ENVS422: Data Analysis of Environmental Records Week 2

Least squares fitting, matrix methods basis functions, errors.

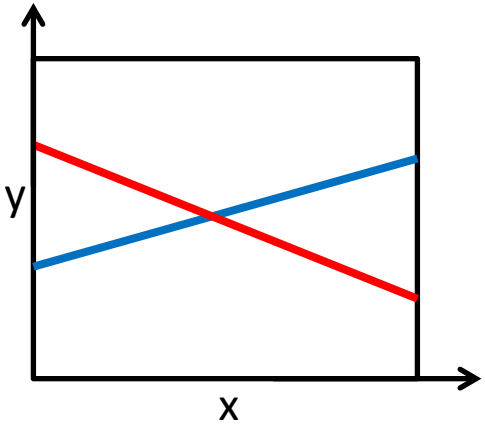
Curve fitting

- You know your data has a particular feature (annual cycle, say). You want to fit that feature and a) estimate its amplitude (and phase), b) remove it so you can focus on the residual.
- You hope that a particular pattern is visible in your data. You want to know how big the amplitude of that pattern is, and whether it is really there, or whether that amplitude occurred by chance.
- You may be interested in multiple patterns.
- Your pattern may occur with variable shape depending on some parameter (e.g. you want to fit an exponential decay, but you don't know the decay constant in advance, so choosing the constant is part of the fit). This involves nonlinear fitting. We will focus here on linear fitting.

Solving simultaneous equations

$$a_1x + b_1y = c_1$$
$$a_2x + b_2y = c_2$$

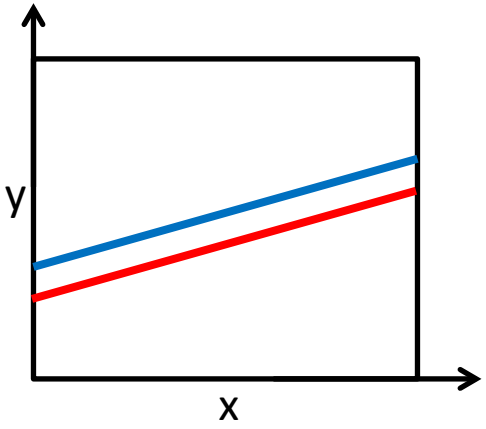
$$\begin{pmatrix} a_1 & b_1 \\ a_2 & b_2 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} c_1 \\ c_2 \end{pmatrix}$$



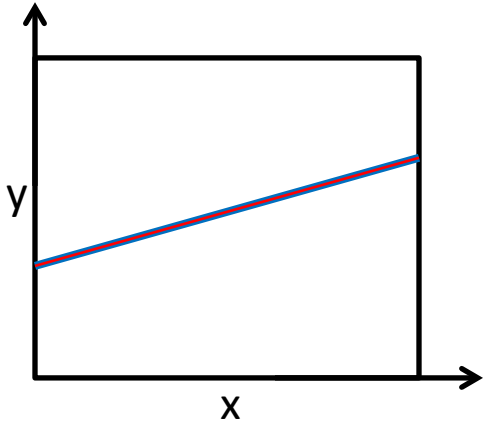
Generic case:
1 solution

$$\mathbb{A}\boldsymbol{x} = \boldsymbol{c}$$

special cases when LHS equations are proportional (i.e. rows of matrix are proportional)



Special case:
No solutions



Special case:
Infinity of solutions

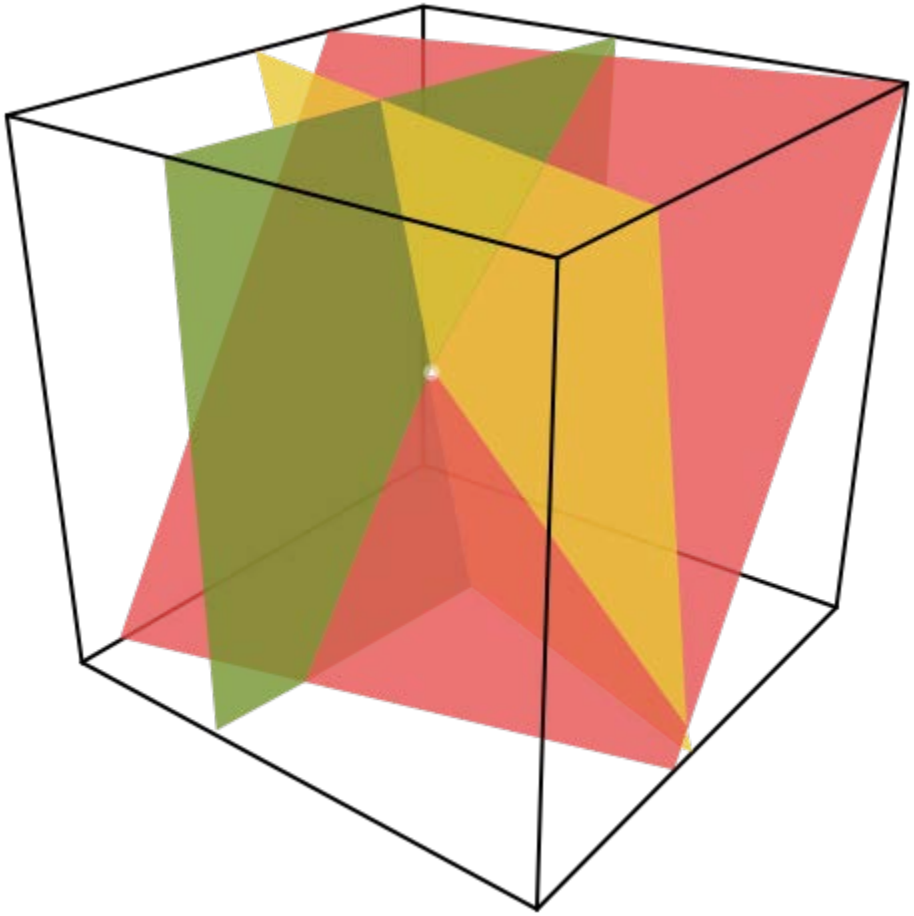
Solving simultaneous equations

$$a_1x + b_1y + c_1z = d_1$$
$$a_2x + b_2y + c_2y = d_2$$
$$a_3x + b_3y + c_3y = d_3$$

$$\begin{pmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ a_3 & b_3 & c_3 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} d_1 \\ d_2 \\ d_3 \end{pmatrix}$$

$$\mathbb{A}\boldsymbol{x} = \boldsymbol{d}$$

Special cases when 1 row of matrix is a linear combination of the others.



Solving simultaneous equations

$$\begin{pmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ a_3 & b_3 & c_3 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} d_1 \\ d_2 \\ d_3 \end{pmatrix}$$

$$\mathbb{A}\boldsymbol{x} = \boldsymbol{d}$$

Matlab is built around matrix algebra. You can solve this system easily by typing

x=A\d

where \ is shorthand for a kind of matrix division.

Can solve by successive elimination of variables. Or can solve by finding the inverse of the matrix:

$$\mathbb{A}^{-1}$$

defined such that

$$\mathbb{A}^{-1}\mathbb{A} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

so then we can write

$$(\mathbb{A}^{-1}\mathbb{A})\boldsymbol{x} = \boldsymbol{x} = \mathbb{A}^{-1}\boldsymbol{d}$$

which is the matrix equivalent of x=d/A, but with matrices the order matters – we have to “left multiply” by the inverse of A.

Matlab example

Solving Ax = b to find x given A and b

```
>> A=[3,2,-1;2,-2,4;-1,0.5,-1]

A =

    3.0000    2.0000   -1.0000
    2.0000   -2.0000    4.0000
   -1.0000    0.5000   -1.0000

>> b=[1;-2;0]

b =

     1
    -2
     0

>> x=A\b

x =

    1.0000
   -2.0000
   -2.0000
```

Matlab example – finds x correctly

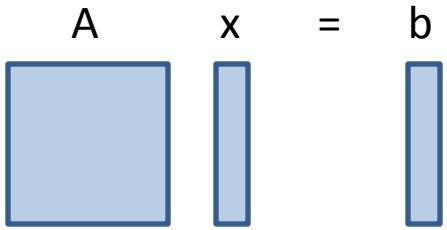
```
>> A*x

ans =

    1.0000
   -2.0000
   -0.0000
```

```
>> A=[3,2,-1;2,-2,4;-1,0.5,-1]

A =

    3.0000    2.0000   -1.0000
    2.0000   -2.0000    4.0000
   -1.0000    0.5000   -1.0000

>> rank(A)

ans =

     3

>> A=[3,2,-1;6,4,-2;-1,0.5,-1]

A =

    3.0000    2.0000   -1.0000
    6.0000    4.0000   -2.0000
   -1.0000    0.5000   -1.0000

>> rank(A)

ans =

     2
```

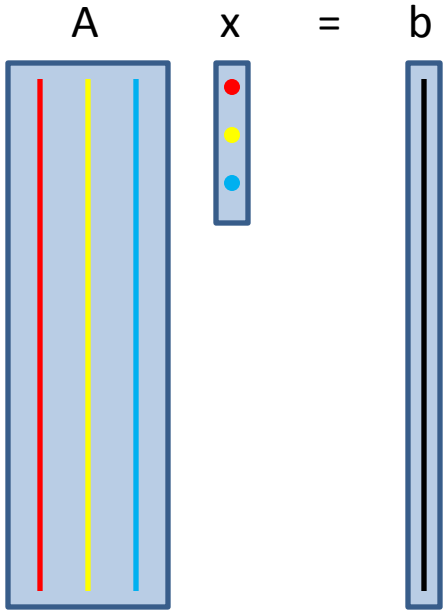
Second version has no solution. If you want to check for special cases, look at the rank of the matrix

```
>> A\b
Warning: Matrix is singular to working precision.

ans =

   NaN
   Inf
   Inf
```

Extend to least squares fitting



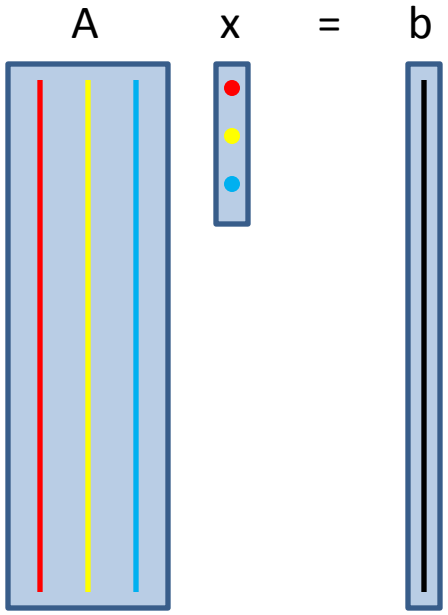
This is what we had before: A is square, x and b have the same number of elements, so we have the same number of parameters to fit (elements in b) as there are numbers to choose in order to obtain the fit (elements in x) – solution may be possible.



What if we have this? Here, we have only a small number of elements in x which we are allowed to choose in order to get a solution, but we are trying to fit a large number of elements in b. Clearly we can't do this exactly, unless the numbers in b happen to be one of the special cases which allows a solution.

But we can choose the elements in x to produce the smallest possible error in the difference between Ax and b. This is a least squares fit!

Extend to least squares fitting



A represents 3 time series $A_1(t)$, $A_2(t)$, $A_3(t)$
x represents 3 coefficients (amplitudes) x_1, x_2, x_3
$Ax = A_1(t).x_1 + A_2(t).x_2 + A_3(t).x_3$

So "solving" Ax = b means finding the amplitudes of the three time series in A which give the best fit to the time series b

Matlab again does the hard work behind the scenes: simply type **x=A\b** and it will give you this solution. **Beware**: x=b\A will also give an answer, but not to this problem. Also x=A/b gives the solution to xA=b, not Ax=b!

Example: red line is a series of ones. Yellow line is $\sin(2\pi t/t_0)$. Blue line is $\cos(2\pi t/t_0)$, where t_0 is a year. Then x would be the best fit of a constant and an annual cycle (sine and cosine components) to the time series b.

Of course it doesn't have to be 3 time series, but we will use 3 here to make things more concrete. You can replace 3 with your number of curves to fit.

What is actually happening?

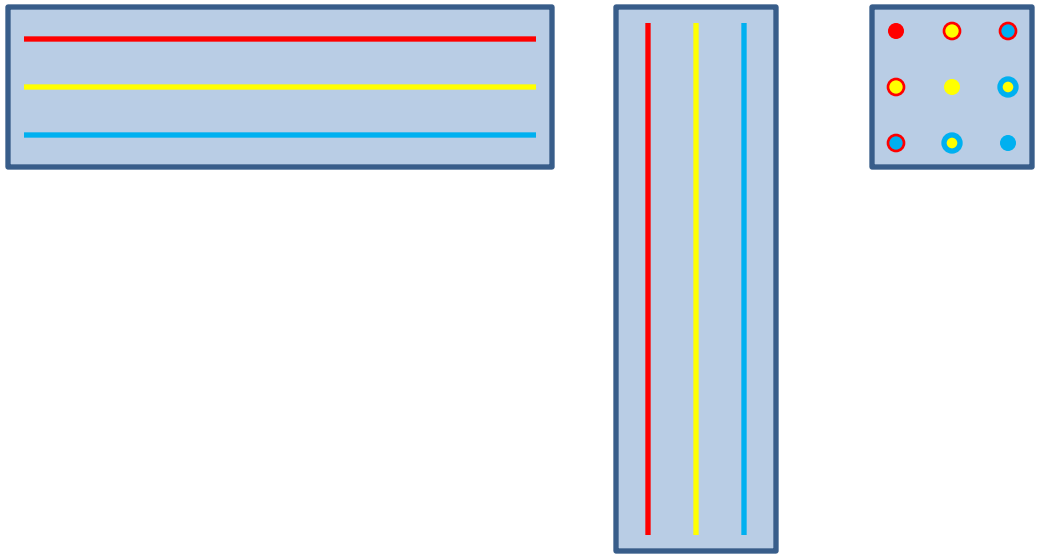
$\mathbb{A}\boldsymbol{x} = \boldsymbol{b}$

is impossible to solve (too many numbers in b to fit with the 3 numbers in x), but if we "left multiply" by the transpose of A:

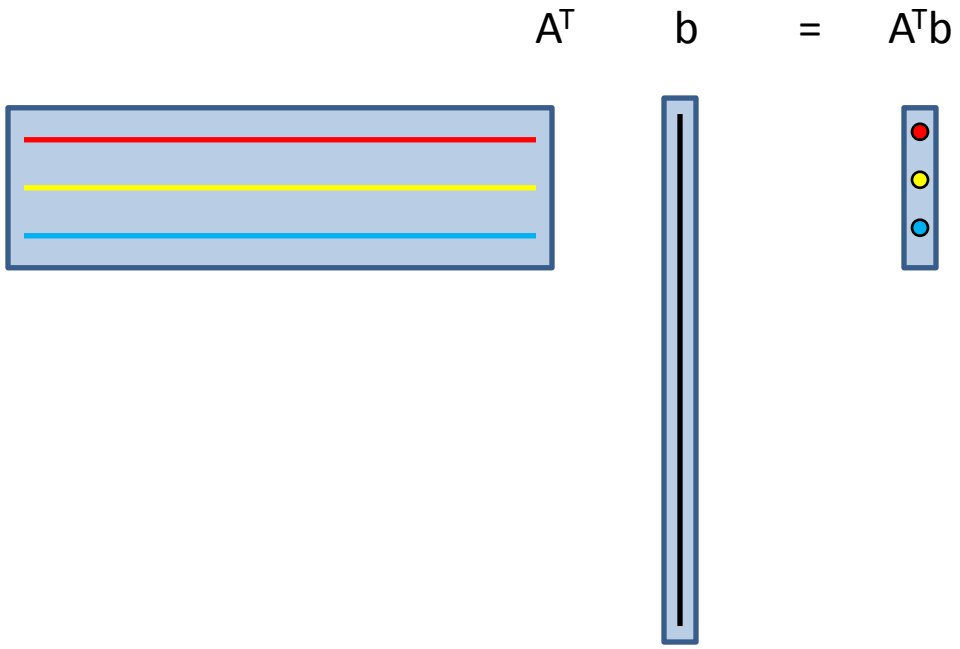
$(\mathbb{A}^T\mathbb{A})\boldsymbol{x} = \mathbb{A}^T\boldsymbol{b}$

is a soluble equation, because the matrix multiplying x is now 3x3, and the RHS is only 3 long

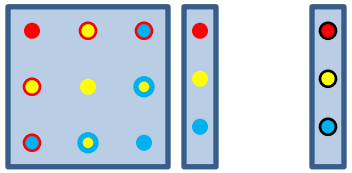
A^T A = (A^TA)



What is actually happening?



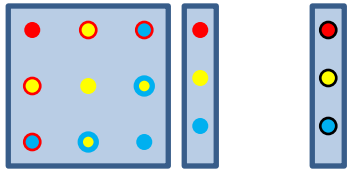
$$(\mathbb{A}^T\mathbb{A})\boldsymbol{x} = \mathbb{A}^T\boldsymbol{b}$$



We are fitting the part of b which “projects” onto the time series in A as well as possible. The bit which does not project onto those time series, we can do nothing about, so this solution minimises the (root mean square) misfit.

What is actually happening?

$$(\mathbb{A}^T\mathbb{A})\boldsymbol{x} = \mathbb{A}^T\boldsymbol{b}$$



(A^TA) is called the covariance matrix. It is symmetrical. The diagonal elements are the variances (squared standard deviations) of the 3 time series.

If the time series are normalized, so the variances are 1, then the off-diagonal elements are correlations between pairs of the three time series.

If the three time series are also uncorrelated, they are said to be "normal" to one another, and the covariance matrix becomes the identity matrix.

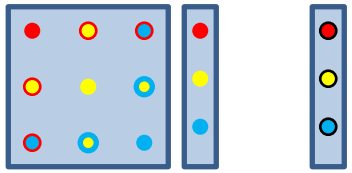
$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

In that case the solution is especially simple: $\boldsymbol{x} = \mathbb{A}^T\boldsymbol{b}$

This is an idea which will come up again, in EOFs and in Fourier analysis.

What is actually happening?

$$(\mathbb{A}^T\mathbb{A})\boldsymbol{x} = \mathbb{A}^T\boldsymbol{b}$$



(A^TA) is called the covariance matrix. It is symmetrical. The diagonal elements are the variances (squared standard deviations) of the 3 time series.

If the time series are normalized, so the variances are 1, then the off-diagonal elements are correlations between pairs of the three time series.

If the three time series are also uncorrelated, they are said to be "orthonormal" to one another, (ortho if uncorrelated, normal if variances are 1) and the covariance matrix becomes the identity matrix. $\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$

In that case the solution is especially simple: $\boldsymbol{x} = \mathbb{A}^T\boldsymbol{b}$

This is an idea which will come up again, in EOFs and in Fourier analysis.

Why “ortho”?

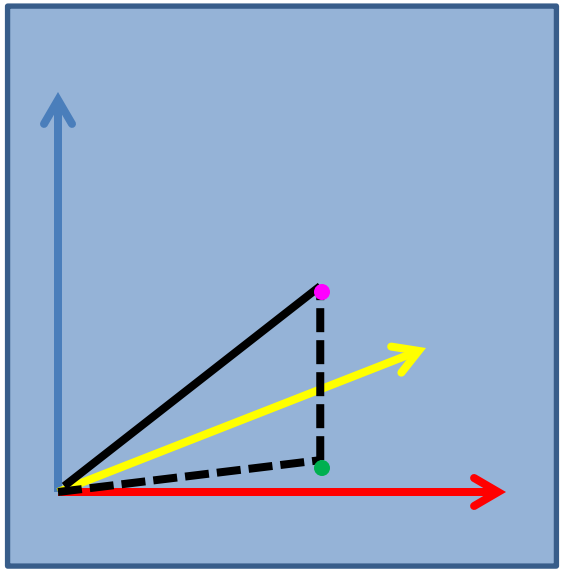
The elements of the covariance matrix are given by multiplying the two time series together, element by element, and summing:

$$A_1(t_1) \times A_2(t_1) \;+\; A_1(t_2) \times A_2(t_2) \;+ \ldots + \; A_1(t_n) \times A_2(t_n)$$

If you had a time series of length 3, you could think of the 3 numbers in A_1 as the components of a 3D vector, and the 3 numbers in A_2 as the components of a 2nd 3D vector. Then the product of the two time series as above is like the dot product of the two vectors.

If the dot product of two vectors is zero, they are perpendicular to each other, i.e. “orthogonal”.

For this reason, time series can be called “vectors”, though typically in much more than 3D!



A way to interpret fitting a 3-point time series with 2 different 3-point time series:

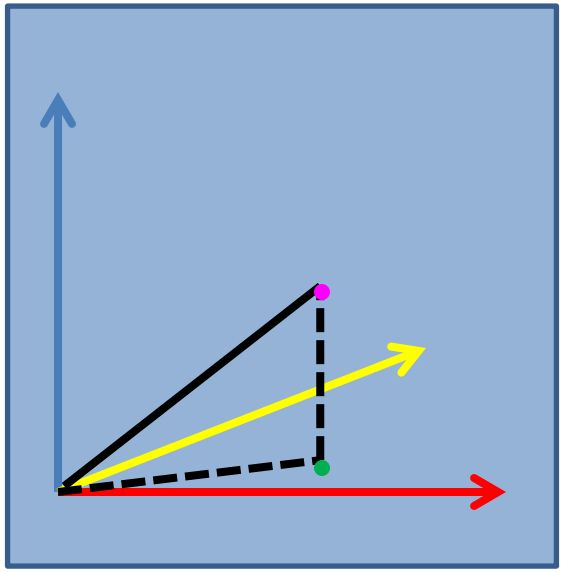
Red and yellow axes represent the 2 different time series (i.e. they are vectors whose components are the 3 numbers in each of those 2 time series.

Blue axis is perpendicular to the red and yellow ones.

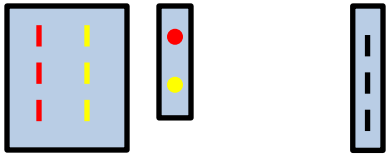
Magenta dot (vector in 3D) is given by the 3 numbers in the time series to fit.

The best fit possible to the magenta dot using only a combination of the red and yellow vectors is the green dot: the projection of the magenta dot into the plane defined by the two vectors.

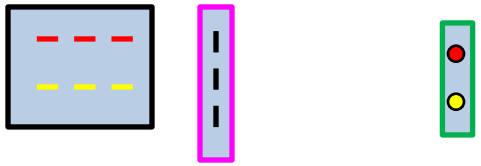
So in least squares fitting, we ignore the blue direction and reduce the problem to finding the combination of red and yellow vectors which gives the green dot.



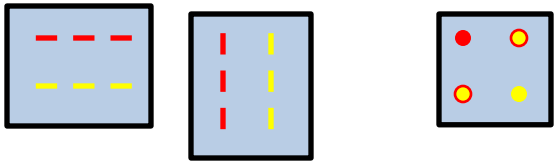
A x = b



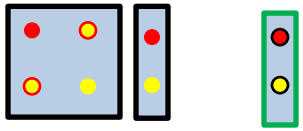
A^T b = A^Tb

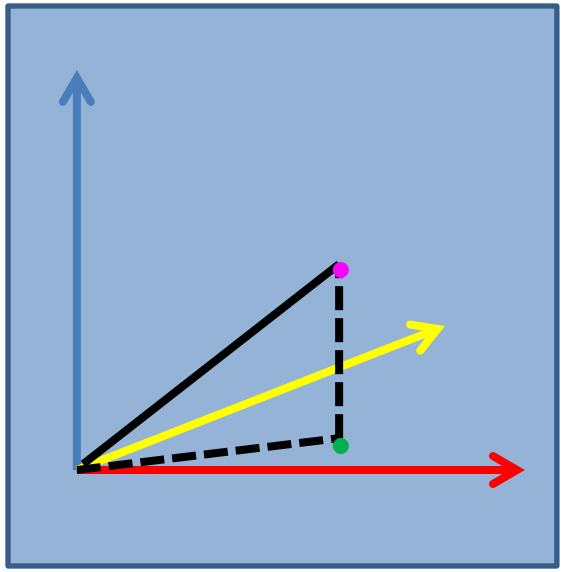


A^T A = (A^TA)



$$(\mathbb{A}^T\mathbb{A})\boldsymbol{x} = \mathbb{A}^T\boldsymbol{b}$$



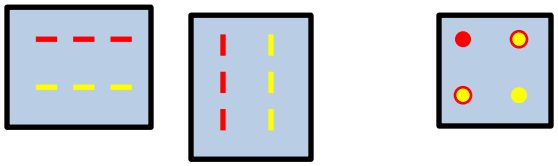


$$A_1(t_1) \times A_1(t_1) + A_1(t_2) \times A_1(t_2) + A_1(t_3) \times A_1(t_3) = A_1 \cdot A_1$$

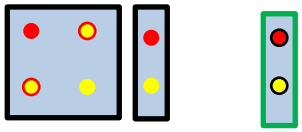
$$A_2(t_1) \times A_2(t_1) + A_2(t_2) \times A_2(t_2) + A_2(t_3) \times A_2(t_3) = A_2 \cdot A_2$$

$$A_1(t_1) \times A_2(t_1) + A_1(t_2) \times A_2(t_2) + A_1(t_3) \times A_2(t_3) = A_1 \cdot A_2$$

A^T A = (A^TA)



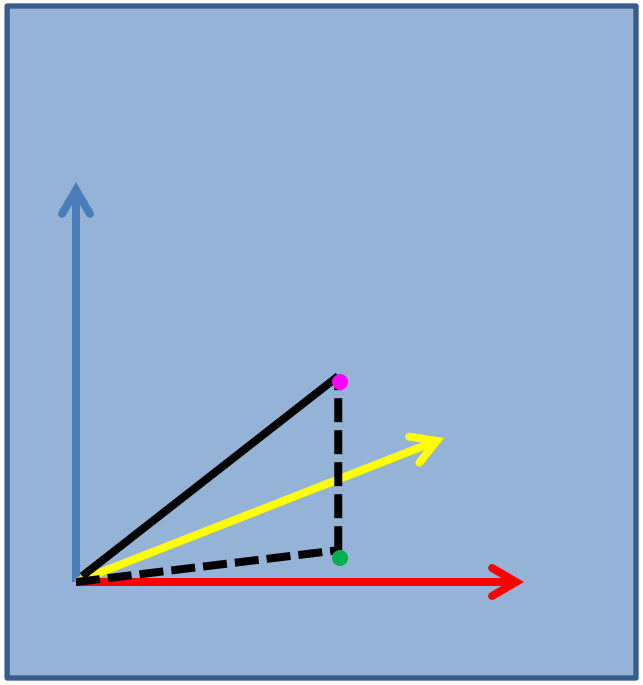
$$(\mathbb{A}^T \mathbb{A})\boldsymbol{x} = \mathbb{A}^T \boldsymbol{b}$$



$$\begin{pmatrix} A_1 \cdot A_1 & A_1 \cdot A_2 \\ A_1 \cdot A_2 & A_2 \cdot A_2 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} A_1 \cdot b \\ A_2 \cdot b \end{pmatrix}$$

If A_1, A_2 are orthonormal:

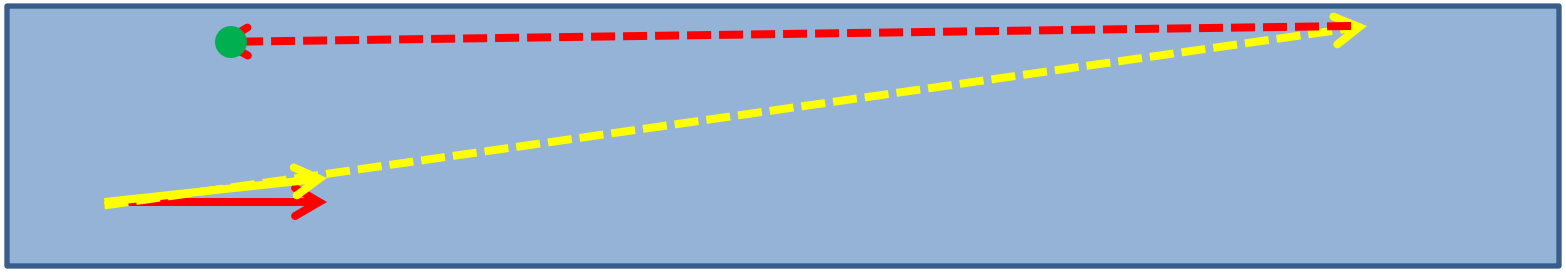
$$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} A_1 \cdot b \\ A_2 \cdot b \end{pmatrix}$$

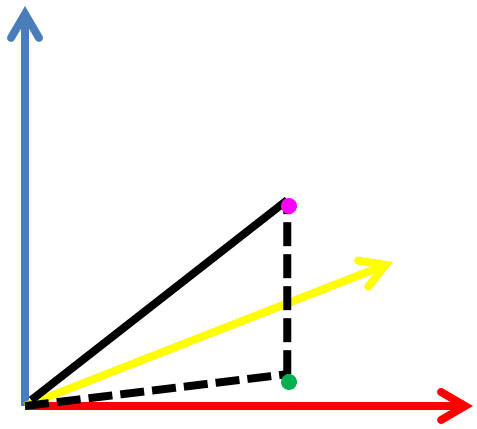


In lucky cases, your time series might be exactly a sum of the curves you are using to fit it (the magenta dot lies in the plane of the red and yellow)

In unlucky cases, the red and yellow axes might point in the same direction (you can still find the best fitting curve, but not how much is due to one curve and how much is due to the other – the individual amplitudes are undefined, but their weighted sum is clearly defined).

Or the red and yellow axes might be nearly parallel. Then you can reach anywhere in the plane defined by the two, but often only by combining very large and nearly cancelling amplitudes. The amplitude estimates become very sensitive to noise. Looking down:





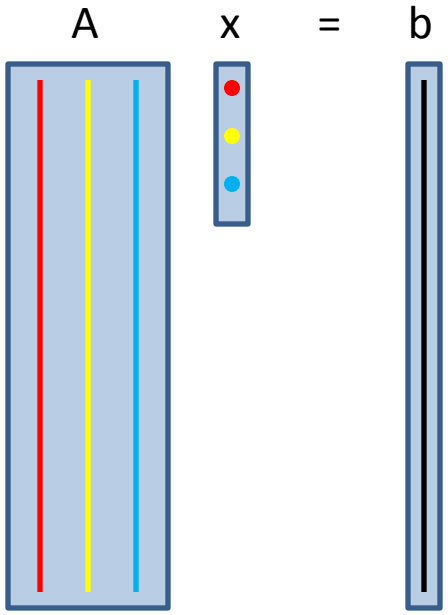
It's only really possible to visualise this for 3-point time series (3D), but the same ideas work for longer time series (higher dimensions). The main difference is that (say) for a 5-point time series fitted by 2 curves, there will be 3 mutually perpendicular directions which are perpendicular to the plane defined by the two (red and yellow) curves (i.e. 3 different blue vectors which are undefined).

The curves you are fitting with are sometimes termed "basis vectors". They define the possible set of fitted curves (the red/yellow plane where the green dot lies).

If there are as many curves as there are points to fit (3 in this case), the basis vectors will usually be capable of reproducing any curve (you will have 3 axes, so you can reach any point in 3D space with them)…. Unless the 3 axes happen to be in the same plane.

Ideally, the 3 axes would be orthonormal – then A^TA becomes the identity matrix … the components of the magenta dot are the answer.

Summary



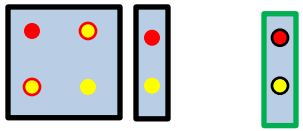
A represents 3 time series $A_1(t)$, $A_2(t)$, $A_3(t)$
x represents 3 coefficients (amplitudes) x_1, x_2, x_3
$Ax = A_1(t).x_1 + A_2(t).x_2 + A_3(t).x_3$

If A is square (time series length = number of time series you are using to fit b), and b is the same length as x, then this can be solved by left multiplying by the inverse of A.

In Matlab you do this simply by typing x=A\b.

If A is not square, there is no exact solution, but typing x=A\b will still give you an answer. This is the solution to

$$(\mathbb{A}^T\mathbb{A})\boldsymbol{x} = \mathbb{A}^T\boldsymbol{b}$$



Which gives values of x such that Ax = β is as close as possible to b in the sense of minimum sum of squares of errors $(\beta - b)\cdot(\beta - b)$ in vector notation, or $(\beta - b)^T(\beta - b)$ (T is transpose – flip around the diagonal – in Matlab b^T is written as b').

An example: how to fit a mean, trend and annual cycle

```
function [x,yfit] = multifit(yin,tin)
%
% fits vector y with constant, trend, sin(2.pi.t), cos(2.pi.t)
% e.g. if t is time in years, fits a mean, trend and annual cycle)
%
ny=numel(yin);
nt=numel(tin);
if nt ~= ny
   fprintf('Vector lengths unequal. Using shorter one')
   ny=min([nt,ny]);
end
y=reshape(yin(1:ny),[1 ny]);
t=reshape(tin(1:ny),[1 ny]);
A=[ones([1 ny]) ; t ; sin(2.*pi*t) ; cos(2.*pi*t)]';
x=A\y';
yfit=A*x;
end
```

All this is just to make sure the two input arrays have the same number of elements ny, and convert to size [1 ny]

Create matrix of functions to fit (entered as rows, then the whole matrix transposed)

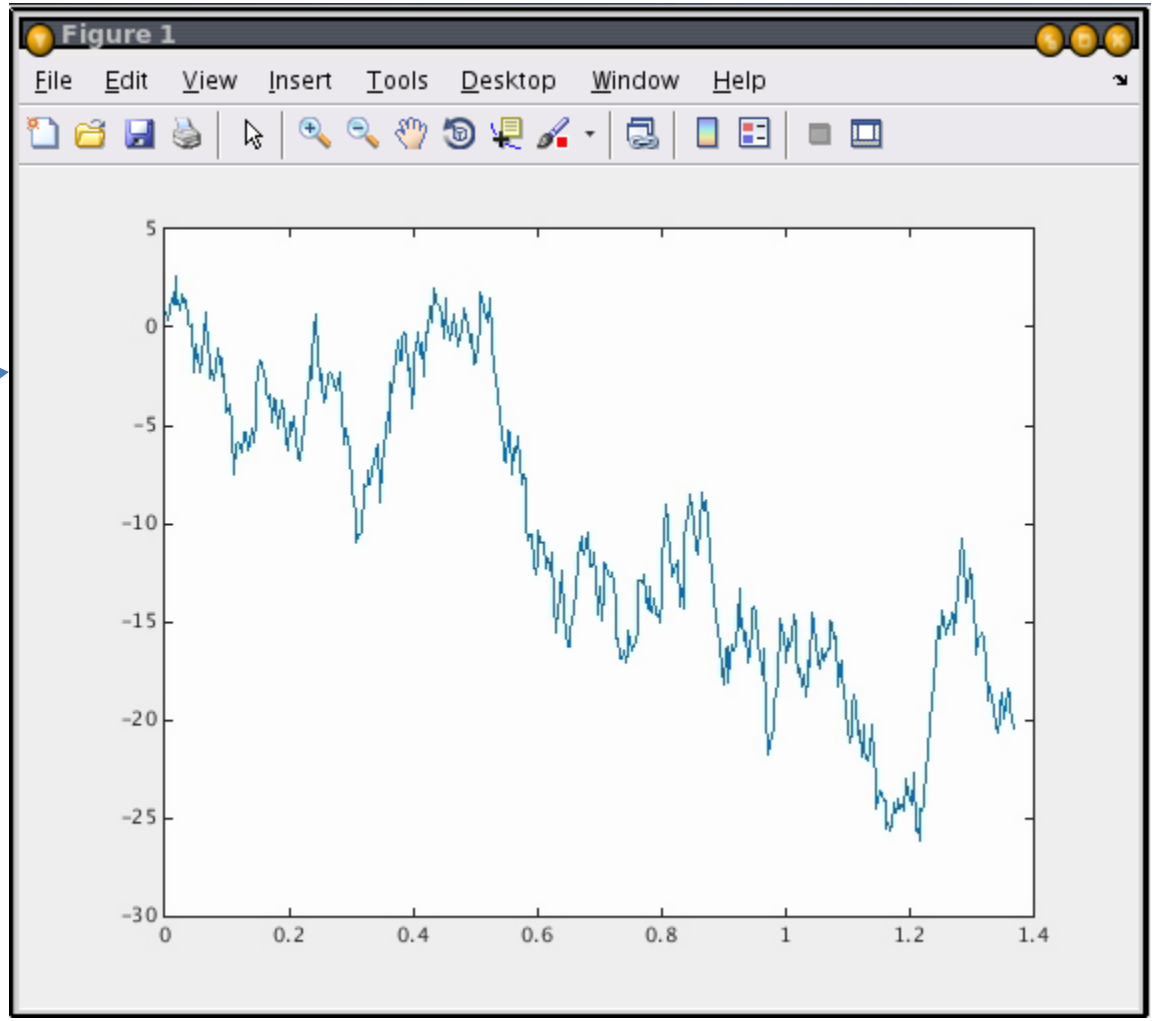
Construct the sum of fitted curves (best fit to y)

Solve for x
Note use of y'
y is a row, but the RHS requires a column

An example: fit to this time series

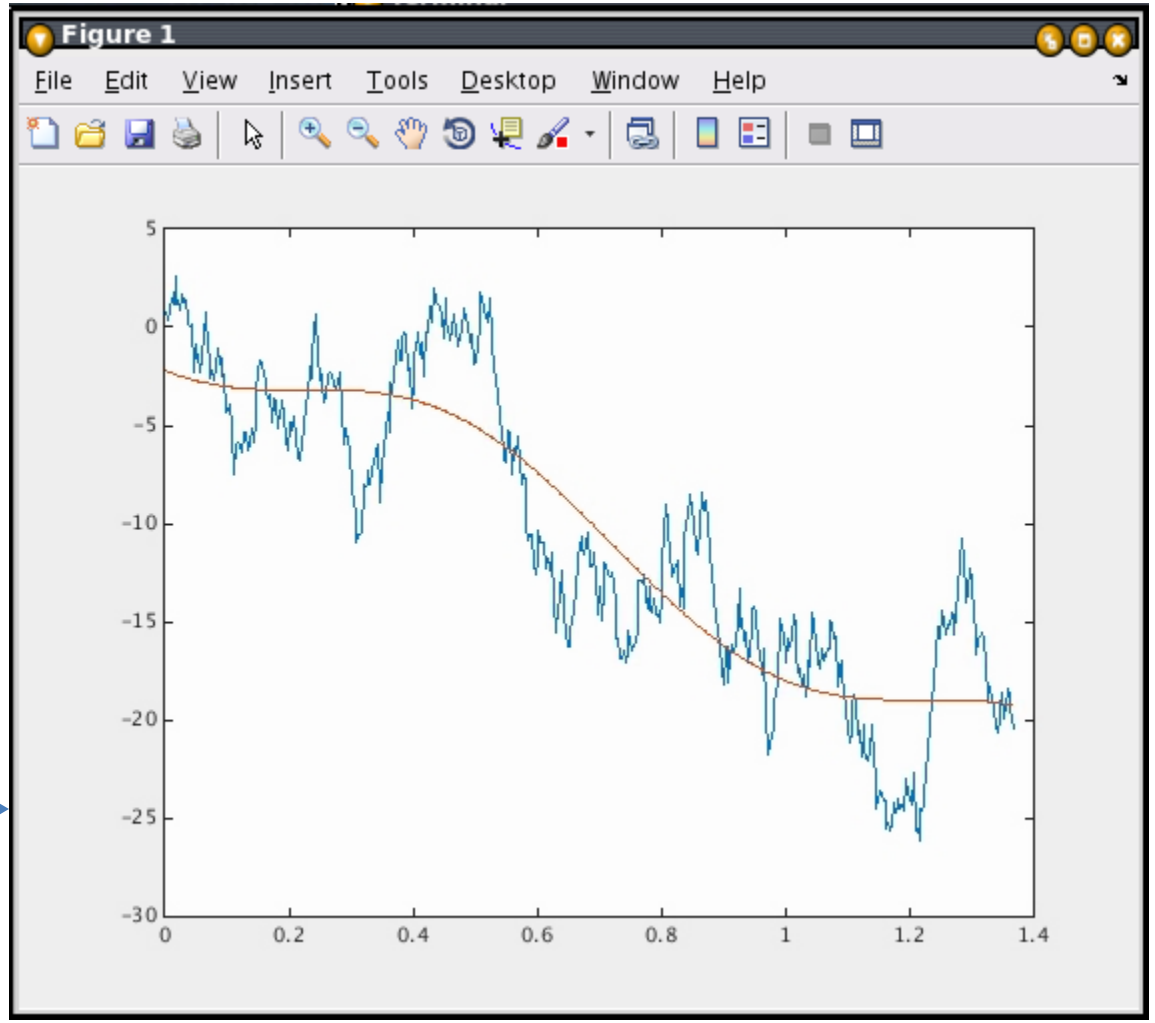
```
>> plot(t,y)
>> [x,yfit]=multifit(y,t);
>> x

x =

    0.3255
  -15.8082
    0.4475
   -2.5157

>> hold on
>> plot(t,yfit)
```

coefficients of the 4 curves fitted

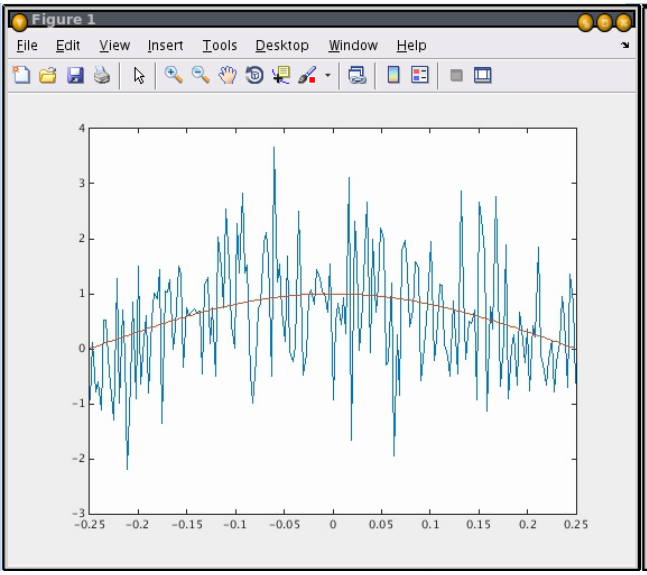


```
>> plot(t,y)
>> [x,yfit]=multifit(y,t);
>> x

x =

    0.3255
  -15.8082
    0.4475
   -2.5157

>> hold on
>> plot(t,yfit)
```

coefficients of the 4 curves fitted



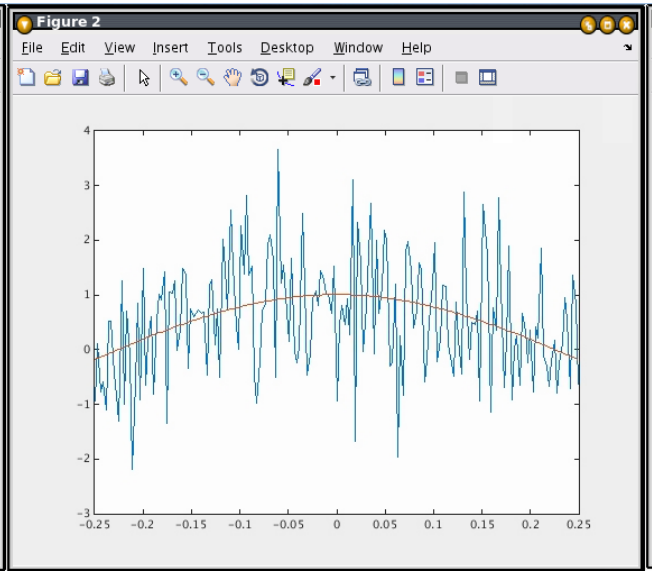
What happens when the curves you fit are too similar (far from orthogonal)



x1 =

0.0000
1.0000

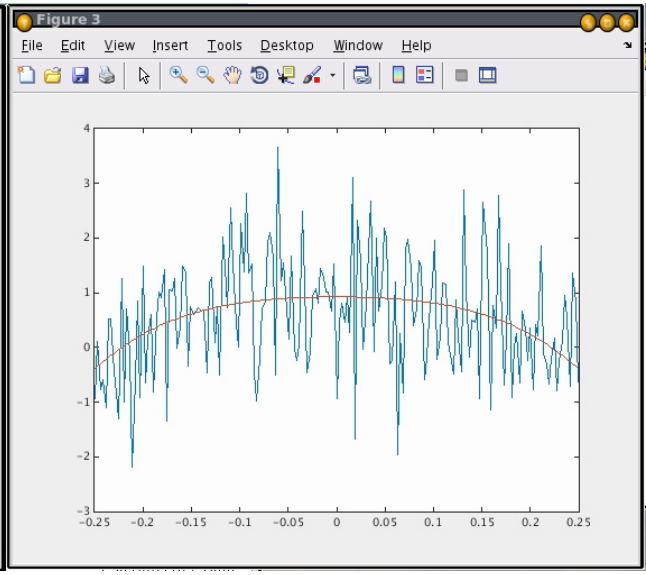
True
curve = cos(2πt) + noise



x1 =

-0.1739
1.1815

Fit
constant +cosine

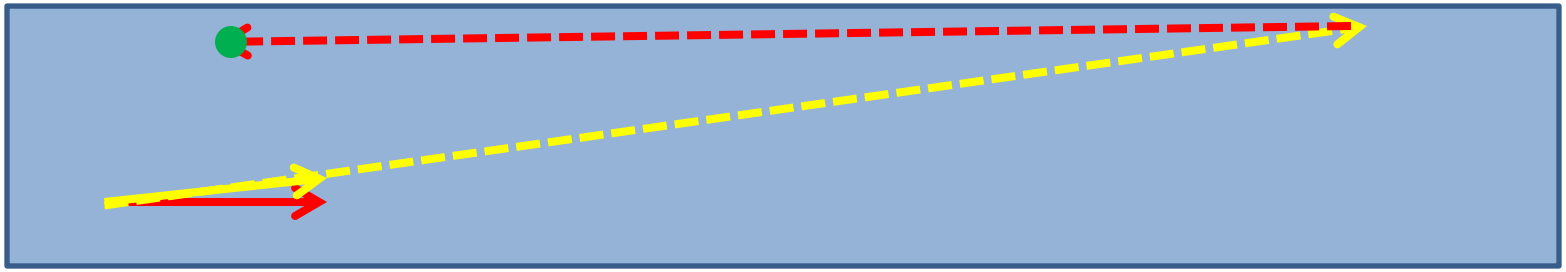


x2 =

4.1180 constant
-3.1932 cosine
-72.2979 t squared

Fit
constant +cosine+t^2
t^2 and cosine look very similar

What happens when the curves you fit are too similar (far from orthogonal)



The combination of cosine and t2 makes it possible to get closer to the data, but only by having large and mostly cancelling coefficients. The fit looks fine, but the estimated amplitudes are horribly wrong. “overfitting”.

x1 =	x1 =	x2 =	
0.0000	-0.1739	4.1180	const
1.0000	1.1815	-3.1932	cosine
		-72.2979	t squared

Fit

constant +cosine+t^2

t^2 and cosine look very similar

Using lscov, obtaining errors

```
>> [x,stx,mse,Sx]=lscov(A,y')

x =

    0.5762
   -3.1932
    4.4939

stx =

    0.0730
    4.1868
    4.2940

mse =

    0.9752

Sx =

    0.0053   -0.0000    0.0000
   -0.0000   17.5289  -17.9492
    0.0000  -17.9492   18.4383
```

Like

xx=A\y’

but also returns

stx=standard error on x

mse= mean squared error (i.e. size of mean square misfit)

Sx is the error covariance on the fitted coefficients.

Using lscov, obtaining errors

stx =

0.0730
4.1868
4.2940

stx=standard error on the fitted coefficients.

Sx =

0.0053	-0.0000	0.0000
-0.0000	17.5289	-17.9492
0.0000	-17.9492	18.4383

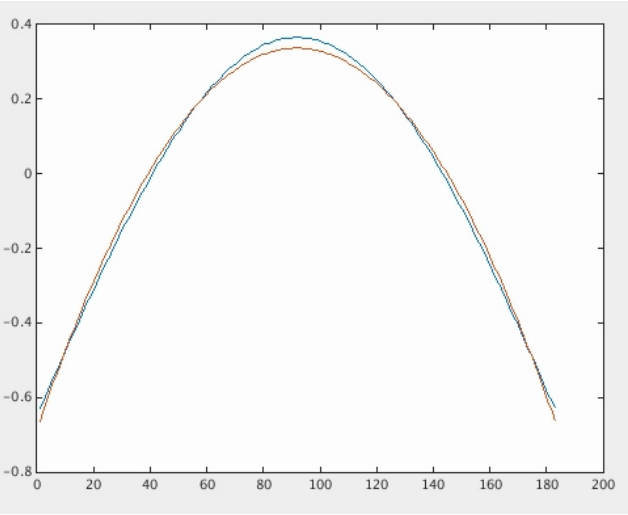
Sx is the error covariance on the fitted coefficients.

The diagonal elements are stx squared.

The off diagonal elements tell us how errors in different coefficients are related. Here, the estimates of coefficients 2 and 3 are large, but they are strongly anticorrelated.

Using lscov, obtaining errors

This is the same case as before – fitting a constant, sine and t^2, except I have removed the mean from the sine and t^2, and renormalized them so they look similar:



The first coefficient is for the mean, and is completely uncorrelated with the other two coefficients (because I subtracted the mean from each of these time series).

The errors in the 2nd and 3rd are strongly anticorrelated because the two curves are so similar. Different combinations of these two curves would fit y almost as well.

Sx =

0.0053	-0.0000	0.0000
-0.0000	17.5289	-17.9492
0.0000	-17.9492	18.4383

Weighted fitting

- Perhaps the noise on one part of the time series is larger than on another part. You then want to allow the fit to be looser over that period (downweight the points).
- Perhaps the data aren't evenly spaced in time – you have data every week, but a patch of data every day. You want the daily values to count only 1/7 as strongly as the weekly ones.
- Perhaps you are working in space, not time. You have data on a regular latitude-longitude grid, and are fitting a spatial function. Points near the pole represent a smaller area than points near the equator. You want to weight them by cos(latitude), or $\cos^2$(latitude) if on a Mercator grid.
- These are all examples of weighted fitting.

Weighted fitting

One way to do it

You have a time series: $y = [y_1 \ y_2 \ y_3 \ \dots \ y_n]$

and weights $w = [w_1 \ w_2 \ w_3 \ \dots \ w_n]$

$W=diag(w)$

Multiply the time series by the weights $yw=(W*y')$

$$\begin{pmatrix} y_1 w_1 \\ y_2 w_2 \\ y_3 w_3 \end{pmatrix} = \begin{pmatrix} w_1 & 0 & 0 \\ 0 & w_2 & 0 \\ 0 & 0 & w_3 \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix}$$

Also multiply the time series you are fitting by the weights: A=W*A

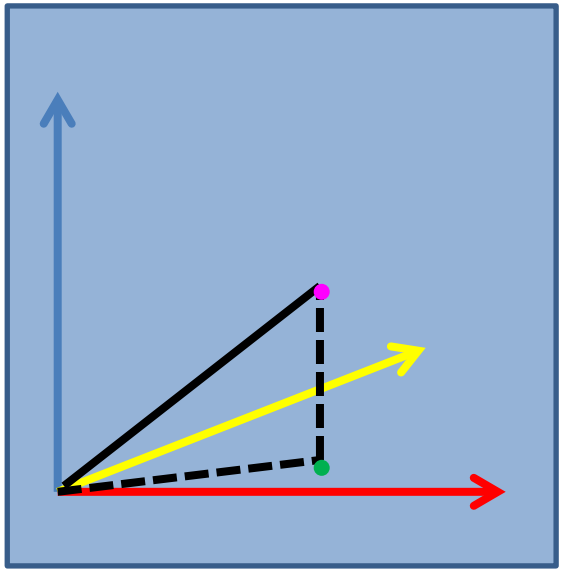
Solve as before: x=A\yw'

Remember when reconstructing the time series to divide by the weights again: yfit=W^(-1)*A*x

Weighted fitting

In our 3D analogy:

If we want to downweight one component of the vector (1 point in the time series), e.g. the red one, we simply shrink the red axis. Distances in that direction get smaller, so the misfit in that direction will be smaller too.



Least squares minimises the square of the misfit, so the linear weighting should be inversely proportional to the standard error (i.e. standard deviation):

$w \propto \frac{1}{\sigma}$ NOTE weight arguments in Matlab are inverse variances $\frac{1}{\sigma^2}$

Or, for uneven spacing, the weighting should be inversely proportional to the amount of time or space associated with each point: $w \propto \frac{1}{\sqrt{\delta t}}$ Matlab: $\frac{1}{\delta t}$

Using lscov with weighting

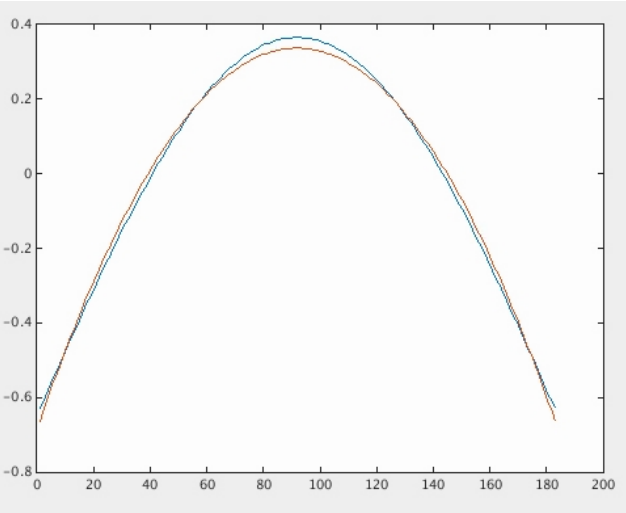
```
>> w=[ones([1  100]),0.5*ones([1  83])];

>> [x,stx,mse,Sx]=lscov(A,y',w)
```

does the same calculation but with the last 83 of 183 points downweighted by 0.25

Result looks very similar, but coefficients are different because of the sensitivity:

old x =	new x =
0.5762	0.5736
-3.1932	-4.9134
4.4939	6.3503

NOTE the values in the w argument are the squares of the "weights" interpreted as a linear scaling.



old Sx =

0.0053	-0.0000	0.0000
-0.0000	17.5289	-17.9492
0.0000	-17.9492	18.4383

new Sx =

0.0053	-0.0097	0.0094
-0.0097	17.2674	-17.7172
0.0094	-17.7172	18.2366

Important!
What if the errors are not independent?

All this machinery for calculating errors assumes that the error (misfit) on each point is independent of the misfit at all other points.

This is very rarely true! And it makes a **big** difference.

If you know how the errors covary with each other, you can account for this by supplying weights which include covariances:

instead of (w_1^2, w_2^2, w_3^2) we supply $\mathrm{V} = \begin{pmatrix} v_{11} & v_{12} & v_{13} \\ v_{21} & v_{22} & v_{23} \\ v_{31} & v_{32} & v_{33} \end{pmatrix}$

Where V is the data error covariance matrix. We call with
>> [x,stx,mse,Sx]=lscov(A,y',V)

Representation of weights

$$\mathrm{V} = \begin{pmatrix} v_{11} & v_{12} & v_{13} \\ v_{21} & v_{22} & v_{23} \\ v_{31} & v_{32} & v_{33} \end{pmatrix}$$

How do we estimate V? Good question. We will cover one way with Fourier methods. Another way is to give up on the formal error analysis and use Monte Carlo methods, which we'll cover at the end.

Note a big difference when V (error covariance matrix) is supplied rather than w (list of squared weights) (w_1^2, w_2^2, w_3^2) The matrix form of this list is

$$\mathrm{W} = \begin{pmatrix} w_1^2 & 0 & 0 \\ 0 & w_2^2 & 0 \\ 0 & 0 & w_3^2 \end{pmatrix}$$

which, if reflecting independent errors, would be

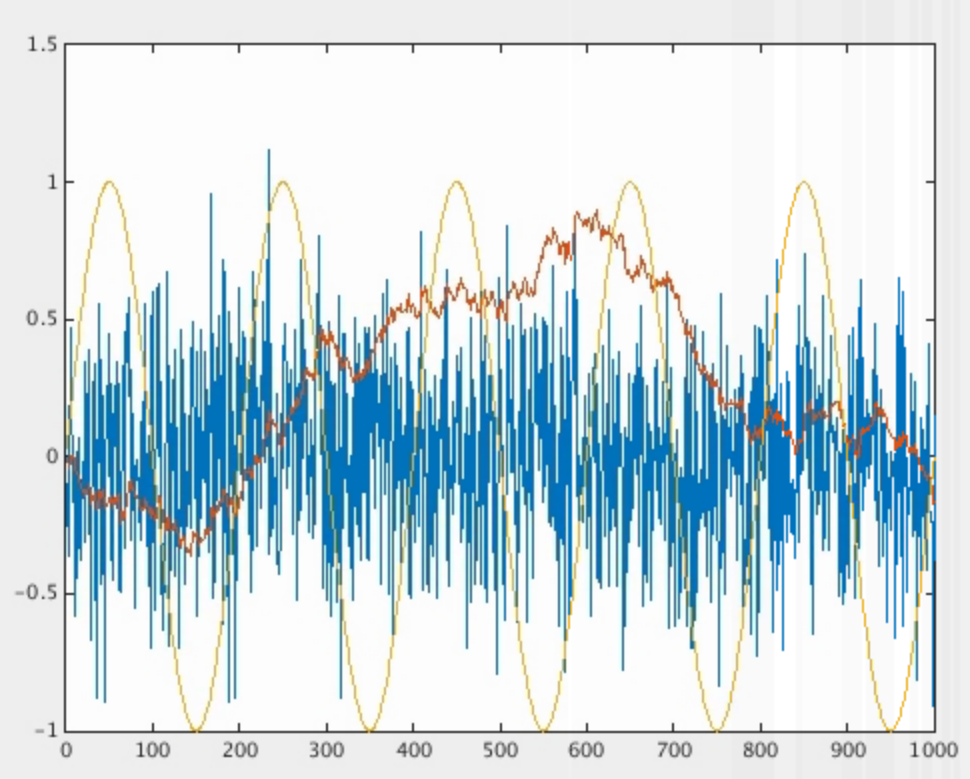
$$\begin{pmatrix} \frac{1}{\sigma_1^2} & 0 & 0 \\ 0 & \frac{1}{\sigma_2^2} & 0 \\ 0 & 0 & \frac{1}{\sigma_3^2} \end{pmatrix}$$

But V represents error covariances. For independent errors, this would be

$$V = \begin{pmatrix} \sigma_1^2 & 0 & 0 \\ 0 & \sigma_2^2 & 0 \\ 0 & 0 & \sigma_3^2 \end{pmatrix}$$

i.e. W is the inverse of V.

Example with correlated errors.



A sine wave

plus noise, either

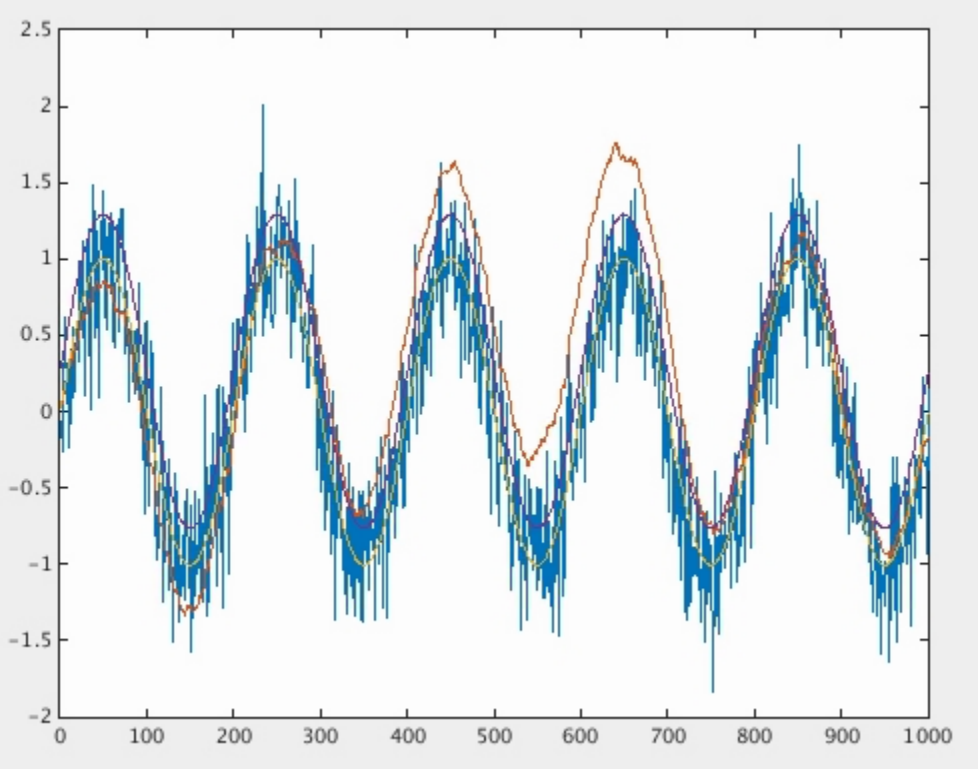
Uncorrelated noise (white noise)

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

stWhite =

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.