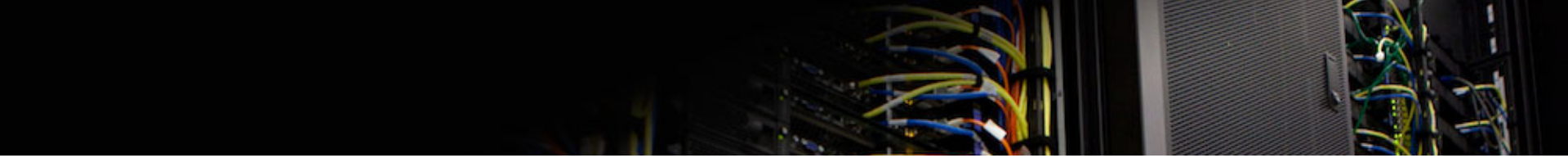


Statistical and Mathematical Software on HPC systems

Jefferson Davis

Research Analytics

Plan of Attack

- Look at three packages on Big Red II: SAS, R, Matlab.
- Look at running a common task in all three.
- Discuss a bit on how the different packages take advantage of multicore or parallel functionality.

What modules am I using?

module load matlab

module load r

module load sas

For all these most recent version of the package is the default.

What modules am I using? (BR II)

module load ccm

module load matlab

module swap PrgEnv-cray PrgEnv-gnu

module load r

module load sas

module load cudatoolkit

Three roads to parallelism

- Implicit parallelism (my favorite)
- Small modifications to existing code
- Fiddling around with mpi

SAS: background

- First developed in 1966 at North Carolina State for regression and analysis of variance.
- Commercialized in 1976 when the SAS Institute incorporates.
- Solid workhorse statistical package
- Still has not been used as a New York Times crossword clue, but both “Special Air Service” and “Scandinavian Airlines System” have.

SAS: starting it up

~> module load sas

SAS data analysis and management system version 9.4 loaded.

~> sas –nodms

More common to write the sas program and run it as a script

~> sas lineExample.sas

A SAS program has two steps: a data step and a proc step.

SAS: loading data and regressing

Look at the example of fitting a line through some data.

The csv file faithful.csv has 272 values.

We use SAS to find the line of best fit.

	A	B
1	3.6	79
2	1.8	54
3	3.333	74
4	2.283	62
5	4.533	85
6	2.883	55
7	4.7	88
8	3.6	85
9	1.95	51
10	4.35	85
11	1.833	54
12	3.917	84
13	4.2	78
14	1.75	47
15	4.7	83
16	2.167	52
17	1.75	62
18	4.8	84
19	1.6	52
20	4.25	79
21	1.8	51
22	1.75	47

SAS: loading data and regressing

The file lineExample.sas

```
data faithful;
 infile "faithful.csv" delimiter=",";
 input x y;
run;
proc reg;
  model y = x;
run;
```

~>sas lineExample.sas

The output goes to lineExample.lst

SAS: loading data and regressing

~> tail -n 7 lineExample.lst

Parameter Estimates

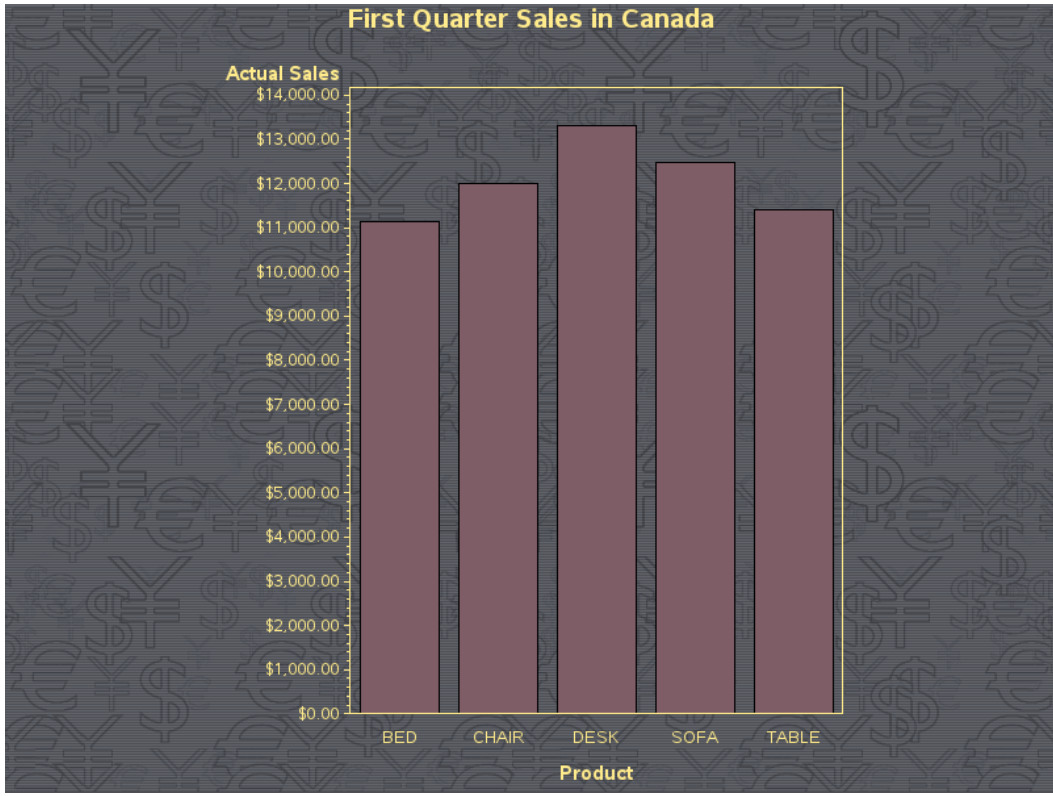
Variable	DF	Parameter Estimate	Standard Error	t Value	Pr > |t|
Intercept	1	33.47440	1.15487	28.99	<.0001
x	1	10.72964	0.31475	34.09	<.0001

Sure, y = 10.72964 x + 33.47440 seems okay.

SAS: producing graphics

Running canada.sas will save create an HTML file and a graphic in gchart.png

```
ods listing close;
ods html style=money
file="CanadaGraph.html" ;
proc gchart data=sashelp.prdsale;
vbar Product / sumvar=actual;
title1 "First Quarter Sales in Canada";
where Quarter=1 and
    Country="CANADA";
run;
quit;
ods html close;
ods listing;
```



~>sas canada.sas

SAS: implicit parallelism exapmple

- SAS will create threads to run faster on multicore environments.
- This happens by default but we can see the improvement if we force SAS to run without threads
- The next example involves taking the mean of a set of 50,000,000 random numbers

SAS: implicit parallelism exapmple

```
%let NObs = 50000000;
data Unif(keep=u);
call streaminit(123);
do i = 1 to &NObs;
  u = rand("Uniform");    /* U[0,1] */
  output;
end;
run;

proc means data=unif;
var u ;
run;
```

NOTE: PROCEDURE MEANS used
(Total process time):
real time 1.76 seconds
cpu time 4.37 seconds

```
%let NObs = 50000000;
data Unif(keep=u);
call streaminit(123);
do i = 1 to &NObs;
  u = rand("Uniform");    /* U[0,1] */
  output;
end;
run;

proc means data=unif;
options nothreads;
var u ;
run;
```

NOTE: PROCEDURE MEANS used
(Total process time):
real time 3.63 seconds
cpu time 3.57 seconds

R: background

- First created in the early 1990s by Ross Ihaka and Robert Gentleman as an implementation of S.
- Development soon shifted to a larger core group.
- Distributed under the GNU General Public License.

R: starting it up

```
~> module load r
~> R
```

R: some useful tidbits

- The question mark will display a function's help text. This is a shortcut for the help() function
  help(sin)
  ? sin
- The command invisible() suppresses output
- The up arrow key will go back to previous commands
- The command system() is used for shell commands
  system("rm core")
- The rm() command clears variables
  rm(list=ls(all=TRUE)) #Clear all variables
- The hash tag is used for comments
  #This is an R comment

R: loading data and regressing

Let's load the file faithful.csv again and rerun the earlier regression.

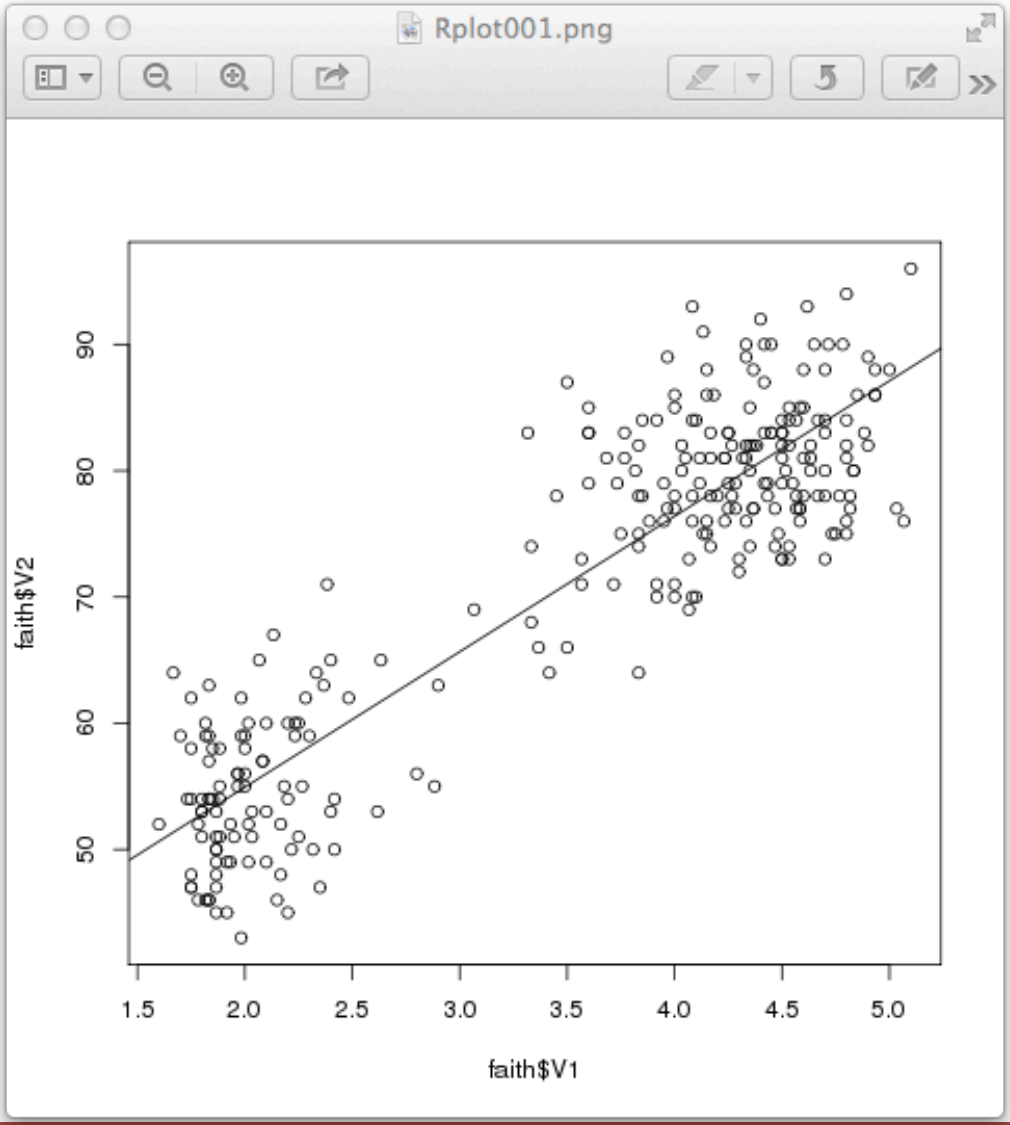
```
faith<-read.csv("faithful.csv", header=FALSE)
faith
        V1 V2
1    3.600 79
2     1.800 54
lm(faith$V2~faith$V1)
Call:
lm(formula = faith$V2 ~ faith$V1)
Coefficients:
(Intercept)     test2$V1
       33.47         10.73
```

R: plotting data

```
fit<-lm(faith$V2~faith$V1)
png()
plot(faith$V1,faith$V2)
abline(fit$coefficients)
dev.off()
```

Default name is “Rplot001.png”.

Multiple plots are saved in multiple files.



R: multicore versions of apply()

R has many mapping functions that apply a function to the elements of a list, vector, what-have-you.
The function lapply() applies a function to the elements of a list. The function mclapply() is a multicore version of lapply.
The function rep(m,n) repeats m for n time. So lapply(rep(100,1000000),rnorm) returns 1000000 lists of 100 random numbers from a normal distribution.

st<-system.time(
  lapply(rep(100,1000000),
  rnorm))
st[3]
	library(parallel)
	stm<-system.time(
	  mclapply(rep(100,1000000),
	  rnorm,mc.cores=32))
	stm[3]

elapsed 22.634	elapsed 4.066

Matlab: background

- Developed by Cleve Moler in the 1970s to give students easier access to numerical libraries for linear algebra (Matrix Laboratory)
- MathWorks company founded in 1984 for commercial development
- About 1900 IU network users 2013-14 academic year
- Decent support for parallelism

Matlab: starting it up

~> module load matlab

MATLAB numerical calculation framework version 2014a loaded.

~> matlab

Matlab: some useful tidbits

- The help command will display a function's help text. The doc command brings up more information
  help sin
  doc sin
- The semi-colon (;) will suppress output
- The up arrow key will go back to previous commands
- Typing and then using the up arrow key goes back to previous commands that start with that text
- The exclamation point is used for shell commands
  ! rm matlab_crash_dump.*
- The percent sign is used for comments
  %This is a Matlab comment

Matlab: loading data and regressing

```
faith=csvread('faithful.csv');
x=faith(:,1);
y=faith(:,2);
fit=fitlm(x,y,'linear')
```

Linear regression model:

y ~ 1 + x1

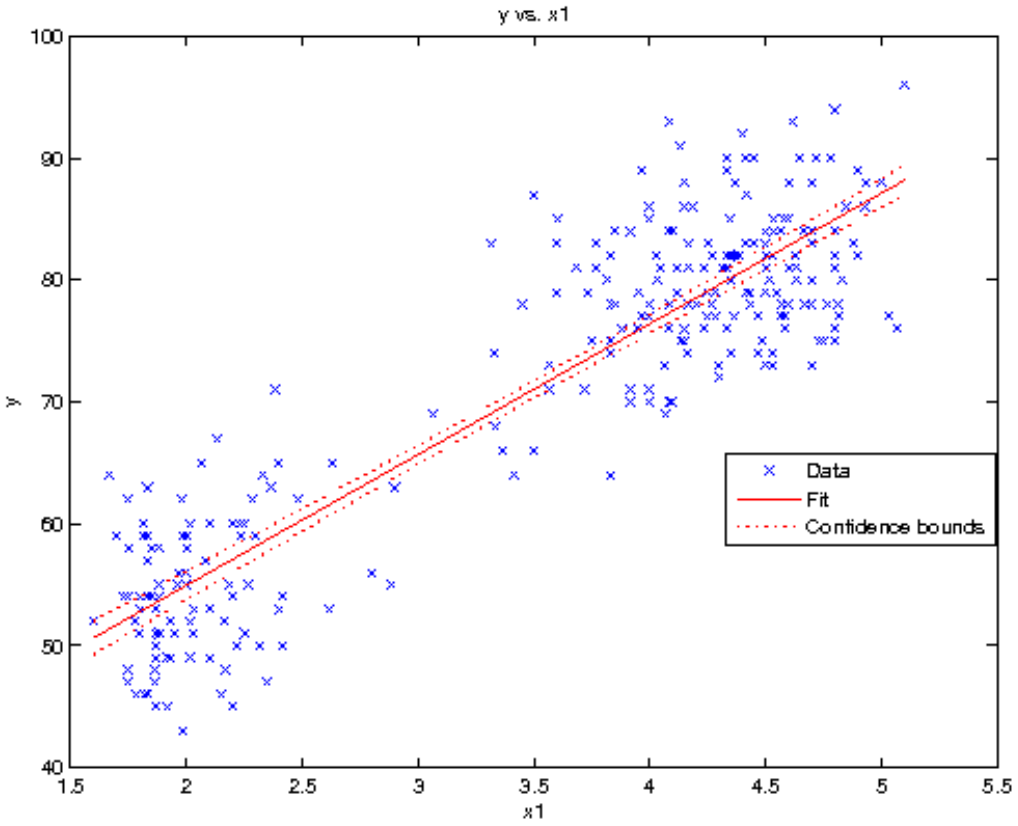
Estimated Coefficients:

	Estimate	SE	tStat	pValue
(Intercept)	33.474	1.1549	28.985	7.136e-85
x1	10.73	0.31475	34.089	8.13e-100

Matlab: plotting data

```
fit.plot
print( gcf,…
  '-dpng', 'MatlabPlot' )
```

This saves the plot as MatlabPlot.png.



Matlab: implicit parallelism in svd

Many functions will recognize that the multicore environment and create an appropriate number of threads.

Look at an example of two singular value decompositions.
(Rewriting a matrix as the product of “nice” matrices.)

Matlab: implicit parallelism in svd

> matlab tic svd(rand(5000)) toc	> matlab -singleCompThread tic svd(rand(5000)) toc
Elapsed time is 67.720321 seconds.	Elapsed time is 157.032804 seconds.

Older versions of Matlab allowed you to set the maximum number of threads with maxNumCompThreads, but this is being deprecated.

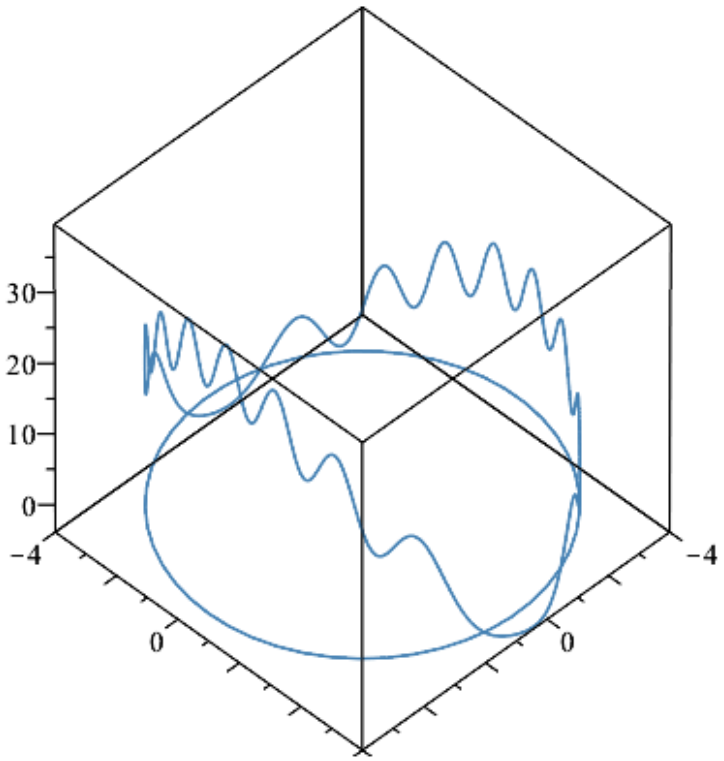
Matlab: svd using the gpu

> qsub -I -q debug_gpu -lgres=ccm > ccmlogin > matlab tic svd(rand(5000)) toc	> qsub -I -q debug_gpu -lgres=ccm > ccmlogin > matlab tic svd(rand(5000,'gpuArray')); toc
Elapsed time is 62.310409 seconds.	Elapsed time is 16.616624 seconds.

Matlab: parallel-enabled functions

Many Matlab functions can use a pool of worker processes if you explicitly create them and tell the function to use them.

Sample problem: maximize the function $x_1^2 + 4*\sin(5*x_2)$ subject to the constraint $(x_1-1)^2 + (x_2-1)^2 = 25$



We first write a function to define the constraint mycon.m.

```
function [c,ceq] = mycon(x)
   c = (x(1)-1)^2 + (x(2)-1)^2 - 25;
   ceq = [];
```

Matlab: parallel-enabled functions

Then we set up the problem

```
opts = optimset('Algorithm','sqp');
problem = createOptimProblem('fmincon','objective', ...
        @(x) x(1)^2 + 4*sin(5*x(2)),'x0',[3 3],'lb',[-5 -5], ...
        'ub',[5 5],'nonlcon',@mycon,'options',opts);
ms = MultiStart;
```

The Matlab Multistart solver runs an optimizer from multiple start points. It's natural to want to run it in parallel.

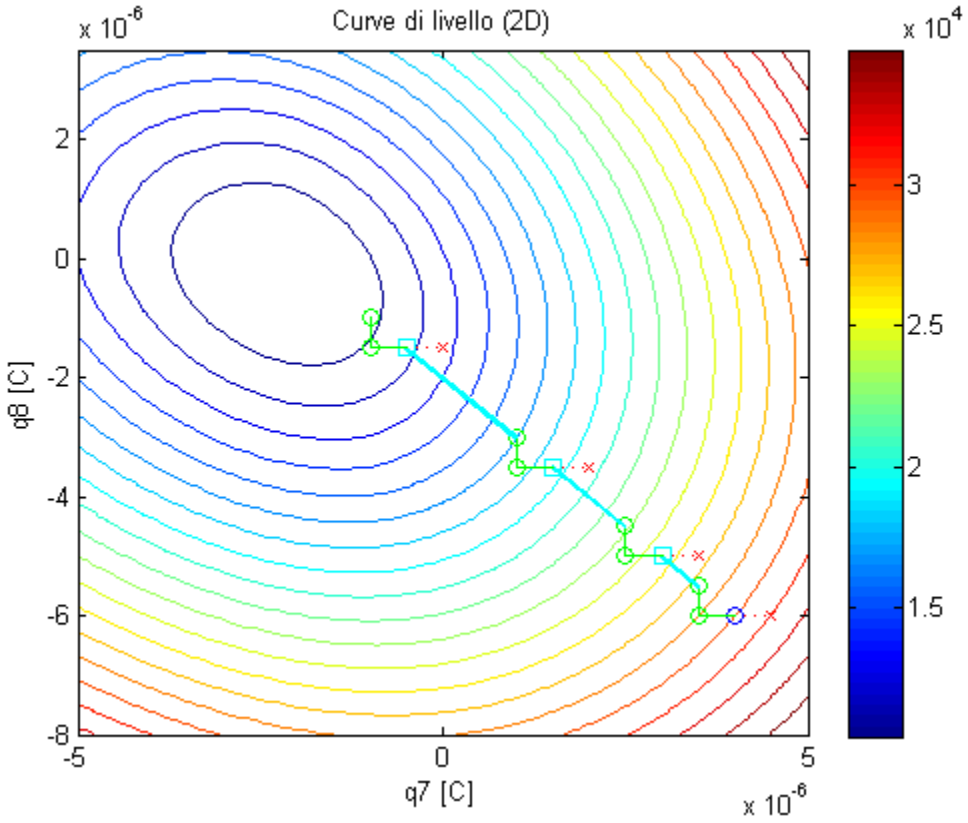
ms.UseParallel = false; tic [x,f] = run(ms,problem,2000); toc	ms.UseParallel = true; tic [x,f] = run(ms,problem,2000); toc
Elapsed time is 48.502433 seconds.	Elapsed time is 4.673513 seconds.

Matlab: parallel-enabled functions

Matlab will more more than happy to let you run things in parallel even is it's a really bad idea.

The optimizer patternsearch is an example. At each step patternsearch checks the values of the objective function at near the current point. The first point with a lower value becomes the current point for the next step.

You can, however, use a pool of workers and check them in parallel. This means checking all the nearby points.



Matlab: parallel-enabled functions

The code on the right is checking a full $2^{10} = 1024$ points at each step. Running it in parallel doesn't make up for that.

```
x0=.4*ones(10,1);
tic
patternsearch(@(x)myFun(x,params),
...
   x0,[],[],[],[],0*x0,1+0*x0,[]);
toc
```

Elapsed time is 0.476223 seconds.

```
parpool(32)
options = psoptimset('UseParallel',
true,...
'CompletePoll', 'on', 'Vectorized',
'off');
x0=.4*ones(10,1);
tic
patternsearch(@(x)myFun(x,params),
...
   x0,[],[],[],[],0*x0,1+0*x0,[],options);
toc


delete(gcp)
```

Elapsed time is 4.673513 seconds.

Matlab: parallel for loops

If you have a pool of parallel workers you use them to run a for-loop with parfor.

tic
for i=1:50000
  a(i)=max(rand(100));
end
toc
	parpool(16)
	tic
	parfor i=1:50000
	  a(i)=max(rand(100));
	end
	toc
	delete(gcp)

Elapsed time is 11.202063 seconds.	Elapsed time is 2.629504 seconds.

Matlab: parallel for loops

The are some restrictions on the loop, but the main one is that the order of evaluation can't matter. So the code below fails

```
%Fibonacci failue
a(1)=1;a(2)=1;
parpool(2)
parfor i=3:100
a(i)=a(i-1)+a(i-2);
end
delete(gcp)
```

Contact info

majdavis@iu.edu

researchanalytics@iu.edu