

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 \ 'klī-ənt \

1 : MATLAB session that submits the job

in·de·pen·dent job
adjective \ ,in-də-'pen-
dənt \ \ 'jāb \

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

com·mu·ni·cate job adjective
\kə-'myü-nə-'kāt \ \ 'jāb \

1 : a job composed of tasks that communicate with each other, running 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*

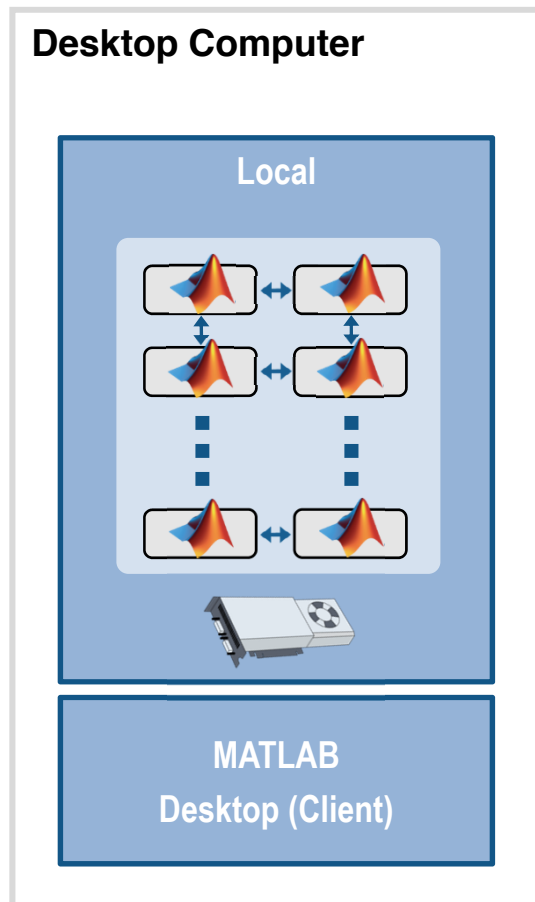
1 : MATLAB Distributed Computing
Server

SPMD *abbreviation*

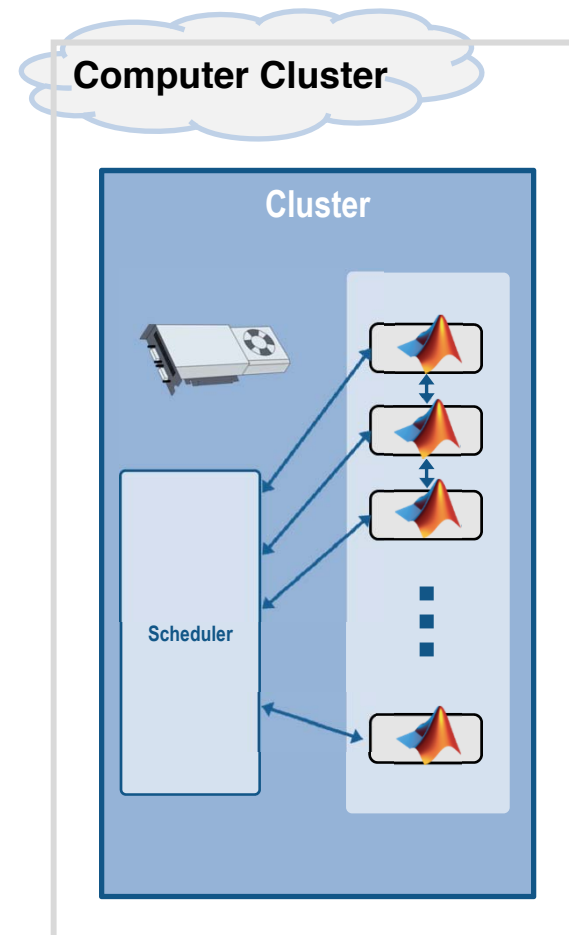
1 : Single Program Multiple
Data

worker *noun* \'wər-kər\
1 : headless MATLAB session that
performs tasks

MATLAB Parallel Computing Solution



Parallel Computing Toolbox



MATLAB Distributed Computing Server



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

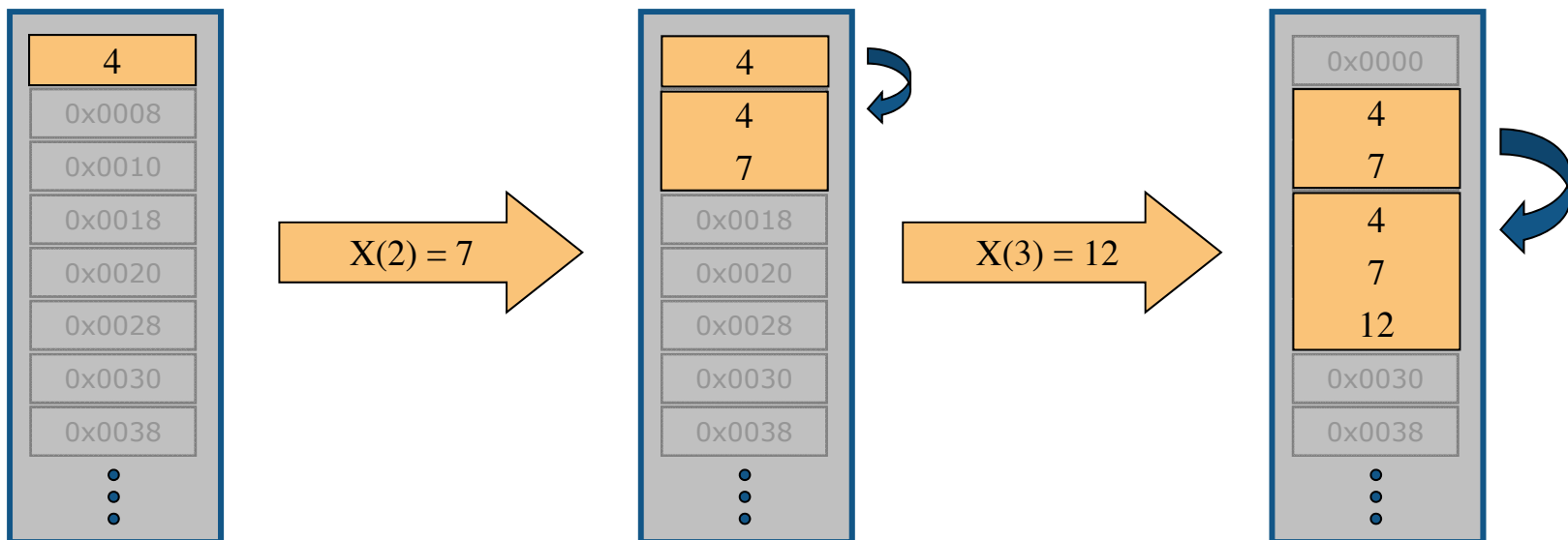
But Before We Get Started...

- Do you preallocate your matrices?

Effect of Not Preallocating Memory

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

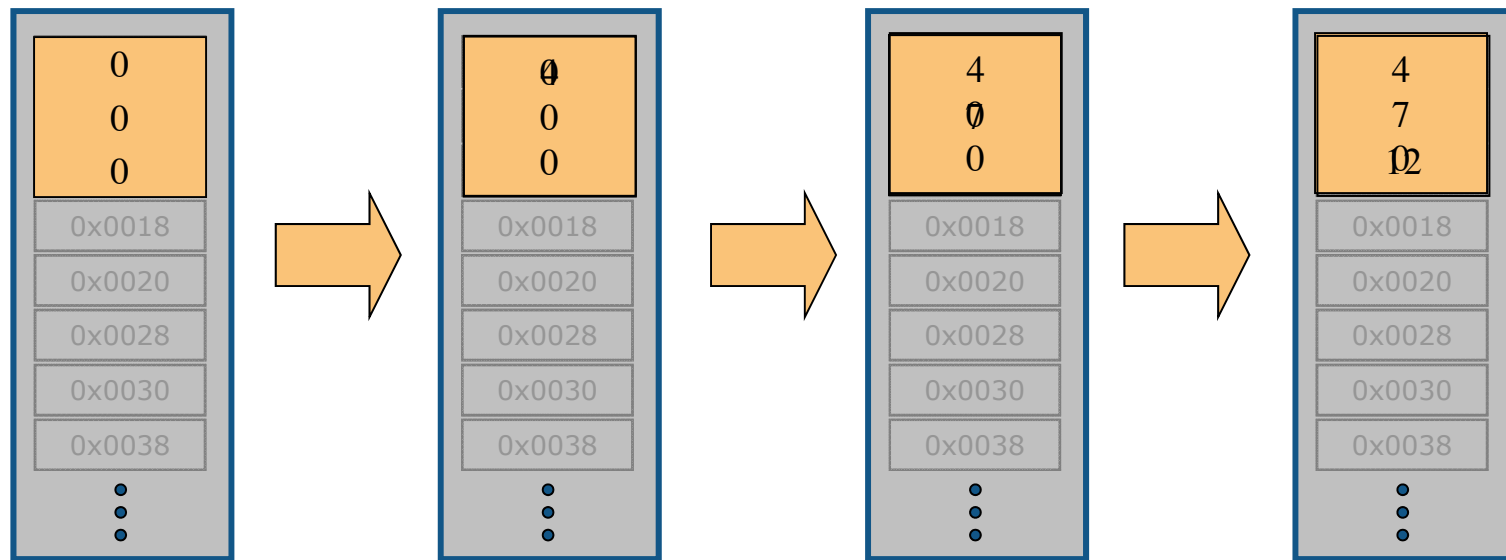
**Resizing
Arrays is
Expensive**



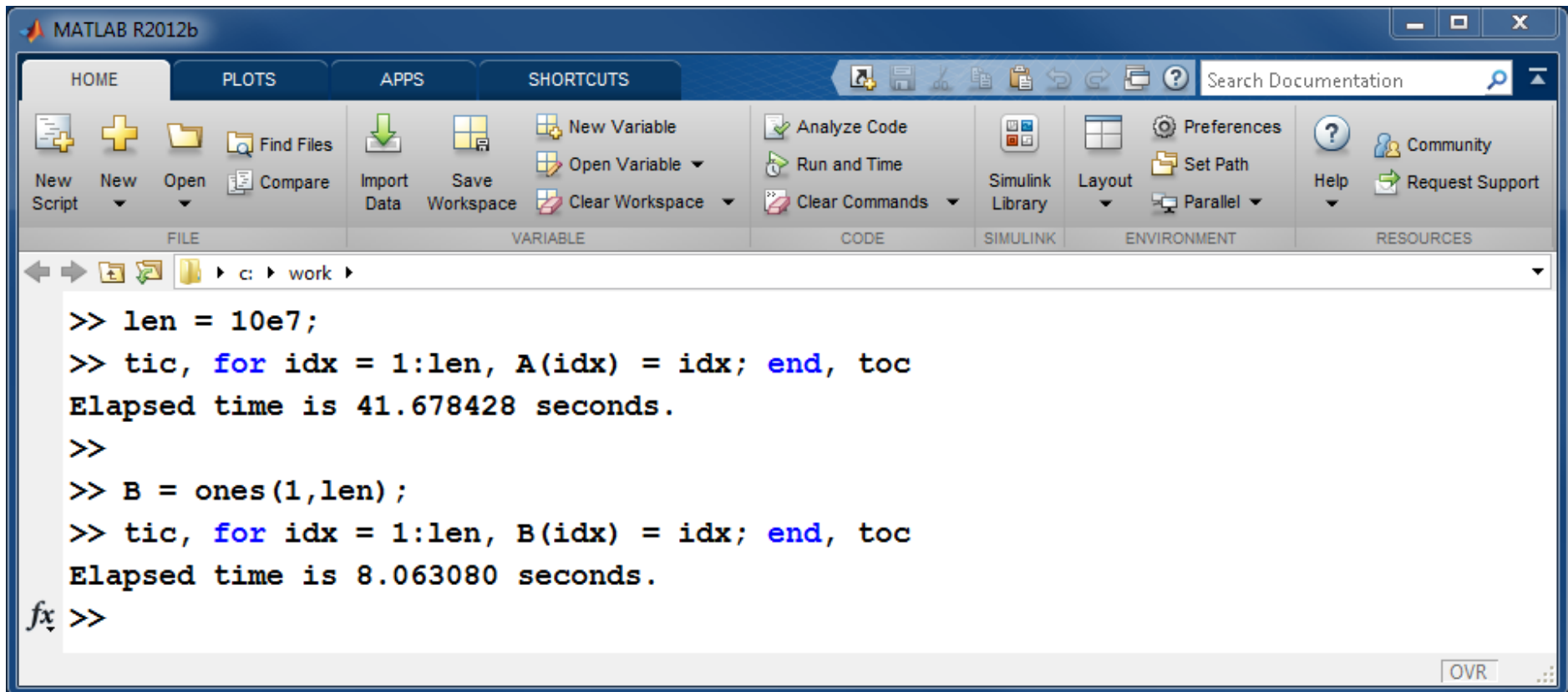
Benefit of Preallocation

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

**Reduced
Memory
Operations**



Let's Try It...



The image shows the MATLAB R2012b software interface. The Command Window displays the following code and output:

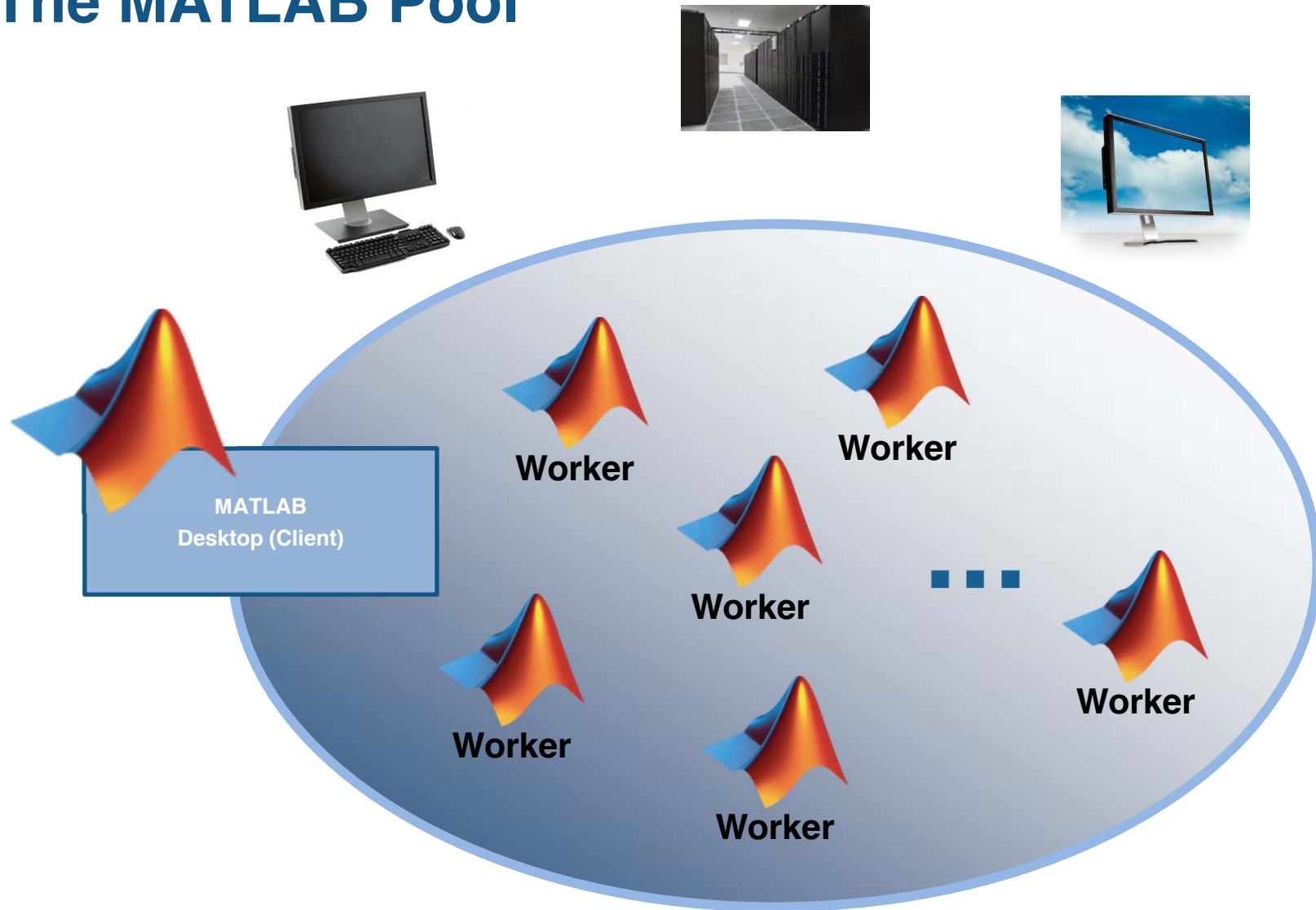
```

>> 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 >>
  
```

The interface includes a ribbon with tabs for HOME, PLOTS, APPS, and SHORTCUTS. The ribbon contains various icons for file operations (New, Open, Save), workspace management (New Variable, Open Variable, Clear Workspace), code execution (Analyze Code, Run and Time, Clear Commands), and environment settings (Preferences, Set Path, Parallel). The Command Window shows the execution of two loops, with the first loop taking 41.678428 seconds and the second loop taking 8.063080 seconds. The Command Window also shows a cursor icon and a status bar with 'OVR'.

Getting Started With the MATLAB Pool

The MATLAB Pool



Connecting to HPC 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

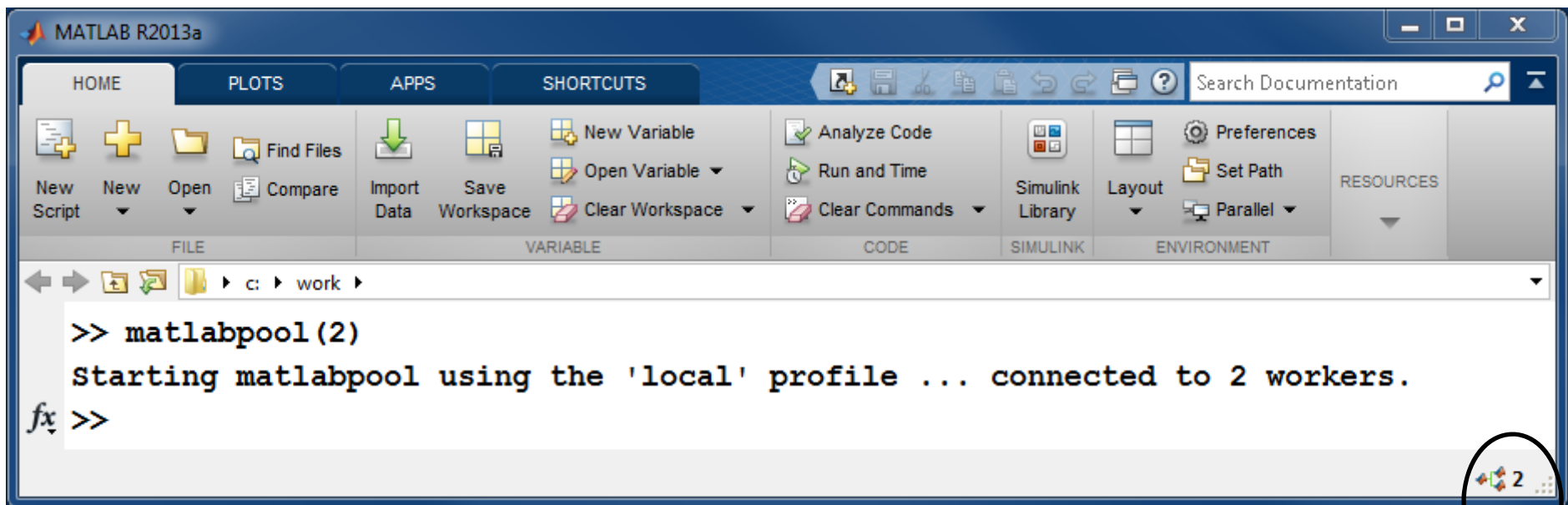
To be updated on
the Wiki

Starting a MATLAB Pool...

Bring up the Windows Task Manager or Linux *top*

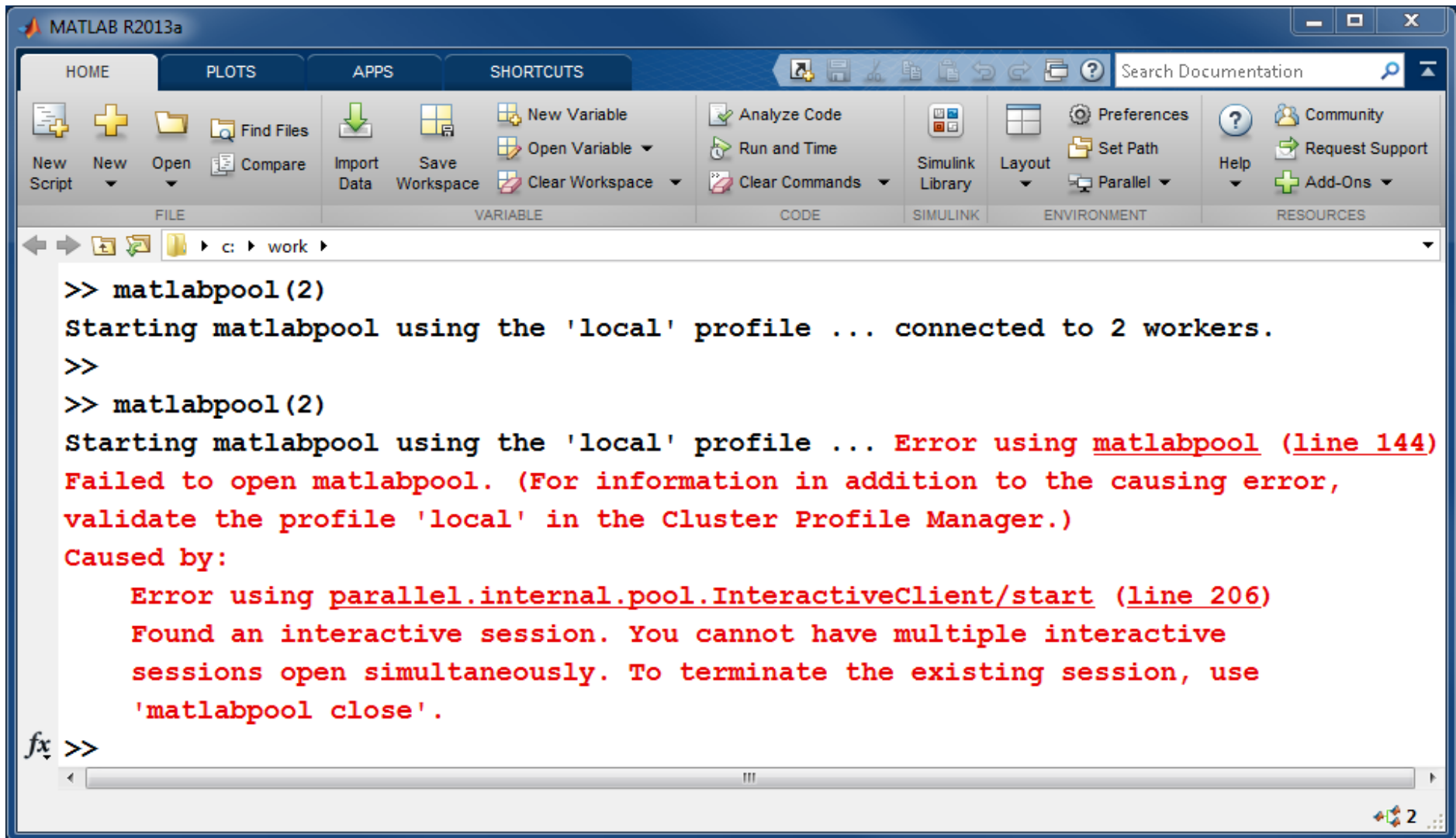
Start MATLAB

Open a MATLAB pool with two workers using the local profile



Maximum of 12 local workers

One MATLAB Pool at a Time

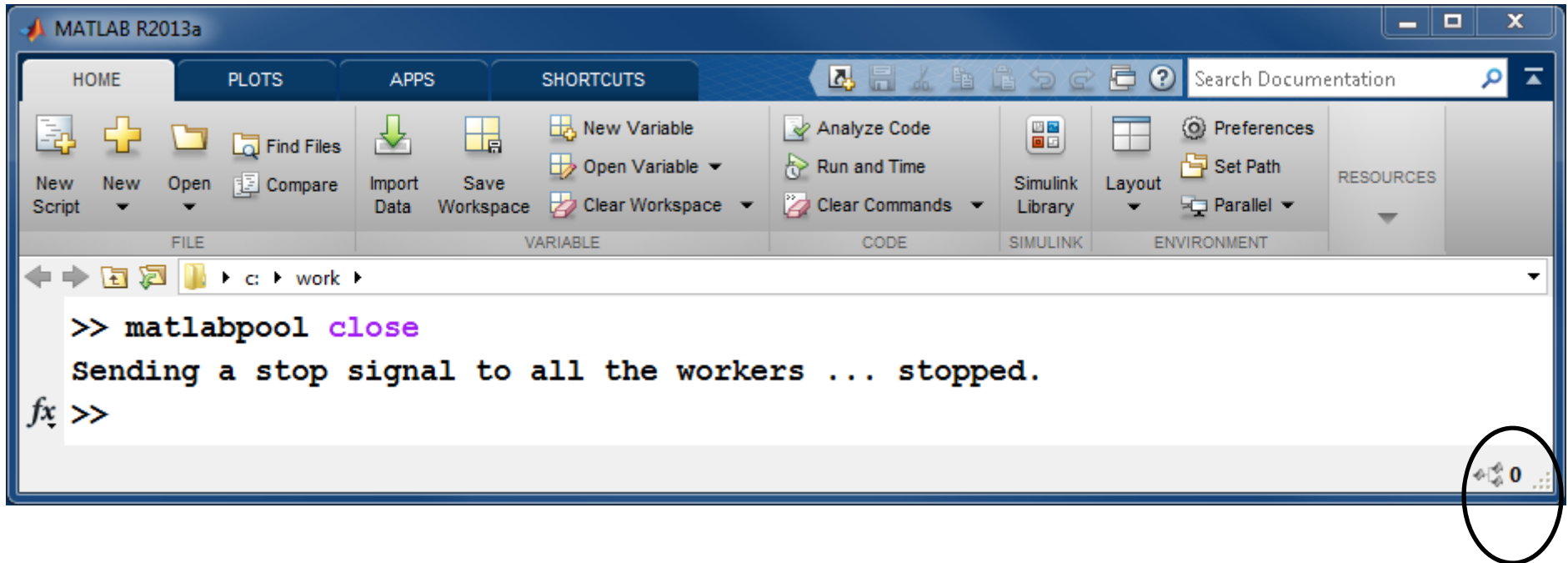


The screenshot shows the MATLAB R2013a interface. The Command Window displays the following text:

```
>> 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



The screenshot shows the MATLAB R2013a interface. The Command Window displays the following text:

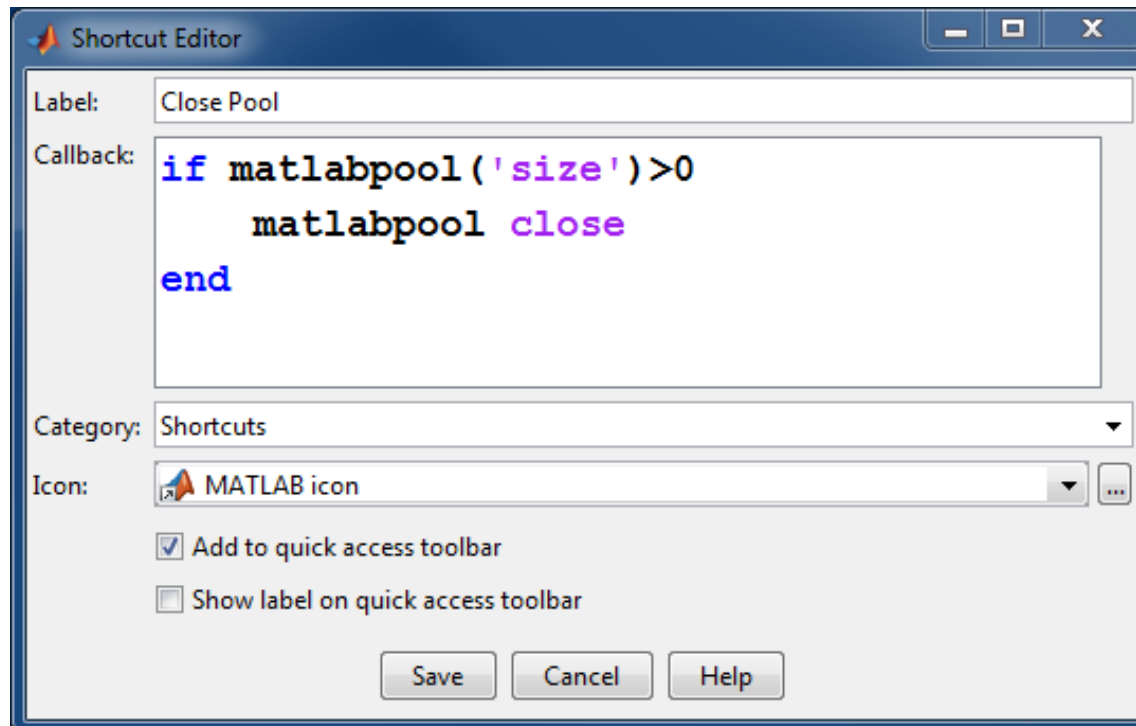
```
>> matlabpool close
Sending a stop signal to all the workers ... stopped.
fx >>
```

In the bottom right corner of the Command Window, a small icon representing the number of workers is circled in black. The icon shows a cluster of four worker nodes and the number '0', indicating that the MATLAB pool has been successfully stopped.

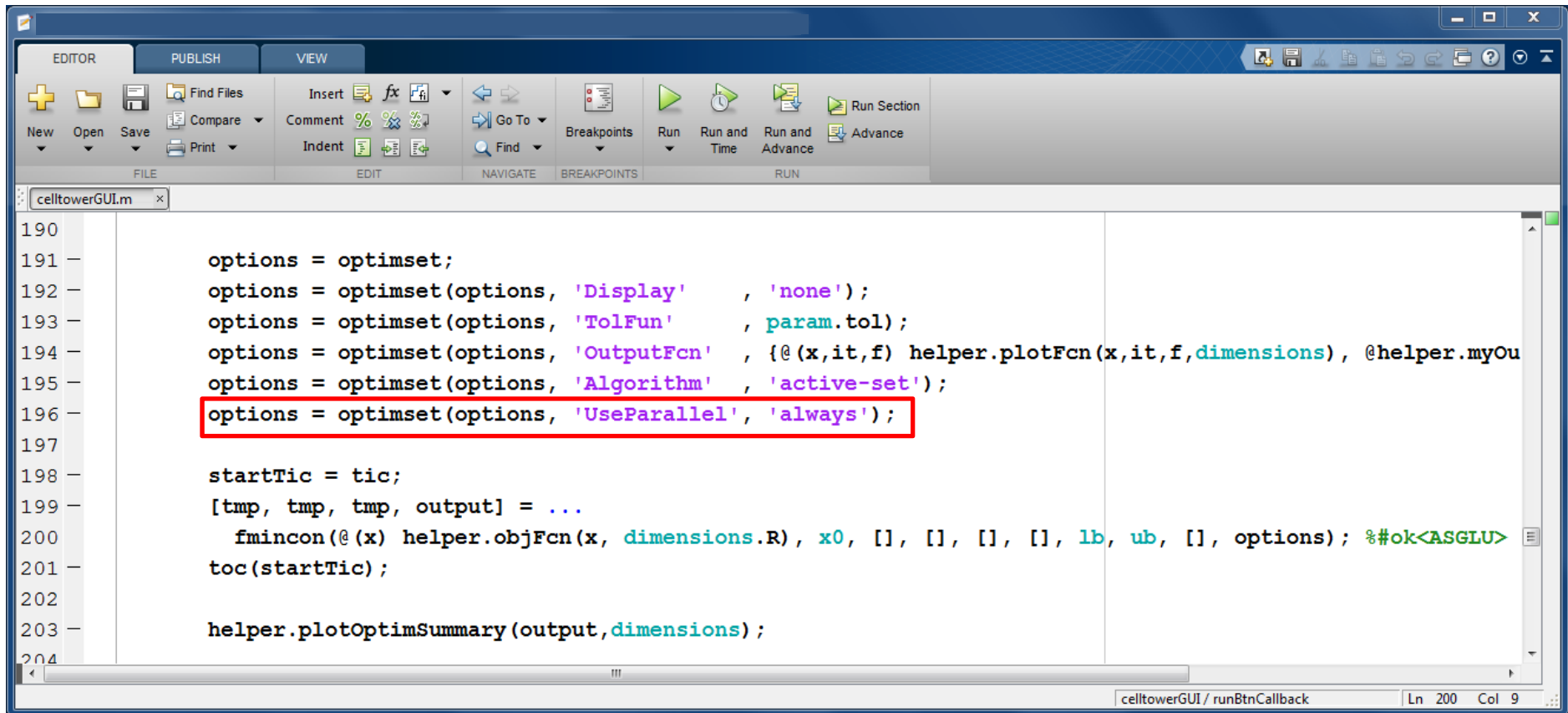
Add Shortcut for Starting the MATLAB Pool



Add Shortcut for Stopping the MATLAB Pool



Toolbox Support for Parallel Computing



```

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.myOu
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

```

celltowerGUI / runBtnCallback Ln 200 Col 9

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

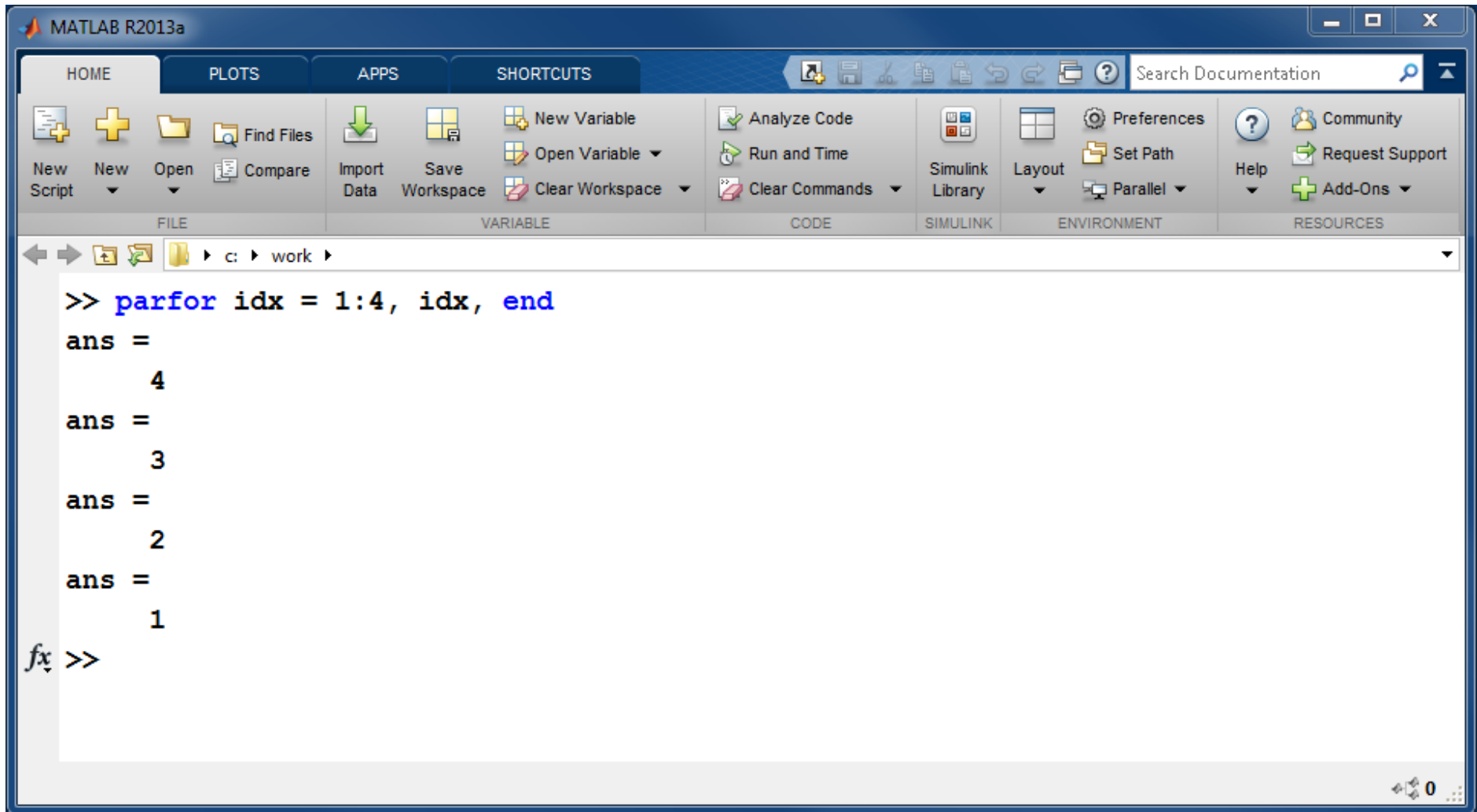
<http://www.mathworks.com/products/parallel-computing/builtin-parallel-support.html>

`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?

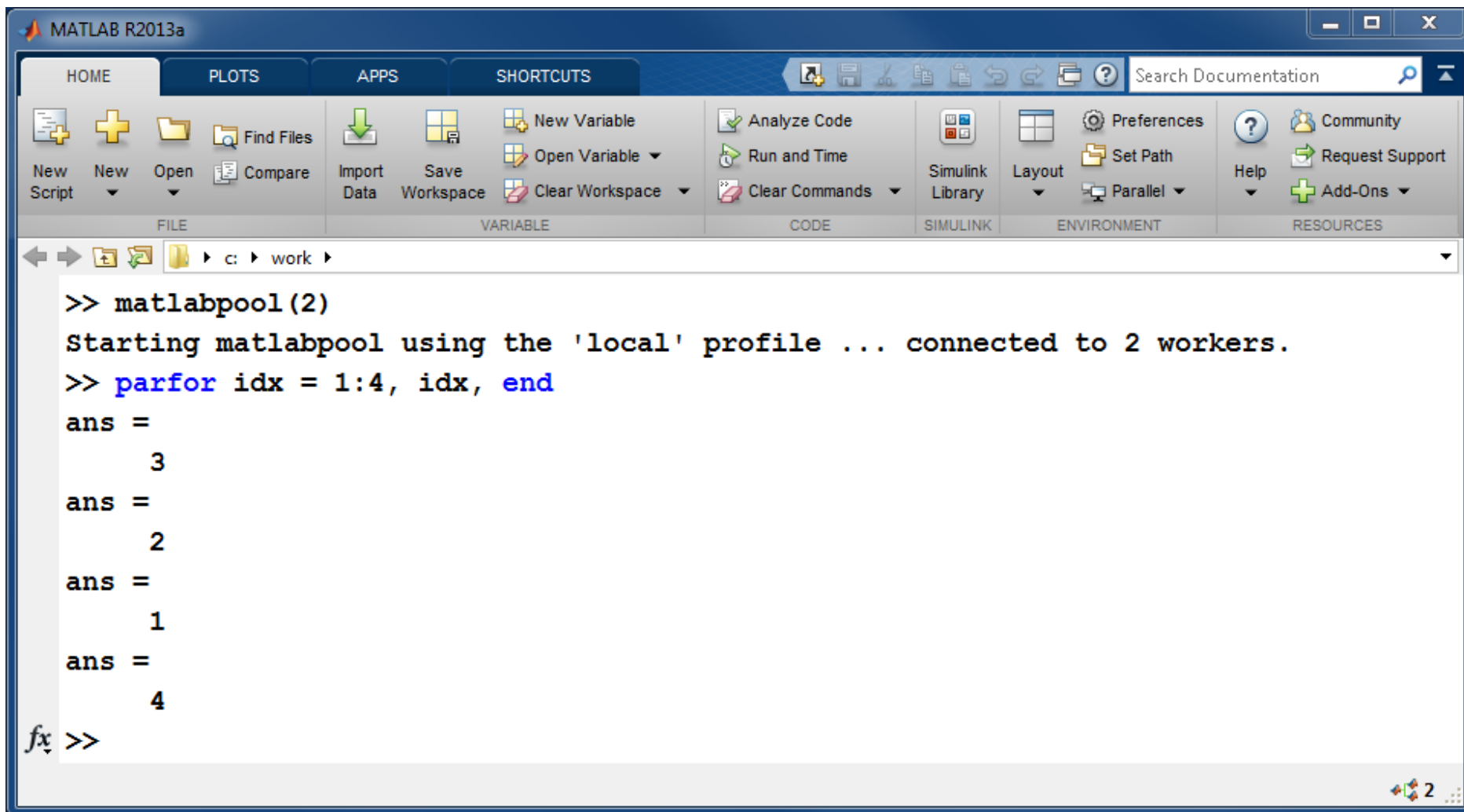


The image shows the MATLAB R2013a interface. The Command Window displays the following code and output:

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

The output shows the results of the parallel loop in descending order (4, 3, 2, 1), demonstrating that the order of execution is not necessarily the order of completion.

What If a MATLAB Pool Is Running?



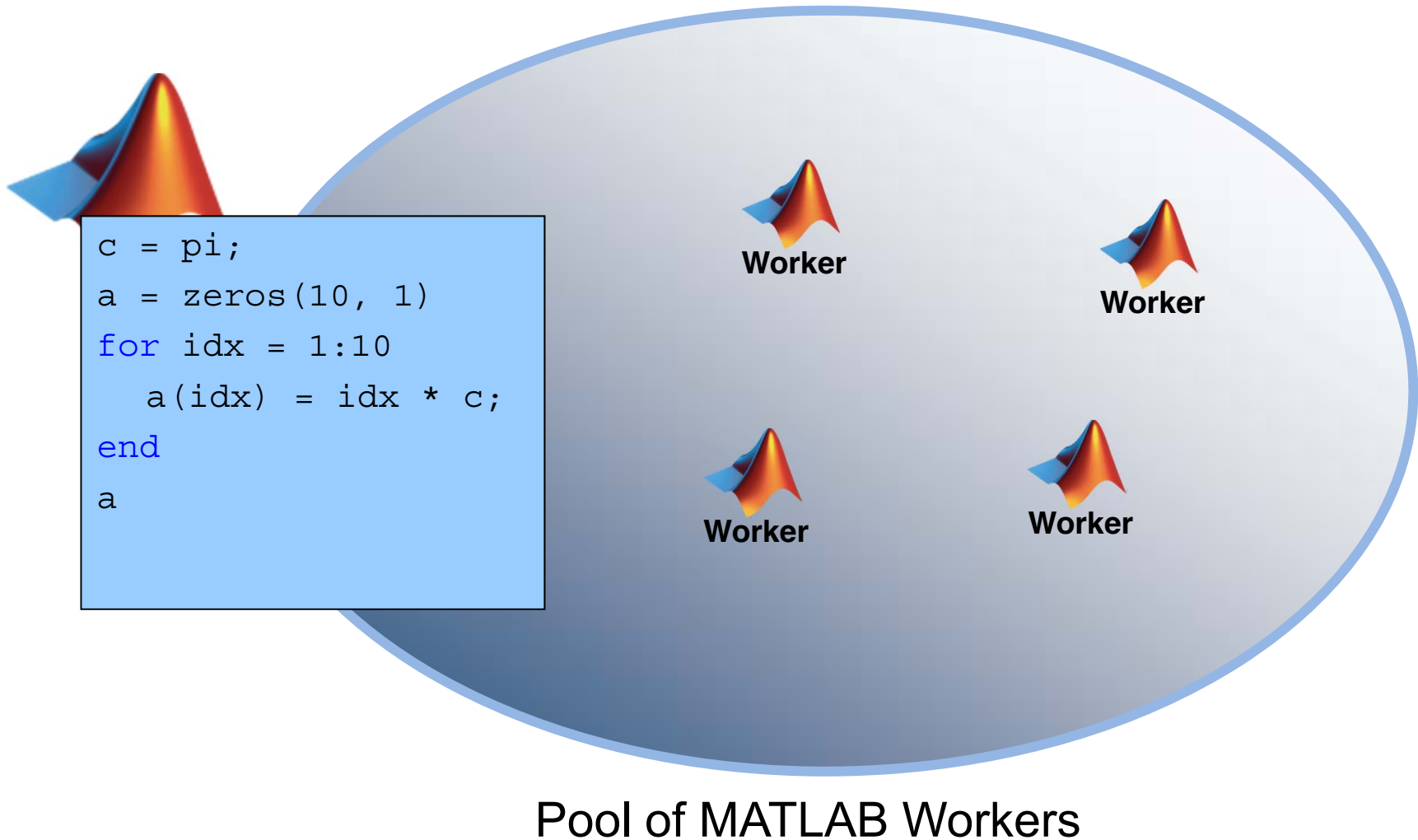
The image shows the MATLAB R2013a interface. The Command Window displays the following text:

```

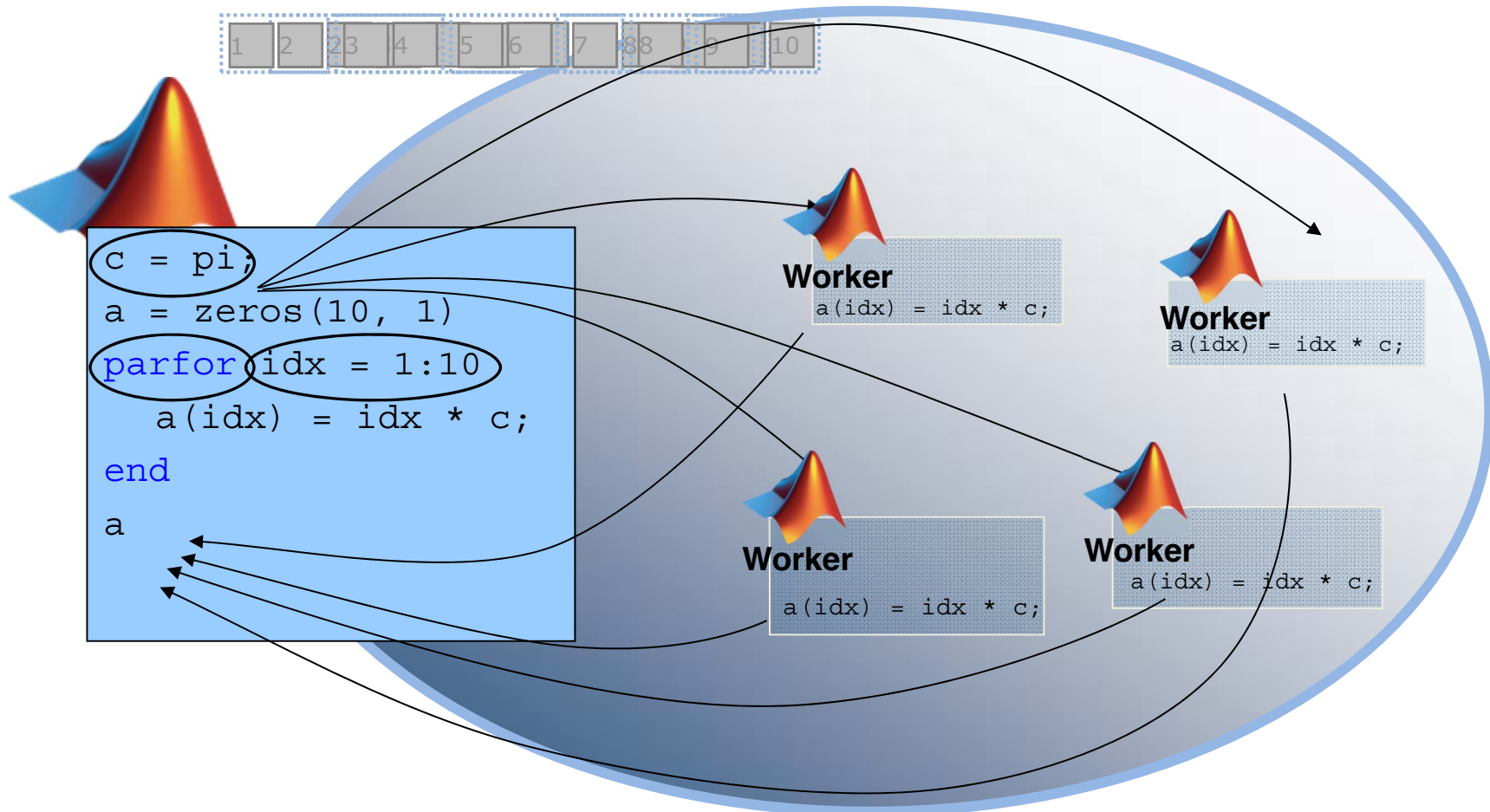
>> matlabpool(2)
Starting matlabpool using the 'local' profile ... connected to 2 workers.
>> parfor idx = 1:4, idx, end
ans =
    3
ans =
    2
ans =
    1
ans =
    4
fx >>
  
```

The interface includes a ribbon with tabs for HOME, PLOTS, APPS, and SHORTCUTS. The Command Window shows the execution of a MATLAB pool and a parallel for loop. The output of the loop is a column vector of the numbers 3, 2, 1, and 4. The status bar at the bottom right indicates 2 workers are active.

The Mechanics of `parfor` Blocks



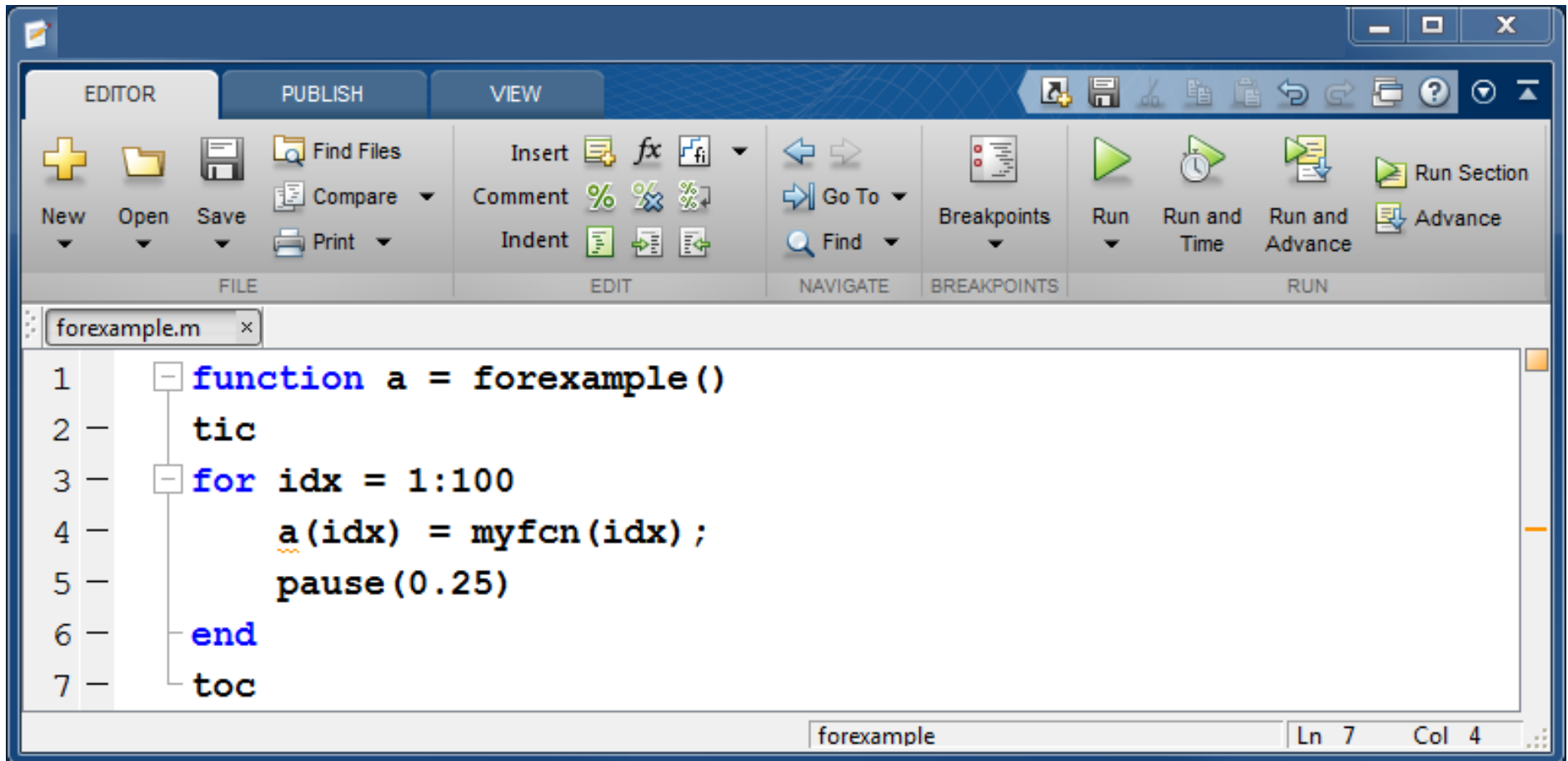
The Mechanics of `parfor` Blocks



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 IDE interface. The editor window displays the following code for 'forexample.m':

```

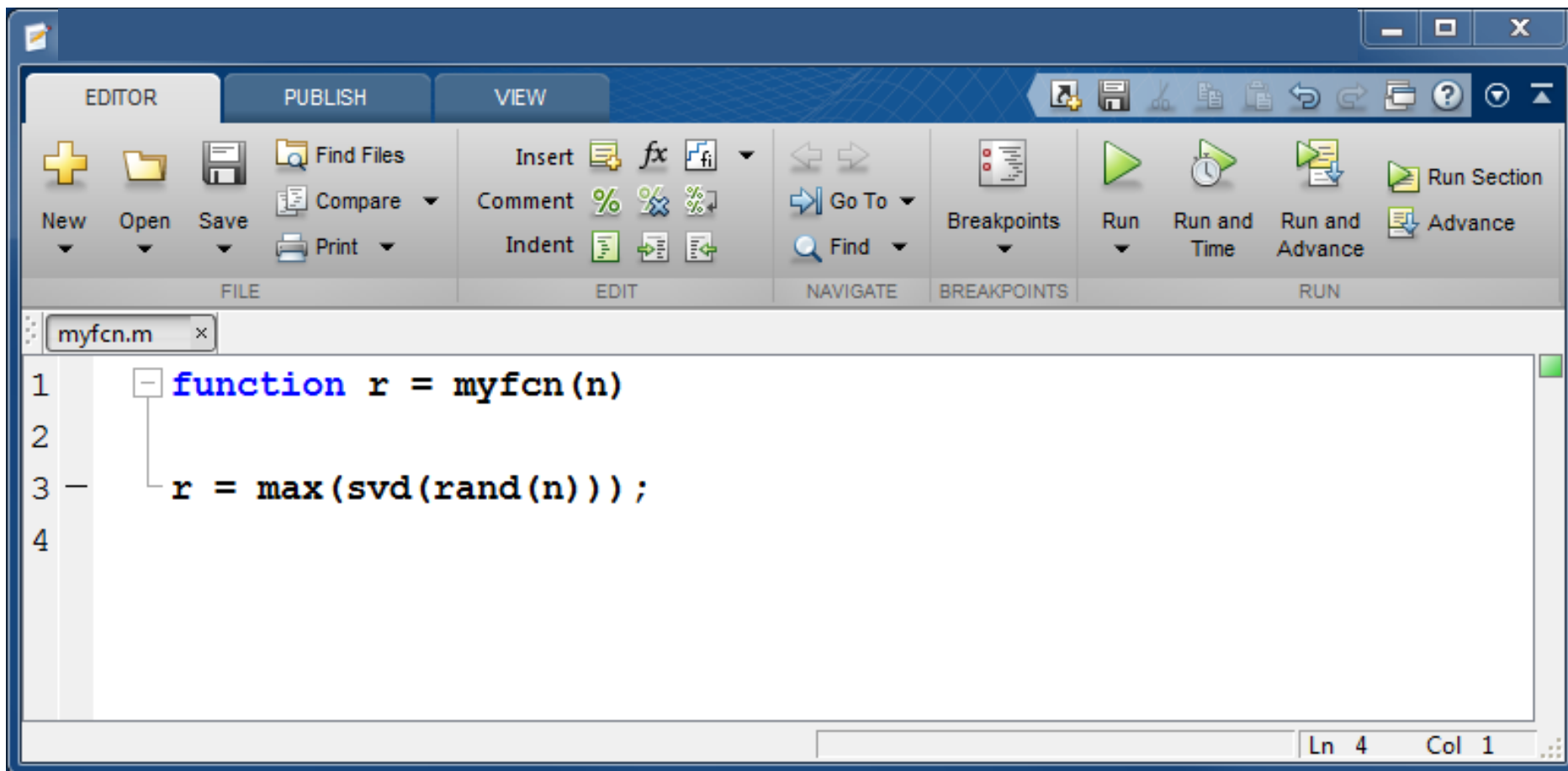
1  function a = forexample()
2  tic
3  for idx = 1:100
4      a(idx) = myfcn(idx);
5      pause(0.25)
6  end
7  toc
  
```

The status bar at the bottom indicates the current position is at line 7, column 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 interface with the file `myfcn.m` open. The code is as follows:

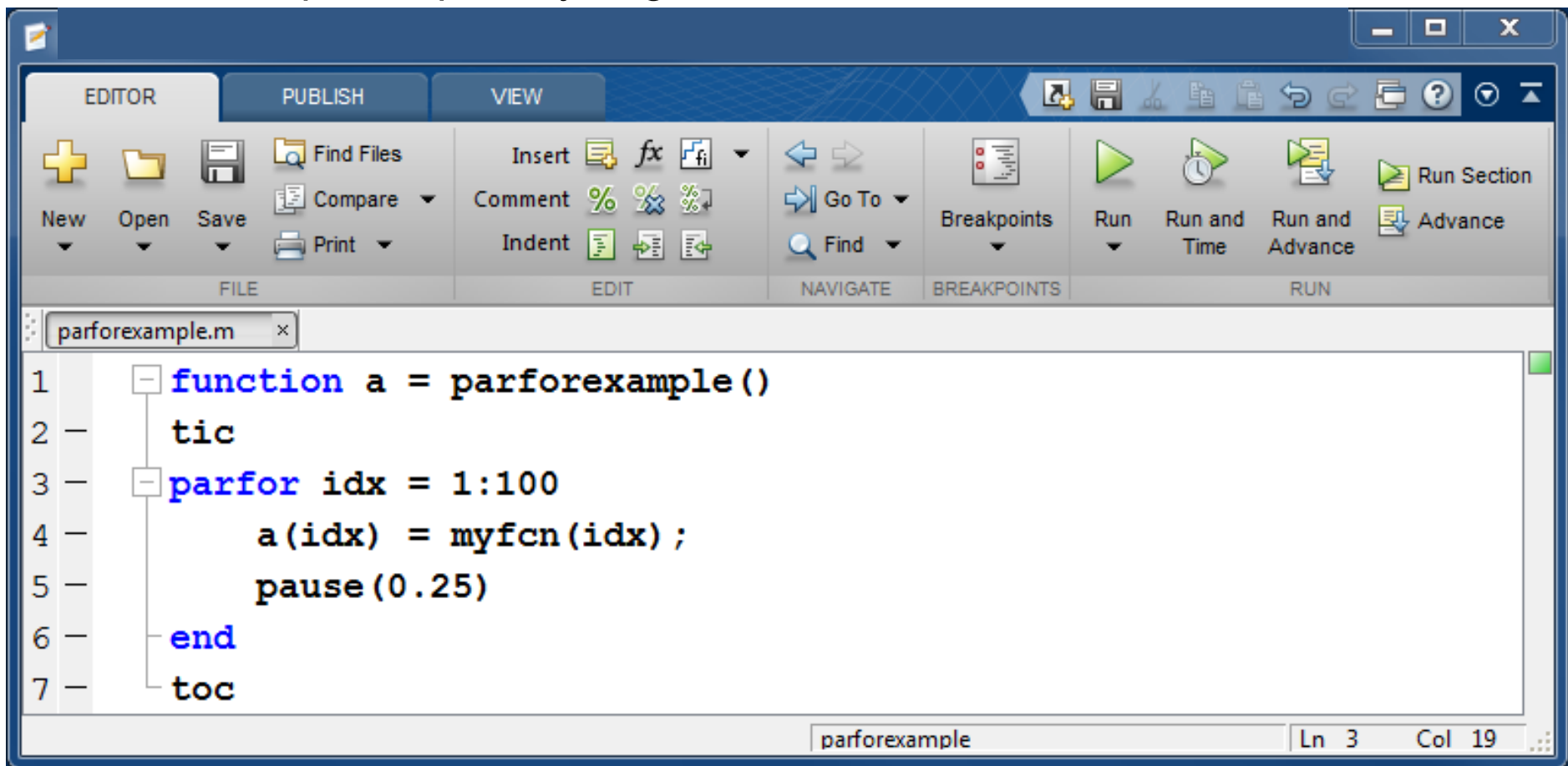
```
1 function r = myfcn(n)
2
3     r = max(svd(rand(n)));
4
```

The status bar at the bottom right indicates the cursor is at Line 4, Column 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 IDE interface. The editor window displays the following code for `parforexample.m`:

```

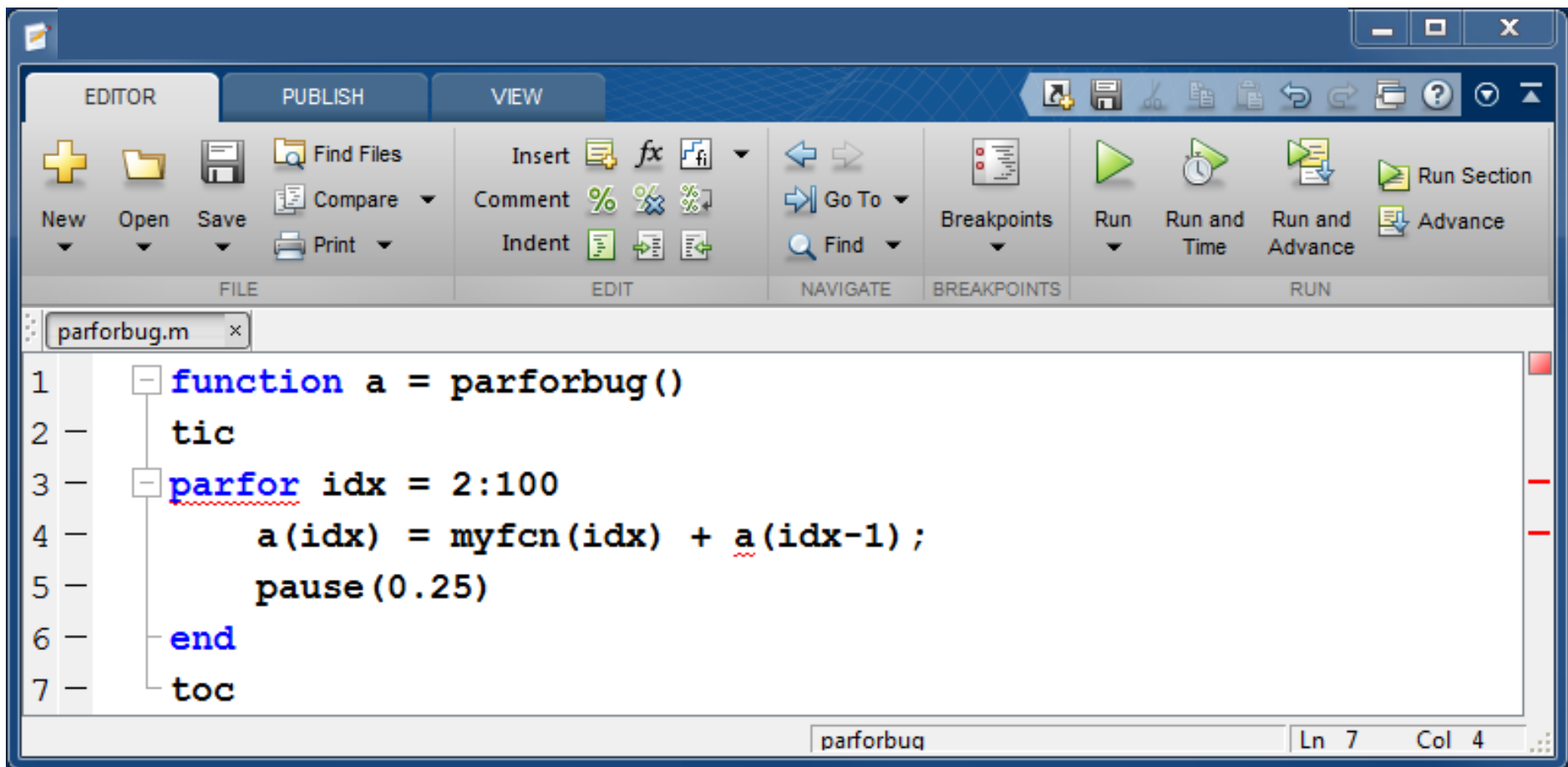
1  function a = parforexample()
2  tic
3  parfor idx = 1:100
4      a(idx) = myfcn(idx);
5      pause(0.25)
6  end
7  toc
  
```

The status bar at the bottom indicates the current position is at line 3, column 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 IDE editor window for a file named `parforbug.m`. The code is as follows:

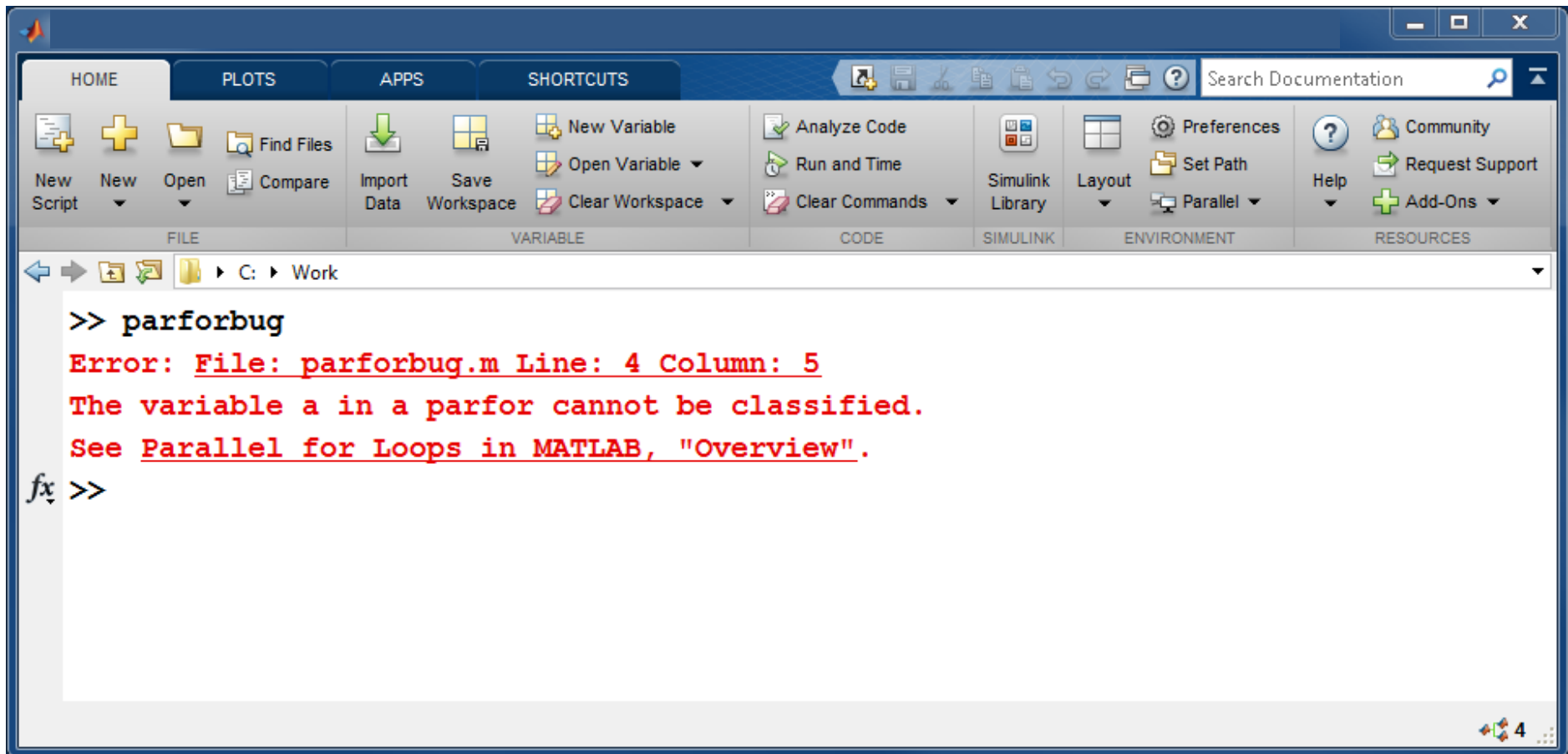
```

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 `parfor` loop is highlighted with a red dashed underline. The status bar at the bottom indicates the current position is at line 7, column 4.

>> parforbug

Example: Break It (2)



The screenshot shows the MATLAB Command Window interface. The Command Window contains the following text:

```
>> parforbug
Error: File: parforbug.m Line: 4 Column: 5
The variable a in a parfor cannot be classified.
See Parallel for Loops in MATLAB, "Overview".
fx >>
```

The error message indicates that the variable 'a' in a parfor loop cannot be classified. The error occurs at line 4, column 5 of the file parforbug.m. A link is provided to the MATLAB documentation for parallel for loops.

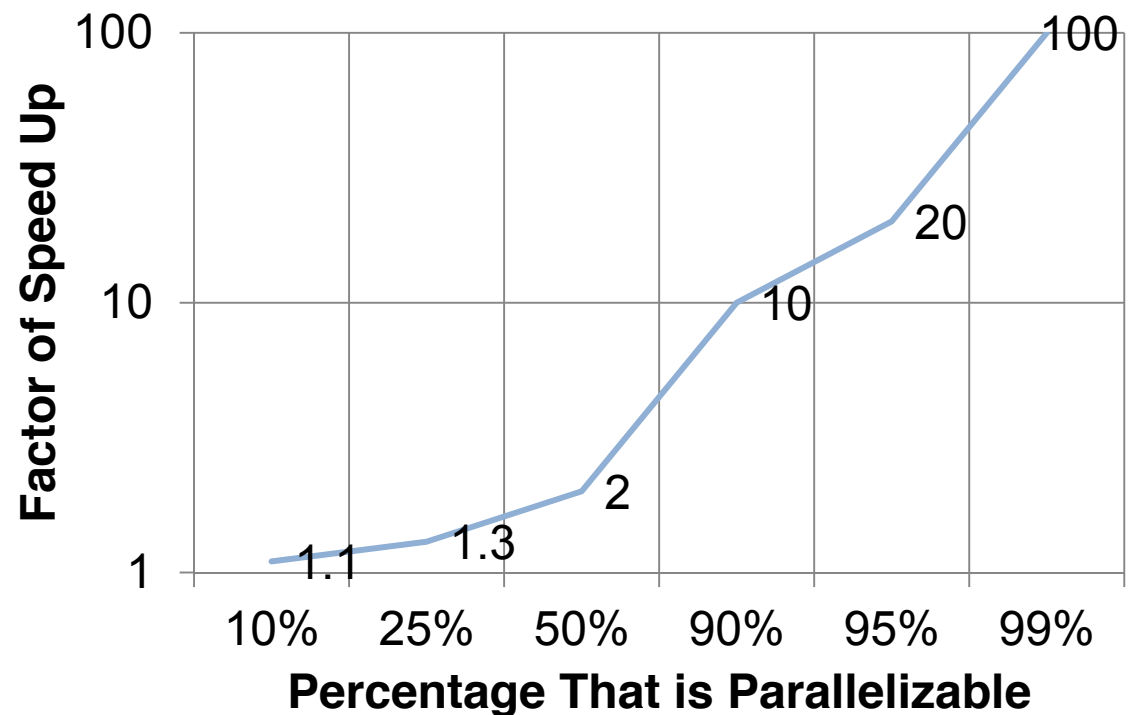
The variable a cannot be properly classified

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

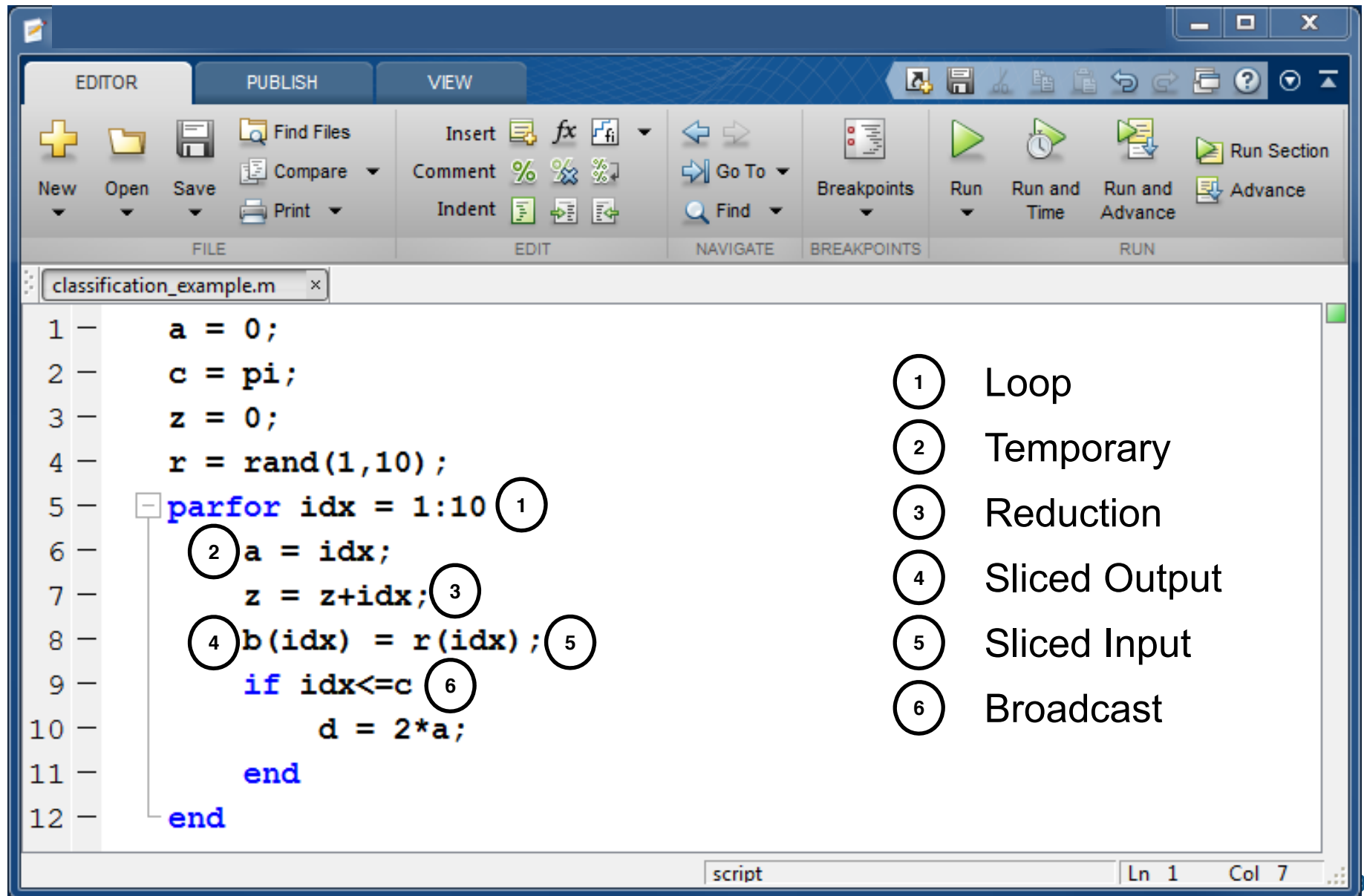
- Should I pre-allocate a matrix?
 - There is no significant speedup, if any, in pre-allocating the matrix
- Should I pre-assign large matrices before the `parfor`?
 - Yes, if they're going to be referenced after the `for` loop (to be explained why later)
 - Otherwise, do all the large creation on the workers
 - So if I have a `for` loop with 100 iterations and 10 workers, are each of the matrices create 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 interface with a script named 'classification_example.m'. The script contains the following code:

```

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

```

Annotations in the code are circled numbers 1 through 6, corresponding to the following legend:

- ① Loop
- ② Temporary
- ③ Reduction
- ④ Sliced Output
- ⑤ Sliced Input
- ⑥ Broadcast

The status bar at the bottom indicates 'script' and 'Ln 1 Col 7'.

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

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

```

EDITOR PUBLISH VIEW
+ New Open Save Find Files Compare Print FILE
Insert Comment Indent EDIT
what_is_it_parfor.m x
1 a = ones(1,10);
2 e = 0;
3 f = 5;
4 g = 0;
5 h = 10;
6 s.field = rand(1,10);
7 parfor idx = 1:10
8     b = 2*a;
9     c = a(idx);
10    d(idx) = idx;
11    e = e+idx;
12    f = idx;
13    g = g+2;
14    h = 20;
15    j = s.field(idx);
16 end
  
```

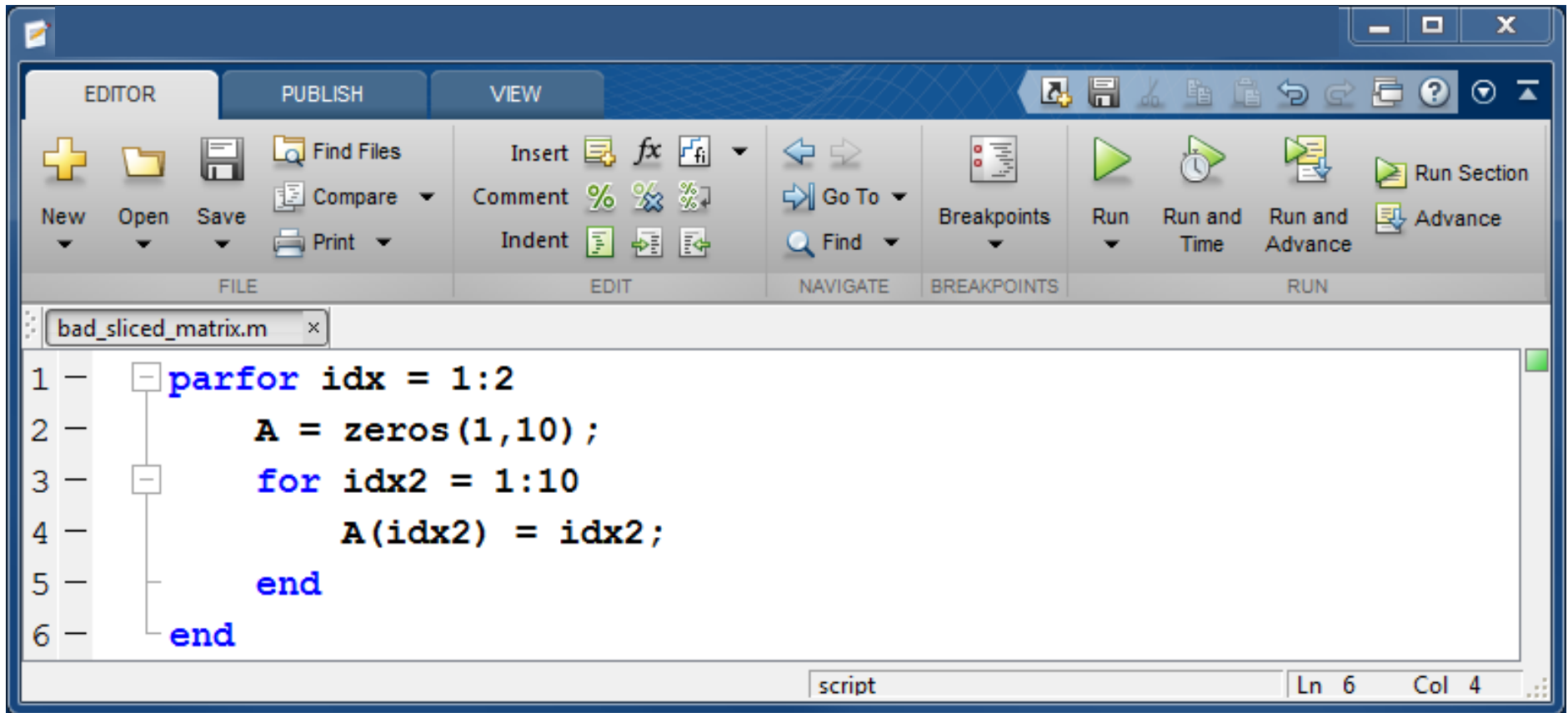
>> what_is_it_parfor

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



```

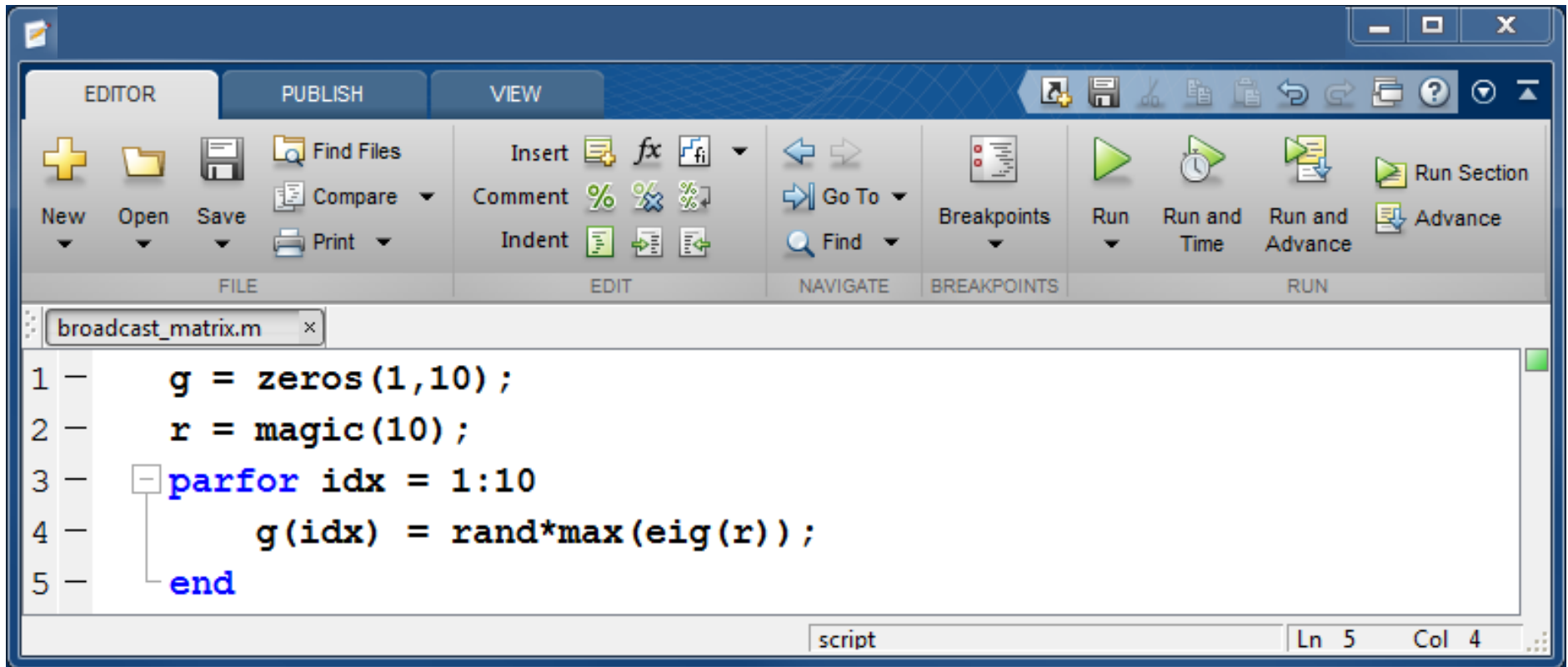
1 - parfor idx = 1:2
2 -     A = zeros(1,10);
3 -     for idx2 = 1:10
4 -         A(idx2) = idx2;
5 -     end
6 - end
  
```

script Ln 6 Col 4

What is the value of A?

```
>> bad_sliced_matrix
```

Implications of Broadcast Variables



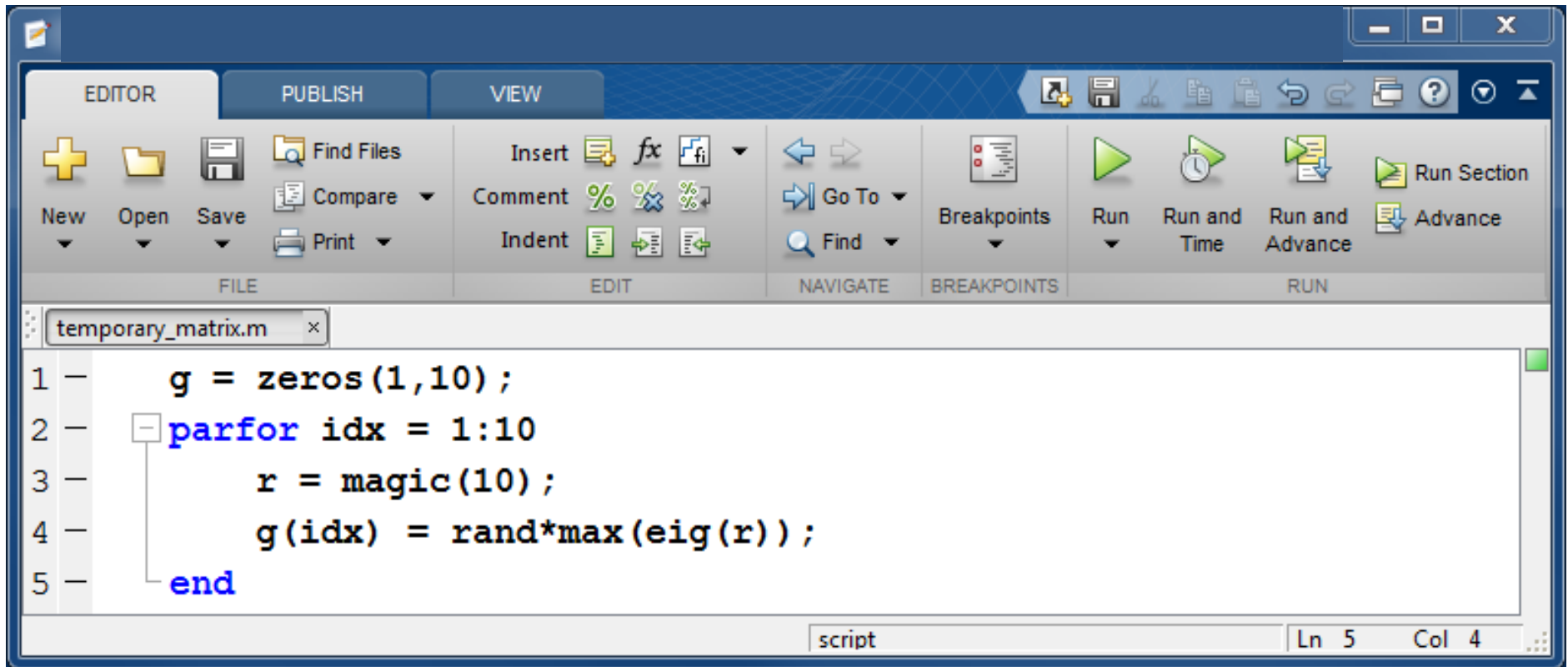
```

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

The entire data set r is broadcast to each worker

```
>> broadcast_matrix
```

Implications of Broadcast Variables



```

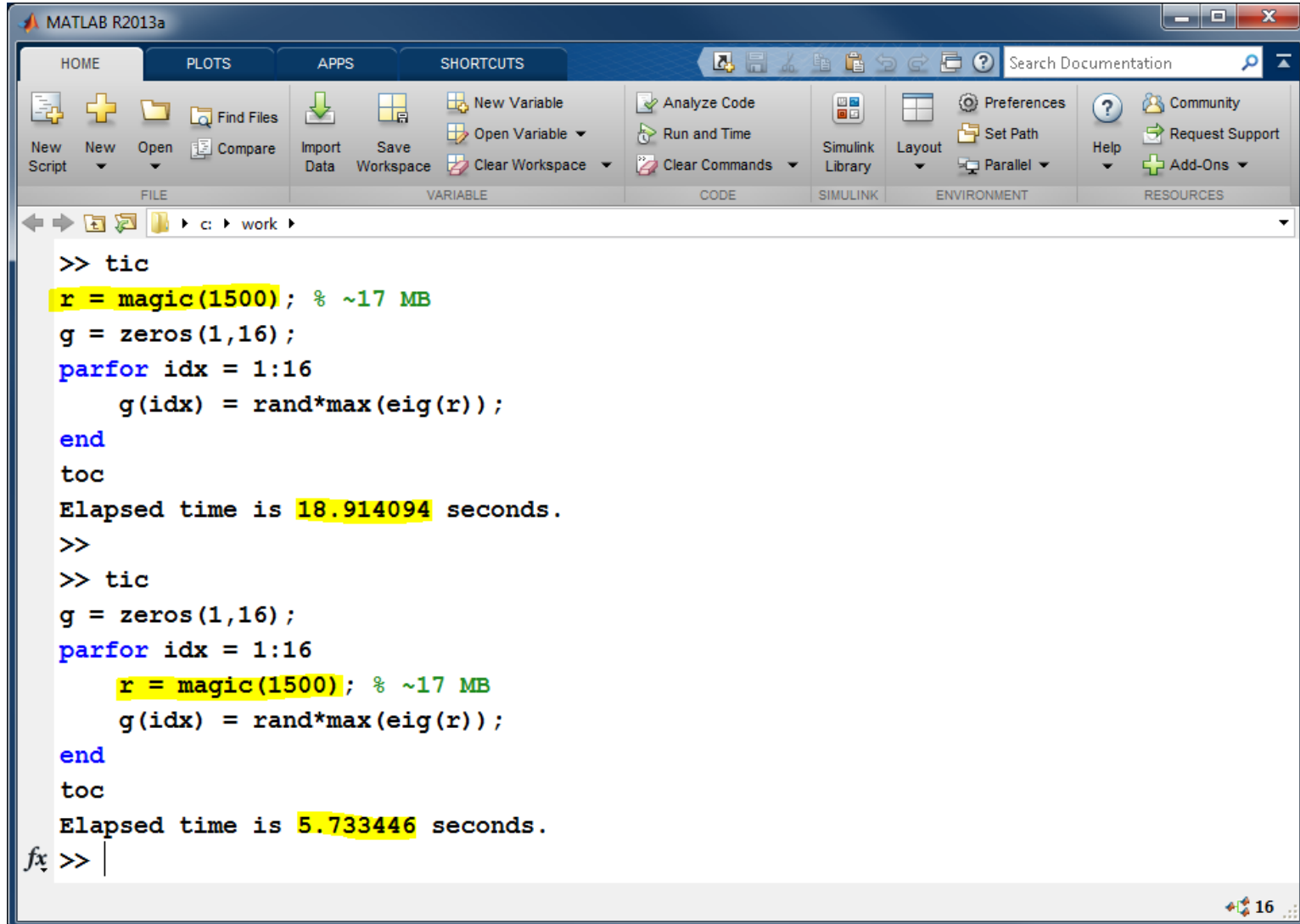
1 -   g = zeros(1,10);
2 -   parfor idx = 1:10
3 -       r = magic(10);
4 -       g(idx) = rand*max(eig(r));
5 -   end
  
```

script | Ln 5 Col 4

Could you create r on the workers instead?

```
>> temporary_matrix
```

Implications of Broadcast Variables



The image shows the MATLAB R2013a interface with a script being executed. The script contains two blocks of code, each starting with a timer (tic) and ending with a stop timer (toc). The first block creates a 1500x1500 magic matrix 'r' (commented as ~17 MB) and a 1x16 vector 'g'. A parfor loop iterates over 'idx' from 1 to 16, where 'g(idx)' is assigned a value from 'rand*max(eig(r))'. The elapsed time for this block is 18.914094 seconds. The second block is identical but places the creation of 'r' inside the parfor loop. The elapsed time for this block is 5.733446 seconds. The difference in execution time is due to the broadcast of the large matrix 'r' to each of the 16 parallel workers in the first case.

```

>> 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 Reductions 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

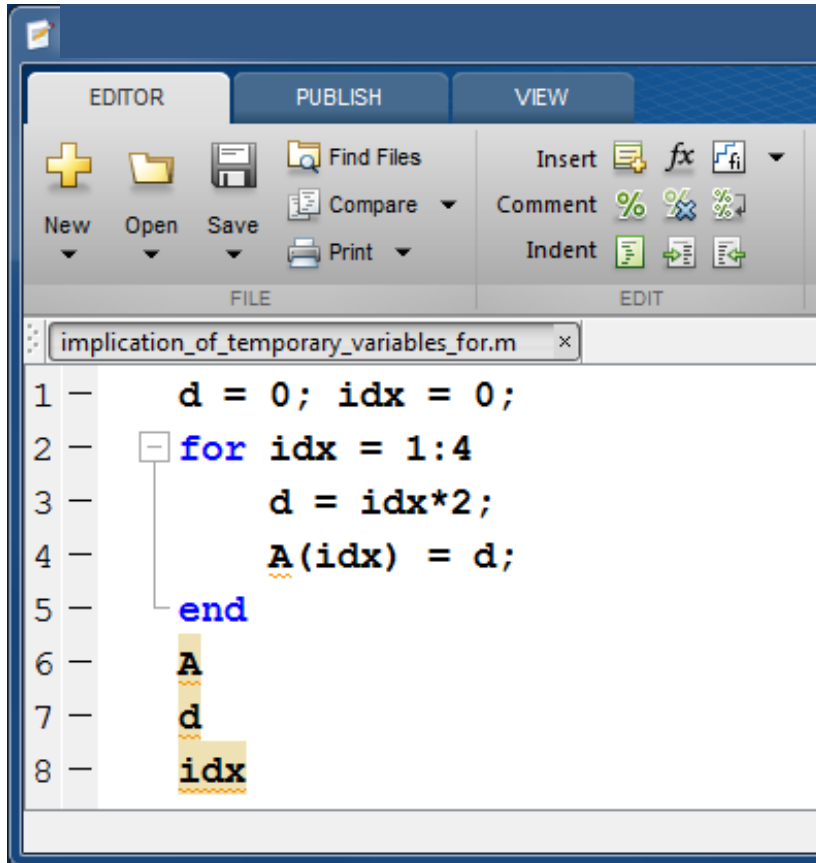
```
>> x = 0;
parfor idx = 1:10
    x = x+idx;
end
x
```

```
>> 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
```

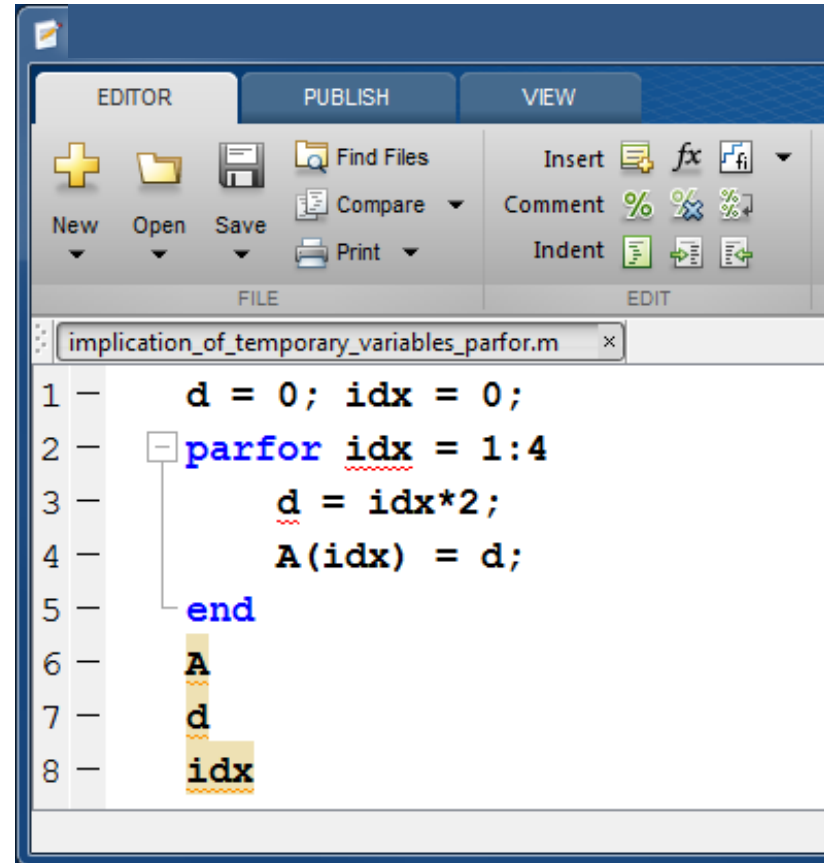
Error: Different reduction functions are used for the same variable x3.
See [Parallel for Loops in MATLAB, "Basic Rules for Reduction Variables"](#).

Implications of Temporary Variables



```

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
  
```

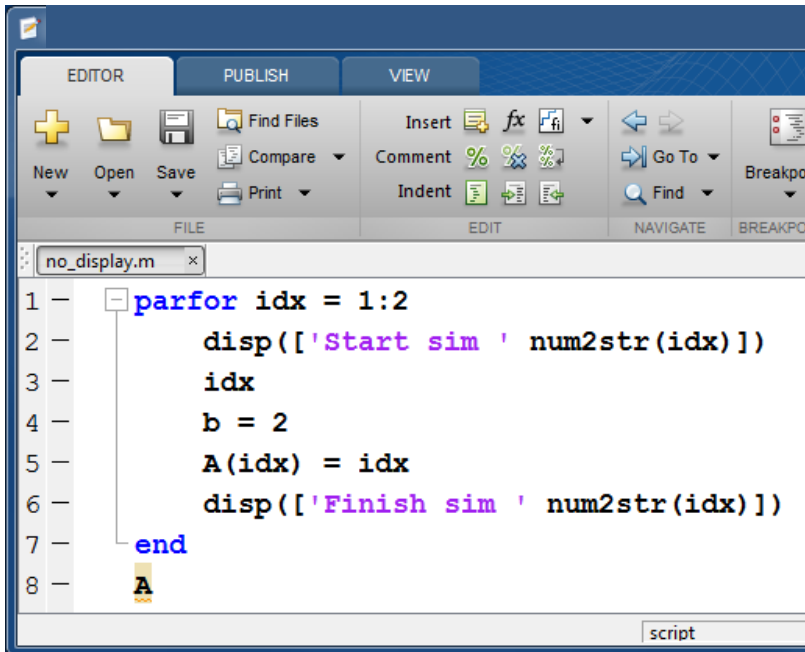


```

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
  
```

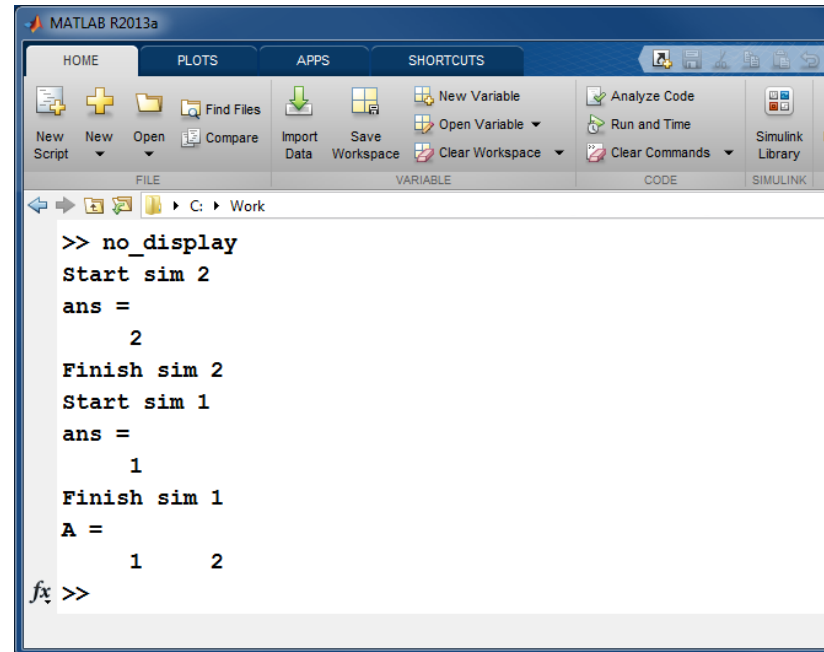
What is the value of A? d? idx?

Variable Assignments Are Not Displayed When Running a `parfor`



```

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
HOME PLOTS APPS SHORTCUTS
New Script New Open Find Files Compare Import Data Save Workspace Clear Workspace
FILE VARIABLE
C:\Work
>> 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
HOME PLOTS APPS SHORTCUTS
New Script New Open Find Files Compare Import Data Save Workspace Clear Workspace
FILE VARIABLE
C:\Work
>> 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 >>

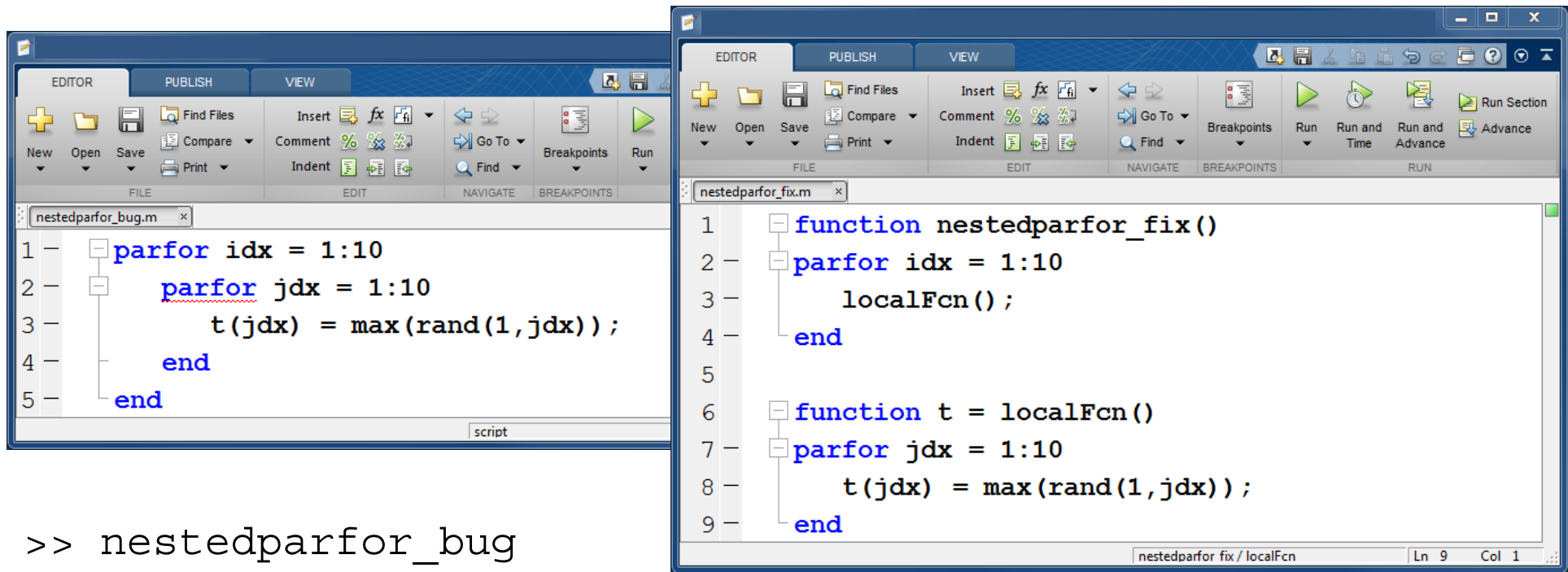
```

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



```

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

function nestedparfor_fix()
parfor idx = 1:10
    localFcn();
end

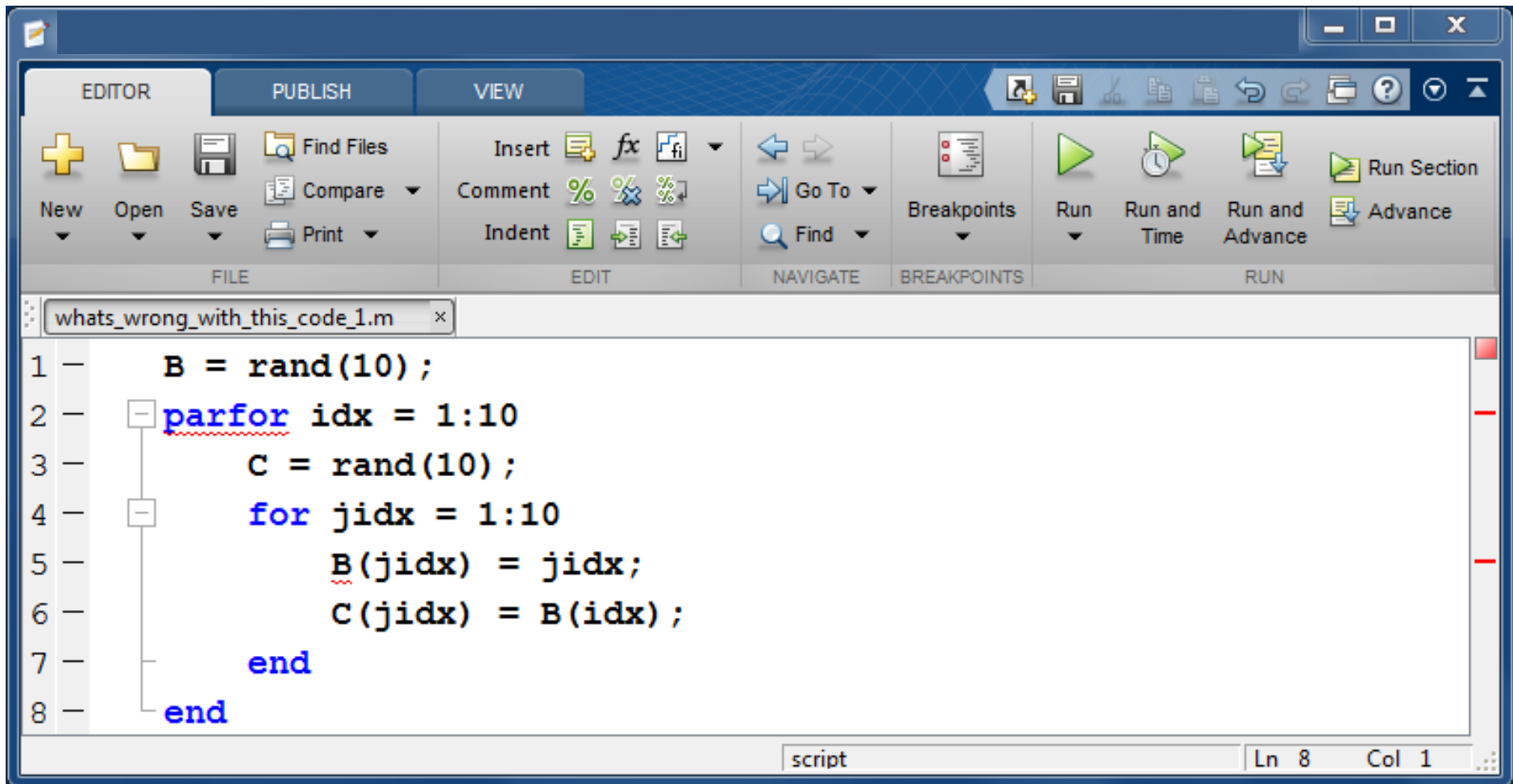
function t = localFcn()
parfor jdx = 1:10
    t(jdx) = max(rand(1,jdx));
end
  
```

>> nestedparfor_bug

>> nestedparfor_fix

What's Wrong With This Code?

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



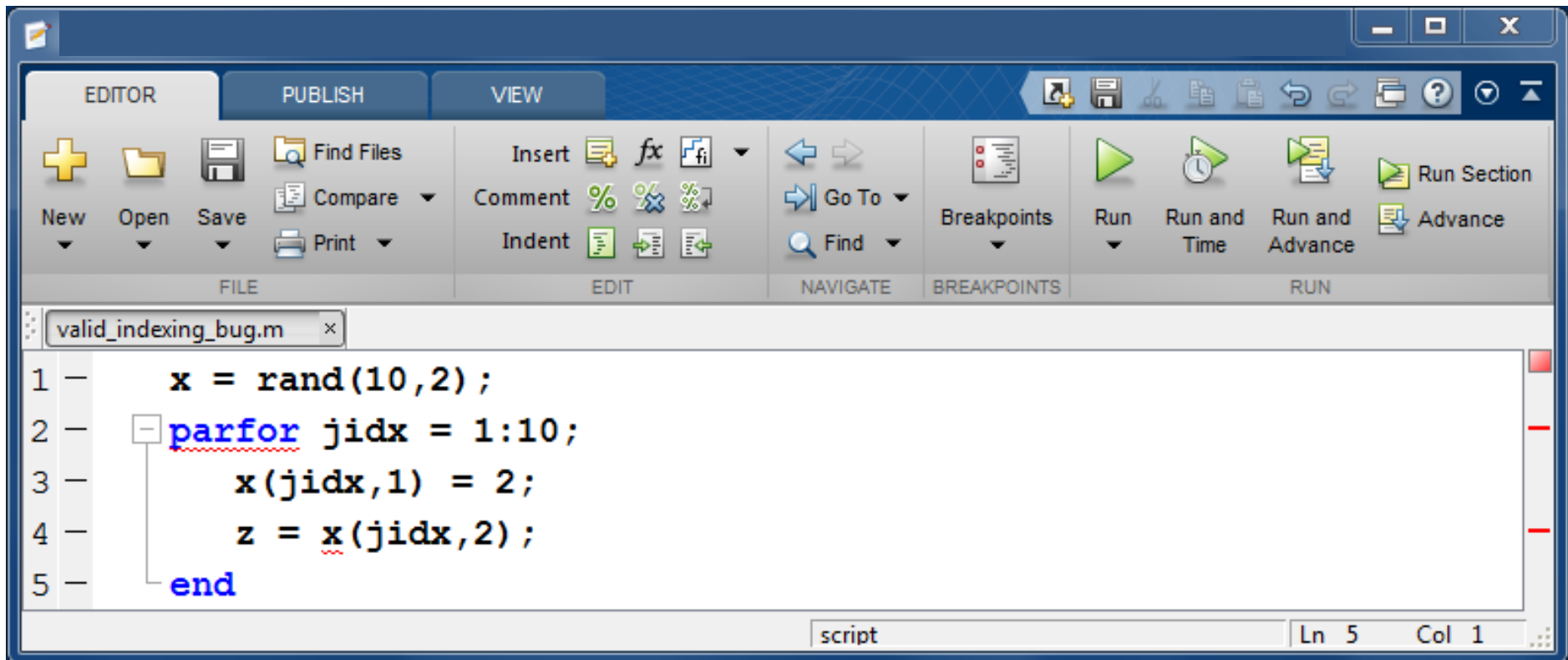
```

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 screenshot shows the MATLAB Editor interface. The code is displayed in a script window titled 'whats_wrong_with_this_code_1.m'. The code consists of a `parfor` loop over `idx` from 1 to 10. Inside the `parfor` loop, there is a `for` loop over `jidx` from 1 to 10. In the `for` loop, `B(jidx)` is assigned the value of `jidx`, and `C(jidx)` is assigned the value of `B(idx)`. The `for` loop ends with `end`, and the `parfor` loop ends with `end`. The status bar at the bottom indicates 'script' and 'Ln 8 Col 1'.

```
>> whats_wrong_with_this_code
```

parfor issue: Indexing With Different Expressions



```

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

The screenshot shows the MATLAB Editor interface. The title bar indicates the file is 'valid_indexing_bug.m'. The script content is as follows:

```

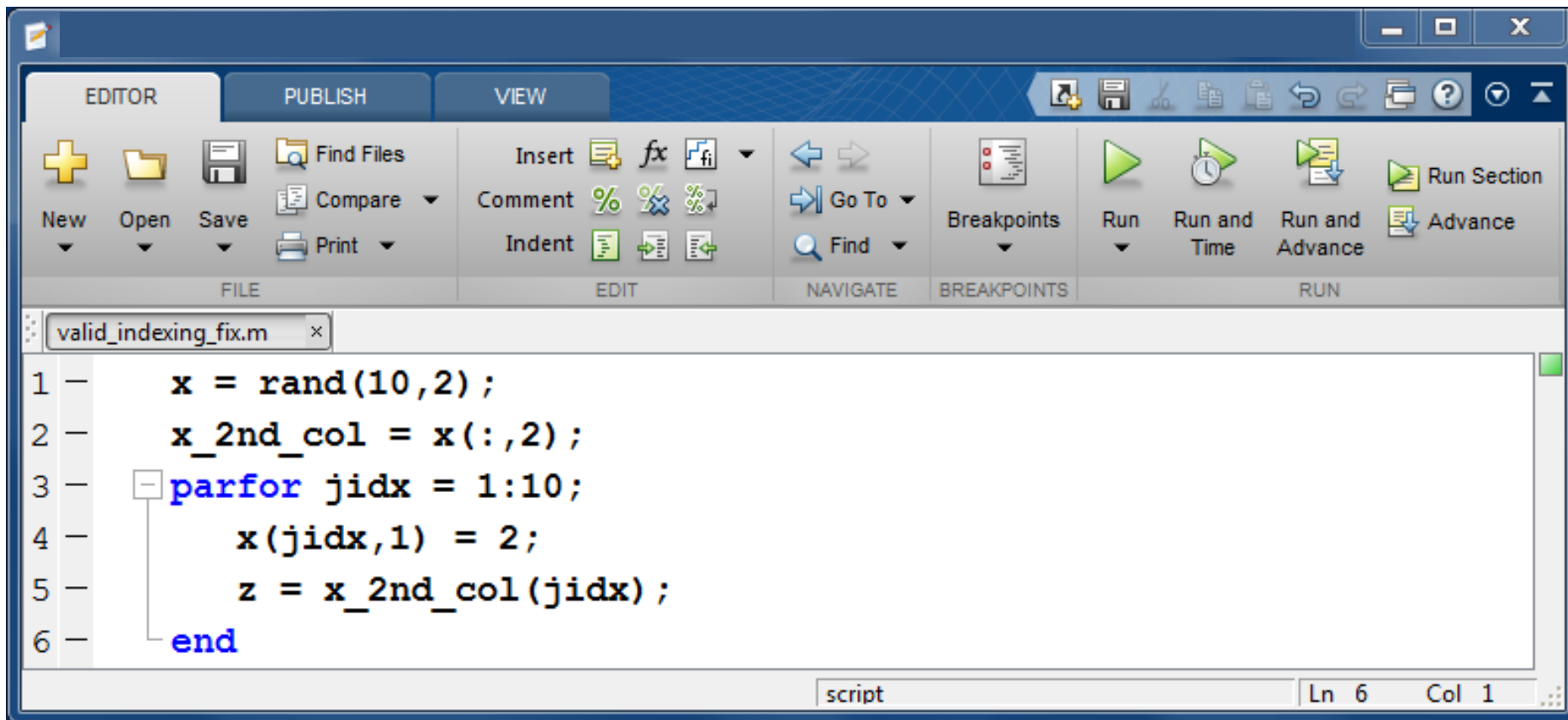
1 -   x = rand(10,2);
2 -   parfor jidx = 1:10;
3 -       x(jidx,1) = 2;
4 -       z = x(jidx,2);
5 -   end
  
```

The status bar at the bottom right shows 'script' and 'Ln 5 Col 1'.

How can we avoid indexing into x two different ways?

```
>> valid_indexing_bug
```

parfor issue: Solution



```

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

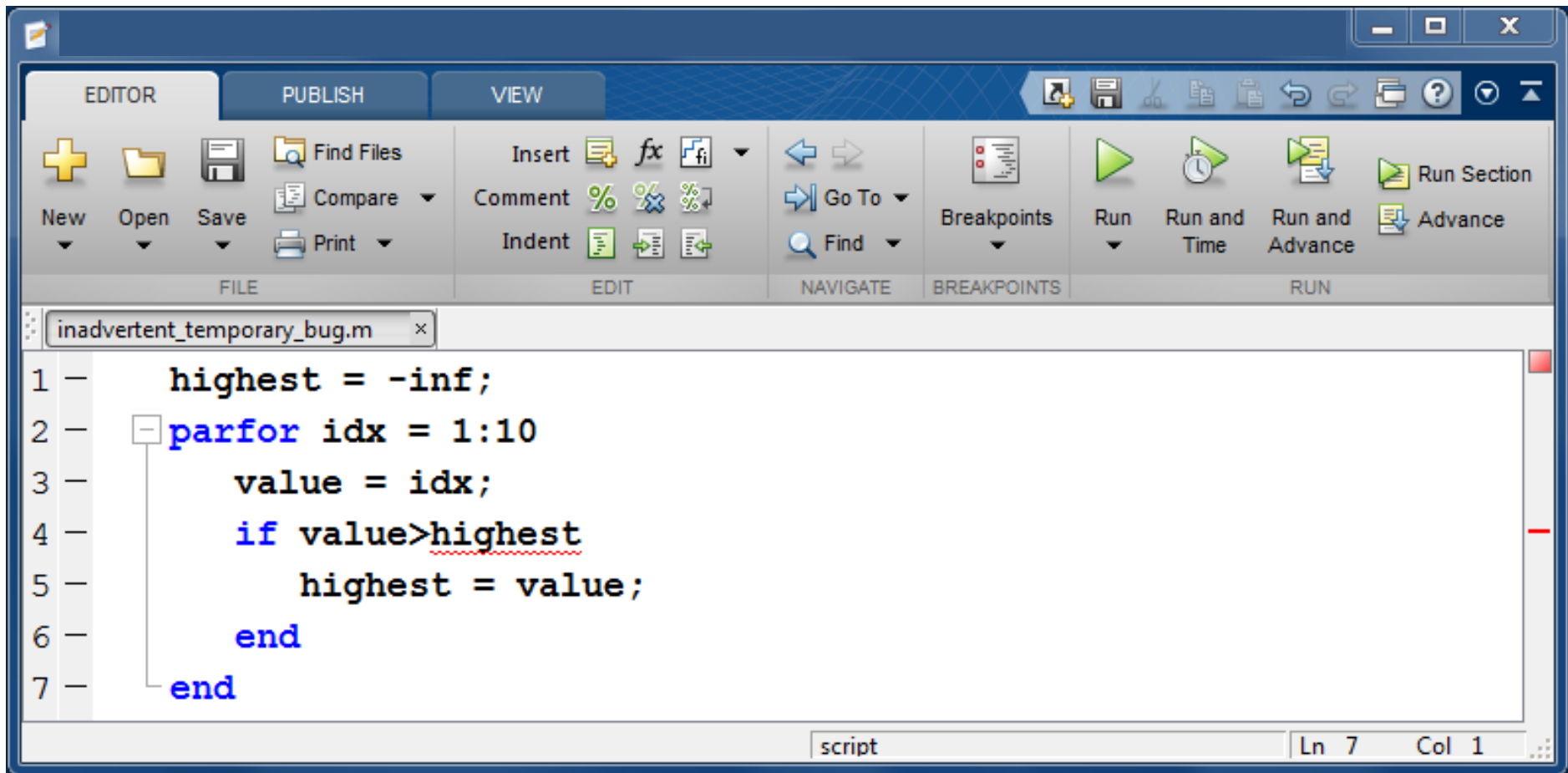
```

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



```

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

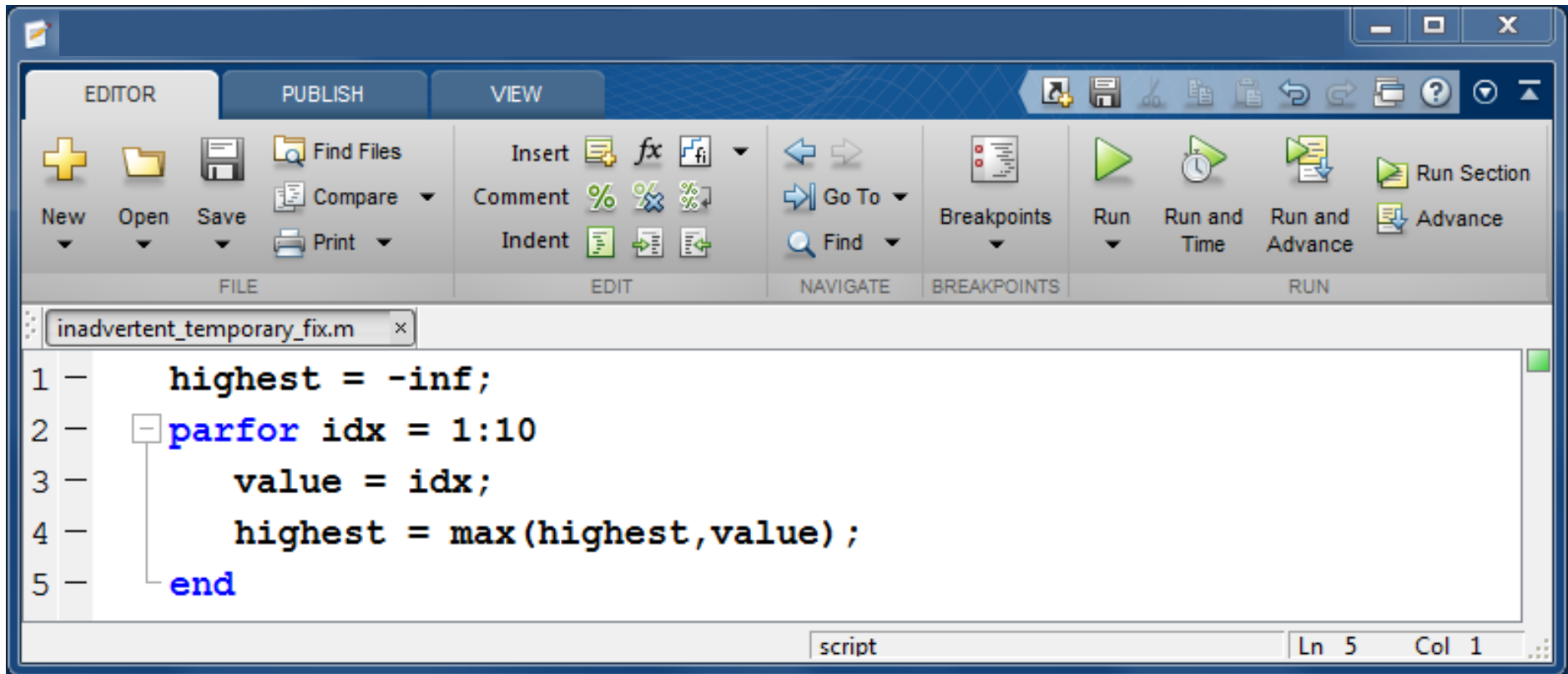
The screenshot shows the MATLAB Editor interface with a file named 'inadvertent_temporary_bug.m'. The code contains a `parfor` loop. A red squiggly line under the variable `highest` in the `if` statement indicates a code analyzer warning. The status bar at the bottom shows 'script' and 'Ln 7 Col 1'.

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_temporary_bug`

parfor issue: Solution



The screenshot shows the MATLAB editor window with the following code in the file 'inadvertent_temporary_fix.m':

```

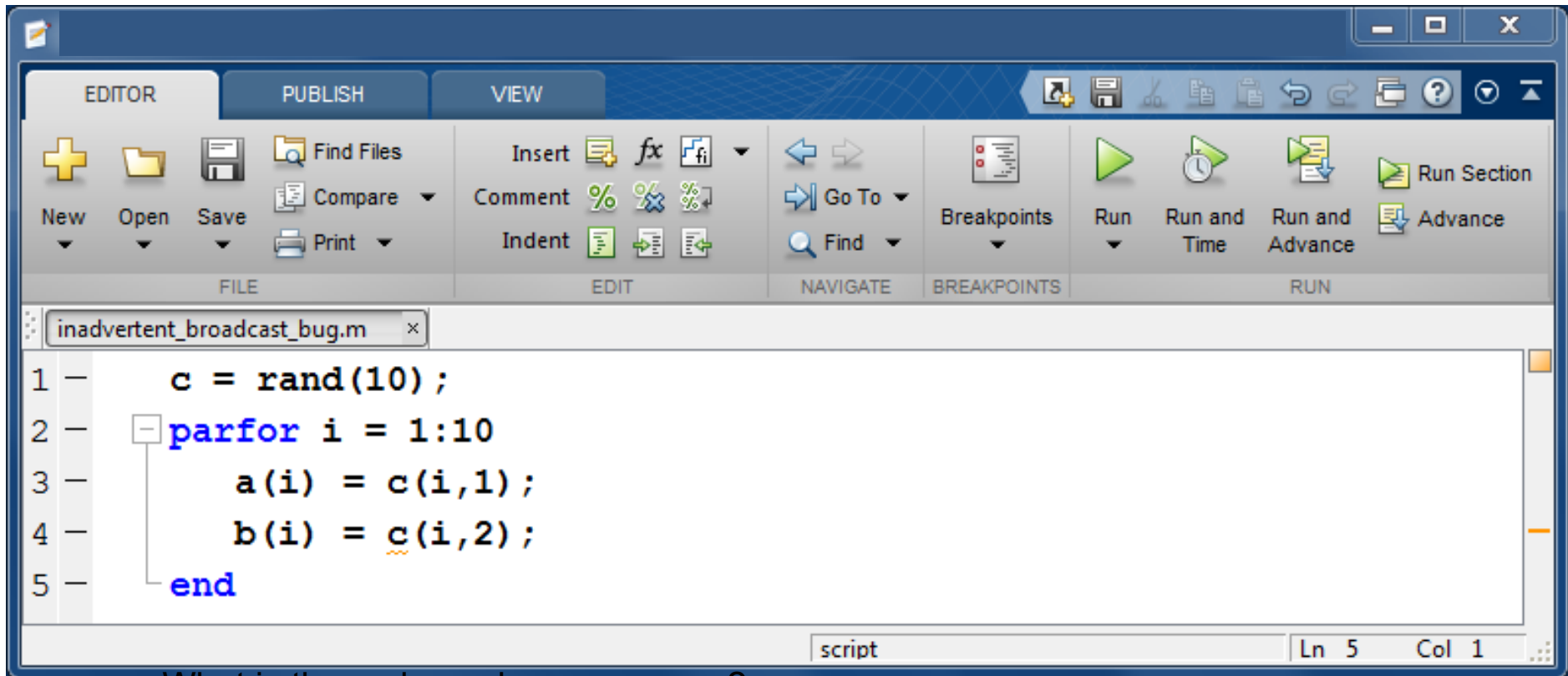
1 -   highest = -inf;
2 -   parfor idx = 1:10
3 -       value = idx;
4 -       highest = max(highest,value) ;
5 -   end
  
```

The status bar at the bottom indicates the current position is 'Ln 5 Col 1'.

Assign highest to the result of a reduction function

```
>> inadvertent_temporary_fix
```

parfor issue: Inadvertently Creating Broadcast Variables



```

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

The screenshot shows the MATLAB IDE editor window for a file named 'inadvertent_broadcast_bug.m'. The code contains a parfor loop where the variable 'c' is used in two different ways: as a sliced variable 'c(i,1)' and as a broadcast variable 'c(i,2)'. The status bar at the bottom indicates the cursor is at line 5, column 1.

What is the code analyzer message?

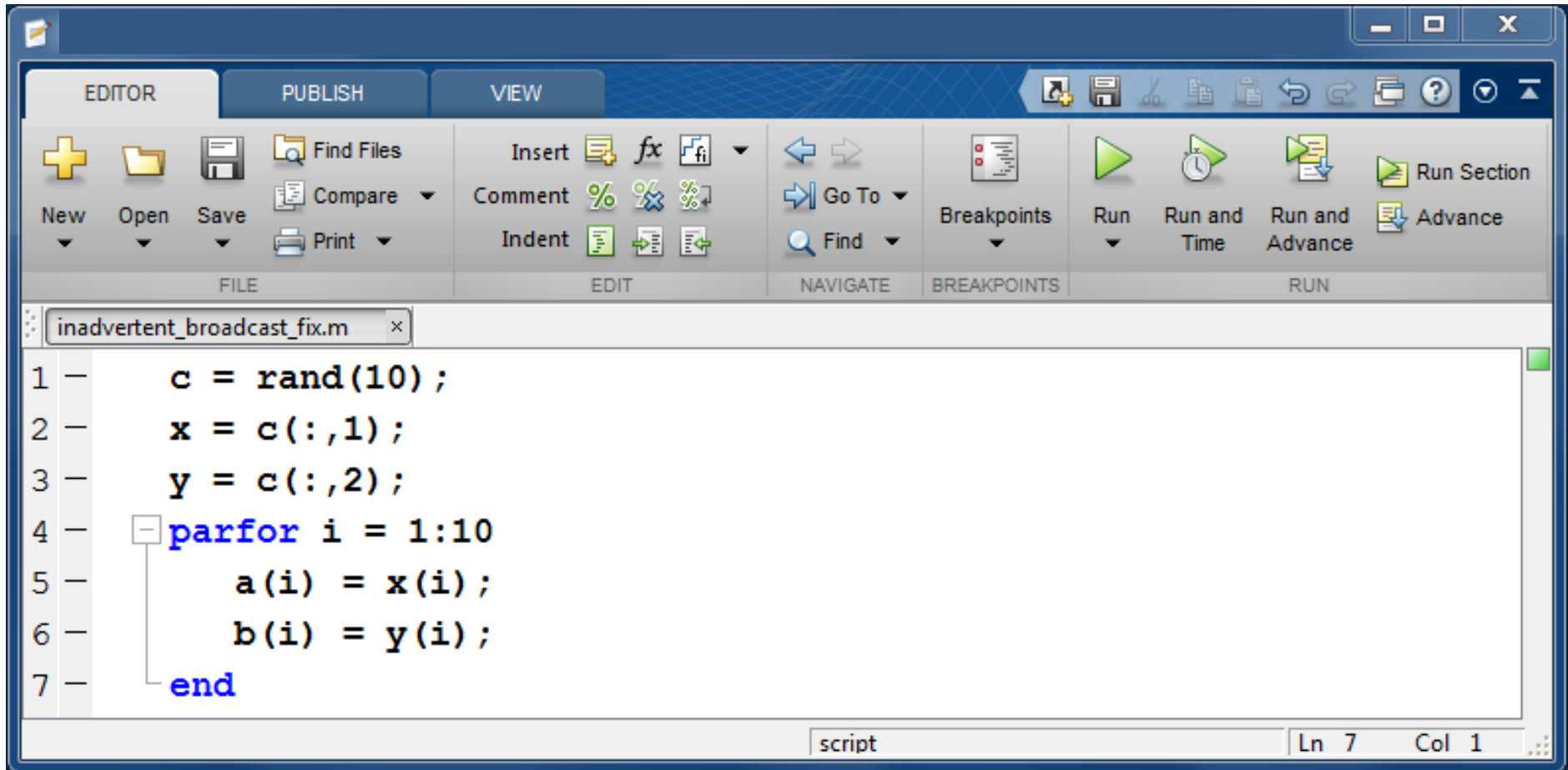
Why isn't `c` a sliced variable? What kind is it?

How can we make it sliced?

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

>> inadvertent_broadcast_bug

parfor issue: Solution



```

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
  
```

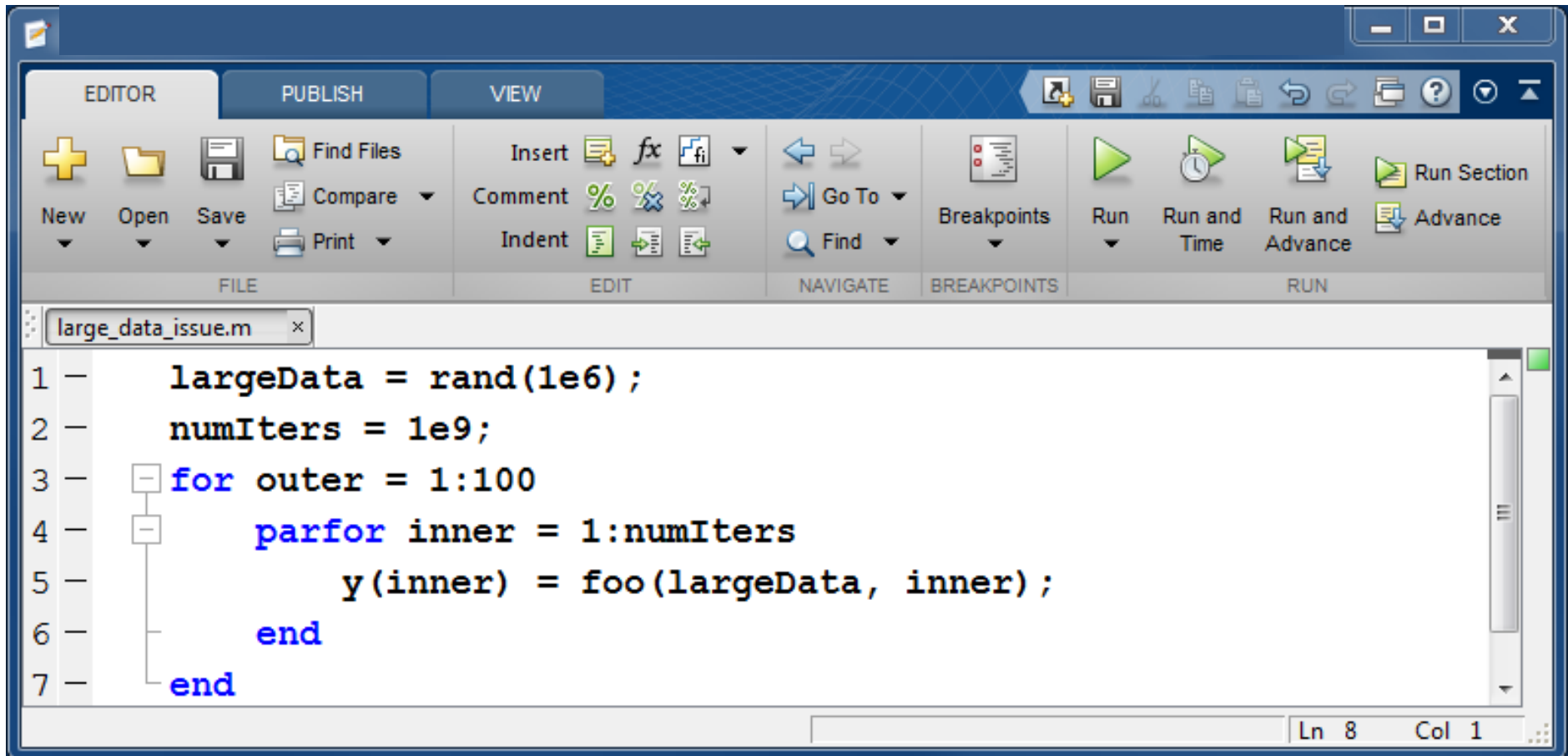
Create the additional variables x and y , which are sliced

```
>> 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 interface. The toolbar includes sections for EDITOR, PUBLISH, and VIEW. The EDITOR section contains icons for New, Open, Save, Find Files, Compare, and Print. The EDIT section contains icons for Insert, Comment, and Indent. The NAVIGATE section contains icons for Go To and Find. The BREAKPOINTS section contains a Breakpoints icon. The RUN section contains icons for Run, Run and Time, Run and Advance, Run Section, and Advance. The script editor shows the following code:

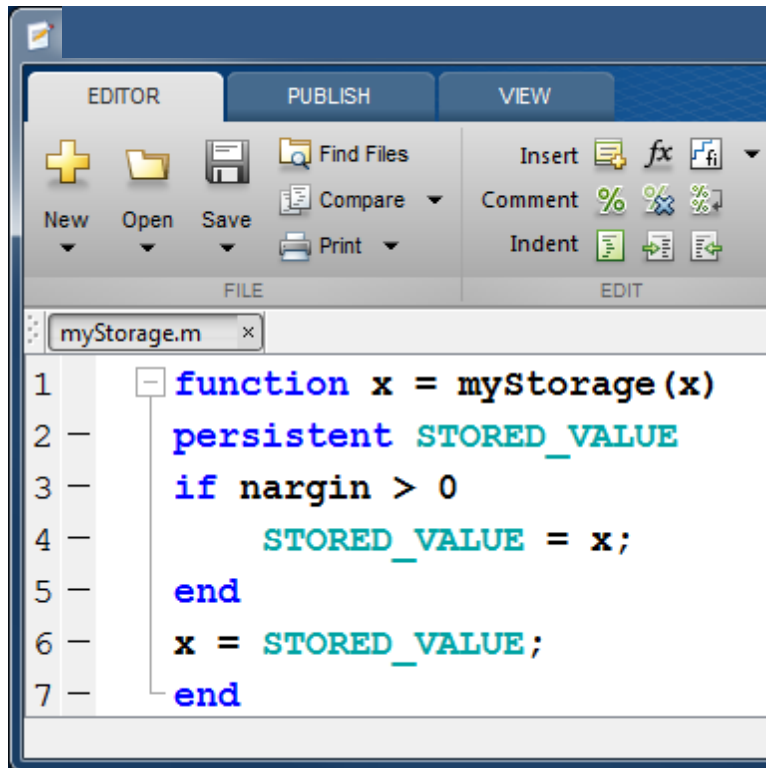
```

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

```

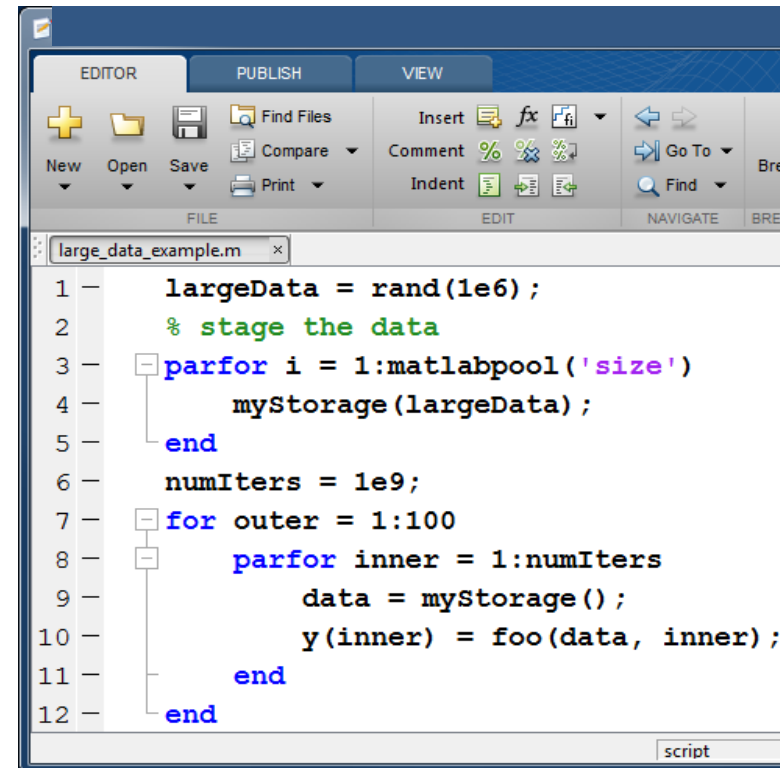
The status bar at the bottom right indicates the current position is Ln 8, Col 1.

Solution: Persistent Storage



```

1  function x = myStorage(x)
2  persistent STORED_VALUE
3  if nargin > 0
4      STORED_VALUE = x;
5  end
6  x = STORED_VALUE;
7  end
    
```



```

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
    
```

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

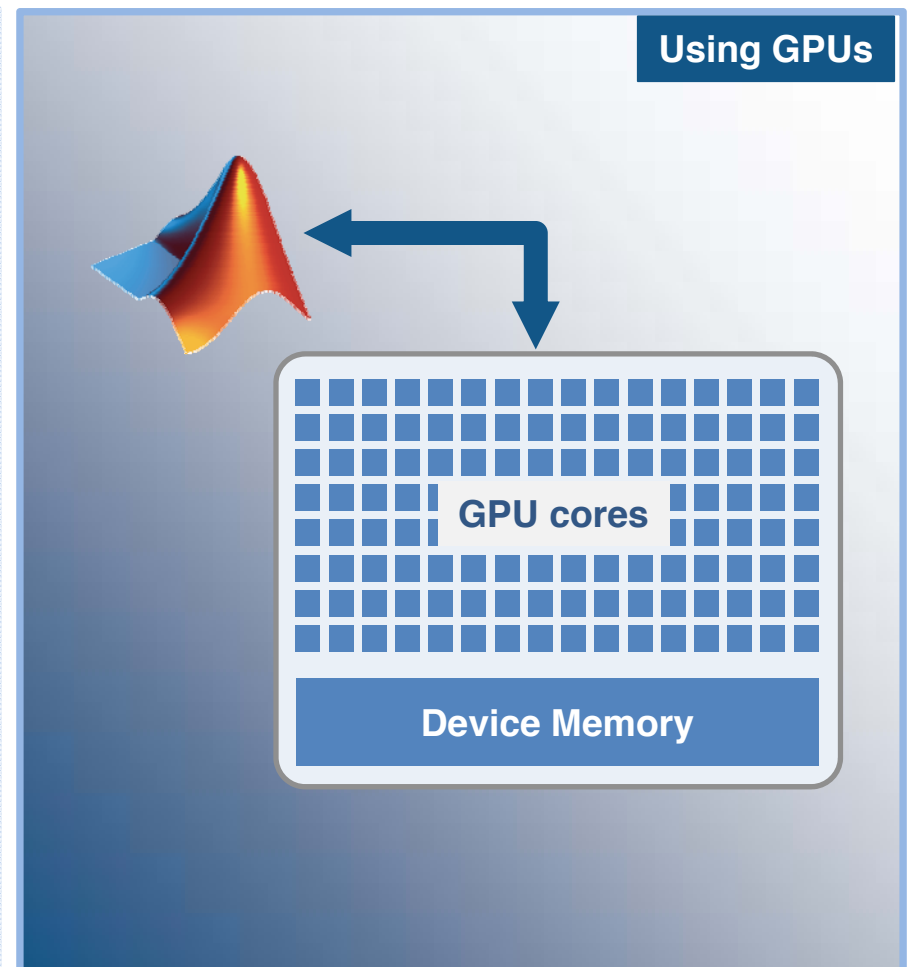
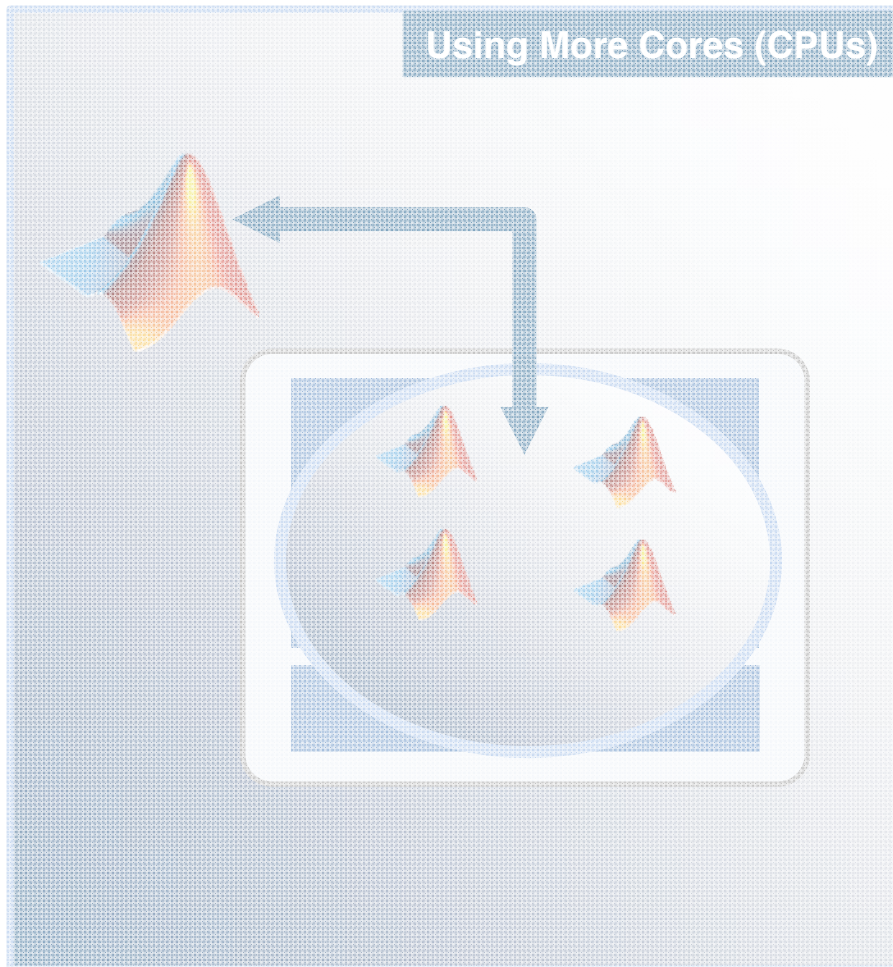
What is a Graphics Processing Unit (GPU)

- 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



* 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

Performance Gain with More Hardware



Programming Parallel Applications (GPU)



Ease of Use

- Built-in support with Toolboxes



Greater Control

Programming Parallel Applications (GPU)



Ease of Use

- Built-in support with Toolboxes
- Simple programming constructs:
`gpuArray`, `gather`



Greater Control

Example: Solving 2D Wave Equation

GPU Computing

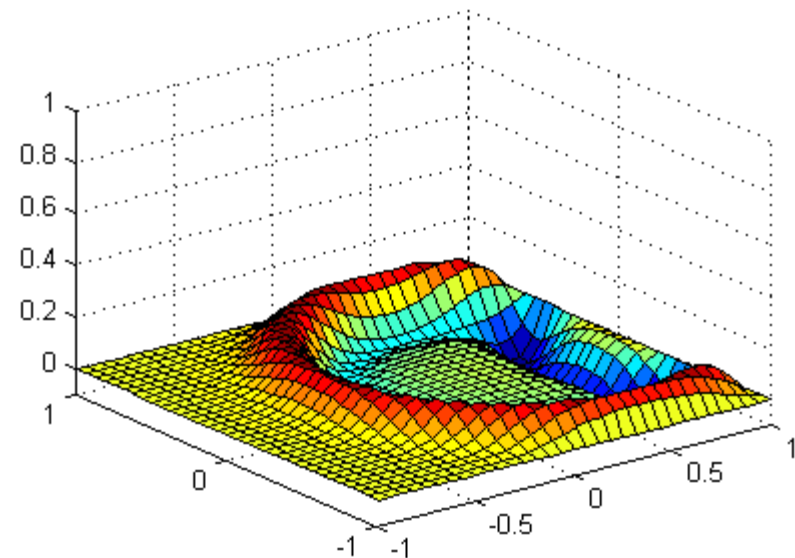
- Solve 2nd order wave equation using spectral methods:

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

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

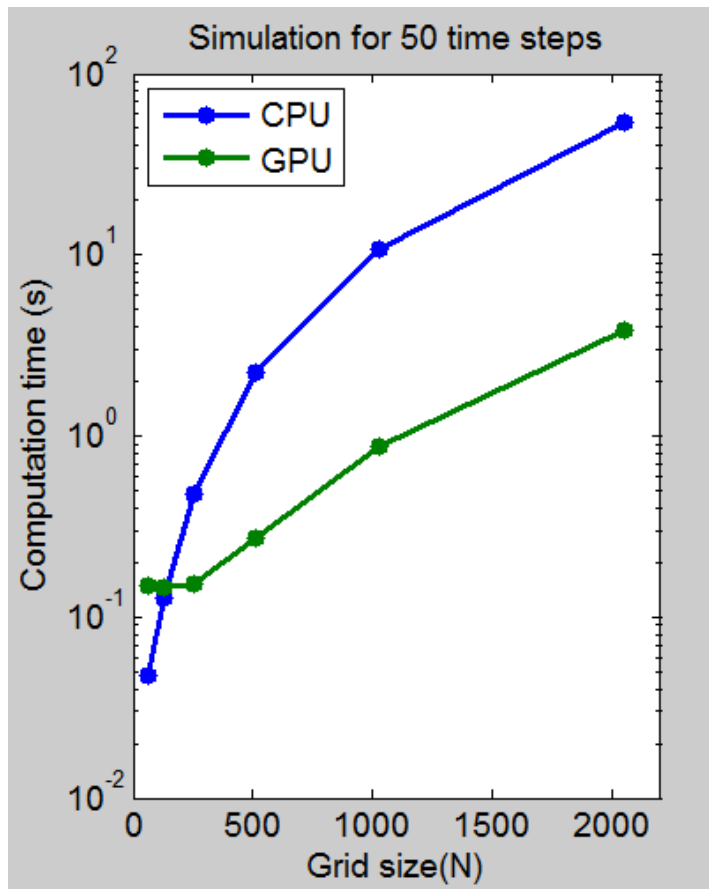
www.mathworks.com/help/distcomp/using-gpuarray.html#bsloua3-1

Solution of 2nd Order Wave Equation



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)



Ease of Use

- Built-in support with Toolboxes
- Simple programming constructs:
`gpuArray`, `gather`
- Advanced programming constructs:
`arrayfun`, `bsxfun`, `spmd`
- Interface for experts:
`CUDAKernel`, `MEX` support



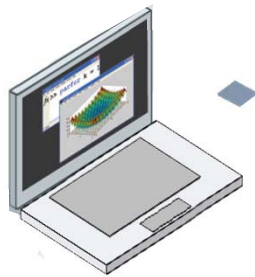
Greater Control

www.mathworks.com/help/releases/R2013a/distcomp/executing-cuda-or-ptx-code-on-the-gpu.html

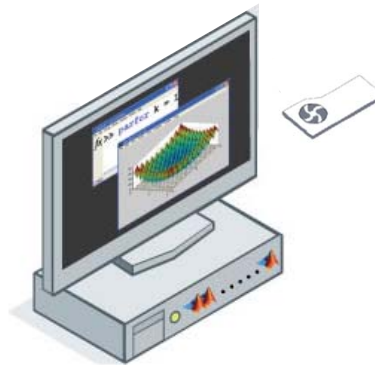
www.mathworks.com/help/releases/R2013a/distcomp/create-and-run-mex-files-containing-cuda-code.html

GPU Performance – not all cards are equal

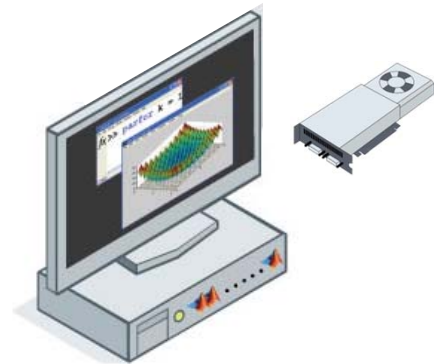
- Tesla-based cards will provide best performance
- Realistically, expect 4x to 15x speedup (Tesla) vs CPU
- See GPUBench on MATLAB Central for examples
www.mathworks.com/matlabcentral/fileexchange/34080-gpubench



Laptop GPU
GeForce



Desktop GPU
GeForce / Quadro

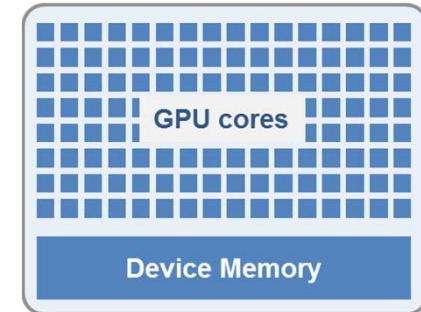


High Performance Computing GPU
Tesla / Quadro

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

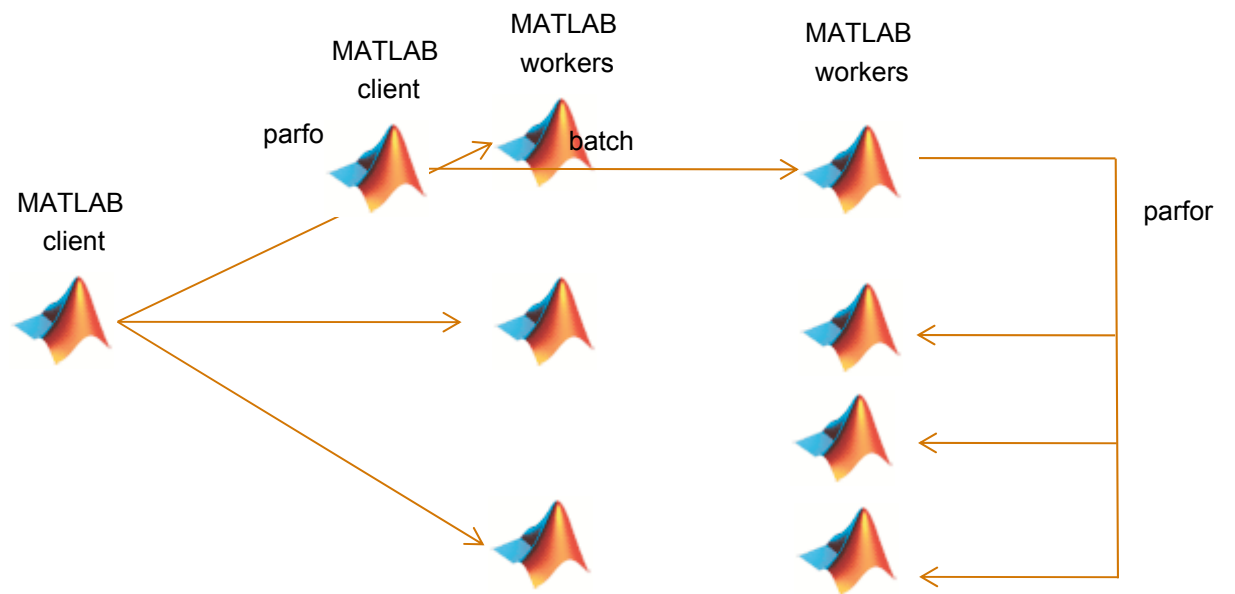
- **Algorithm consists of supported functions:**

- Growing list of Toolboxes with built-in support
 - www.mathworks.com/products/parallel-computing/builtin-parallel-support.html
- Subset of core MATLAB for `gpuArray`, `arrayfun`, `bsxfun`
 - 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

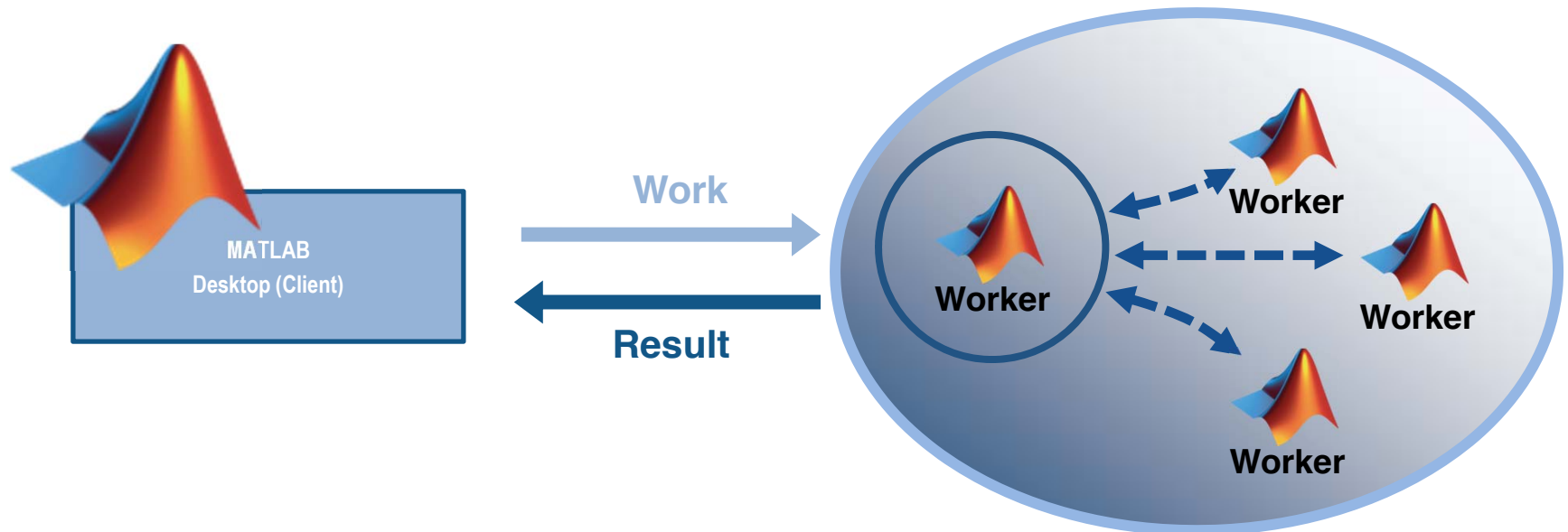
Outline

- 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

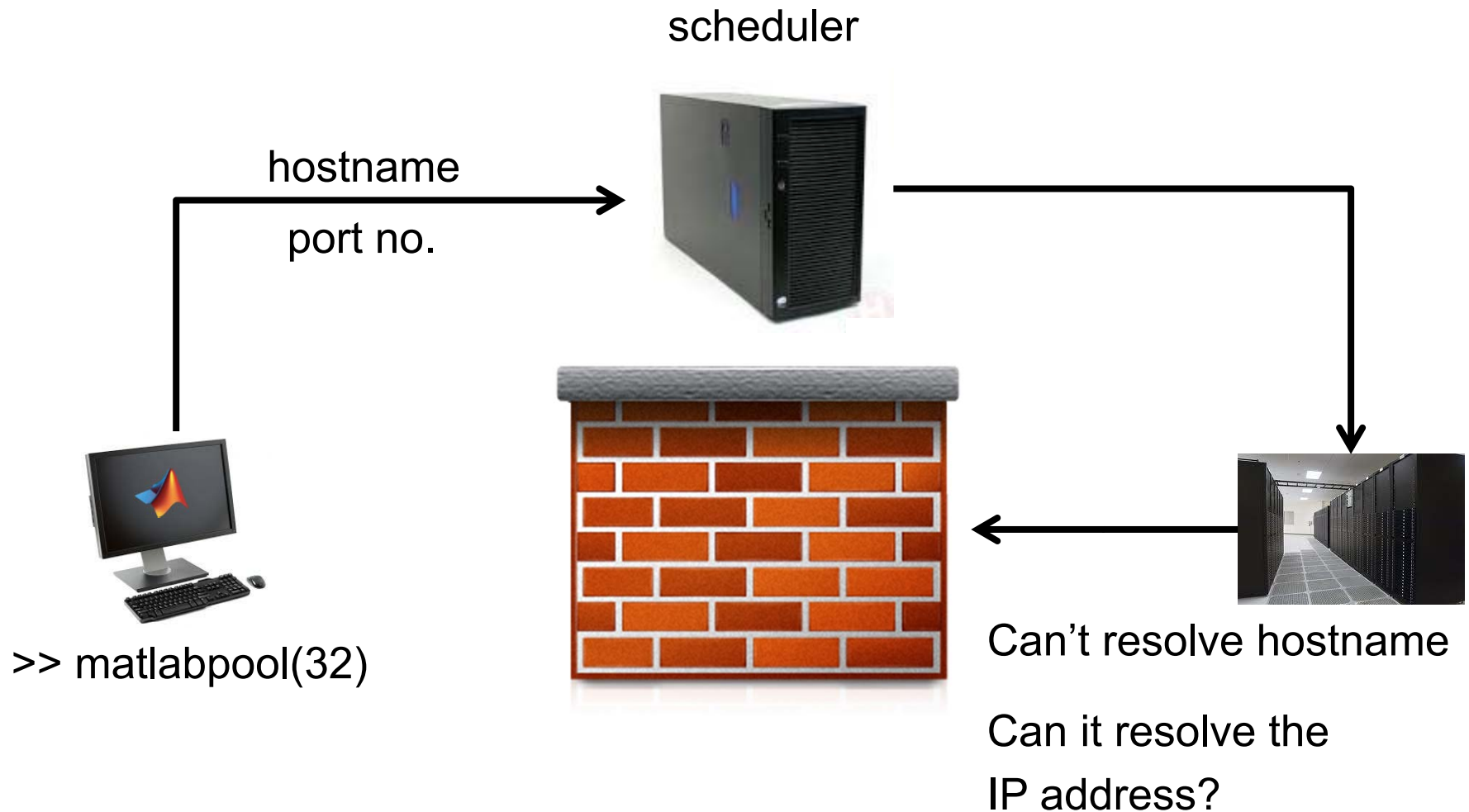




Can't I Just Use `matlabpool` to Connect to the Cluster/Cloud?

- 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?



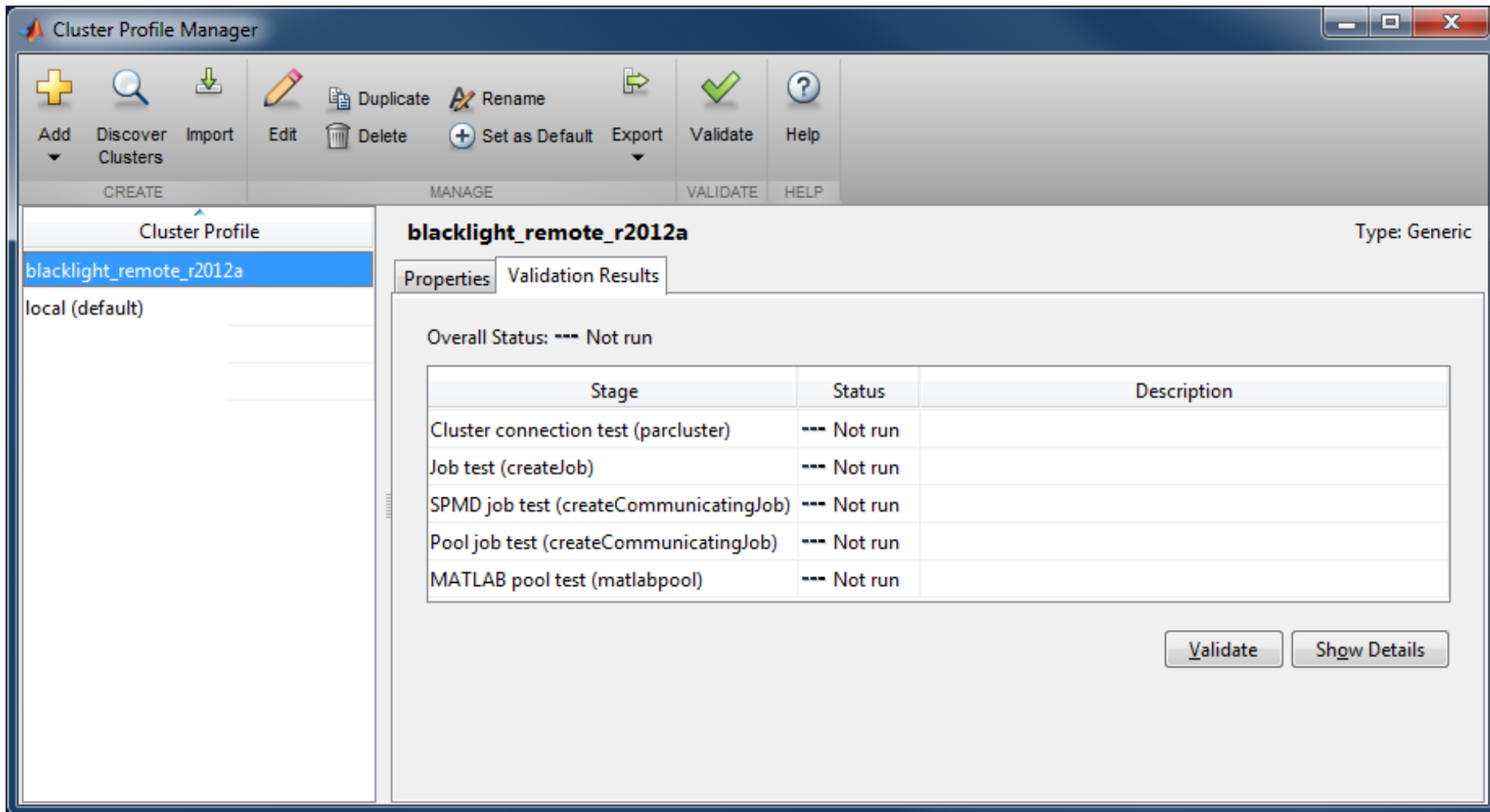
```
>> pctconfig('hostname', '12.34.56.78')
```



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

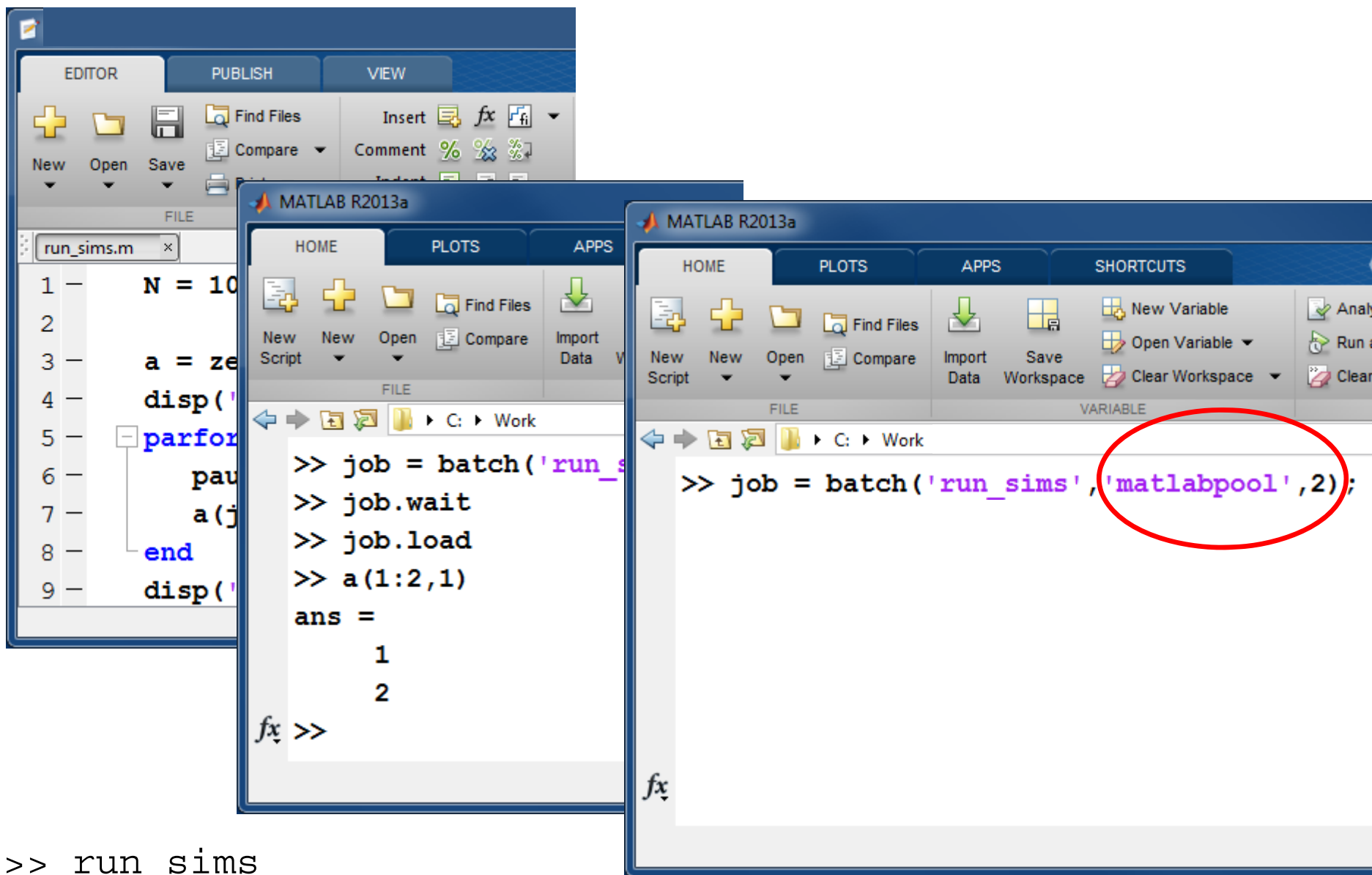
Import and Validating a Profile



The screenshot shows the Cluster Profile Manager window. The left pane lists cluster profiles, with 'blacklight_remote_r2012a' selected. The right pane shows the 'Validation Results' tab for this profile. The overall status is 'Not run'. A table below lists the stages and their statuses.

Stage	Status	Description
Cluster connection test (parcluster)	--- Not run	
Job test (createJob)	--- Not run	
SPMD job test (createCommunicatingJob)	--- Not run	
Pool job test (createCommunicatingJob)	--- Not run	
MATLAB pool test (matlabpool)	--- Not run	

Submitting Scripts with batch



The image shows three overlapping MATLAB R2013a windows. The background window displays a script named 'run_sims.m' with the following code:

```

1 - N = 10
2 -
3 - a = zeros(N,2,1)
4 - disp('')
5 - parfor j = 1:N
6 -     pause(1)
7 -     a(j,1:2,1) = j;
8 - end
9 - disp('')

```

The middle window shows the command window with the following commands and output:

```

>> job = batch('run_sims')
>> job.wait
>> job.load
>> a(1:2,1)
ans =
     1
     2
fx >>

```

The foreground window shows the command window with the following command, where 'matlabpool' and '2' are circled in red:

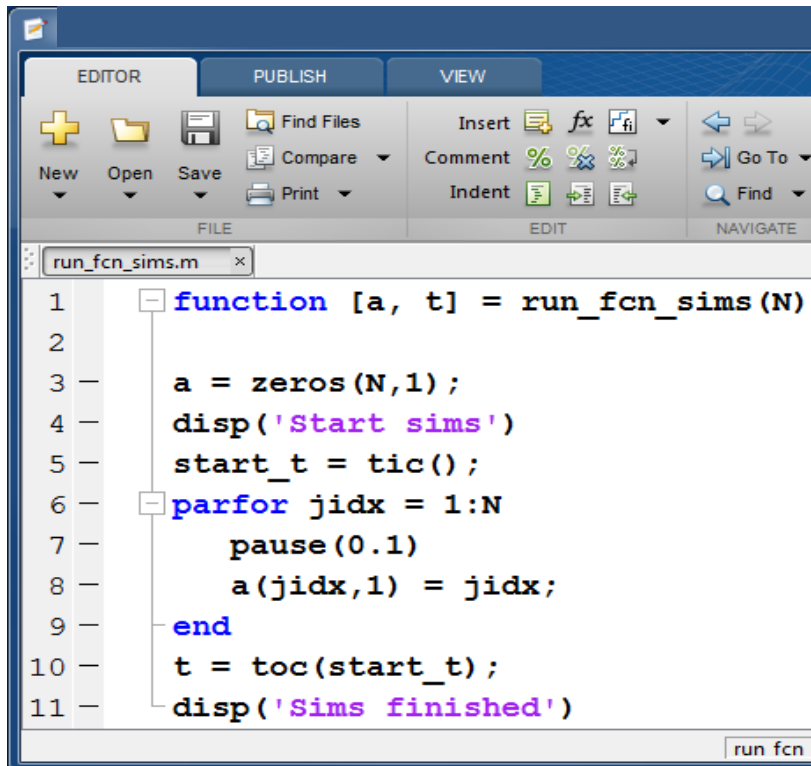
```

>> job = batch('run_sims', 'matlabpool', 2);

```

At the bottom left of the slide, the command `>> run_sims` is displayed.

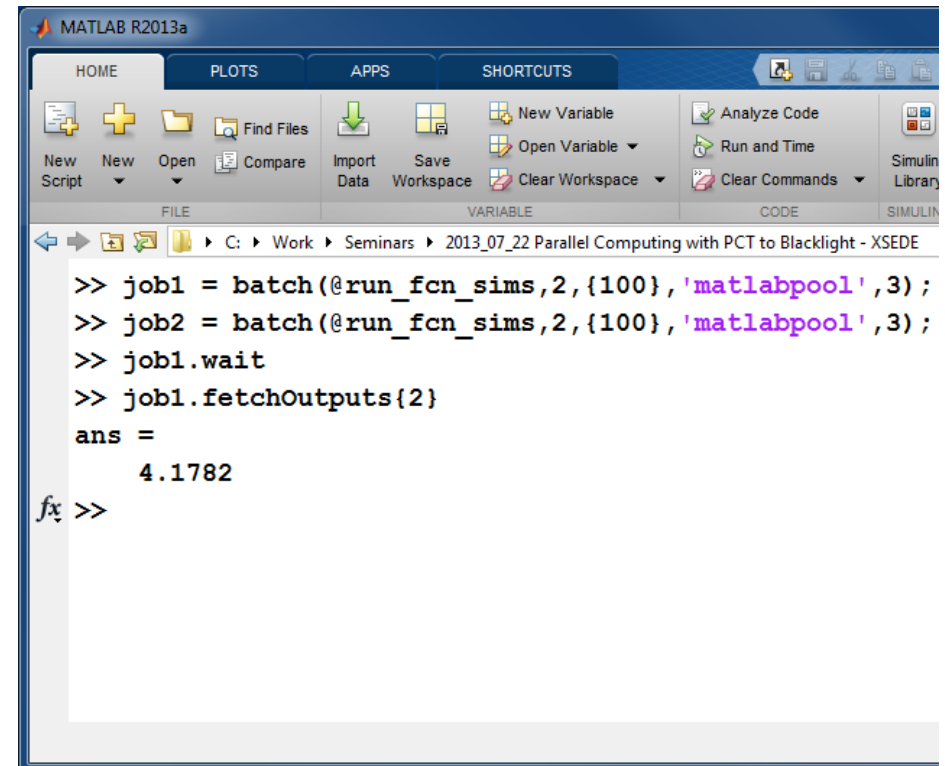
Submitting Functions with batch



```

1  function [a, t] = run_fcn_sims(N)
2
3  a = zeros(N,1);
4  disp('Start sims')
5  start_t = tic();
6  parfor jidx = 1:N
7      pause(0.1)
8      a(jidx,1) = jidx;
9  end
10 t = toc(start_t);
11 disp('Sims finished')

```



```

>> 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 >>

```

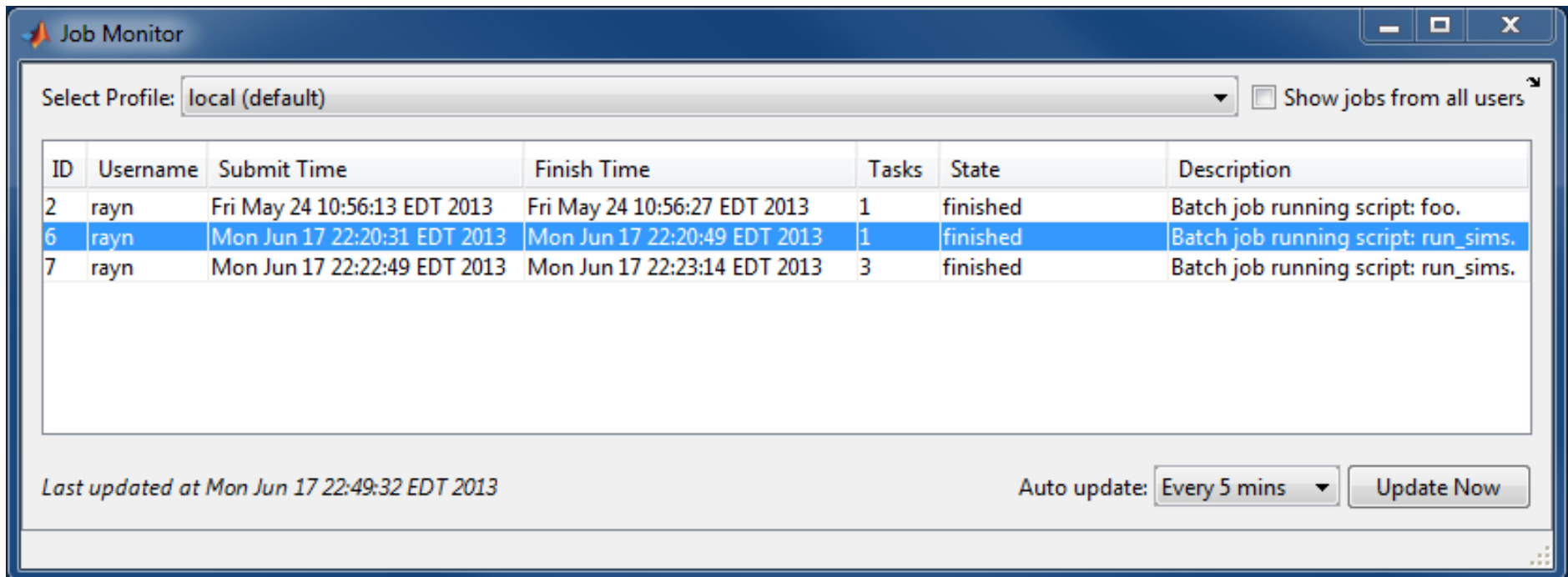
```
>> run_fcn_sims
```

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

Select Profile: local (default) Show jobs from all users

ID	Username	Submit Time	Finish Time	Tasks	State	Description
2	rayn	Fri May 24 10:56:13 EDT 2013	Fri May 24 10:56:27 EDT 2013	1	finished	Batch job running script: foo.
6	rayn	Mon Jun 17 22:20:31 EDT 2013	Mon Jun 17 22:20:49 EDT 2013	1	finished	Batch job running script: run_sims.
7	rayn	Mon Jun 17 22:22:49 EDT 2013	Mon Jun 17 22:23:14 EDT 2013	3	finished	Batch job running script: run_sims.

Last updated at Mon Jun 17 22:49:32 EDT 2013

Auto update: Every 5 mins

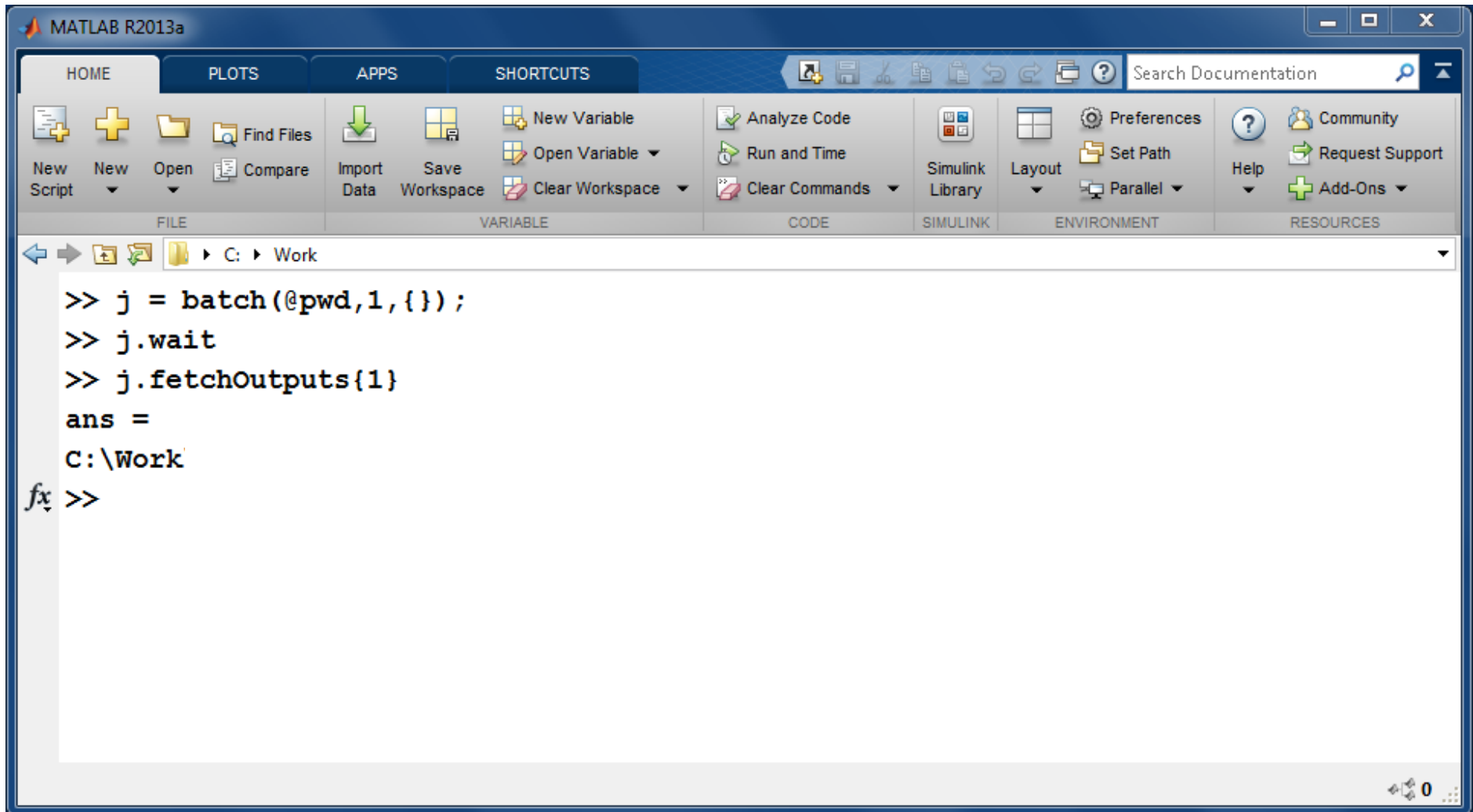
Job Monitor

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)

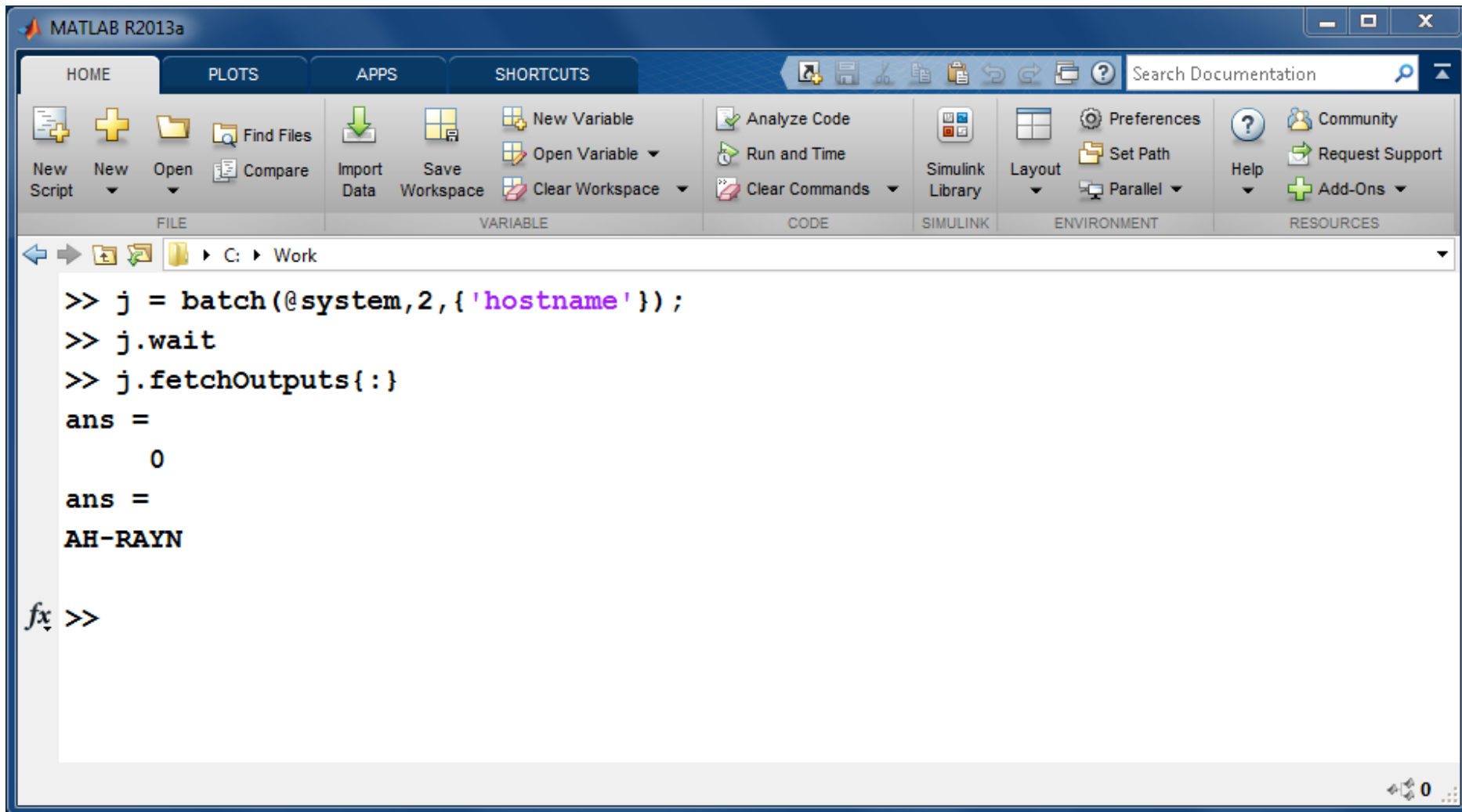


The image shows the MATLAB R2013a interface. The Command Window displays the following code and output:

```

>> j = batch(@pwd,1,{});
>> j.wait
>> j.fetchOutputs{1}
ans =
C:\Work
fx >>
  
```

Final Exam: Solution (2)



The image shows the MATLAB R2013a interface. The Command Window displays the following code and output:

```

>> 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 3rd 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