Matlab Programming Guidelines

Joan Solà
LAAS-CNRS

March 27, 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, brake 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 "
%    + func. 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 Names of variables and Jacobians.

For convention, we are going to do the following:

1. Variables inside functions have short names in small letters normally.

2. Jacobians are `BIG_small`, where `X_y = dx/dy`.

3. Jacobians are not `Xy`, better `X_y`.

4. Robot, landmark, sensor etc INDICES are always rob, sen, lmk, obs.

5. Robot, landmark, sensor etc IDENTIFIERS are rid, sid, lid, etc.

3 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));
   ```

4 Code readability I: 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.
```

5 Code readability II: 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'); % lmks that are inverse-depth
drawn   = (strcmp((get([MapFig.estLmk.ellipse],'visible')),'on'))'; % previously drawn
delIdps = drawn & idps & ~used;        % these need to be deleted
```

3. Separate all 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.

6 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.

7 Error messages.

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.