BOSTON UNIVERSITY

MATLAB for High Performance Computing

Shaohao Chen

Research Computing Services

Information Services and Technology

Boston University

Why using Matlab for HPC?

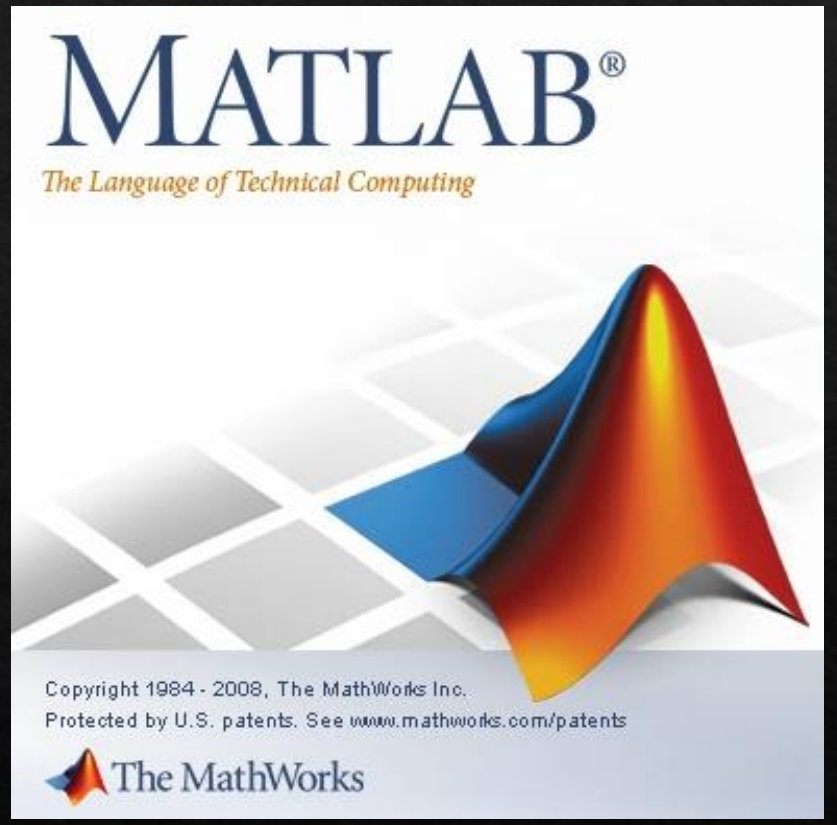
- Many Matlab programs run faster on high-performance computing (HPC) clusters than on regular laptops/desktops.
- Boston University (BU) Shared Computing Cluster (SCC) is an HPC cluster with over 11,000 CPU cores and over 250 GPUs.
- MATLAB site license is available to all BU users. (Unlimited number on SCC).

- Matlab programs can be exceptionally fast if they are well-designed, and painfully slow if not. It is necessary to optimize MATLAB codes to obtain good performance.
- The code-optimized methods to be mentioned in this tutorial can be used not only on HPC clusters but also on regular laptops/desktops.

- The skills you learn today should enable you to solve bigger problems faster using MATLAB.

Outline

- Use Matlab on BU SCC
  - ✓ Start up
  - ✓ Submit Matlab jobs
  - ✓ Distributed jobs
- Optimize Matlab codes
  - ✓ Remove unnecessary works
  - ✓ Optimize memory usage
  - ✓ Use optimized built-in functions/operators



Access to BU SCC resources

- Log in to SCC:

```
$ ssh -X username@scc1.bu.edu
```

- Interactive session: working interactively on compute nodes.

```
$ qrsh          #  Start an interactive session
```

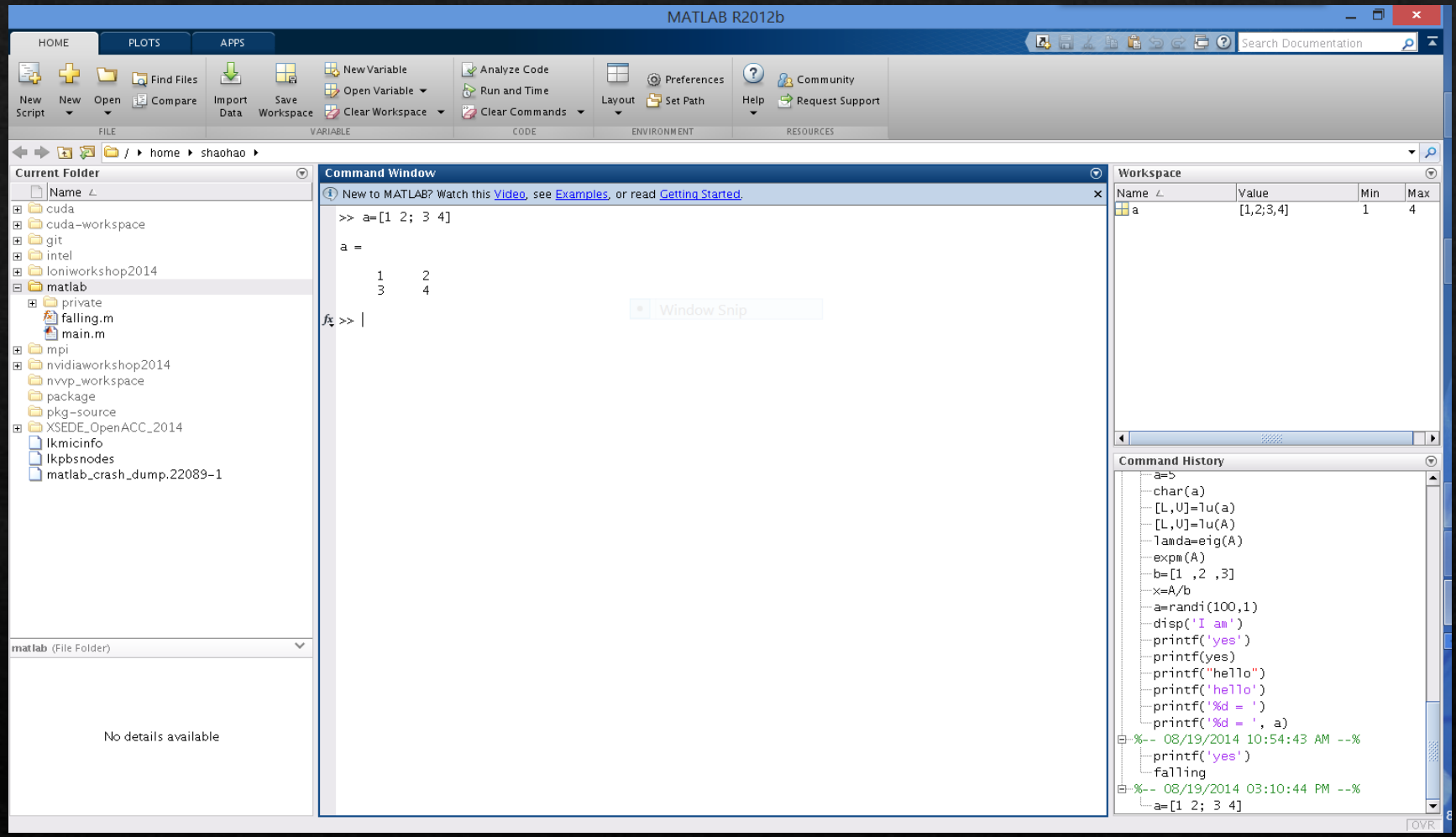
✓ Any job costing more than 15-minute CPU time on the login node will be killed.

- Load Matlab module:

```
$ moduleavail | grep matlab      # See all available versions
$ module load matlab             # Set up environment variables
```

Graphic platform

- Open Matlab
  $ matlab &

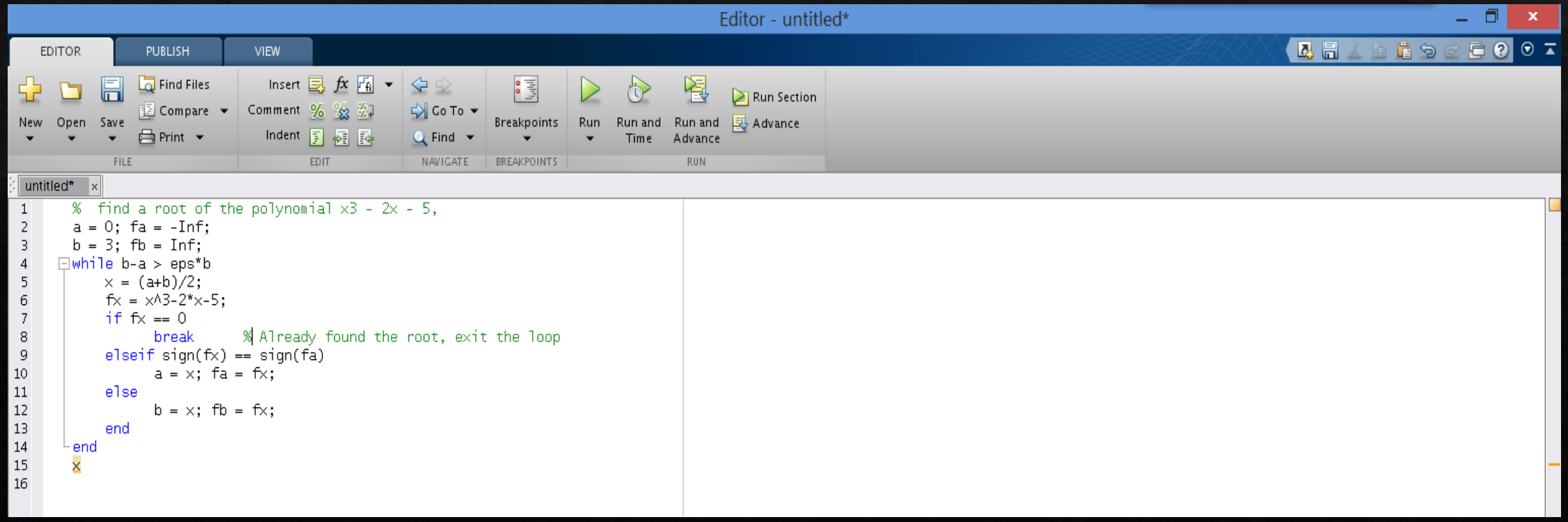


- Use VNC to speed up graphical interface.

Refer to: https://www.bu.edu/tech/support/research/system-usage/getting-started/remote-desktop-vnc/

M-file

- An m-file is a simple text file where you can place MATLAB commands.
- Save your works
- Convenient for debugging
- Run directly. Pre-compiling is unnecessary.



Text platform

```
$ matlab -nodisplay      % Work in text interface. Does not display any graph.
$ matlab -nodesktop      % Program in text interface. Pop out graphs when necessary.
```

- Many Linux commands (some prefixed with an exclamation mark) are available within Matlab platform, such as:

  cd, ls, pwd, !cp, !rm, !mv, !cat, !vim, !diff, and !grep

- Edit M-file and run the program:

```
>> !vim mfilename.m        %  edit in text window
>> edit mfilename.m        %  create a new or open an existing m-file in graphical window
>> open mfilename.m        %  open an existing m-file in graphical window
>> run mfilename.m         %  run the program
>> mfilename               %  run the program
```

Submit a batch job to background

- Submit a batch job to background using a script

```
$ qsub script.sh
```

- ✓ Write a batch job script using any Linux editor (such as vim, emacs, gedit, or nano).
- ✓ A typical batch script for a Matlab job (Supposed that source codes are saved in an m-file):

```
#!/bin/bash -l      #  Start a bash script. The option -l is necessary to enable module tool.
#$ -pe omp 1        #  Request 1 CPU core
#$  -l h_rt=12:00:00    # Request wall time
#$ -N jobname           # Give a job name
module load matlab    #  Set up environment variables for running matlab
matlab -nodisplay -singleCompThread -r "addpath /path/to/work; mfile_name"    # Run program
```

Exercise 1

- Run a Matlab program on BU SCC:

  Request for an interactive session first.

  i) Write a Matlab code in an m-file to print “Hello world” (e.g. use the disp function).

  ii) Run the program interactively in Matlab platform.

  iii) Submit a batch job to run the program in the background.

Standalone executable

- Create a standalone executable

```
$ mcc -mv -o myexe  name.m
```

- ✓ -mv produces a standalone and shows actions taken
- ✓ -o myexe specifies the executable name
- ✓ The name of the m file must be the same as the main function name in it.
- ✓ A script named run_myexe.sh is automatically created for running jobs in Linux bash shell.

- Run the executable (in Linux shell environment)

```
$ module load mcr      # Set up environment variables for running the executable.
$ mcr ./myexe      # Execute the executable using MATLAB Compiler Runtime (MCR).
```

- ✓ There is no considerable performance difference between running a standalone executable and running an m-file directly. The later one is more recommended since it is simpler.

Distributed jobs

- Submit multiple jobs using one command line.
- Distribute independent jobs:

```
$ qsub  -t  1-4  script.sh
```

- ✓ The batch system sets up the SGE_TASK_ID environment variable which can be used to pass the task ID to the program, for example:

```
matlab -nodisplay -singleCompThread -r "rand($SGE_TASK_ID); exit"
matlab -nodisplay -singleCompThread -r "id='$SGE_TASK_ID'; disp(id); exit"
matlab -nodisplay -singleCompThread -r "id=getenv('SGE_TASK_ID'); disp(id); exit"
```

- Distribute dependent jobs:

```
$ qsub  -N  job1  script1.sh
$ qsub  -N  job2  -hold_jid  job1  script2.sh
```

- ✓ Job 2 does not start until job1 ends.

Optimize Matlab codes

- Tools for measuring performance and optimizing codes
- Remove unnecessary work
- Optimize memory usage
- Use built-in functions/operators

Tools for measuring performance and optimizing codes

```
>> tic       %   Start measuring time
>> toc       %   End measuring time
>> timeit (function_name)    %  Measure time required to run function
>> mlint ('mfile_name')  %  Reports potential problems and opportunities for code optimization
>> profile on       %   Turn on profile (before the program starts).
>> profile viewer       %   View the results in the Profiler window (after the program ends).
```

Unnecessary work (1): redundant operations

- Avoid redundant operations in loops

bad

```
for i=1:N
    x = 10;
      .
      .
end
```

good

```
x = 10;
for i=1:N
      .
      .
end
```

Unnecessary work (2): reduce overhead

..from function calls

bad

```
function myfunc(i)
  % do stuff
end

for i=1:N
   myfunc(i);
end
```

good

```
function myfunc2(N)
   for i=1:N
      % do stuff
   end
end


myfunc2(N);
```

..from loops

bad

```
for i=1:N
   x(i) = i;
end
for i=1:N
   y(i) = rand();
end
```

good

```
for i=1:N
   x(i) = i;
   y(i) = rand();
end
```

Unnecessary work (3): logical tests

Avoid unnecessary logical tests...

...by using short-circuit logical operators

bad

```
if (i == 1 | j == 2) & k == 5
    % do something
end
```

good

```
if (i == 1 || j == 2) && k == 5
    % do something
end
```

...by moving known cases out of loops

bad

```
for i=1:N
    if i == 1
        % i=1 case
    else
        % i>1 case
    end
end
```

good

```
% i=1 case
for i=2:N
        % i>1 case
end
```

Unnecessary work (4): reorganize equations

Reorganize equations to use fewer or more efficient operators

Basic operators have different speeds:

```
Add          3- 6 cycles
Multiply     4- 8 cycles
Divide       32-45 cycles
Power, etc   (worse)
```

bad

```
c = 4;
for i=1:N
   x(i)=y(i)/c;
   v(i) = x(i) + x(i)^2 + x(i)^3;
   z(i) = log(x(i)) * log(y(i));
end
```

good

```
s = 1/4;
for i=1:N
   x(i) = y(i)*s;
   v(i) = x(i)*(1+x(i)*(1+x(i)));
   z(i) = log(x(i) + y(i));
end
```

Unnecessary work (5): avoid re-interpreting code

MATLAB improves performance by interpreting a program only once, unless you tell it to forget that work by running “clear all”

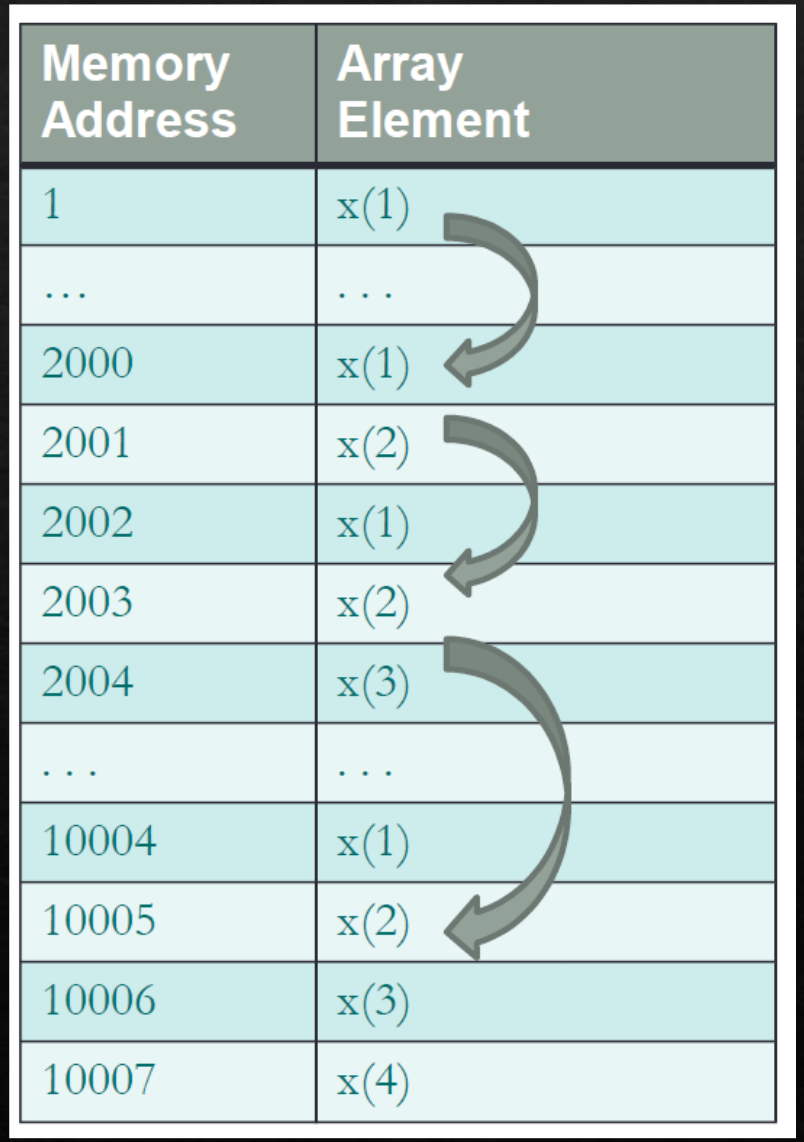
Value of `ItemType`	Items Cleared							
	Variables in scope	Scripts and functions	Class definitions	Persistent variables	MEX functions	Global variables	Import list	Java classes on the dynamic path
`all`	✓	✓		✓	✓	✓	From command prompt only	
`variables`	✓							

Memory (1): preallocate arrays

- Arrays are always allocated in contiguous address space.
- If an array changes size, and runs out of contiguous space, it must be moved. For example,

```
x = 1;
for i = 2:4
    x(i) = i;
end
```

- This can be very very bad for performance when variables become large.



Memory (1): preallocate arrays

- Preallocating array to its maximum size prevents intermediate array movement and copying.

```
A = zeros(n,m); % initialize A to 0
A(n,m) = 0;     % or touch largest element
```

- If maximum size is not known, estimate with upper bound. Remove unused memory after.

```
A=rand(100,100);
% . . .
% if final size is 60x40, remove unused portion
A(61:end,:)=[]; A(:,41:end)=[];  % delete
```

Memory (2): for-loop order

- It is faster to access continuous memory addresses than separated ones.
- Multidimensional arrays are stored in memory along columns (column-major).
- Best if inner-most loop is for array left-most index, etc.

bad

```
n=5000; x = zeros(n);
for i = 1:n          % rows
   for j = 1:n       % columns
     x(i,j) = i+(j-1)*n;
   end
end
```

good

```
n=5000; x = zeros(n);
for j = 1:n          % columns
   for i = 1:n       % rows
     x(i,j) = i+(j-1)*n;
   end
end
```

Memory (3): avoid unnecessary variables

- Avoid time needed to allocate and write data to main memory.
- Compute and save array in-place improves performance and reduces memory usage.

bad

```
x = rand(5000);
y = x.^2;
```

good

```
x = rand(5000);
x = x.^2;
```

Exercise 2

- Optimize this Matlab code to obtain a better performance.
- ✓ Hint: Use mlint to report potential problems and opportunities for code optimization.

```
n=5000;
for i=1:n
  for j=1:n
     a=3./2.;
     if (i==1)
       x(i, j) = 5.;
     else
       x(i, j) = i * log(2.) * log(a) + j^2 / 2.;
     end
  end
end
y=x.^2;
```

Matlab built-in functions/operators

- Some useful Matlab built-in functions/operators:

  Matrix operations: *, mtimes, inv, eig

  Solve linear equation: mldivide, linsolve, \

  Decomposition: lu, qr

  Optimization: fminsearch, fzero

- ✓ A full list:

  https://www.mathworks.com/help/matlab/functionlist.html#linear-algebra

- Built-in functions/operators are optimized and performs well in general.

Matrix multiplication

- The built-in operator * has been optimized and has much better performance.

```
n = 500
for j=1:1:n          % initialize data
    for i=1:1:n
        A(i,j)=i+j;   B(i,j)=2*i-j;
    end
end
C=zeros(n); D=zeros(n);
tic
for i=1:1:n          % mattix multiplication by loops
    for j=1:1:n
        C(i,j)= A(i,:)*B(:,j);
    end
end
toc
tic
D = A * B;    % mattix multiplication by built-in operator
toc
isequal(C,D)      % Check results
```

The backslash operator: solve linear equations

- Solve the linear system A*x = b

x = A\b

x = mldivide(A,b)

x = linsolve(A,b)

x = linsolve(A,b,opts)

```
A = triu(rand(5,5));      % random 5*5 up-triangle matrix
b = rand(5,1);            % random column array
opts.UT = true;           % up-triangle is ture
x = linsolve(A,b,opts)
x = A\b       % What does the backslash operater actually do behind the scene?
```

Decomposition

- LU decomposition

[L,U] = lu(A) % expresses a matrix A as the product of two essentially triangular matrices, one of them a permutation of a lower triangular matrix and the other an upper triangular matrix. The decomposition is often called the LU, or sometimes the LR, decompostion.

```
[L,U] = lu(A);  % obtain L and U matrices
y=L\b;
x=U\y   % These 3 lines together are equivalent to x=A\b
```

- QR decomposition

[Q,R] = qr(A) % expresses a matrix A into a product A = QR of an orthogonal matrix Q and an upper triangular matrix R.

```
[Q,R] = qr(A);  % obtain Q and R matrices
y=Q'*b;
x=R\y
```

Exercises 3

- Exercise 3.1: Create an n*n symmetric matrix A in one of the following ways:

  i) use built-in function *triu* (up-triangle matrix) and the matrix transpose operation ';

  ii) use control flow (*for, if, else,* …).

- Exercise 3.2: Given a symmetric matrix A from exercise 3.1, solve the linear algebra equation A*x=b using the following two methods:

  i) use the matrix inverse function *inv* .

  ii) use the backslash operator \ .

  Compare the computational time and numerical error of the two cases.

Further Information

- MathWorks Web:
  - ✓ MATLAB documentation: http://www.mathworks.com/help/matlab/

- BU Research Computing Services (RCS) Web:
  - ✓ MATLAB basics: http://www.bu.edu/tech/support/research/software-and-programming/common-languages/matlab/

- A book: *Accelerating MATLAB Performance: 1001 tips to speed up MATLAB programs* by *Yair M. Altman* .

- RCS help: help@scc.bu.edu , shaohao@bu.edu