Table of Contents

 1
CHANGING SAMPLING RATES & INTERPOLATING 2
downsample.m: Downsample by an integer factor 3
decimate.m: Downsample with a low pass filter 5
Repeat but with two frequency components split by new Nyquist 7
Upsample 10
Interpolatation 11
Resample - change sampling frequency by rational number - or provide sample positions 23
TREND ANALYSIS 24
Transformations 25
Curve fitting - later 25
Linear filter 25
Slutsky-Yule effect 27
2.6 Analysis of seasonal data 28
Autocorrelation 28
sine wave 28
white noise 29
trend 30
outlier 31
What about sine with noise? 32
Outlier removal 33
Missing data 34
Can we detrend it? 35
Can we filter it - first a moving-average? 36
Now try a butterworth filter 37
Can we compute FFT? 38
Filling gaps 39
use nanmean! 41
Complicated example 43
What about non-linear trends? 47


```
figure
```

CHANGING SAMPLING RATES & INTERPOLATING

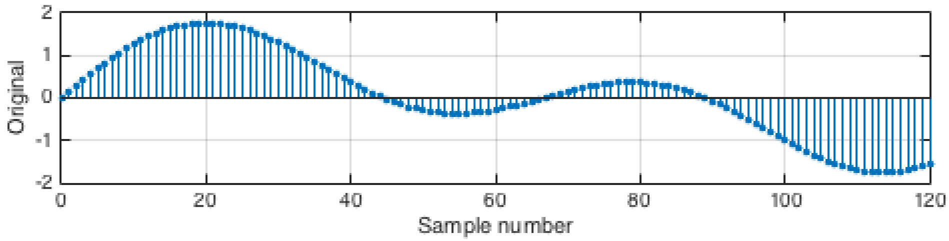
There are a lot of functions related to this in MATLAB Let's figure some of them out

Signal as sum of 30 & 60 Hz sines for 1 second at sampling rate 4kHz

```
fs = 4000;
t = 0:1/fs:1;
f1=30;
f2=60;
x = sin(2*pi*f1*t) + sin(2*pi*f2*t);
```

plot original signal

```
subplot(3,1,1)
stem(0:120,x(1:121),'filled','markersize',3)
grid on
xlabel('Sample number')
ylabel('Original')
```



downsample.m: Downsample by an integer factor

```
help downsample
```

```
 DOWNSAMPLE Downsample input signal.
    DOWNSAMPLE(X,N) downsamples input signal X by keeping every
    N-th sample starting with the first. If X is a matrix, the
    downsampling is done along the columns of X.

    DOWNSAMPLE(X,N,PHASE) specifies an optional sample offset.
    PHASE must be an integer in the range [0, N-1].

    % Example 1:
    %   Decrease the sampling rate of a sequence by 3.

    x = [1 2 3 4 5 6 7 8 9 10];
    y = downsample(x,3)

    % Example 2:
    %   Decrease the sampling rate of the sequence by 3 and add a
    %   phase offset of 2.

    x = [1 2 3 4 5 6 7 8 9 10];
    y = downsample(x,3,2)
```

```
% Example 3:
%   Decrease the sampling rate of a matrix by 3.

x = [1 2 3; 4 5 6; 7 8 9; 10 11 12];
y = downsample(x,3)

See also UPSAMPLE, UPFIRDN, INTERP, DECIMATE, RESAMPLE.

Reference page in Help browser
   doc downsample
```

downsample by factor of 3

```
n = 1:10
f=downsample(n,3)
```

```
n =

     1     2     3     4     5     6     7     8     9    10


f =

     1     4     7    10
```

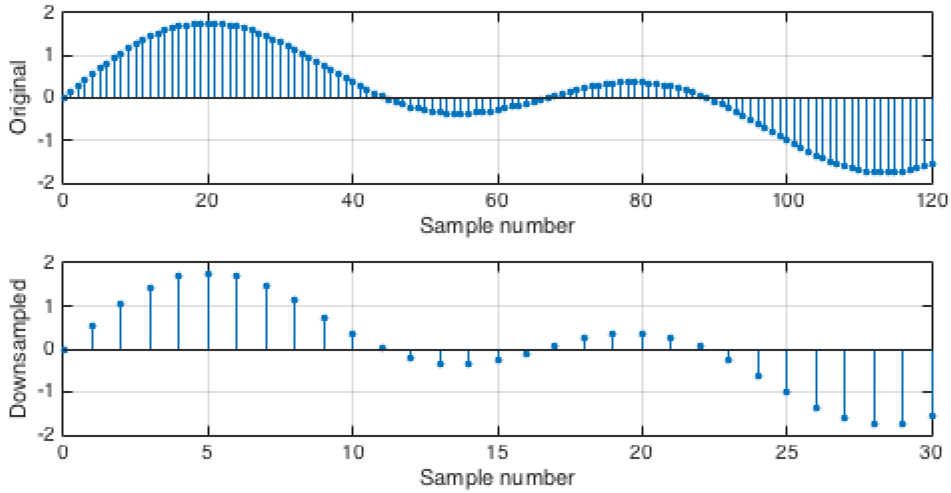
add a sample of 2

```
f=downsample(n,3,2)
```

```
f =

     3     6     9
```

try on signal - downsample by factor 4

```
y = downsample(x,4);

% plot result
subplot(3,1,2)
stem(0:30,y(1:31),'filled','markersize',3)
grid on
xlabel('Sample number')
ylabel('Downsampled')
```



decimate.m: Downsample with a low pass filter

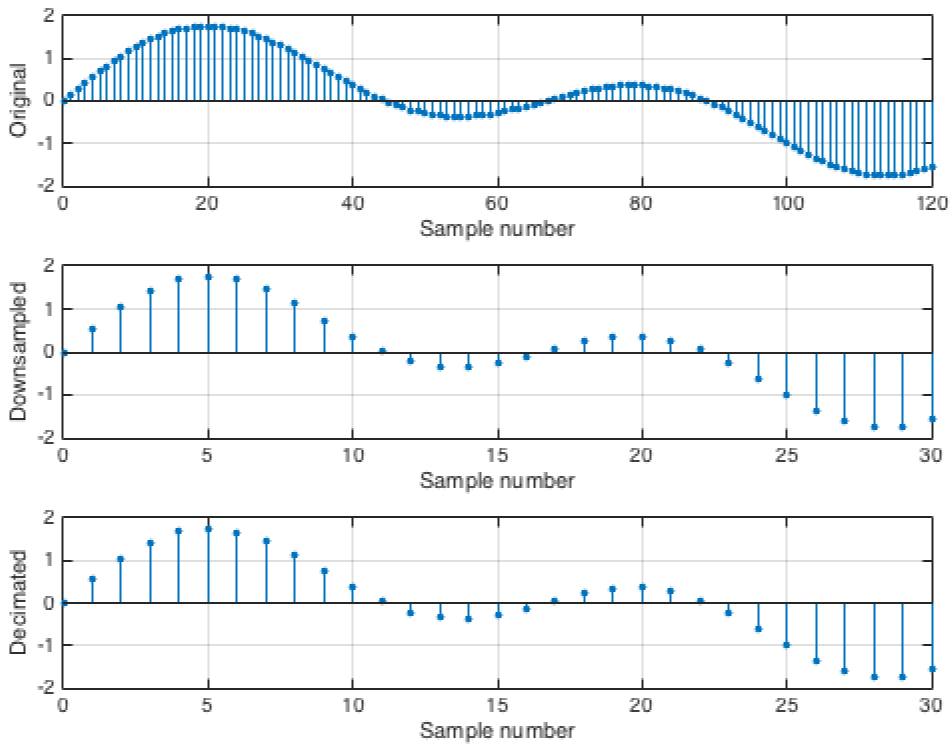
```
help decimate
```

```
 DECIMATE Resample data at a lower rate after lowpass filtering.
    Y = DECIMATE(X,R) resamples the sequence in vector X at 1/R times
 the
    original sample rate.  The resulting resampled vector Y is R times
    shorter, i.e., LENGTH(Y) = CEIL(LENGTH(X)/R). By default, DECIMATE
    filters the data with an 8th order Chebyshev Type I lowpass filter
 with
    cutoff frequency .8*(Fs/2)/R, before resampling.

    Y = DECIMATE(X,R,N) uses an N'th order Chebyshev filter.  For N
 greater
    than 13, DECIMATE will produce a warning regarding the
 unreliability of
    the results.  See NOTE below.

    Y = DECIMATE(X,R,'FIR') uses a 30th order FIR filter generated by
    FIR1(30,1/R) to filter the data.

    Y = DECIMATE(X,R,N,'FIR') uses an Nth FIR filter.

    Note: For better results when R is large (i.e., R > 13), it is
```

```
   recommended to break R up into its factors and calling DECIMATE
several
   times.

   EXAMPLE: Decimate a signal by a factor of four
   t = 0:.00025:1;  % Time vector
   x = sin(2*pi*30*t) + sin(2*pi*60*t);
   y = decimate(x,4);
   subplot(1,2,1);
   stem(x(1:120)), axis([0 120 -2 2])   % Original signal
   title('Original Signal')
   subplot(1,2,2);
   stem(y(1:30))                        % Decimated signal
   title('Decimated Signal')

   See also DOWNSAMPLE, INTERP, RESAMPLE, FILTFILT, FIR1, CHEBY1.

   Reference page in Help browser
      doc decimate
```

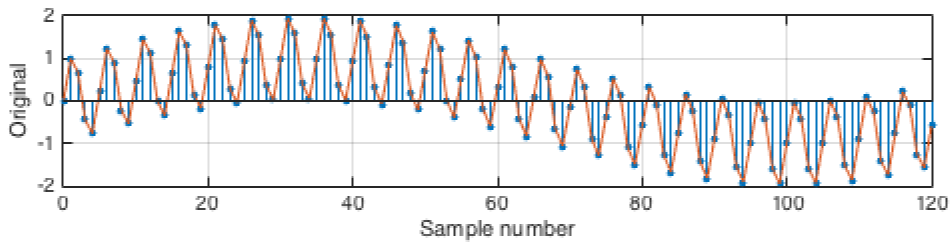
try on signal - decimate by factor 4

```
y = decimate(x,4);
subplot(3,1,3)
stem(0:30,y(1:31),'filled','markersize',3)
grid on
xlabel('Sample number')
ylabel('Decimated')
```



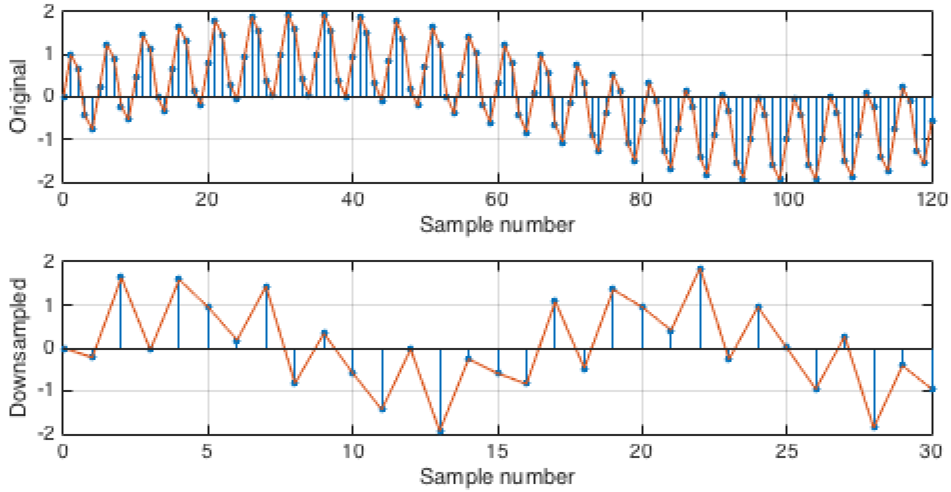
Repeat but with two frequency components split by new Nyquist

```
fs = 4000;
t = 0:1/fs:1;
f1=30;
f2=800;
x = sin(2*pi*f1*t) + sin(2*pi*f2*t);

figure
subplot(3,1,1)
stem(0:120,x(1:121),'filled','markersize',3)
hold on
plot(0:120,x(1:121))
grid on
xlabel('Sample number')
ylabel('Original')
```



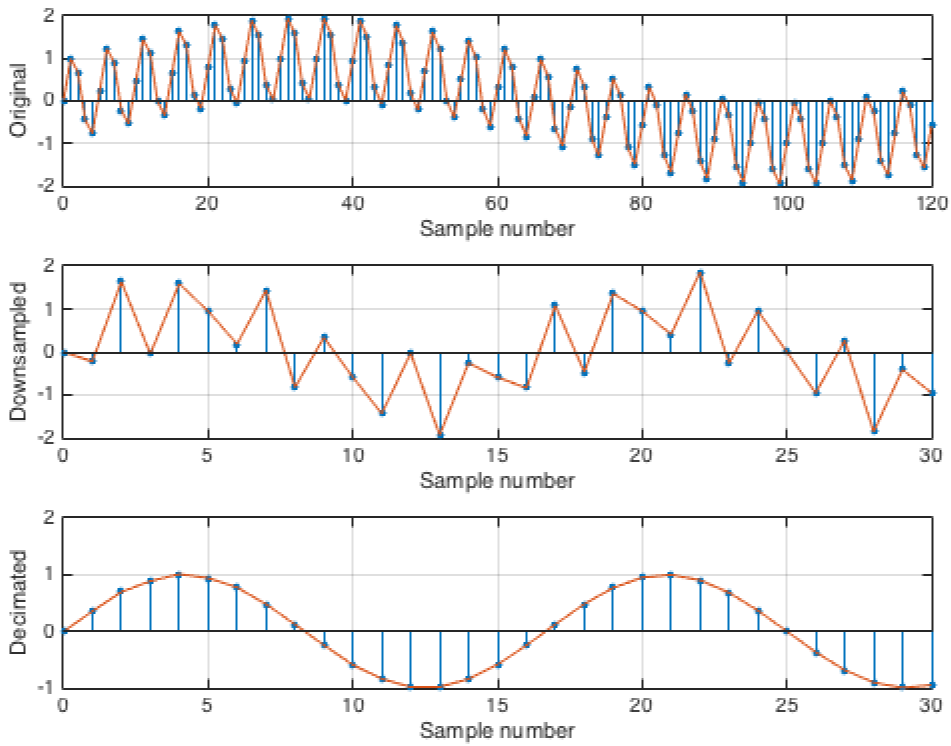
downsample.m: Downsample by an integer factor

```
y = downsample(x,8);
subplot(3,1,2)
stem(0:30,y(1:31),'filled','markersize',3)
hold on
plot(0:30,y(1:31))
grid on
xlabel('Sample number')
ylabel('Downsampled')
```



decimate.m: Downsample with an anti-alias filter

```
y = decimate(x,8);
subplot(3,1,3)
stem(0:30,y(1:31),'filled','markersize',3)
hold on
plot(0:30,y(1:31))
grid on
xlabel('Sample number')
ylabel('Decimated')
```



Difference is decimate applies a low-pass filter BEFORE downsampling!

Upsample

```
n = 1:4
y = upsample(n,3)


n =

     1     2     3     4


y =

     1     0     0     2     0     0     3     0     0     4     0
     0


y = upsample(n,3,2)

y =

     0     0     1     0     0     2     0     0     3     0     0
     4
```

really cannot think when I'd use this

Interpolatation

```
help interp
```

```
 INTERP Resample data at a higher rate using lowpass interpolation.
    Y = INTERP(X,R) resamples the sequence in vector X at R times
    the original sample rate.  The resulting resampled vector Y is
    R times longer, LENGTH(Y) = R*LENGTH(X).

    A symmetric filter, B, allows the original data to pass through
    unchanged and interpolates between so that the mean square error
    between them and their ideal values is minimized.
    Y = INTERP(X,R,L,CUTOFF) allows specification of arguments
    L and CUTOFF which otherwise default to 4 and .5 respectively.
    2*L is the number of original sample values used to perform the
    interpolation.  Ideally L should be less than or equal to 10.
    The length of B is 2*L*R+1. The signal is assumed to be band
    limited with cutoff frequency 0 < CUTOFF <= 1.0.
    [Y,B] = INTERP(X,R,L,CUTOFF) returns the coefficients of the
    interpolation filter B.

    % Example:
    %   Interpolate a signal by a factor of four.

    t = 0:0.001:.029;                          % Time vector
    x = sin(2*pi*30*t) + sin(2*pi*60*t);    % Original Signal
    y = interp(x,4);                           % Interpolated Signal
    subplot(211);
    stem(x);
    title('Original Signal');
    subplot(212);
    stem(y);
    title('Interpolated Signal');

    See also DECIMATE, RESAMPLE, UPFIRDN.

    Other functions named interp:
       DynamicSystem/interp

    Reference page in Help browser
       doc signal/interp
```

```
help interp1
```

```
 INTERP1 1-D interpolation (table lookup)

    Some features of INTERP1 will be removed in a future release.
    See the R2012a release notes for details.

    Vq = INTERP1(X,V,Xq) interpolates to find Vq, the values of the
    underlying function V=F(X) at the query points Xq.
```

```
   X must be a vector. The length of X is equal to N.
   If V is a vector, V must have length N, and Vq is the same size as
Xq.
   If V is an array of size [N,D1,D2,...,Dk], then the interpolation
is
   performed for each D1-by-D2-by-...-Dk value in V(i,:,:,...,:). If
Xq
   is a vector of length M, then Vq has size [M,D1,D2,...,Dk]. If Xq
is
   an array of size [M1,M2,...,Mj], then Vq is of size
   [M1,M2,...,Mj,D1,D2,...,Dk].

   Vq = INTERP1(V,Xq) assumes X = 1:N, where N is LENGTH(V)
   for vector V or SIZE(V,1) for array V.

   Interpolation is the same operation as "table lookup".  Described
in
   "table lookup" terms, the "table" is [X,V] and INTERP1 "looks-up"
   the elements of Xq in X, and, based upon their location, returns
   values Vq interpolated within the elements of V.

   Vq = INTERP1(X,V,Xq,METHOD) specifies alternate methods.
   The default is linear interpolation. Use an empty matrix [] to
specify
   the default. Available methods are:

     'nearest'  - nearest neighbor interpolation
     'next'     - next neighbor interpolation
     'previous' - previous neighbor interpolation
     'linear'   - linear interpolation
     'spline'   - piecewise cubic spline interpolation (SPLINE)
     'pchip'    - shape-preserving piecewise cubic interpolation
     'cubic'    - same as 'pchip'
     'v5cubic'  - the cubic interpolation from MATLAB 5, which does
not
                  extrapolate and uses 'spline' if X is not equally
                  spaced.

   Vq = INTERP1(X,V,Xq,METHOD,'extrap') uses the interpolation
algorithm
   specified by METHOD to perform extrapolation for elements of Xq
outside
   the interval spanned by X.

   Vq = INTERP1(X,V,Xq,METHOD,EXTRAPVAL) replaces the values outside
of the
   interval spanned by X with EXTRAPVAL.  NaN and 0 are often used
for
   EXTRAPVAL.  The default extrapolation behavior with four input
arguments
   is 'extrap' for 'spline' and 'pchip' and EXTRAPVAL = NaN (NaN
+NaNi for
   complex values) for the other methods.
```

```
   PP = INTERP1(X,V,METHOD,'pp') uses the interpolation algorithm
specified
   by METHOD to generate the ppform (piecewise polynomial form) of V.
The
   method may be any of the above METHOD except for 'v5cubic'. PP may
then
   be evaluated via PPVAL. PPVAL(PP,Xq) is the same as
   INTERP1(X,V,Xq,METHOD,'extrap').

   For example, generate a coarse sine curve and interpolate over a
   finer abscissa:
       X = 0:10; V = sin(X); Xq = 0:.25:10;
       Vq = interp1(X,V,Xq); plot(X,V,'o',Xq,Vq,':.')

   For a multi-dimensional example, we construct a table of
functional
   values:
       X = [1:10]'; V = [ X.^2, X.^3, X.^4 ];
       Xq = [ 1.5, 1.75; 7.5, 7.75]; Vq = interp1(X,V,Xq);

   creates 2-by-2 matrices of interpolated function values, one
matrix for
   each of the 3 functions. Vq will be of size 2-by-2-by-3.

   Class support for inputs X, V, Xq, EXTRAPVAL:
      float: double, single

   See also INTERPFT, SPLINE, PCHIP, INTERP2, INTERP3, INTERPN,
PPVAL.

   Other functions named interp1:
      gpuArray/interp1

   Reference page in Help browser
      doc interp1
```

```
help spline
```

```
SPLINE Cubic spline data interpolation.
   PP = SPLINE(X,Y) provides the piecewise polynomial form of the
   cubic spline interpolant to the data values Y at the data sites X,
   for use with the evaluator PPVAL and the spline utility UNMKPP.
   X must be a vector.
   If Y is a vector, then Y(j) is taken as the value to be matched at
X(j),
   hence Y must be of the same length as X  -- see below for an
exception
   to this.
   If Y is a matrix or ND array, then Y(:,...,:,j) is taken as the
value to
   be matched at X(j),  hence the last dimension of Y must equal
length(X) --
```

```
   see below for an exception to this.

   YY = SPLINE(X,Y,XX) is the same as  YY = PPVAL(SPLINE(X,Y),XX),
thus
   providing, in YY, the values of the interpolant at XX.  For
information
   regarding the size of YY see PPVAL.

   Ordinarily, the not-a-knot end conditions are used. However, if Y
contains
   two more values than X has entries, then the first and last value
in Y are
   used as the endslopes for the cubic spline.  If Y is a vector,
this
   means:
       f(X) = Y(2:end-1),  Df(min(X))=Y(1),    Df(max(X))=Y(end).
   If Y is a matrix or N-D array with SIZE(Y,N) equal to LENGTH(X)+2,
then
   f(X(j)) matches the value Y(:,...,:,j+1) for j=1:LENGTH(X), then
   Df(min(X)) matches Y(:,:,...:,1) and Df(max(X)) matches
Y(:,:,...:,end).

   Example:
   This generates a sine-like spline curve and samples it over a
finer mesh:
       x = 0:10;  y = sin(x);
       xx = 0:.25:10;
       yy = spline(x,y,xx);
       plot(x,y,'o',xx,yy)

   Example:
   This illustrates the use of clamped or complete spline
interpolation where
   end slopes are prescribed. In this example, zero slopes at the
ends of an
   interpolant to the values of a certain distribution are enforced:
      x = -4:4; y = [0 .15 1.12 2.36 2.36 1.46 .49 .06 0];
      cs = spline(x,[0 y 0]);
      xx = linspace(-4,4,101);
      plot(x,y,'o',xx,ppval(cs,xx),'-');

   Class support for inputs x, y, xx:
      float: double, single

   See also INTERP1, PCHIP, PPVAL, MKPP, UNMKPP.

   Reference page in Help browser
      doc spline
```

```
help pchip
```

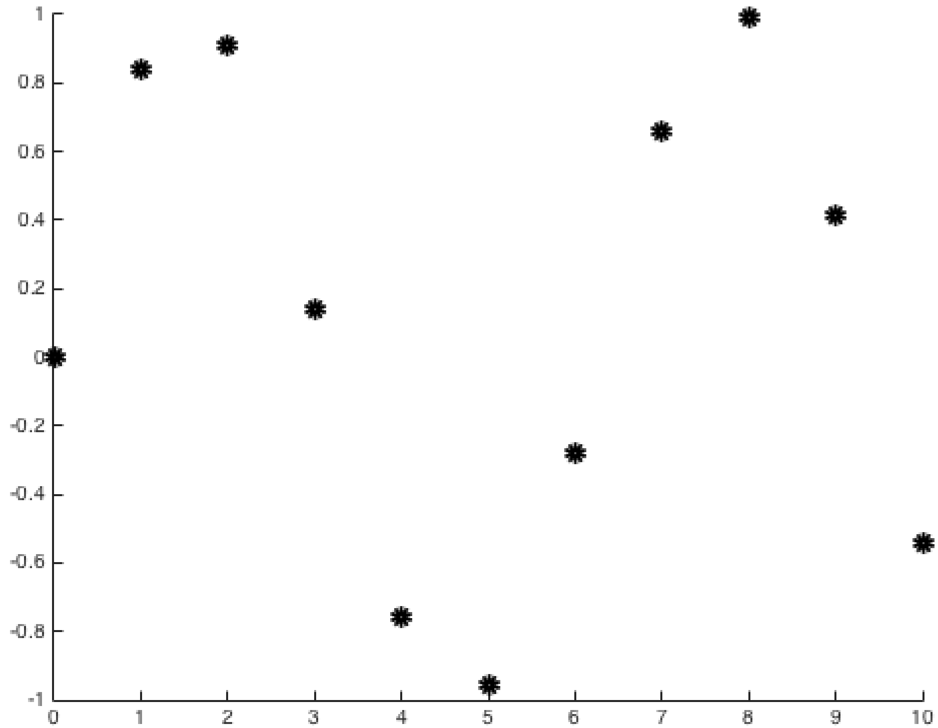
```
 PCHIP  Piecewise Cubic Hermite Interpolating Polynomial.
```

```
   PP = PCHIP(X,Y) provides the piecewise polynomial form of a
certain
   shape-preserving piecewise cubic Hermite interpolant, to the
values
   Y at the sites X, for later use with PPVAL and the spline utility
UNMKPP.
   X must be a vector.
   If Y is a vector, then Y(j) is taken as the value to be matched at
X(j),
   hence Y must be of the same length as X.
   If Y is a matrix or ND array, then Y(:,...,:,j) is taken as the
value to
   be matched at X(j),  hence the last dimension of Y must equal
length(X).

   YY = PCHIP(X,Y,XX) is the same as YY = PPVAL(PCHIP(X,Y),XX), thus
   providing, in YY, the values of the interpolant at XX.

   The PCHIP interpolating function, p(x), satisfies:
   On each subinterval,  X(k) <= x <= X(k+1),  p(x) is the cubic
Hermite
      interpolant to the given values and certain slopes at the two
endpoints.
   Therefore, p(x) interpolates Y, i.e., p(X(j)) = Y(:,j), and
       the first derivative, Dp(x), is continuous, but
       D^2p(x) is probably not continuous; there may be jumps at the
X(j).
   The slopes at the X(j) are chosen in such a way that
      p(x) is "shape preserving" and "respects monotonicity". This
means that,
   on intervals where the data is monotonic, so is p(x);
   at points where the data have a local extremum, so does p(x).

 Comparing PCHIP with SPLINE:
   The function s(x) supplied by SPLINE is constructed in exactly the
same way,
   except that the slopes at the X(j) are chosen differently, namely
to make
   even D^2s(x) continuous. This has the following effects.
   SPLINE is smoother, i.e., D^2s(x) is continuous.
   SPLINE is more accurate if the data are values of a smooth
function.
   PCHIP has no overshoots and less oscillation if the data are not
smooth.
   PCHIP is less expensive to set up.
   The two are equally expensive to evaluate.

   Example:

     x = -3:3;
     y = [-1 -1 -1 0 1 1 1];
     t = -3:.01:3;
     plot(x,y,'o',t,[pchip(x,y,t); spline(x,y,t)])
     legend('data','pchip','spline',4)
```

```
Class support for inputs x, y, xx:
   float: double, single

See also INTERP1, SPLINE, PPVAL, UNMKPP.

Reference page in Help browser
   doc pchip
```

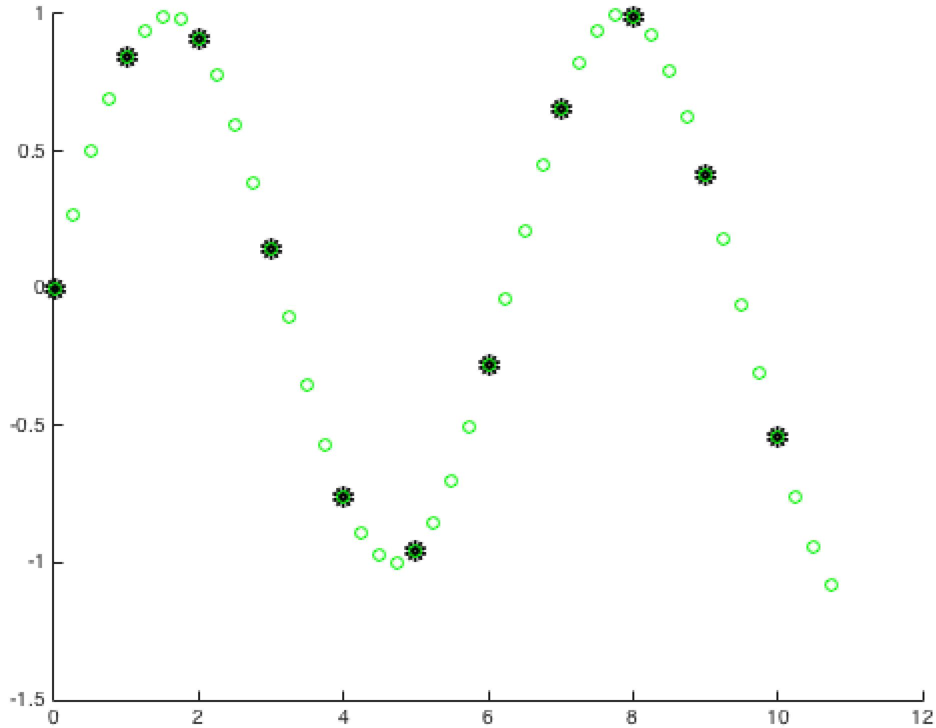
Create original time series

```
t = 0:10;
x = sin(t);
figure
scatter(t,x,'k','LineWidth',5)
hold on
```



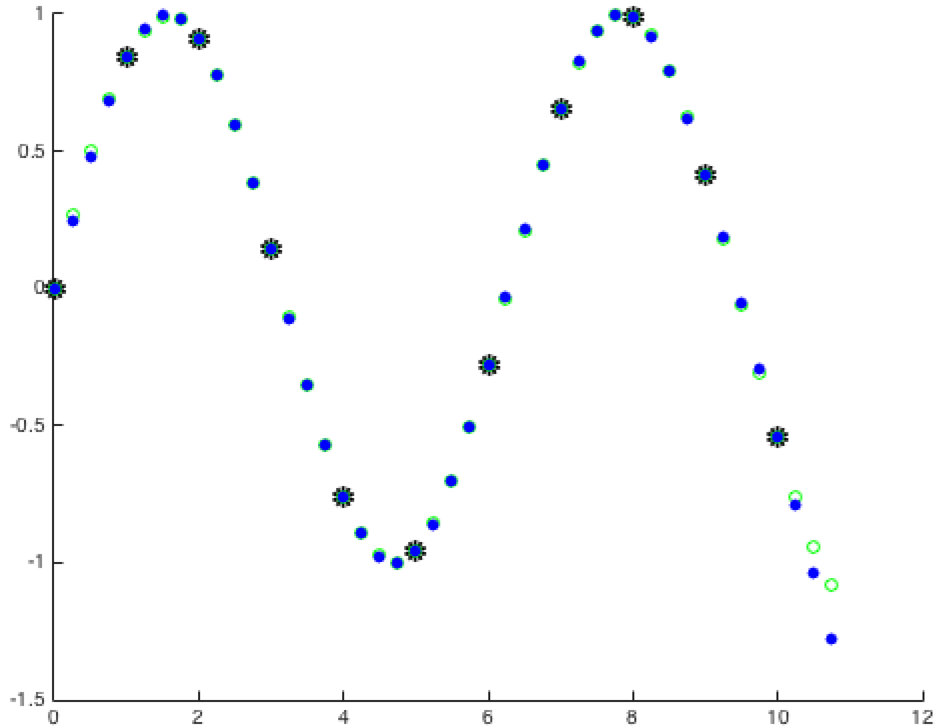
Resample at 4 times higher rate with splines

```
tt=interp(t,4);
xx = spline(t,x,tt);
scatter(tt,xx,'g','o')
```



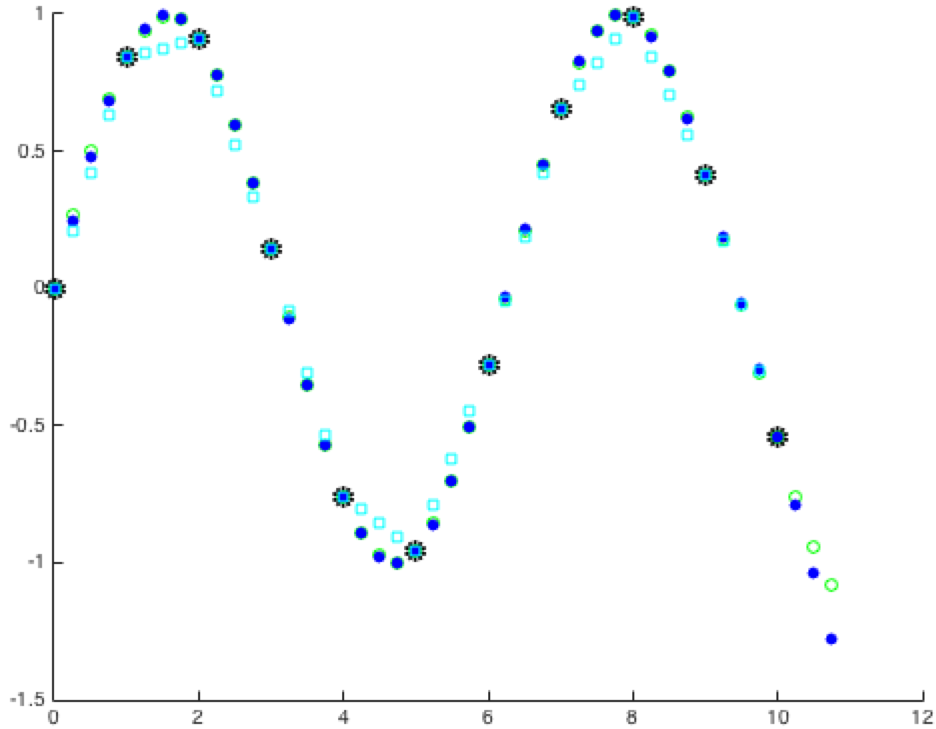
Default interpolation with interp

```
xx=interp(x,4);
scatter(tt,xx,'b','o','filled')
```



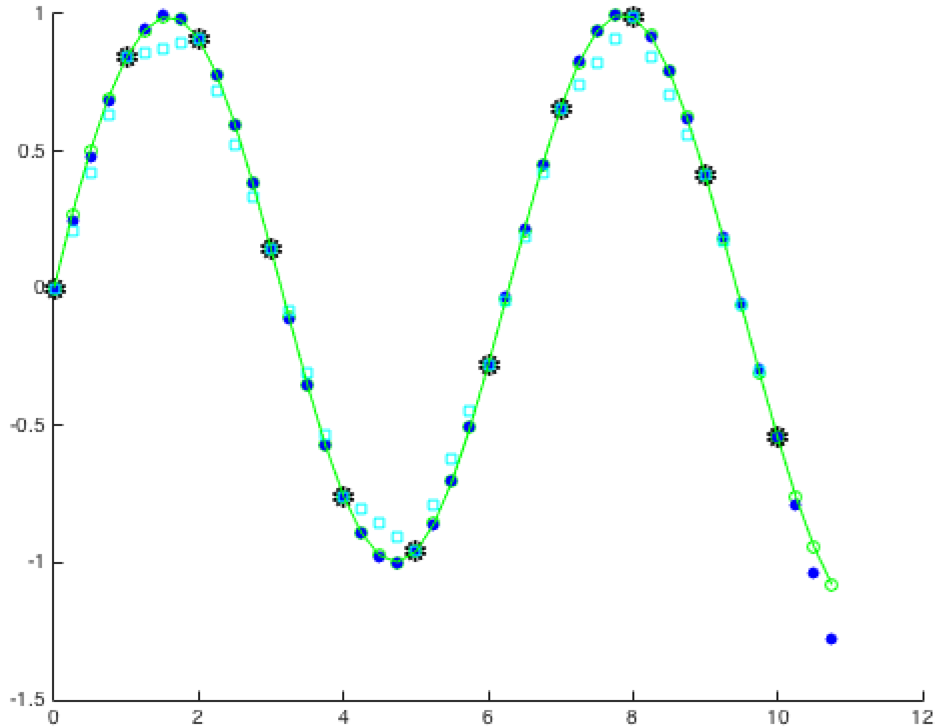
Linear interpolation with interp1

```
xx=interp1(t,x,tt,'linear');
scatter(tt,xx,'c','s')
```



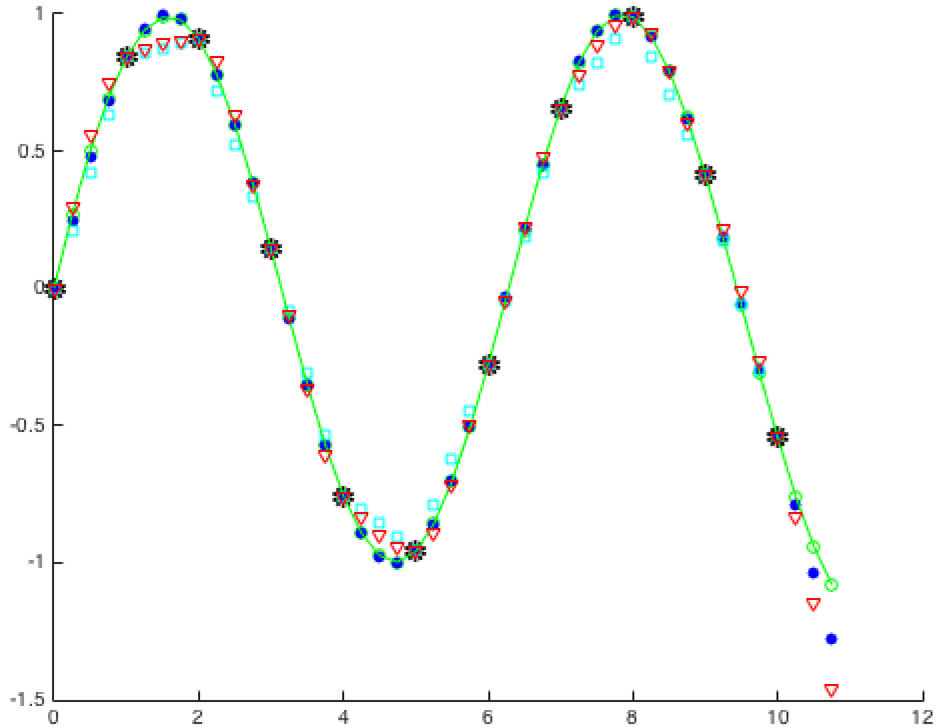
Spline interpolation with interp1

```
xx=interp1(t,x,tt,'spline');
plot(tt,xx,'g')
```



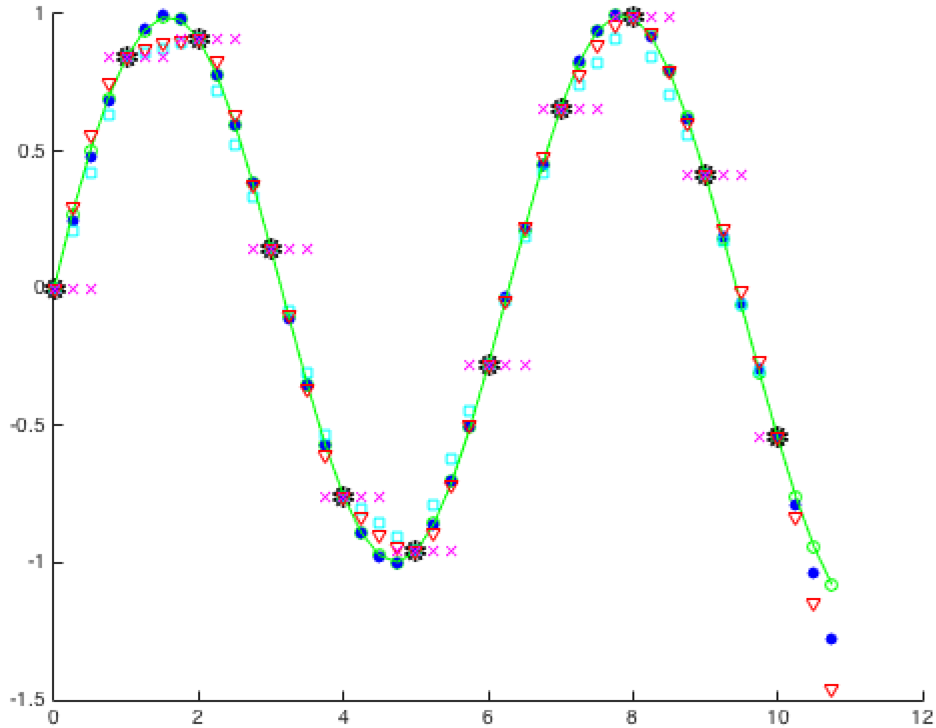
Piecewise Cubic Hermite interpolation with interp1

```
xx=interp1(t,x,tt,'pchip');
scatter(tt,xx,'r','v')
```

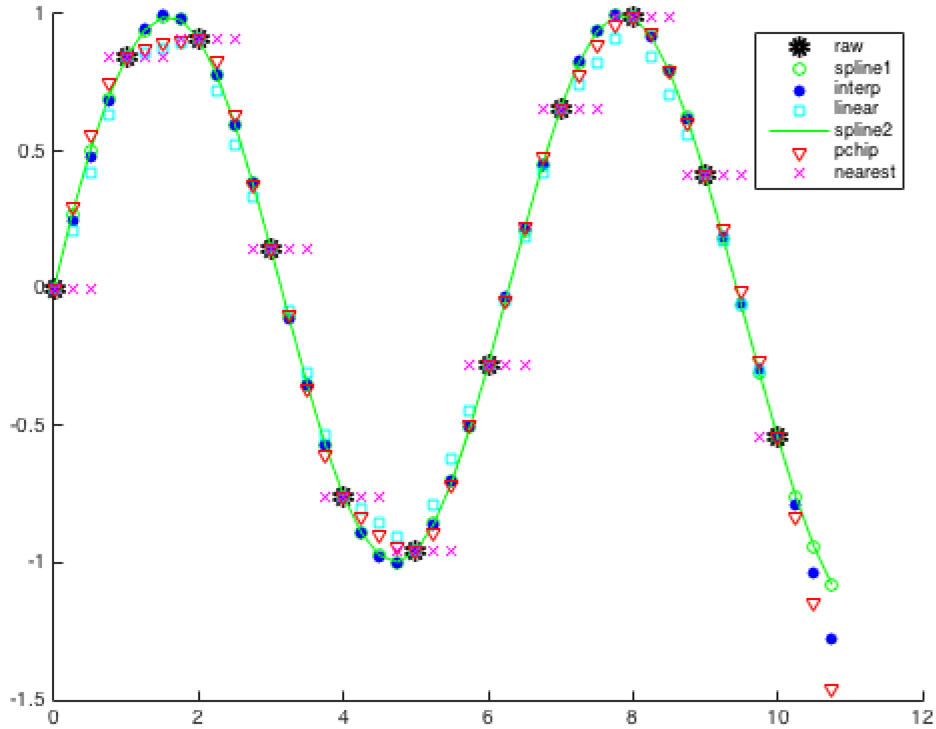


Nearest neighbour interpolation with interp1

```
xx=interp1(t,x,tt,'nearest');
scatter(tt,xx,'m','x')
```




```
hold off
legend('raw','spline1','interp','linear','spline2','pchip','nearest')
```



Looks like best match to sine wave is with spline or interp

Resample - change sampling frequency by rational number - or provide sample positions

Changing by a rational number P/Q is similar to interpolation except it has to interp(t,x,P) and then decimate(x,Q) - using an anti-alias filter

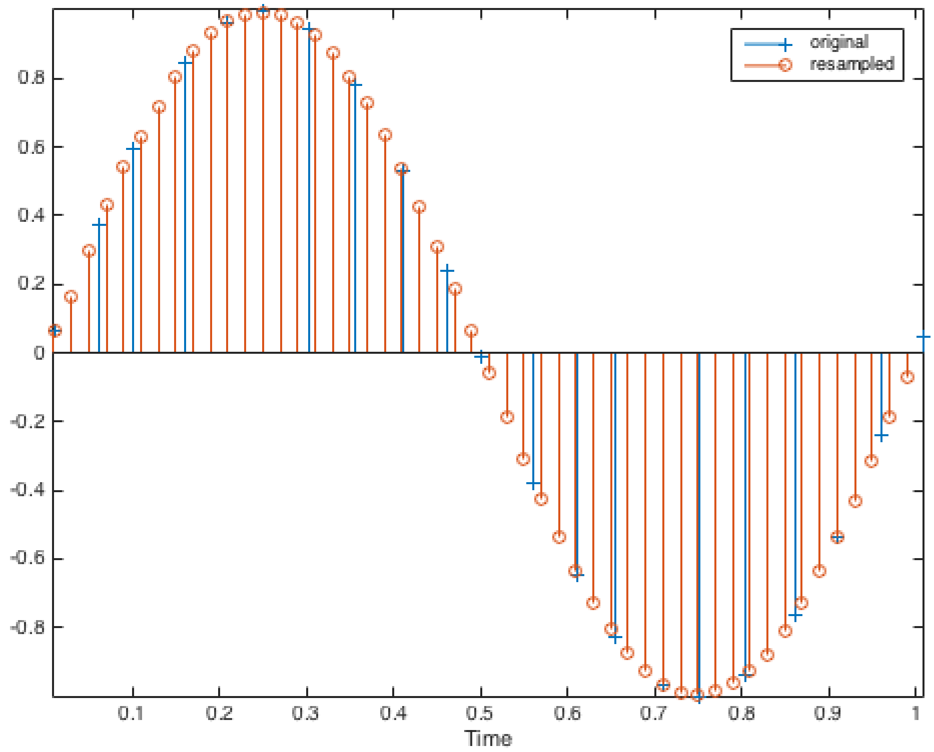
Resample can go to lower and higher sample rates

Resample can do 'linear', 'spline' or 'pchip' interpolation

Even better, resample can turn irregularly spaced data into regular spaced data - just sample the sample times

```
Fs = 50;
tx = linspace(0,1,21) + .012*rand(1,21);
x = sin(2*pi*tx);
[y, ty] = resample(x, tx, Fs);

figure
stem(tx,x,'+')
hold on
stem(ty,y,'o')
legend('original','resampled');
xlabel('Time')
```

```
axis tight
```



TREND ANALYSIS

Summary statistics (e.g. mean, median, mode) are not useful for time series that has a trend. First have to make the time series "stationary" e.g. for Y[t] = S[t] + mt + C, can either: (i) detrend - fit a regression line and then subtract from time series --> Z[t] = S[t] (ii) 1st difference Y'[t] = Y[t]-Y[t-1] = S[t]-S[t-1] + m

Time series can be decomposed into:

- Trend
- Signal
- Noise

Trend may be linear, a higher degree polynomial, sinusoidal, seasonal (annual), diurnal etc.

Time series might look like it has a linear trend, e.g. temperature over an hour. But on longer time scales it has more complex trends - daily, annual.

Sometimes interested in the trend (e.g. global warming). Sometimes interested in short-term variations (signal + noise)

Step 1 is always to do a time plot

Simple linear trend x[t] = a + bt + e[t], where e[t] = random error with mean=0

Trend term is m[t] = (a + bt), where slope is b

Trends are difficult to fit. They may be:

- Piecewise linear
- non-linear
- a & b may be time dependent (vary randomly)

Transformations

Various transformations can be done to stabilize the variance and make the time series stationary e.g. logarithmic, power, Box-Cox

Curve fitting - later

Linear filter

MA = moving average - we know this is symmetric, zero-phase, acausal, have end-effects

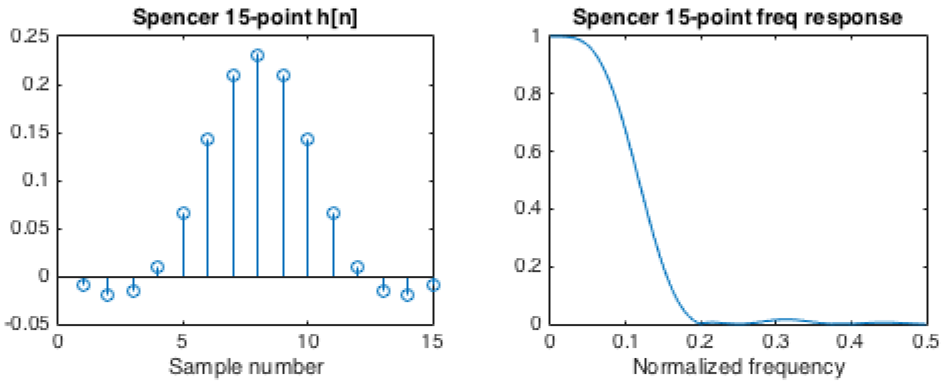
can be implemented with filter command, e.g. y=filter(0.25 * [1 1 1 1],1,x) or with smooth command y=smooth(x,4)

Weighted MA - better for trend removal e.g. 15 point Spencer MA

```
h = conv( 0.25*ones(4,1), 0.25*ones(4,1) );
h2 = conv(h, 0.2*ones(5,1));
h3 = conv(h2, [-3/4 3/4 1 3/4 -3/4]);
320*h3
sum(h3)
figure
subplot(2,2,1), stem(h3), title('Spencer 15-point h[n]'),
 xlabel('Sample number')
subplot(2,2,2)
[H,W]=freqz(h3);
plot(W/(2*pi),abs(H)), title('Spencer 15-point freq response'),
 xlabel('Normalized frequency')
```

```
ans =

   -3.0000
   -6.0000
   -5.0000
    3.0000
   21.0000
   46.0000
   67.0000
   74.0000
   67.0000
   46.0000
   21.0000
    3.0000
```

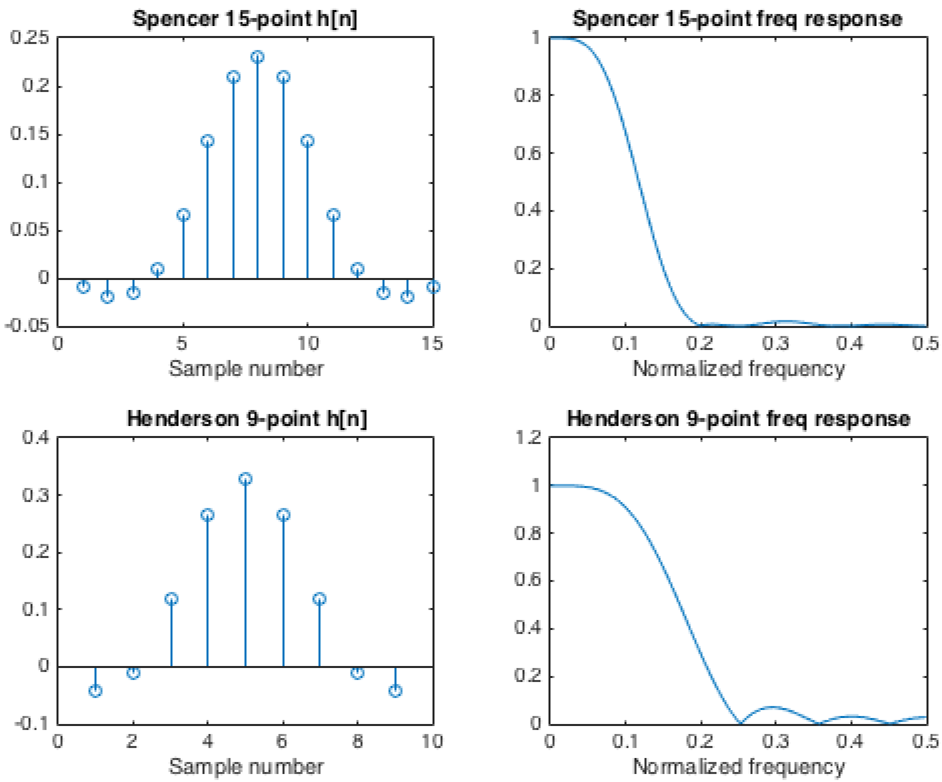
```
    -5.0000
    -6.0000
    -3.0000


ans =

     1.0000
```



e.g. Henderson MA 9-point

```
h = [-0.041 -0.010 0.119 0.267 0.330 0.267 0.119 -0.010 -0.041];
sum(h)
subplot(2,2,3),stem(h), title('Henderson 9-point h[n]'),
 xlabel('Sample number')
subplot(2,2,4)
[H,W]=freqz(h);
plot(W/(2*pi),abs(H)),title('Henderson 9-point freq response'),
 xlabel('Normalized frequency')
```

```
ans =

     1.0000
```

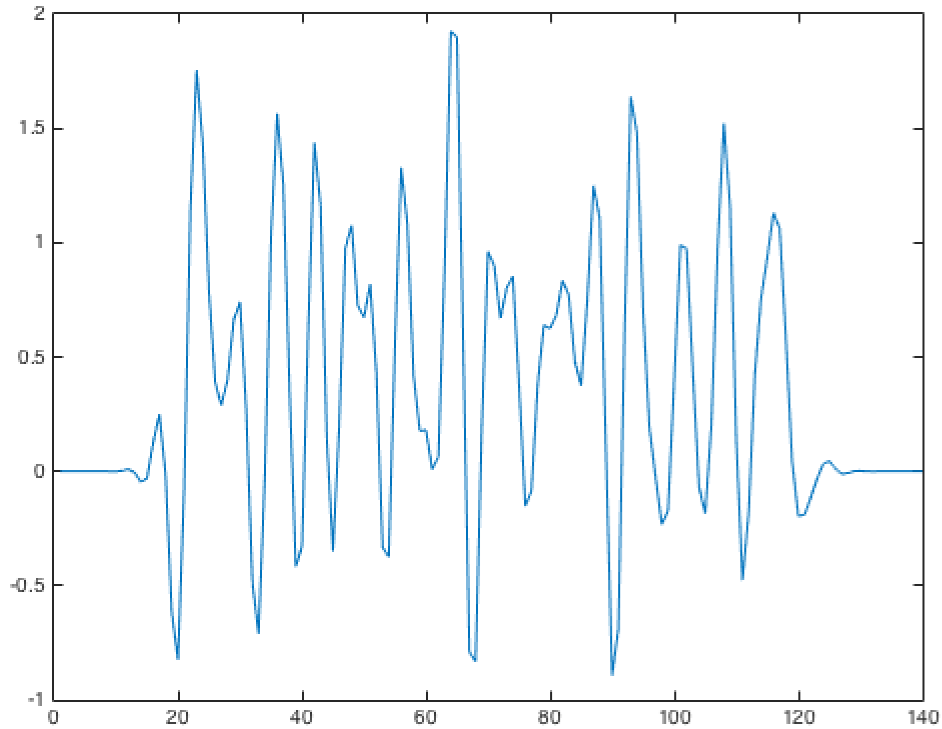


asymmetric filters - causal - current input & past inputs/outputs only

Slutsky-Yule effect

Applying a moving average filter to random data multiple times causes spurious sines

```
figure
maf = 1/6 * [-1 2 4 2 -1];
x = rand(100,1);
for c=1:10
    x = conv(x,maf);
end
plot(x)
```



2.6 Analysis of seasonal data

x[t] = m[t] + s[t] + e[t] % additive

x[t] = m[t]s[t] + e[t]

x[t] = m[t]s[t]e[t] % multiplicative

log[x[t]] = log[m[t]] + log[s[t]] + log[e[t]]

monthly data - use 13-point moving average,but ends are weighted 1/2. keeps centered on sample time.

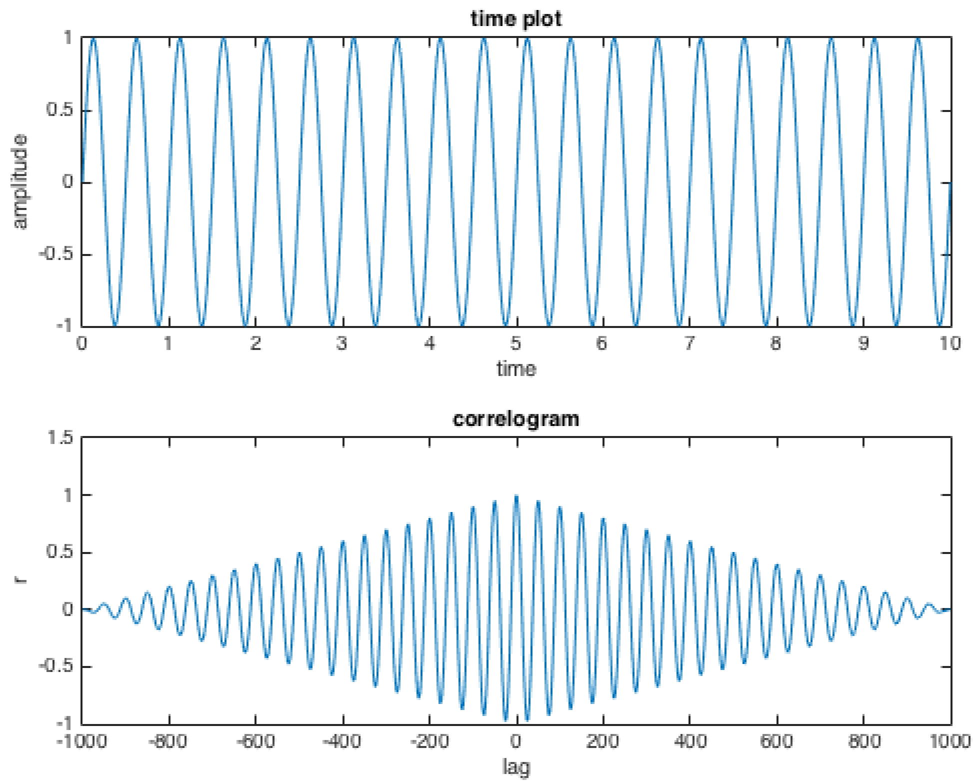
Autocorrelation

correlation between a signal and a copy of itself delayed by k samples get a value for each value of k

sine wave

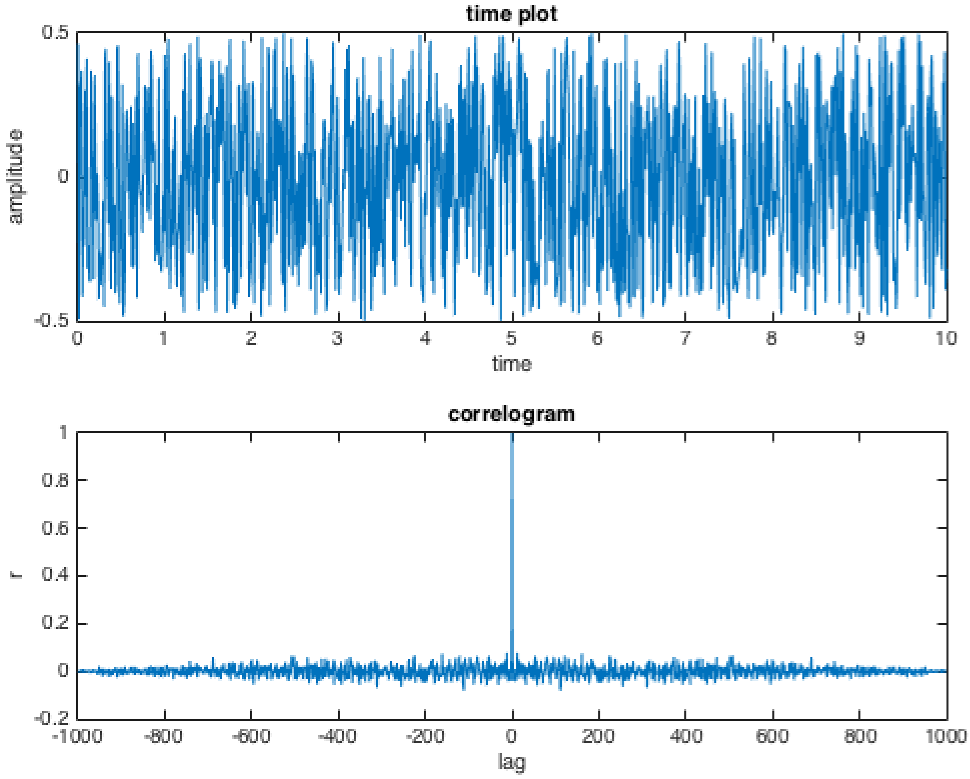
```
t=0:1/100:10;
x=sin(2*pi*2*t);
figure
subplot(2,1,1), plot(t,x), title('time plot'), xlabel('time'),
 ylabel('amplitude')
[r,lag]=xcorr(x,x,'coeff');
```

```
subplot(2,1,2), plot(lag,r), title('correlogram'), xlabel('lag'),
 ylabel('r')
```



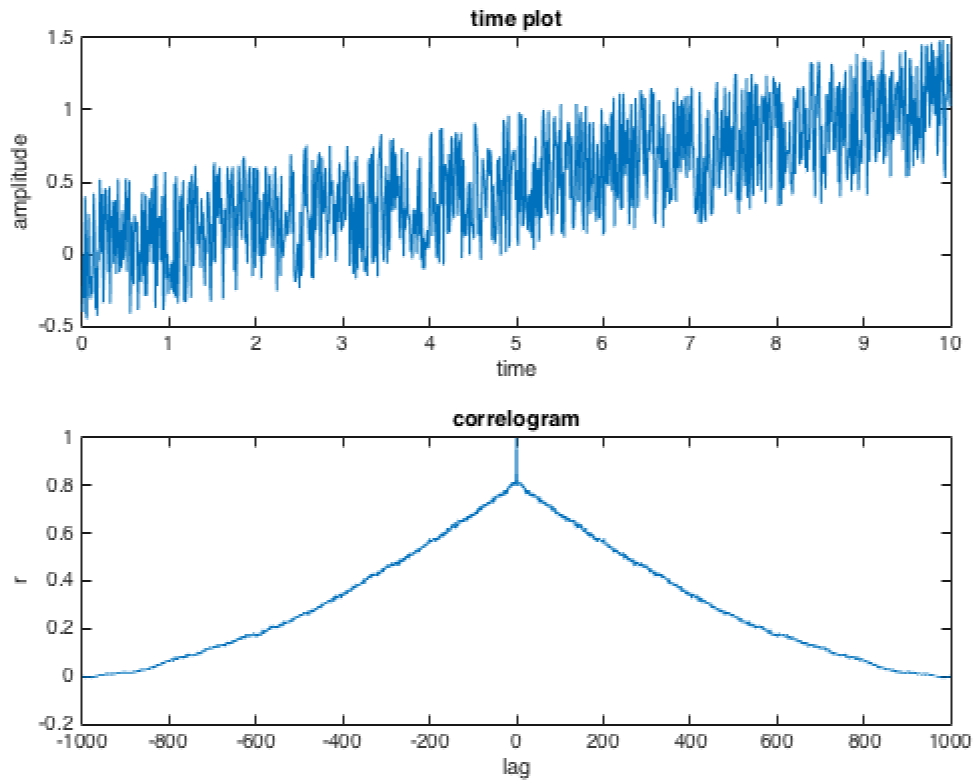
white noise

```
x=rand(size(t))-0.5;
figure
subplot(2,1,1), plot(t,x), title('time plot'), xlabel('time'),
 ylabel('amplitude')
[r,lag]=xcorr(x,x,'coeff');
subplot(2,1,2), plot(lag,r), title('correlogram'), xlabel('lag'),
 ylabel('r')
```



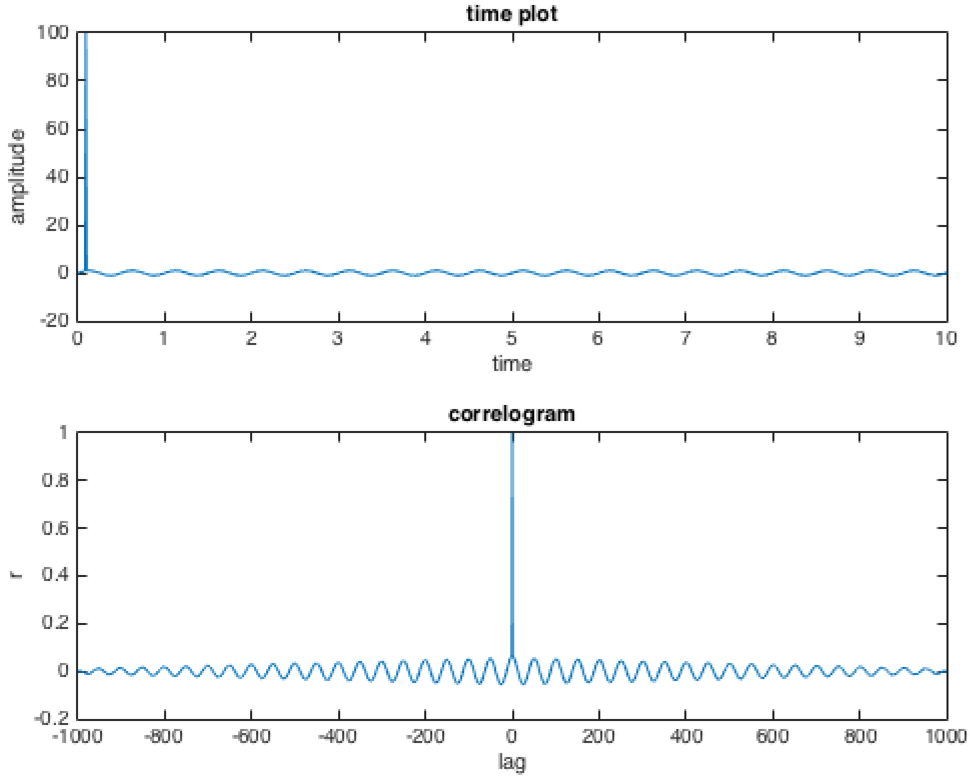
trend

```
x=rand(size(t))-0.5+t/10;
figure
subplot(2,1,1), plot(t,x), title('time plot'), xlabel('time'),
 ylabel('amplitude')
[r,lag]=xcorr(x,x,'coeff');
subplot(2,1,2), plot(lag,r), title('correlogram'), xlabel('lag'),
 ylabel('r')
```



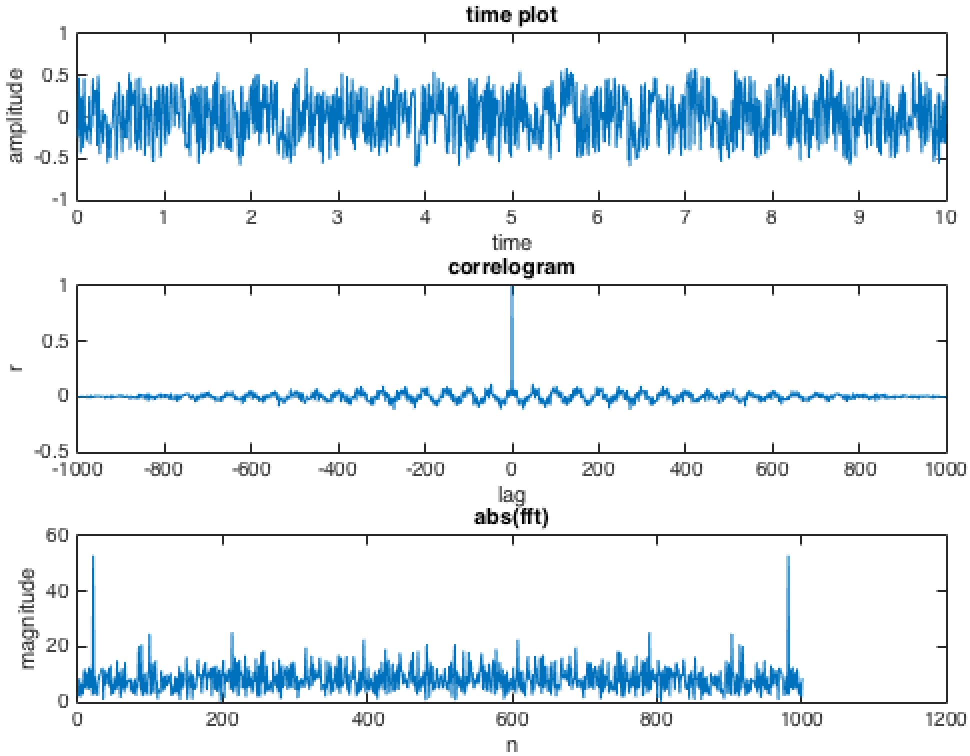
outlier

```
x=sin(2*pi*2*t);
x(10)=100;
figure
subplot(2,1,1), plot(t,x), title('time plot'), xlabel('time'),
 ylabel('amplitude')
[r,lag]=xcorr(x,x,'coeff');
subplot(2,1,2), plot(lag,r), title('correlogram'), xlabel('lag'),
 ylabel('r')
```



What about sine with noise?

```
x=0.1*sin(2*pi*2*t) + rand(size(t))-0.5;
figure
subplot(3,1,1), plot(t,x), title('time plot'), xlabel('time'),
 ylabel('amplitude')
[r,lag]=xcorr(x,x,'coeff');
subplot(3,1,2), plot(lag,r), title('correlogram'), xlabel('lag'),
 ylabel('r')
subplot(3,1,3), plot(abs(fft(x))), title('abs(fft)'), xlabel('n'),
 ylabel('magnitude');
```

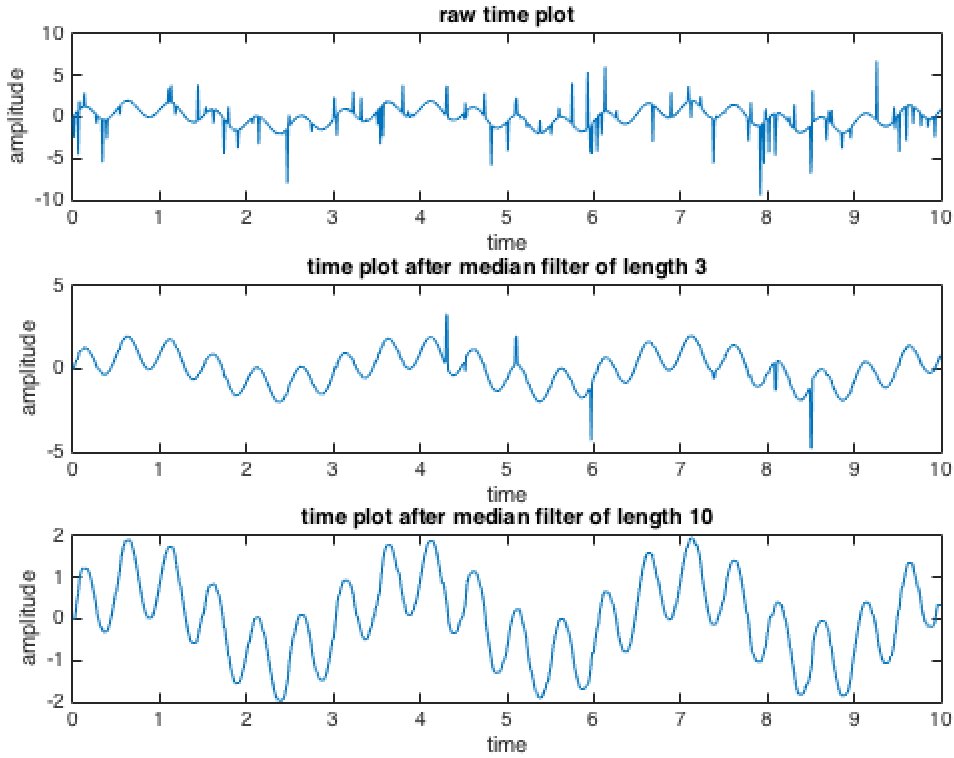


Outlier removal

```
x=sin(2*pi*2*t) + sin(2*t);
```

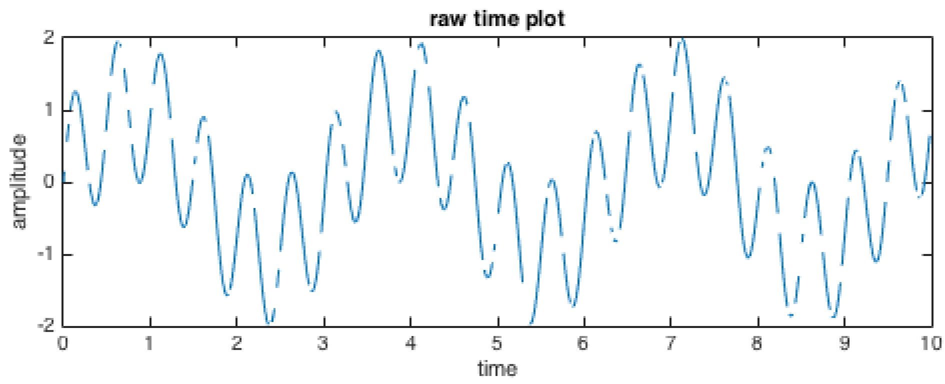
add spikes at random positions of random size

```
spikepos = randi([1 length(x)], 1, 100);
xspikes = zeros(size(x));
xspikes(spikepos) = 3;
xspikes = randn(size(x)) .* xspikes;
x=x+xspikes;

figure
subplot(3,1,1), plot(t,x), title('raw time plot'), xlabel('time'),
 ylabel('amplitude')
y = medfilt1(x,3);
subplot(3,1,2), plot(t,y), title('time plot after median filter of
 length 3'), xlabel('time'), ylabel('amplitude')
y = medfilt1(x,10);
subplot(3,1,3), plot(t,y), title('time plot after median filter of
 length 10'), xlabel('time'), ylabel('amplitude')
```



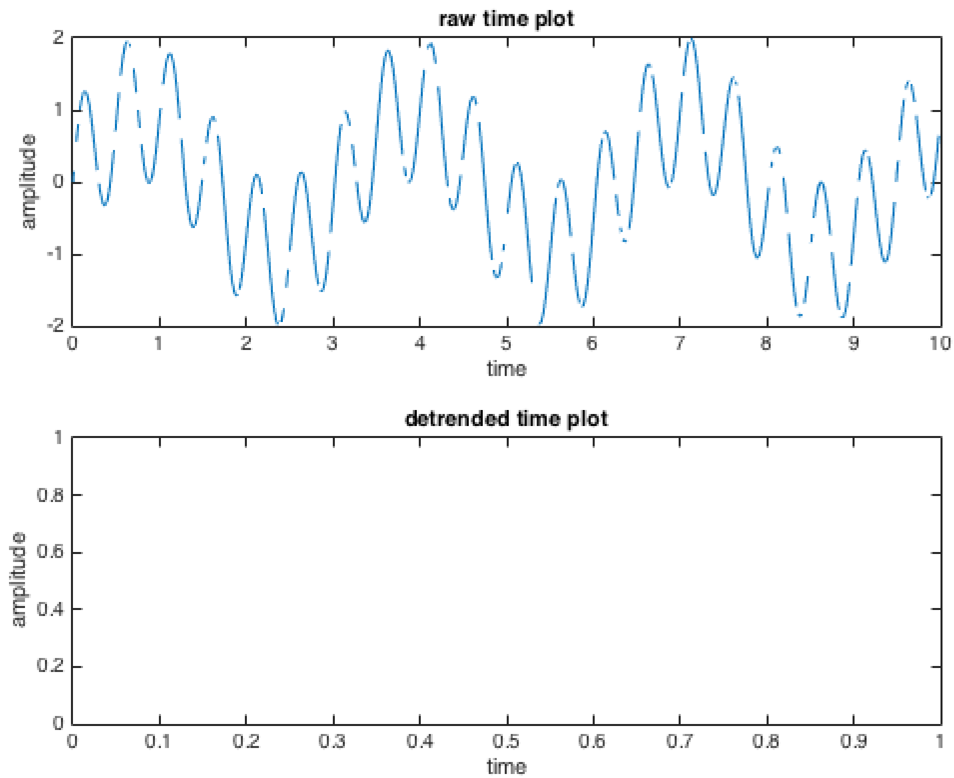
Missing data

```
x=sin(2*pi*2*t) + sin(2*t);
% Drop some samples
gappos = randi([1 length(x)], 1, 100);
xgap = zeros(size(x));
xgap(gappos) = NaN;
x=x+xgap;
figure
subplot(2,1,1), plot(t,x), title('raw time plot'), xlabel('time'),
 ylabel('amplitude')
```



Can we detrend it?

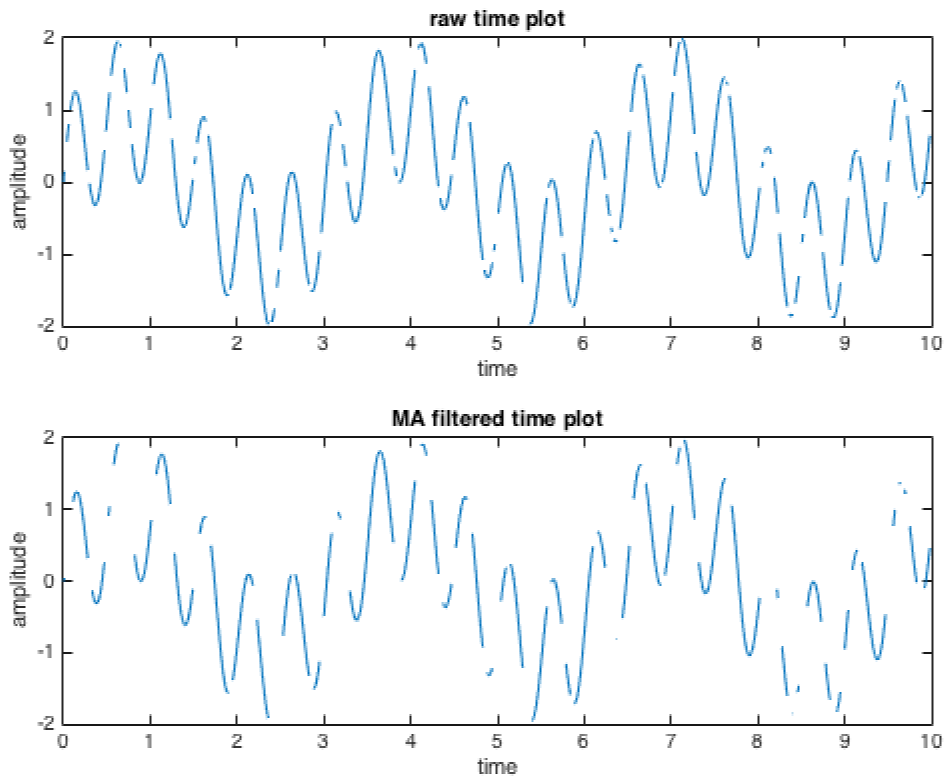
```
subplot(2,1,2), plot(t,detrend(x)), title('detrended time plot'),
 xlabel('time'), ylabel('amplitude')
```



no - result is NaN everywhere

Can we filter it - first a moving-average?

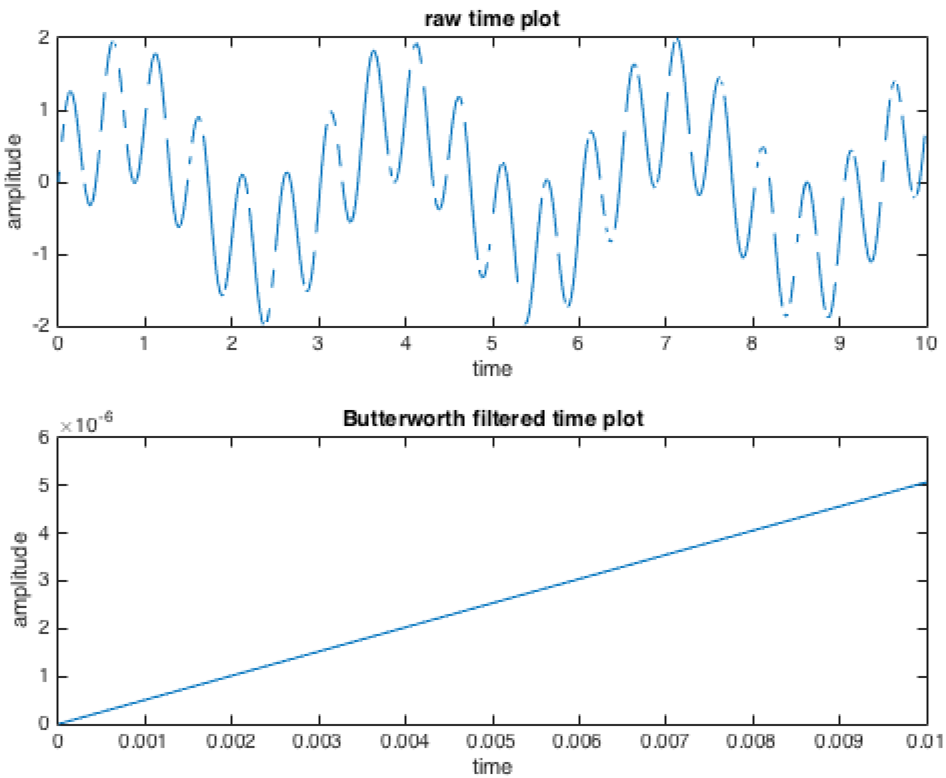
```
y=filter([0.25 0.25 0.25 0.25],1,x);
subplot(2,1,2), plot(t,y), title('MA filtered time plot'),
 xlabel('time'), ylabel('amplitude')
```



yes - but we lose more values - more NaNs

Now try a butterworth filter

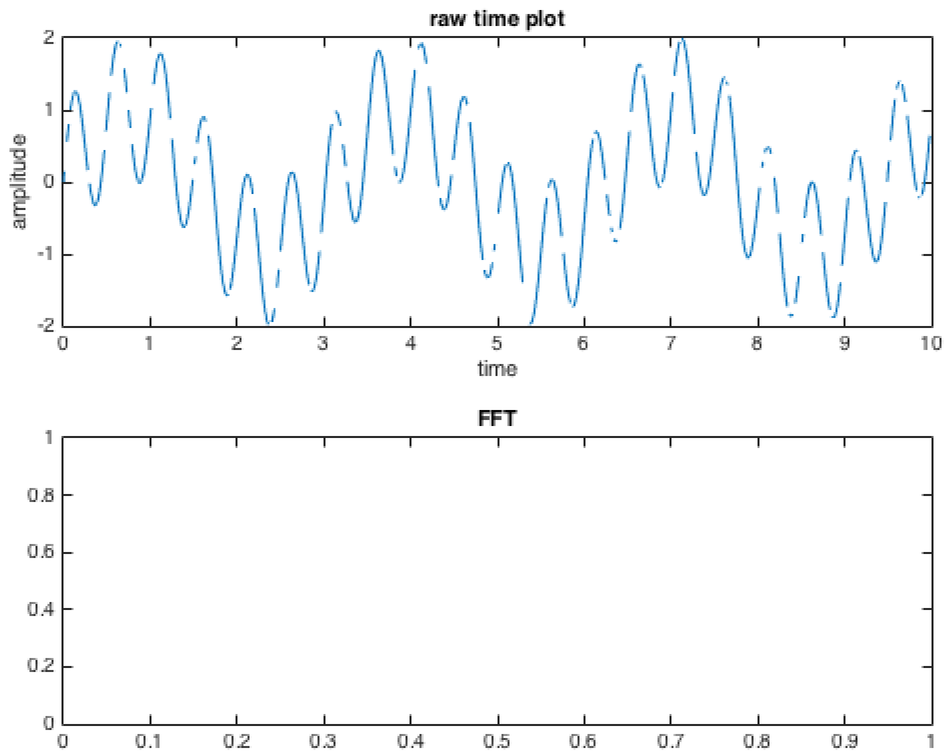
```
[b a]=butter(4, 0.03, 'high');
w=tukeywin(length(x),0.5)';
y=filter(b,a,x.*w);
subplot(2,1,2), plot(t,y), title('Butterworth filtered time plot'),
 xlabel('time'), ylabel('amplitude')
```



just filters up to the first NaN

Can we compute FFT?

```
subplot(2,1,2), plot(abs(fft(x))), title('FFT')
```



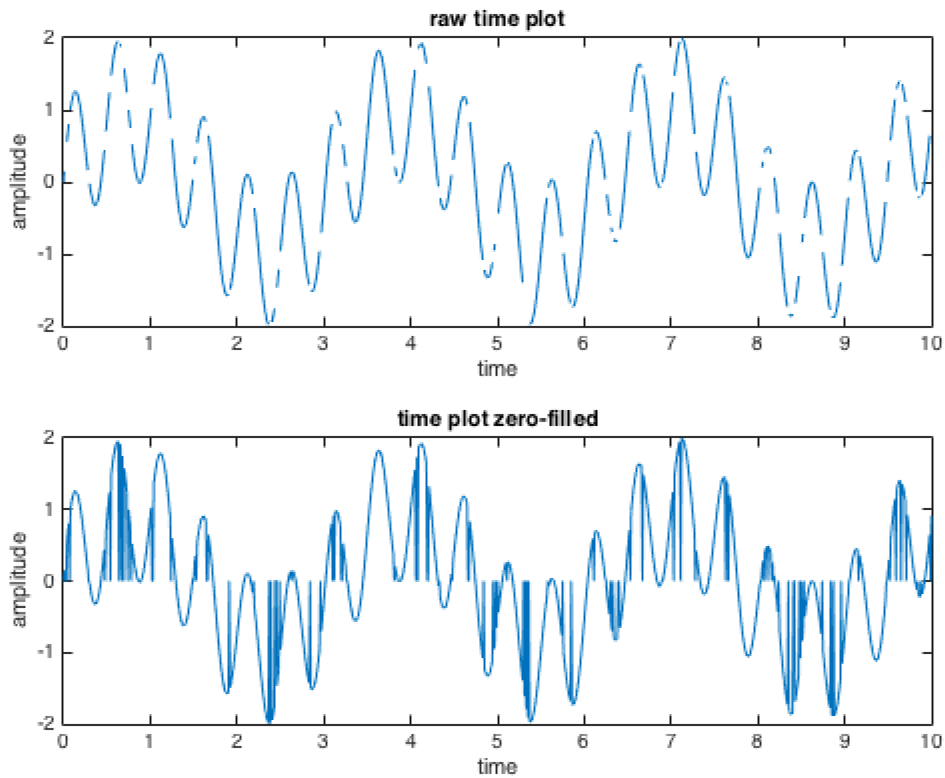
answer is NaN everywhere

Filling gaps

So how do we treat this?

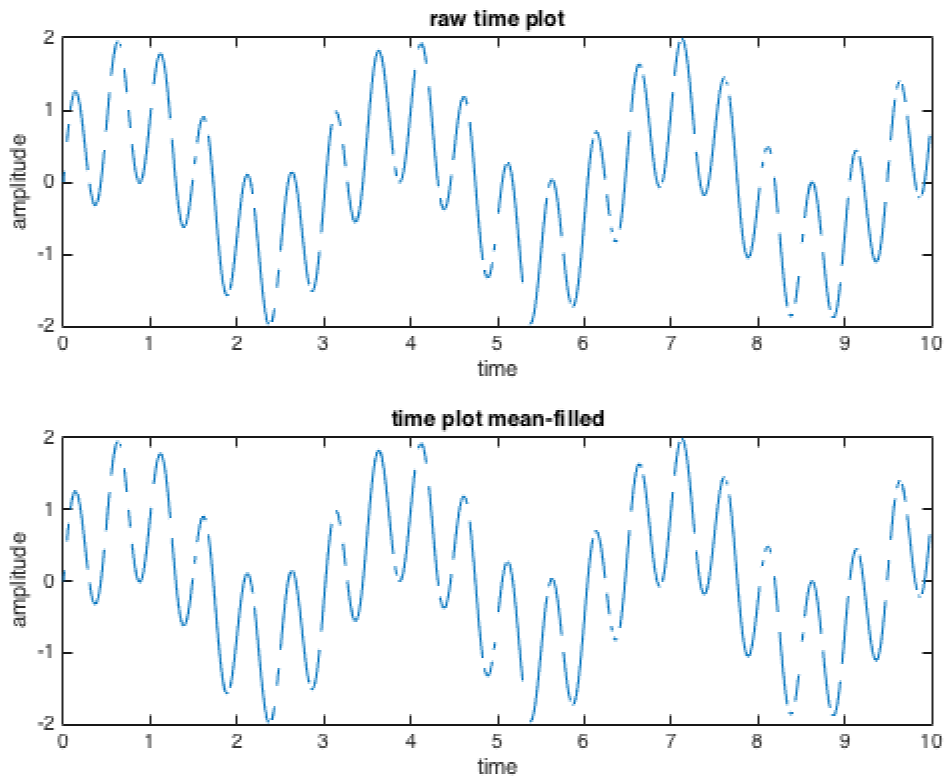
Fill gaps with 0?

```
y=x;
y(isnan(y))=0;
subplot(2,1,2), plot(t,y), title('time plot zero-filled'),
 xlabel('time'), ylabel('amplitude')
```



Fill gaps with mean value?

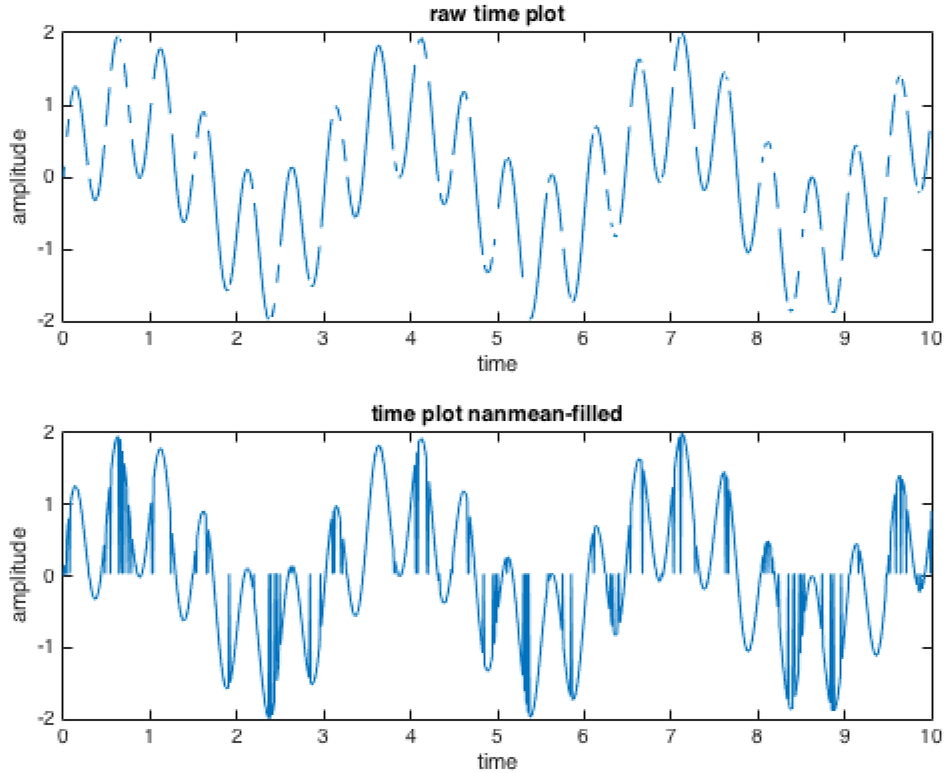
```
y=x;
y(isnan(y))=mean(y);
subplot(2,1,2), plot(t,y), title('time plot mean-filled'),
 xlabel('time'), ylabel('amplitude')
```



nothing changes, mean(x) is NaN!

use nanmean!

```
y=x;
y(isnan(y))=nanmean(y);
subplot(2,1,2), plot(t,y), title('time plot nanmean-filled'),
 xlabel('time'), ylabel('amplitude')
```

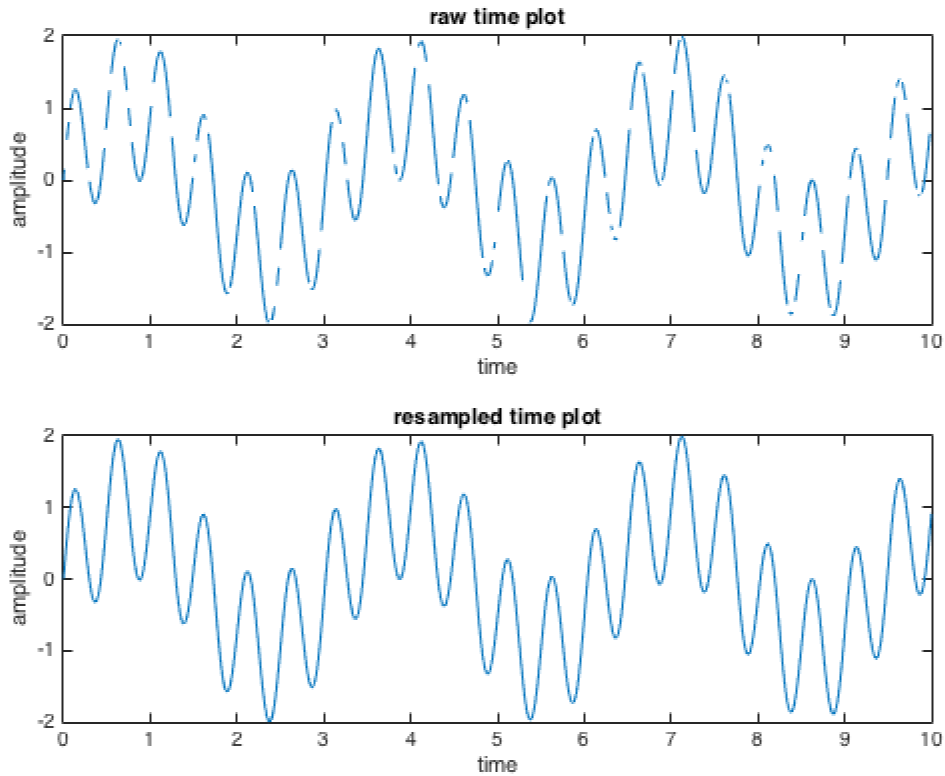


Other ideas???

Could replace the missing samples (NaN's) with interpolated values How do we code that?

Easier - use resample, but at same sample positions

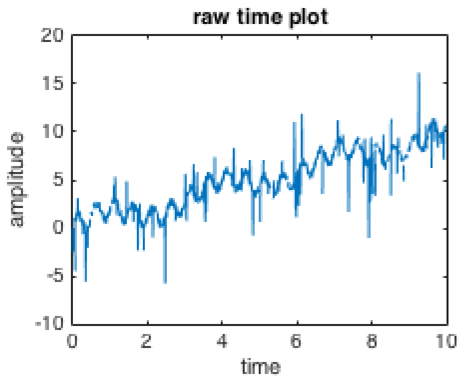
```
[y,ty]=resample(x,t,'spline');
subplot(2,1,2), plot(ty,y), title('resampled time plot'),
 xlabel('time'), ylabel('amplitude')
```



Complicated example

like real data - mix of signal, noise, trend, spikes,

```
x=sin(2*pi*2*t) + sin(2*t) + t + rand(size(x))-0.5;
x=x+xgap+xspikes;
figure
subplot(2,2,1), plot(t,x), title('raw time plot'), xlabel('time'),
 ylabel('amplitude')
```



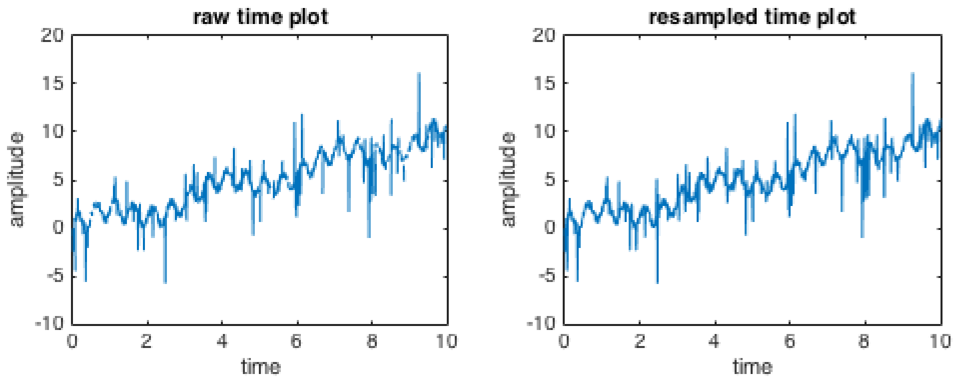
we have used

```
* detrend() to remove a (linear) trend
* medfilt1() to remove spikes/outliers
* resample() to remove gaps
```

So in which order should these be applied?

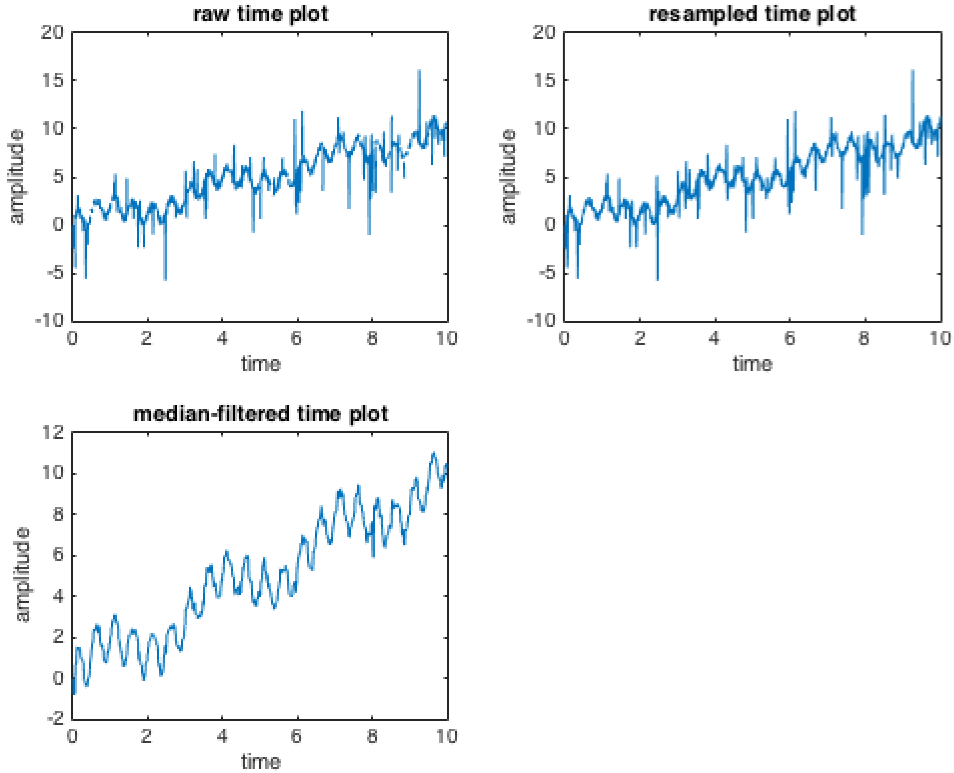
1. resample first to remove gaps

```
[y,ty]=resample(x,t,'spline');
subplot(2,2,2), plot(ty,y), title('resampled time plot'),
 xlabel('time'), ylabel('amplitude')
```



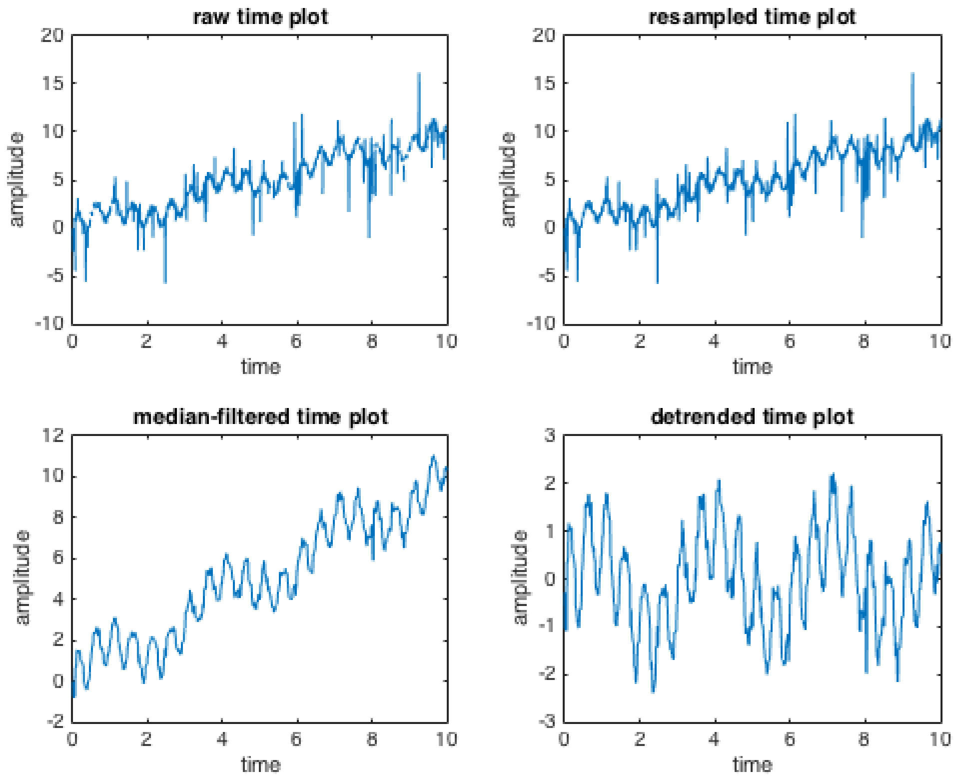
2. remove outliers

```
y=medfilt1(y,5);
subplot(2,2,3), plot(ty,y), title('median-filtered time plot'),
 xlabel('time'), ylabel('amplitude')
```



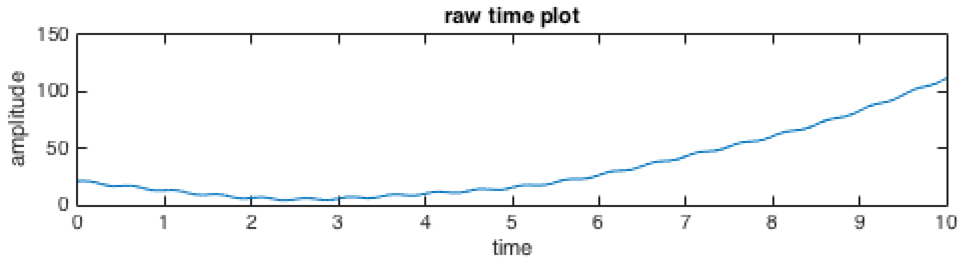
3. remove trend

```
y=detrend(y);
subplot(2,2,4), plot(ty,y), title('detrended time plot'),
 xlabel('time'), ylabel('amplitude')
```



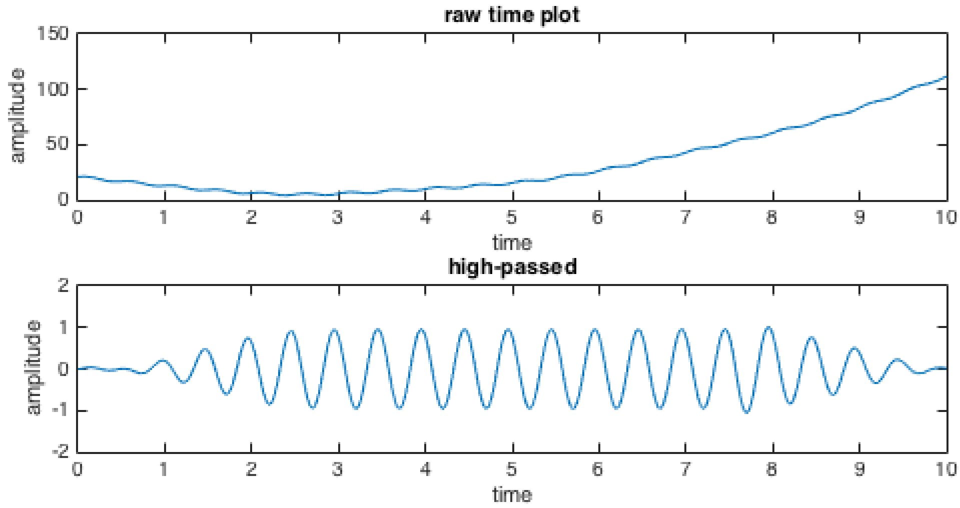
What about non-linear trends?

```
x=sin(2*pi*2*t) + sin(2*t);
x=x + 3 + t + 2*(t-3).^2;
figure
subplot(3,1,1), plot(t,x), title('raw time plot'), xlabel('time'),
 ylabel('amplitude')
```



1. High-pass filter

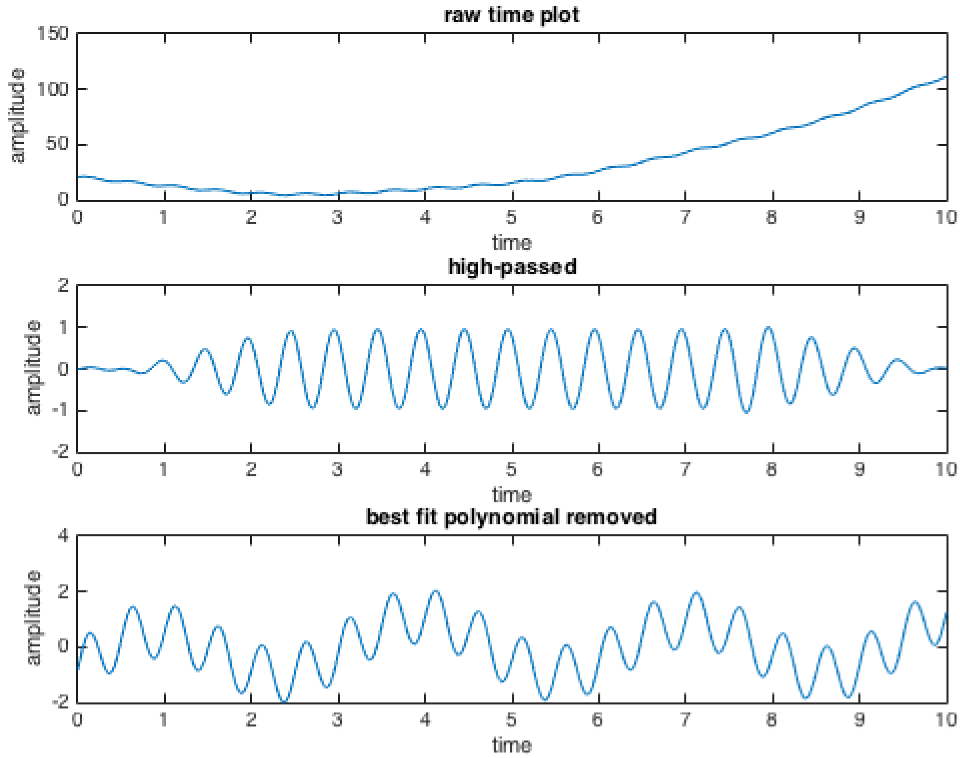
```
[b a]=butter(4, 0.03, 'high');
w=tukeywin(length(x),0.5)';
y=filter(b,a,x.*w);
subplot(3,1,2), plot(t,y), title('high-passed'), xlabel('time'),
 ylabel('amplitude')
```



2. Polynomial fit

```
p = polyfit(t, x, 3)
f = polyval(p, t);
y = x-f;
subplot(3,1,3), plot(t,y), title('best fit polynomial removed'),
 xlabel('time'), ylabel('amplitude')
```

```
p =

   -0.0067    2.1134  -11.5789   21.8205
```



Published with MATLAB® R2015a