sublimeFigure Manual

M. Sich, the University of Sheffield, m.sich@sheffield.ac.uk

Source: https://github.com/maxsich/sublimeFigure

Contents

Why sublimeFigure? 1
Layout 2
Presets 5
Properties 4
Basic Properties 4
Font sizes 4
Padding (margin) control 5
Colour bar ..... **Error! Bookmark not defined.**
Methods 3
Helpful Tips 6
Further Examples 6
Multi-column & multi-row subplots with tight margins 6
Multi-column & multi-row subplots with sparse margins 8
Legal Bit 10

Why *sublimeFigure*?

MATLAB® is great for data processing, and has a wealth of data plotting, imaging functions. However, when it comes to producing figures, which would be suitable for presentations or publications it falls short with its default settings. The problem is usually with huge margins, which are hard to control, especially with subplots.

The aim of *sublimeFigure* project is to create a way to plot figures quickly in MATLAB®, which immediately would be suitable for publications or presentations.

sublimeFigure is built around the idea that the user sets figure size needed, and margins (or padding) between subplots. The rest is calculated automatically to fill the maximum space.

For example, the code below will produce a figure with a single plot of the right size for a one-column figure in an APS publication:

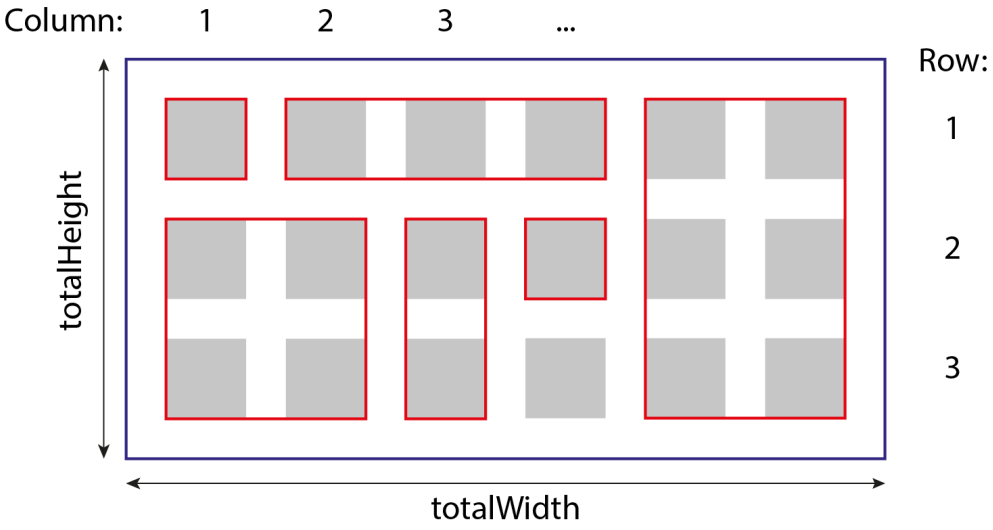
```
f = sublimeFigure;
f.subplot(1,1);
plot(x,y);
print( 'test', '-dpdf', '-r1200' );
```

sublimeFigure is realised as a class utilising the lately introduced object-oriented functionality of MATLAB®, and, therefore, requires the r2017a to function.

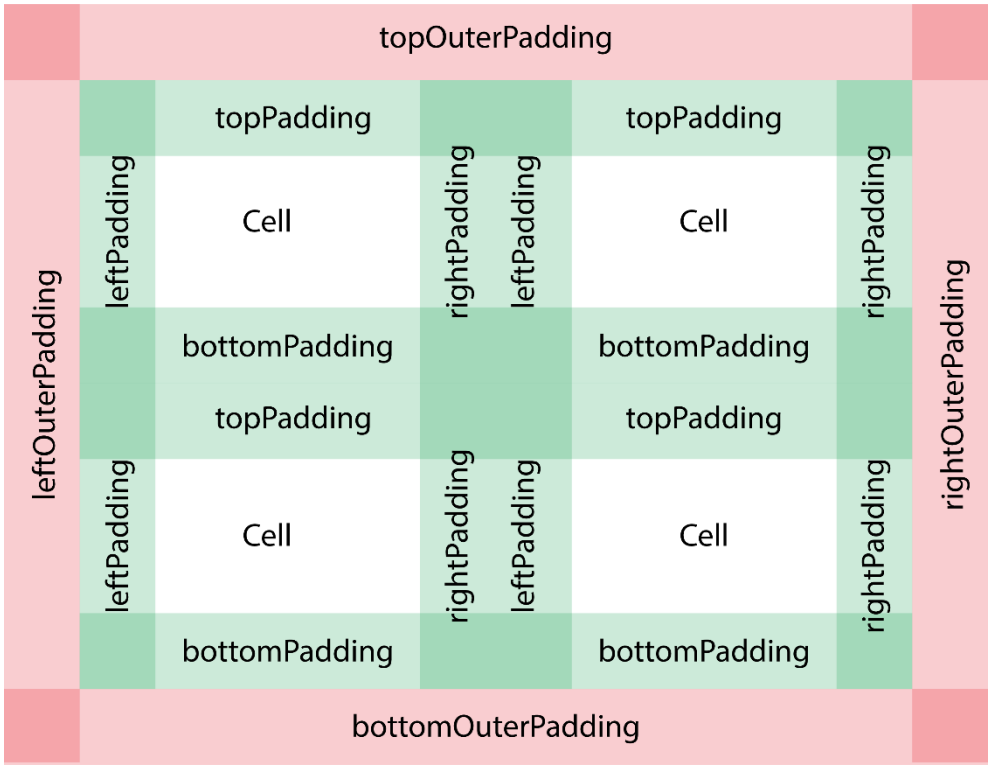
Layout

sublimeFigure uses the same idea of a 2D matrix of cells, as MATLAB®, with a set number or rows and columns. A subplot can be in any cell or can span over several row, columns, or both. With *sublimeFigure* you can precisely control margins in each row and column, as well, as overall figure margins.

Figure below shows an example of a grid with 6 columns and 3 rows. Grey areas are the areas where the data is to be plotted and white spaces around are margins left for axis labels and tick labels. Red rectangles show some possible examples of spanning subplots over several grid cells.



There are two types of margins, called paddings, in the code: ‘outer’ and ‘normal’. Sketch below shows how these are implemented. Note that by default, all of the subplots have the same corresponding top, bottom, left, and right paddings. Outer paddings (margins) are always nonnegative scalars, and define space outside all subplot cell matrix. This space can be used to add extra annotations, such as colour bars or text, or it can be used as an extra white space.



Methods

function obj = **sublimeFigure**(varargin)

This function creates a sublimeFigure object in memory (which in turn contains one figure). At this point in takes only one optional argument – a preset, described in Presets.

Once you create the object you can change any of its properties obj.prop = new_value. Which means you can dynamically change number of columns or rows of the grid and any other parameter, even if you have already created subplots with data.

function [ax, axID] = **subPlot**(cCol, cRow, varargin)

By default there are no subplots inside the figure, so if you would like to plot anything, you need to create a subplot. Even if it is just one graph, this would be still required (as ax = f.subplot(1,1)), unlike typically in MATLAB®. The function takes the following arguments:

cCol – column number in the grid (must be less or equal than the total number of columns currently set).

cRow – same as with columns.

Two optional arguments are *cSpan* and *rSpa*n (in this order) to allow the subplot to span over several grid cells. If you only want the subplot to span over several rows use 1 as *cSpan* value: obj.subPlot(1,1,1,2), for example.

Note that the function returns two arguments:

ax – is a handle to the axis, which was created (empty) by the function. After this you can use all the usual plotting functions. You can store this handle separately if you wish to return to this axis after creating another subplot.

axID – is an integer number which is an ID of the axis within the current sublimeFigure object. Usually this is an ordinal number of the subplot. You will need this number in order to plot.

function lb = **label**(axID, location, str)

Adds a neat label in one of the corners of the subplot.

location can be one of the following four options: 'topleft', 'topright', 'bottomleft', and 'bottomright'.

str is a string of the label, such as '(a)' or 'A' depending on the publication style.

lb is a handle to the text object of the label. You can use it to change any of the properties of the label once it is created, such as colour, font, or size.

function cb = **colorbar**(axID)

Creates a neat colour bar on the right side of the subplot (given by axID). The function return a handle to the colour bar. At this time, there are no parameters other than axID, to customise the size or span of the colour bar; these are planned in the future releases.

Properties

Basic Properties

totalWidth – total width of the figure. Default is 8.6 cm, which is the typical width of a single column in a two-column publication.

totalHeight – total height of the figure. Default is 8.6 cm.

numColumns – number of columns of the cell matrix. Default is 1.

numRows – number of rows of the cell matrix. Default is 1.

Font sizes

There are two properties of the class, which affect font sizes of the figure: *defFontSize* and *enableFontControl*. The class does not change the default units, which are points. A typical publication would use a smaller font than the standard 10pt of MATLAB®. On the other hand, for presentations you might want to have it larger, increased to 14pt or 16pt.

defFontSize – changes the default size of the figure font. Default is 8pt. Takes effect only if *enableFontControl* is set to *true* (which is default).

enableFontControl – when set to *true* (default), resets default font size of all MATLAB graphics to *defFontSize* and reverts back when the figure is close (the figure object is deleted from memory).

Warning: *sublimeFigure* uses *groot*'s (graphics root) *DefaultAxesFontSize* property to control font size, because after when *plot* or other plotting function (such as *imagesc* is called) the font size of the axis is reset to the system default values (it is unfortunately a bug from MATLAB®). So, if you have several *sublimeFigure* objects at the same time, and each of those has *enableFontControl* set to *true*, the last created (or set) font size will take effect on all new subplots in all figures. In this case, it is better to individually set font size for each plot after plotting:

```
f = sublimeFigure;
f.enableFontControl = false;
ax = f.subplot(1,1);
plot( ax, ____ );    % or any other plotting function
ax.FontSize = 8;
```

Padding (margin) control

Outer paddings (margins) are always nonnegative scalars, and define space outside all subplot cell matrix. This space can be used to add extra annotations, such as colour bars or text, or it can be used as an extra white space.

leftOuterPadding – default is 1.1 cm.

rightOuterPadding – default is 0.1 cm.

topOuterPadding – default is 0.1 cm.

bottomOuterPadding – default is 1.0 cm.

leftPadding – default is 0.

rightPadding – default is 0.

topPadding – default is 0.

bottomPadding – default is 0.

Presets

Presets are very handy since they remove the need for you to specify paddings and get the type of figure right away when calling the object constructor. If you use a preset, it overrides the default values for paddings shown above. Currently the following presets are available:

presentation	changes font sizes to 16pt and increases the size of the figure to 15cm x 12cm, which should work well for the majority of presentation software.
tight	ideal for multi-plot figures where *x* and *y* scales are the same for different subplot, so that there is no need to repeat tick labels and axis labels for each subplot, but instead to maximise the area used by the data.
sparse	

See Further Examples to see how these presets work.

More presets will be added in the future.

Helpful Tips

If you are preparing a figure for a TeX file it is best to use .pdf as the output format of the figure instead of .ps or .eps since .pdf if the latest version of Post Script and supports more features such as transparency. *sublimeFigure* automatically adjusts paper sizes so that when you run a command like this you get file ready to be used in your publication:

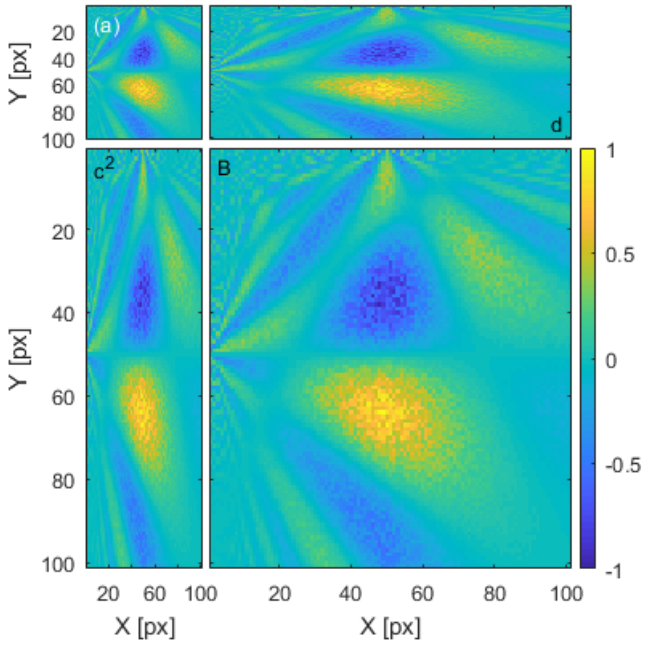
```
print( 'test', '-dpdf', '-r1200' );
```

Further Examples

Multi-column & multi-row subplots with tight margins

```
% Load a sample data set
load( 'sampleData.mat', 'data');

% Create a sublimeFigure object using a 'tightmulti' preset
f = sublimeFigure( 'tight' );
f.numRows = 4;
f.numColumns = 4;
f.rightOuterPadding = 1; % Leave extra space outside to plot a
colour bar

% Create a sub-plot in the top left corner and return axes %
handle and subplot ID number within the sublimeFigure.
[ax, id] = f.subPlot(1,1); % Sub plot occupies one cell
imagesc( data );
```

```
% Create a label and return a handle
lbl = f.label( id, 'topleft', '(a)');
% Use handle to label, which is text object, to change colour
lbl.Color = 'w';
% Remove tick labels on x axis
ax.XTickLabel = [];
ylabel( 'Y [px]' );

% Create a subplot spanning over 3 rows and 3 columnns
[ax, id] = f.subPlot(2,2,3,3);
imagesc( data );
f.label( id, 'topleft', 'B');
ax.YTickLabel = [];
xlabel( 'X [px]' );
% Add a color bar on the right of the plot.
c = f.colorbar( id );

% No need to get axes handle, since not changing any propeties
[~, id] = f.subPlot(1,2,1,3);
imagesc( data );
f.label( id, 'topleft', 'c^2');
xlabel( 'X [px]' );
ylabel( 'Y [px]' );

[ax, id] = f.subPlot(2,1,3,1);
imagesc( data );
f.label( id, 'bottomright', 'd');
ax.XTickLabel = [];
ax.YTickLabel = [];

set(gcf, 'Color', 'w');
set(gcf, 'InvertHardCopy', 'off');

print( 'example01' , '-dpng', '-r600' );
```

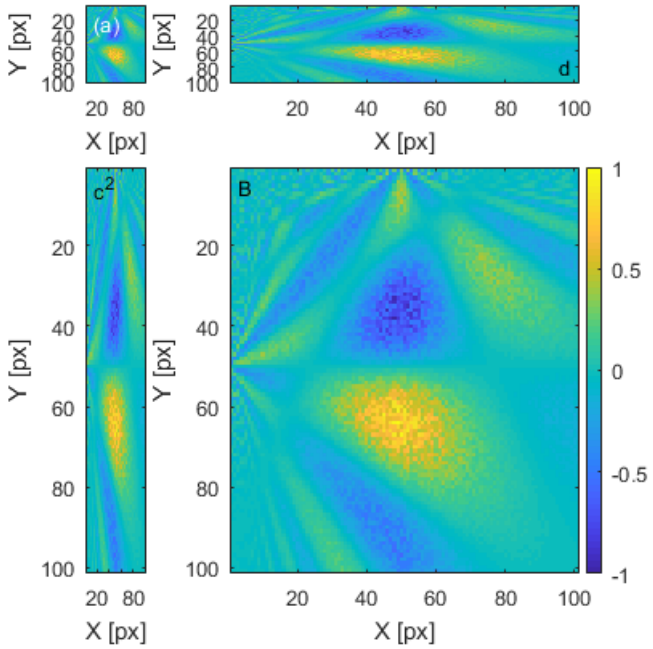


Multi-column & multi-row subplots with sparse margins

The 'sparse' preset can be used when each subplot will have different scales or zooming factors. The above example was modified by changing the preset, when creating a *sublimeFigure* object, and adding labels to x and y axes on all subplots.

```
% Load a sample data set
load( 'sampleData.mat', 'data');

% Create a sublimeFigure object using a 'sparse' preset
f = sublimeFigure( 'sparse' );
f.numRows = 4;
f.numColumns = 4;
f.rightOuterPadding = 1; % Leave extra space outside to plot a
colour bar

% Create a sub-plot in the top left corner and return axes
handle and
% subplot ID number within the sublimeFigure.
[~, id] = f.subPlot(1,1); % Sub plot occupies one cell
imagesc( data );
% Create a label and return a handle
lbl = f.label( id, 'topleft', '(a)');
```

```
% Use handle to label, which is text object, to change colour
lbl.Color = 'w';
xlabel( 'X [px]' );
ylabel( 'Y [px]' );

% Create a subplot spanning over 3 rows and 3 columnns
[~, id] = f.subPlot(2,2,3,3);
imagesc( data );
f.label( id, 'topleft', 'B');
xlabel( 'X [px]' );
ylabel( 'Y [px]' );
% Add a color bar on the right of the plot.
c = f.colorbar( id );

[~, id] = f.subPlot(1,2,1,3);
imagesc( data );
f.label( id, 'topleft', 'c^2');
xlabel( 'X [px]' );
ylabel( 'Y [px]' );

[~, id] = f.subPlot(2,1,3,1);
imagesc( data );
f.label( id, 'bottomright', 'd');
xlabel( 'X [px]' );
ylabel( 'Y [px]' );

set(gcf, 'Color', 'w');
set(gcf, 'InvertHardCopy', 'off');

print( 'example02' , '-dpng', '-r600' );
```



Legal Bit