ENVS422: Data Analysis of Environmental Records Week 6

Cross-spectra, covariances, Monte Carlo error estimation.

Correction

Regarding `fft` and `ifft`:

Two applications of `fft` do not get you back to where you started (apart from the multiplication by N). You also need to take the complex conjugate in order to get the phases to reverse correctly.

The result of this is that

```
fft(conj(fft(x))) = Nx
ifft(conj(ifft(x))) = x/N
ifft(fft(x)) = x
```

If you try , without the complex conjugate, and using size(x,1) to calculate N,

```
xx=fft(fft(x))./size(x,1)
```

you will get

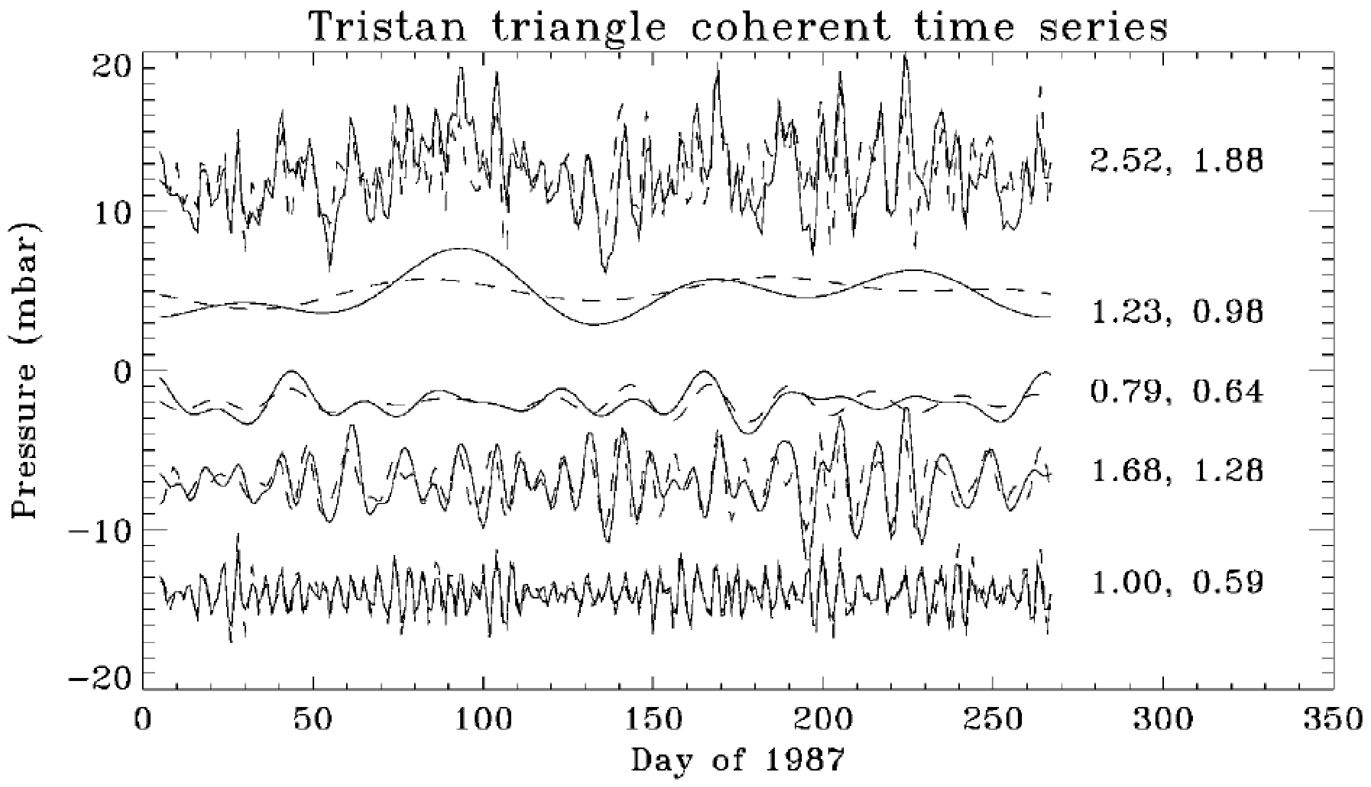
```
xx=circshift(flipud(x),1)
```

1	1
2	N
3	.
4	.
5	.
6	6
.	5
.	4
.	3
N	2
(1)	(1)

In other words, you get the original time series, but reversed in time, and then shifted one place to the right (with wrap around, so the last point becomes the first again). Like this

Correlations and cross-spectra

- You can calculate correlations between 2 time series.
- You could band pass filter them first, and obtain a correlation for a particular frequency band.
- You can do this in the frequency domain, and obtain broader information including lags.
- That is cross-spectral analysis.



Full time series

longer than 60 days

20-60 days

6-20 days

2-6 days

In this example, I calculated variance explained after band pass filtering in several different bands. I could have calculated correlations too.

But what happens if I use very narrow filters – then both curves would be pure sine/cosine waves. Allowing also for a phase lag between them, I would always get a correlation of 1. Correlation information becomes meaningless as the band pass becomes very fine.

Clearly there is some compromise between bandwidth and value of the estimate, just as there is for the calculation of spectra. This trade-off can be explored systematically with cross-spectral analysis.

Correlation between 2 vectors is $\boldsymbol{c} = \dfrac{\boldsymbol{a}\cdot\boldsymbol{b}}{|\boldsymbol{a}||\boldsymbol{b}\,|}$

$$\boldsymbol{c}^2 = \frac{(\boldsymbol{a}\cdot\boldsymbol{b})^2}{(\boldsymbol{a}\cdot\boldsymbol{a})(\boldsymbol{b}\cdot\boldsymbol{b})}$$

i.e.

$$c = \frac{a_1b_1 + a_2b_2 + \cdots}{\sqrt{(a_1^2 + a_2^2 + \cdots)(b_1^2 + b_2^2 + \cdots)}}$$

$$c^2 = \frac{a_1^2b_1^2 + a_2^2b_2^2 + \cdots}{(a_1^2 + a_2^2 + \cdots)(b_1^2 + b_2^2 + \cdots)}$$

Note, if you only have a single pair of numbers (a and b) instead of vectors, c=1 (1D vectors are in the same direction).

or the dot product of the normalized vectors. You can think of it as the cosine of the angle between the two vectors.

In the spectral domain, you can think of the squared amplitudes as the power at each frequency (FT of a multiplied by its complex conjugate, and similarly for b), and the cross term in the numerator becomes a cross-spectrum in which the FT of **a** is multiplied by the complex conjugate of the FT of **b**.

Writing these as

$$S_{ab} = F(a)F(b)^*$$ Sab=fft(a).*conj(fft(b))

$$S_{aa} = F(a)F(a)^*$$ Sbb=fft(a).*conj(fft(a)) = abs(fft(a)).^2

$$S_{bb} = F(b)F(b)^*$$ Sbb=fft(b).*conj(fft(b)) = abs(fft(b)).^2

$S_{ab} = F(a)F(b)^*$ Sab=fft(a).*conj(fft(b))

$S_{aa} = F(a)F(a)^*$ Sbb=fft(a).*conj(fft(a)) = abs(fft(a)).^2

$S_{bb} = F(b)F(b)^*$ Sbb=fft(b).*conj(fft(b)) = abs(fft(b)).^2

Note, using .*, we are not summing the products of elements, we are producing a vector (as a function of frequency) in which each element is a product of elements.

We can define an analogue of the squared correlation, called (magnitude) squared coherence (or squared coherency):

$$C^2(f) = \frac{|S_{ab}|^2}{S_{aa}S_{bb}}$$

which measures the equivalent of a correlation as a function of frequency. As noted above, this is meaningless when done with straight FTs as defined here (the S values at each frequency are just complex numbers, so C^2 would be 1), but using instead estimates of the spectra using averages of FTs of windowed segments, a compromise between spectral resolution and noise can be found. The cross spectrum S_{ab} is therefore not defined as above, but as the average of a number of such products of FTs from multiple realizations of the time series.

In addition to the squared coherency: $$C^2 = \frac{|S_{ab}|^2}{S_{aa}S_{bb}}$$

which describes the fraction of variance of a which can be explained by a scaled and phase-lagged version of b (or vice versa), the cross spectrum S_{ab}also contains information about the optimal scaling (analogous to a regression coefficient), known as gain, and the optimal phase lag at each frequency.

These quantities can be calculated in Matlab using

```
mscohere
```

and

```
cspd
```

$$\text{Gain}[a/b] = \frac{|S_{ab}|}{S_{bb}}$$

$$\text{Gain}[b/a] = \frac{|S_{ab}|}{S_{aa}}$$

Note the two gains are not simply inverses of each other – they should be thought of as regression coefficients of a on b, and of b on a.

`mscohere` and `cspd`

`mscohere` directly gives the squared coherency, but nothing else. `cspd` works in the same way as `pwelch`, and gives the cross spectrum but nothing else – to calculate squared coherency, phase lag, and gain, you also need the individual power spectra.

Also, to judge whether two time series are "strongly" or "weakly" coherent, you need some measure of how likely it is that the squared coherency will be found by chance (like the confidence level `pxxc` in `pwelch`). Matlab does not provide this statistic, which requires a knowledge of the number of "equivalent degrees of freedom", which depends on window shape, and overlap, as well as the number of windowed segments.

For this reason, I have written a wrapper function to make calculation of squared coherency and phase lag more straightforward, and to provide a value of the squared coherency corresponding to your chosen confidence level.

```
function [csq,xamp,xphs,f,nseg,cconf]=cwhxspec(x,y,dt,winwid,nperfreq,pconf)
%
% calculates amplitude, phase, squared coherence from
% cross spectrum of time series x and y,
% time step dt using hamming windows of width winwid
% (measured in time units) overlapped by 50%.
%
% nperfreq specifies the number of estimates to give for each
% independent frequency.
%
% pconf specifies the confidence limit desired for csq, as a
% percentage (can be an array of values)
%       e.g. if you want to know the value of csq which has a 5%
%       chance of being exceeded, choose pconf = 95
%
% returns:
%  csq  = squared coherency
% xamp  = amplitude of cross spectrum
% xphs  = phase of cross spectrum in degrees.
%         Positive means y lags behind x.
%    f  = corresponding frequencies (cycle per unit time)
% nseg  = number of windowed segments used
% cconf = value of csq which is not exceeded by chance,
%         pconf percent of the time.
%
```

```
function [csq,xamp,xphs,f,nseg,cconf]=cwhxspec(x,y,dt,winwid,nperfreq,pconf)
%
% … comments deleted
%
fracoverlap=0.5;
lenx=size(x,1);
lenseg=floor(winwid/dt);
noverlap=floor(fracoverlap.*lenseg);
nseg=floor((lenx-noverlap)/(lenseg-noverlap));
ww=hamming(lenseg);
[pxy,f]=cpsd(x,y,ww,noverlap,lenseg.*nperfreq,1./dt);
[pxx,f,pxxc]=pwelch(x,ww,noverlap,lenseg.*nperfreq,1./dt,'ConfidenceLevel',pconf(1)./100);
[pyy,f,pyyc]=pwelch(y,ww,noverlap,lenseg.*nperfreq,1./dt,'ConfidenceLevel',pconf(1)./100);
csq=abs(pxy).^2./(pxx.*pyy);
xphs=angle(pxy)*180/pi;
xamp=abs(pxy);
%
% calculate the chosen confidence value(s)
%
edof=nseg*0.69*4./3;
% the factor 0.69 is derived empirically from Monte Carlo
% simulations with 50% window overlap.
cconf=1.-(1-pconf/100).^(1/(edof-1));

end
```

How to use the confidence limit:

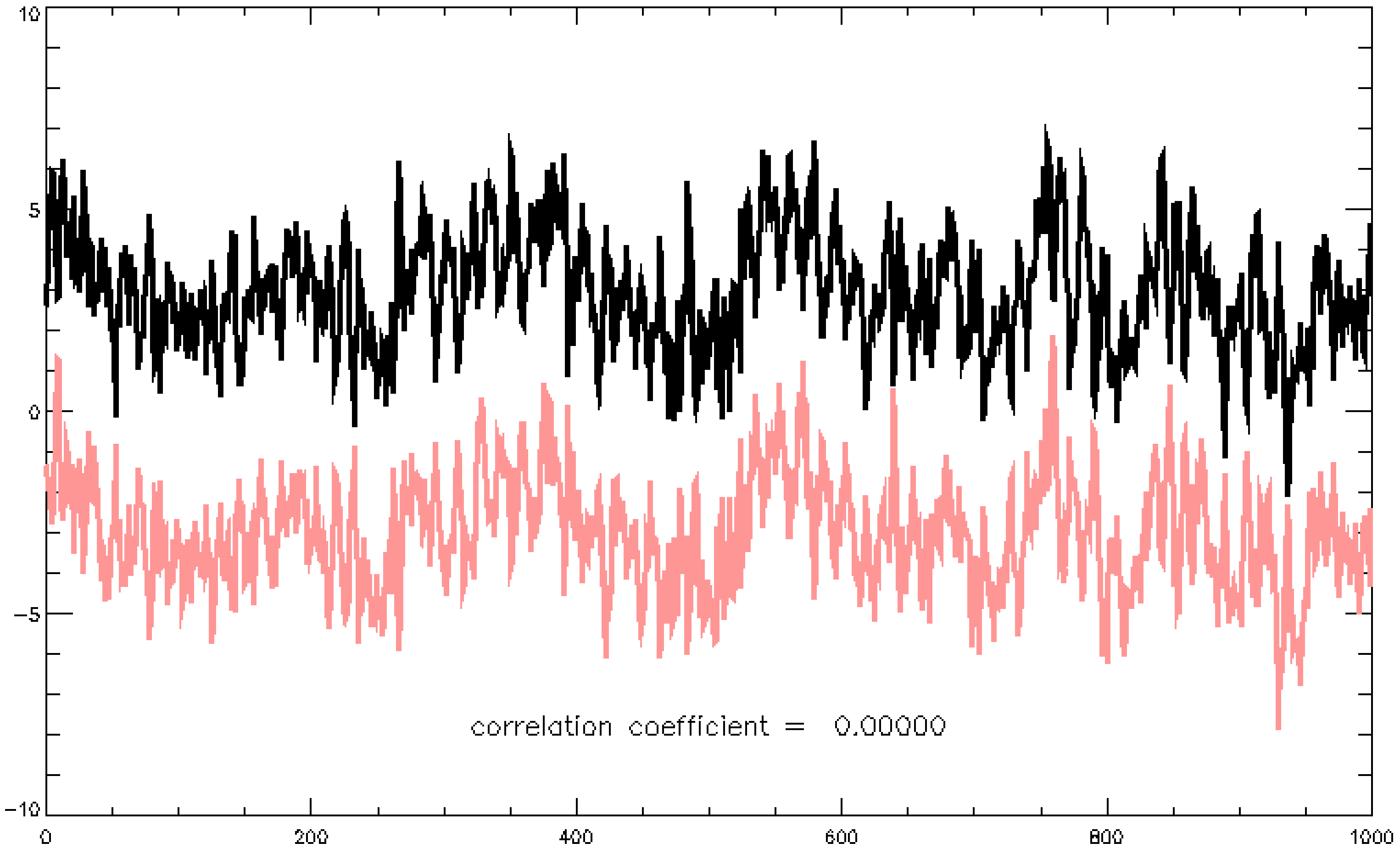
If you choose pconf = 95, this gives the value of C^2 which exceeds 95% of values obtained by chance.

This means you should expect 1 in 20 of the independent values to exceed this value by chance. If you have 200 frequencies, you would expect purely by chance that 10 of them would be “significant” by this definition of exceeding the threshold (use nperfreq = 1 to obtain the independent frequencies).

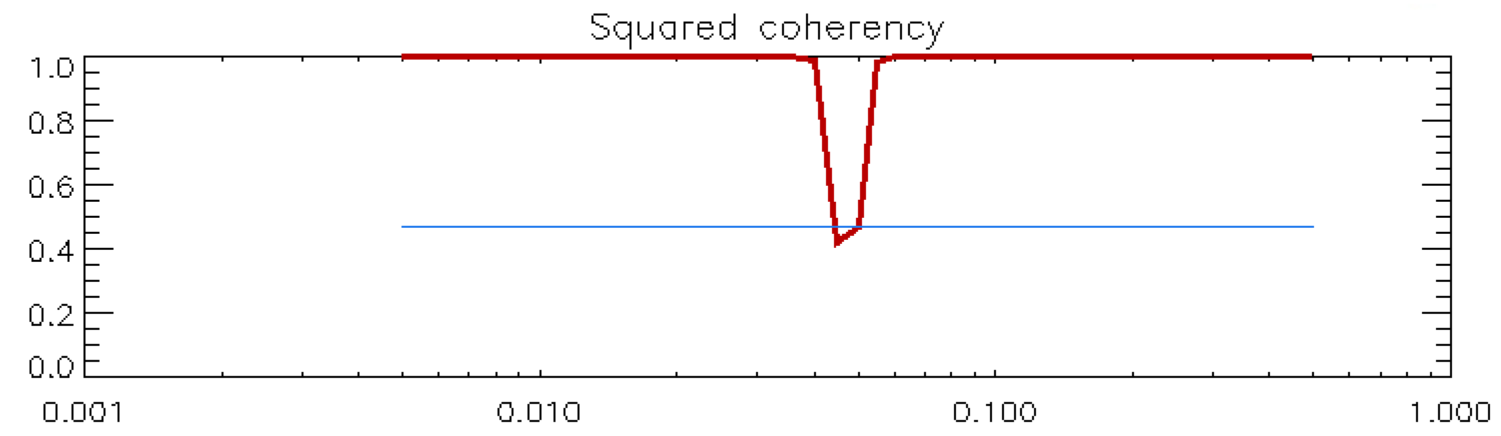
It is a qualitative guide only. Statistics alone cannot tell you that something is “significant”, though they can help guide you in deciding whether something is worth pursuing.

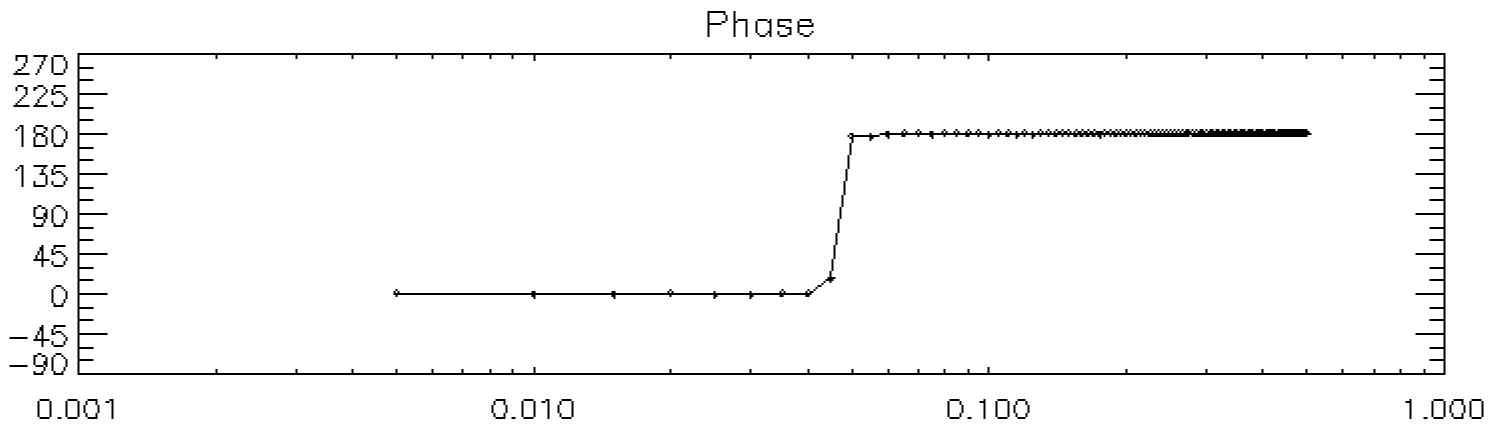
Low coherence also means poorly-determined phase. You can use the chosen confidence value as a threshold to decide which points you might be able to interpret the phase from.

Cross spectra are a good way of seeing whether relationships among variables change on different time scales. An extreme example:

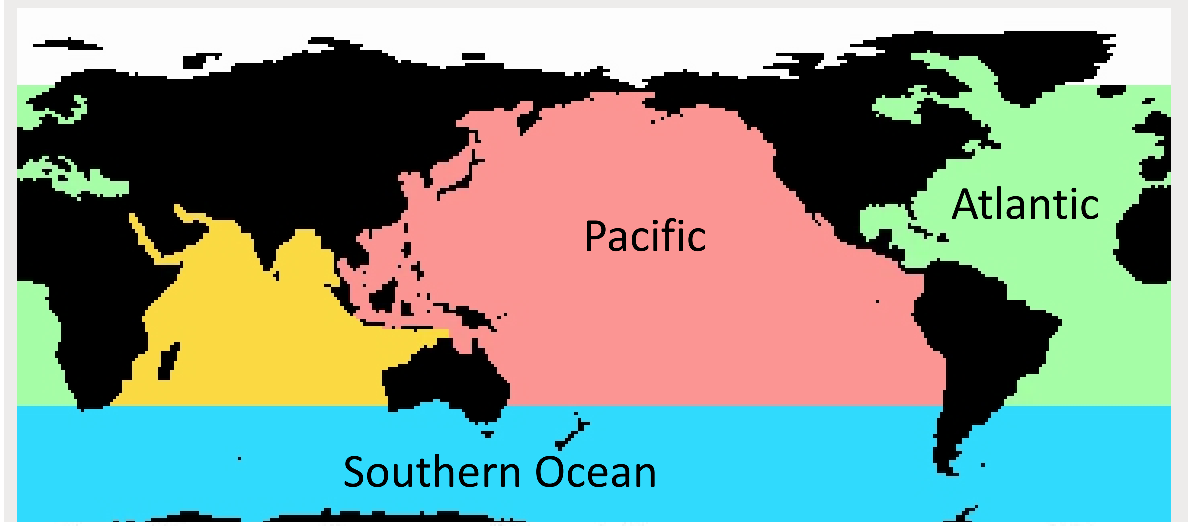


At first sight it seems hard to believe these two curves are not correlated. The problem is that correlation does not distinguish different time scales.



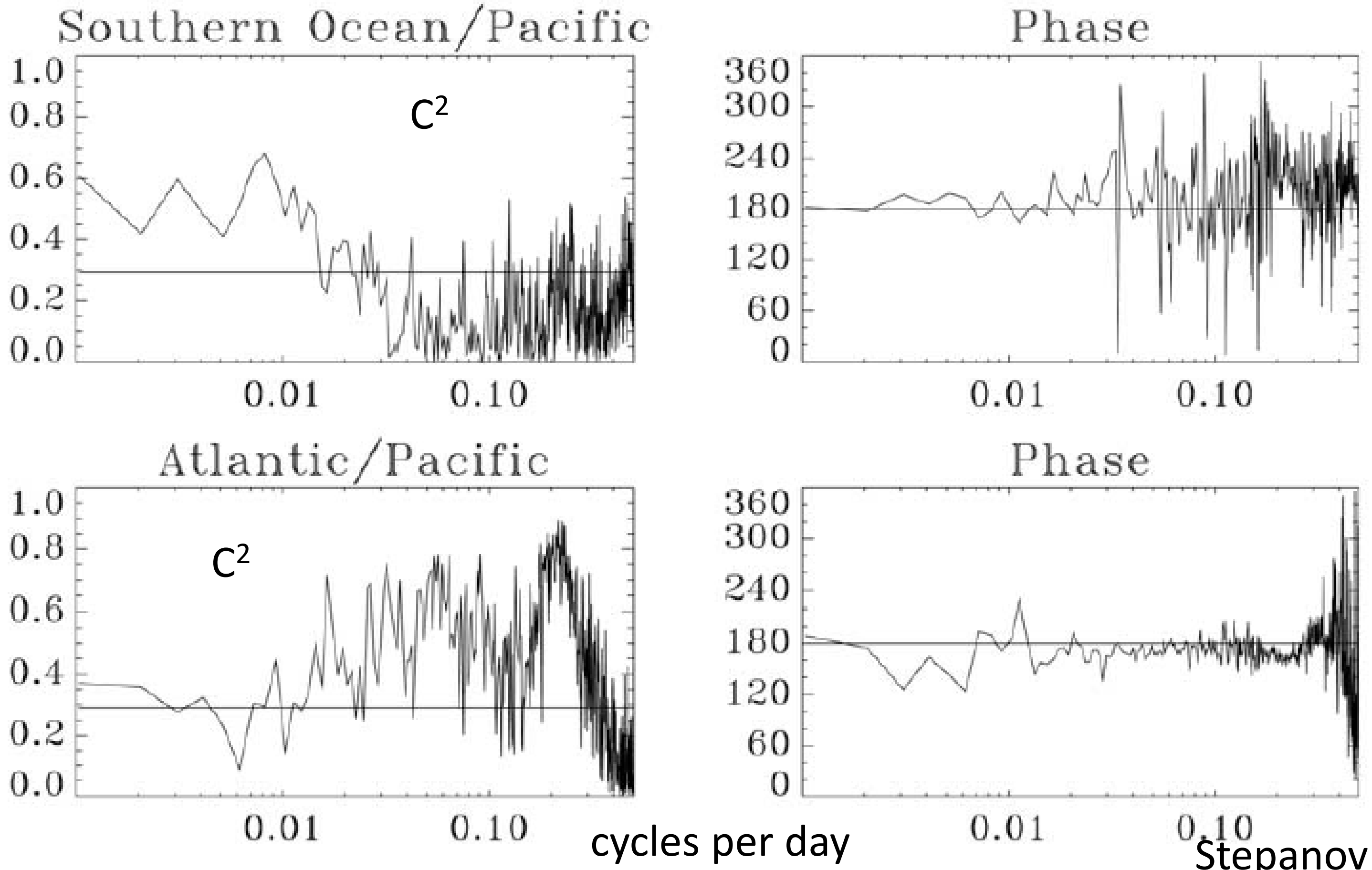


The cross spectrum shows that low frequencies are strongly correlated, and high frequencies are strongly anticorrelated. The amplitudes happen to be such that these correlations exactly cancel.



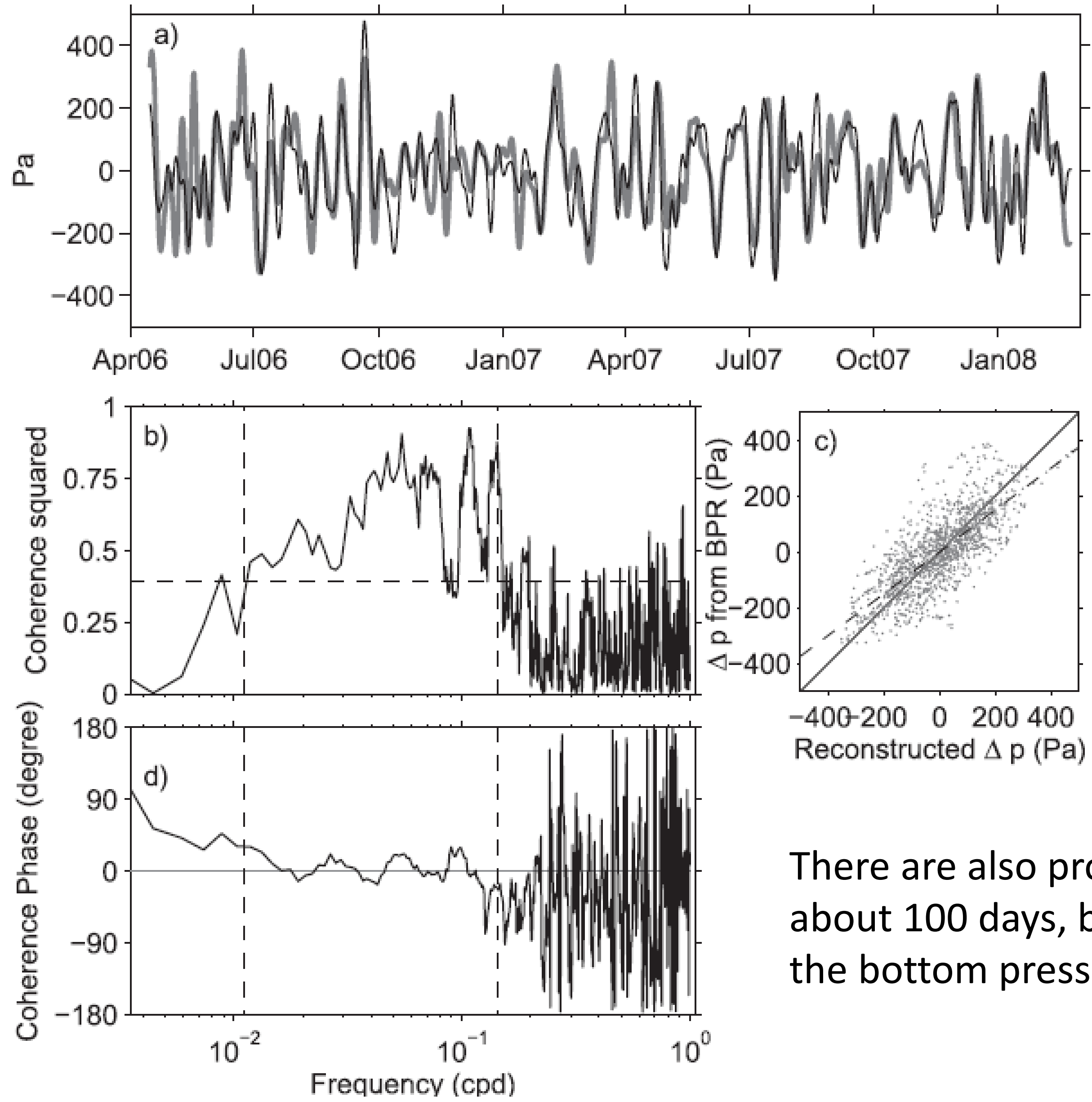
Time series of bottom pressure averaged over different ocean basins.

Pacific exchanges with Atlantic at short periods and Southern Ocean at long periods



Phase shows that this is a direct exchange between basins not (e.g.) a progressive wave moving from Atlantic to Pacific to Indian.

Stepanov and Hughes, JGR 2006

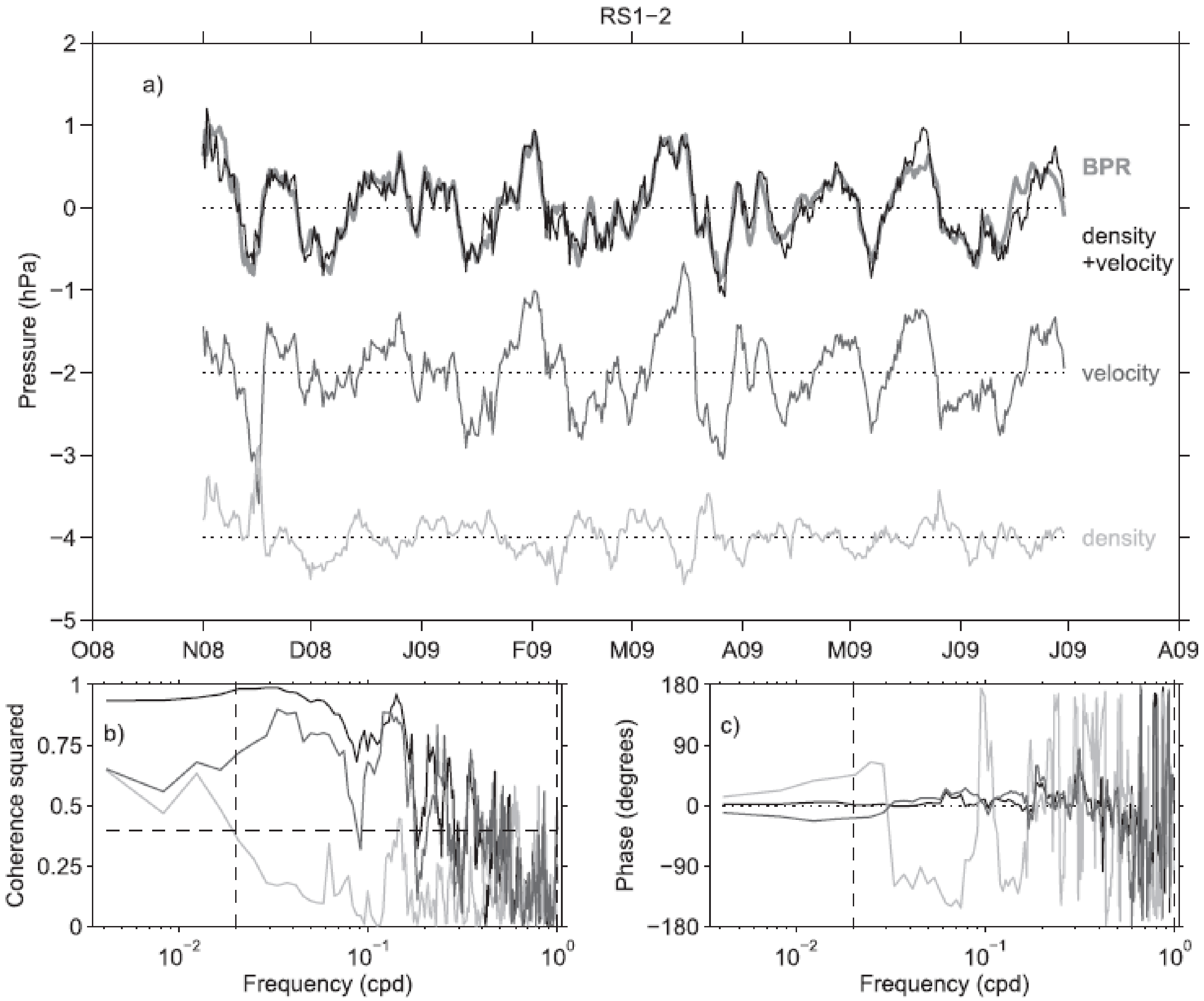


Time series of pressure difference directly measured, and inferred from current and density measurements

Squared coherency is low at periods shorter than about 7 days (currents are not geostrophic at periods near to a day or less).

There are also problems at period longer than about 100 days, but this is because of drifts in the bottom pressure measurement.

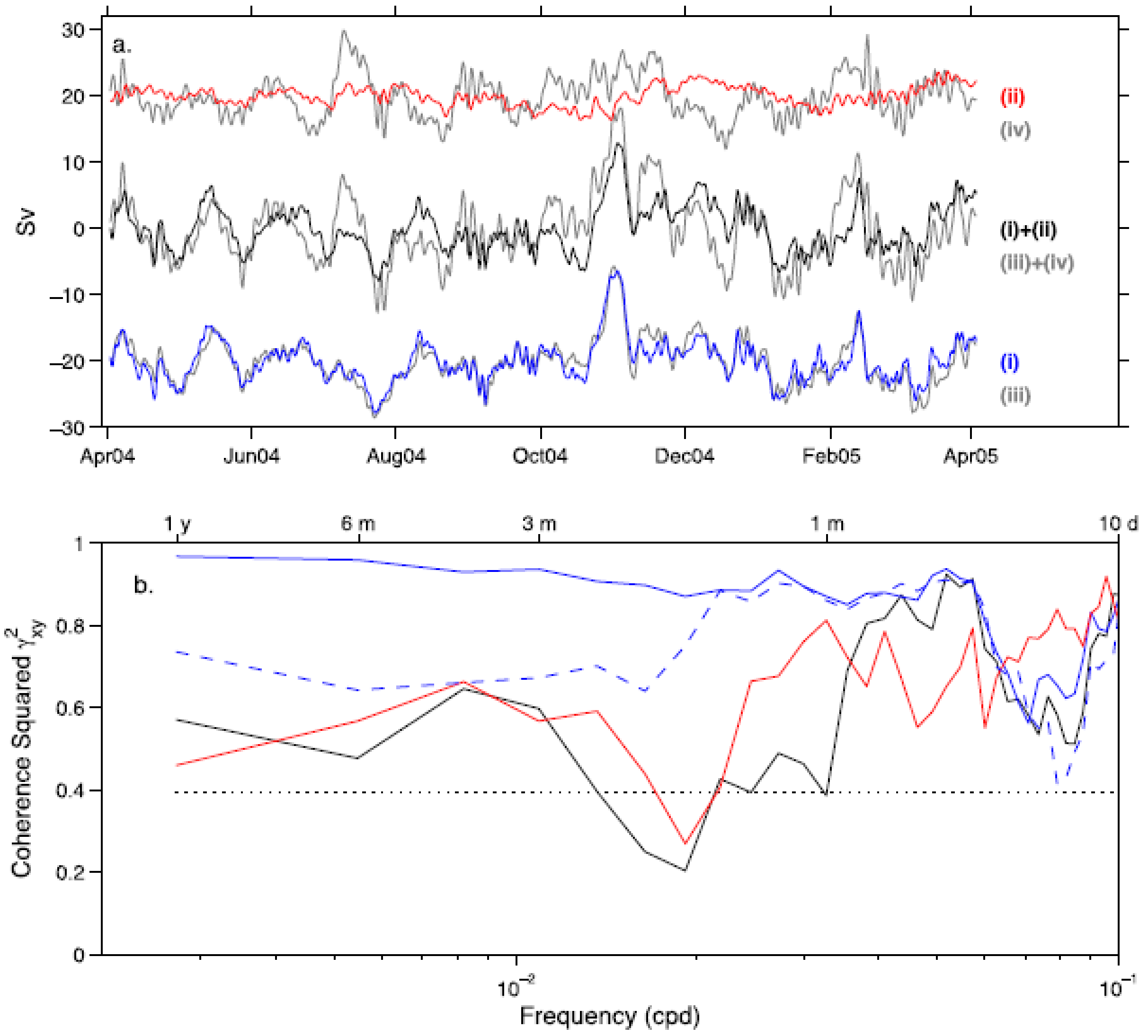
Elipot et al., JPO 2013



Similar analysis (from a different place), looking at how current velocity and density each contribute to the pressure difference.

Density matters less than current, but becomes more important at longer time scales, and improves the phase when added to the current effect.

Hughes et al., JAOT 2013



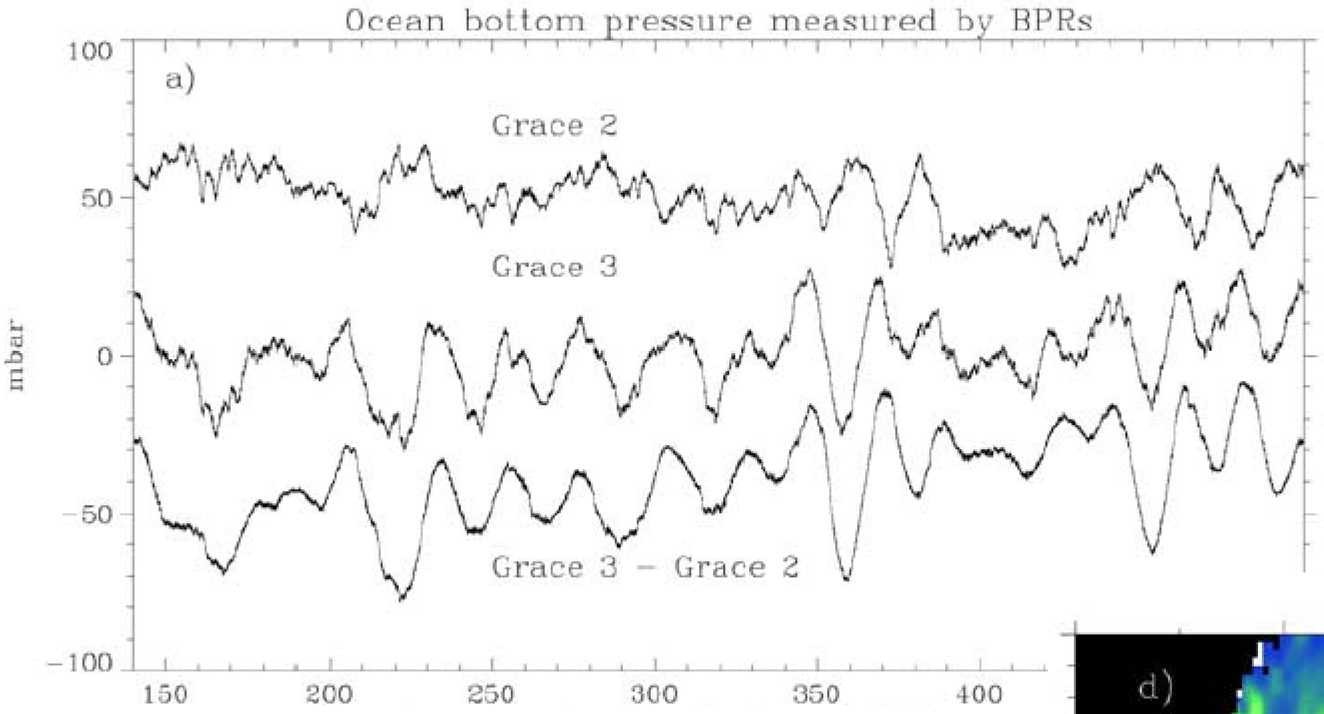
Atlantic MOC at 26N, inferred from density (plus currents on the west), or from **pressure**.

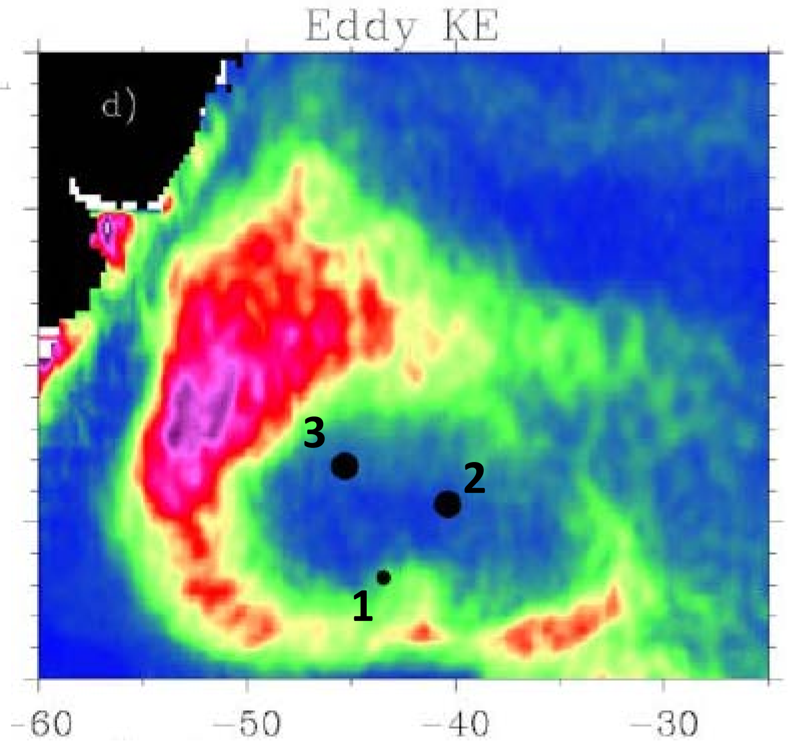
Western contribution, **Eastern contribution**, **Sum**.

Western contribution is well reproduced at periods longer than about 2 weeks.

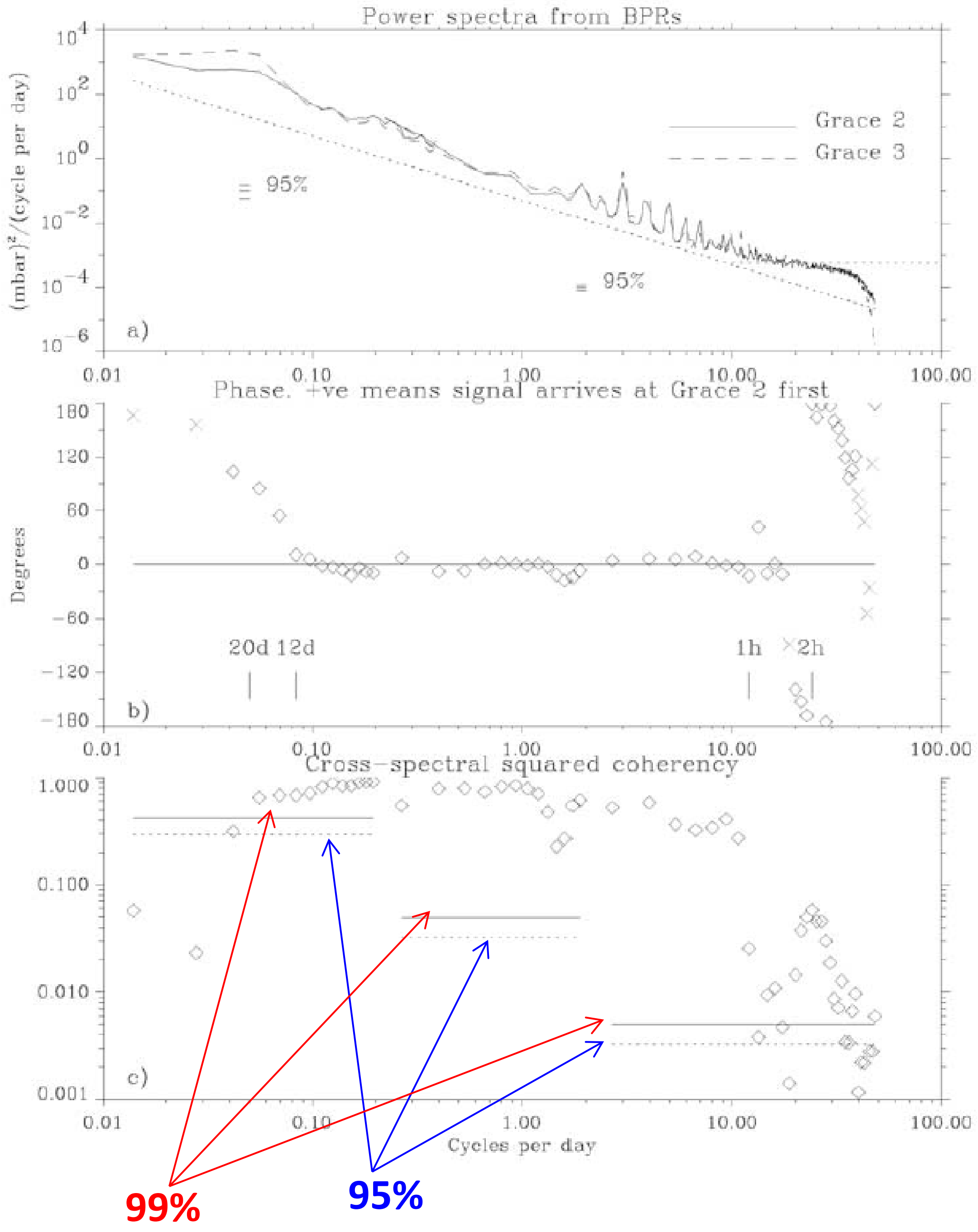
Eastern contribution is useful, but much less good due to lack of current measurements.

Elipot et al., JPO 2014





Hughes et al., JGR 2007

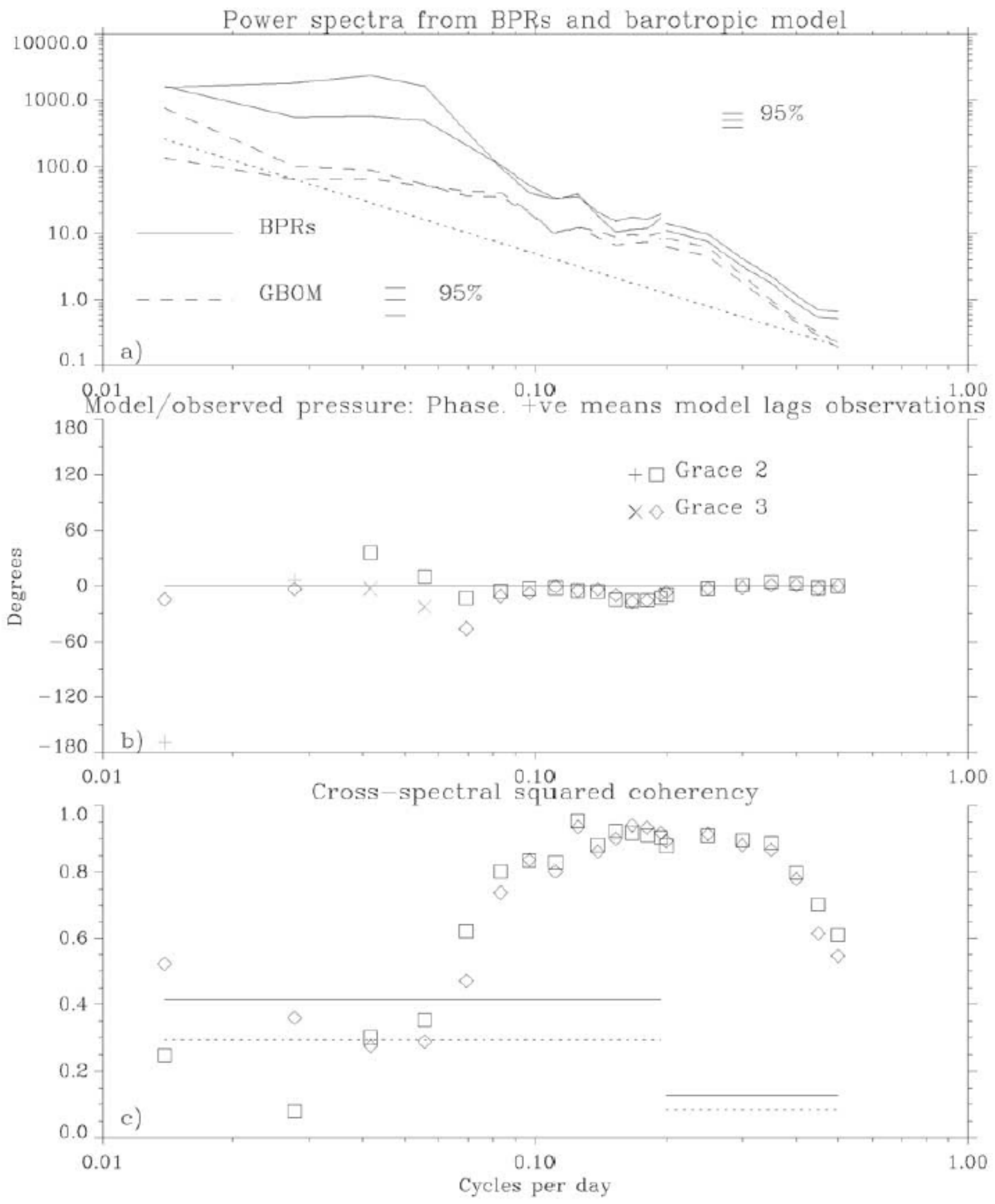


Log-log power spectral density. Bulge at 20 day period barely visible (it dominates on a variance preserving plot)

No phase lag between about 1 hour and 12 day periods. Lag at shorter periods (waves take half an hour to propagate between sites). Also a clear lag at 20 days.

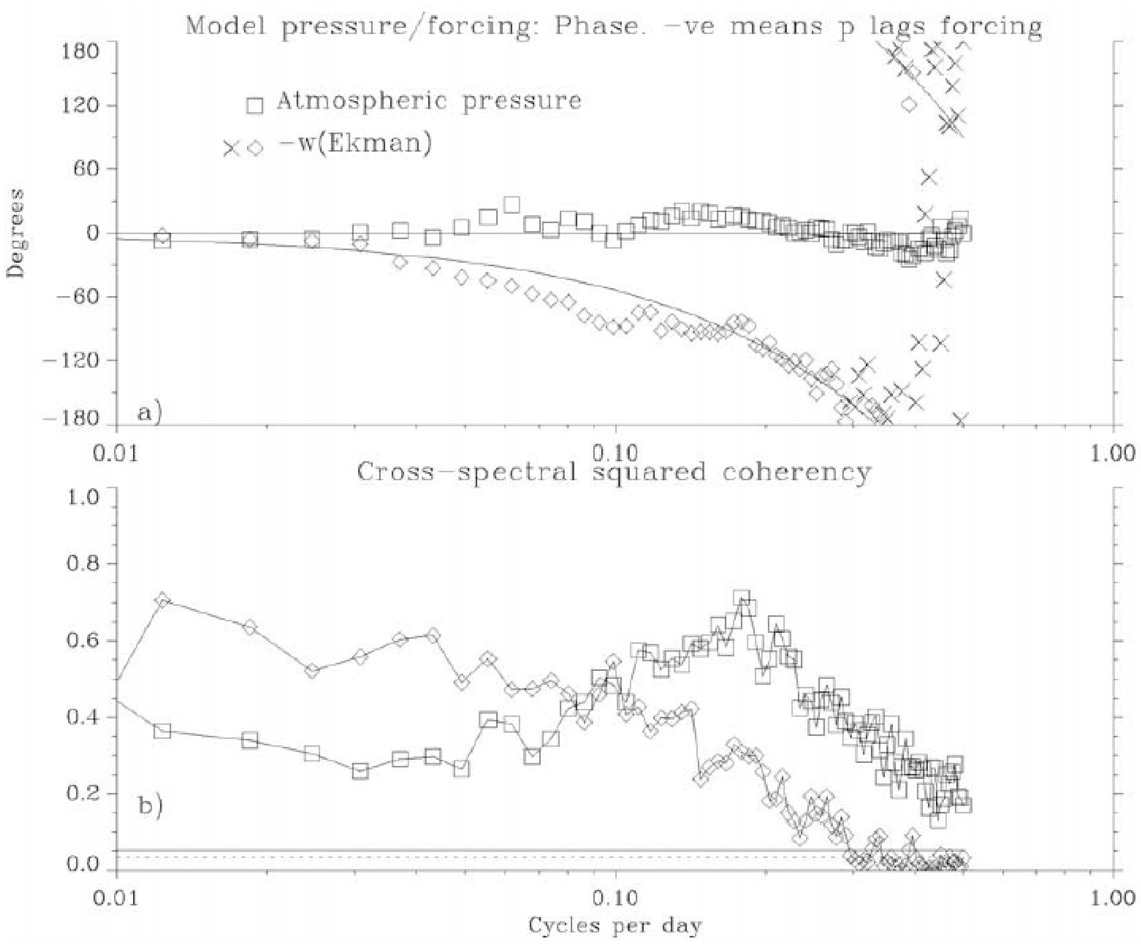
Strong coherence over much of the range. Plot uses different window widths for different frequency ranges.

Hughes et al., JGR 2007



Model simulations capture the in-phase part of the variability well, but fail completely to capture the 20-day mode which has a phase lag.

Hughes et al., JGR 2007



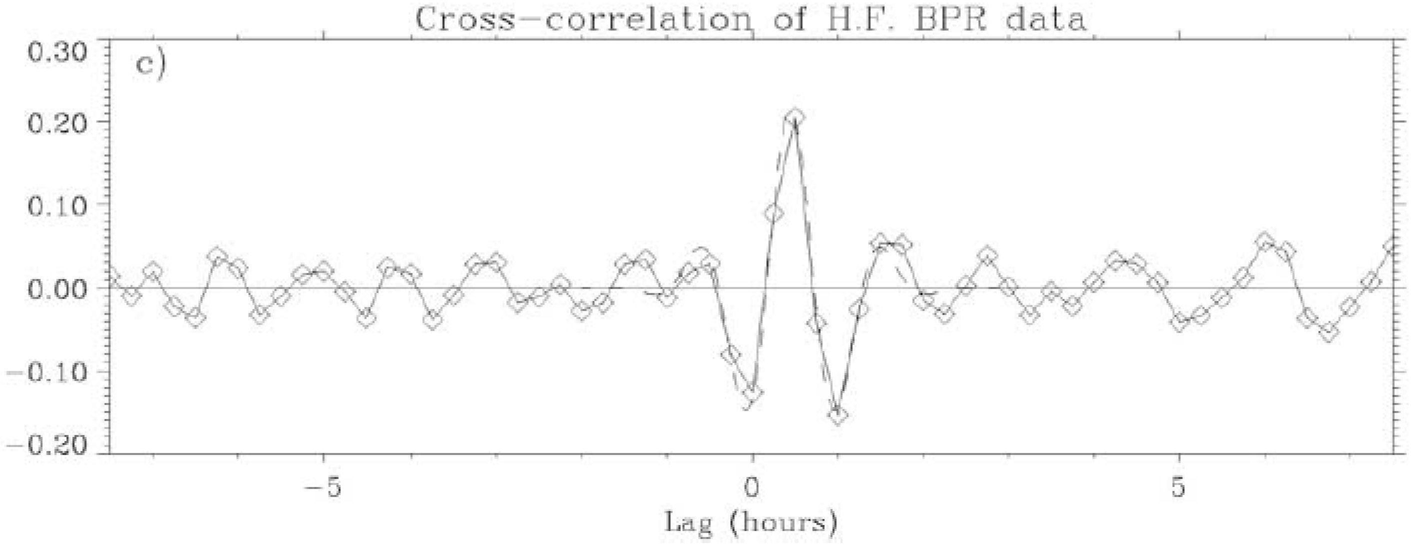
In the model, is it atmospheric pressure or Ekman pumping (from wind stress) which is responsible for exciting the coherent mode?

Both seem to contribute, but atmospheric pressure matters more at high frequencies, and wind stress at low frequencies. The response to wind stress is delayed by about 1.5 days.

Note the ambiguity in phase. A phase lag of 240 degrees could be a phase lead of 120 degrees. Such ambiguity can sometimes be resolved by looking at lagged correlations.

Hughes et al., JGR 2007

Lagged correlations: The cross correlation between one time series and the lagged version of the other. In this case, after high pass filtering to pass only periods shorter than 2 hours.



It becomes clear that the signal is arriving at Grace 3 after Grace 3, with about 30 minute lag. If the lag had been longer (e.g. 1.5 hours), a signal with 2 hour period would have had the opposite phase, although the lagged correlations would still have identified the correct relationship.

Hughes et al., JGR 2007

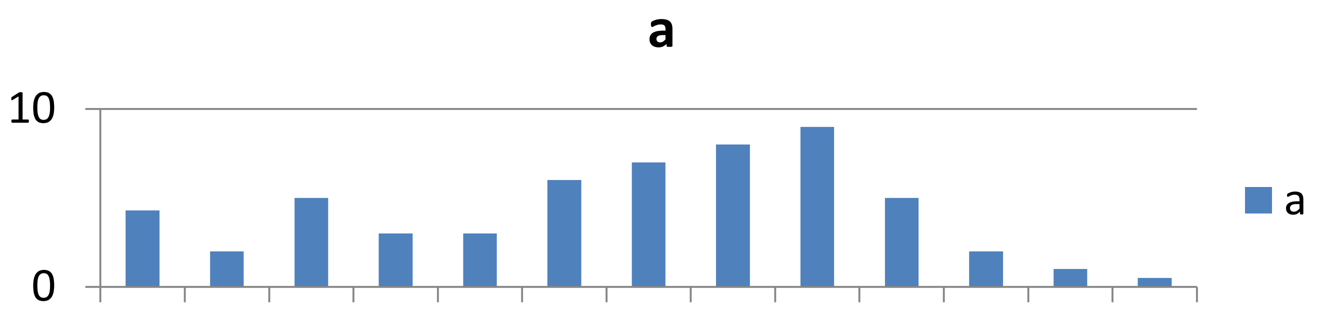
Correlations, lagged autocorrelations, lagged cross correlations, covariances and convolutions.

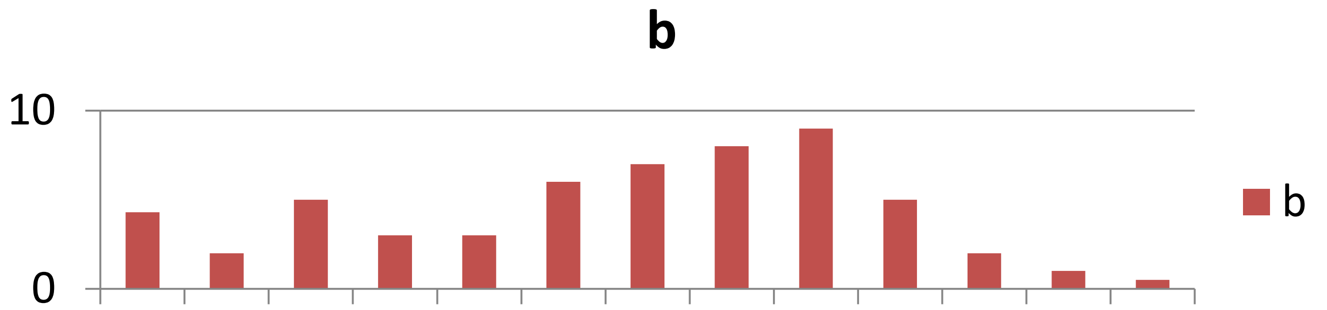
$$\boldsymbol{c} = \frac{\boldsymbol{a}\cdot\boldsymbol{b}}{|\boldsymbol{a}||\boldsymbol{b}\,|} = \frac{\boldsymbol{a}}{|\boldsymbol{a}|}\cdot\frac{\boldsymbol{b}}{|\boldsymbol{b}|}$$

$$\boldsymbol{c}^2 = \frac{(\boldsymbol{a}\cdot\boldsymbol{b})^2}{(\boldsymbol{a}\cdot\boldsymbol{a})(\boldsymbol{b}\cdot\boldsymbol{b})}$$

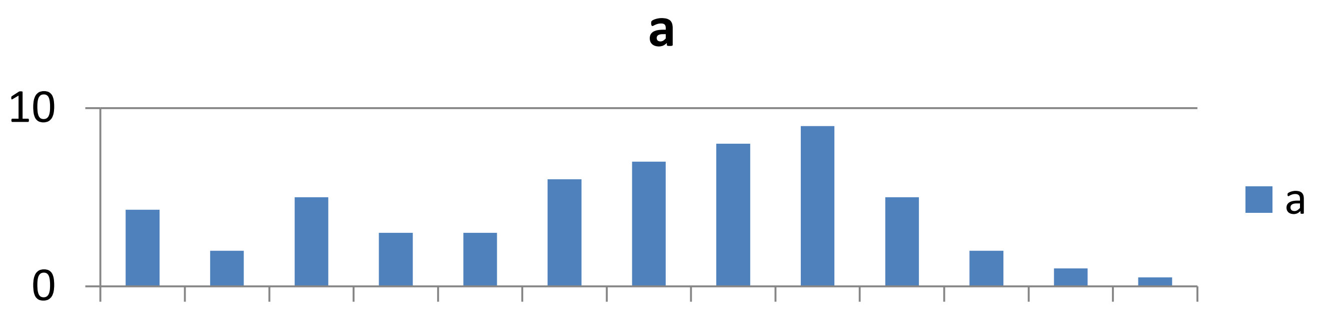
$$c = \frac{a_1b_1 + a_2b_2 + \cdots}{\sqrt{(a_1^2 + a_2^2 + \cdots)(b_1^2 + b_2^2 + \cdots)}}$$

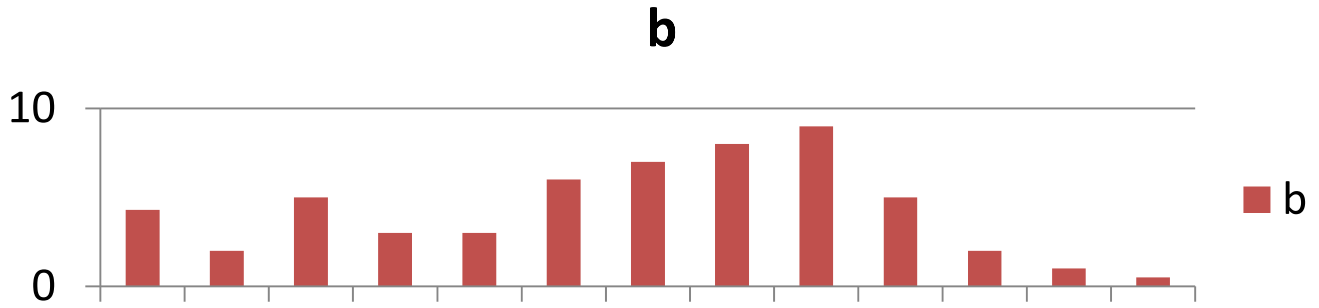
$$c^2 = \frac{a_1^2b_1^2 + a_2^2b_2^2 + \cdots}{(a_1^2 + a_2^2 + \cdots)(b_1^2 + b_2^2 + \cdots)}$$





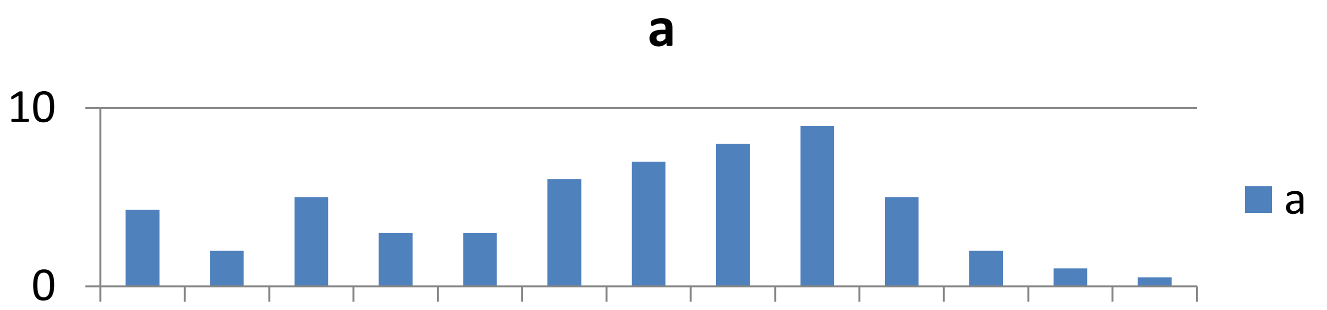
Correlations, lagged autocorrelations, lagged cross correlations, convolutions, spectra and cross spectra.

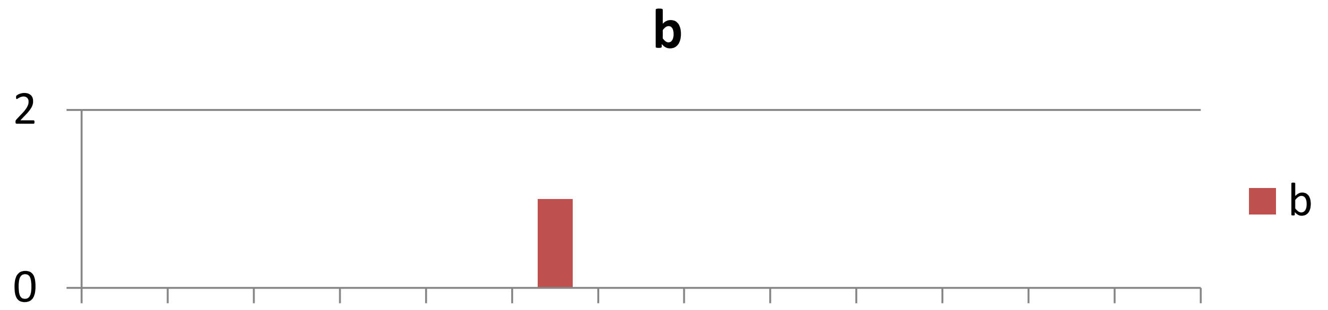




Correlation: Divide each time series by its standard deviation (i.e. normalize it), then calculate the sum of the squares of the corresponding elements. If the two time series are exactly proportional, the correlation is 1.

Correlations, lagged autocorrelations, lagged cross correlations, convolutions, spectra and cross spectra.

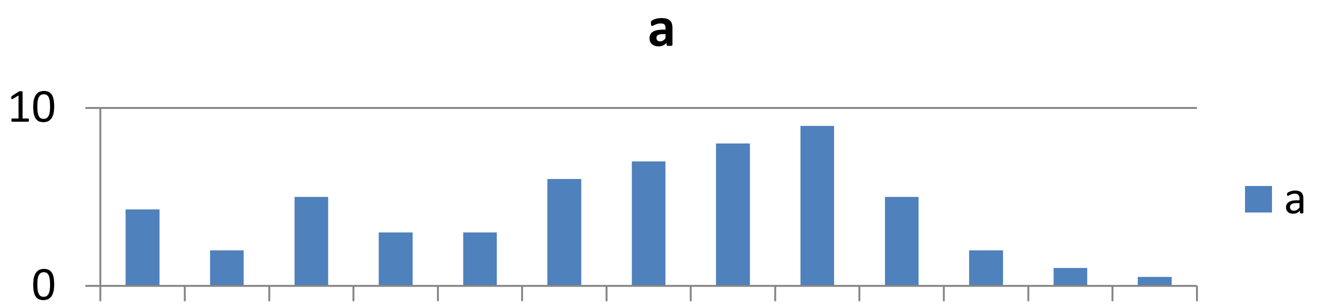


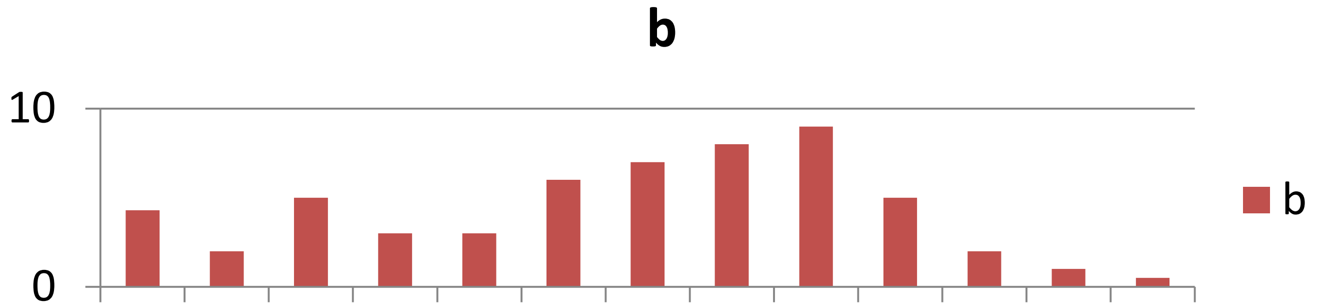


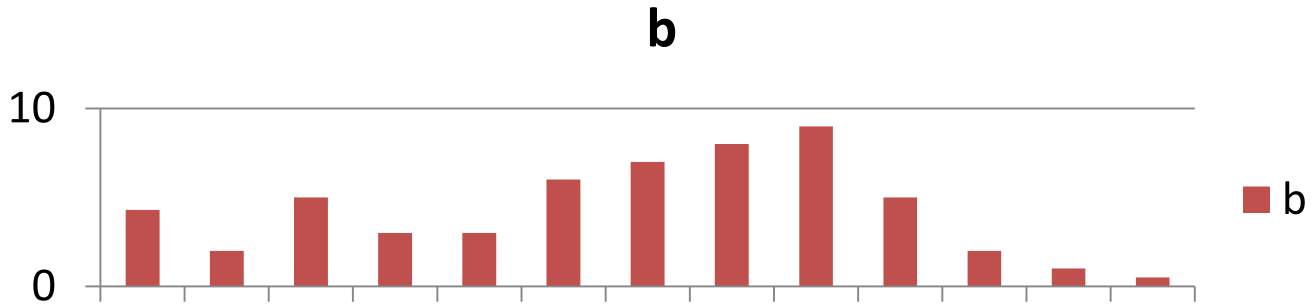
Correlation: If **b** is zero except for a single element then, after normalization, that element becomes 1 (i.e. the discrete version of a delta function).

The correlation is then simply the value of a at that time (divided by the root mean square of **a**).

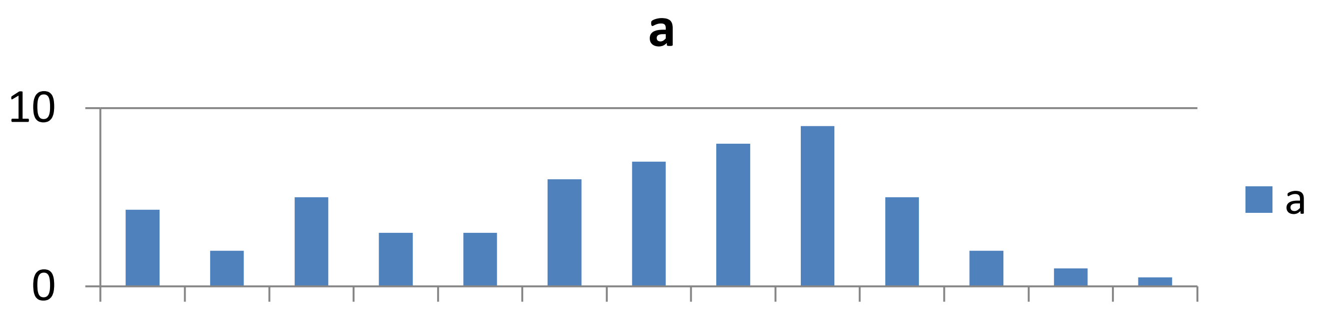
Lagged correlation at lag n: correlate **a** with **b** after shifting **b** n places to the right. If **b** = **a** this is called the lagged autocorrelation (which must be 1 at lag zero, since **a** is perfectly correlated with itself).

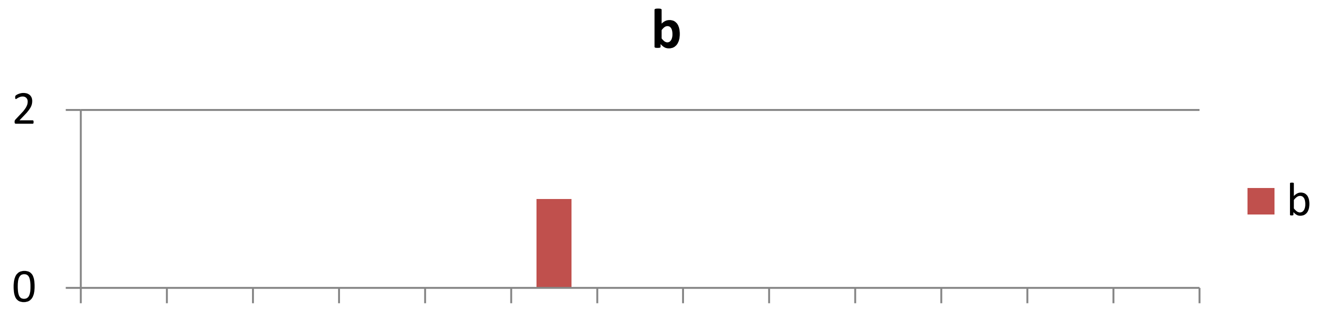


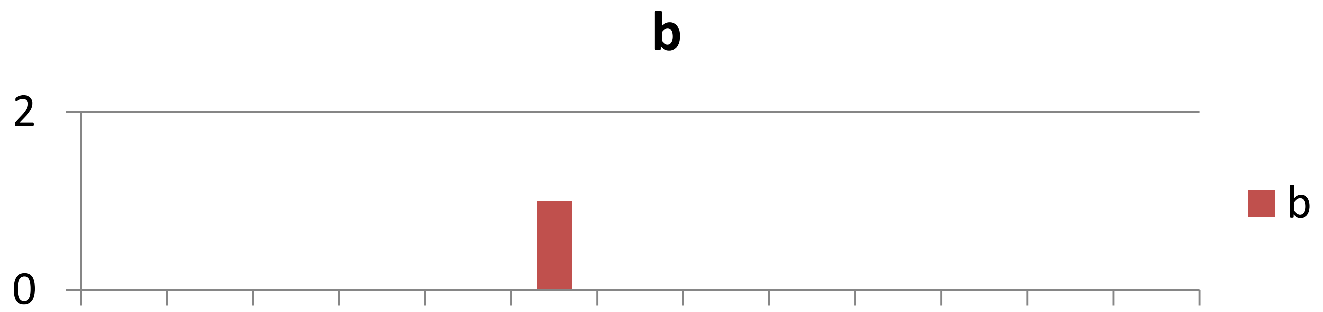




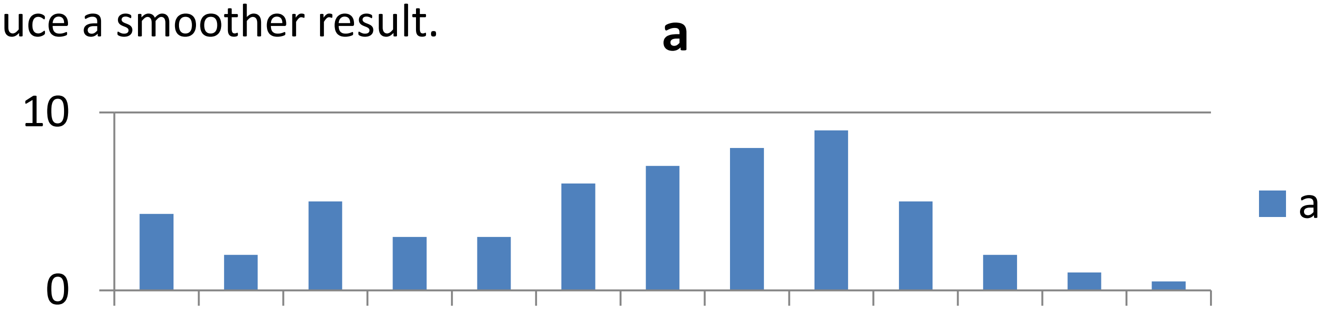
Lagged cross covariance of a vector (**a**) with the delta function, is simply a way of picking out the elements of a one by one. The cross covariance is simply a copy of **a**.

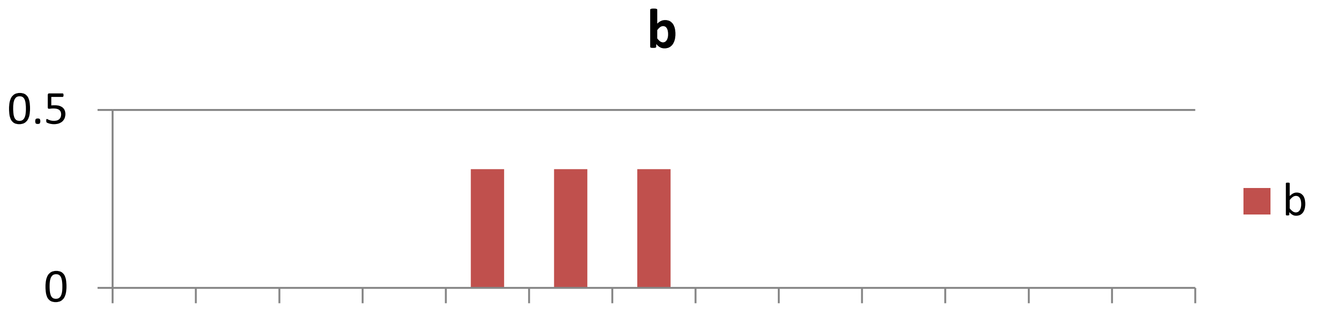


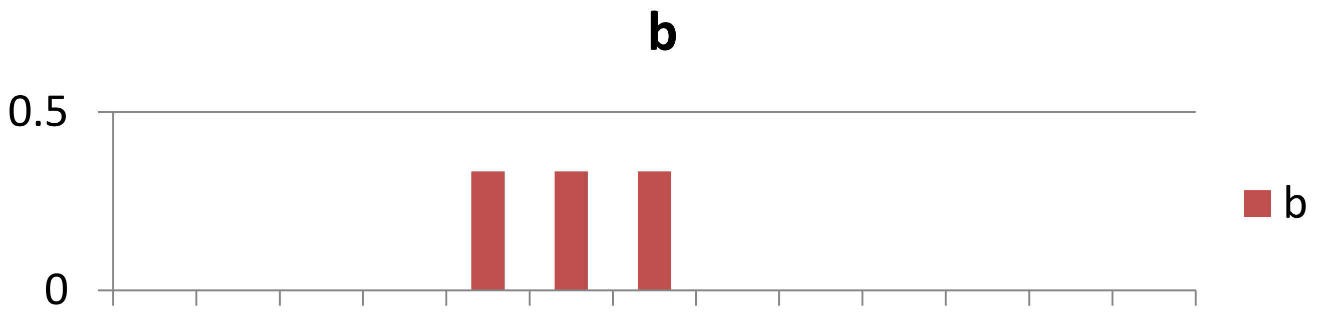




Instead, if b is constant over a certain range (and normalized), the cross covariance is a running average of **a** (a boxcar smoother). Thinking this way, you can think of the cross covariance as a way of "smoothing" **a**, using **b** as a "kernel" (or vice versa). Of course, if **b** has oscillations, this may not actually produce a smoother result.

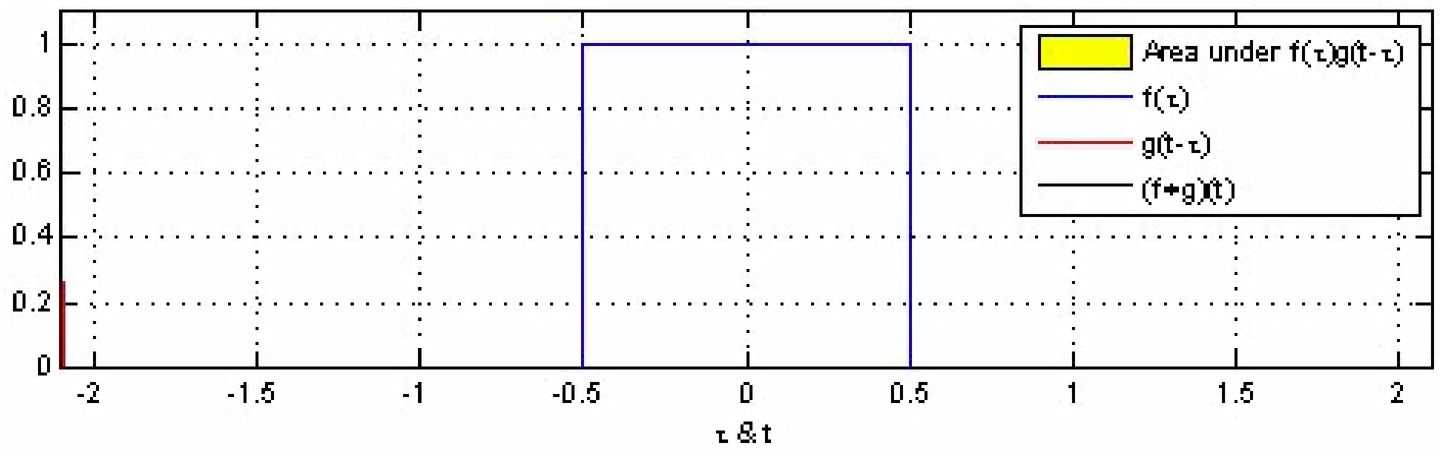






In the same way, the lagged autocorrelation or autocovariance of a time series can be thought of (apart from the different normalizations) as using **a** as a smoothing kernel on itself!

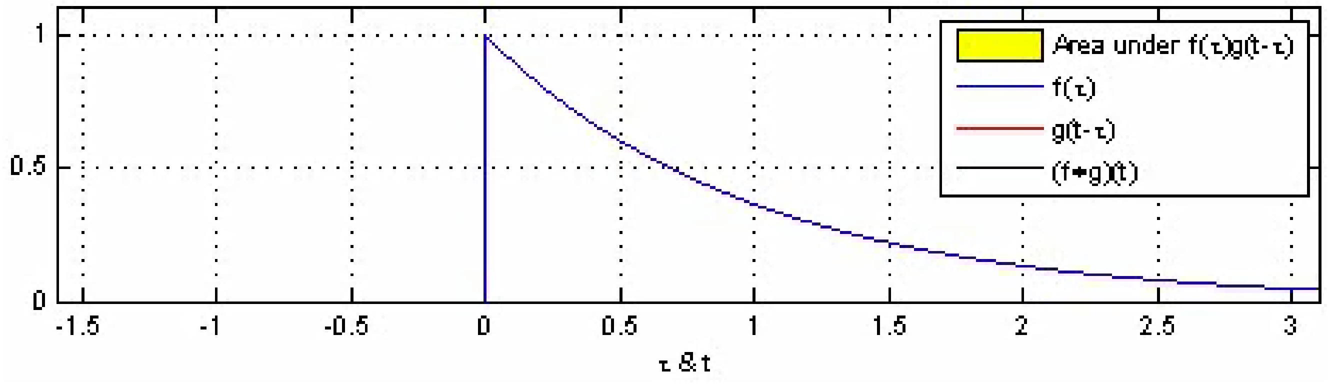
When thought of in these terms, this process is known as the convolution of two time series (actually, convolution also involves a reversal in time compared to calculating covariance).



Animation from Wikipedia – illustrates the convolution of 2 “top hats” or boxcars.

The convolution can be thought of as measuring the area under the product of the two time series (with one lagged), as a function of the lag.

Animation from Wikipedia –the convolution of a top hat with a more complicated function.



An annoying point of terminology:

Sometimes, what we call covariance is referred to as correlation, and what we call correlation is referred to as correlation coefficient.

Matlab does this: `xcorr`, produces a lagged covariance function (as does `xcov`, the difference being whether the mean is subtracted out first). If you want a lagged correlation function, you need to call `xcorr` with the option `'coeff'`.

corrcoef does produce a correlation coefficient (but not lagged).

Why is this all relevant here?

There's a neat relationship between convolution and Fourier transforms (the convolution theorem):

FT(convolution of a with b) = FT(a) × FT(b)

where the product is element by element (.* in Matlab).

There is also a related "correlation theorem", which accounts for the reversal of one of the time series in a lagged correlation compared to convolution:

FT(lagged covariance of a with b) = FT(a) × FT(b)$^+$ where the + indicates complex conjugate.

in Matlab: `FFT(lagcovariance) = FFT(a).*conj(FFT(b))`

```
%fouconv
y=randn(1001,1);
for i=2:1001;
    y(i)=0.95*y(i-1)+y(i);
end;
y=y/std(y,1);
x=(-500:500)';
g=exp(-x.^2/150^2);
yg=y.*g;
```

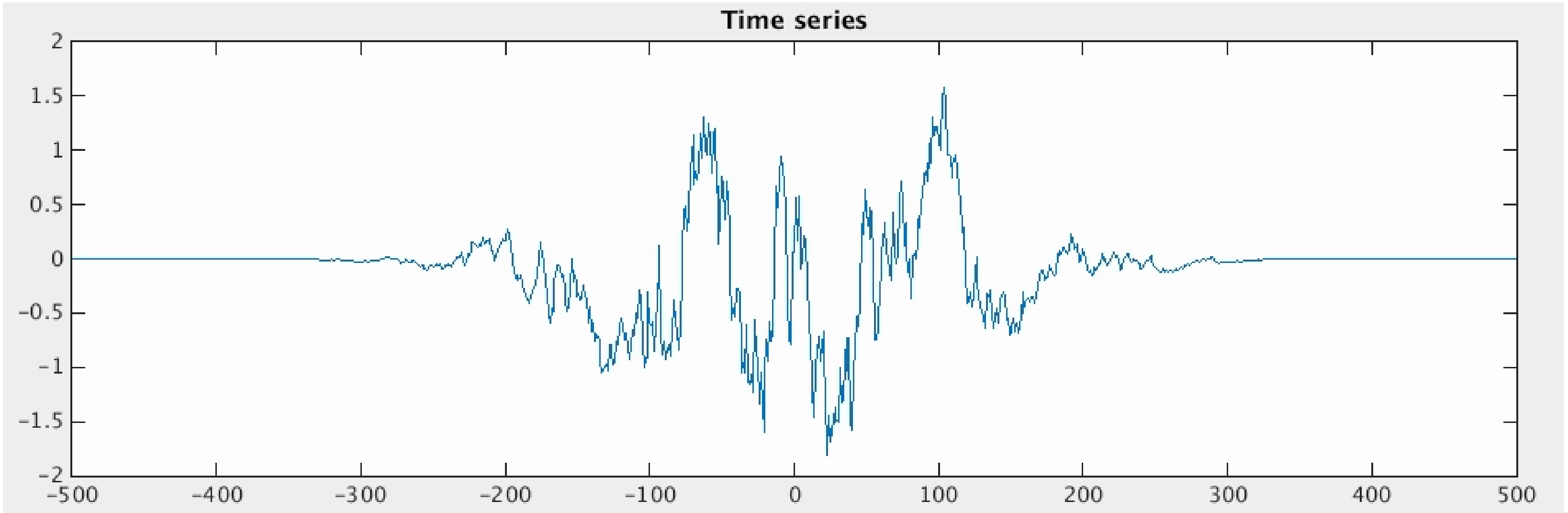
Create a time series with some kind of structure (I multiply it by a Gaussian so that it reduces to zero at the edges, to avoid issues with wrap-around)

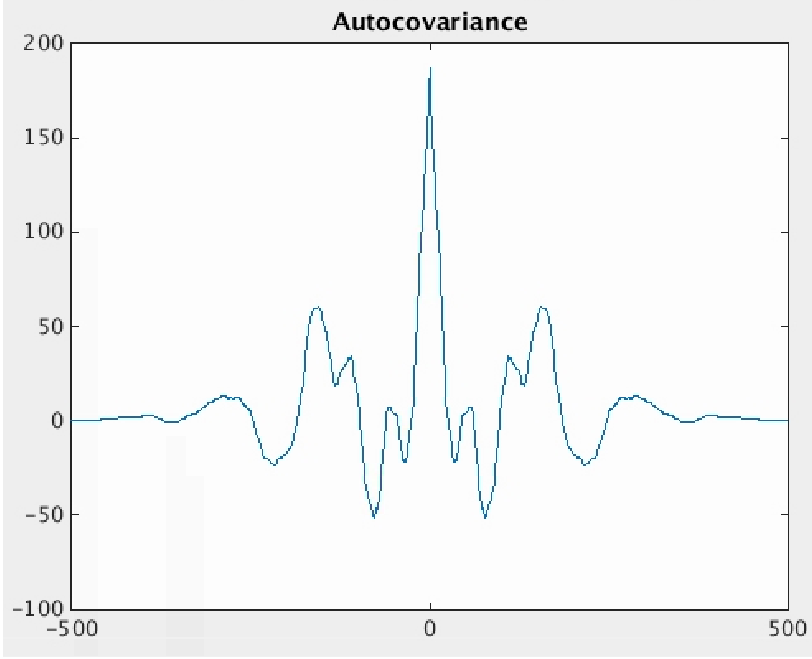
```
ycov=zeros(size(yg));
for i=-500:500;
  ycov(i+501)=yg'*circshift(yg,i);
end
```

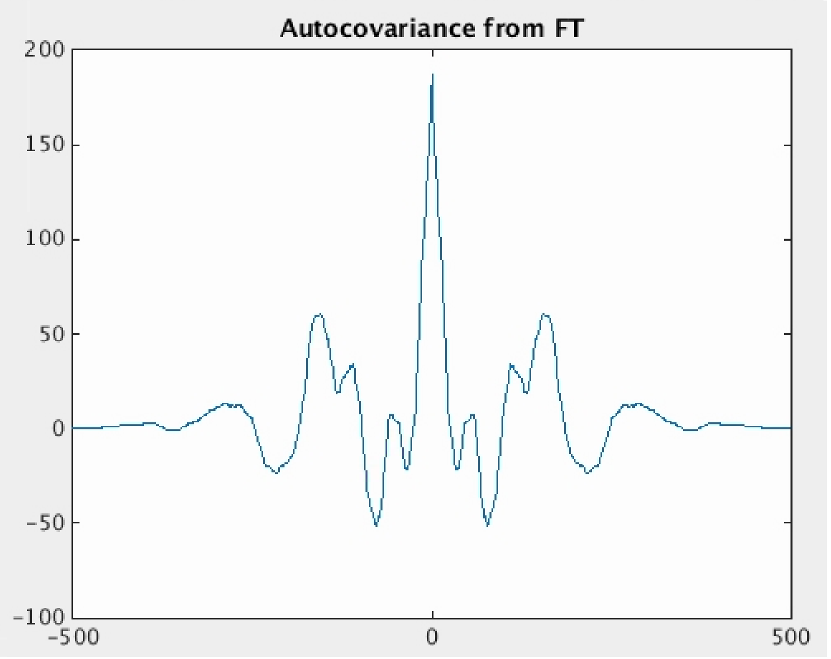
Calculate autocovariance by multiplying the time series by a shifted version of itself, and summing the products of elements.

```
ygft=fft(yg);
ysp=ygft.*conj(ygft);
yfcov=ifft(ysp,'symmetric');
```

Calculate autocovariance by performing fft, calculating periodogram (approximation to spectrum), and then performing inverse fft.







(NB these are not normalized by dividing by the number of elements, as we defined covariance before)

```
%fouconv2
x=randn(1001,1);
for i=2:1001;
%     y(i)=0.95*y(i-1)+y(i);
    x(i)=0.8*x(i-1)+x(i);
end;
x=x/std(y,1);
y=0.8*circshift(x,100)+randn(1001,1)*0.2;
tlag=(-500:500)';
```

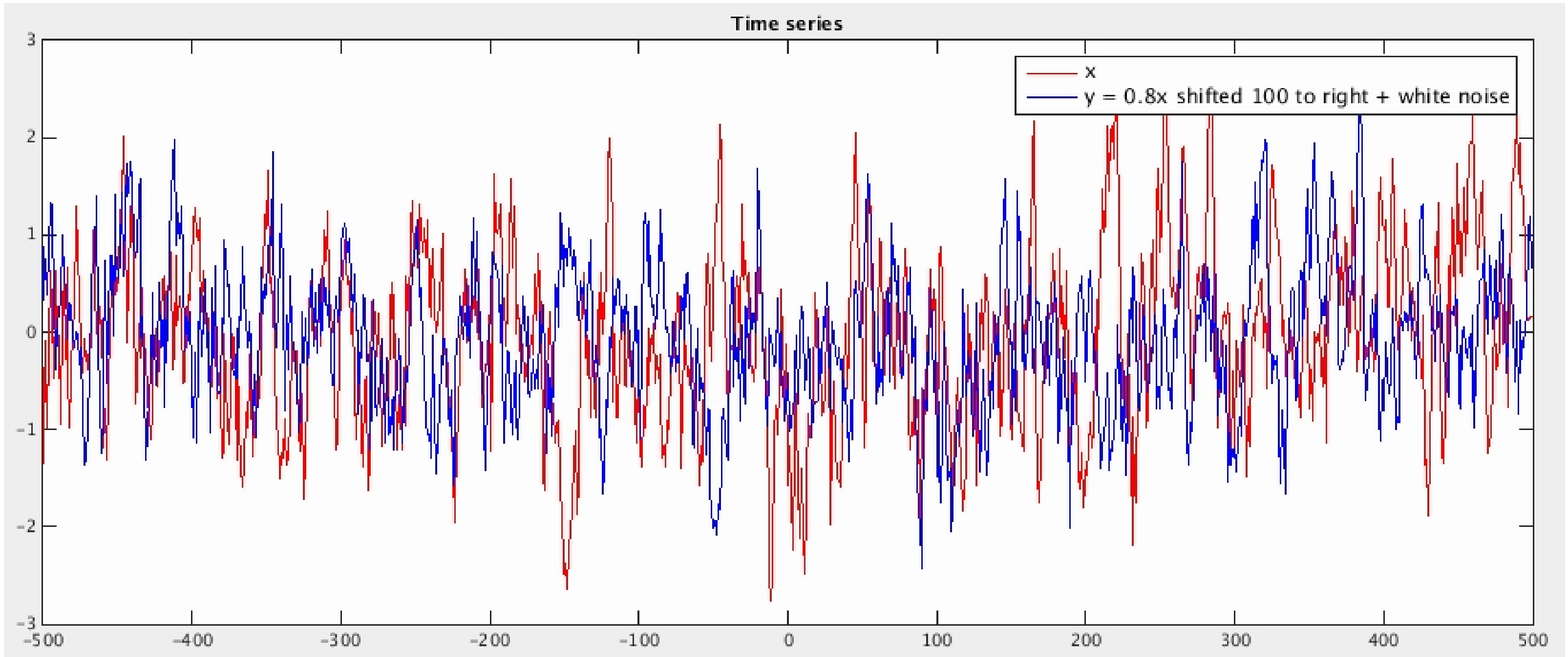
Create a time series x with some kind of structure , and y is x, shrunk by a factor 0.8, shifted 100 to the right, with white noise added.

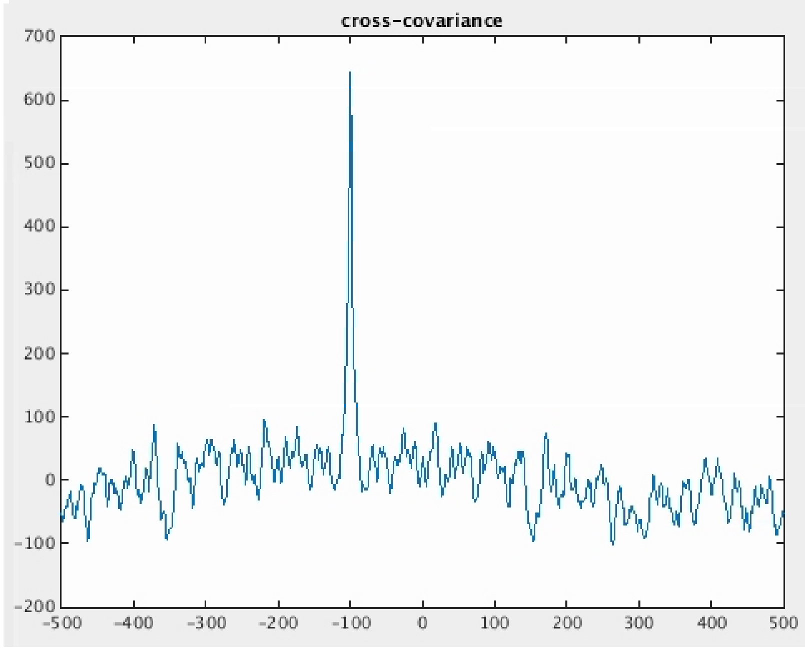
```
xycov=zeros(size(tlag));
for i=-500:500;
  xycov(i+501)=x'*circshift(y,i);
end
```

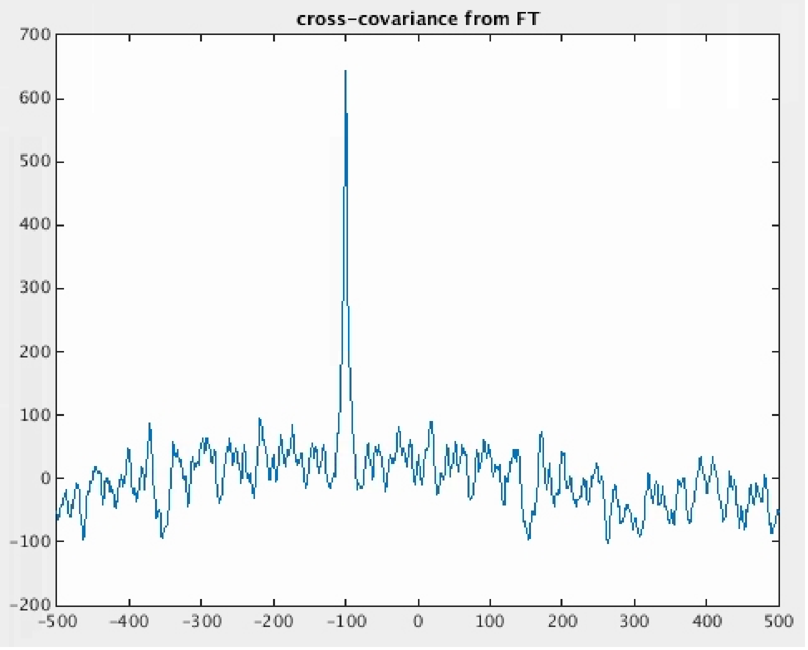
Calculate cross-covariance by multiplying x by a shifted version of y, and summing the products of elements.

```
xyfcov=ifft(fft(x).*conj(fft(y)),'symmetric');
```

Calculate cross-covariance by performing ffts, calculating product (as in cross-spectrum calculation) and then performing inverse fft.


Time series
x
y = 0.8x shifted 100 to right + white noise
3
2
1
0
-1
-2
-3
-500
-400
-300
-200
-100
0
100
200
300
400
500


cross-covariance
700
600
500
400
300
200
100
0
-100
-200
-500
-400
-300
-200
-100
0
100
200
300
400
500


cross-covariance from FT
700
600
500
400
300
200
100
0
-100
-200
-500
-400
-300
-200
-100
0
100
200
300
400
500

Autocovariance is the Fourier transform of the power spectral density, and vice versa

Cross-covariance is the Fourier transform of the cross power spectral density, and vice versa.

Recall, we needed an estimate of error covariance if we were to obtain meaningful errors when performing a least squares fit in the case when errors are not white noise.

This relationship, in principle, gives us a way to the estimate those error covariances, if we know the power spectrum of the errors.

However, there are complications. Since the dominant time scale for f^{-2} (red) noise is the longest one available, the autocovariance function varies strongly with length of time series. Similarly there are peculiarities in the autocovariance for flicker noise (f^{-1}).

What this does tell us though, is that the information we need about errors is contained in the power spectrum (and cross spectra if we are simultaneously fitting multiple time series).

Assessing significance: what do we mean?

There is no way for statistics to tell us whether a result is significant or not.

All we can assess is the likelihood of our results happening by chance, **given some model for what we mean by chance.**

For example, in least squares fitting, it is simple to obtain an estimate of the likely bounds on the estimated parameters **given the assumption** that the curve we are fitting to consists of the function we are fitting plus uncorrelated white noise.

Or, we can obtain a more realistic estimate if we can supply a covariance estimate for the structure of the “errors”. Note, in this context, “errors” means everything apart from the function being fitted, and the bounds on our estimate **still depend** on that covariance estimate being correct.

Assessing significance: what do we mean?

Is a correlation significant or not? Again, all we can say is that the correlation would have an x% probability of occurring by chance, **given some model** of the random/stochastic process we are calling “chance”.

When looking at data from the ocean, it is very rare that we have a complete model in which we understand where the random variability is coming from.

The best that can be done is to use the measurements themselves to build such a model, and then use that model.

In practise, propagation of uncertainties using covariance estimates can prove very difficult. In consequence, it is often best to estimate a power spectrum of the errors based on the measured power spectrum, and use that to generate a series of random time series with similar characteristics to the observations.

Whatever relationship we have found in the observations, we can test how often an equally strong relationship occurs among similar randomly-generated variables. This leads to Monte Carlo error analysis

Monte Carlo methods

Generate (e.g.) 1000 random datasets with similar spectral characteristics to your observations.

For each of these 1000 datasets, calculate the same kind of correlation or fit as the one you have been looking for in the observations.

Sort the correlations/coefficients into ascending order.

Compare the relationship found in the observations with this distribution, to assess how likely it is to happen by chance.

If you want to be able to say that the correlation has a 0.1% probability of happening by chance, then you'll need more than 1000 random datasets (as only one value is in the top 0.1%, leaving a lot of uncertainty in where that division occurs). 10,000 datasets would give you 10 values in the top 0.1%, enough to define it quite well.

Monte Carlo methods

You also have to think about how targeted your calculation was.

If you were looking for a positive correlation at lag between 2 and 3 days (because that is what you expect from your model or understanding of the science), then you only need to consider the top part of the correlation distribution, but also to consider how many different lags you considered in that range, and how many of them produced a strong correlation.

If you were trawling for any strong (positive or negative) correlations among 20 different variables, then you have 20x19/2=190 pairs of variables, so finding one with only a 5% probability (i.e. 1/20) of happening by chance shouldn't be a surprise (you would typically expect to find 190/20=9 or 10 at this level, and should be very surprised if you find none).

There is a place for this kind of speculative analysis, but it comes at the "forming a hypothesis" stage of analysis. You then need to find a way to test that hypothesis which doesn't depend on the same aspect of the data as that which gave you a hint about the hypothesis.

Monte Carlo methods

We tend to be very bad at honest representation of how we formed and tested our hypotheses, especially when writing papers where “telling a clear story” often trumps a strictly honest (which tends to make it boring, excessively detailed and get it rejected) presentation. There’s a degree of trust in a paper, that the presentation is representative of the true level of confidence.

Ultimately, we need to be aware that even the best assessment of how robust a result is, can still be wrong. Even without cheating or self-deception, a paper can be misleading or just plain wrong, even if everything was done correctly.

Judgement of the value of a paper ultimately relies on an assessment of how well it fits with external tests – independent replication is the only true test (and even that is not proof, just support).

Monte Carlo methods

```
%montecarlo1
x=randn(1000000,1);
y=randn(1000000,1);

winwid=10000;
dt=1;
nperfreq=1;
pconf=95;
[csq,xamp,xphs,f,nseg,cconf]=cwhxspec(x,y,dt,winwid,nperfreq,pconf);

n=size(csq,1);
p=(-0.5+[1:n])/n;
edof=nseg*0.69*4./3;
ccdftheory=1.-(1-p).^(1/(edof-1));
ccdf=sort(csq)

figure
p1=plot(ccdf,p'Color',[0.5,0.5,1]);
set(p1,'LineWidth',3)
hold on
p2=plot(ccdftheory,p'r');
set(p2,'LineWidth',1)
label('cumulative probability')
xlabel('squared coherency')
```

Generate 2 white noise time series

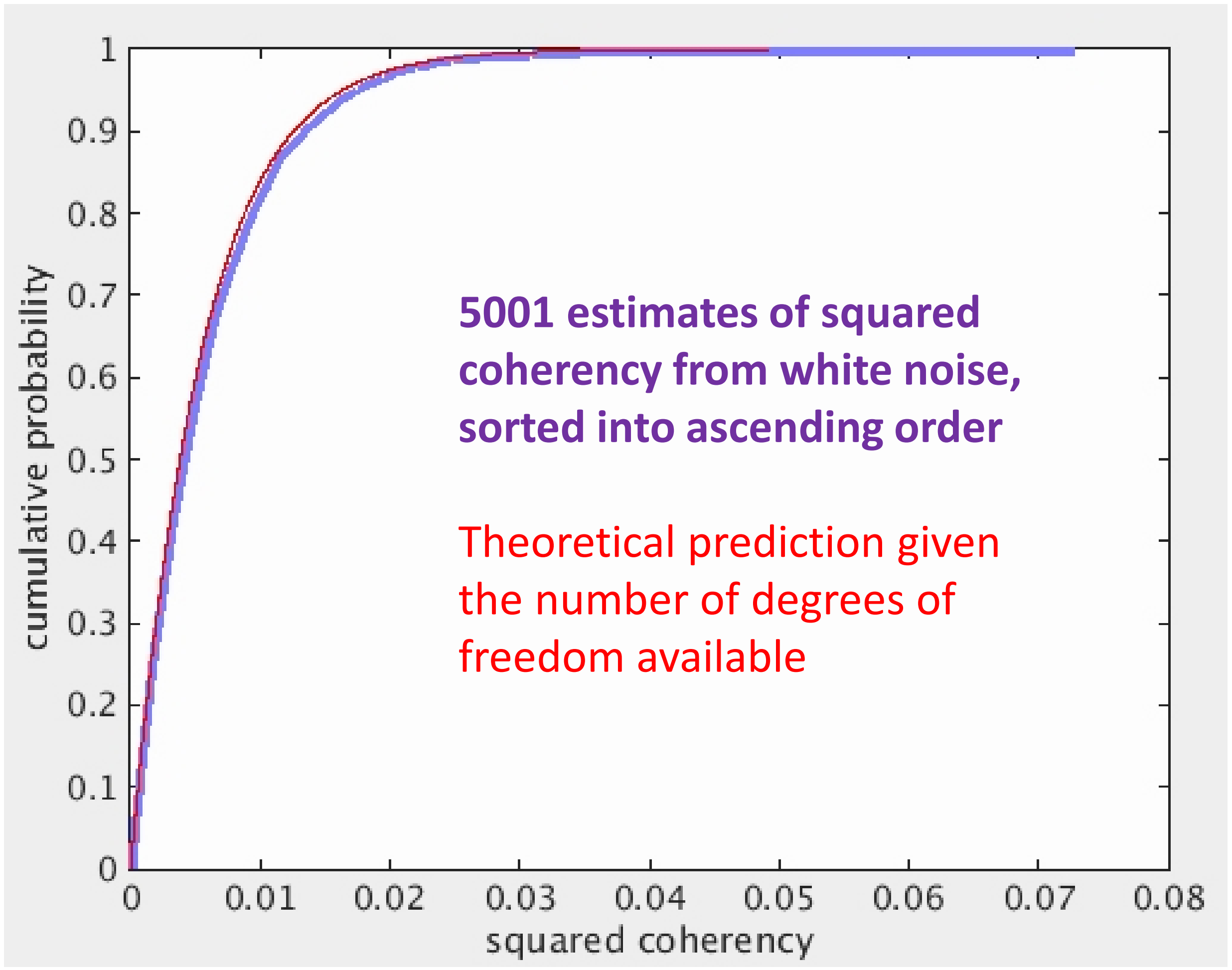
Calculate squared coherency at 5001 frequencies

Sort the n=5001 values into increasing order, assign a cumulative probability

Calculate the theoretical distribution for white noise

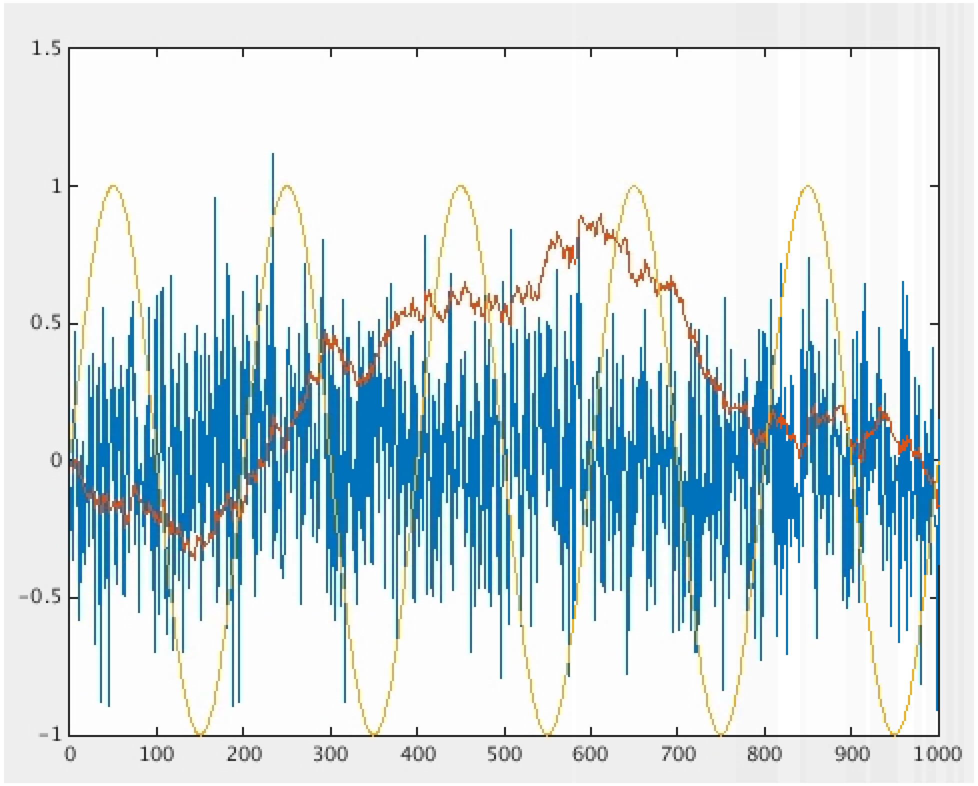
Plot the Monte Carlo distribution, and the theoretical distribution on top

Monte Carlo methods



since there are 5001 values, they represent cumulative probabilities of (0.5,1.5,2.5… 5000.5)/5001.

Example with correlated errors.



A sine wave

plus noise, either

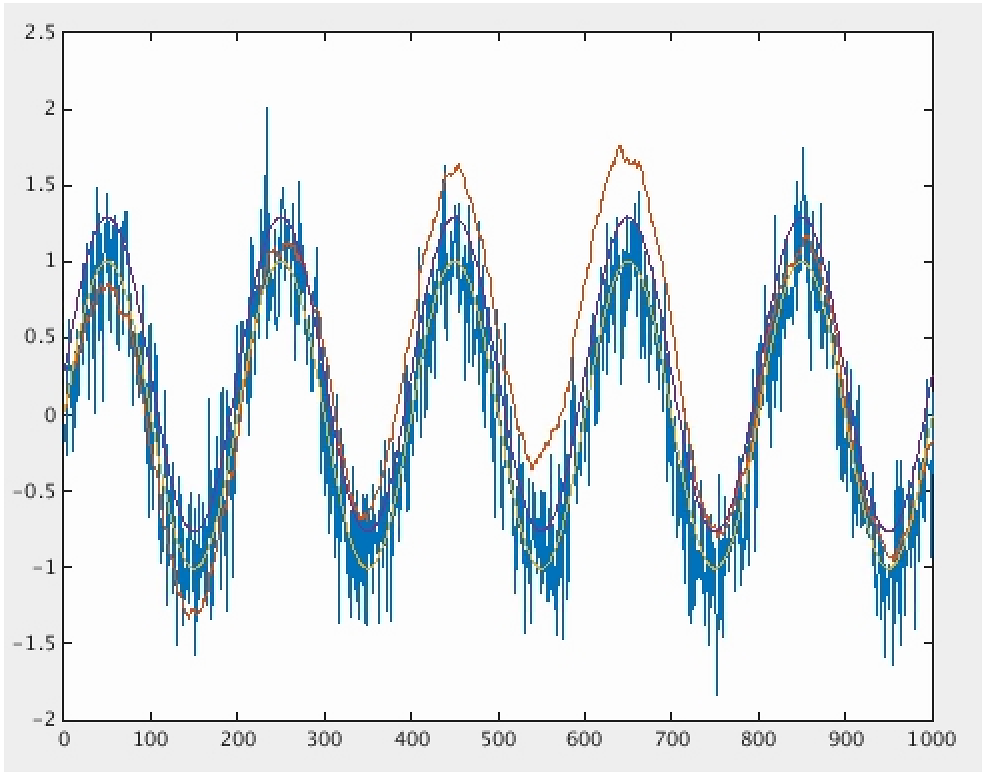
Uncorrelated noise (white noise)

or

Correlated noise (red noise in this case)

Standard deviation of noise is 0.3 times amplitude of sine wave, in both cases

Example with correlated errors.



Sine plus uncorrelated noise (white noise)

Fitted sine wave

Sine plus correlated noise (red noise in this case)

Fitted sine wave

Example with correlated errors.

```
xTrue =

    0.0000
    1.0000
```

```
xWhite =

    0.0031
    1.0099


stxWhite =

    0.0106    (cf 0.0031)
    0.0151    (cf 0.0099)
```

actual error within 1 estimated standard error

```
xRed =

    0.3694
    1.0430


stxRed =

    0.0105     (cf 0.3694)
    0.0149     (cf 0.0430)
```

actual error much larger than estimated standard error (35 and and 2.9 times)

If the error distribution was correct, the chance of being at 2.9 standard deviations would be 0.4%, and for 35 standard deviations it's so small Matlab can't calculate it.

Errors in a fit with red noise

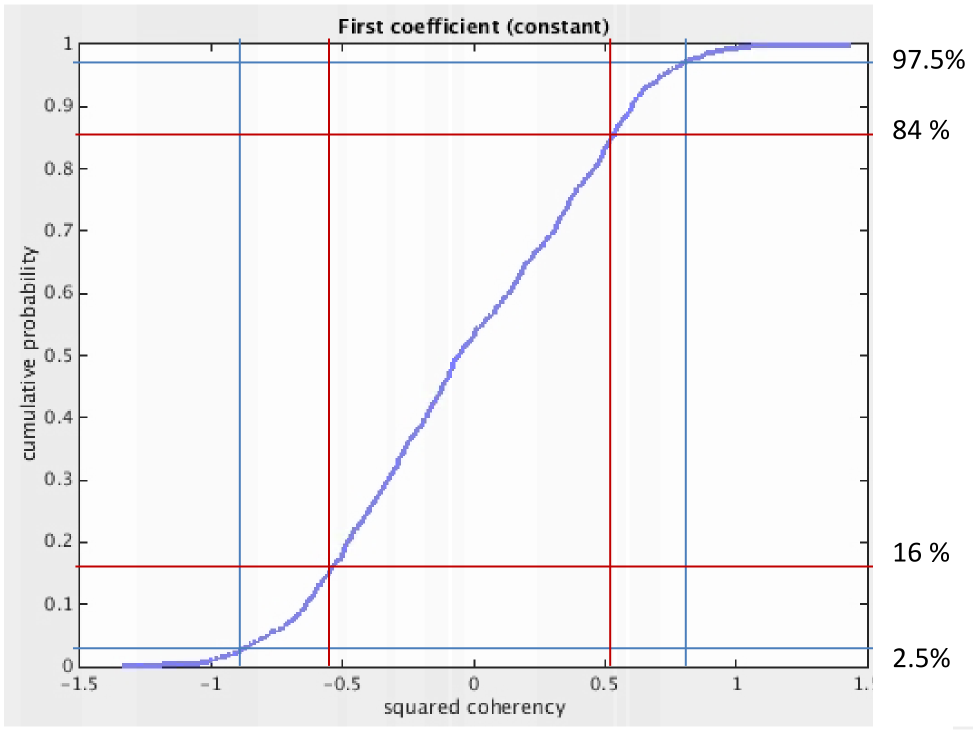
```
facnoise=3.;
ysin=sin(2.*pi*(1:1000)/200.)';
mm=ones([1000 1]);
ss=ysin-mean(ysin);
A=[mm,ss];
xccr=zeros(1000,2);
xccw=zeros(1000,2);
for i=1:1000;
  ynwhite=randn([1000,1]);
  ynred=cumsum(ynwhite);
  ynred=ynred./std(ynred);
  ynred=ynred./facnoise;
  xcoef=A\ynred;
  xccr(i,:)=xcoef;
end
p=((1:1000)-0.5)/1000;
c1dr=sort(xccr(:,1));
c2dr=sort(xccr(:,2));
```

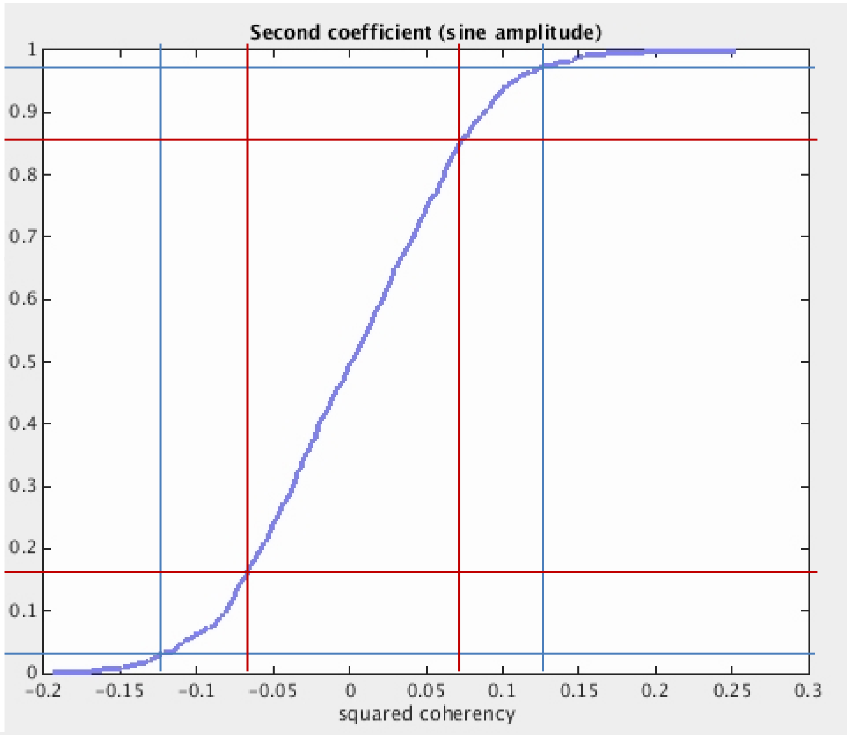
Set up the parameters for the noise, and the functions to fit (constant plus sine function)

Perform 1000 fits on pure red noise time series (generated using the same parameters as in the sinfit demonstration)

Assign a cumulative probability, and sort the coefficients into ascending order.

Errors in a fit with red noise





Mean: -0.0576

95% between -0.8930 and 0.8215

68% between -0.5412 and 0.5100

cf 0.3694 (white noise estimate 0.0031)

Mean: 0.0011

95% between -0.1263 and 0.1265

68% between -0.0674 and 0.0693

cf 0.0430 (white noise estimate 0.0099)

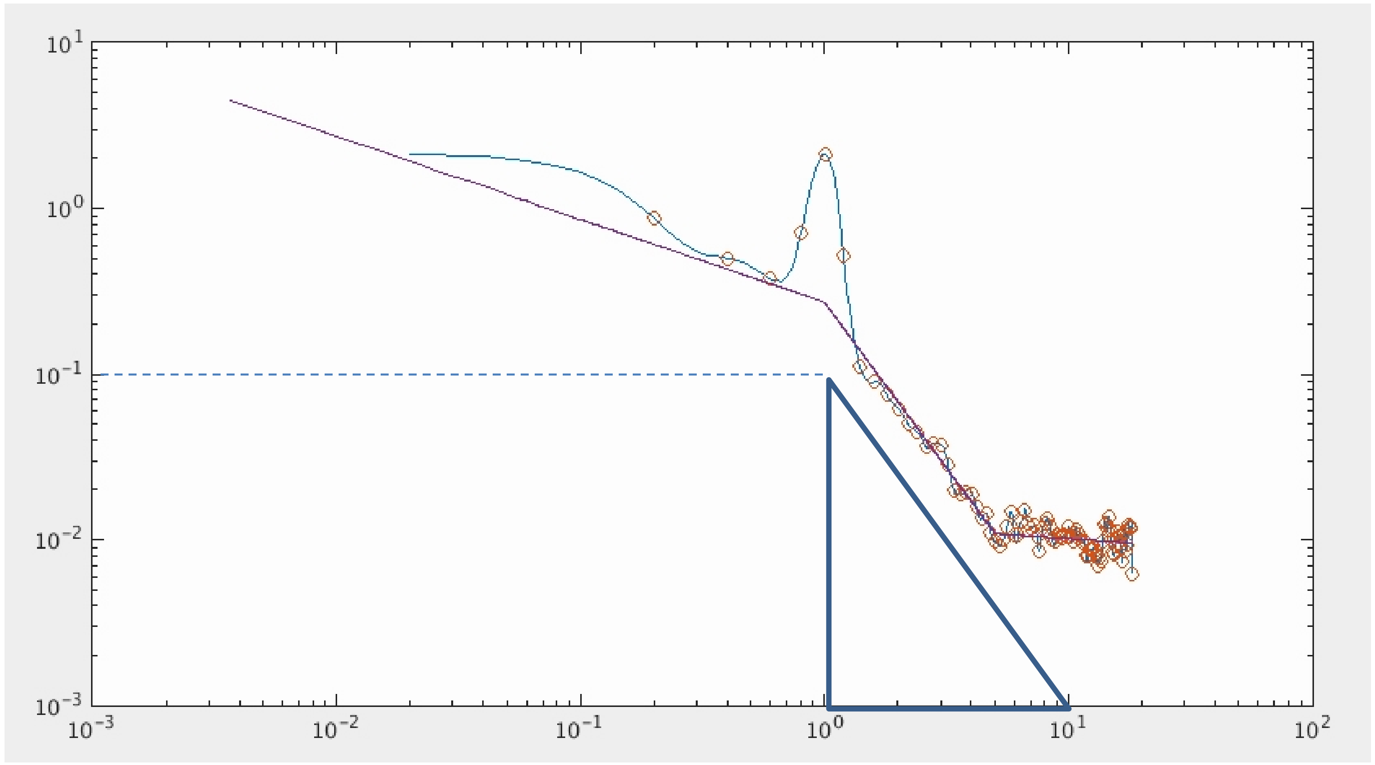
Monte Carlo recipe.

- Determine clearly what, for your current purposes, is meant by “signal” and what is meant by “noise”.
- Make a model which represents the noise as well as possible (including its spectrum).
- Generate a large number (e.g. 1000) of pure noise time series with that spectrum, of the same length as your dataset.
- Process those 1000 time series in exactly the same way you processed your data, to obtain 1000 realizations of the parameter(s) you want (e.g. correlations, coefficients of fitted parameters).
- Sort those 1000 “pure noise” realizations into ascending order, and compare your actual result with them to see how high in the list it occurs. This gives you the probability of your result happening by chance.

Monte Carlo recipe.

- So, how do we make that model?
- We know how to generate power law noise: start with white noise, multiply by $f^{(power/2)}$, and perform an FFT (then renormalize to get the right standard deviation).
- The same works for more complicated spectra, though it can be difficult to program.
- There are many other ways of generating models (autoregressive models are very popular), but direct generation from a model spectrum is relatively straightforward.

Power spectrum from a time series (nperfreq=10 and 1 shown)
Model based on multiple power law noise (assuming the peak is an annual cycle, so not part of the random noise)



cycles per year

Model generated by choosing break points, and joining them with straight lines. Slope of the line (in log space) is the power law. e.g. the middle section descends at 2 factors of 10 in power for every 1 in frequency, so the power law for that section is f $^{-2}$.

```
function ampspec=makespec(f,powers,flims,nsmooth)
%
% Inputs:
%
%   f      is a list of frequencies (as provided by fouper, for example)
%   powers is a list (length n) of power laws to apply in different sections of frequency range
%   flims  is list of frequencies (length is n-1 ) representing the end
%          of the range of validity of each power law (must be in increasing order)
%  nsmooth is the number of frequency bands to smooth over
%          (if even, will use smooth+1, nsmooth=1 means no smoothing)
%
% noise combined so that is changes smoothly between power law ranges
%
% Outputs:
%
%  ampspec is f raised to HALF the chosen set of power laws (i.e.
%             ampspec.^2 gives the chosen power spectrum). This means
%             real(fft(whitenoise.*ampspec)) is a time series with the
%             chosen power spectrum
%
% f = 0 is treated as a special case to avoid divide by zero
%       (corresponding ftopow is set to zero).
%
```

```
function ampspec=makespec(f,powers,flims,nsmooth)
ftopow=zeros(size(f));
npow=size(powers,1);
fmid=log(flims);
pcoef=zeros(size(powers));
pcoef(1)=1;
for i=2:npow;
    pcoef(i)=pcoef(i-1) + (powers(i-1)-powers(i)).*fmid(i-1);
end
ff=f;
ff(ff == 0)=min(min(ff(ff ~= 0)));
ff=log(ff);
fpows=repmat(ff,1,npow);
for i=1:npow;
    fpows(:,i)=pcoef(i)+ powers(i).*fpows(:,i);
end
fsharp=fpows(:,1);
for i=2:npow;
    f1=fpows(:,i);
    flist=find(ff >= fmid(i-1));
    fsharp(flist)=f1(flist);
end
nsmo=ceil((nsmooth-1)/2);
ampspec=zeros(size(fsharp));
fwt=zeros(size(fsharp));
df=f(3)-f(2);
fp=f+nsmo*df;
%fp=f;
for i=-nsmo:nsmo;
    ampspec=ampspec+circshift(fsharp.*fp,i);
    fwt=fwt+circshift(fp,i);
end
ampspec=ampspec./fwt;
ampspec(1:nsmo+1)=fsharp(1:nsmo+1);
ampspec(end-nsmo+1:end)=fsharp(end-nsmo+1:end);
ampspec=exp(0.5*ampspec);

end
```

So, for example, you have a time series of length nt=10000, with points every 10 days, standard deviation xsd, and power laws
-0.5 at frequencies below 1 cycle per year
-2 at frequencies between 1 and 5 cycles per year
-0.1 at higher frequencies

Because real(fft(random data)) is symmetric, we want to generate data at least twice as long as the original dataset (we'll use three times), and use only a subsection of it of length nt.

First calculate f in cycles per year (uses fouper from previous lectures):

```
t=(10./365.25)*[1:nt*3];
dt=t(2)-t(1);
[f,per]=fouper(t,dt);
```

Then set the powers and breakpoints (nsmooth rounds the corners slightly, over nsmooth different frequencies – can be set to 1 for pure power laws):

```
powers=[-0.5;-2;-0.1];
flims=[1;5];
nsmooth=15;
```

Then calculate the corresponding amplitude spectrum (power spectrum is this squared);

```
ampspec=makespec(f,powers,flims,nsmooth);
```

Now generate a thousand time series with the chosen power law:

```
ts1000=zeros(nt,1000);
for i=1:1000;
   tsdata=real(fft(randn(nt*3,1).*ampspec));
   tsdata=tsdata(2:2+nt-1);
   tsdata=tsdata-sum(tsdata)./nt;
   ts1000(:,i)=tsdata.*xsd./std(tsdata,1);
end;
```