CME 192: Introduction to MATLAB
Lecture 5

Stanford University

January 28, 2019

Outline

Review

Timing

Optimization
- Preallocation
- Vectorization
- Using in-built functions
- Memory Layout
- Summary

Profiling

Error Handling

Review

Lecture 4

- Plain text vs binary
- Saving and loading workspaces (binary)
- Comma Separate Values files (plain text)
- Delimited files (plain text), `dlmread`, `dlmwrite`
- Custom files (plain text), `fprintf`, `textscan`
- Java Script Object Notation (plain text), `jsondecode`
- Data Treatment
  - Interpolation
  - Filtering
  - Polynomial Fitting

Outline

Review

Timing

Optimization

- Preallocation
- Vectorization
- Using in-built functions
- Memory Layout
- Summary

Profiling

Error Handling

How to measure time?

- one way to measure time is since a given point
  - since the computer was turned on
  - since this program started
  - since January 1st 1970 (Unix computers)
- `now()` gives number of days since January 1st 0000 (2019 years ago)

```
>> now()
ans = 737452.777159401
>> now() / 365.25
ans = 2019.03566723481
```

Representing Date, `clock()`

`clock()`

- returns a vector, not a single number
- represents years, months, days, hours, minutes and seconds separately
- good accurracy, takes operating system time
- not great resolutions, but seconds have fractional values

```
date = clock()

years = date(1)

seconds = date(end)
```

```
date =
   2.0190e+03   1.0000e+00
   2.7000e+01   1.9000e+01
   0.0000e+00   5.7416e+01
years =  2019
seconds =  26.926
```

Timing Execution

tic and toc

- timers have resolution
- execution timing requires high resolution timers
- MATLAB provides the tic and toc pair
- general execution timing tips
  - try to time several runs and average
  - each loop run has a small overhead

```
A = rand(1e3, 1e3);

tic();
Ainv = pinv(A);
toc();

t = tic();
Ainv = pinv(A);
toc(t);

t = tic();
Ainv = pinv(A);
elapsed_s = toc(t)
```

```
Elapsed time is 1.3045 seconds.
Elapsed time is 1.2820 seconds.
elapsed_s =  1.2992
```

Outline

Review

Timing

Optimization
- Preallocation
- Vectorization
- Using in-built functions
- Memory Layout
- Summary

Profiling

Error Handling

Preallocation

- MATLAB arrays are resizable
- but memory regions aren't actually resizable
- each time an array is resized, MATLAB:
  - allocates a new, bigger memory area
  - copies old contents to the new memory area
  - deletes the old memory area
- MATLAB attempts to avoid doing that often by:
  - allocating more memory than strictly required
  - guessing how long the array's going to be

Dynamic Resizing

```
a = [];
a(1) = 2;
a(2) = 3;
a(3) = 5;
a(end + 1) = 7;

% missing is filled with zeros
a(end + 14) = 73;
```

Preallocation

```
a = zeros(1, 21);
a(1) = 2;
a(2) = 3;
a(3) = 5;
a(4) = 7;
a(end) = 73;
```

Vectorization Operations

- element-wise math operations
- element-wise in-built functions
- vector indexing
- logical indexing

```matlab
x1 = rand(1, 1e5);
x2 = rand(1, 1e5);

% element-wise math
y = x1 ./ x2;

% in-built functions
y = exp(x1 + x2);

% vector indexing
y = x1(1:2:end);
y = x2(1:floor(length(x2), 2));
y = x1;
y(2:2:end) = -x2(2:2:end);

% logical indexing
y = x1((x1 > 0.5) & (x1 < 0.75));
y(x2 > 0.2) = x2(x2 > 0.2);
```

Benefits of Vectorization

- speed-up (up to 100s times)
- parallelization
- shorter, cleaner, more readable code

In-Built Functions are Faster

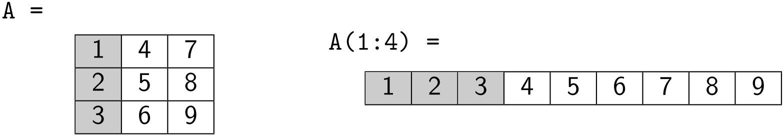
- search documentation for an existing function
- in-built functions are **compiled**
  - slower to write
  - difficult to read
  - **faster**
- user functions are dynamically interpreted
  - faster to write
  - easy to read
  - **slower**

```
function ax = my_abs(x)
  ax = x;
  x(x < 0.0) = -x(x < 0.0);
end
```

```
>> x = rand(1, 1e5) - 0.5;
>> y = my_abs(x); % slow
>> % vs in-built
>> y = abs(x); % much faster
```

Memory Layout

- all memory is laid out linearly
- MATLAB uses **column-major** order
- CPUs optimize accessing memory (vector entries) close to each other
  - CPU has a cache
  - each element access loads neighboring elements
  - if neighboring element is in cache, retrieval is **very** fast
- cache aware looping not that important in dynamic languages like MATLAB

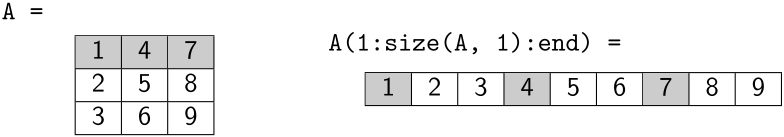


`A =`

1	4	7
2	5	8
3	6	9

`A(1:4) =`

1	**2**	**3**	4	5	6	7	8	9



`A =`

1	**4**	**7**
2	5	8
3	6	9

`A(1:size(A, 1):end) =`

1	2	3	**4**	5	6	**7**	8	9

Summary

Technique	Impact
Preallocation	Small
Vectorization	Large
Using in-built functions	Medium
Memory Layout	Negligible (in MATLAB)

Outline

Review

Timing

Optimization
- Preallocation
- Vectorization
- Using in-built functions
- Memory Layout
- Summary

Profiling

Error Handling

Profiling

- `profile` tool in MATLAB
- best way to optimize code is to determine which operations are time consuming
- profiling measures time spent in each function
- useful for finding bottlenecks

```
% turn on profiling
profile on
% <operations>
% ...
% <operations>
profile off

profile viewer % MATLAB only

info = profile('info');
profile clear

% use info data structure
info.FunctionTable.TotalTime
info.FunctionTable.FunctionName
```

Outline

Review

Timing

Optimization
- Preallocation
- Vectorization
- Using in-built functions
- Memory Layout
- Summary

Profiling

Error Handling

Displaying Errors/Warnings

Errors

- `error` prints an error and breaks execution immediately

```
function b = mat_mult(A, x)
  if size(A, 1) ~= length(x)
    error('Matrix dimensions do not match');
  end

  b = A * x; % matrix multiplication
end
```

Warnings

- `warning` prints an warning and continues with execution

```
function b = mat_mult(A, x)
  if size(A, 1) ~= length(x)
    warning('Matrix dimensions do not match. Returning x');
    b = x;
  else
    b = A * x; % matrix multiplication
  end
end
```

Catching/Handling Errors

- try, catch block
- attempt to do normal operations in the try block
- as soon as an error occurs, execution jumps to the catch block
- ME refers to the error
- try, catch blocks can be **nested**

```
1  a = zeros(1, randi(10));
2  try
3    % a might not be long enough
4    disp(a(6));
5  catch ME
6    warning('A is not long enough.
       Resizing...');
7    a = zeros(1, 6);
8  end
9
10 a
```

```
warning: A is not long enough. Resizing...
warning: called from
    test at line 7 column 5
a =

   0   0   0   0   0   0
```