Matlab Programming Guidelines

Joan Solà
LAAS-CNRS

April 2, 2009

Contents

1 Matlab Help

Prepare your help headers to look really Matlab-like!

```
% FUN One line description with one space between % and FUN.
%    FUN(X,Y) Longer description, with explanation of function
%    inputs X and Y and the output. There are 4 spaces between
%    % and FUN(). The function name is in CAPITAL LETTERS.
%    Preferably, the input variables X and Y are also in
%    capital letters.
%
%    If the paragraph above is too complex, break it into
%    different paragraphs.
```

```
%
%    If the list of input arguments is too complex, make a
%    list here. Explain ALL input arguments. The list is
%    indented another 4 spaces:
%        X:    one Bourbon
%        Y:    one Scotch
%
%    FUN(X,Y,Z) explain extra inputs Z here and what they do.
%    Explain if they have a default value. If you need to
%    make a new list, remember the 4 spaces!
%        Z:    one beer.
%
%    [out, OUT_x, OUT_y] = FUN(...) returns the Jacobians
%    wrt X and Y. Maybe you have to explain something else.
%    You do not need to repeat the input parameters so you
%    can use the form [out, OUT_x] = FUN(...), with the (...).
%
%    Before saving, select entire paragraphs and do RIGHT
%    CLICK, "Wrap selected comments". This equals all line
%    lengths to approximately the page width.
%
%    See also FUN2, FUN3. Use it exactly like this, "See also "
%    + function names in CAPITAL LETTERS. Matlab parses this line
%    and will create links to the functions' helps ONLY IF YOU
%    FOLLOW THESE GUIDELINE STRICTLY.

%    (c) 2009 You @ LAAS-CNRS. Make yourself famous. See that
%    this comment line is disconnected from the Help body (the
%    previous line has no % sign).
```

2 Code readability

2.1 Aligned code reads well!

1. When using consecutive lines of code, try to vertically align all EQUAL signs. Examples:

```
% GOOD: code reads easy
x        = f(y);
variable = fun(z);
JAC_x    = JAC_y*Y_x;

% BAD: code is a pack
x = f(y);
variable = fun(z);
JAC_x = JAC_y*Y_x;
```

2. Similarly, when commenting multiple lines on the right margin, align comments. Examples:

```
% GOOD: comments read well
x        = f(y);         % these lines
variable = fun(z);       % are all easy
JAC_x    = JAC_y*Y_x;    % to read

% BAD: comments are packed within the code
x        = f(y); %these lines
variable = fun(z); % are not easy
JAC_x    = JAC_y*Y_x; % to read
```

3. Exceptions are accepted, but use common sense. Examples

```
% GOOD: all possible alignments coincide
x        = f(y);             % these comments are aligned
variable = g(z);             % with the fourth line.
JAC_x    = JAC_y*Y_x + Z_a*A_variable*VARIABLE_x; % Oops!
output   = JAC_x*P*JAC_x';   % this defines the alignment above.
extra    = I*dont*know;      % over all it is easy to read.

% NOT SO GOOD, BUT OK: alignments come in groups
x        = f(y);      % these comments are NOT aligned
variable = g(z);      % with the fourth and fifth lines.
JAC_x    = JAC_y*Y_x + Z_a*A_variable*VARIABLE_x; % Oops!
output   = JAC_x*P*JAC_x';   % this margin is new
extra    = I*dont*know;      % over all it is easy to read.
```

4. Still, you can try to align consecutive groups of lines. Example

```
x        = f(y);             % these comments aligned,
variable = g(z);             % and the alignment
output   = JAC_x*P*JAC_x';   % continues in next group

y        = 4;                % this follows the same alignment
extra    = 5*eye(3);         % over all it is easy to read.
```

2.2 Line grouping and commenting

1. Comment every group of lines performing a coherent action before the group. Example:

```
% get idps to delete
used    = [Lmk.used];
idps    = strcmp({Lmk.type},'idpPnt');
drawn   = (strcmp((get([MapFig.estLmk.ellipse],'visible')),'on'))';
delIdps = drawn & idps & ¬used;
```

2. Comment individual lines on the right if more info is needed. Example:

```
% get idps to delete
used    = [Lmk.used];                   % used lmks
idps    = strcmp({Lmk.type},'idpPnt'); % inverse−depth landmks
delIdps = drawn & idps & ¬used;         % to be deleted
```

3. Separate small groups of lines with an empty line so that the code does not look packed. As a rule, no more than 4 lines should go together.

4. Before saving the function, do CNTRL+A, CNTRL+I to make all the indents look nice.

2.3 Line breaking "..."

Make exceptional use of line breaking "...", particularly when functions have long names or many long parameters:

```
[out, OUT_x, OUT_y, OUT_z, OUT_par, OUT_calibration] = ...
    functionNameThatMightBeVeryLong(...
        Lmk.state.x,...           % you can put
        Sen(4).par.y,...          % comments here
        Obs(sen,lmk).nom.N,...    % if necessary
        Sen(4).par.k,...          % to explain the
        Sen(4).par.cal);          % input data
```

See **userData.m**, **createMapFig.m** to see examples of this.

3 Names of variables

For convention, we are going to do the following:

1. Variables inside functions have short names in small letters normally.
2. Robot, sensor, landmark etc INDICES are always **rob**, **sen**, **lmk**: For example,

```
Rob(rob).rob      = rob;
Obs(sen,lmk).sen = sen;
```

3. Robot, sensor, landmark etc IDENTIFIERS are **rid**, **sid**, **lid**. For example,

```
Rob(rob).id       = rid;
Obs(sen,lmk).sid = Sen(sen).id;
```

4. Jacobians are **BIG_small**, where **Y_x = dy/dx**.
5. Jacobians are not **Yx**, better **Y_x**.

4 Jacobians and the chain rule

Systematically make use of the chain rule when constructing Jacobians. While MAPLE code may be faster to compute in some cases, the chain rule permits a modular organization and a better comprehension of the code. Both features are crucial in a toolbox because they allow us to modify parts of the code without compromising the rest.

Follow these guidelines:

1. Name all Jacobians as specified in the previous section, that is, if **y = f(x)** then **Y_x = dy/dx**
2. Build functions returning output variable and optional Jacobians. Here is an example:

```
function [z, Z_x, Z_y] = f(x, y)

z = sin(x-y);          % this is the output value
if nargout > 1         % Jacobians requested
    Z_x =  cos(x-y);     % this is  dz/dx
    Z_y = -cos(x-y);     % this is  dz/dy
end
```

3. Use the chain rule for functions using other functions. Keep the Jacobians optional. Example:

```
function [q, Q_a, Q_b, Q_c] = g(a, b, c)

if nargout == 1       % No Jacobians requested
    q = a + f(b,c);
else                  % Jacobians requested
    [p, P_b, P_c] = f(b, c); % This uses function f() above.
    q = a + p;
    n = length(a);           % This is plain code
    Q_a = eye(n);            % and Jacobians are
    Q_p = eye(n);            % computed directly

    Q_b = Q_p*P_b;           % This is the chain rule
    Q_c = Q_p*P_c;           % to compose Jacobians.
end
```

4. Observe how the chain rule 'chains' Jacobians by matching leading and trailing name parts. The leading and trainling parts of the whole chain define the resulting Jacobian name. Examples:

```
LEAD_trail    = LEAD_x * X_trail ;
FOURTH_first  = FOURTH_third * THIRD_first ;
FOURTH_second = FOURTH_third * THIRD_second ;
```

5. Beware of the possibility of long chains and multi-path chains. Examples

```
Z_w = Z_y * Y_x * X_w;         % a chain of three elements
D_a = D_b * B_a + D_c * C_a;   % a chain with two paths
```

5 Vectorizing structure arrays

1. Use vectorization to obtain arrays. Examples:

```
% 3 logical vectors
used   = [Lmk.used];
vis    = [Obs.vis];
drawn  = (strcmp((get([MapFig.estLmk.ellipse],'visible')),'on'))';
```

```
% a numeric vector of IDs
lmkIds = [Lmk.id];
```

2. If the field you want to access is a string, try this

```
idps = strcmp({Lmk.type},'idpPnt')  % a logical vector
```

3. Operate with the logicals to get new logicals. Example:

```
erase    = ¬vis & drawn;
usedIdps = used & idps;
```

4. When setting logicals individually, always use **`true`**/**`false`**, not **`1`**/**`0`**:

```
Obs(1).vis = true;   % Do not use 1 instead of true, otherwise
Obs(2).vis = false;  % you turn the whole vector to numeric.
```

5. You can access an array directly with the logical vector

```
Lmk(used)   % all the Lmk's that are used
```

6. You can get the indices with **`FIND`**

```
usedIdx = find(used);
```

7. You can also access an array with indices, of course:

```
Lmk(usedIdx)    % this is equivalent to Lmk(used)
```

8. If you want the first N unused **`Lmk`**'s, do for example

```
Lmk(find(¬used,N,'first'))
```

or, easier to read:

```
notUsed = find(¬[Lmk.used]);
Lmk(notUsed(1:N));
```

6 Error messages

Be kind to your fellows and stick to Matlab standards:

```
    error('??? Unknown sensor type ''%s''.',Sen(sen).type)
```

gives a 'nice' Matlab error message (the second line is ours!):

```
    ??? Error using ==> createSensors at 46
    ??? Unknown sensor type 'pinPole'.

    Error in ==> createSLAMstructures at 10
    Sen = createSensors(Sensor);

    Error in ==> universalSlam at 36
    [Rob,Sen,Lmk,Obs,Tim] = createSLAMstructures(...
```

This error information is enough. Matlab has debugging mechanisms to go find further info for the error.