MATLAB: BEYOND BASICS

Table of Contents

Tips and tricks, optimizations for speed, debugging, working with figures

NYU DATA SERVICES

- Tutorials and support for academic software
- One-on-one consultations by appointment

Accessing MATLAB

At NYU:

- Data Services Workstations - 5th floor of Bobst Library
- Most ITS labs
- Contact **hpc@nyu.edu** for license

At home:

- Virtual Computing Lab (vcl.nyu.edu, for students only)

- HPC Clusters (requires an account)

New Features

New Features

Optimizations for speed: preallocation

```
tic
    for k=1:10000 a(k,1) = k^2; end
toc
tic
    b = zeros(10000,1);
    for k=1:10000 b(k) = k^2; end
toc
clear a b k
```

Optimizations for speed: suppressing output

```
tic
    b = zeros(100,1);
    for k=1:100 b(k) = k^2; end
t1 = toc;
tic
    b = zeros(100,1);
    for k=1:100 b(k) = k^2, end
t2 = toc;
fprintf('\n%s%0.6f\n','Elapsed Time in Seconds: ',t1);
fprintf('\n%s%0.6f\n','Elapsed Time in Seconds: ',t2);
clear a b k t*
```

Optimizations for speed: variable types

```
a = randn(1e7,1); tic; o1 = cumsum(a); toc
b = cast(a,'single'); tic; o2 = cumsum(b); toc
format long
disp([o1(end); o2(end)])
disp(['Difference: ' num2str(o1(end)-o2(end))])
disp(['% Difference: ' num2str(100*(o1(end)-o2(end))/o1(end))])
format short
```

Linear indexing, multidimensional matrices

```
A = [1 3 7; 2 4 9]

A(1,3) == A(5)

[i1,i2] = ind2sub(size(A),5) % go from linear indices to subscripts
i3 = sub2ind(size(A),i1,i2)  % go from subscripts to linear indices
clear i*
```

```
[i1,i2] = ind2sub(size(A),[1 6]) % go from linear indices to subscripts
i3 = sub2ind(size(A),i1,i2)      % go from subscripts to linear indices
```

reshape() and permute()

```
p = phantom; % a 2D image
imagesc(p); colormap gray; xlabel('X'); ylabel('Y'); shg
```

For the sake of this exercise, start with a 2D image above and create a sinusoidal time modulation with added noise (100 points)

```
s = 1+0.2*sin(linspace(0,10*pi,100))'; % a sine wave
r = repmat(s,[1 size(p)]); % replicate to correct size:
pr = repmat(p,[1 1 size(r,1)]).*permute(r,[2 3 1]) + 0.2*rand([size(p) 100]);
implay(pr)
```

Look at the 1st slice

```
figure; imagesc(squeeze(pr(:,:,1))); colormap gray; xlabel('X'); ylabel('Y')
```

Look at the middle cross-section

```
figure; imagesc(squeeze(pr(:,end/2,:))); colormap gray; xlabel('Time');ylabel('Y')
```

Rearrange indices to go t-x-y instead of x-y-t

```
prp = permute(pr,[3 1 2]);
```

squeeze() is not necessary when averaging over the last index

```
figure; imagesc(mean(prp,3)'); colormap gray; xlabel('Time'); ylabel('X')

plot(mean(reshape(prp,100,[]),2)),shg % plot average intensity of the image
```

Optimizations for speed: logical indexing and find()

Find positive elements of a matrix

```
z=randn(3000);
tic
    z(z>0);
toc
whos z ans
tic
    z(find(z>0));
toc
whos z ans
```

Note that the output is not the same size as `z`

What if we want to simply replace the negative elements of z with zeros?

```
z.*z>0; %this is not correct because multiplication takes precedence over logicals
```

```
whos ans z
z.*(z>0);
whos ans z
```

Short Exercise

Write a function called `signs` that puts a `1` for every element of the matrix `A`, which is positive, `-1` for every element of `A` which is negative, `0`'s elsewhere.

```
edit signs
```

Finding extreme of values

- If your data isn't sorted, use `SORT` or `SORTROWS`
- `UNIQUE` is another function that is useful if you're looking for distinct values (returns sorted distinct values)
- `MAX` and `MIN` will give you values and indices
- `PRCTILE` will give you a cutoff point for a certain percentile
- If you're looking for top `N` values, `SORT`, then use `FIND(X,N,'first')`

```
clc,clear
load carsmall
pw = Horsepower./Weight;
% Find 5 car models with the highest power to weight ratio
tic;
pwi = sortrows([pw (1:100)'],-1); % sort by 1st column in descending order
pwi(isnan(pwi(:,1)),:) = []; % Get rid of NaNs if there are any
Model(pwi(1:5,2),:) % print top 5
toc
disp('---')
clear pwi
tic;
[pwi(:,1),pwi(:,2)] = sort(pw,'descend'); % sort by 1st column in descending order
pwi(isnan(pwi(:,1)),:) = []; % Get rid of NaNs if there are any
Model(pwi(1:5,2),:) % print top 5
toc
disp('---')
tic
Model(find(pw>prctile(pw,95),5,'first'),:)
toc
```

Optimizations for speed: Vectorization vs. Loops

Example: a simple computation of volumes of a cone for 100 cones Inputs a vector D which lists the diameters and a vector H which lists the heights

```
clear,clc
```

```
D=rand(1000,1); H=rand(1000,1);
% Naively, use a loop:
tic
for n = 1:1000
    V(n) = 1/12*pi*(D(n)^2)*H(n);
end
toc
% Now use vectorization and note the improvement in speed of computation:
tic
V = 1/12.*pi.*(D.^2).*H;
toc
```

More information online: Vectorization

Optimization for speed: Vectorization with repmat()

Consider the following problem: you have a matrix A and you want to normalize each row independently so that each row sums to 1. Without vectorization, you'd write something like the code below, where you sum up each element in a row in A, then divide each element in that row with that value.

```
clear,clc
A = rand(3, 4);
for i = 1 : size(A, 1)
    z = 0;  % sum all elements in the "i"th row
    for j = 1 : size(A, 2)
        z = z + A(i, j);
    end     % divide all elements in the "i"th row by the sum, z
    for j = 1 : size(A, 2)
        A(i, j) = A(i, j) / z;
    end
end
% This is a silly way write code in Matlab. It's much too verbose and slow.
% The vectorized version (no loops) would look something like this:
z = sum(A, 2);
A = A ./ repmat(z, [1 size(A,2)]);
% This requires fewer lines of code, is more concise, is arguably easier to read,
% and, most importantly, runs faster.
% Note that repmat(x, [1 3]) is equivalent to x(:, ones(3,1))
A = A ./ z(:, ones(size(A,2), 1));
```

Optimizations for speed: repmat() vs. (:,ones())

```
clear all; close all; clc
N = round(logspace(1,4.5,10));
for n = 1:length(N)
    a = 5000;
    tic; for k=1:500 a(ones(N(n),1),:); end; t1(n)=toc;
    tic; for k=1:500 repmat(a,N(n),1);  end; t2(n)=toc;
    s(n)=numel(ans)*8/1024;
end
plot(s,t1,s,t2,'g')
```

```
xlabel('Variable Size (Kb)')
ylabel('Time (sec)')
legend('(:,ones())','repmat()','Location','NorthWest');
```

Optimizations for speed: strfind() vs. findstr()

```
n2 = 1e6; % n2 is 1 million
a = 'a':'z'; % create a vector of 26 characters 'a' through 'z'
str = a(randi([1,26],[1,n2])); %create a 10^6 long string of characters 'a' to 'z'
str = reshape(str,1,[]); %make sure str is a row vector; unnecessary, in this case
pattern = a(randi([1,26],[1,5])); % create a 5 character pattern to find
tic; ix1 = strfind(str,pattern); toc
tic; ix2 = findstr(str,pattern); toc
all(ix1==ix2) % check that all indices found are exactly the same
```

Optimizations for speed: JIT accelerator

- Improves computation time for FOR loops
- Generates on-the-fly multithreaded code
- Continually improving (no need to specifically optimize code for it)

```
feature accel off
tic;
a = 2.5;
b = 3.33;
c = 6.23;
for i = 1:1e6
    a = b + c;
    b = a - c;
    c = b * a;
    if(a > b)
        a = b - c;
    else
        b = b + 12.3;
    end
end
toc
feature accel on
tic;
a = 2.5;
b = 3.33;
c = 6.23;
for i = 1:1e6
    a = b + c;
    b = a - c;
    c = b * a;
    if(a > b)
        a = b - c;
    else
        b = b + 12.3;
    end
```

```
end
toc
% Even though the code is identical, the JIT optimization in the background
% results in major speed improvement.
```

Other recommendations:

- Use **`var = load('A.mat')`** and not just **`load('A.mat')`**
- Use **`addpath()`** and **`fullfile()`** instead of changing directories with **`cd ..`**
- Minimize changing variable class: instead of **`x=1; x='hello'`** use **`x=1; xstr='hello'`**

Optimization for speed: Write your own functions

Sometimes you might be better off writing a more specific function that can be faster: Ex. 1

```
st1 = 0; st2 = pi; n = 10;
tic; x1 = linspace(st1,st2,n); toc
tic; x2 = st1:st2./(n-1):st2; toc
x1==x2
format long
[x1; x2]'
format short
```

Ex. 2

```
x = randn(1000,1); y = randn(1000,1);
% Matlab function
tic; a = corr(x,y); disp(a); toc
tic; b = nancorr(x,y); disp(b); toc
tic; c = nancov(x,y)/nanstd(x)/nanstd(y); disp(c(2)); toc
% edit nancorr
```

bsxfun()

```
M = rand(1000, 1000);
v = rand(1000,    1);
tic; c1 = bsxfun(@plus, M, v); toc
tic; c2 = M + repmat(v,1,1000); toc;
all(c1(:)==c2(:))
```

Elementwise operations are not always faster

```
n = 100;
[s,o] = deal(cell(n,1));
a = 'a':'z'; % create a vector of 26 characters 'a' through 'z'
% create a cell array of random strings
tic; for k=1:n s{k} = a(randi([1 26],[1 100000])); end; toc
% elementwise search for string 'x' vs. a FOR loop
tic; o = cellfun(@(x) strfind(s,'a'),s,'UniformOutput',false); toc
tic; for k = 1:n o{k} = strfind(s{k},'a'); end; toc
```

For more on anonymous functions: Anonymous Functions

Initialize multiple variables: deal()

```
[a,b] = deal(ones(4,3))

[c,d] = deal(rand)
% Alternatively,
rng('default'); c = rand;
rng('default'); d = rand;
% or
s = rng;
rng(s); c = rand;
rng(s); d = rand;
```

Scripting vs. Functions

- Scripts operate on variables in main MATLAB workspace
- Can be split into independently executable sections
- Functions have their own workspace; variables in those workspaces exist while the function is running
- Functions usually require inputs and outputs, but are a good way of encapsulating sections of code or making code reuseable
- Functions may be easier to debug

Debugging

```
edit db
```

Profiler - time your code

```
profile viewer
```

Optimization: my way or MATLAB way?

```
edit shiftleft
```

General Notes on Coding

- Write commennts
- Use proper indentation in scripts (Ctrl+i)
- Use meaningful variable names
- Be consistent, e.g. varA vs. var_a
- Do not abuse loops, think in terms of matrices

- Compartmentalize code via subfunctions in functions or sections in scripts
- Write code that can be generalized

Resources

Cody:

- Cody - MATLAB Central game that challenges and expands your knowledge of MATLAB

Webinars:

- Speeding up MATLAB Applications
- GPU Compiling with MATLAB
- Large Data Sets in MATLAB

Published with MATLAB® R2014b