KU LEUVEN

MATLAB

performance

Correct code is always more important than speed

- If you find yourself puzzling over the code, or more importantly if you find yourself wondering if the code performs the correct calculation, then stop trying to optimize performance.
- Code that gives incorrect, or inaccurate results is useless, no matter how fast it executes.

Gerald Recktenwald *(http://web.cecs.pdx.edu/~gerry/MATLAB/programming/performance.html)*

Contents

- techniques to improve the performance of your MATLAB program
  - vectorisation
  - preallocation
  - other tips & tricks

Vectorisation

- Vectorization means converting for and while loops to equivalent vector or matrix operations.
- vectors and matrices are the basic data types used in MATLAB , try to exploit it.
- use vector and/or matrices instead of for or while loops
- example

```
i=0;
for t=0:0.1:10
    i=i+1;
    y(i)=sin(t);
end
```

more efficient

```
t=0:0.1:10;
    y=sin(t);
```

- *for_loop.m*

Vectorisation

- Using the for loop in *MATLAB* is relatively expensive. It is much more efficient to perform the same task using the vector method.
  example:

```
for j=1:n
    for i=1:m
        A(i,j) = B(i,j) + C(i,j);
    end
end
```

  can be more compactly and efficiently represented by the vector method as follows:

```
A = B + C;
```

- For sufficiently large matrix operations, this method is superior in performance
- *File: loop_or_vectorial.m*

Array Operations

Array Operations vs. Matrix Operations

- Use array operations to replace loops that perform only simple arithmetic on scalar data.

```
for n = 1:100
  V(n) = 1/12*pi*(D(n)^2)*H(n);
end
```

Perform the vectorized calculation

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

- difference is the use of the .* and ./ operators.
- differentiate array operators (element-by-element operators) from the matrix operators (linear algebra operators), * and /.

Logical Array Indexing

- logical array indexing: the index parameter is a logical matrix that is the same size as an array and contains only 0's and 1's.
- The array elements selected have a '1' in the corresponding position of the logical indexing matrix.

```
D = [-0.2 1.0 1.5 3.0 -1.0 4.2 3.14];
D >= 0
ans=
0 1 1 1 0 1 1
```

- select the subset of V for which the corresponding elements of D are nonnegative.

```
Vgood = V(D>=0);
```

- vectorized Boolean operators, **`any`** and **`all`**, which perform Boolean AND and OR functions over a vector.

```
if all(D < 0)
   warning('All values of diameter are negative.');
   return;
end
```


Preallocating Arrays

- for and while loops that incrementally increase the size of a data structure each time through the loop can adversely affect performance and memory use.
- Advice: improve on code execution time by preallocating the maximum amount of space that would be required for the array ahead of time.

```
x = 0;
for k = 2:1000
   x(k) = x(k-1) + 5;
end
```

Preallocate a 1-by-1000 block of memory for x initialized to zero.

```
x = zeros(1, 1000);
for k = 2:1000
   x(k) = x(k-1) + 5;
end
```

Preallocation

- Avoids overhead of dynamic resizing idea: allocate memory, in order to eliminate the search for continuous memory blocks
- Reduces memory fragmentation
- Use array appropriate preallocation
- example: *alloc_ex_1.m*

Array Type	Function	Examples
Numeric array	`zeros`	`y = zeros(1, 100);`

Cell array	`cell`	`B = cell(2, 3);`
		`B{1,3} = 1:3;`
		`B{2,2} = 'string';`

Structure array	`struct, repmat`	`data = repmat(struct('x',[1 3],...` `'y',[5 6]), 1, 3);`

More tips

- Matlab stores column-wise
  - **Demo:** scroll_by_row.m
  - **Demo:** scroll_by_column.m
- Suppress output
  - The obvious killer is output to the screen: use `;` to suppress output and minimize graphical output
  - Test memory requirements for a simple plot
  - **Demo:** plot_memoryreq.m

Scripts & functions

Use functions, not scripts

- Scripts are always read and executed one line at a time (interpreted). No matter how many times
- you execute the same script, MATLAB must spend time parsing your syntax. By contrast, functions are effectively parsed into memory when called for the first time or modified. Subsequent invocations skip the interpretation step.
- As a rule of thumb, scripts should be called only from the command line, and they should themselves call only functions, not other scripts.

Memory Usage

- MATLAB functions available for memory management:
  - **`clear`**: clear all variables from the workspace, liberating memory
  - **`pack`**: extended MATLAB-sessions can fragment memory. Fragmented memory usually means enough meory but not in continuous blocks. MATLAB Out of memory can be cured with *pack* : free up memory without deleting variables
    saves existing variables to disk and reloads then continguously
  - **`quit`**: stop
  - **`save`**: fastest way to save data
  - **`load`**: fastest way to read data
  - **`whos`**: shows how much memory has been allocated for variables in workspace

Summary

- profiler
  - use it also to understand a lengthy program (debugging tool)
  - points at the time-consuming code
- timing
  - tic toc
  - cputime
- techniques to improve the performance of your MATLAB program
  - vectorisation: try to use it as much as possible
  - preallocation
  - functions vs scripts
  - ...

References

- Performance

https://nl.mathworks.com/help/matlab/matlab_prog/techniques-for-improving-performance.html

- Memory management

https://nl.mathworks.com/help/matlab/performance-and-memory.html