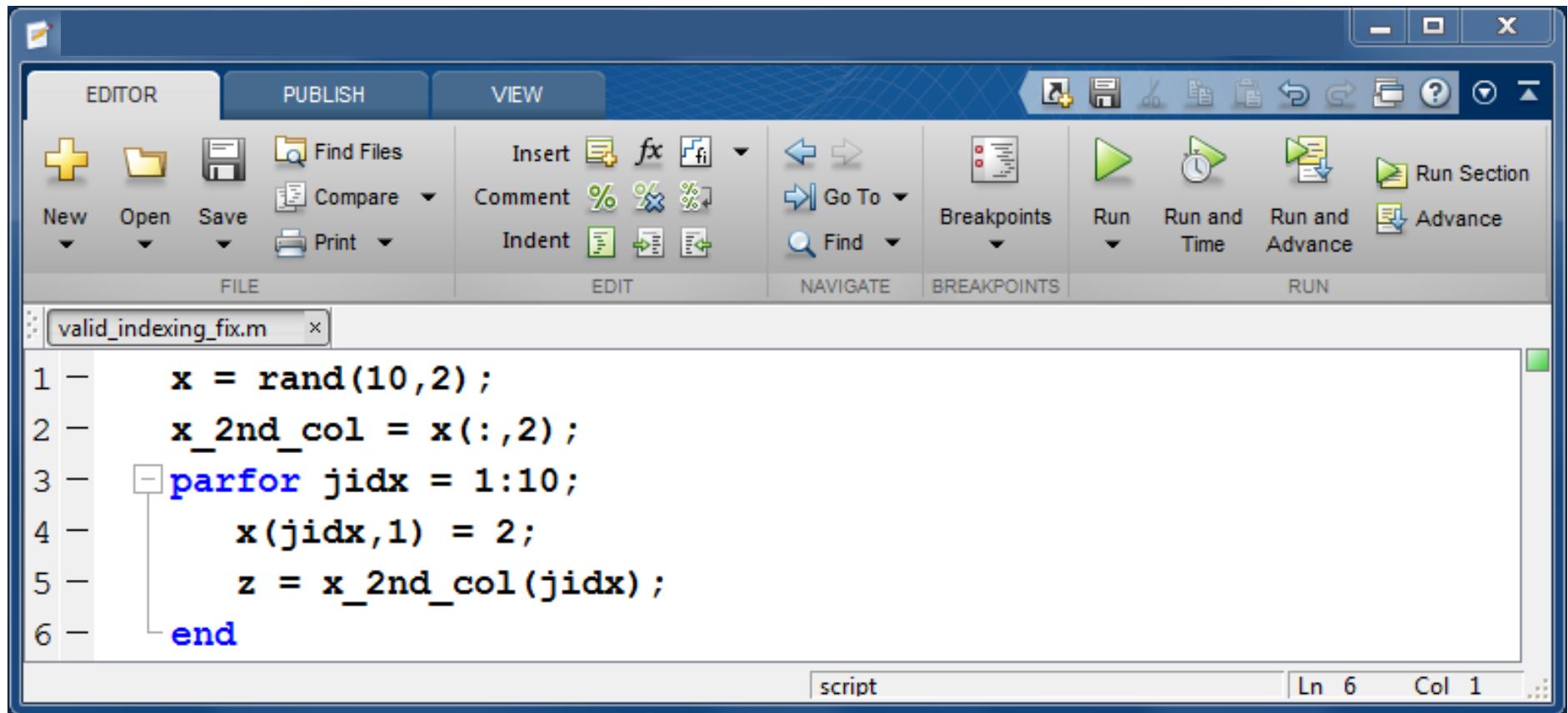
```
1 - x = rand(10,2);
2 - parfor jidx = 1:10;
3 -     x(jidx,1) = 2;
4 -     z = x(jidx,2);
5 - end
```

The word "parfor" is underlined with a red squiggly line, indicating a syntax error. The code is saved as "valid\_indexing\_bug.m".

How can we avoid indexing into `x` two different ways?

```
>> valid_indexing_bug
```

# parfor issue: Solution



The screenshot shows the MATLAB Editor window with the following details:

- Toolbar:** Includes buttons for New, Open, Save, Find Files, Compare, Print, Insert, Comment, Indent, Go To, Breakpoints, Find, Run, Run and Time, Run and Advance, and Advance.
- MenuBar:** Shows tabs for EDITOR, PUBLISH, and VIEW.
- Code Area:** Displays the script file `valid_indexing_fix.m` with the following content:

```

1 - x = rand(10,2);
2 - x_2nd_col = x(:,2);
3 - parfor jidx = 1:10;
4 -     x(jidx,1) = 2;
5 -     z = x_2nd_col(jidx);
6 - end

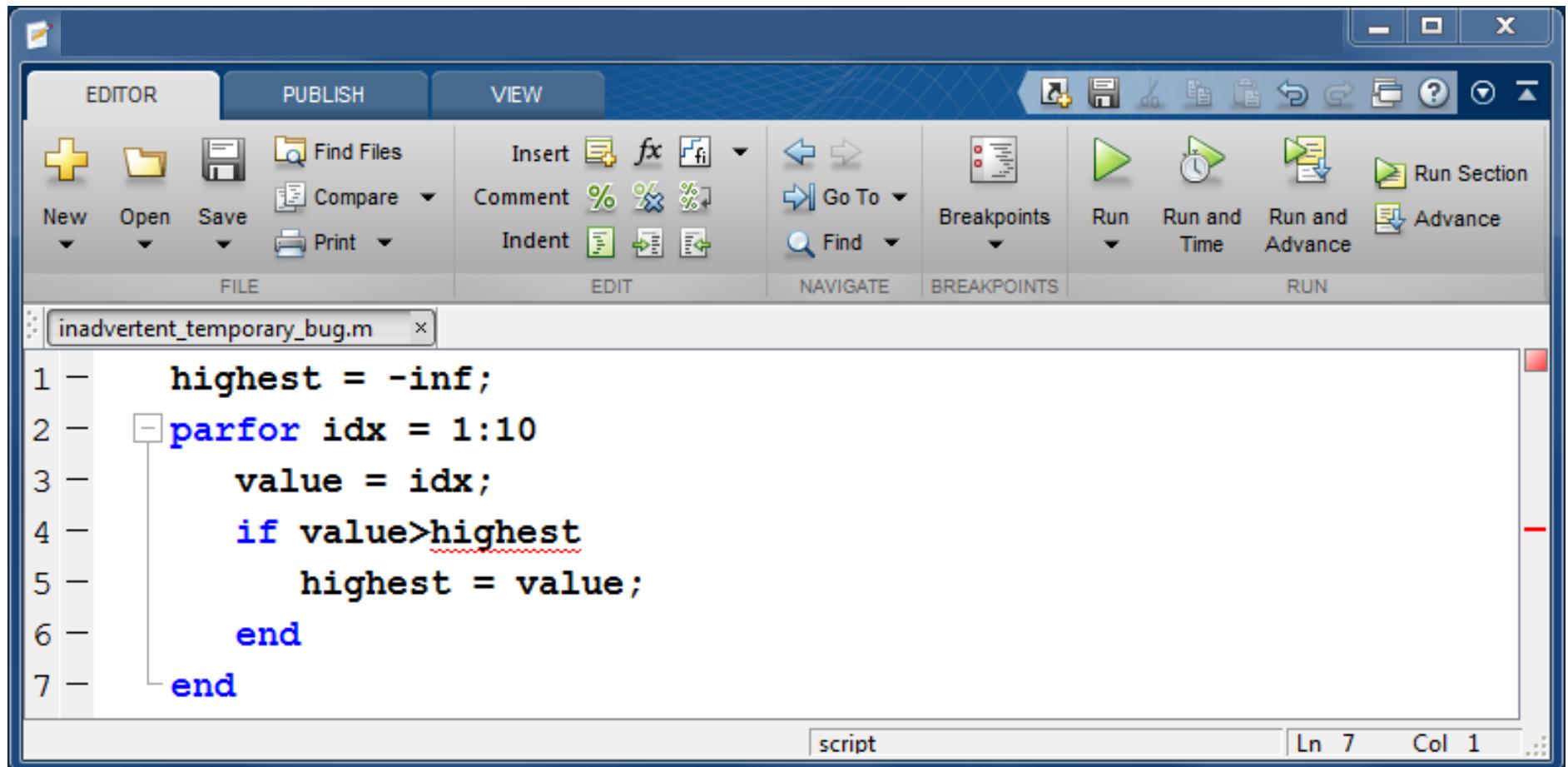
```
- Status Bar:** Shows "script" at Ln 6 Col 1.

Create a temporary variable, `x_2nd_col`, to store the column vector. Then loop into the vector using the looping index, `jidx`, rather than the into a matrix.

Note: This doesn't scale very well if we needed to index into `x` many ways.

>> `valid_indexing_fix`

# parfor issue: Inadvertently Creating Temporary Variables



The screenshot shows the MATLAB Editor window with the following code:

```

1 -     highest = -inf;
2 -     parfor idx = 1:10
3 -         value = idx;
4 -         if value>highest
5 -             highest = value;
6 -         end
7 -     end

```

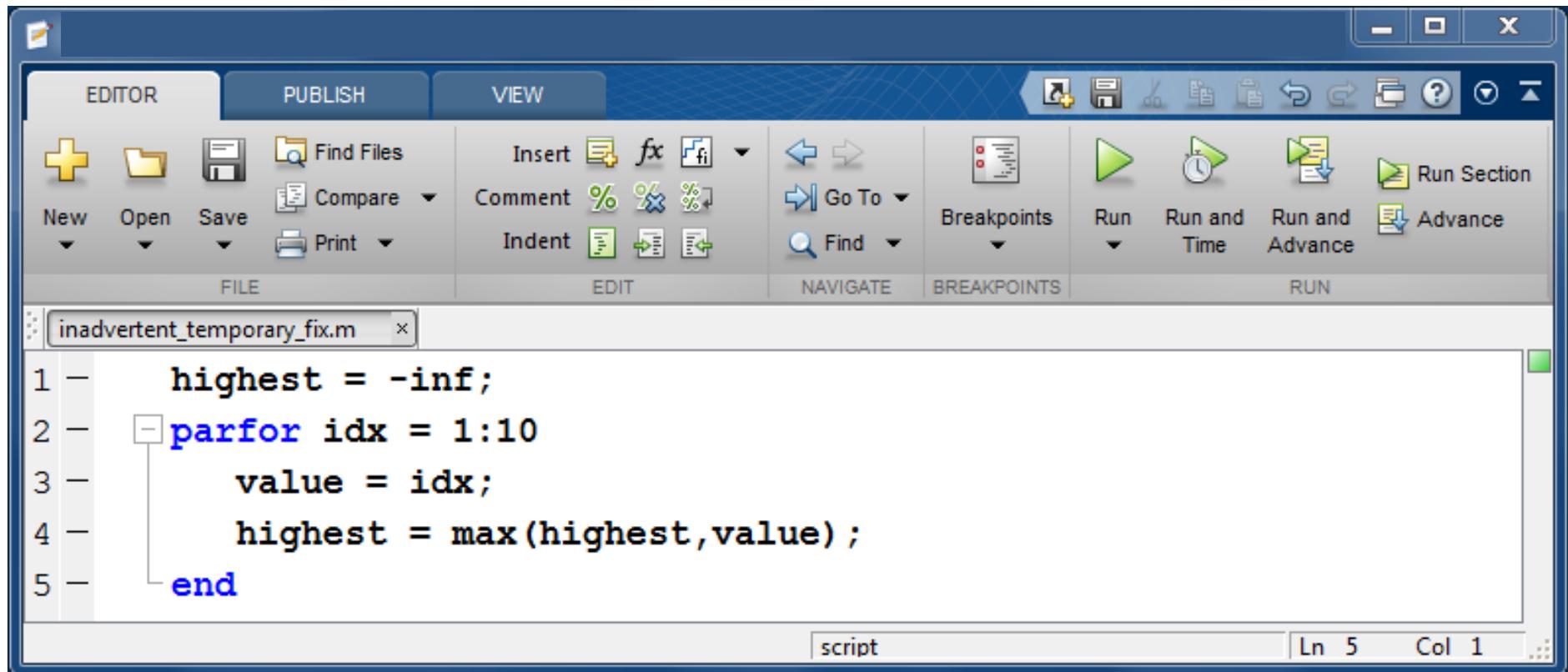
The code analyzer has flagged the variable `highest` as a temporary variable, indicated by a red underline. The editor interface includes tabs for EDITOR, PUBLISH, and VIEW, and toolbars for FILE, EDIT, NAVIGATE, and RUN.

What is the code analyzer message? And how can we solve this problem?

Why does the code analyzer think `highest` is a temporary variable?

>> inadvertent临时\_bug

# parfor issue: Solution



The screenshot shows the MATLAB Editor window with the following code:

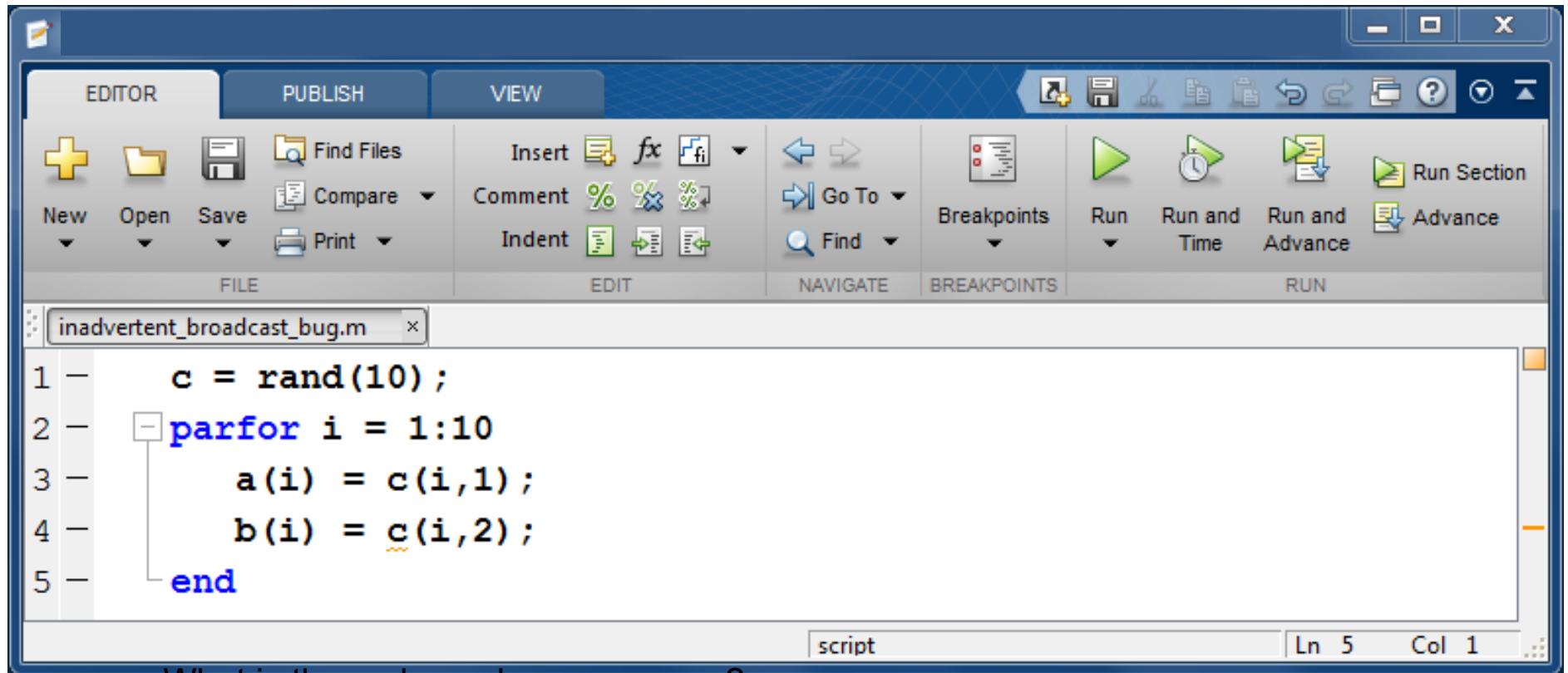
```
inadvertent_temporary_fix.m
1 highest = -inf;
2 parfor idx = 1:10
3     value = idx;
4     highest = max(highest,value);
5 end
```

The code defines a variable `highest` and initializes it to `-inf`. It then uses a `parfor` loop to iterate from 1 to 10. Inside the loop, it assigns the current index `idx` to `value`, and then updates `highest` to be the maximum of the current `highest` value and `value`. Finally, it ends the loop.

Assign `highest` to the result of a reduction function

```
>> inadvertent_temporary_fix
```

# parfor issue: Inadvertently Creating Broadcast Variables



The screenshot shows the MATLAB Editor window with the following details:

- Toolbar:** Includes buttons for New, Open, Save, Find Files, Compare, Print, Insert, Comment, Indent, Go To, Breakpoints, Run, Run and Time, Run and Advance, and Advance.
- MenuBar:** Shows EDITOR, PUBLISH, and VIEW tabs.
- Code Area:** Displays the following MATLAB script:

```

1 - c = rand(10);
2 - parfor i = 1:10
3 -     a(i) = c(i,1);
4 -     b(i) = c(i,2);
5 - end
    
```
- Status Bar:** Shows "script" at Ln 5 Col 1.

What is the code analyzer message?

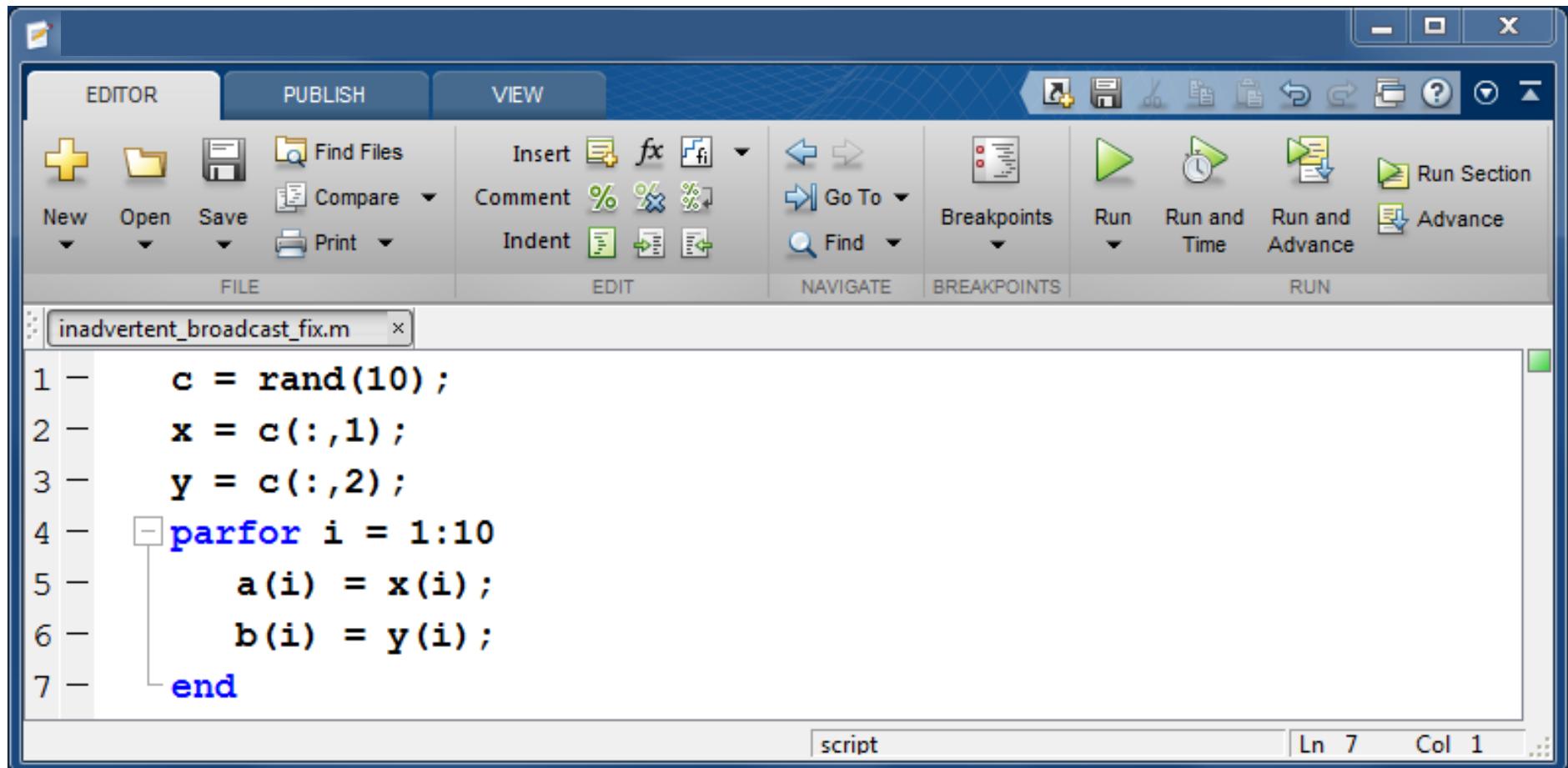
Why isn't `c` a sliced variable? What kind is it?

How can we make it sliced?

If we didn't have the `b` assignment, would `c` be sliced?

>> inadvertent\_broadcast\_bug

# parfor issue: Solution



The screenshot shows the MATLAB Editor window with the following code:

```
1 - c = rand(10);
2 - x = c(:,1);
3 - y = c(:,2);
4 - parfor i = 1:10
5 -     a(i) = x(i);
6 -     b(i) = y(i);
7 - end
```

The code defines a matrix *c*, creates two sliced variables *x* and *y*, and then uses a *parfor* loop to assign values to arrays *a* and *b*. The editor interface includes tabs for EDITOR, PUBLISH, and VIEW, and toolbars for FILE, EDIT, NAVIGATE, and RUN.

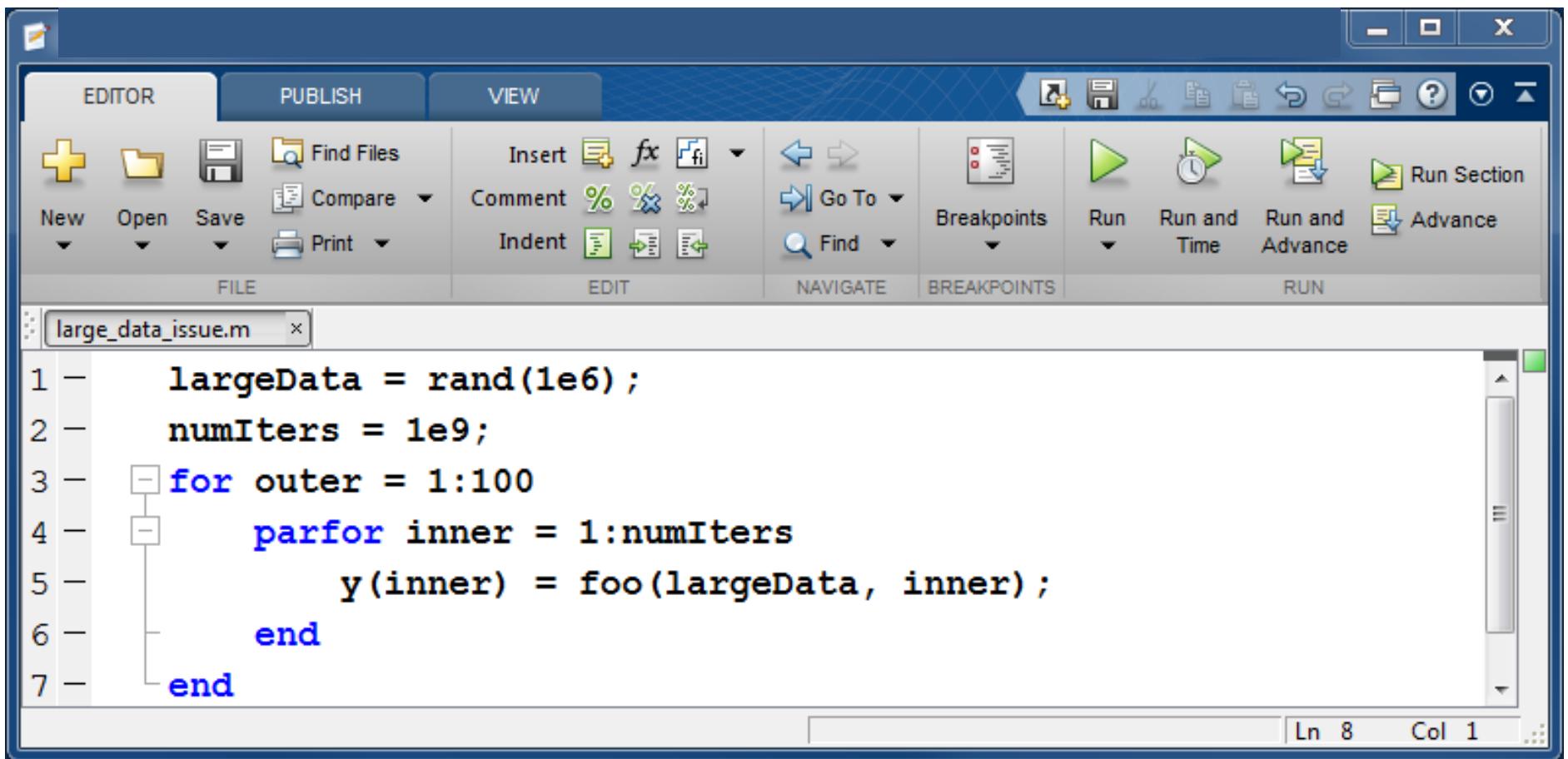
Create the additional variables *x* and *y*, which are sliced

```
>> inadvertent_broadcast_fix
```

## Persistent Storage (1)

- I cannot convert the outer loop into `parfor` because it's in someone else's top level function. However, if I convert the inner loop into `parfor` in the straightforward manner, we end up sending large data to the workers  $N$  times.

## Persistent Storage (2)

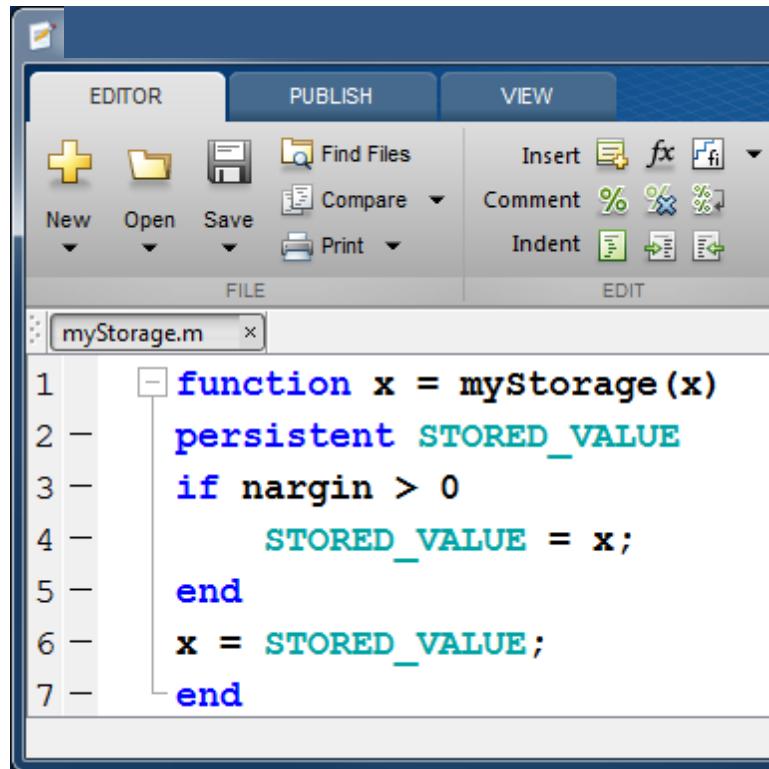


The screenshot shows the MATLAB Editor window with the following details:

- Toolbar:** Includes buttons for New, Open, Save, Find Files, Compare, Print, Insert, Comment, Indent, Go To, Breakpoints, Find, Run, Run and Time, Run and Advance, and Advance.
- Menu Bar:** Shows FILE, EDIT, NAVIGATE, and BREAKPOINTS.
- Code Area:** A file named "large\_data\_issue.m" is open, containing the following MATLAB code:

```
1 -     largeData = rand(1e6);
2 -     numIters = 1e9;
3 -     for outer = 1:100
4 -         parfor inner = 1:numIters
5 -             y(inner) = foo(largeData, inner);
6 -         end
7 -     end
```
- Status Bar:** Shows Ln 8 Col 1.

# Solution: Persistent Storage

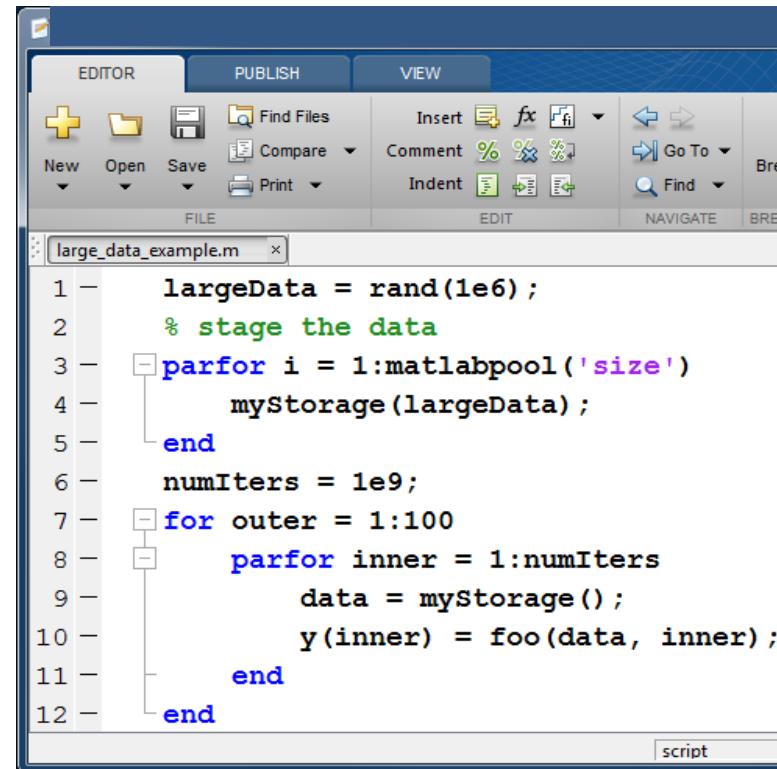


```

EDITOR          PUBLISH        VIEW
New Open Save  Find Files  Insert fx fi
Find Files  Compare Comment % % % %
Open       Print   Indent  % % % %
Save      Go To   Find   % % % %
FILE           EDIT           NAVIGATE
myStorage.m x

function x = myStorage(x)
persistent STORED_VALUE
if nargin > 0
    STORED_VALUE = x;
end
x = STORED_VALUE;
end

```



```

EDITOR          PUBLISH        VIEW
New Open Save  Find Files  Insert fx fi
Find Files  Compare Comment % % % %
Open       Print   Indent  % % % %
Save      Go To   Find   % % % %
FILE           EDIT           NAVIGATE
large_data_example.m x

1 -     largeData = rand(1e6);
2 - % stage the data
3 - parfor i = 1:matlabpool('size')
4 -     myStorage(largeData);
5 - end
6 - numIters = 1e9;
7 - for outer = 1:100
8 -     parfor inner = 1:numIters
9 -         data = myStorage();
10 -        y(inner) = foo(data, inner);
11 -    end
12 - end
script

```

Store the value in a persistent variable in a function



# Best Practices for Converting `for` to `parfor`

- Use code analyzer to diagnose `parfor` issues
- If your `for` loop cannot be converted to a `parfor`, consider wrapping a subset of the body to a function
- If you modify your `parfor` loop, switch back to a `for` loop for regression testing
- Read the section on classification of variables

```
>> docsearch 'Classification of Variables'
```

# Outline

- Parallelizing Your MATLAB Code
- Tips for Programming with a Parallel for Loop
- Computing to a GPU
- Scaling to a Cluster
- Debugging and Troubleshooting

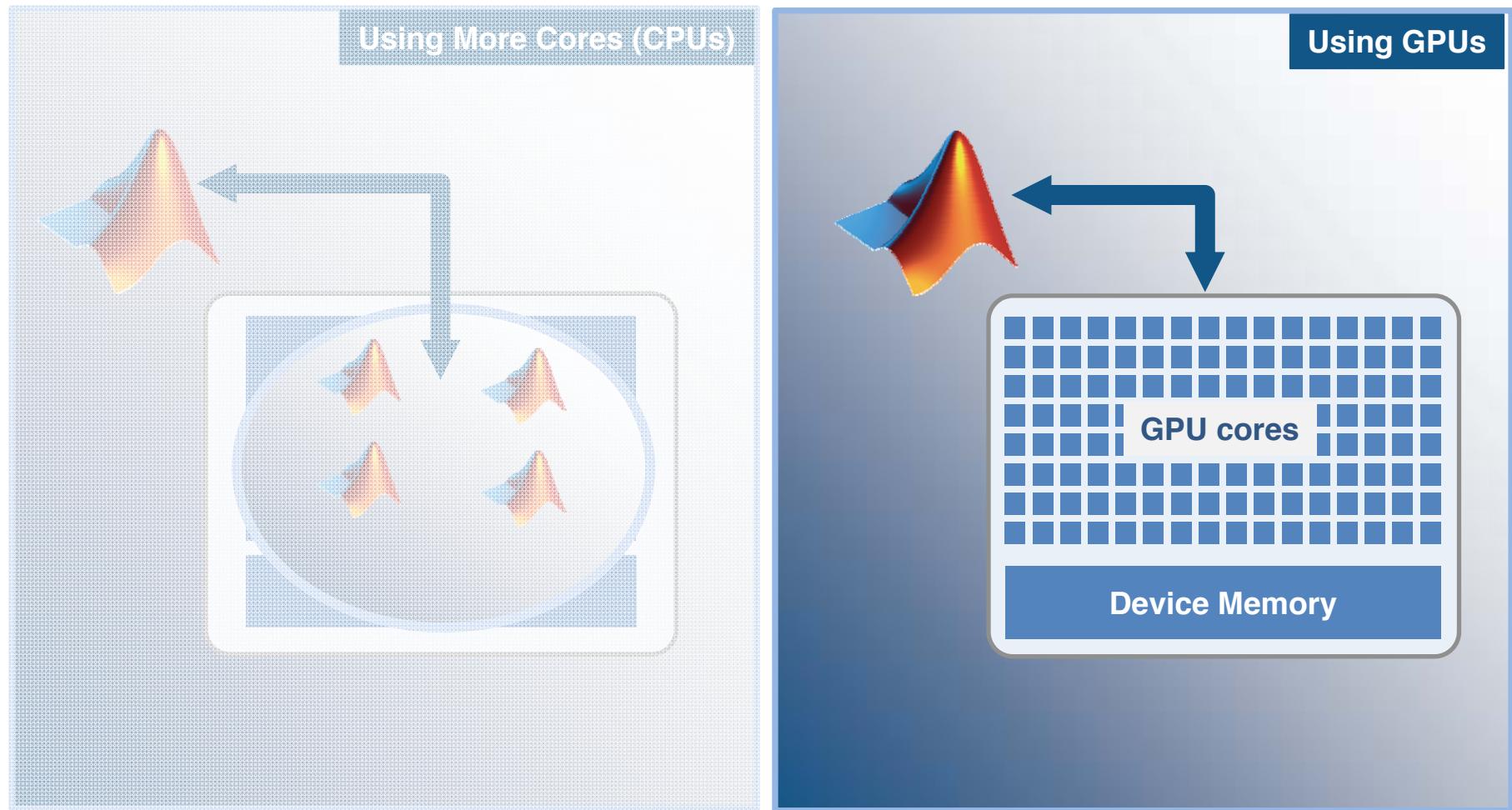
# What is a Graphics Processing Unit (GPU)

- Originally for graphics acceleration, now also used for scientific calculations
- Massively parallel array of integer and floating point processors
  - Typically hundreds of processors per card
  - GPU cores complement CPU cores
- Dedicated high-speed memory
- [blogs.mathworks.com/loren/2013/06/24/running-monte-carlo-simulations-on-multiple-gpus](http://blogs.mathworks.com/loren/2013/06/24/running-monte-carlo-simulations-on-multiple-gpus)

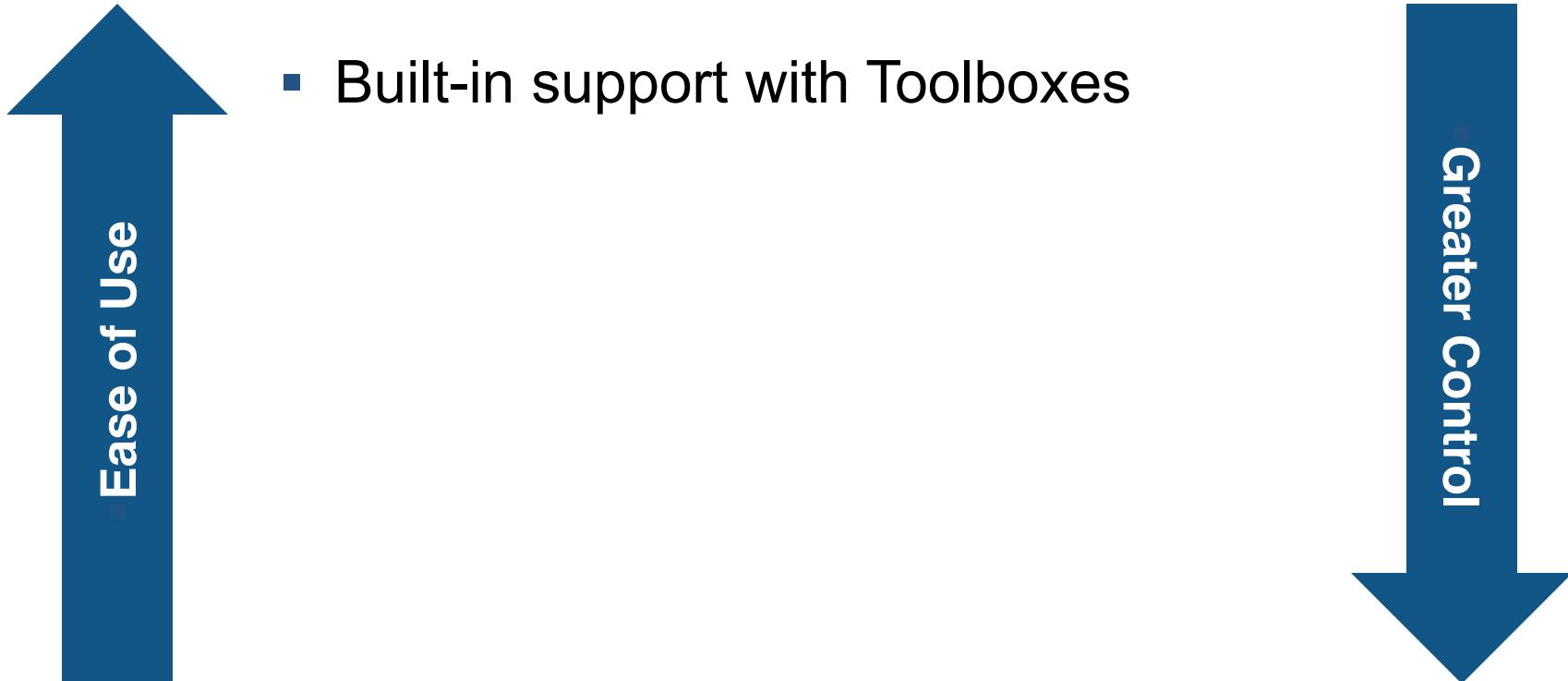


\* Parallel Computing Toolbox requires NVIDIA GPUs with Compute Capability 1.3 or higher, including NVIDIA Tesla 20-series products. See a complete listing at [www.nvidia.com/object/cuda\\_gpus.html](http://www.nvidia.com/object/cuda_gpus.html)

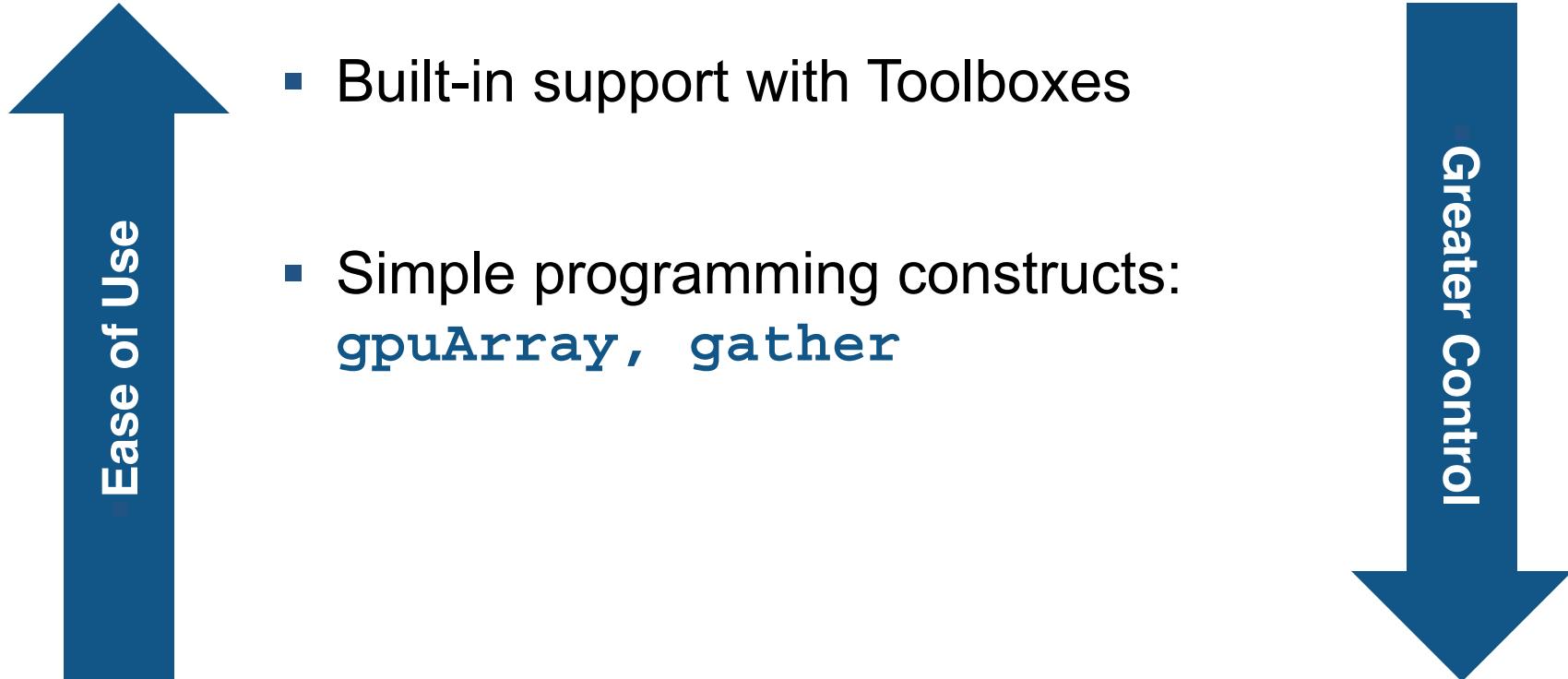
# Performance Gain with More Hardware



# Programming Parallel Applications (GPU)



# Programming Parallel Applications (GPU)



# Example: Solving 2D Wave Equation

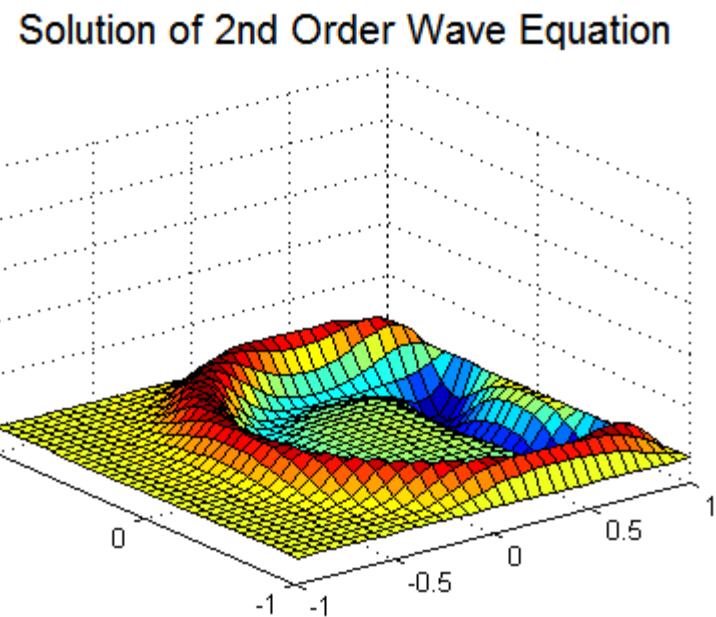
## GPU Computing

- Solve 2<sup>nd</sup> order wave equation using spectral methods:

$$\frac{\partial^2 u}{\partial t^2} = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}$$

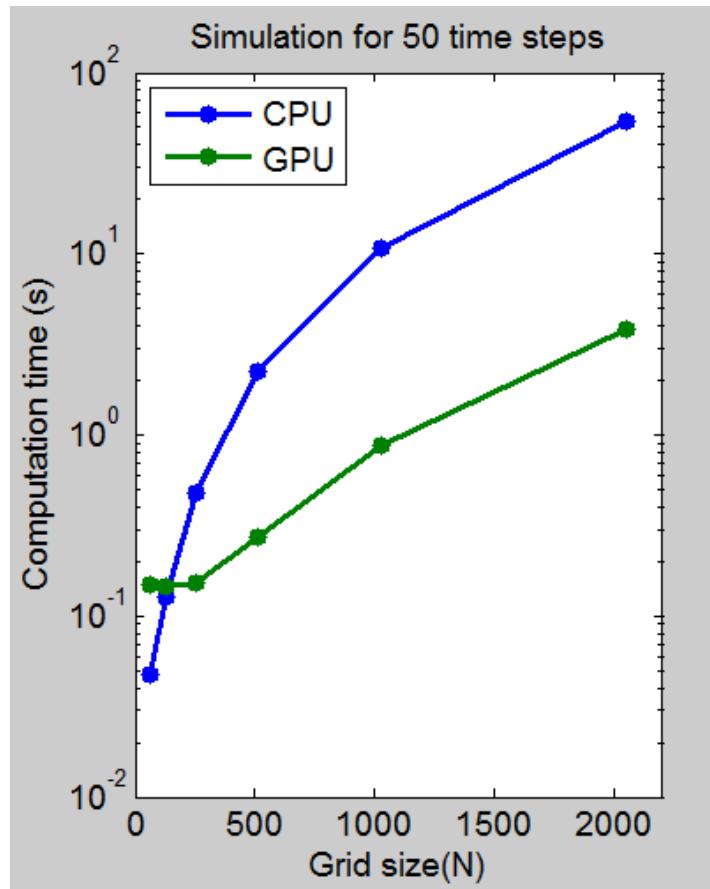
- Run both on CPU and GPU
- Using `gpuArray` and overloaded functions

[www.mathworks.com/help/distcomp/using-gpuarray.html#bsloua3-1](http://www.mathworks.com/help/distcomp/using-gpuarray.html#bsloua3-1)



# Benchmark: Solving 2D Wave Equation

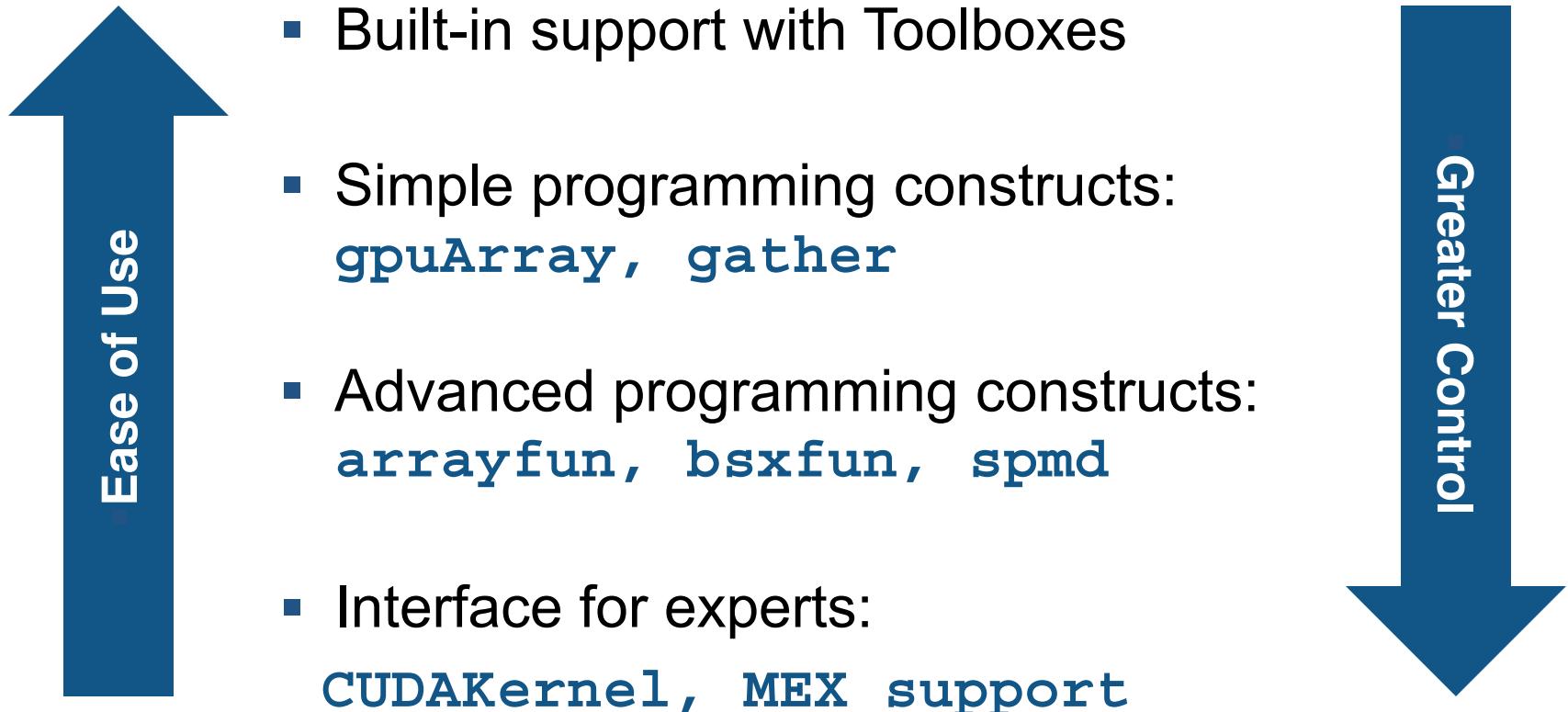
## CPU vs GPU



Grid Size	CPU (s)	GPU (s)	Speedup
64 x 64	0.05	0.15	0.32
128 x 128	0.13	0.15	0.88
256 x 256	0.47	0.15	3.12
512 x 512	2.22	0.27	8.10
1024 x 1024	10.80	0.88	12.31
2048 x 2048	54.60	3.84	14.22

Intel Xeon Processor W3690 (3.47GHz),  
NVIDIA Tesla K20 GPU

# Programming Parallel Applications (GPU)



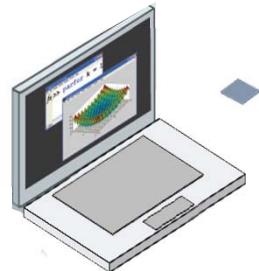
[www.mathworks.com/help/releases/R2013a/distcomp/executing-cuda-or-ptx-code-on-the-gpu.html](http://www.mathworks.com/help/releases/R2013a/distcomp/executing-cuda-or-ptx-code-on-the-gpu.html)

[www.mathworks.com/help/releases/R2013a/distcomp/create-and-run-mex-files-containing-cuda-code.html](http://www.mathworks.com/help/releases/R2013a/distcomp/create-and-run-mex-files-containing-cuda-code.html)

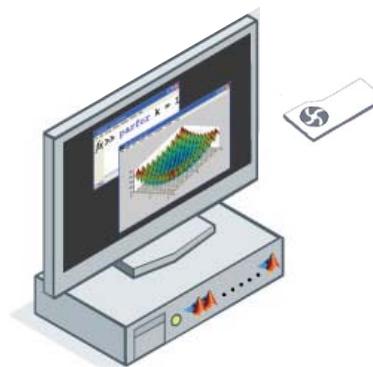
# GPU Performance – not all cards are equal

- Tesla-based cards will provide best performance
- Realistically, expect 4x to 15x speedup (Tesla) vs CPU
- See GPUBench on MATLAB Central for examples

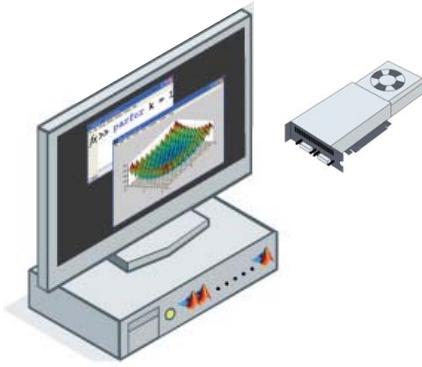
[www.mathworks.com/matlabcentral/fileexchange/34080-gpubench](http://www.mathworks.com/matlabcentral/fileexchange/34080-gpubench)



Laptop GPU  
GeForce



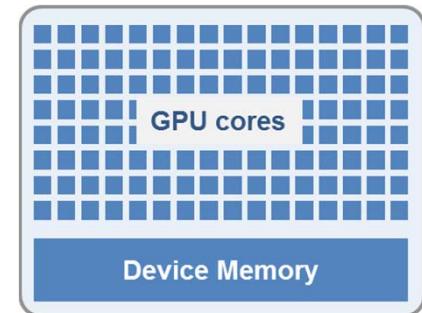
Desktop GPU  
GeForce / Quadro



High Performance Computing GPU  
Tesla / Quadro

# Criteria for Good Problems to Run on a GPU

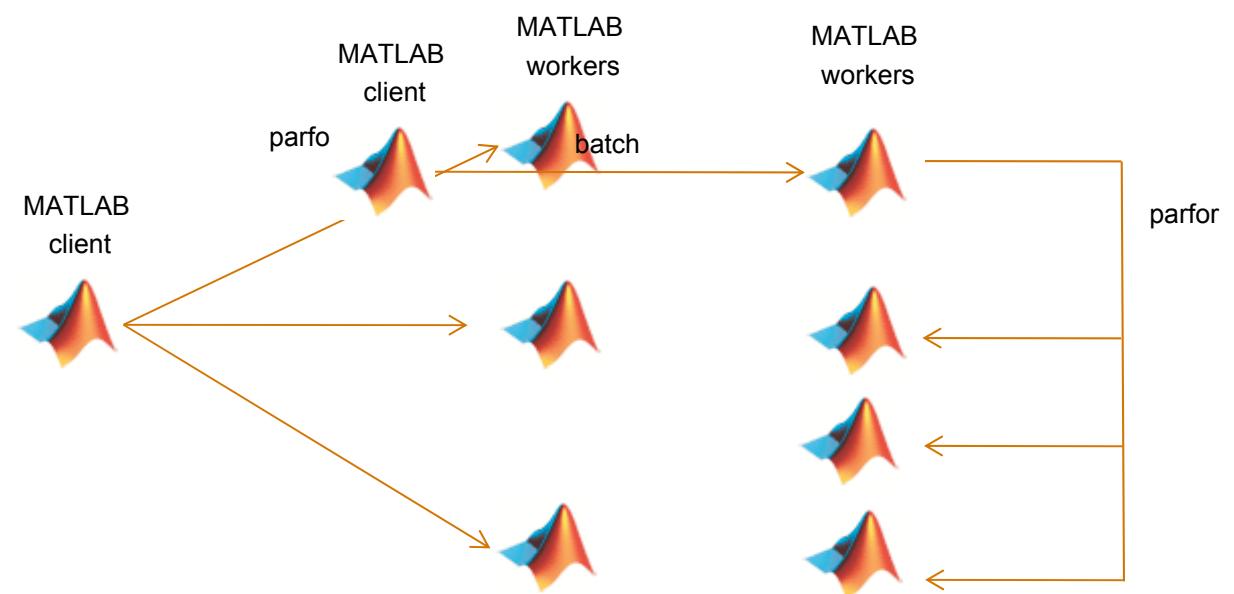
- **Massively parallel:**
  - Calculations can be broken into hundreds or thousands of independent units of work
  - Problem size takes advantage of many GPU cores
- **Computationally intensive:**
  - Computation time significantly exceeds CPU/GPU data transfer time
- **Algorithm consists of supported functions:**
  - Growing list of Toolboxes with built-in support
    - [www.mathworks.com/products/parallel-computing/builtin-parallel-support.html](http://www.mathworks.com/products/parallel-computing/builtin-parallel-support.html)
  - Subset of core MATLAB for `gpuArray`, `arrayfun`, `bsxfun`
    - [www.mathworks.com/help/distcomp/using-gpuarray.html#bsloua3-1](http://www.mathworks.com/help/distcomp/using-gpuarray.html#bsloua3-1)
    - [www.mathworks.com/help/distcomp/execute-matlab-code-elementwise-on-a-gpu.html#bsnx7h8-1](http://www.mathworks.com/help/distcomp/execute-matlab-code-elementwise-on-a-gpu.html#bsnx7h8-1)



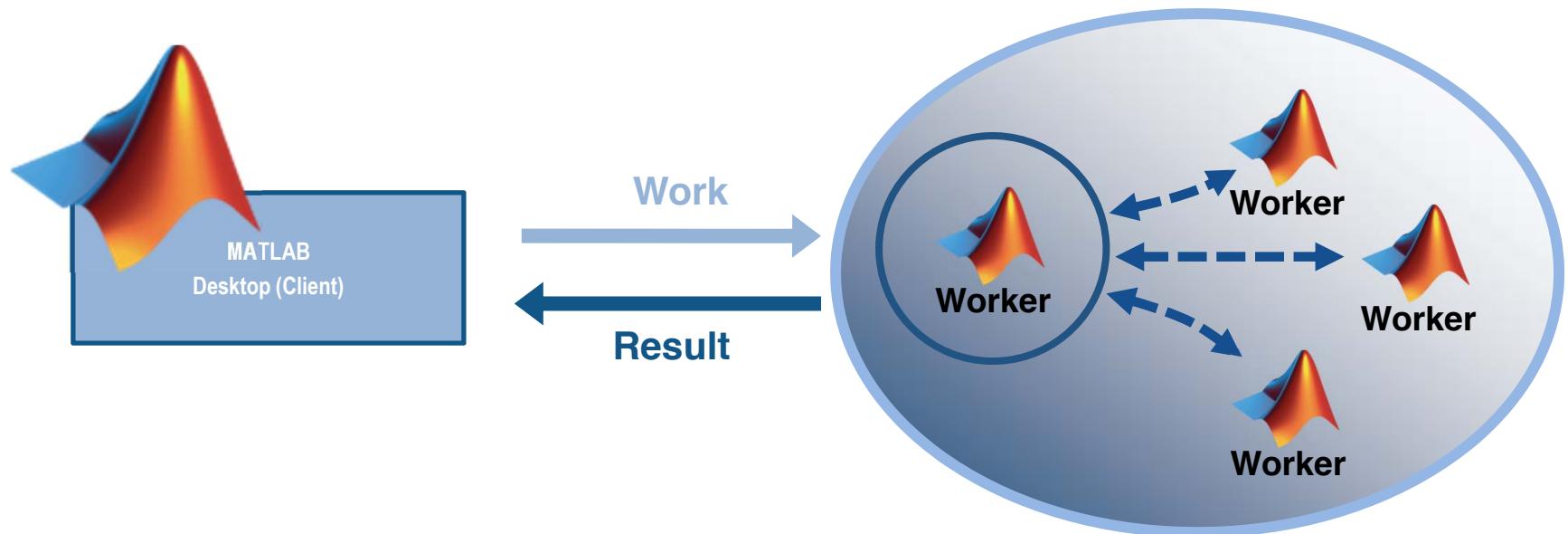
# Outline

- Parallelizing Your MATLAB Code
- Tips for Programming with a Parallel for Loop
- Computing to a GPU
- Scaling to a Cluster
- Debugging and Troubleshooting

# Migrating from Local to Cluster



# Offload Computations with `batch`

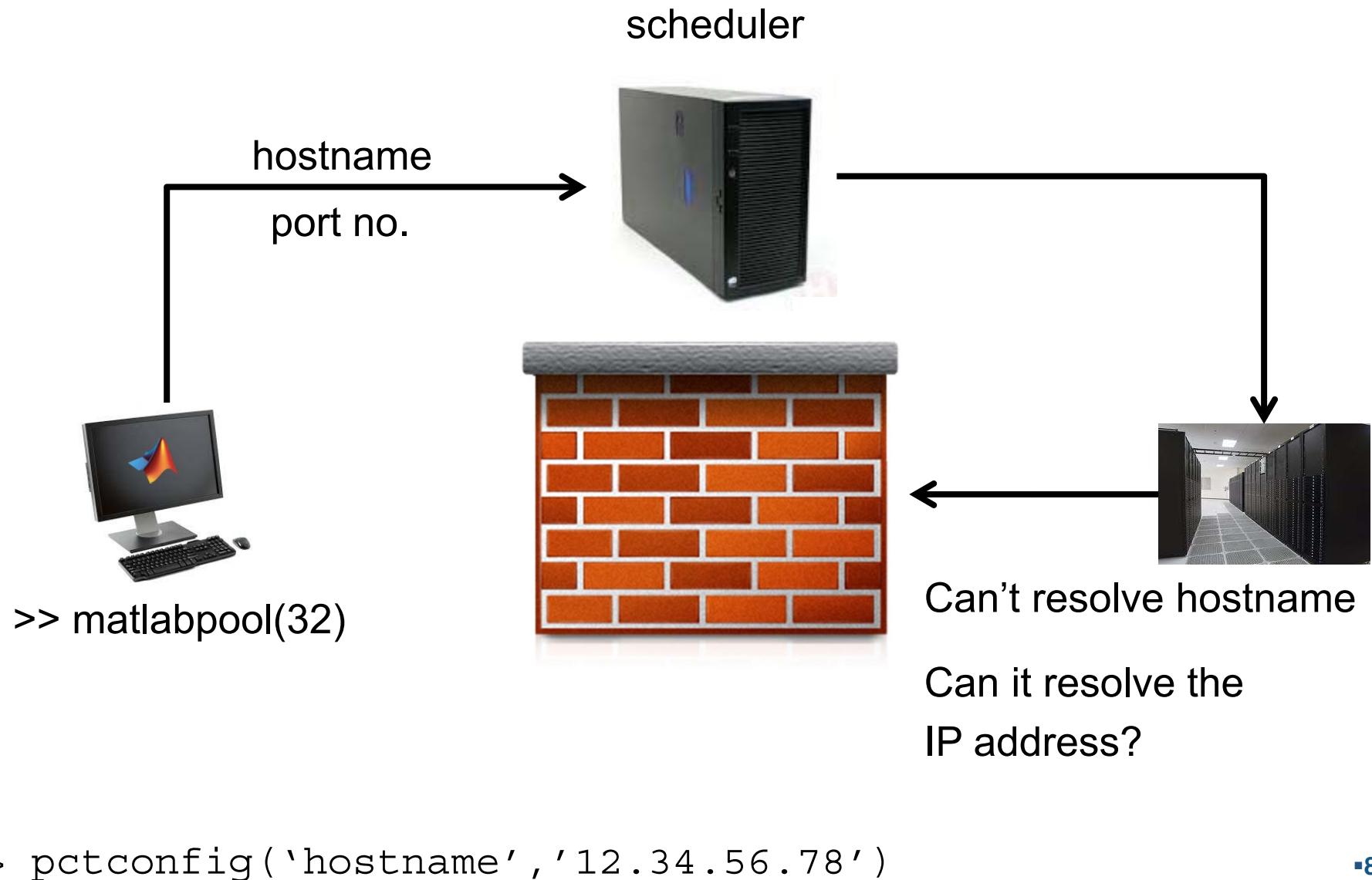




# Can't I Just Use `matlabpool` to Connect to the Cluster/Cloud?

- MATLAB pool
  - So long as the compute nodes can reach back to your local desktop, then yes, you can run jobs on the cluster using `matlabpool`
  - Recall, the MATLAB Client is blocked
  - Cannot run other parallel jobs
  - Consumes MDCS licenses while the pool is open, even if they aren't being used
- Batch
  - Ideal if:
    - the local desktop is not reachable from the cluster, or
    - if I want shutdown my desktop, or
    - if I want submit multiple jobs at once

# Why Can't I Open a MATLAB Pool to the Cluster?

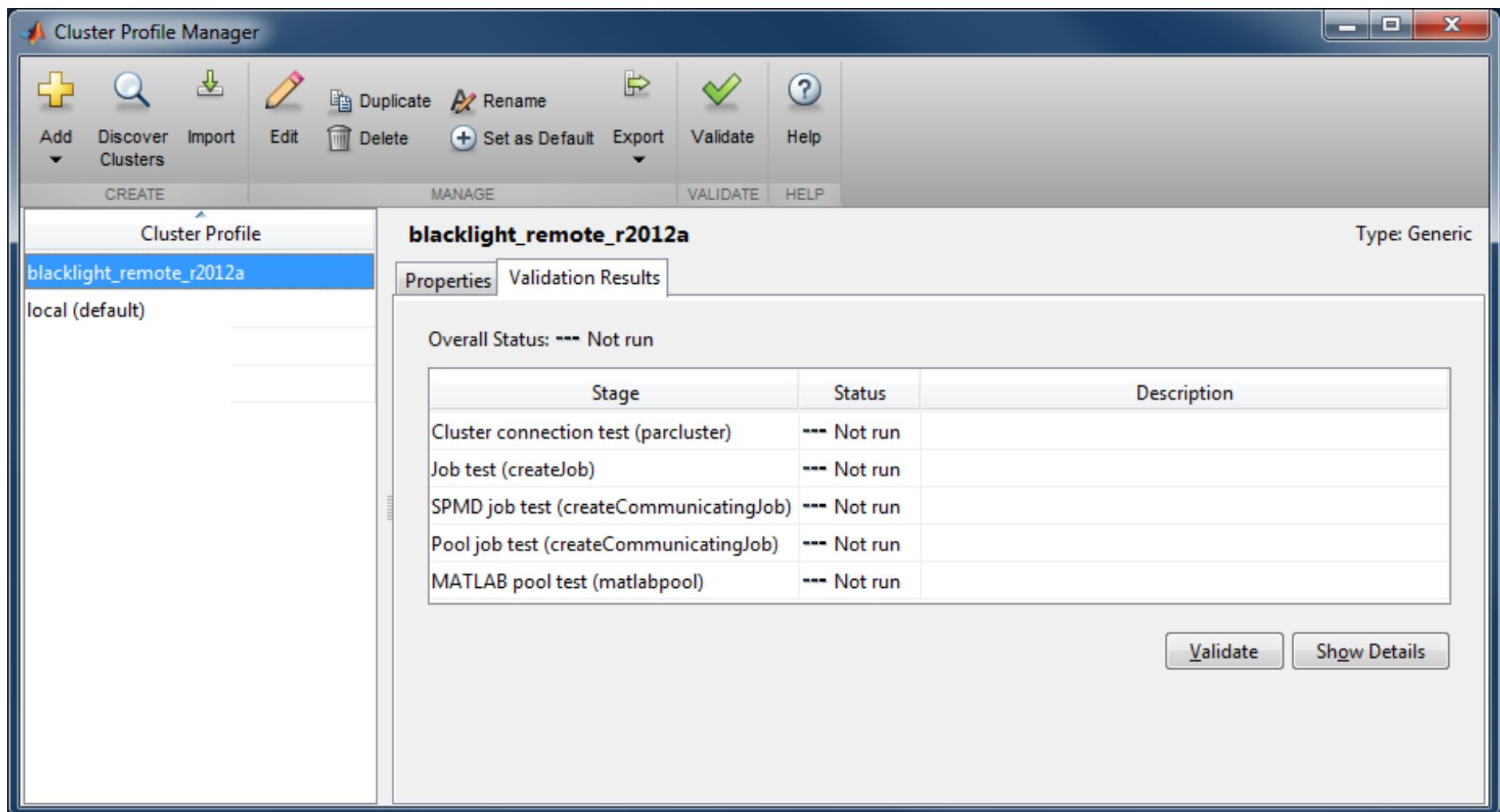




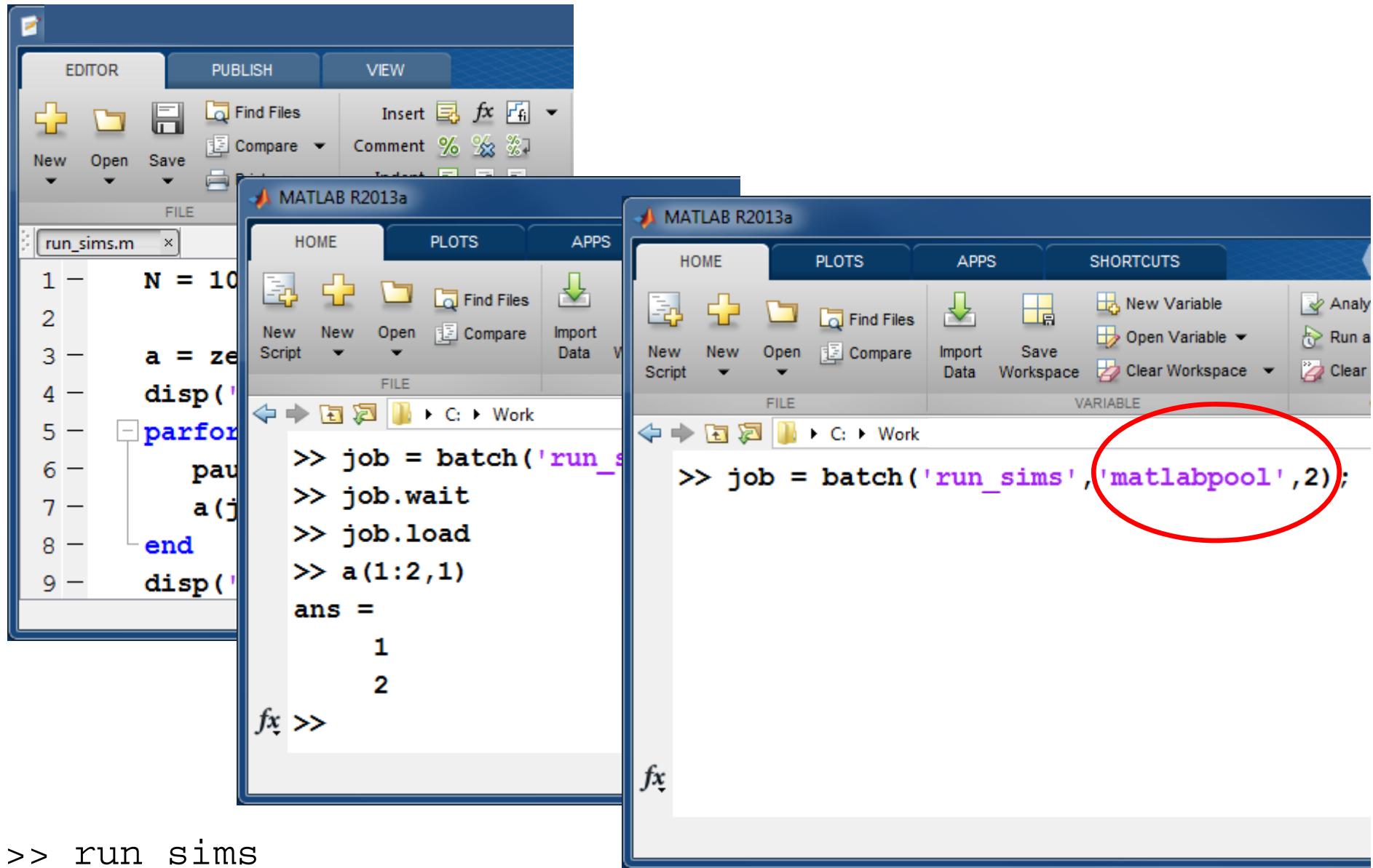
# Profiles

- Think of cluster profiles like printer queue configurations
- Managing profiles
  - Typically created by Sys Admins
  - Label profiles based on the version of MATLAB
    - E.g. *hpcc\_local\_r2013a*
- Import profiles generated by the Sys Admin
  - Don't modify them with two exceptions
    - Specify the JobStorageLocation
    - Setting the ClusterSize
- Validate profiles
  - Ensure new profile is properly working
  - Helpful when debugging failed jobs

# Import and Validating a Profile



# Submitting Scripts with batch



The image shows three windows of the MATLAB R2013a interface:

- Editor:** A script named `run_sims.m` is open. It contains the following code:

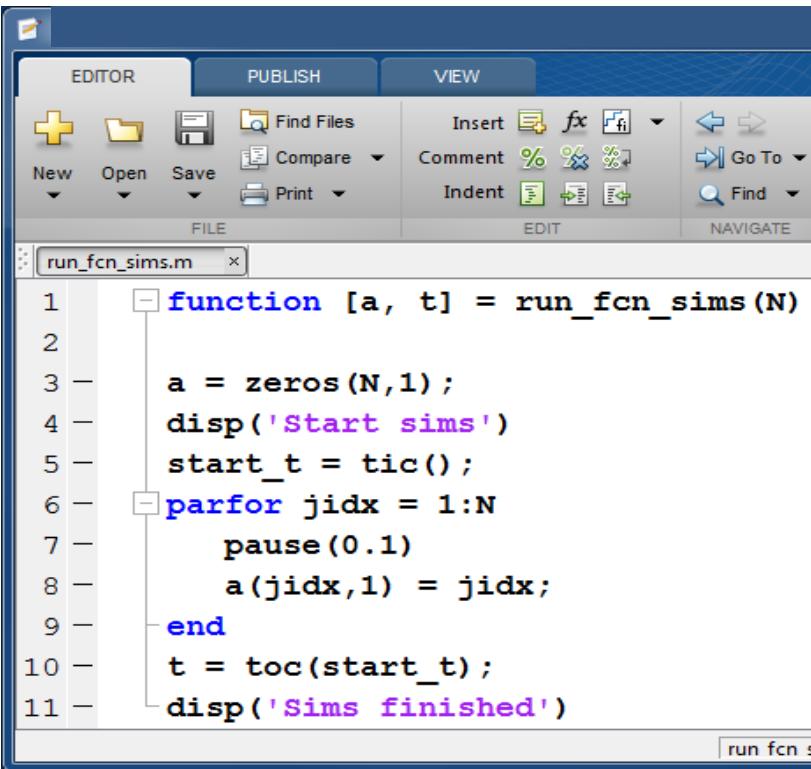
```
N = 10
a = zeros(N);
disp('Starting simulations')
parfor j = 1:N
    pause(0.1);
    a(j) = j;
end
disp('Simulations completed')
```
- Home Tab:** The `HOME` tab is selected. The command bar at the top shows:

```
>> job = batch('run_sims');
```
- Command Window:** The command window shows the continuation of the command:

```
>> job.wait
>> job.load
>> a(1:2,1)
ans =
    1
    2
```

A red circle highlights the part of the command `'matlabpool', 2)` in the Editor's command bar.

# Submitting Functions with batch

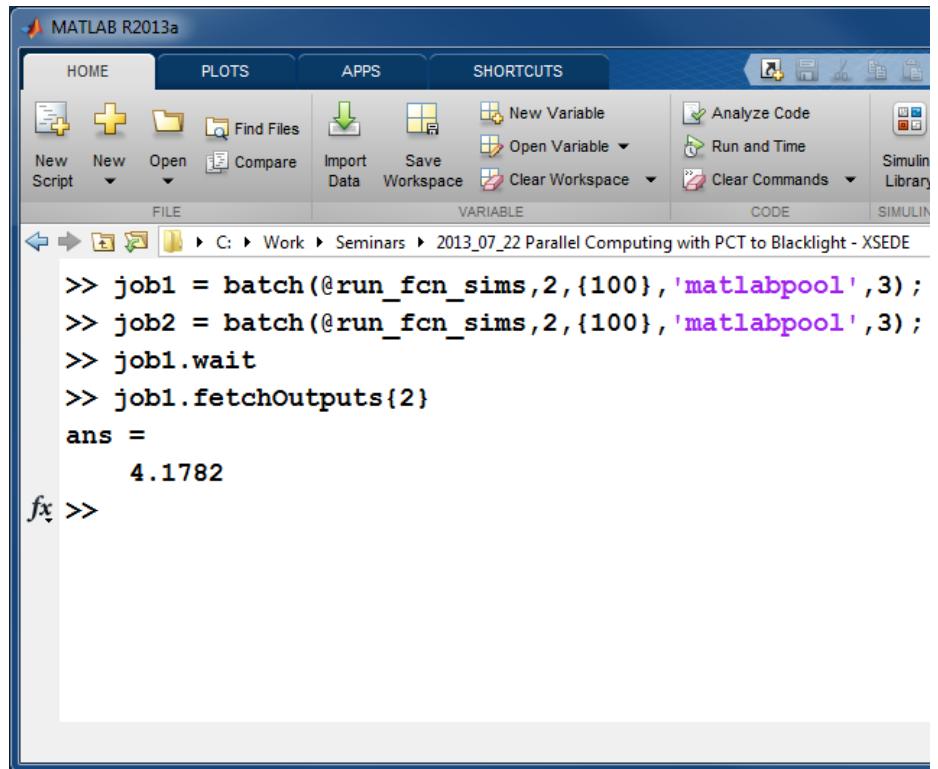


The screenshot shows the MATLAB Editor window with the file `run_fcn_sims.m` open. The code implements a parallel simulation using a parfor loop:

```

function [a, t] = run_fcn_sims(N)
a = zeros(N,1);
disp('Start sims')
start_t = tic();
parfor jidx = 1:N
    pause(0.1)
    a(jidx,1) = jidx;
end
t = toc(start_t);
disp('Sims finished')

```



The screenshot shows the MATLAB Command Window with the following session:

```

>> job1 = batch(@run_fcn_sims,2,{100},'matlabpool',3);
>> job2 = batch(@run_fcn_sims,2,{100},'matlabpool',3);
>> job1.wait
>> job1.fetchOutputs(2)
ans =
    4.1782
fx >>

```

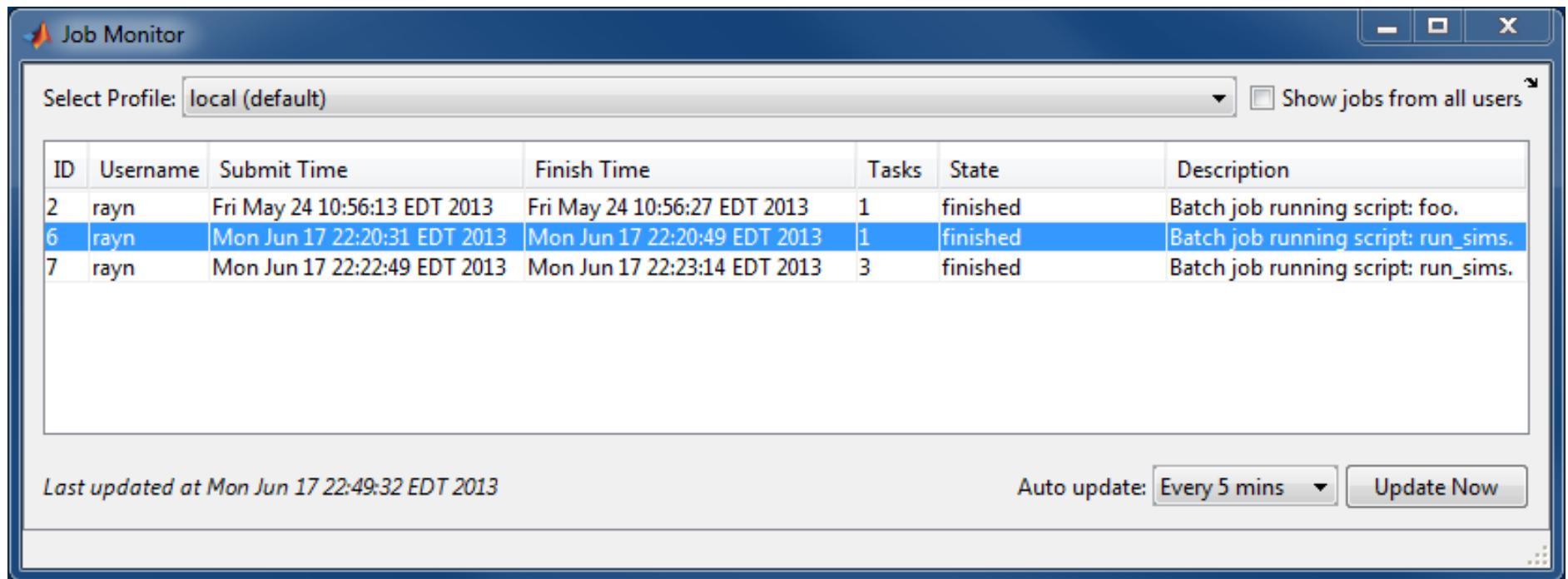
```
>> run_fcn_sims
```

# Fixing the batch Warning Message

Warning: Unable to change to requested working directory.  
Reason :Cannot CD to C:\Work (Name is nonexistent or not  
a directory) .

- Call batch with CurrentFolder set to '.'
  - `job = batch(..., 'CurrentFolder', '.') ;`

# How Can I Find Yesterday's Job?

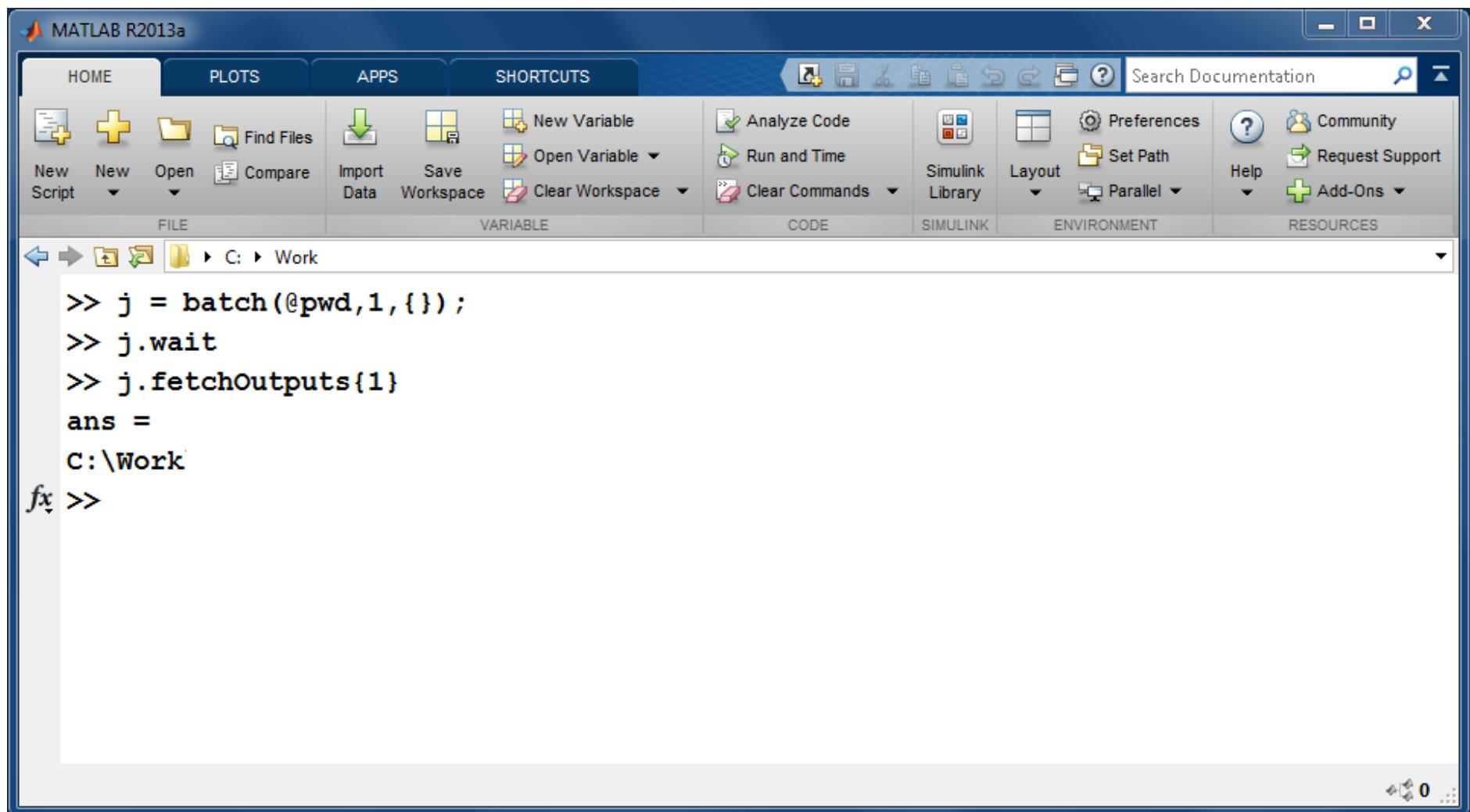


Job Monitor

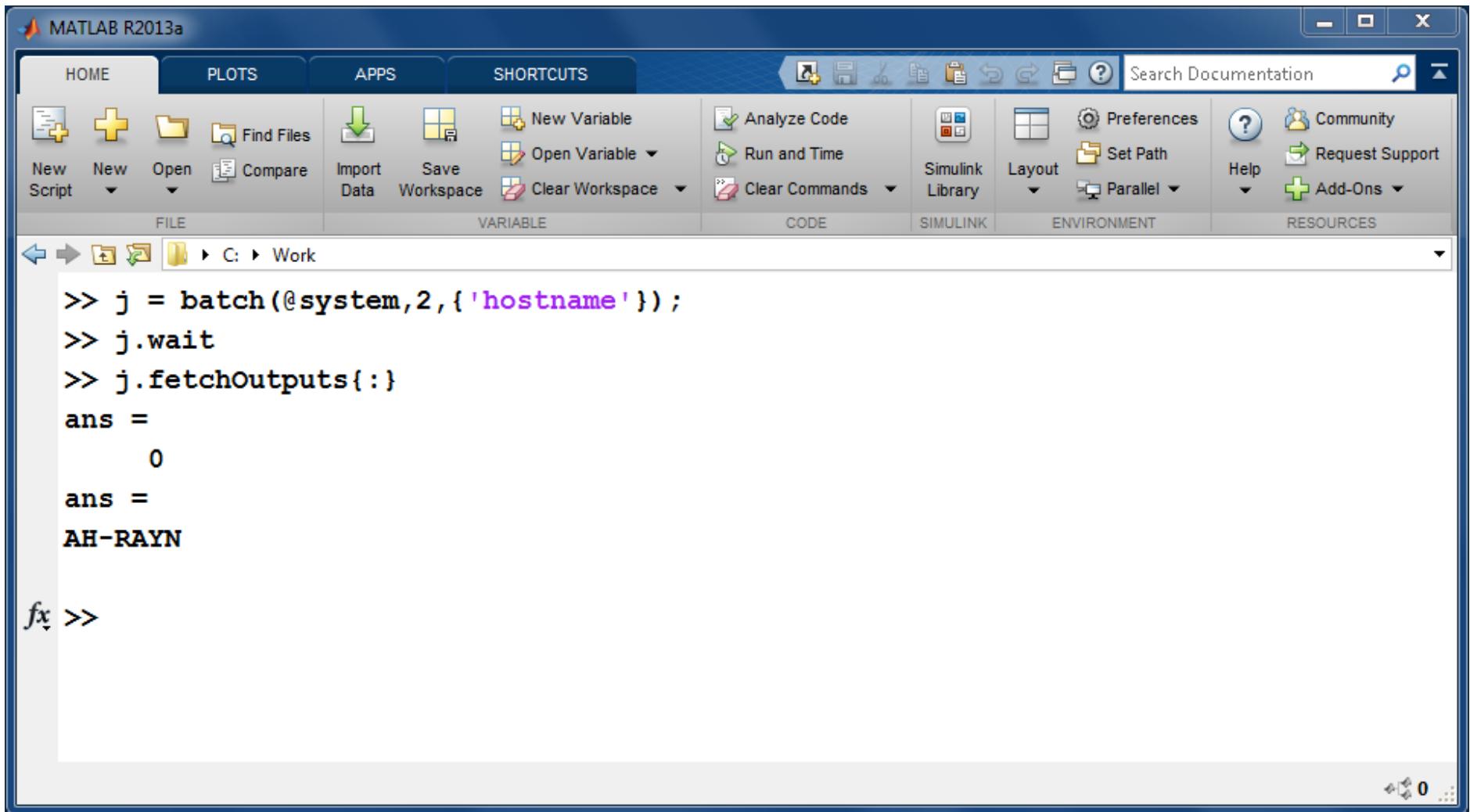
## Final Exam: What Final Exam?

- Choose one of the following:
  - → Submit a job that determines the MATLAB directory your task ran in
  - Submit a job that determines the machine that ran your task
    - Hint: system(), hostname.exe
- Clear your MATLAB workspace and get a handle to the job you ran above

# Final Exam: Solution (1)



# Final Exam: Solution (2)



The screenshot shows the MATLAB R2013a interface. The top menu bar includes HOME, PLOTS, APPS, and SHORTCUTS. The HOME tab is active, displaying various file operations like New Script, New, Open, Find Files, Import Data, Save Workspace, and Clear Workspace. The central workspace shows the following command history:

```
>> j = batch(@system,2,{'hostname'});  
>> j.wait  
>> j.fetchOutputs{ : }  
ans =  
    0  
ans =  
AH-RAYN  
  
fx >>
```



# Recommendations

- Profile your code to search for bottlenecks
- Make use of **code analyzer** when coding `parfor` and `spmd`
- Display the correct amount of verbosity for debugging purposes
- Implement an error handler, including capture of calls to 3<sup>rd</sup> party functions – **don't assume calls to libraries succeed**
- Beware of multiple processes writing to the same file
- Avoid the use of global variables
- **Avoid hard coding path and filenames** that don't exist on the cluster
- Migrate from scripts to functions
- Consider whether or not you'll need to recompile your MEX-files
- After migrating from `for` to `parfor`, switch back to `for` to make sure nothing has broken
- If calling `rand` in a `for` loop, while debugging call `rand('seed',0)`, to get consistent results each time
- When calling `matlabpool/batch`, parameterize your code

# Outline

- Parallelizing Your MATLAB Code
- Tips for Programming with a Parallel for Loop
- Computing to a GPU
- Scaling to a Cluster
- Debugging and Troubleshooting

# Troubleshooting and Debugging

- Object data size limitations
  - Single transfers of data between client and workers

System Architecture	Maximum Data Size Per Transfer (approx.)
64-bit	2.0 GB
32-bit	600 MB

- Tasks or jobs remain in Queued state even though cluster scheduler states it's finished
  - Most likely MDCS failed to startup
- No results or job failed
  - `job.load` or `job.fetchOutputArguments{()}`
  - `job.Parent.getDebugLog(job)`

# System Support

# System Requirements

- Maximum 1 MATLAB worker / CPU core
- Minimum 1 GB RAM / MATLAB worker
- Minimum 5 GB of disk space for temporary data directories
- GPU
  - CUDA-enabled NVIDIA GPU w/ compute capability 1.3 or above <http://www.nvidia.com/content/cuda/cuda-gpus.html>
  - Latest CUDA driver  
<http://www.nvidia.com/Download/index.aspx>

## What's New In R2013a?

- GPU-enabled functions in Image Processing Toolbox and Phased Array System Toolbox
- More MATLAB functions enabled for use with GPUs, including `interp1` and `ismember`
- Enhancements to MATLAB functions enabled for GPUs, including `arrayfun`, `svd`, and `mldivide (\)`
- Ability to launch CUDA code and manipulate data contained in GPU arrays from MEX-functions
- **Automatic detection and transfer of files required for execution in both batch and interactive workflows**
- More MATLAB functions enabled for distributed arrays

# Training: Parallel Computing with MATLAB

- Two-day course introducing tools and techniques for distributing code and writing parallel algorithms in MATLAB. The course shows how to increase both the speed and the scale of existing code using PCT.
  - Working with a MATLAB pool
  - Speeding up computations
  - Task-parallel programming
  - Working with large data sets
  - Data-parallel programming
  - Increasing scale with multiple systems
  - **Prerequisites:** *MATLAB Fundamentals*
- [mathworks.com/training](http://mathworks.com/training)