How To Make Pretty Figures With Matlab

Damiano Varagnolo

January, 15th, 2010   -   Version 0.8

Contents

1 Introduction

Take a look at figures 1 and 2 (pages 2 and 2), and think which one is more professional. If your answer has been *the second* and you would like to learn how to produce fancy figures using Matlab, then continue reading. This guide wants to show you that:

- it is possible to modify *every single property* of Matlab figures using the command-line;

- the process of modify figures using the command-line is really time-inexpensive (once you have understood how);

- the process of understanding how to modify figures using the command-line is as fast as reading this guide;

- once finished reading this guide (i.e. understanding how to modify figures using the command-line) you will have lots of templates that will help you saving **enormous** amount of time and producing fancy figures.
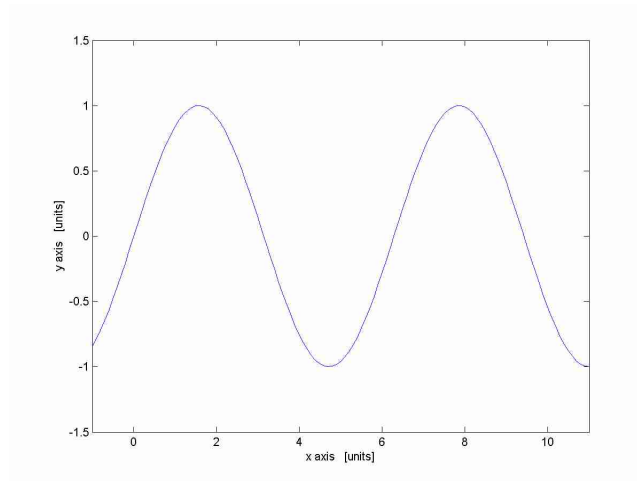

Figure 1: Generic figure with the following problems: • it uses much more space than necessary; • it is colored; • it uses wrong fonts.
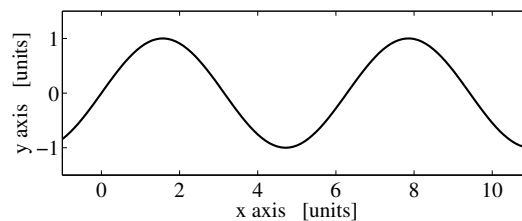

Figure 2: Generic figure without the problems of figure 1.

1.1   Hints on how to format figures

Dimensions

Obviously figures should be "as big as needed". The problem is that "needed" is not an objective feeling, and this imply that there is no exact rule.

Fonts

Try to use the same font of the document in which the figure will be included. It is preferred if the font you use for the figure is the same font you use for the normal text.

Files extensions

It is better to use vectorial file formats (as .pdf or .eps). Try to minimize the usage of .jpeg since it will introduce artifacts (please note the "noise" in figure 1. If you do not notice it please magnify this file and note the different behavior of the 2 figures).

Colors, kind and width of the lines

Lines should be sufficiently thick and should not be colored. Take care that people **really** prefer printing in black and white (very cheaper). If your figures use colors and they are printed in b/w, they could not be intellegible. Also not-so-thick black lines could be not intellegible if printed in economy mode. Be careful when using markers: they must not hide informations (i.e. be careful they do not cover important pieces of the graph).

Captions and labels

Write all the times the dimensional units of the various axes, and what they mean. Note: for standalone figures captions should be sufficiently long in order to make them "understandable by themselves".

1.2 Hints on how to make source code

Keep the code producing the data and the code producing the *final* figures separate (figures for debugging purposes can be placed everywhere). This thing should be done by these steps:

1. for each results plot you want to make, save a separate *.mat* file[1] with a self-explaining name[2];

2. for each plot you want to make produce a *.m* file that loads the correspondent *.mat* file, creates the plot and saves without requiring any user action.

Note that this strategy is useful both when you are working in a team (so people do not get crazy with the data-producing code of other people) and also when you know that you will have to use the figures in the future (so *you* will not get crazy with the data-producing code of *yourself*).

1.3 Hints on how to manage files

It is useful to keep separate folders for:

- *.m* files (if you have lots of *.m* think at the possibility of also separating the data-producing files from the figures-producing ones);

- *.mat* files;

- graphical files.

The usefulness increases monotonically with the complexity of your work.

2 How to set figure's dimensions

In this section we show which is the code in order to change the dimension of a single figure. For understanding the results, compare the size of figures 1 and 2.

NOTE: Matlab uses 2 different references systems for dimensioning and positioning the figures: one for the screen and one for the "printing" (i.e. when you save the figure on file). If you modify the position and dimension of the figure on the screen, then you save it, you will have file figure different from the one you were looking at the screen. Code example:

---

[1]It would be better to save it in *.txt* extension, so it can be loaded by other programs like Gnuplot; anyway here we omit this procedure.

[2]Well, names of 3, 4 characters can save 7, 8 bytes in your hard-disk and 1, 2 seconds in typing it. Consider the fact that small names can make you or your collaborator thinking for several minutes at the question "what the hell is this?", then decide which is the best name you have to use.

```
% we set the units of the measures used through the file
%
% [ inches | centimeters | normalized | points | {pixels} | characters ]
set(gcf, 'Units', 'centimeters');



% we set the position and dimension of the figure ON THE SCREEN
%
% NOTE: measurement units refer to the previous settings!
afFigurePosition = [1 1 20 5.5];          % [pos_x pos_y width_x width_y]
set(gcf, 'Position', afFigurePosition);  % [left bottom width height]



% we link the dimension of the figure ON THE PAPER in such a way that
% it is equal to the dimension on the screen
%
% ATTENTION: if PaperPositionMode is not 'auto' the saved file
% could have different dimensions from the one shown on the screen!
set(gcf, 'PaperPositionMode', 'auto');



% in order to make matlab to do not "cut" latex-interpreted axes labels
set(gca, 'Units','normalized',... %
 'Position',[0.15 0.2 0.75 0.7]);
```

3   How to set axes

Axes have lots of properties; in the following example we show not all of them, but the most used
ones. The various options are self-explaining and have comments indicating what they are. If you
want to know more about them, search on the Matlab's guide. Note: this code has to be replied
for each figure (i.e. each time you make a new figure). Code example:

```
% general properties
iFontSize           = 20;
strFontUnit         = 'points'; % [{points} | normalized | inches | centimeters | pixels]
strFontName         = 'Times';  % [Times | Courier | ]               TODO complete the list
strFontWeight       = 'normal'; % [light | {normal} | demi | bold]
strFontAngle        = 'normal'; % [{normal} | italic | oblique]     ps: only for axes
strInterpreter      = 'latex';  % [{tex} | latex]
fLineWidth          = 2.0;       % width of the line of the axes



% note: it is quite difficult to use the "latex" interpreter for the ticks;
% if absolutely needed google for "format_ticks.m" by Alexander Hayes
set(gca,                         ...
... 'Position',              [1 1 20 10],     ... TODO
... 'OuterPosition',         [1 1 20 10],     ... TODO
    ...
    'XGrid',                 'on',            ... [on | {off}]
    'YGrid',                 'on',            ... [on | {off}]
    'GridLineStyle',         ':',             ... [- | -- | {:} | -. | none]
    'XMinorGrid',            'off' ,          ... [on | {off}]
    'YMinorGrid',            'off',           ... [on | {off}]
    'MinorGridLineStyle',    ':',             ... [- | -- | {:} | -. | none]
    ...
    'XTick',                 0:10:100,        ... ticks of x axis
    'YTick',                 0:1:10,          ... ticks of y axis
```

```
        'XTickLabel',           {'-1','0','1'}, ...
        'YTickLabel',           {'-1','0','1'}, ...
        'XMinorTick',           'off' ,         ... [on | {off}]
        'YMinorTick',           'off',          ... [on | {off}]
        'TickDir',              'out',          ... [{in} | out] inside or outside (for 2D)
        'TickLength',           [.02 .02],      ... length of the ticks
        ...
        'XColor',               [.1 .1 .1],     ... color of x axis
        'YColor',               [.1 .1 .1],     ... color of y axis
        'XAxisLocation',        'bottom',       ... where labels have to be printed [top | {bottom}]
        'YAxisLocation',        'left',         ... where labels have to be printed [left | {right}]
        'XDir',                 'normal',       ... axis increasement direction [{normal} | reverse]
        'YDir',                 'normal',       ... axis increasement direction [{normal} | reverse]
        'XLim',                 [0 100],        ... limits for the x-axis
        'YLim',                 [-10 10],       ... limits for the y-axis
        ...
        'FontName',             strFontName,    ... kind of fonts of labels
        'FontSize',             iFontSize,      ... size of fonts of labels
        'FontUnits',            strFontUnit,    ... units of the size of fonts
        'FontWeight',           strFontWeight,  ... weight of fonts of labels
        'FontAngle',            strFontAngle,   ... inclination of fonts of labels
        ...
        'LineWidth',            fLineWidth);    %   width of the line of the axes
```

4   How to set axes labels

Axes labels inherits the properties of the text objects. Notice that when using the "latex" interpreter
sometimes the x-axis label is "cutted", so it is needed to call explicitly to resize the dimension of
the figure inside the window. Code example:

```
% fonts properties
iFontSize      = 20;
strFontUnit    = 'points'; % [{points} | normalized | inches | centimeters | pixels]
strFontName    = 'Times';  % [Times | Courier | ]                TODO complete the list
strFontWeight  = 'normal'; % [light | {normal} | demi | bold]
strFontAngle   = 'normal'; % [{normal} | italic | oblique]     ps: only for axes
strInterpreter = 'latex';  % [{tex} | latex]
%
strXLabel      = 'label of x axis';
strYLabel      = 'label of y axis';
%
fXLabelRotation = 0.0;
fYLabelRotation = 90.0;




xlabel( strXLabel,                      ...
        'FontName',     strFontName,    ...
        'FontUnit',     strFontUnit,    ...
        'FontSize',     iFontSize,      ...
        'FontWeight',   strFontWeight,  ...
        'Interpreter',  strInterpreter);
%
ylabel( strYLabel,                      ...
        'FontName',     strFontName,    ...
        'FontUnit',     strFontUnit,    ...
```

```
            'FontSize',     iFontSize,        ...
            'FontWeight',   strFontWeight,    ...
            'Interpreter',  strInterpreter);
%
set(get(gca, 'XLabel'), 'Rotation', fXLabelRotation);
set(get(gca, 'YLabel'), 'Rotation', fYLabelRotation);



% in order to make matlab to do not "cut" latex-interpreted axes labels
set(gca, 'Units',    'normalized', ...
         'Position', [0.15 0.2 0.75 0.7]);
```

5   How to set plots properties

Actually Matlab can make lots of kinds of plots; here we report the code for the most general plot, i.e. 2D cartesian plots. In the future we will complete the list. Code example:

```
% we set the plots properties
%
% notes:
% - each property is actually an array of properties;
%
% line styles:  [{-} | -- | : | -.]
% marker types: [+ | o | * | . | x | square | diamond | > | ...
%               ... < | ^ | v | pentagram | hexagram | {none}]
%
% -- lines
afPlotLineWidth         = [2.0, 1.3];
astrPlotLineStyle       = [{'-.'}, {':'}]; % NOTE: do not insert '.-' but '-.'
aafPlotLineColor        = [[0.1 0.1 0.1] ; [0.2 0.2 0.2]]; % RGB
%
% -- markers
aiPlotMarkerSize        = [25, 21]; % in points
astrPlotMarkerType      = [{'.'}, {'x'}];
aafPlotMarkerFaceColor  = [[0.1 0.1 0.1] ; [0.2 0.2 0.2]]; % RGB
aafPlotMarkerEdgeColor  = [[0.3 0.3 0.3] ; [0.4 0.4 0.4]]; % RGB



% we want to plot several curves
hold on;
%
% here you can plot several plots (each can have its properties)
% note: it is convenient to store the handles to the single plots
% in order to easily manage legends
%
handleToPlot1 = plot(                           ...
    afX1,                                       ...
    afY1,                                       ...
    'LineStyle',       astrPlotLineStyle{1},    ...
    'LineWidth',       afPlotLineWidth(1),      ...
    'Color',           aafPlotLineColor(1,:),   ...
    'Marker',          astrPlotMarkerType{1},   ...
    'MarkerSize',      aiPlotMarkerSize(1),     ...
    'MarkerFaceColor', aafPlotMarkerFaceColor(1,:), ...
    'MarkerEdgeColor', aafPlotMarkerEdgeColor(1,:));
%
```

```
handleToPlot2 = plot(                                 ...
    afX2,                                             ...
    afY2,                                             ...
    'LineStyle',       astrPlotLineStyle{2},          ...
    'LineWidth',       afPlotLineWidth(2),            ...
    'Color',           aafPlotLineColor(2,:),         ...
    'Marker',          astrPlotMarkerType{2},         ...
    'MarkerSize',      aiPlotMarkerSize(2),           ...
    'MarkerFaceColor', aafPlotMarkerFaceColor(2,:), ...
    'MarkerEdgeColor', aafPlotMarkerEdgeColor(2,:));
```

6 How to set markers

If you have to plot markers on your file you can choose some parameters in order to change their appearance. You can refer to code example in section 5.

7 How to plot annotations

Annotations are positioned by Matlab using the "figure" measurement units, i.e. the reference system is not the one induced by the axes plotted inside the figure, but the one induced by the figure itself (i.e. the pixel at the left-bottom corner is the zero of the coordinate system). It means that if we want to plot a double arrow starting in $(1, 2)$ and finishing in $(8, 4)$ and the points refer to axes coordinate system, we have to find calculate how to change the references. In the following code example steps *(p1)* and *(p2)* are dedicated to this change:

```
% annotation type
%
% --------------------------------------------------------------------------
% |      kind of annotation   |              required arguments           |
% --------------------------------------------------------------------------
% |  line arrow doublearrow   |       starting_point ending_point         |
% | textbox ellipse rectangle | starting_point ending_point width height |
% --------------------------------------------------------------------------
strAnnotationType = 'line';
%
% arguments
afStartingPoint = [1, 2];
afEndingPoint   = [8, 4];
fWidth          = 10;      % could be not necessary
fHeight         = 3;       % could be not necessary


% (p1): we obtain the position of the axes inside the figure
%
% we use normalized units of measure (i.e. figure reference system has coordinates
% between 0 and 1)
set(gcf, 'Units', 'normalized');
%
afXAxisLimits              = get(gca, 'XLim');
afYAxisLimits              = get(gca, 'YLim');
%
afAxesDimensionsAndPositions = get(gca, 'Position');
fXAxisPosition             = afAxesDimensionsAndPositions(1);
fYAxisPosition             = afAxesDimensionsAndPositions(2);
fXAxisLength               = afAxesDimensionsAndPositions(3);
fYAxisLength               = afAxesDimensionsAndPositions(4);
```

```
fXonYAxesRatio                  = fXAxisLength / fYAxisLength;
%
afFigurePosition                = get(gcf, 'Position'); % [left bottom width height]
fXonYDimensionRatio             = afFigurePosition(3) / afFigurePosition(4);



% (p2): convert the axes measurement units into the figure measurement units
afStartingPoint_FU(1) =  ( afStartingPoint(1) - afXAxisLimits(1) )
                      / ( afXAxisLimits(2) - afXAxisLimits(1) )
                      * fXAxisLength + fXAxisPosition;
afStartingPoint_FU(2) =  ( afStartingPoint(2) - afYAxisLimits(1) )
                      / ( afYAxisLimits(2) - afYAxisLimits(1) )
                      * fYAxisLength + fYAxisPosition;
afEndingPoint_FU(1)   =  ( afEndingPoint(1)   - afXAxisLimits(1) )
                      / ( afXAxisLimits(2) - afXAxisLimits(1) )
                      * fXAxisLength + fXAxisPosition;
afEndingPoint_FU(2)   =  ( afEndingPoint(2)   - afYAxisLimits(1) )
                      / ( afYAxisLimits(2) - afYAxisLimits(1) )
                      * fYAxisLength + fYAxisPosition;



% we plot the annotation
handleToAnnotation1 = annotation(                   ...
... handleToFigure,                                 ... uncomment if necessary
    strAnnotationType,                              ...
    [afStartingPoint_FU(1) afEndingPoint_FU(1)],    ...
    [afStartingPoint_FU(2) afEndingPoint_FU(2)]); %... uncomment if necessary
%   fWidth,                                         ... uncomment if necessary
%   fHeight);                                          % uncomment if necessary
```

8   How to plot labels

Labels are quite easy to be plotted; note that you can easily change font size / format / dimension by command line. Code example:

```
% fonts properties
iFontSize      = 20;
strFontUnit    = 'points'; % [{points} | normalized | inches | centimeters | pixels]
strFontName    = 'Times';  % [Times | Courier | ]                TODO complete the list
strFontWeight  = 'normal'; % [light | {normal} | demi | bold]
strFontAngle   = 'normal'; % [{normal} | italic | oblique]     ps: only for axes
strInterpreter = 'latex';  % [{tex} | latex | none]



% note: the xPosition and yPosition are referred in the "axes" units of measure
text(   xPosition,                  ...
        yPosition,                  ...
        'this is the text plotted', ...
        'FontName',    strFontName,    ...
        'FontSize',    iFontSize,      ...
        'FontWeight',  strFontWeight,  ...
        'Interpreter', strInterpreter);
```

9   How to set legends

Legends work via handles to the lines previously plotted. Note also that legends inherit some properties from the setting of axes (see Section 3). Code example:

```
% we set the legend properties; note:
%
%
% Locations could be [N S E W {NE} NW SE SW NO SO EO WO NEO NWO SEO SWO B BO]
% where the characters have the following meanings:
% - N = North
% - S = South
% - E = East
% - W = West
% - O = Outside the plot
% - B = Best (least conflict with data in plot)
% OR you can also specify an 1-by-4 position vector ([left bottom width height])
%
%
% The following properties are inherited when specified in "set(gca, ...)":
% - FontName
% - FontSize
% - FontUnits
% - FontWeight
% - FontAngle
% - LineWidth
%
atArrayOfHandlesToLines = [handleToFirstLine; handleToSecondLine; handleToThirdLine];
astrArrayOfLabels       = [{'first line'}; {'second line'}; {'third line'}];
%
strLegendLocation       = 'Best';    % combinations of N S E W B O or a vector
strLegendOrientation    = 'vertical'; % [{vertical} horizontal]
%
afEdgeColor             = [0.0 0.0 0.0]; % RGB
afTextColor             = [0.0 0.0 0.0]; % RGB
%
strInterpreter          = 'latex';   % [{tex} | latex | none]
%
%
legend( atArrayOfHandlesToLines,                    ...
        astrArrayOfLabels,                          ...
        'Location',            strLegendLocation,    ...
        'Orientation',         strLegendOrientation, ...
        'Box',                 'off',                ... [{on} off]
        'Color',               'none',               ... none => transparent
        'EdgeColor',           afEdgeColor,          ...
        'TextColor',           afTextColor,          ...
        'Interpreter',         strInterpreter);
```

10   How to set the background of the figure

This section needs to be improved!! Current code example:

```
% TODO
%
% we select the output color
afFigureBackgroundColor = [0.3, 0.1, 0.1];
%
```

```
set(gcf, 'color', afFigureBackgroundColor);
set(gcf, 'InvertHardCopy', 'off');
```

11   How to set default properties

TO IMPROVE

```
set(0, 'DefaultAxesLineStyleOrder', '-|--|:');
set(0, 'DefaultAxesColorOrder', [0.0 0.0 0.0; 0.4 0.4 0.4; 0.6 0.6 0.6]);
```

12   How to save figures

Matlab can save figures in various formats, from *.ps* to *.png*. We will consider here the case you are working with LaTeX, so the only files you need are *.pdf* or *.eps*[3]. Code example:

```
% here we select which output file extension we want
bPrintOnFile_Pdf    = 1; % [0 (false)   1 (true)]
bPrintOnFile_Eps    = 1; % [0 (false)   1 (true)]


% we select the file path
%
% NOTE: do NOT insert extensions!
%strFilePath = '../images/my_figure';
strFilePath = 'c:\\my_figure';


% we select the printing resolution
iResolution = 150;


% we select to crop or not the figure
bCropTheFigure = 1; % [0 (false)   1 (true)]


% ATTENTION: if PaperPositionMode is not 'auto' the saved file
% could have different dimensions from the one shown on the screen!
set(gcf, 'PaperPositionMode', 'auto');


% saving on file: requires some checks
if( bPrintOnFile_Pdf || bPrintOnFile_Eps )
    %
    % NOTE: if you want a .pdf with encapsulated fonts you need to save an
    % .eps and then convert it => it is always necessary to produce the .eps
    %
    % if we want to crop the figure we do it
    if( bCropTheFigure )
        print('-depsc2', sprintf('-r%d', iResolution), strcat(strFilePath, '.eps'));
    else
        print('-depsc2', '-loose', sprintf('-r%d', iResolution), strcat(strFilePath, '.eps'));
    end;
    %
```

---

[3]Please note that *latex* compiler behaves worse than *pdflatex* compiler when including graphics. We **strongly** recommend you to use *pdflatex* compiler.

```
    % if we want the .pdf we produce it
    if( bPrintOnFile_Pdf )
        %
        % here we convert the .eps encapsulating the fonts
        system(                                                  ...
          sprintf(                                               ...
            'epstopdf --gsopt=-dPDFSETTINGS=/prepress --outfile=%s.pdf  %s.eps',  ...
            strFilePath,                                         ...
            strFilePath));
        %
    end;
    %
    % if we do not want the .eps we remove it
    if( ~bPrintOnFile_Eps )
        delete(sprintf('%s.eps', strFilePath));
    end;
    %
end;% saving on file
```

12.1   How to save figures silently

It could be useful to produce and save figures but do not show them on the monitor. In this case the figure has to start with the code:

```
figure( 'Visible', 'off' );
```

13   How to send emails from Matlab

It is convenient to generate an identificative string of the message you want to receive from Matlab (for example the current date). Please note that it is required to configure the Matlab's mailserver.

```
tParameters.bEnableMailAlert = true;
tParameters.strCurrentDate   = GenerateCurrentDateString;


if( tParameters.bEnableMailAlert )
    %
    setpref('Internet', 'SMTP_Server', 'mail.dei.unipd.it');
    setpref('Internet', 'E_mail', 'my.email@dei.unipd.it');
    %
end;%


if( tParameters.bEnableMailAlert )
    %
    sendmail(                                                   ...
        'my.email@dei.unipd.it',                               ...
        'Alert Message from Matlab: simulation concluded',     ...
        sprintf('Simulation started on %s is just terminated', tParameters.strCurrentDate) );
    %
end;%
```