MONASH
ENGINEERING
ENG1060

EFFICIENT CODING

Presented by Tony Vo

Slides by Tony Vo

EFFICIENT CODING

- When you start dealing with complex and large problems, speed is important
  - Make sure your code is correct and well documented before making it faster

- General rules for faster MATLAB code
  - Pre-allocate matrices
  - Avoid creating variables unnecessarily
  - Use functions for repeated code
  - Vectorise your code where possible
  - Suppress unimportant outputs

PRE-ALLOCATE MATRICES

- MATLAB will automatically resize a matrix if you assign a value to non-existent elements

```
>> A = 1:4
A =
     1     2     3     4

>> A(5) = 5
A =
     1     2     3     4     5
```

- Resizing a matrix takes time
  - Pre-allocate a matrix if you know it's final size

PRE-ALLOCATE MATRICES

- We often pre-allocate matrices with the zeros function
  - Complementary functions are length and size

```
% preallocate2.m
clc; clear all; close all;
tic

%setting up variables
time = [1:10; 11:20; 21:30; 31:40];
[r,c] = size(time);

%pre-allocating
speed = zeros(r,c);
```

```
%stepping through each time value
for i = 1:r
    for j = 1:c
        if time(i,c) < 30
            speed(i,c) = time(i,c).^2;
        elseif time(i,c) < 60
            speed(i,c) = time(i,c).^3;
        else
            speed(i,c) = time(i,c);
        end
    end
end
toc
```

THE DIFFERENCE PRE-ALLOCATION MAKES

- Pre-allocating can speed up the code by orders of magnitude!
  - Pre-allocated: Elapsed time is 0.000201 seconds.
  - No allocation: Elapsed time is 0.012240 seconds.

```
% no_preallocate.m
clc; clear all; close all;
tic

max_n = 6e4;
% A lot of values in my_matrix2
% my_matrix2(1) = 1^2; my_matrix2(2) = 2^2 etc...


for n = 1:max_n
    my_matrix2(n) = n^2;
end

toc
```

```
% preallocate.m
clc; clear all; close all;
tic

max_n = 6e4;
% A lot of values in my_matrix2
% my_matrix2(1) = 1^2; my_matrix2(2) = 2^2 etc...
my_matrix2 = zeros(1, max_n);

for n = 1:max_n
    my_matrix2(n) = n^2;
end

toc
```

AVOID CREATING VARIABLES UNNECESSARILY

- It takes time to create variables, especially large matrices
  - So, the fewer variables you create, the faster your code

- As with pre-allocation, this is important with large matrices

- Example:

```
>> A = ones(1e6,1);
>> B = A + 3;

>> % C is same as above
>> C = 4*ones(1e6,1);
```

USE FUNCTIONS FOR REPEATED CODE

- Functions facilitate code reusability and can make your code faster
- MATLAB will compile your functions and store them in memory for future use
  - This means that after calling a function once, every call to the same function will be faster than the first
- Example: SineTaylor() function from previous lectures
  - Function is recalled in a for loop for each term

$$\boxed{\sin(x) = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n+1)!} x^{2n+1}}$$

VECTORIZE YOUR CODE WHERE POSSIBLE

- Loops in MATLAB are very slow
  - Especially compared to C/C++ and Fortran
- MATLAB is designed to rapidly perform matrix-to-matrix operations
  - Hence its name MATrix LABoratory
- You can vectorise your code by using matrix operations to replace loops
  - Vectorisation takes a bit of thinking and planning before programming

VECTORIZE YOUR CODE WHERE POSSIBLE

- Examples
  - Computes the sine of t values ranging from 0 to 10
  - Computes the volume of a cone for D and H values

```
tic
i = 0;
for t = 0:0.01:10
    i = i + 1;
    y(i) = sin(t);
end
toc
```

```
tic
t = 0:0.01:10;
y = sin(t);
toc
```

```
tic
D=linspace(1,100,10000);
H=linspace(pi,40,10000);

for n = 1:length(D)
   V(n) = 1/12*pi*(D(n)^2)*H(n);
end
toc
```

```
tic

D=linspace(1,100,10000);
H=linspace(pi,40,10000);

V = 1/12*pi*(D.^2).*H;

toc
```

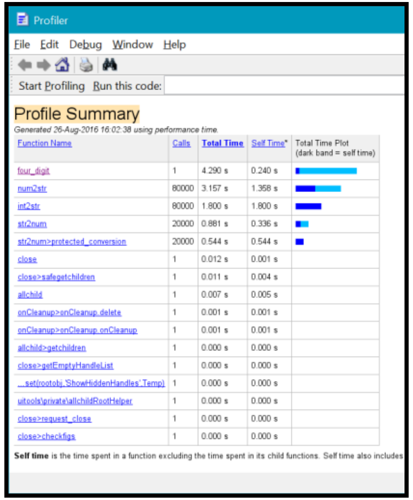
SUPPRESS OUTPUTS

- Frequent printing and plotting of outputs can significantly slow down the execution of your code
  - Ensure that you suppress everything and only print important information with fprintf
  - Only plot data that is useful

- Example: In a 4-for loop structure, suppressing 7 lines of code yielded almost 5 times increase in speed
  - Elapsed time is 11.189162 seconds. (unsuppressed code)
  - Elapsed time is 2.477280 seconds. (suppressed code)

TIMING CODE WITH THE MATLAB PROFILER

- Use the profiler to find sections of code that run slowly
  - Only do this if you NEED to make your program faster
  - Time invested in this task may not be worth it

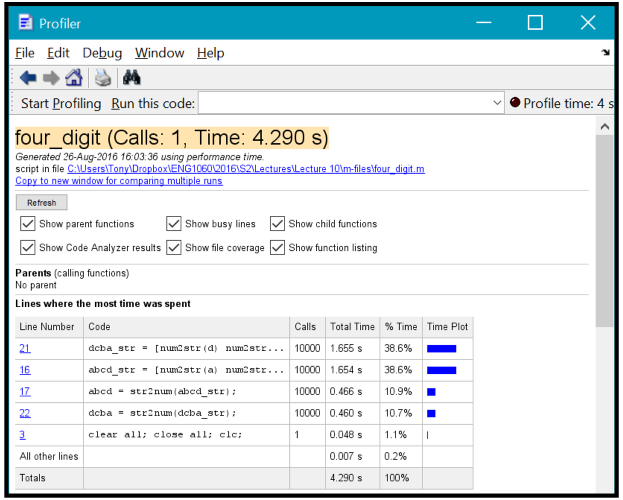


Profile Summary

Generated 26-Aug-2016 16:02:38 using performance time.

Function Name	Calls	Total Time	Self Time*	Total Time Plot (dark band = self time)
four_digit	1	4.290 s	0.240 s	■
num2str	80000	3.157 s	1.358 s	■
int2str	80000	1.800 s	1.800 s	■
str2num	20000	0.881 s	0.336 s	■
str2num>protected_conversion	20000	0.544 s	0.544 s	■
close	1	0.012 s	0.001 s	
close>safegetchildren	1	0.011 s	0.004 s	
allchild	1	0.007 s	0.005 s	
onCleanup>onCleanup.delete	1	0.001 s	0.001 s	
onCleanup>onCleanup.onCleanup	1	0.001 s	0.001 s	
allchild>getchildren	1	0.000 s	0.000 s	
close>getEmptyHandleList	1	0.000 s	0.000 s	
...set(rootobj,'ShowHiddenHandles',Temp)	1	0.000 s	0.000 s	
uitools\private\allchildRootHelper	1	0.000 s	0.000 s	
close>request_close	1	0.000 s	0.000 s	
close>checkfigs	1	0.000 s	0.000 s	

Self time is the time spent in a function excluding the time spent in its child functions. Self time also includes



four_digit (Calls: 1, Time: 4.290 s)

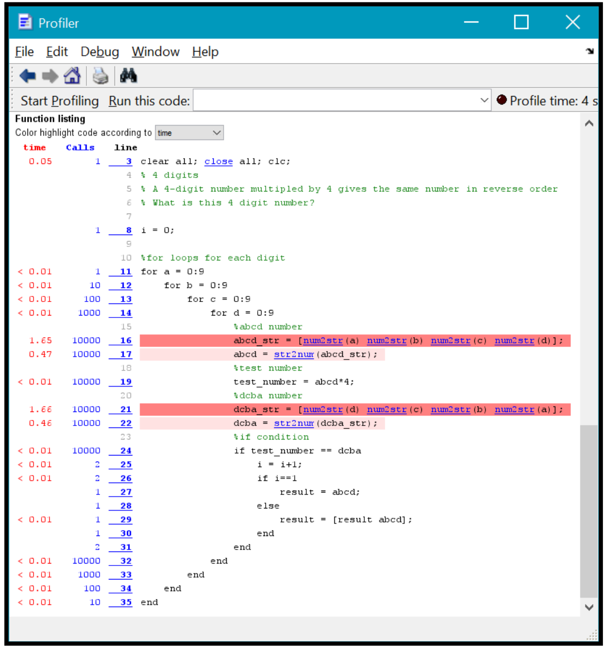
Generated 26-Aug-2016 16:03:36 using performance time.
script in file C:\Users\Tony\Dropbox\ENG1060\2016\S2\Lectures\Lecture 10\m-files\four_digit.m
Copy to new window for comparing multiple runs

☑ Show parent functions ☑ Show busy lines ☑ Show child functions
☑ Show Code Analyzer results ☑ Show file coverage ☑ Show function listing

Parents (calling functions)
No parent

Lines where the most time was spent

Line Number	Code	Calls	Total Time	% Time	Time Plot
21	`dcba_str = [num2str(d) num2str...`	10000	1.655 s	38.6%	■
16	`abcd_str = [num2str(a) num2str...`	10000	1.654 s	38.6%	■
17	`abcd = str2num(abcd_str);`	10000	0.466 s	10.9%	■
22	`dcba = str2num(dcba_str);`	10000	0.460 s	10.7%	■
3	`clear all; close all; clc;`	1	0.048 s	1.1%	■
All other lines			0.007 s	0.2%	
Totals			4.290 s	100%	



Function listing

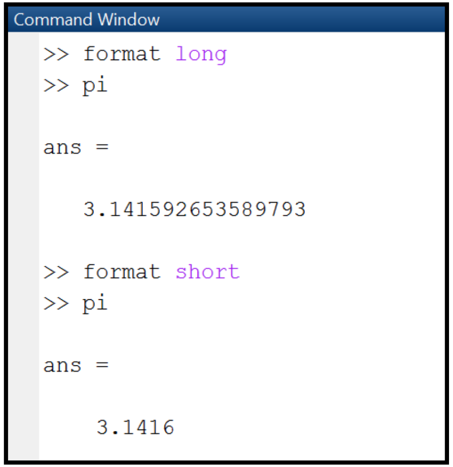
Color highlight code according to time

```
  time    Calls   line
  0.05        1    3  clear all; close all; clc;
                   4  % 4 digits
                   5  % A 4-digit number multipled by 4 gives the same number in reverse order
                   6  % What is this 4 digit number?
                   7
                  1    8  i = 0;
                   9
                  10  %for loops for each digit
< 0.01        1   11  for a = 0:9
< 0.01       10   12      for b = 0:9
< 0.01      100   13          for c = 0:9
< 0.01     1000   14              for d = 0:9
                  15                  %abcd number
  1.65    10000   16                  abcd_str = [num2str(a) num2str(b) num2str(c) num2str(d)];
  0.47    10000   17                  abcd = str2num(abcd_str);
                  18                  %test number
< 0.01    10000   19                  test_number = abcd*4;
                  20                  %dcba number
  1.66    10000   21                  dcba_str = [num2str(d) num2str(c) num2str(b) num2str(a)];
  0.46    10000   22                  dcba = str2num(dcba_str);
                  23                  %if condition
< 0.01    10000   24                  if test_number == dcba
< 0.01        2   25                      i = i+1;
< 0.01        2   26                      if i==1
              1   27                          result = abcd;
              1   28                      else
< 0.01        1   29                          result = [result abcd];
              1   30                      end
              2   31                  end
< 0.01    10000   32              end
< 0.01     1000   33          end
< 0.01      100   34      end
< 0.01       10   35  end
```

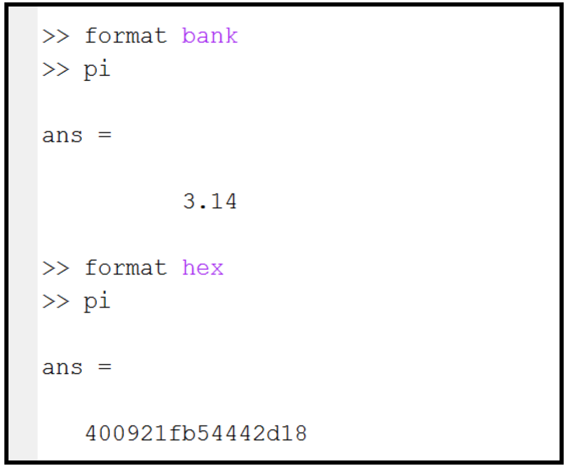
THE FORMAT COMMAND

- When dealing with real numbers that are very large or small, you may want to adjust the way MATLAB displays information
  - You can do this using the "format" command

- Number format examples:
  - format short/long
  - format bank
  - format hex



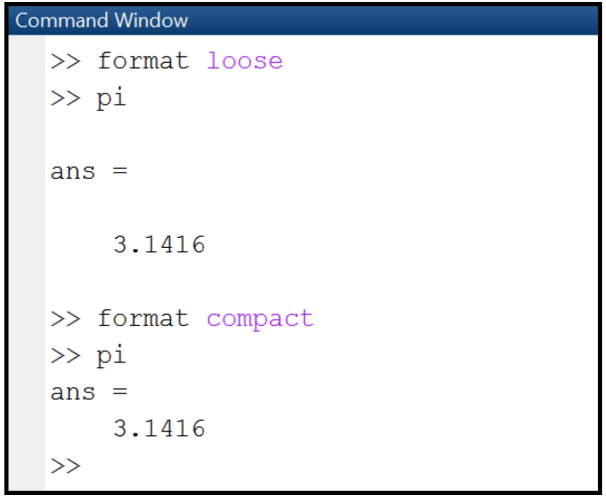
```
>> format long
>> pi

ans =

   3.141592653589793

>> format short
>> pi

ans =

    3.1416
```



```
>> format bank
>> pi

ans =

           3.14

>> format hex
>> pi

ans =

   400921fb54442d18
```

THE FORMAT COMMAND

- The format command can also adjust the line spacing
  - format compact/loose



```
>> format loose
>> pi

ans =

    3.1416

>> format compact
>> pi
ans =
    3.1416
>>
```

SUMMARY

- Efficient coding
- Pre-allocating, functions, vectorisation and suppression
- Formatting your output

- Should you try to replace all loops with vectorisation?