telecommunications & signal processing laboratory

Improving the Presentation of Matlab Plots

P. Kabal

Department of Electrical & Computer Engineering

McGill University

McGill

Version 1: 2006-01-30

Version 1.1: 2006-06-05

Table of Contents

Improving the Presentation of Matlab Plots

1 Introduction

This document describes a number of strategies which go towards the goal of producing publication quality plots from Matlab. One finds much to criticize in the quality of plots that are reproduced in today's journals. This is due to the fact that the authors supply the plots without having a clear view of how they will be processed to produce the final plot on the printed page. We give some guidelines and supply Matlab routines that streamline the application of these guidelines.

The focus of this report is on 2-D plots. Three groups of routines are described. The first group sets up the default plot size, default fonts and provides for saving the figure to a file. The second group of routines is used to customize the axis ticks and labels. The third group can be used to customize the appearance of the plot lines.

2 Matlab Parameters and Matlab Plot Files

Matlab provides a user control of the plot parameters directly from menus on the figure. However, it is suggested that the setting of the plot parameters plotting be encapsulated in a Matlab script. The script can be rerun if the plot data is updated and/or the plot parameters need tweaking.

A companion document [1] shows that PostScript figures are the format of choice for high quality reproduction on the page. To ensure that the PostScript figures are reproduced on all types of printers, it is strongly recommended that the document be distributed in PDF format.

3 Plot Size and Plot Fonts

The first group of routines set the plot size and plot font. Some points to consider are:

- The aspect ratio of the plot should be chosen based on the relative "importance" of the *x* and *y* data. For instance, it is traditional to plot a probability of error versus signal-to-noise ratio on a graph which is taller than it is wide. This allows for easier reading of the logarithmically spaced *y* values on the plot. Plots of waveforms are often presented as "strip" charts: the graph is shorter than it is wide: the "envelope" of the waveform is more important than the exact amplitude values. Notwithstanding special cases, a good starting point is to choose a 5:4 ratio of width to height.

- The plot should be able to fit into the available space. For a two-column journal page this means that the overall plot width should less than about 9 cm. The axes of the graph should then be 8 cm or less in width. For a technical report (such as this one), the width can be somewhat larger.
- The font size of the plot should be compatible with the font size of the text. It is common to see the figure fonts being one size smaller than the text font. This report is set in 11 point font and the figure text fonts for the examples are set in 10 pt size. Journal papers are often set in 10 points (full papers) or 9 points (correspondence papers). The text font should be reduced accordingly.
- The font type of the plot should be compatible with the font in the main text. If any of the annotations includes mathematics, the font type in the figure should be the *same* as the main text. In this report, the text typeface is Times New Roman (a serif font); the Matlab figure fonts will be chosen to be the closely related Times font (a serif font).
- The plot should be inserted at full scale into the final document. Any scaling in the document will affect both the plot and the text on the plot. Even worse, if different scaling is applied to the x and y dimensions, the text will be distorted.

Given the desire to match the font size and the figure size to the document, it seems clear that the figure must be created in its final size and inserted into the final document without further scaling.[1] For a plot which has been created in Matlab with the proper fonts and figure size, it is then important that these properties not change on exporting the file.

3.1 Plot Size and Plot Fonts in Matlab

We introduce three Matlab routines: **SetPlotSize**, **SetPlotFon**t, and **WritePlot**. These implement the suggestions on setting the plot size and plot fonts. The Matlab code for these routines is given in the Appendix. The full set of calling options is given there.

[1] As noted later, to decrease the size of dashes in dashed lines for small plots (as in a journal publication), it may be desirable to create a plot at a larger scale and import the plot into the document at a reduced magnification. The result of scaling and reduction should be chosen so that the final size of the plot and the size of the font are as desired..

SetPlotSize: Sets the default axes size of the plot using whatever units are appropriate. A typical call is of the form **SetPlotSize([7.5 6], 'centimeters')**.

SetPlotFont: Sets the default font type and the default font size. A typical call is of the form **SetPlotFont('Times',10)**. This routine also scales the default plot symbols to have a size which is a fixed fraction of the font size.

WritePlot: Writes the plot to a file. A typical call is of the form **WritePlot('Myplot.eps')**. This routine freezes the axes scaling and annotations and sets a number of plotting parameters to ensure that the plot axes are the same as appear on the screen. It also modifies some settings to ensure good resolution for bitmap output.[2]

4 Plot Axes and Tick Labels

For publication quality plots, the plot scale and labelled ticks must be chosen carefully. Some considerations are:

- Matlab chooses a scale for the plotted data. The default mode is to choose a scale which encompasses the data but extends to a "nice" number on either side. In many cases, a "tight" axis is preferred. The suggestion here is that the axis scale should be set by the user using the Matlab **axis** routine.[3]
- With small plots (such as will fit into the column of a journal page), one must avoid a plot that has overly "busy" axis annotations. This means that there should probably be a maximum of 6 labelled ticks on each axis. Since Matlab generally generates more than this number of labelled ticks, the tick values should be set manually with the **XTick** and/or **YTick** options.
- Matlab allows for text labels to be associated with the tick marks. Matlab provides a TeX interpreter to allow mathematical symbols to be placed in text strings. However, the TeX interpreter is not enabled for tick labels. Two routines (**SetXTickLabel** and **SetYTickLabel**) are described below which allow symbols to appear in tick labels.

[2] Matlab normally uses a vector format (**renderer** property set to **painter**) for PostScript files. It switches to a bitmap format (**renderer** property set to **zbuffer**) for "complicated" plots.

[3] Several of the routines (**DashLine** and **XYmerge**) need to know the axis scaling and so it is prudent to set the scaling before actually plotting the data. In practice, one would plot the data first letting Matlab choose the scaling and then refine this setting and fix it using the Matlab **axis** command.

- Minor ticks are shorter ticks lying between the labelled ticks. These can provide additional scaling information for the reader. Matlab provides a **MinorTick** option, but this option creates a lot of intermediate ticks. Instead, a routine is described below which adds a single "minor" tick between each labelled tick for linear plots.
- Matlab only provides for axes around the plot. In certain cases, it is useful to have an axis placed at the origin of the plot. Routines (**Xaxis** and **Yaxis**) to do this are described below.

4.1 Plot Axes and Label Modification

The suggestions above involve using existing mechanisms to customize the plot axes. The additional routines that are sometimes useful are:

MidTick: This routine creates a minor tick between each normal tick. This minor tick is 75% of the length of the normal tick. This routine should be called after plotting the data.

SetXTickLabel / **SetYTickLabel**: These routines allow symbols to be included in the tick labels. They work by turning off Matlab's tick labels and then positioning a text string beside the ticks.

Xaxis / **Yaxis**: These routines provide for additional axes to be positioned on the graph. By default, these axes pass through the origin of the graph. These axes are created by superimposing a scrunched Matlab axes on top of the existing axes.

5 Distinguishing Plot Lines

The third group of Matlab routines are used to distinguish lines on a plot. Colour is a useful way to differentiate curves. However, for the sake of non-colour reproduction, multiple lines in a plot have to be distinguishable in other ways than just colour.[4] One can change the colour, line width, line style, or line marker. Here we consider colour and line style changes. Some considerations:

- For multiple curves in the same plot, Matlab cycles through a set of 6 colours (blue, mid-green, red, cyan, magenta, and yellow). The last three colours are too light to reproduce well in non-colour printouts. A routine (**SetPlotColors**) which modifies the default colours is described below.

[4] One can (and perhaps should) use both colour and line style to distinguish lines on a plot. The reader of an electronic version of the document with colour gets both; the reader of a black and white printout sees only the line style differences.

- Different line styles (dashed, etc.) are useful to distinguish lines on a plot. Matlab has four line styles, solid, dashed, dotted, and dash-dot. The dotted line appears much fainter than the others and so does not mix well with the other line styles. For fewer than 3 lines on a plot, the built in Matlab line styles solid, dashed, and dash-dot work well. A routine (**DashLine**) which can create arbitrary line styles is described below.
- Note that the line segments in the different line styles available in Matlab do not scale with plot size. For instance a dashed line in a small plot has dashes which are proportionally too long. This can cause problems in displaying a legend. The sample line styles in the legend may have too short a length to properly identify the type of line. One solution to this problem is to draw the plot at, say, 2 times the final size. The plot size and the plot font sizes should be scaled by this factor. When the plot is included into a document, it should be scaled by the inverse factor. This way the “pitch” of the dashes and dots on the lines is modified to give an improved look to small plots.

5.1 Modifying the Line Styles

Additional customizations of the line styles are possible with the following routines.

SetPlotColors: This Matlab routine modifies the default colours for plots. The last three colours of the set of line colours are set to a darker cyan, a darker magenta, and a darker yellow.

DashLine: This Matlab routine creates arbitrary dashed line patterns. This is accomplished by actually creating short line segments which can then be plotted. It should be noted that in order to determine the lengths of the line segments, the axis scaling must be specified before this routine is called.

6 Sample Plots

This section shows sample plots using some of the suggestions given earlier.

6.1 Example: Plot of Four Curves

The first Matlab plotting script is shown below. Comments on the code are given below. The resulting plot is shown in Fig. 1.

- The plotting commands are surrounded by the “boiler-plate” routines **SetPlotSize** and **SetPlotFont** at the beginning, and **WritePlot** at the end

- The plot has four lines that will be distinguished both by colour and line style. **SetPlot-Colour** darkens the colour of the fourth line to a darker cyan.
- The routine **DashLine** is used to set up 4 different dash patterns. Each pattern consists of a sequence of alternating positive and negative values. The positive values represent line segment lengths in points and the negative values represent line gap lengths in points. The Matlab function **axis** is called before calling **DashLine**. We will have to restore the axis scaling after plotting.
- The plot itself uses the Matlab **plot** command. The axis scaling is reset to the value before plotting.
- The axis ticks are set manually. The routine **SetXTickLabel** is used to label the *x* axis tick marks in terms of fractions of π. There are 5 labelled *x* ticks and 5 labelled *y* ticks.
- A legend is added to the plot. A special routine **SetLegendProp** reaches inside the legend axes and redraws the lines therein with the patterns used for the plots. It also “freezes” the legend so that the customization of the plot lines is not overwritten later.
- The routine **MidTick** adds minor ticks between the labelled ticks for both the *x* and *y* axes.
- The routines **Xaxis** and **Yaxis** overlay additional axes passing the origin of the plot.

```
function TestPlot1
% Test plot functions

% Set the plot size and default font
figure;
SetPlotSize ([1.5 1.5 12, 8], 'centimeters');
SetPlotFont ('Times', 10);

% Set the plot colours (darker colours)
SetPlotColors;

% Generate some offset sinusoids
Ngraph = 4;
for (i = 1:Ngraph)
  w(:,i) = (linspace (-pi, pi, 201))';
  Theta = (i - 1) * pi / Ngraph;
  s(:,i) = sin(w(:,i) + Theta);
end

% Set the axis limits prior to calling DashLine
axis ([-pi pi -1.1 1.1]);
axdata = axis;

% Make dash/dot line styles
```

```
Pattern{1} = [5 -5]; % Long dash
Pattern{2} = [];     % Line
Pattern{3} = [2 -2]; % Short dash
Pattern{4} = [5 -2 2 -2]; % Long dash - short dash

% Create the dashed lines
for (i = 1:Ngraph)
  [wd{i} sd{i}] = DashLine (w(:,i), s(:,i), Pattern{i});
end

% Plot the data
plot (wd{1}, sd{1}, wd{2}, sd{2}, wd{3}, sd{3}, wd{4}, sd{4});

% Restore the axis limits
axis (axdata);

% Set the tick marks
set (gca, 'XTick', [-pi, -pi/2, 0, pi/2, pi]);
SetXTickLabel ({'-\pi' '-\pi/2' '0' '\pi/2' '\pi'});
set (gca, 'YTick', [-1 -0.5 0 0.5 1]);

% Labels
xlabel ('angle (radians)');
ylabel ('amplitude');

% Add a legend
h = legend ('\theta = 0', '\theta = \pi/4', ...
            '\theta = \pi/2', '\theta = 3\pi/2');
LemonChiffon = [1 0.9725 0.7805];
set (h, 'Color', LemonChiffon);
SetLegendProp (h, 'Pattern', Pattern);

% Add a minor tick between each existing tick
MidTick ('XY');

% Add axes going through the origin
Xaxis;
Yaxis;

% Write the plot to a file
WritePlot ('TestPlot1.eps');

return
```

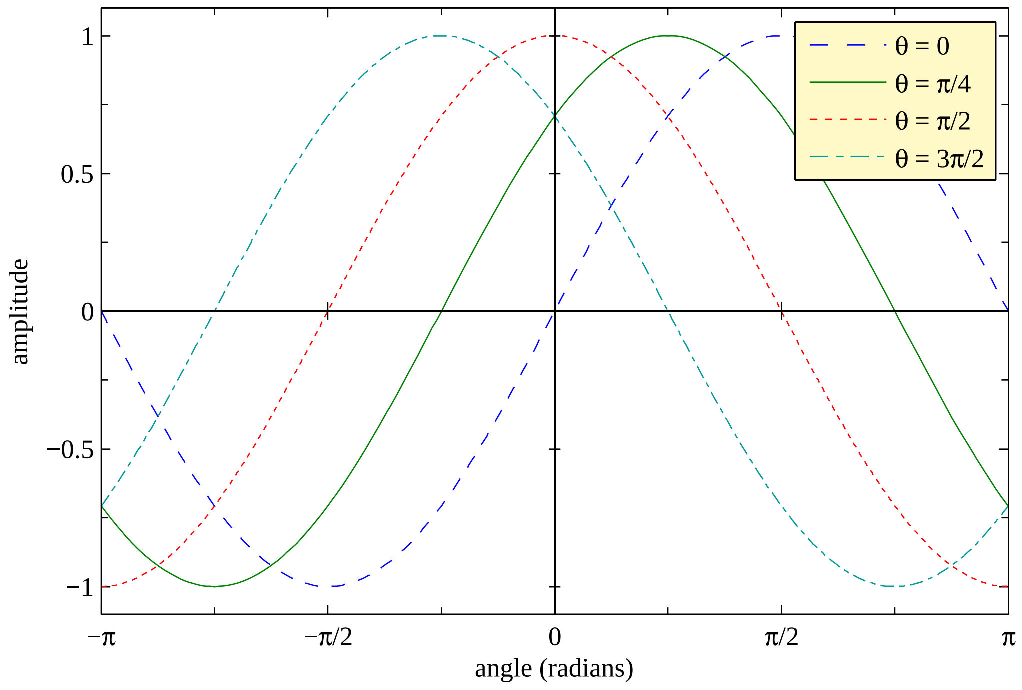

Fig. 1 Sample plot

6.2 Example: Plot of a Large Number of Points

A second sample plot is generated by the Matlab script shown below. This plot uses much of the same setup as the earlier plot. The frequency response that is to be plotted is sampled at a large number of points. This ensures that some of the sampled points land in the deep nulls in the response. However, with such a large number of sample points, the plot size would be excessively large. The data points are passed though the routine **XYmerge** which merges nearly co-linear segments. In smooth parts of the curve, the number of plot points is dramatically reduced. The original 8193 points in the function are reduced to 413 plot points with no noticeable change in the appearance of the plot. The plot itself is shown in Fig. 2.

```
function TestPlot2
% Test plot functions

% Set the plot size and default font
figure;
SetPlotSize ([1.5 1.5 12, 6], 'centimeters');
SetPlotFont ('Times', 10);

% Generate a Hamming window and its frequency response
alpha = 0.08;
N = 48;
b = (1+alpha)/2 - (1-alpha)/2 * cos (pi * (2*(0:N-1)+1)/N);
```

```
w = linspace (0, pi, 8193);
hdB = 20 * log10(abs (freqz (b, 1, w)));
hdB = hdB - hdB(1);

% Set the axis limits prior to calling XYmerge
axis ([w(1) w(end) -80 5]);
axdata = axis;

% Merge nearly co-linear vectors
[wm, hdBm] = XYmerge (w, hdB);

% Plot the frequency response
plot (wm, hdBm);

% Restore the axis limits
axis (axdata);

% Set the tick marks
set (gca, 'XTick', [0, pi/4, pi/2, 3*pi/4, pi]);
SetXTickLabel ({'0' '\pi/4' '\pi/2' '3\pi/4', '\pi'});
set (gca, 'YTick', -80:20:0);

% Labels
xlabel ('frequency (radians)');
ylabel ('amplitude (dB)');

% Add a minor tick between each existing tick
MidTick ('XY');

% Write the plot to a file
WritePlot ('TestPlot2.eps');

return
```

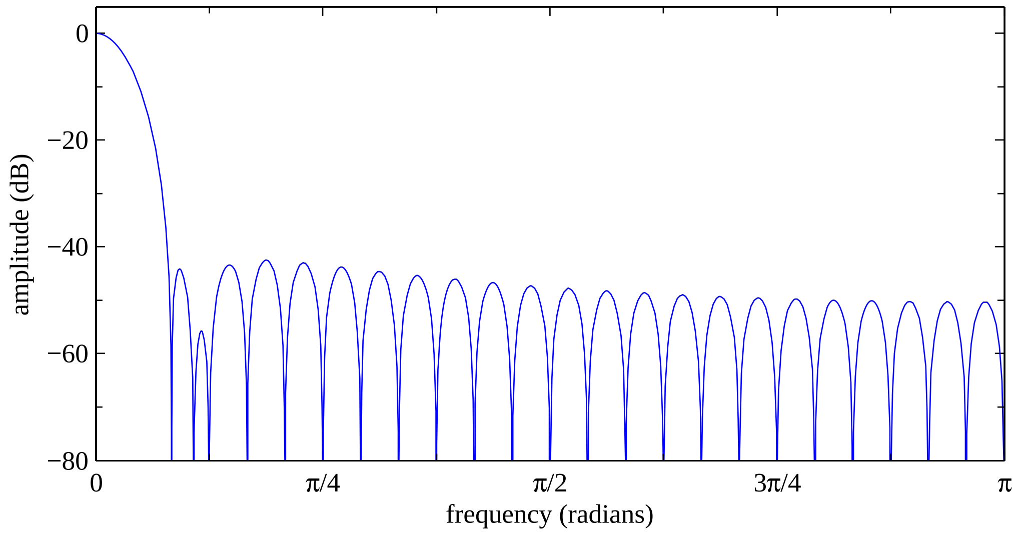

Fig. 2 Frequency response plot for a Hamming window

References

1. P. Kabal, *Matlab Plot in Microsoft Word*, Technical Report, Electrical & Computer Engineering, McGill University, Jan. 2006 (on-line at www-MMSP.ECE.McGill.ca/Documents).

Appendix A Matlab Routines

This appendix give the listings of the Matlab routines used to in the sample plots in the main text.. Downloadable versions of the latest revisions are available on-line at www-MMSP.ECE.McGill.ca/Documents.

A.1 SetPlotSize

This routine sets the size of the plot in the units specified. The size of the **axes** box can be specified as 2 values or 4 values. In the first case, the 2 values are the *x* dimension and the *y* dimension of the **axes** box. In the second case, the first two values are the *x* offset and *y* offset of the **axes** box within the figure box, and the second two values are the **axes** box dimensions. The **figure** box is resized to accommodate the specified **axes** box.

```
function SetPlotSize (pos, units, varargin)
% SetPlotSize  Set the size of the current plot
%   SetPlotSize(pos,units,figurecolor)
%   SetPlotSize(pos,units,options)
%   SetPlotSize(pos)        % default units "inches" assumed
%
% pos   - axes position [xsize, ysize] or
%                       [xleft, ybottom, xsize, ysize]
% units - units for the dimensions, default 'inches'. The
%         choices are 'centimeters', 'pixels', 'inches',
%         'points', and 'normalized'.
% figurecolor - Figure and axes background color.
%         Use 'none' to get transparent axes.
% options - Paired options, e.g. to set the figure color to
%         transparent, and the axes color to white, use
%           'Color', 'none', 'DefaultAxesColor', 'w'
%
% Note: The 'Units' for the current axes are set to the units
%       specified.  If the units are absolute measurements
%       (i.e. not 'normalized'), then figure can be resized on
%       the screen using the mouse.  This feature can be used to
%       "crop" white space above and to the left of the axes.

% $Id: SetPlotSize.m,v 1.17 2006/01/27 22:32:38 pkabal Exp $

% Notes:
% - The default axes position for Matlab is given in normalized
%   units as [0.130 0.110 0.775 0.815].  These values are used
%   to determine the space around the axes.

if (nargin <= 1)
  units = 'inches';
end

Np = length(pos);
if (Np ~= 2 & Np ~= 4)
  error ('SetPlotSize: pos must have 2 or 4 elements');
end

posnorm = [0.130 0.110 0.775 0.815];
if (Np == 2)
  figsize = pos ./ posnorm(3:4);
  axespos = [(posnorm(1:2) .* figsize) pos];
else
  figsize = (pos(1:2) + pos(3:4)) ./ (posnorm(1:2) +
posnorm(3:4));
```

```
  axespos = pos;
end

% Set the figure
saveunits = get (gcf, 'Units');
set (gcf, 'Units', units);
figpos = get (gcf, 'Position');
set (gcf, 'Position', [figpos(1:2) figsize]);
set (gcf, 'Units', saveunits);

% Set the axes
set (gca, 'Units', units);
set (gca, 'Position', axespos);

Nv = length (varargin);
if (Nv == 1)
  set (gcf, 'Color', varargin{1}, 'DefaultAxesColor',
varargin{1});
elseif (Nv > 1)
  set (gcf, varargin{:});
end

disp (sprintf ('SetPlotSize: Figure size: %g x %g', figsize));

return
```

A.2 SetPlotFont

This routine sets the *default* font typeface and the default font size for both the text strings and axis labels. The line **MarkerSize** parameter is also reset to be a fixed fraction of the font size.

```
function SetPlotFont (FontName, FontSize)
% SetPlotFont - Set font name and size for plots
%   SetPlotFont(FontName,FontSize)
%   SetPlotFont(FontName)          % Default 10 pt font size
%   SetPlotFont                    % Default 10 pt Times

% $Id: SetPlotFont.m,v 1.3 2006/06/05 19:01:37 pkabal Exp $
if (nargin < 2)
   FontSize = 10;
end
if (nargin < 1)
   FontName = 'Times';
end

% For figure children, not yet created
set (gcf, ...
   'DefaultTextFontName', FontName, 'DefaultTextFontSize',
FontSize, ...
   'DefaultAxesFontName', FontName, 'DefaultAxesFontSize',
FontSize, ...
   'DefaultLineMarkerSize', 0.5*FontSize);
% For axes already set up in the figure
set (gca, ...
   'FontName', FontName, 'FontSize', FontSize);

return
```

A.3 SetPlotColors

This routine sets the default ColorOrder colours for plot lines. Plot lines are cycled through 6 colours in this matrix. The last three colours are made somewhat darker as otherwise they do not reproduce well on a black and white printer.

```
function CO = SetPlotColors (Colors)
% CO = SetPlotColors (Colors)
% This routine sets the default colors for plot lines. The
% default settings override the lighter standard Matlab colors.
% These darker colors are rendered better on a black and white
% printer.
%
% The input cell array can be used to override the default
% colors set by this routine. Any empty cells in the input leave
% the corresponding colors unchanged.
%   SetPlotColors({[]; [0 0.2 0]});    % Darker green
%
% N.B. Call this routine before plotting

% $Id: SetPlotColors.m,v 1.3 2006/01/27 22:36:51 pkabal Exp $

% Original colors
Blue        = [ 0    0    1    ];
Green       = [ 0    0.5  0    ];      % Mid-green
Red         = [ 1    0    0    ];
Cyan        = [ 0    0.75 0.75 ];
Magenta     = [ 0.75 0    0.75 ];
Yellow      = [ 0.75 0.75 0    ];
Gray        = [ 0.25 0.25 0.25 ];

% Darker colors
DarkCyan    = [ 0    0.6  0.6  ];
DarkMagenta = [ 0.5  0    0.5  ];
DarkYellow  = [ 0.6  0.6  0    ];

if (nargin < 1)
  Colors = {[]; []; []; []; []; [];};
end

CO = get (gcf, 'DefaultAxesColorOrder');
% Matlab cycles through the first six of these colors
CO(4:6,:) = [DarkCyan; DarkMagenta; DarkYellow];

% Insert the user defined colors
NC = length (Colors);
for (i = 1:NC)
  if (~ isempty (Colors{i}))
    CO(i,:) = Colors{i};
  end
end

% Set the color order
set (gcf, 'DefaultAxesColorOrder', CO);
```

```
return
```

A.4 DashLine

This routine takes a pattern of on-off lengths and applies these to vectors of *x* and *y* values. It creates line segments for plotting. The resulting plot vector consists of plot segments separated by NaN values. The NaN values interrupt plotting. The next pair of *x* and *y* values become the beginning of the next line segment. The pattern is specified in units of points (1 inch is equal to 72 points). The axis scaling (Matlab **axis** command) and plot size (**SetPlotSize**) must be set before calling **DashLine**. It uses the axis limits and the plot size to determine the mapping between data units and points.

```
function [xd, yd] = DashLine (x, y, pattern, DataPt)
% [xd, yd] = DashLine (x, y, pattern [, DataPt])
% Generate data with general dash/dot line styles
%  - x: vector of X-values
%  - y: vector of Y-values
%  - pattern: vector of pattern lengths. These are alternating
%    on (positive) / off (negative) values. The absolute value
%    of each element is the length in points. The pattern is
%    repeated as needed.
%  - PtData: Optional two element conversion factor. If this
%    parameter is not supplied, this routine calculates this
%    value based on the current axis limits and axis sizes.
%    - PtData(1): X data units per point (1/72 inch)
%    - PtData(2): Y data units per point (1/72 inch)
%    If the data is to be plotted on a log scale (as indicated
%    by the XScale and YScale properties of the current plot),
%    the conversion correponding factor is log10(units) per pt.
%
% [xd yd]: output data points consisting of visible segments
%    separated by NaN values
%
%  Notes:
%    - To determine the proper lengths of the segments, the size
%      of the plot and the data ranges to be plotted have to be
%      known.
%        SetPlotSize([xdim, ydim], 'centimeters');
%        axis ([Xmin Xmax Ymin Ymax]);
%    - For log plots, this routine needs to know if a plot is
%      log or not.
%        set (gca, 'XScale', 'log')  % Use log X

% $Id: DashLine.m,v 1.8 2006/01/27 22:26:22 pkabal Exp $

N = length (x);
if (N ~= length(y))
  error ('DashLine: x and y vectors not the same length');
end

XScale = 'linear';
YScale = 'linear';
if (nargin < 4)
  [DataPt, XScale, YScale] = SFdataXpt;
end
```

```
LogX = strcmp (XScale, 'log');
LogY = strcmp (YScale, 'log');
if (LogX)
  x = log10 (x);
end
if (LogY)
  y = log10 (y);
end

% Flag for long lines (could generate huge number of dashes)
Wlong = 0;
Winf = 0;
if (length (pattern) == 0)
  Maxdp = Inf;    % No pattern
else
  LenPat = sum (abs (pattern));
  Maxdp = 1000 * LenPat;
end

[vis, remain, m] = RemPat (pattern, 0);

k = 0;
xp = x(1);
yp = y(1);

for (i = (1:N))
  dx = x(i) - xp;
  dy = y(i) - yp;
  dp = sqrt ((dx / DataPt(1))^2 + (dy / DataPt(2))^2);

  if (~ isfinite (dp))
    % NaN end point or infinite line segment
    if (isinf (dp))
      if (~ Winf)
        fprintf ('>>> DashLine - Infinite line segment(s)
omitted\n');
      end
      Winf = Winf + 1;
    end
    xp = x(i);
    yp = y(i);
    [vis, remain, m] = RemPat (pattern, 0);  % Reset the pattern
    continue;  % Get the next point
  end

  if (dp >= Maxdp)
    if (~ Wlong)
      fprintf ('>>> DashLine - Long line segment, dashes turned
off\n');
    end
    Wlong = Wlong + 1;
    xp = x(i);
    yp = y(i);
    k = k + 1;
    xd(k) = xp;
    yd(k) = yp;
    continue;  % Get the next point
  end
```

```
    while (dp >= remain)
      % Finished a pattern segment
      % Update the position if the visibility changes
      [visN, remainN, m] = RemPat (pattern, m);
      if (vis ~= visN)
         fract = remain / dp;
        xp = xp + fract * dx;
        yp = yp + fract * dy;
        if (vis)
          k = k + 1;
          xd(k) = xp;        % End a visible segment
          yd(k) = yp;
        else
          k = k + 1;
          xd(k) = NaN;       % End an invisible segment
          yd(k) = NaN;
          k = k + 1;
          xd(k) = xp;        % Start a visible segment
          yd(k) = yp;
        end
        remain = remainN;          % Switched visibility
      else
        remain = remain + remainN; % Same visibility
      end
      vis = visN;
      dx = x(i) - xp;
      dy = y(i) - yp;
      dp = sqrt ((dx / DataPt(1))^2 + (dy / DataPt(2))^2);
    end

    % Reached a data point without finishing a pattern segment
    xp = x(i);
    yp = y(i);
    if (vis)
      k = k + 1;
      xd(k) = xp;
      yd(k) = yp;
    end
    remain = remain - dp;
  end

  if (LogX)
    xd = 10^xd;
  end
  if (LogY)
    yd = 10^yd;
  end

  return

  %===========
  function [vis, remain, m] = RemPat (pattern, m)
  % Get the next pattern element (increment m, modulo the number
  % of pattern elements)

  Lp = length (pattern);
  if (Lp <= 0)
    m = 1;
    remain = Inf;
    vis = 1;
  else
    m = mod (m, Lp) + 1;
    remain = abs (pattern(m));
    vis = (pattern(m) > 0);
  end

  return
```

A.5 SetLegendProp

This routine sets properties for a plot legend. The additional pseudo-property 'Pattern' is recognized (see **DashLine**). Some of the customizations done by this routine are “fragile”. Normally, Matlab will redraw the legend when the plot is printed. This redrawing will undo customizations such as the pattern lines. To prevent this, the legend tag is nulled out so that the axes box containing the legend is no longer recognized as a legend and will not be redrawn.

```
function h = SetLegendProp (varargin)
% h = SetLegendProp ([h], 'Property1', Value, ... )
% This routine sets properties for a plot legend. The additional
% pseudo-property 'Pattern' is recognized (see DashLine). The
% legend is also "frozen". Normally, Matlab will redraw the
% legend when the plot is printed. This redrawing may undo
% customizations such as the Pattern lines. The legend Tag is
% nulled out so that the axes box containing the legend is no
% longer recognized as a legend and will not be redrawn.
%
% $Id: SetLegendProp.m,v 1.4 2006/06/02 11:59:14 pkabal Exp $

% Find the legend handle
if (ishandle (varargin{1}))
  h = varargin{1};
  varargin(1) = [];
else
  fig = get (gca, 'Parent');
  h = findobj (fig, 'Type', 'axes', 'Tag', 'legend');
end

hsave = gca;

% Pick up the pattern
Pattern = [];
i = 1;
while (i <= length (varargin))
  if (strcmpi (varargin{i}, 'Pattern'))
    Pattern = varargin{i+1};
    varargin(i) = [];
    varargin(i) = [];
  else
    i = i + 1;
  end
end

% Set the legend options
Nv = length (varargin);
Field = {varargin{1:2:end}};
Value = {varargin{2:2:end}};
set (h, Field, Value);

% Find lines in the legend axis
hl = findobj (h, 'Type', 'line');
```

```
% Keep only sample lines

% The lines appear in reverse order to the legend, i.e. most
% recently added first
i = 1;
while (i <= length(hl))
  XData = get (hl(i), 'XData');
  if (length (XData) ~= 2)
    hl(i) = [];
  else
    i = i + 1;
  end
end
hl = flipud (hl);

% Set the patterns
Nl = length (hl);
Np = length (Pattern);
Ns = 0;
for (i = 1:min(Nl,Np))
  if (~ isempty (Pattern{i}))
    XData = get (hl(i), 'XData');
    YData = get (hl(i), 'YData');
    axes (h);   % Bring the legend to the front so that DashLine
                % sees the correct scaling
    [XData, YData] = DashLine (XData, YData, Pattern{i});
    set (hl(i), 'XData', XData);
    set (hl(i), 'YData', YData);
    Ns = Ns + 1;
  end
end

% To avoid lines in the legend being redrawn when printing
%  - Turn off the legend propery tag
%  - Make the legend axes handle invisible
% Leave the legend on top so that it is not hidden by the plot
set (h, 'Tag', '');
set (h, 'HandleVisibility', 'off');

return
```

A.6 MidTick

This routine overlays the current plot with a transparent **axes** box. For linear axes, the ticks on that **axes** box are 75% of the default tick size and are placed midway between the ticks on the current plot. For log axes, the ticks are 50% of the default tick size and are placed between decade values. The argument of this function specified whether *x*, *y* and/or *z* axes are to receive the minor ticks.

```
function h = MidTick (varargin)
% h = MidTick ([[ax,] Axes])
% h = MidTick ([ax,] <Tick locations>)
% Generate intermediate ticks on a plot axis.
% This function generates intermediate ticks between the
% existing X, Y, or Z axis ticks in a plot. For a linear axis,
% the intermediate ticks (0.75 of the normal length) lie midway
% between the existing ticks. For a log axis, the ticks (0.5 of
% the normal length) lie at 1, 2, ..., 9 times the decade
% points.
%
% There are two basic call sequences.
% (1) Specify the axes to have mid-ticks 'X', 'Y', or 'Y' or a
%     combination.
%     h = MidTick ('XY');
% (2) Specify mid-tick locations using 'XTick', 'YTick', and/or
%     'ZTick'.
%     h = MidTick ('XTick', [x1, ..., xN],...);
% Any remaining arguments are used to set other plot options,
% such as 'TickLength'.
%
% Note that Matlab generates minor ticks for log axes. The only
% reason to have this routine do so for log axes is for those
% cases where some of the minor ticks are missing due to a bug
% in Matlab (for versions up to 7.2 at least).
%
%  A new transparent axis is generated on top of the existing
%  axis. Note that this new axis must remain on top for the
%  newly generated ticks to be visible. The handle for this new
%  axis is returned.
%  *** Invoke this routine after all plotting and labelling is
%      done ***
%  *** It is suggested that the graph limits be set (using
%      a call to 'axis', for instance) before invoking this
%      routine ***

% $Id: MidTick.m,v 1.22 2006/06/01 23:25:13 pkabal Exp $

[ax, XA, YA, ZA, varg] = Proc_Args (varargin{:});

Freeze = [];
XTick = [];
YTick = [];
ZTick = [];
if (XA)
```

```
  XTick = Gen_Tick (get (ax, 'XTick'), get (ax, 'XScale'));
  if (~ isempty (XTick) & ...

      strcmpi (get (ax, 'XLimMode'), 'auto'))
    Lim = get (ax, 'XLim');
    set (ax, 'XLim', Lim);
    Freeze = [Freeze 'X'];
  end
end
if (YA)
  YTick = Gen_Tick (get (ax, 'YTick'), get (ax, 'YScale'));
  if (~ isempty (YTick) & ...
      strcmpi (get (ax, 'YLimMode'), 'auto'))
    Lim = get (ax, 'YLim');
    set (ax, 'YLim', Lim);
    Freeze = [Freeze 'Y'];
  end
end
if (ZA)
  ZTick = Gen_Tick (get (ax, 'ZTick'), get (ax, 'ZScale'));
  if (~ isempty (ZTick) & ...
      strcmpi (get (ax, 'ZLimMode'), 'auto'))
    Lim = get (ax, 'ZLim');
    set (ax, 'ZLim', Lim);
    Freeze = [Freeze 'Z'];
  end
end

% Generate short ticks at the mid-points
% There is only one TickLength parameter for the whole plot
% (not one per axis)
if (strcmpi (get (ax, 'XScale'), 'log') || ...
    strcmpi (get (ax, 'YScale'), 'log') || ...
    strcmpi (get (ax, 'ZScale'), 'log'))
  TickLength = 0.5 * get (ax, 'TickLength');
else
  TickLength = 0.75 * get (ax, 'TickLength');
end

h = BlankAxes;
set (h, 'TickLength', TickLength);
set (h, 'XTick', XTick);
set (h, 'XMinorTick', 'off');
if (length (XTick) > 1 )
  set (ax, 'XMinorTick', 'off');
end
set (h, 'YTick', YTick);
set (h, 'YMinorTick', 'off');
if (length (YTick) > 1)
  set (h, 'YMinorTick', 'off');
end
set (h, 'ZTick', ZTick);
set (h, 'ZMinorTick', 'off');
if (length (ZTick) > 1)
  set (h, 'ZMinorTick', 'off');
end

% Appy the rest of the arguments
if (length (varg) > 1)
```

```
  set (h, varg{1:2:end}, varg{2:2:end});
end



if (~ isempty (Freeze))
  disp (['>>> MidTick - Plot limits (', Freeze, ') frozen']);
end

return

%==========
% Process arguments, filling in defaults
function [ax, XA, YA, ZA, varg] = Proc_Args (varargin)

% Axis, default to gca
if (nargin > 0)
  if (ischar (varargin{1}))
    varargin = {gca, varargin{:}};
  end
end
ax = varargin{1};

% The plot options are paired. If the current number of
% arguments is even, the Axes option is present:
%   ax, Axes, paired-arguments
%   ax, paired-arguments
N = length (varargin);

XA = 0;
YA = 0;
ZA = 0;
if (mod (N, 2) == 0)
  option = varargin{2};
  if (length (varargin) > 2)
    varargin = {varargin{1} varargin{3:end}};    % Remove the
                                                 % argument
  end
  if (~ isempty (strfind (option, 'X')))
    XA = 1;
  end
  if (~ isempty (strfind (option, 'Y')))
    YA = 1;
  end
  if (~ isempty( strfind (option, 'Z')))
    ZA = 1;
  end
end

% Rest of the arguments
varg = {};
if (length (varargin) >= 2)
  varg = {varargin{2:end}};
end

return
```

```
%==========
% Generate ticks midway between existing ticks
function htick = Gen_Tick (Tick, Scale)

NTick = length (Tick);

htick = [];

if (NTick > 1)
  if (strcmpi (Scale, 'linear'))
    htick = 0.5 * (Tick(1:NTick-1) + Tick(2:NTick));

  else
    LTick = floor (log10(Tick));
    htick = [1 2 3 4 5 6 7 8 9]' * 10.^LTick;
    htick = (htick(:))';
    htick = htick (htick >= Tick(1) & htick <= Tick(end));
  end

end

return
```

A.7 SetXTickLabel and SetYTickLabel

These routines turn off the default tick labels and put in their place specified text strings. The text strings can contain TeX commands to create symbols in the new labels. For instance **\pi** will specify the character pi.

```
function SetXTickLabel (ax, dyp, labels)
% h = SetXTickLabel(ax, [dyp], labels)
% Set X-tick labels.
% This function sets X-tick labels using text strings.  The
% labels can include LaTeX-type strings.  Each label is
% centered below the corresponding X-tick value.
% ax     - axis handle (defaults to current axis)
% dyp    - distance from the bottom of the graph to the top
%          of the labels (in points), default 0.5 times the
%          font size.
% labels - string, array of strings or cell array containing
%          the labels
%
% Example: SetXTickLabel (gca, '0|\pi|2\pi');
%          SetXTickLabel (['0   '; '\pi '; '2\pi']);
%          SetXTickLabel ({'0'; '\pi'; '2\pi'});

% $Id: SetXTickLabel.m,v 1.12 2006/01/27 21:55:07 pkabal Exp $

if (nargin == 1)
  labels = ax;
  ax = gca;
  FontSizePt = Get_FontSize_Pt (ax);
  dyp = 0.5 * FontSizePt;
elseif (nargin == 2)
  labels = dyp;
  FontSizePt = Get_FontSize_Pt (ax);
  dyp = 0.5 * FontSizePt;
end

% Note:
%   Invoking this routine a second time on a given axis does NOT
%   erase the first set of tick labels.

%===========
% Create a cell array of strings for the labels
if (iscell (labels))
  lab = labels;          % Cell array of strings

elseif (size (labels, 1) > 1)
  lab = cellstr (labels); % String array; convert to cell array

else
  rem = labels;           % Single string; look for separators
  n = 0;
  lab = {};
  while (length (rem) > 0)
    n = n+1;
    [lab{n},rem] = strtok (rem, '|'); % Pick off pieces
    if (length (rem) > 1)
      rem(1) = [];        % Remove the '|' character
    end
  end

end

%==========
set (ax, 'XTickLabel', ' '); % Remove normal tick labels
                             % But leave a blank, so xlabel
                             % is positioned properly

% Find the conversion from points to data units
SF = SFdataXpt (ax);

% Put text strings below the tick marks at (x, y)
XTick = get (ax, 'XTick');  % Tick positions
XLim = get (ax, 'XLim');
YLim = get (ax, 'YLim');

if (strcmp (get (ax, 'YScale'), 'log'))
  yt = YLim(1) * 10^(-dyp * SF(2));
else
  yt = YLim(1) - dyp * SF(2);
end

NL = length (lab);

for i = 1:length(XTick)
  xt = XTick(i);
  if (xt >= XLim(1) & xt <= XLim(2))
    n = mod (i-1, NL) + 1;
    text (xt, yt, lab(n), ...
          'VerticalAlignment', 'cap', ...
            'HorizontalAlignment', 'center');
  end
end

return

% -----
function FontSizePt = Get_FontSize_Pt (h)
% Returns the font size in points

FUnits = get (h, 'FontUnits');

set (h, 'FontUnits', 'points');
FontSizePt = get (h, 'FontSize');

set (h, 'FontUnits', FUnits);

return
```

```
function SetYTickLabel (ax, dxp, labels)
% h = SetYTickLabel(ax, [dxp], labels)
% Set Y-tick labels.
% This function sets Y-tick labels using text strings.  The
% labels can include LaTeX-type strings.  Each label is
% centered beside the corresponding Y-tick value.
% ax     - axis handle (defaults to current axis)
% dxp    - distance from the side of the graph to the edge
%          of the labels (in points), default 0.5 times the
%          font size.
% labels - string, array of strings or cell array containing
%          the labels
%
% Example: SetYTickLabel (gca, '0|\pi|2\pi');
%          SetYTickLabel (['0   '; '\pi '; '2\pi']);
%          SetYTickLabel ({'0'; '\pi'; '2\pi'});

% $Id: SetYTickLabel.m,v 1.12 2006/03/03 16:28:46 pkabal Exp $

if (nargin == 1)
  labels = ax;
  ax = gca;
  FontSizePt = Get_FontSize_Pt (ax);
  dxp = 0.5 * FontSizePt;
elseif (nargin == 2)
  labels = dxp;
  FontSizePt = Get_FontSize_Pt (ax);
  dxp = 0.5 * FontSizePt;
end

% Note:
%   Invoking this routine a second time on a given axis does NOT
%   erase the first set of tick labels.

%===========
% Create a cell array of strings for the labels
if (iscell (labels))
  lab = labels;        % Cell array of strings

elseif (size (labels, 1) > 1)
  lab = cellstr (labels); % String array; convert to cell array

else
  rem = labels;        % Single string; look for separators
  n = 0;
  lab = {};
  while (length (rem) > 0)
    n = n+1;
    [lab{n},rem] = strtok (rem, '|'); % Pick off pieces
    if (length (rem) > 1)
      rem(1) = [];     % Remove the '|' character
    end
  end

end

%==========
% Find the conversion from points to data units
SF = SFdataXpt (ax);
```

```
YTick = get (ax, 'YTick');  % Tick positions
XLim = get (ax, 'XLim');
YLim = get (ax, 'YLim');
NL = length (lab);

% Y axis labels on the right-hand side
if (strcmp (get (ax, 'YAxisLocation'), 'right'))

% Put text strings to the right of the tick marks at (xt, yt)
  if (strcmp (get (ax, 'XScale'), 'log'))
    xt = XLim(2) * 10^(dxp * SF(1));
  else
    xt = XLim(2) + dxp * SF(1);
  end

  W = 0;
  for i = 1:length(YTick)
    yt = YTick(i);
    if (yt >= YLim(1) & yt <= YLim(2))
      n = mod (i-1, NL) + 1;
      h = text (xt, yt, lab(n), ...
                  'VerticalAlignment', 'middle', ...
                  'HorizontalAlignment', 'left');
      W = max (W, GetStrLen_Pt (h));
    end
  end

else

% Put text strings to the left of the tick marks at (xt, yt)
  if (strcmp (get (ax, 'XScale'), 'log'))
    xt = XLim(1) * 10^(-dxp * SF(1));
  else
    xt = XLim(1) - dxp * SF(1);
  end

  W = 0;
  for i = 1:length(YTick)
    yt = YTick(i);
    if (yt >= YLim(1) & yt <= YLim(2))
      n = mod (i-1, NL) + 1;
      h = text (xt, yt, lab(n), ...
                  'VerticalAlignment', 'middle', ...
                  'HorizontalAlignment', 'right');
      W = max (W, Get_StrLen_Pt (h));
    end
  end

  % Make a dummy YTickLabel filled with blanks
  % The goal is to fool ylabel into positioning itself to the
  % left of the Y-axis labels.
  % In 10pt Times, each blank is 3.15 pt wide.
  WBlank = 0.315;       % Normalized width of a blank
  FontSizePt = Get_FontSize_Pt (ax);
```

```
    NBlank = round (W / (WBlank * FontSizePt));
    set (ax, 'YTickLabel', repmat (' ', 1, NBlank));


    % If this doesn't work well
    % set (gca, 'YTickLabel', '  ') % Use blanks as needed

  end

  return

  % -----
  function [WPt, HPt] = Get_StrLen_Pt (h)
  % Returns the width and height of the text object in points

  Units = get (h, 'Units');

  set (h, 'Units', 'points');
  Extent = get (h, 'Extent');

  set (h, 'Units', Units);

  WPt = Extent(3);
  HPt = Extent(4);

  return

  % -----
  function FontSizePt = Get_FontSize_Pt (h)
  % Returns the font size in points

  FUnits = get (h, 'FontUnits');

  set (h, 'FontUnits', 'points');
  FontSizePt = get (h, 'FontSize');

  set (h, 'FontUnits', FUnits);

  return
```

A.8 Xaxis and Yaxis

These routines overlay the current plot with a transparent axes box. For **Xaxis** the axes box has essentially zero height; for **Yaxis** the axes box has essentially zero width. The overlaid axes then appear as a line with (unlabelled) tick marks on the current plot. The default axes are drawn through the origin of the current plot.

```
function h = Xaxis (y, varargin)
% h = Xaxis (y, <plot options>)
%
% Generate an X axis at the given y-value
% This function generates a new (nearly) zero height axis. The
% new axis is placed on top of the existing axis.
%   - position: y-value from the input value
%               x-range from the current axis
%   - ticks: tick labels are turned off. The top and bottom
%     X-axes overlap (appearing as a single axis) with the tick
%     marks extending to either side of the axis.
%
% Additional axes property values for this axis can be specified
% in the variable length argument list.
%   - 'Box': Set it to 'off' to get ticks only up or down
%   - 'XAxisLocation: With 'Box' set to 'off', setting this to
%     'top' gives downward ticks, while 'bottom' gives upward
%     ticks

% $Id: Xaxis.m,v 1.7 2006/05/31 01:47:11 pkabal Exp $

YLim = get (gca, 'YLim');
Pos  = get (gca, 'Position');
XTick = get (gca, 'XTick');

if (nargin < 1)
  y = 0;
end

if (y < YLim(1) | y > YLim(2))
  disp ('Xaxis: position off scale');
  h = [];
  return;
end

Peps = 1e-6;
yn = (y - YLim(1)) / (YLim(2) - YLim(1));
ym = yn - 0.5 * Peps;
yp = yn + 0.5 * Peps;

Pos(2) = Pos(2) + ym * Pos(4);
Pos(4) = Peps * Pos(4);

DL = YLim(2) - YLim(1);
YS = YLim(1);
YLim(1) = YS + ym * DL;
YLim(2) = YS + yp * DL;
```

```
h = BlankAxes;
set (h, 'Position', Pos, ...
    'Box', 'on', ...
    'XTick', XTick, ...
    'YLim', YLim, ...
    'YTick', [], ...
    'XAxisLocation', 'bottom', ...
    varargin{:});

return
```

```
function h = Yaxis (x, varargin)
% h = Yaxis (x, <plot options>)
%
% Generate a Y axis at the given x-value
% This function generates a new (nearly) zero width axes. The
% new axis is placed on top of the existing axis.
%   - position: x-value from the input value
%               y-range from the current axis
%   - ticks: tick labels are turned off. The left and right
%     Y-axes overlap (appearing as a single axis) with the tick
%     marks extending to either side of the axis.
%
% Additional axes property values for this axis can be specified
% in the variable length argument list.
%   - 'Box': Set to 'off' to get ticks only to the right or left
%   - 'YAxisLocation: With 'Box' set to 'off', setting this to
%     'left' gives rightward ticks, while 'right' gives leftward
%     ticks

% $Id: Yaxis.m,v 1.9 2006/05/31 01:47:18 pkabal Exp $

XLim = get (gca, 'XLim');
Pos  = get (gca, 'Position');
YTick = get (gca, 'YTick');

if (nargin < 1)
  x = 0;
end

if (x < XLim(1) | x > XLim(2))
  disp ('Yaxis: position off scale');
  h = [];
  return;
end

Peps = 1e-6;
xn = (x - XLim(1)) / (XLim(2) - XLim(1));
xm = xn - 0.5 * Peps;
xp = xn + 0.5 * Peps;
```

```
Pos(1) = Pos(1) + xm * Pos(3);
Pos(3) = Peps * Pos(3);



DL = XLim(2) - XLim(1);
XS = XLim(1);
XLim(1) = XS + xm * DL;
XLim(2) = XS + xp * DL;

h = BlankAxes;
set (h, 'Position', Pos, ...
    'Box', 'on', ...
    'XLim', XLim, ...
    'YTick', YTick, ...
    'XTick', [], ...
    'YAxisLocation', 'left', ...
    varargin{:});

return
```

A.9 XYclip

This routine (not used in the examples in the main text) removes line segments that are entirely outside of the plot region. Line segments that cross the boundary of the plot region are not affected. The axis scaling (Matlab **axis** command) must be set before calling **XYclip**.

```
function [u,v] = XYclip (x, y, Axes)
% Clip plot vectors
% This routine clips an array of plot vectors so that vectors
% that are completely outside the clipping rectangle are
% eliminated.
%  x - vector of X coordinates
%  y - vector of Y coordinates
%  Axes - 4 element vector specifying the clipping rectangle
%        [xmin xmax ymin ymax]. If this argument is missing, the
%        clipping rectangle is taken from the current axes
%        limits (use the axis command to set this before calling
%        this routine).

% $Id: XYclip.m,v 1.6 2006/01/31 03:08:32 pkabal Exp $

% Notes:
% - This is not a full clipping algorithm. Vectors straddling
%   the clipping rectangle are left in place. Points between
%   invisible segments are replaced by NaN.

N = length (x);
if (N ~= length(y))
  error ('XYclip: Unequal length vectors');
end

if (nargin <= 2)
  Figs = get (0, 'Children');
  if (isempty (Figs))
    fprintf ('>>> Warning, axis size and limits are
undefined\n');
  else
    XLimMode = get (gca, 'XLimMode');
    YLimMode = get (gca, 'YLimMode');
    if (strcmp (XLimMode, 'auto') | strcmp (YLimMode, 'auto'))
      fprintf ('>>> Warning, axis limits not explicitly set\n');
    end
  end
  Axes = axis;
end

% Set the location code for each point as the sum of
%  0 - The point is within the rectangle
%  1 - The point is to the Left of the rectangle
%  2 - The point is to the Right of the rectangle
%  4 - The point is Below the rectangle
%  8 - The point is Above the rectangle
% If the x is NaN, vpos is 0
vpos = (x < Axes(1)) + 2*(x > Axes(2)) + 4*(y < Axes(3)) + ...
```

```
        8*(y > Axes(4));
```

```
% Determine the visibility of each segment
% Segment i is between points i and i+1
%  0 - inside or partly inside the rectangle
%  1 - completely outside the rectangle
segvis = (bitand(vpos(1:N-1), vpos(2:N)) ~= 0);

% For the visibility of each point
%  0 - left and right segments are at least partially visible
%  1 - one side is visible, the other at least partially visible
%  2 - both sides invisible
% First point: segment to left is invisible
% Last point: segment to right is invisible
ptvis = [segvis 1] + [1 segvis];

% Set the output values
u = x;
v = y;
ind = (ptvis >= 2); % Points with adjacent invisible segments
u(ind) = NaN;
v(ind) = NaN;

ind2 = ((ind + [1 ind(1:N-1)]) >= 2);   % Redundant NaN's
u(ind2) = [];
v(ind2) = [];
Nu = length (u);
if (isnan(u(Nu)))
  u(Nu) = [];
  v(Nu) = [];
end

fprintf ('XYclip: No. points (in/out): %d/%d\n', N, ...\
         length (u));

return
```

A.10 XYmerge

This routine takes vectors of *x* and *y* values and deletes some of the points in those vectors. The algorithm determines whether a point can be omitted if the line without the point is sufficiently close to the point to be omitted. This way, this routine merges nearly co-linear line segments. The default tolerance is 1/720 inches (1/10 point). The axis scaling (Matlab **axis** command) and plot size (**SetPlotSize**) must be set before calling **XYmerge**. It uses the axis limits and the plot size to determine the mapping between data units and points for use in evaluating the error in omitting a point.

```
function [u,v] = XYmerge (x, y, EXY)
% Merge co-linear plot vectors
% This routine compresses an array of plot vectors by merging
% co-linear vectors.
%  x - vector of X coordinates
%  y - vector of Y coordinates
%  EXY - two element vector with the allowable error in X and Y
%        allowed when merging nearly co-linear segments. If
%        this parameter is missing, an error of 1/10 of a point
%        (maximum error 1/720 of an inch) is allowed. The
%        corresponding error in data units is based on the
%        current axis limits and axis sizes.
%
% Notes on Setting EXY:
% - The default axes size in Matlab is 434 by 342 pixels.
%   Using a conversion factor of 96 pixels per inch, this is
%   4.52 inches by 3.56 inches. Call these dimensions
%   Xsize and Ysize. The routine SetPlotSize can be used to
%   set the axes size to other values.
% - We will assume that the graph is to be plotted will be
%   imported into a document at full size.
% - Assume that the printout will occur on a printer with
%   resolution DPI (dots per inch). For instance a standard
%   laser printer resolution is 600 dpi.
% - The allowable error will be Edot (measured in printer
%   dots).
% - Let the data range in the X and Y directions be
%   [Xmin, Xmax] and [Ymin, Ymax]. Note that we may want to
%   print only part of the data range to "zoom" in on details.
%   If the whole data range is to be plotted, these values can
%   be found using the min and max operations.
% - Lets focus on the X direction. The total extent of the
%   plot in dots is Xsize * DPI. The allowable relative
%   error is Edot / (Xsize * DPI). In data units this is
%   (Xmax-Xmin) * Edot / (Xsize * DPI).
% - Assume an error of Edot=1 is acceptable and using the
%   default Xsize=4.52 inches and DPI = 600. Then the
%   X component of the error should be set to
%     EXY(1) = (Xmax-Xmin) / 2712.
% - Even easier, let this routine determine a reasonable
%   value for the acceptable error.
%   (a) Set the plot size (SetPlotSize). Choose the plot size
%       here so that it can be included in a document with no
%       further scaling.
%   (b) Call axis(Xmin,Xmax,Ymin,YMax) to set the plot limits.
%   (c) Call XYmerge to merge line segments (omitting the EXY
%       argument.
%   (d) Plot the data
%   (e) Set the plot limits again (plot wiped out the settings).
%   (f) Write the plot to a file (using WritePlot to ensure
%       no scaling of the plot).
%   (g) Include the plot into a document - don't scale the plot
%       while including it.

% $Id: XYmerge.m,v 1.13 2006/03/03 16:29:07 pkabal Exp $

% Notes:
% - The error criterion (when EXY is specified) is absolute
%   error. When used with data that is to be plotted on a
%   log scale, the linear data should converted to log before
%   being input to this routine. The returned data can then
%   be taken back to the linear domain for plotting.
% - The algorithm works by processing 3 points at a time.
%   Consider a line joining the end points of the triplet. The
%   length of the perpendicular distance from the middle point
%   to that line determines whether the middle point can be
%   omitted or not. The perpendicular distance has components
%   in the X and Y directions, and as such, even for equi-
%   spaced data, some tolerance should be allowed on the
%   X-error.
% - The algorithm is greedy but short-sighted. When it hits
%   a point at which the error in the line exceeds the
%   tolerance, it does not try to extend the line beyond
%   that point, when in fact there may be a viable longer
%   line.

N = length (x);
if (N ~= length(y))
  error ('XYmerge: Unequal length vectors');
end

if (nargin < 3)
  [DataPt, XScale, YScale] = SFdataXpt; % Data units per point
  ExA = 0.1 * DataPt(1);
  EyA = 0.1 * DataPt(2);
else
  XScale = 'linear';
  YScale = 'linear';
  ExA = EXY(1);
  EyA = EXY(2);
end

LogX = strcmp (XScale, 'log');
LogY = strcmp (YScale, 'log');
if (LogX)
  x = log10 (x);
end
if (LogY)
  y = log10 (y);
end
```

```
B = 1;

k = 0;

k = k + 1;
kv(k) = B;
while (B <= N-1)

  % B - is a base point
  % G - is a known good point (the line from B to G can be
  %     approximated by a single line)
  % T - is a test point beyond G - we will test the line from
  %     B to T and reset G to T if we are successful.
  % Infinite or NaN values for B, G and/or T result in OK = 0

  G = B + 1;
  step = 1;
  straddle = 0;
  Lx = length (x);
  while (1)
    T = G + step;
    if (T > Lx)   % Don't look beyond the array
      straddle = 1;
      OK = 0;
    else

      % Main test loop (loop over B+1:T-1, permuted)
      % The indices are arranged from the middle out to abort
      % the test as quickly as possible - Tests indicate a 25%
      % reduction in evaluations of Test3 over a sequential
      % search
      I = B+1:T-1;
      Ni = length (I);
      II = reshape ([I; fliplr(I)], 1, []);
      II = II(Ni+1:end);   % Search from middle out
      for (i = II)
        OK = Test3 (x(B), y(B), x(i), y(i), x(T), y(T), ExA,
EyA);
        if (~ OK)
          straddle = 1;
          break
        end
      end

    end

    %  (1) If we have found a T which does not work, we know
    %      the end point is in the interval G:T-1. Use a
    %      binary search (decreasing the step size) to find the
    %        end point.
    %  (2) If we have not found a T which does not work, keep
    %      looking by increasing the step size.
    if (straddle)
      if (OK)
        G = T;
      end
      if (step == 1)
```

```
        break
      end
      step = step / 2;
      continue
    else    % success, but not straddling the end point
      G = T;
      step = 2 * step;
      continue
    end
  end

  B = G;
  k = k + 1;
  kv(k) = B;
end

if (LogX)
  u = 10.^x(kv);
else
  u = x(kv);
end
if (LogY)
  v = 10.^y(kv);
else
  v = y(kv);
end

fprintf ('XYMerge: No. points (in/out): %d/%d\n', N, k);

return

%==========
function OK = Test3 (x1, y1, x2, y2, x3, y3, ExA, EyA)

% Consider 3 points.  Draw a straight line between the end
% points. The middle point will be skipped if the X and Y
% components of the perpendicular distance from the middle
% point to the straight line are smaller than given
% tolerances.
%
% The first step is to translate the axes so that (x(m),y(m))
% becomes (0,0).
%
%               [x2,y2]                 [p1,q1]
%                  o                        o
%                 / \                      / \
%                /   \                    /   \
%               /     o                  /     o
%              /   [x3,y3]              /   [p2,q2]
%     [x1,y1] o                   [0,0] o

% A rotation about the origin through an angle w (CCW)
% can be expressed as
%   [r] = [cos(w) -sin(w)] [p]
%   [s]   [sin(w)  cos(w)] [q].
% The perpendicular error at (p1,q1) can be determined by a
% rotation about (0,0) such that the line from the (0,0) to
% (p2,q2) is horizontal.  The rotation is through the angle -a,
% where
%   cos(a) = p2 / D,
%   sin(a) = q2 / D,
% and D = sqrt(p2^2 + q2^2).  The perpendicular error is the
% ordinate value of (p1,q1) after rotation,

%   errN = q1 cos(a) - p1 sin(a).

% The components of this error in the original X and Y
% directions can be found by projecting errN,
%   errX = sin(a) errN,
%   errY = -cos(a) errN.

% The test for the absolute X error is
%   |errX| > ExA.
% Or,
%   |sin(a) (q1 cos(a) - p1 sin(a))| > ExA
%
%    q2     p2      q2
%   |-- (q1 -- - p1 --)| > ExA
%    D      D       D
% or
%   |q2 (q1 p2 - p1 q2)| > D^2 ExA
% This rearrangement avoids possible divisions by zero.
%
% Similarly the check for the Y error is,
%   |p2 (q1 p2 - p1 q2)| > D^2 EyA

% The error region is a rectangle. It would be easy to change
% the error region to be an ellipse with the given errors
% as the axes.

% Any NaN in these expressions will result in OK = 0
p1 = x2 - x1;
q1 = y2 - y1;
p2 = x3 - x1;
q2 = y3 - y1;

D2 = p2^2 + q2^2;
err = q1 * p2 - p1 * q2;
OK = (abs(q2 * err) <= D2 * ExA) & (abs(p2 * err) <= D2 * EyA);

return
```

A.11 WritePlot

This routine writes a figure to a graphics file. Before writing the figure, plot parameters are set to freeze the plot axes. If the figure background colour is the default gray value, the figure background colour is set to transparent.

```
function WritePlot (h, filename, option)
% WritePlot ([FigureHandle,] filename[, option])
% Write a PostScript, PDF, or EMF file for a Matlab figure.
% FigureHandle - Optional figure handle.  If not specified, the
%                current figure is used.
% filename - If the file extension is '.eps' or '.ps', write
%            an encapsulated PostScript file.
%          - If the file extension is '.pdf', write a PDF file
%          - If the file extension is '.emf', write a Windows
%            Metafile.

% $Id: WritePlot.m,v 1.16 2006/01/27 21:05:23 pkabal Exp $

% Resolve the arguments
if (nargin == 0)
  h = gcf;
  filename = [];
  option = [];
elseif (nargin == 1)
  if (ishandle (h))
    filename = [];
  else
    filename = h;
    h = gcf;
  end
  option = [];
elseif (nargin == 2)
  if (ishandle (h))
    option = [];
  else
    option = filename;
    filename = h;
    h = gcf;
  end
end

% Return if no file name
if (strcmp (filename, ''))
  return;
end

% Check for the type of file (PS, PDF, or EMF)
[pathstr, name, ext, versn] = fileparts (filename);
PDF = strcmpi (ext, '.pdf');    % Ignore case
EMF = strcmpi (ext, '.emf');
PS = (~PDF & ~EMF);

% Warning message if 'Renderer' is not 'painters' (might result
% in a big file)
```

```
figure (h);
Renderer = get (h, 'Renderer');
if (~ strcmpi (Renderer, 'painters'))
  fprintf ('  Warning - "Renderer" property: "%s"\n', Renderer);
end

% - Set the figure color to transparent if it is the default
%   (gray) color
% - Set figure and axis options so that the saved plot is the
%   same size
%   as the plot on the screen
% - Set the resolution (in case 'Renderer' is not 'painter') to
%   300 dpi
FColor = get (gcf, 'Color');
FColor_Default = [0.8 0.8 0.8];
if (~ strcmpi (FColor, 'none'))
  if (FColor == FColor_Default)
    set (h, 'Color', 'none');    % Transparent figure
  end
end
set (h, 'PaperPositionMode', 'auto', 'InvertHardCopy', 'off');
set (gca, 'XTickMode', 'manual', 'YTickMode', 'manual', ...
      'ZTickMode', 'manual');

% Print the plot
if (EMF)
  print ('-dmeta', '-r300', option, filename);
elseif (PS)
  print ('-depsc', '-r300', option, filename);
elseif (PDF)
  print ('-dpdf', '-r300', option, filename);
end

return
```

A.12 Core Routines

A number of core routines are also listed. These are called by the Matlab routines listed above.

```
function nax = BlankAxes ()
% Create a new set of blank axes at the same position as the
% existing ones

% $Id: BlankAxes.m,v 1.3 2006/01/27 21:04:48 pkabal Exp $

nax = copyobj (gca, gcf);
delete (get (nax, 'Children'));

% No titles, labels or grid for the new axes
axes (nax); % New axes on top
xlabel ('');
ylabel ('');
zlabel ('');
title ('');

% The new axes are on top, so we make the plot area transparent
% so the original plot shows through
% The new axes handle visibility is set to off so that gca finds
% the old axes
% N.B., The new axes must remain on top; issuing axes(gca)
%       brings the old axes to the top.
set (nax, 'Color', 'none');
set (nax, 'HandleVisibility', 'off');
set (nax, 'Visible', 'on');

set (nax, 'XTick', []);
set (nax, 'YTick', []);
set (nax, 'ZTick', []);

set (nax, 'XTickLabel', []);
set (nax, 'YTickLabel', []);
set (nax, 'ZTickLabel', []);

return
```

```
function [SF, XScale, YScale] = SFdataXpt (ax)
% SFDataXPt - Return the conversion factor units/pt. The
% conversion factor is determined from the figure. If the
% an axis is in log units, the corresponding conversion
% factor is log10(units)/pt.
%  SF - data units / points
%  ax - Axes handle, defaults to gca

% This routine gets its information from a figure. A typical
% preamble before calling this routine is as follows.
%     SetPlotSize([xdim, ydim], 'centimeters');
%     axis([xmin, xmax, ymin, ymax]);
%     set (gca, 'XScale', 'log');  % For X log scale
%     set (gca, 'YScale', 'log');  % For Y log scale

% $Id: SFdataXpt.m,v 1.6 2006/01/27 21:09:51 pkabal Exp $

if (nargin == 0)
  ax = gca;
end

Figs = get (0, 'Children');
if (isempty (Figs))
  fprintf ('>>> Warning, axis size and limits are undefined\n');
else
  XLimMode = get (ax, 'XLimMode');
  YLimMode = get (ax, 'YLimMode');
  if (strcmp (XLimMode, 'auto') | strcmp (YLimMode, 'auto'))
    fprintf ('>>> Warning, axis limits not explicitly set\n');
  end
end

% Get the axis size in points
Units = get (ax, 'Units');
set (ax, 'Units', 'points');
PosPt = get (ax, 'Position');
set (ax, 'Units', Units);  % Restore the axes units

XLim = get (ax, 'XLim');
YLim = get (ax, 'YLim');

XScale = get (ax, 'XScale');
if (strcmp (XScale, 'log'))
  XLim = log10 (XLim);
end
YScale = get (ax, 'YScale');
if (strcmp (YScale, 'log'))
  YLim = log10 (YLim);
end

SF = [diff(XLim) diff(YLim)] ./ PosPt(3:4);

return
```