Matlab DOs and DON’Ts

Dmitri Nikonov

“the unrepentant Matlab programmer”

Dmitri.e.Nikonov@intel.com
Outline

Matlab positioning among simulation tools
Tools: profiler, interrupt debug, windiff
Tips: loops vs. vectorization
Tips: vector function calls
Tips: sparse matrices
Tips: self-made vs. built-in
Program structuring: modularization, encapsulation, memory management
My philosophy on Matlab
Matlab positioning

Perceived as a tool for small calculations ("homework in a graduate course")

In fact can handle most complex simulation projects

On par with C – structures and classes, function overloading, inheritance, GIU capabilities and many more – not the topic of this lecture

Benefits compared to C – natural use of complex numbers, unique operations with matrices, powerful library of functions, efficient built-in numerical algorithms (need to be left to professionals), integrated plotting

Can be the tool of choice for scientific simulations

Matlab suffers from improper use (esp. C or Fortran habits) – the topic of this lecture
Tools: profiler

<table>
<thead>
<tr>
<th>Parent functions:</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>sparesolu</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Child functions:</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>none</td>
<td></td>
</tr>
</tbody>
</table>

100% of the total time in this function was spent on the following lines:

```matlab
78:  % Create a sparse matrix from its diagonals
0.0000067 0% 79:  B = arg1;
80:  d = arg2(:,);
81:  p = length(d);
0.0000000 0% 82:  moda = (margin == 3);
83:  if moda
90:  % Process A in compact form
0.0100000 11% 91:  [i,j,a] = find(A);
0.0003260 0% 92:  a = [i j a];
0.0100000 11% 93:  [m,n] = size(A);
94:  for k = 1:p
```

Times the run
Enables you to find the bottlenecks in the program
See which function called which and how many times
Invoked as follows

```matlab
profile on
chloop = 'slow';
simatr = 1000;
a = rand(simatr,simatr);
b = rand(simatr,simatr);
c = suexpo(a,simatr,chloop);
profile report
```
Tools: interrupt debug

Invoked by placing a line
**keyboard**
at some point in the Matlab program
Allows you to examine the workspace of the function normally not available,
Execute any Matlab commands to find what goes wrong

**K>> return**
To go back to execution

**K>> dbquit**
To interrupt execution
Tools: windiff

Part of Microsoft Visual Studio
Extremely useful to track changes, look for bugs
Loops

function c = multi(a,b,simatr,chloop)
switch chloop
    case 'slow'
        for j = 1:simatr
            for k = 1:simatr
                c(j,k) = a(j,k)*b(j,k);
            end
        end
    case 'fast'
        c = a.*b;
end

SLOW 139s

FAST 0.1s

You shall not use a “for”-loop when you can use a vectorized operation
Function call

function c = suexpo(a,simatr,chloop)
switch chloop
    case 'slow'
        for j = 1:simatr
            for k = 1:simatr
                c(j,k) = exp(a(j,k));
            end
        end
    case 'fast'
        c = exp(a);
end

You shall not call a function element-by-element when you can call a function of a matrix
function sparesolu(simatr,chloop)
    di1 = rand(simatr,1);
    di2 = rand(simatr,1);
    di3 = rand(simatr,1);
    b = rand(simatr,1);
    switch chloop
        case 'slow'
            a = zeros(simatr,simatr);
            di1red = di1(1:end-1);
            di3red = di3(2:end);
            a = sparse(diag(di2)) + sparse(diag(di1red,-1)) + sparse(diag(di3red,1));
            c = a\b;
        case 'fast'
            a = speye(simatr,simatr);
            % starting with an empty matrix does not work for some reason
            ditot = [di1 di2 di3];
            a = spdiags(ditot,-1:1,a);
            c = a\b;
        end
    end

Sparse matrices

SLOW 12s

You shall not form a full matrix and then try to convert to a sparse one

FAST 0.1s
Recursive algorithms

Function ‘tridiag’ copied with adaptation from “Numerical Recipes” book

function u = tridag(a,b,c,r,n)
bet=b(1);
  u(1)=r(1)/bet;
  for j=2:n % decomposition and forward substitution
    gam(j)=c(j-1)/bet;
    bet=b(j)-a(j)*gam(j);
    u(j)=(r(j)-a(j)*u(j-1))/bet;
  end
  for j=n-1:-1:1 % backsubstitution
    u(j)=u(j)-gam(j+1)*u(j+1);
  end

You shall not use “clever” algorithms which cannot be vectorized
Program structure: modularization

Make any piece of code which performs a separate task into a function. It will be easy to improve and re-use it.
Do not use M-scripts except for the “main” module – scripts do not let you control input and output variables.
In any case, do not use a module longer than 100 lines – it is impossible to improve later.

You shall break your program in many small functions.
Program structure: documentation

Write many more comments than you consider sufficient.
>> help block2full will show the set of lines just after the ‘function’.

function Fmat = block2full(Bdmat,Np,Nc,ndiag)
% function Fmat = block2full(Bdmat,Np,Nc,ndiag)
% convert a block-diagonal form of matrix to full one
% Bdmat = matrix in the band-diagonal form
% Fmat = matrix in the full form
% Np = number of blocks/slices
% Nc = number of elements in a block/slice
% ndiag = index of a block-diagonal,
% =0 -- main, <0 -- lower, >0 upper diagonals
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Lmat = Np-abs(ndiag);                   % number of blocks on the diagonal
Fmat = zeros(Np*Nc,Np*Nc);         % size of the full matrix
for k=1:Lmat
    jna = 1 + (k-1)*Nc;                     % beginning horizontal index for a block
    if(ndiag<0)
        jna = jna - ndiag*Nc;             % correct if lower diagonals
    end
    jko = jna + Nc - 1;                     % ending horizontal index
    ina = jna + ndiag*Nc;               % beginning vertical index
    iko = jko + ndiag*Nc;               % ending vertical index
    Fmat(jna:jko,ina:iko) = Bdmat(:,;,:);   % fill in the block
end

You shall comment every line of your code

Really diligent programmers insert authors’ name, version history, copyright
Program structure: variable scope

The benefit of the function – tight control of input and output arguments. You avoid a lot of errors with unintended modification of data in other pieces of your code.

Encapsulation: No variable with the same name defined in other parts of your (or somebody else’s) code can be confused with local variables. They are only defined in the scope of this function. ‘global’ variables defeat this useful feature. Do not use to pass input and output arguments.

Avoid them in most cases !!! Exceptions:
a) global constants. You do not want to assign them in many parts of your code and chase all of them to change a value
b) a library function that calls a function

\[
t, y_{\text{evol}} = \text{ode113('urav', [t_{\text{sta}}, t_{\text{fin}}], y, opa)};
\]

and you need to change the names and number of arguments passed to the lower level function (‘urav’ in this case)

You shall not use global variables with few necessary exceptions
Memory management

Three type of memory a program may use
  • on chip cache
  • RAM
  • hard drive

In the order of decreasing speed but increasing capacity

RAM storage is fast enough. If your variable space does not fit in RAM, then the program will spool on hard drive. It makes execution EXTREMELY SLOW.

Get as much RAM for your computer as feasible.

Check the variable space with

>> whos

If the matrices do not fit in RAM, re-do your algorithm, free memory with a command in the code, e.g.

clear MyBigMat

or make your simulation less ambitious (less variables).

You shall not use array size that does not fit in you RAM
My philosophy on Matlab

I admit that optimized C will be faster than optimized Matlab

But Matlab will save you on development, maintenance, and re-use time

If you write Matlab in the style of C, performance will be horrible

**Need to use unique Matlab features to optimize the program**

Do not forget general rules of good programming: structuring your program, control scope of variables, manage memory