Adjustments for Histograms of Logarithms of Quantized Values

J. Alex Stark and Stephen B. Hladky

Document composed November 3, 1999; filedate 1999/11/03
Revision 1.0

Abstract

When the logarithms of quantized data values such as duration times are described using histograms, the results can be a mess. The code described in this document provides adjustments for the visual display of such histograms. A demonstration is included which illustrates the problems with these histograms and how the adjustments perform.

Contents

1 Outline

1.1 The document

This documentation is not intended to be a polished work, but rather an aid to anyone who needs to display log-histograms of exponentially distributed values. As such, it is really a set of annotations rather than formal descriptions.

This document presents code which provides adjustments for histograms of logarithmic data which has been quantized. The code can be used in Octave (a program available which is free under the GPL licence) and Matlab (a commercial program produced by The Mathworks).

This source for this document is written using the DocStrip utility of LaTeX; it combines comments with the source code. Run `adjust.dtx` through LaTeX to generate the documentation; run `adjust.ins` through LaTeX to generate the program function files.

1.2 The algorithm

The algorithm(s) used to generate the adjustments are described in the paper *Adjustments for the Display of Quantized Ion Channel Dwell Times in Histograms with Logarithmic Bins*, by J. Alex Stark and Stephen B. Hladky, Biophysical Journal, in press.

1.3 The licence

1.4 The programs

The program functions provided here fall into three parts. First is the function `dismass` which provides all the adjustments. Second is the function `massplot` which is really a demonstration

of the problem we are trying to solve and the solution(s) which we propose. Third is a set of functions which enable this to work in different software packages.

2 dismass

2.1 Header

```
▷ ⟨*dismass⟩
▷ function [hout,dwell,Weight,dwellL,dwellH]=dismass(SScaleT,hin,method,histgen)
▷ %% function [hout, dwell,Weight, dwellL,dwellH] =
▷ %%                       dismass(SScaleT, hin, method, histgen)
▷ %%
▷ %% Troublesome SScaleT values include 19.5 and 19.25
▷ %%
▷ %% method is plotting method:
▷ %%   'equal':  [default] equal width bins: +/- 0.5 either side
▷ %%   'midpt':  logarithmic mid-pt between bin centres
▷ %%   'split':  split equal-midpt: equal below threshold if isolated points
▷ %%   'hybrid': old method, provided for compatibility
▷ %% split is a compromise of equal and midpt.
▷ %%
▷ %% dwellL and dwellH give lower and upper ranges of histogram dwell
▷ %% times (linear)
▷ %%
▷ %% If a 4th argument is given (of any sort), then all output arguments are
▷ %%    constructed for a neat histogram.
▷ %%
▷ %% **** If 4th argument and type equal, then the central dwell times are taken
▷ %%      from the non-discrete conversion, that is they are evenly separated.
▷ %%
▷ %% SScaleT =SBins/log10(MaxSampl) where SBins is length(hin)
▷ %% hin is the input histogram assumed to have been binned
▷ %% into SBins equal log width bins
▷ %%
▷ %% For COPYRIGHT information, see the accompanying documentation.
```

2.2 Argument processing

```
▷ if (nargin<3)
▷    method='equal';
▷ end
▷ method=lower(method);
▷
▷ [SBins InCols] = size(hin);
▷ if (SBins==1)
▷   hin=hin';
▷   [SBins InCols] = size(hin);
▷ end
```

2.3 Binning characterization

The vector `a_x` gives values of the function $a(x)$. This provides, for logarithmic bin x, the lowest linear bin that maps to x or higher under the mapping used to generate the histogram data. You may need to edit this section if you use these routines with your own data.

The first stage is to use a formula that should work well, but may be subject to rounding anomalies. The next stage will be to push values up or down as necessary. To test this correction procedure, `a_xRand` can be set to a non-zero value. The calculation of `a_x` is then deliberately damaged.

N.B. Matlab indices start at 1, so `a_x`(1) corresponds to log bin 0.

```
▷ a_xRand=0.0;
▷
▷ a_x = ceil(10.^(((0:SBins)-0.5)/SScaleT)+...
▷     (rand(1,SBins+1)-0.5)*a_xRand-1e-10);
```

While the introduction of the small factor 10^{-10} should avoid most rounding anomalies, we push the values of `a_x` up or down whenever there is an error. First we find any values which are too low, that is $b(a(x)) < x$, and need to be raised. You will rarely get the warning message.

```
▷ tmp = round(SScaleT*log10(a_x))<(0:SBins);
▷ a_x(tmp) = a_x(tmp)+1;
▷ if any(tmp)
▷   fprintf(2,'Dismass a(x) push ups: %d\n',a_x(tmp));
▷ end
```

To find values which need to be pushed down, we test for $b(a(x)-1) \geq x$ The term `a_x` $== 1$ avoids taking $\log(0)$. Also require that `a_x` > 1, which excludes the special case of $x = 0$, the only bin for which `a_x` ≤ 1

```
▷ tmp = round(SScaleT*log10(a_x-1+(a_x==1)))>=(0:SBins);
▷ tmp = tmp&(a_x>1);
▷ a_x(tmp) = a_x(tmp)-1;
▷ if any(tmp)
▷   fprintf(2,'Dismass a(x) push downs: %d\n',a_x(tmp));
▷ end
```

2.4 Bin centres

For the normal case (`nargin` < 4, or `nargin` $= 4$ and `method` $\neq$ `'equal'`), `dwell` equals the geometric mean of the first and last dwell times mapping to the log bin. This encompasses the simple situation when one linear bin maps to a log bin: `dwell` then equals the linear bin number.

If `nargin` ≥ 4 and `method` $=$ `'equal'`, which means that a staircase histogram is to be generated, then the central dwell times are taken from the non-discrete conversion: they are evenly separated.

```
▷ if ((method(1)=='e') & (nargin>3))
▷    dwell=10.^(((0:SBins-1)')/SScaleT);
▷ else                % normal situation
▷    dwell = sqrt(a_x(1:SBins).*(a_x(2:SBins+1)-1))';  % column vector
▷ end
```

2.5 Durations mapped to bin

The calculation of the number of linear bins that map to log bin x is simple, and is explained in the Biophysical Journal paper.

`ActiveBins` is a vector of same length as `a_x` with 1 when the bin can receive counts and 0 when it cannot.

```
▷ s_x = diff(a_x)';     % column vector
▷ ActiveBins = s_x>0;
▷ NWgt=sum(ActiveBins);
```

2.6 Bin range boundaries

Construct the ranges (`dwellL` to `dwellH`) of durations which we think are represented by each bin. `GapAbove` is a logical vector which is true if there exists a rounding scheme in which there would be an empty bin for the next duration.

`dwellP` is a vector containing the actual dwell times to plot. `HFac` and `LFac` are factors to add and subtract 1/2 log bin width on the log scale.

```
▷ HFac=10.^(0.5/SScaleT);
▷ LFac=10.^(-0.5/SScaleT);
▷
▷ dwellP = dwell(ActiveBins);
```

First, we cover the 'equal' method of histogram construction. Bars are of equal size. The problem with this is that the centres are in the wrong place.

```
▷ if (method(1)=='e')
▷    dwellH=dwellP*HFac;
▷    dwellL=dwellP*LFac;
▷    GapAbove=[s_x([0==1;ActiveBins(1:SBins-1)])==0;0];
```

In the 'midpt' method we put boundaries at the logarithmic mid-point between 'centres'. This is the traditional form of histogram, but the viewer is likely to get a false impression: these bin boundaries are arbitrary but likely to be interpreted as meaningful.

```
▷ elseif (method(1)=='m')
▷    dwellH = [a_x(2:SBins+1)'-0.5];
▷    dwellH = dwellH(ActiveBins);
▷    dwellL = [dwellP(1)*LFac; dwellH(1:NWgt-1)];
```

This method, 'split', is a combination of the equal and mid-point methods. First we evaluate the upper and lower boundaries for bins as in the mid-point method. We then find a threshold which determines which bins are 'isolated'. There is a gap between bins if a rounding scheme exists which would place an empty bin between. Another way of looking at this is that bins x_1 and x_2 are contiguous if

$$\omega \log_{10}(a(x_2)) - \omega \log_{10}(a(x_1+1)-1) \leq 1 \tag{1}$$

More simply, a duration maps to an isolated bin (that is, with a gap above it as well as below) if it is less than

$$\frac{1}{10^{1/\omega}-1} \tag{2}$$

When there is a gap, we put the boundaries at half a logarithmic bin width from the highest and lowest durations mapped to the bin.

```
▷ elseif (method(1)=='s')
▷    dwellH = [a_x(2:SBins+1)'-0.5];
▷    dwellH = dwellH(ActiveBins);
▷    dwellL = [dwellP(1)-0.5; dwellH(1:NWgt-1)];
▷
▷    thresh = round(SScaleT*log10(  ceil(1/(10.^(1/SScaleT)-1))-1 ));
▷    tmp=logical([ones(thresh+1,1);zeros(SBins-thresh-1,1)]);
▷
▷    GapAbove=tmp(ActiveBins);
▷    dwellH(GapAbove) = (dwellH(GapAbove)-0.5)*HFac;
▷    GapBelow = [1;GapAbove(1:NWgt-1)];
▷    dwellL(GapBelow) = (dwellL(GapBelow)+0.5)*LFac;
```

This 'hybrid' method is really only included for compatibility: it doesn't necessarily work properly.

```
▷ elseif (method(1)=='h')
▷    dwellH = [a_x(2:SBins+1)'-0.5];
▷    dwellH = dwellH(ActiveBins);
▷    dwellL = [dwellP(1)*LFac; dwellH(1:NWgt-1)];
▷
▷    tmp=[s_x(2:SBins);0];
▷    dwellH(tmp(ActiveBins)==0) = dwellP(tmp(ActiveBins)==0)*HFac;
▷    GapAbove = tmp(ActiveBins)==0;
▷    tmp=[0; s_x(1:SBins-1)];
▷    dwellL(tmp(ActiveBins)==0) = dwellP(tmp(ActiveBins)==0)*LFac;
▷ else
▷    error('Unknown dwell time range method.');
▷ end
```

2.7 Count rescaling factor

We work with inverse of weight, which can be zero. The equation is described in the paper. For longer durations, the weight factor approaches unity.

```
▷ IWgtVec = s_x(ActiveBins)./(dwellH-dwellL)...
▷    .*(log10(dwellH)-log10(dwellL));
```

Calculate the adjusted values for the bins that can contain counts.

```
▷ if (InCols==1)                        % if input is a vector
▷    hout = hin(ActiveBins)./IWgtVec;
▷ else                                  % for matrix input
▷    hout = hin(ActiveBins,:)./(IWgtVec*ones(1,InCols));
▷ end
```

2.8 Build traditional histogram

This is for the user's convenience. We reprocess the output arguments so as to make plotting simple. See the `massplot` examples for usage. Basically, if there is no 4th argument, only return the non-zero or adjusted dwell times. If there is such an argument, we part-build a staircase.

```
▷ if (nargin<4)
▷    dwell =dwell(ActiveBins);
▷ else
▷    if ((method(1)~='s') & (method(1)~='h') & (method(1)~='e'))
▷       GapAbove = [];
▷       % For 'split' etc. provided above
▷    end
▷    Itmp=logical([GapAbove'*0+1; GapAbove']);
▷    tmpH=dwellH;
▷    tmpL=dwellL;
▷
▷    dwellH=[tmpH';tmpL(2:NWgt)' 0];
▷    dwellH = dwellH(:);
▷    dwellH = dwellH(Itmp);
▷    dwellL=[tmpL';tmpH(1:NWgt)'];
▷    dwellL = dwellL(:);
▷    dwellL = dwellL(Itmp);
▷    hout = [hout'; hout'*0];
▷    hout = hout(:);
▷    hout = hout(Itmp);
▷ end
▷ if (nargout>2)
▷    Weight=zeros(size(s_x));
▷    Weight(ActiveBins)=IWgtVec;
▷ end
▷ %  hout = sqrt(hout);
▷ ⟨/dismass⟩
```

3 massplot

This demonstration generates some synthetic data and uses `dismass` to adjust the results. We also include an example with two components.

3.1 Header

In addition to the usual housekeeping, we do some version checking and give `SScaleT` a helpful default value.

```
▷ ⟨*massplot⟩
▷ function massplot(SScaleT)
▷ %% function massplot(SScaleT)
▷ %%
▷ %% This function demonstrates problems with histograms of logarithmic
```

```
%% durations and presents some solutions.
%%
%% Troublesome SScaleT values include 19.385 and 40.25
%%
%% For COPYRIGHT information, see the accompanying documentation.
if exist('OCTAVE_VERSION')
  InMatlab=0;
else
  InMatlab=1;
end

format compact

errorplot = (1==1);

if nargin<1
  if errorplot
    SScaleT=40.25;
  else
    SScaleT=19.385;
  end
end
```

3.2 Generate simple histogram

This generates ideal and ideal-empirical histograms for a single exponential distribution of durations. The latter is designed to give the average (expected) values of counts.

First set parameters. The mean for the component is `tmean`, the maximum plotted value `tmax`, the resolution of the ideal function is `ixdiv` and the quality (versus) speed of the approximation is determined by `tres`.

```
tmean = 70;
tau=tmean;
r=1-1/tau;
tmax=1000;

xmax=floor(SScaleT*log10(tmax));   % number of log bins -1
ixdiv=10;

tres = 10;
```

The idea we use is that, to generate random variables with an exponential distribution over t, take logs of uniform random variables between 0 and 1. Therefore we define a vector with equally spaced values of $\exp(-t/\texttt{tmean})$ then use this to define a time vector with values whose number in any interval is proportional to the value of $\exp(-t/\texttt{tmean})$ for that interval.

```
u=([1/tres:1/tres:tmax]'-0.5/tres)/tmax;
t = -tmean*log(u);
ft=t*0+1/tres/tmax;  % same length as t, all entries the same

```

```
▷ x = (0:xmax)/SScaleT;
▷ ix=(0:1/ixdiv:xmax)/SScaleT;
▷ xsel=1:ixdiv:length(ix);
▷
▷ ifhist = ix-log10(tmean);
▷                                % Generates ideal pdf for base10 logs
▷ ifhist = log(10)*10.^(ifhist-10.^ifhist/log(10));
```

We have a set of values t with the required distribution, and so we now histogram them into bins. *The first rounding operation is key to the whole exercise!* The second rounding operation means that we bin by rounding.

Scale by `tres` $\times$ `tmax`, which is the number of entries used to build `fhist`. The sum of the values in `fhist` is therfore 1, as in a pdf.

```
▷ tmp = round(t);
▷
▷ fhist = hist(round(SScaleT*log10(tmp(tmp>0))),(0:xmax)-0.1)/tres/tmax;
▷
▷ ex=10.^(x);
▷ eix=10.^(ix);
```

Whenever we plot `fhist`, we do so in the manner of a pdf. Therefore it is necessary to scale down by the bin width w, where $w = 1/$`SScaleT`. An 'integral' of the histogram will then evaluate to 1.

3.3 Raw data

Compare the raw corrupted values to the true underlying distribution which they should represent.

```
▷ if InMatlab
▷   figure(1)
▷ end
▷ plot(ex,fhist*SScaleT,'o',eix,ifhist,'b-')
▷
▷ prntmass(1,'fluct')
```

3.4 Unadjusted traditional histogram

The traditional histogram switches either side of the true distribution.

```
▷ if InMatlab
▷   figure(2)
▷ end
▷ plot([ex(1)*10.^(-1/2/SScaleT) ex(floor(1:0.5:xmax+1))*10.^(1/2/SScaleT)],...
▷     fhist(floor(1:0.5:xmax+1.5))*SScaleT)
▷
▷ prntmass(2,'ftrad')
```

3.5 Adjusted values

The adjustments clean up the picture considerably.

```
▷ [a,b,Weight,l,h]=dismass(SScaleT,fhist,'e');
▷
▷ if InMatlab
▷   figure(3)
▷ end
▷ plot(b,a,'o',eix,ifhist,'b-')
▷
▷ prntmass(3,'adjpt')
```

3.6 First error plot

We might like to be able to place bounds on the correction factors, but the following example shows this not to be ideal. The second example shows that bounds on the deviations from true values are simpler.

The weights returned by `dismass` are really inverse weights, some of which are likely to be zero. In this example we plot the deviations of the correction factors from unity, and we do so on a log-log plot. In this case, a unity scale factor already incorporates scaling by `SScaleT`.

The bounds are,

$$\frac{\phi^2/2+1}{\phi b-1} \tag{3}$$

and

$$\frac{1}{\phi b+1} \tag{4}$$

where b is the duration and

$$\begin{aligned} \phi &= 10^{1/(2\omega)} - 10^{-1/(2\omega)} \\ &\approx \ln(10)/\omega \end{aligned} \tag{5}$$

when ω is large, and hence the bin width is small.

```
▷ NWeight = Weight*SScaleT;
▷
▷ if errorplot
▷   if InMatlab
▷     figure(4)
▷   end
▷
▷   DegFac = (10.^(1/2/SScaleT)-10.^(-1/2/SScaleT));
▷   fprintf(2,'Ignore any warnings: positive errors in red.\n');
▷   plot(b,((1./NWeight(NWeight>0)-1)),'ro', ...
▷       b,(-(1./NWeight(NWeight>0)-1)),'bo',...
▷       b,(1./(b*DegFac)),'m-',...
▷       b,(1./(b*DegFac-1)*(DegFac*DegFac+2)/2),'r-',...
▷       b,(1./(b*DegFac+1)),'b-')
▷
▷   prntmass(-4,'error1')
```

3.7 Second error plot

If, instead, we consider the factor by which counts deviate from their true value, we find that the bounds are,

$$\frac{\phi^2/2+1}{\phi^2/2+\phi b} \tag{6}$$

and

$$\frac{1}{\phi b} \tag{7}$$

These are almost identical. Up to $b = 1/\phi$ the second is more conservative than the first, and even then the difference is small. Therefore the simpler second bound is a good approximate bound. In fact, it may well be that the extra conservatism for $b < 1\phi$ is unnecessary. Hence the first bound could be universal, whereas the second is a more useful general rule.

```
▷   if InMatlab
▷     figure(5)
▷   end
▷
▷   plot(b,(NWeight(NWeight>0)-1),'bo', ...
▷       b,(1-NWeight(NWeight>0)),'ro',...
▷       b,(1./b/DegFac),'b-',...
▷       b,(1./(2*b*DegFac+DegFac*DegFac)*(DegFac*DegFac+2)),'r-')
▷
▷   prntmass(-5,'error2')
▷
▷ end
```

Note, however, that there are often only blue dots for $b < 1/\phi$. This arises because $1/\phi$ always lies between the isolated and contiguous bins in the 'split'-style histogram.

3.8 Traditional histogram

The obvious traditional style of histogram, with boundaries midway between centres, is potentially very misleading. The fact that the data has considerable variation is hidden.

```
▷ [a,b,Weight,l,h]=dismass(SScaleT,fhist,'m');
▷
▷ nmass = length(b);
▷ hx=[l h]';
▷ hx=hx(:);
▷ hy=[a a]';
▷ hy=hy(:);
▷
▷ if InMatlab
▷   figure(4)
▷ end
▷ plot(hx,hy,'r-',eix,ifhist,'b-')
▷
▷ prntmass(4,'adjtrad')
```

3.9 Equal-size histogram

It's no good dividing up the duration axis evenly, since this puts bars in the wrong place.

```
▷ [a,b,Weight,l,h]=dismass(SScaleT,fhist,'e',1);
▷
▷ nmass = length(b);
▷ hx=[l h]';
▷ hx=hx(:);
▷ hy=[a a]';
▷ hy=hy(:);
▷
▷ if InMatlab
▷   figure(5)
▷ end
▷ plot(hx,hy,'r-',eix,ifhist,'b-')
▷
▷ prntmass(5,'adjehist')
```

3.10 Mixed histogram

A mixed scheme uses isolated bars, correctly centred, when the counts are (potentially) isolated, and uses contiguous steps when the bins are contiguously occupied.

```
▷ [a,b,Weight,l,h]=dismass(SScaleT,fhist,'s',1);
▷
▷ nmass = length(b);
▷ hx=[l h]';
▷ hx=hx(:);
▷ hy=[a a]';
▷ hy=hy(:);
▷
▷ if InMatlab
▷ figure(6)
▷ end
▷ plot(hx,hy,'r-',eix,ifhist,'b-' )
▷
▷ prntmass(6,'adjhist')
```

3.11 Two-component illustration

This example is of a mixture of two exponentials. In this case we draw `Nrands` durations from the distribution and bin the rounded values into a log histogram.

```
▷ rand('seed',7);
▷ dfhist = zeros(xmax+1,1);
▷
▷ tau=[70 4];
▷ qq=0.35;  % amount of dist in component 2
▷ Nrands = 3000;
▷
```

```
▷ t = -tau(1+(rand(Nrands,1)<qq))' .* log(rand(Nrands,1));
▷
▷ tmp = round(t);
▷ ft=t*0+1/Nrands;
▷
▷ dfhist = hist(round(SScaleT*log10(tmp(tmp>0))),(0:xmax)-0.1)/Nrands;
▷
▷ ifhistA = ix-log10(tau(1));
▷ ifhistB = ix-log10(tau(2));
▷ ifhist = log(10)*qq*10.^(ifhistB-10.^(ifhistB)/log(10)) + ...
▷     log(10)*(1-qq)*10.^(ifhistA-10.^(ifhistA)/log(10));
▷
▷ [a,b,Weight]=dismass(SScaleT,dfhist);
```

We plot everything together.

```
▷ if InMatlab
▷   figure(7)
▷ plot(ex,dfhist*SScaleT,'o', ...
▷     b,a,'.',  ...
▷     eix,ifhist,'-')
▷ ch=get(gca,'children');
▷ set(ch(2),'markersize',14);
▷ else
▷ plot(ex,dfhist*SScaleT,'o', ...
▷     b,a,'mx',  ...
▷     eix,ifhist,'-')
▷ end
▷ prntmass(7,'dblt')
▷ ⟨/massplot⟩
```

4 Octave functions

4.1 `logical`

Our version of Octave doesn't have a 'logical' casting function.

```
▷ ⟨*logical⟩
▷ function y=logical(x);
▷ %% function y=logical(x)
▷ %%
▷ %% This function is simply for compatibility with the casting function
▷ %% logical in Matlab
▷ %%
▷ %% For COPYRIGHT information, see the accompanying documentation.
▷
▷ y=x;
▷ ⟨/logical⟩
```

4.2 prntmass

```
▷ ⟨*oprntmass⟩
▷ function prntmass(handle, name)
▷ %% function prntmass(handle, name)
▷ %%
▷ %% Tidy up plots, doing different things depending on the handle
▷ %% number
▷ %%
▷ %% For COPYRIGHT information, see the accompanying documentation.
▷ errorplot = handle<0;
▷ handle = abs(handle);
▷
▷ if errorplot
▷   gset yrange [0.001:100]
▷   gset logscale xy
▷   grid
▷   replot
▷
▷   fprintf(2,'Pause for keypress...');
▷   pause();
▷   fprintf(2,'\n');
▷ else
▷   gset logscale x
▷   gset xrange [1:1000]
▷   grid
▷
▷   if (handle==7)
▷     gset yrange [0:1.49]
▷   else
▷     gset yrange [0:1.14]
▷   end
▷   replot
▷
▷   fprintf(2,'Pause for keypress...');
▷   pause();
▷   fprintf(2,'\n');
▷ end
▷ ⟨/oprntmass⟩
```

5 Matlab functions

5.1 prntmass

```
▷ ⟨*mprntmass⟩
▷ function prntmass(handle, name)
▷ %% function prntmass(handle, name)
▷ %%
▷ %% Tidy up plots, doing different things depending on the handle
▷ %% number
```

```
%%
%% For COPYRIGHT information, see the accompanying documentation.
errorplot = handle<0;
handle = abs(handle);

sbhmod=1;
mylwd=0.25;

if sbhmod
   if (handle==1)
      ylabelstr='Scaled counts & pdf';
   elseif (handle==7)
      ylabelstr='Scaled counts, estimated pdf values & pdf';
   else
      ylabelstr='Estimated pdf values & pdf';
   end
else
   if (handle==1)
      ylabelstr='Scaled counts / bin';
   elseif (handle==7)
      ylabelstr='Scaled counts, adjusted counts';
   else
      ylabelstr='Adjusted counts / bin';
   end

   ylabelstr = [ylabelstr ' & pdf'];
end

figure(handle);

if errorplot
  set(gca,'xscale','log')
  set(gca,'yscale','log')
  grid
else
set(gca,'xscale','log');
set(gca,'fontsize',17);
set(gca,'xlim',[1e0 1e3]);
if (handle==7)
  ytop=1.49;
else
  ytop=1.14
end
set(gca,'ylim',[0 ytop]);
xlabel('log_{10}(dwell time / sample interval)');
set(gca,'xticklabelmode','manual');
set(gca,'xticklabel',[0 1 2 3]);
ylabel(ylabelstr);
grid
```

```
▷ ch=get(gca,'children');
▷ set(ch(strcmp(get(ch,'type'),'line')),'linewidth',mylwd)
▷ aa=[text(1,ytop,'1') text(10,ytop,'10') ...
▷         text(100,ytop,'100') text(1000,ytop,'1000')];
▷ set(aa,'fontsize',14,'verticalalignment','bottom',...
▷         'horizontalalignment','center');
▷ end
▷
▷ fprintf(2,'Pause for keypress...');
▷ pause();
▷ fprintf(2,'\n');
▷
▷ %%set(gca,'Position',[0.13 0.11 0.775 0.815]+[0 1 0 0]*0.05)
▷ %%eval(['print -painters -deps ' name '.eps'])
▷ ⟨/mprntmass⟩
```