Department of Mathematics and Statistics

MATLAB ® Guide for Advanced Users

Converting a Diary into a Program (Input File)

You are recommended to create the input file to begin with, but in case you accidently lose your original program but retain your diary file, then rather than do a step by step edit, the following code will re-create your program.
For this example I will assume your diary file is called fred and that you wish your file to be called yourfile.m.

```
cat fred | grep \ >\ > | grep -v diary | grep \
-v K\ >\ > | grep -v 'On line' | cut -c4- >yourfile.m
```

Matlab Debugger

Matlab comes complete with a debugger for examining matlab functions, which has the following commands.

dbtype	List M-file with line numbers.
dbstop	Set breakpoint.
dbstep	Execute one or more lines.
dbcont	Resume execution.
dbdown	Change local workspace context.
dbup	Change local workspace context.
dbquit	Quit debug mode.
dbstack	List who called whom.
dbstatus	List all breakpoints.
dbclear	Remove breakpoint.
mexdebug	Enable MEX-file debugging.

Example. This example relies on hist.m and bar.m being the same as when this document was written. I have enclosed a copy of each at the end of this document.
To modify hist.m[1] to eliminate the gaps between the bars, you could proceed as follows.

1. Upon entering matlab, type
   ```
   >> dbtype hist
   ```
   This display a listing of the program which includes added line numbers.

[1] the one supplied with Matlab, type *type hist* to get a copy

2. Examine the program listing for key areas where you suspect the relevant code might occur. Locate where any loops or subprograms may occur.

3. Use the dbstop statement for each of the locations found in the previous step.
   e.g.
   >> dbstop at 23 in hist
   >> dbstop at 30 in hist
   >> dbstop at 46 in hist
   >> dbstop at 53 in hist
   >> dbstop at 57 in hist

4. Repeat the first three steps on any of the subprograms that you wish to examine.
   e.g. for the subprogram bar.m
   >> dbtype bar
   >> dbstop at 35 in bar
   >> dbstop at 51 in bar
   >> dbstop at 63 in bar
   >> dbstop at 79 in bar

5. Run hist with whatever parameters you wish to use.
   e.g. >> x=rand(1,100); >> hist(x) You will get output
   23 if nargin == 0
   This indicates that the debugger has stopped at line 23, as specified by the earlier dbstop statement.

6. At this point you have several options

   (a) Use the who, whos statements to view the variables.

   (b) Examine the variables.

   (c) Use the dbstack statement to locate where you are in the program heirarchy.

7. To procede on to the next line in the program type
   >> dbstep
   To procede to the next break (dbstop specification) in the program, type
   >> dbcont
   For this example we will use the dbcont statement twice to get to line 53, followed by the dbstep statement three times. You should now have the output
   49 nn(i,:) = sum(y <= xx(i));
   Typing dbstep twice will return you to back to line 53, this is due to you being within a “for” loop. This feature is handy for when you wish to test each step within a loop, but should you wish to get to the next line after the loop, it is important to have inserted a dbstop statement for directly after the for loop.
   i.e. dbstop at 57 in hist.
   Exit the “for” loop using dbclear, e.g. dbclear 53, then use a dbcont statement to get to.
   57 bar(x,nn,’hist’);
   Type *whos* to obtain a list of all the variables. Then type *dbstep* and again type whos. You will find that the list of variables have changed. Typing

K>> dbstack
will result in the output
In /usr/local/matlab/toolbox/matlab/plotxy/bar.m at line 36
In /usr/local/matlab/toolbox/matlab/plotxy/hist.m at line 57
This output informs you where the breakpoint is in bar.m and which line in hist.m took you to bar.m. This is handy when you have several calls to a subprogram such as bar.m. You now have the option of examining the current variables in bar.m or reexamining the variables from hist.m just prior to the line that took you into bar.m, this is done as follows
K>> dbup
This take you from the bar.m workspace to the hist.m workspace. Before proceding with a *dbstep* or *dbcont* statement, you should return to the bar.m workspace via
K>> dbdown

8. Should you wish to stop using the debugger, type
   K>> dbquit

9. You can now use the dbstop to add additional break points or you can use dbclear to eliminate some.

10. Once you have finished correcting/modifying the programs, type >> dbclear all
    This will allow you to re-run the programs without all the break points.

hist.m

```
function [no,xo] = hist(y,x)
%HIST  Histogram.
%   N = HIST(Y) bins the elements of Y into 10 equally spaced containers
%   and returns the number of elements in each container.  If Y is a
%   matrix, HIST works down the columns.
%
%   N = HIST(Y,M), where M is a scalar, uses M bins.
%
%   N = HIST(Y,X), where X is a vector, returns the distribution of Y
%   among bins with centers specified by X.
%
%   [N,X] = HIST(...) also returns the position of the bin centers in X.
%
%   HIST(...) without output arguments produces a histogram bar plot of
%   the results.

%   J.N. Little 2-06-86
%   Revised 10-29-87, 12-29-88 LS
%   Revised 8-13-91 by cmt, 2-3-92 by ls.
%   Copyright (c) 1984-97 by The MathWorks, Inc.
%   $Revision: 5.10 $  $Date: 1997/04/08 05:22:50 $

if nargin == 0
    error('Requires one or two input arguments.')
```

```
end
if nargin == 1
    x = 10;
end
if min(size(y))==1, y = y(:); end
if isstr(x) | isstr(y)
    error('Input arguments must be numeric.')
end
[m,n] = size(y);
if length(x) == 1
    miny = min(min(y));
    maxy = max(max(y));
    binwidth = (maxy - miny) ./ x;
    xx = miny + binwidth*(0:x);
    xx(length(xx)) = maxy;
    x = xx(1:length(xx)-1) + binwidth/2;
else
    xx = x(:)';
    miny = min(min(y));
    maxy = max(max(y));
    binwidth = [diff(xx) 0];
    xx = [xx(1)-binwidth(1)/2 xx+binwidth/2];
    xx(1) = miny;
    xx(length(xx)) = maxy;
end
nbin = length(xx);
nn = zeros(nbin,n);
for i=2:nbin
    nn(i,:) = sum(y <= xx(i));
end
nn = nn(2:nbin,:) - nn(1:nbin-1,:);
if nargout == 0
    bar(x,nn,'hist');
else
  if min(size(y))==1, % Return row vectors if possible.
    no = nn';
    xo = x;
  else
    no = nn;
    xo = x';
  end
end
```

bar.m

```
function [xo,yo] = bar(varargin)
%BAR Bar graph.
%    BAR(X,Y) draws the columns of the M-by-N matrix Y as M groups of N
%    vertical bars.  The vector X must be monotonically increasing or
%    decreasing.
%
%    BAR(Y) uses the default value of X=1:M.  For vector inputs, BAR(X,Y)
%    or BAR(Y) draws LENGTH(Y) bars.  The colors are set by the colormap.
%
%    BAR(X,Y,WIDTH) or BAR(Y,WIDTH) specifies the width of the bars. Valu
%    of WIDTH > 1, produce overlapped bars.  The default value is WIDTH=0
%
%    BAR(...,'grouped') produces the default vertical grouped bar chart.
%    BAR(...,'stacked') produces a vertical stacked bar chart.
%    BAR(...,LINESPEC) uses the line color specified (one of 'rgbymckw').
%
%    H = BAR(...) returns a vector of patch handles.
%
%    Use SHADING FACETED to put edges on the bars.  Use SHADING FLAT to
%    turn them off.
%
%    Examples: subplot(3,1,1), bar(rand(10,5),'stacked'), colormap(cool)
%              subplot(3,1,2), bar(0:.25:1,rand(5),1)
%              subplot(3,1,3), bar(rand(2,3),.75,'grouped')
%
%    See also HIST, PLOT, BARH.

%    C.B Moler 2-06-86
%    Modified 24-Dec-88, 2-Jan-92 LS.
%    Modified 8-5-91, 9-22-94 by cmt; 8-9-95 WSun.
%    Copyright (c) 1984-97 by The MathWorks, Inc.
%    $Revision: 5.26 $  $Date: 1997/04/08 06:44:19 $

error(nargchk(1,4,nargin));

[msg,x,y,xx,yy,linetype,plottype,barwidth,equal] = makebars(varargin{:});
if ~isempty(msg), error(msg); end

if nargout==2,
  warning(sprintf(...
     ['BAR with two output arguments is obsolete.  Use H = BAR(...) \n',.
      '          and get the XData and YData properties instead.']))
  xo = xx; yo = yy; % Do not plot; return result in xo and yo
else % Draw the bars
  cax = newplot;
  next = lower(get(cax,'NextPlot'));
  hold_state = ishold;
  edgec = get(gcf,’defaultaxesxcolor’);
  facec = ’flat’;
  h = [];
  cc = ones(size(xx,1),1);
  if ~isempty(linetype), facec = linetype; end
  for i=1:size(xx,2)
    numBars = (size(xx,1)-1)/5;
    for j=1:numBars,
       f(j,:) = (2:5) + 5*(j-1);
    end
    v = [xx(:,i) yy(:,i)];

    h=[h patch(’faces’, f, ’vertices’, v, ’cdata’, i*cc, ...
        ’FaceColor’,facec,’EdgeColor’,edgec)];
  end
  if length(h)==1, set(cax,’clim’,[1 2]), end
  if ~equal,
    hold on,
    plot(x(:,1),zeros(size(x,1),1),’*’)
  end
  if ~hold_state,
    % Set ticks if less than 16 integers
    if all(all(floor(x)==x)) & (size(x,1)<16),
      set(cax,’xtick’,x(:,1))
    end
    hold off, view(2), set(cax,’NextPlot’,next);
    set(cax,’Layer’,’Bottom’,’box’,’on’)
    % Switch to SHADING FLAT when the edges start to overwhelm the colors
    if size(xx,2)*numBars > 150, shading flat, end
  end
  if nargout==1, xo = h; end
end
```