PY410 / 505
Computational Physics 1

Salvatore Rappoccio

Code for this lecture

- Code will be found in
  - https://github.com/ubsuny/CompPhys/tree/main/DataAnalysis/Fitting

Some documentation for you for today's class

- Statistics derivation is best described in the Particle Data Group :
  - http://pdg.lbl.gov/2013/reviews/rpp2012-rev-statistics.pdf

- You can also check the Numerical Recipes if you want (although not necessary, strictly speaking):
  - http://apps.nrbook.com/empanel/index.html

First Example : Hubble's Law

- As a first physics application, we will study Hubble's Law, and learn how to perform a least squares fit to Edwin Hubble's measurements on extra-galactic nebulae given in his 1929 article.
- This is a linear fit, which will give us some intuition about fitting in general

Hubble's Law

- Galaxies are collections of hundreds of billions of stars
  - Very enormous!

- Here's a picture from the Hubble telescope released in 2012 showing lots of different galaxies from the Hubble extreme Deep Field
- http://en.wikipedia.org/wiki/Hubble_Extreme_Deep_Field
- This is a photo of an event 13.2 billion years ago, just after the universe underwent inflation!
- And for a comical perspective :

Hubble’s Law

- We’re going to analyze the data from the original 1929 paper
- Local Group Galaxy NGC 6822 (Barnard’s Galaxy)
- r = 0.214 Mpc (1 Mega-parsec = 3.086x10^{19} km) moving towards us with speed v = 130 km/s.

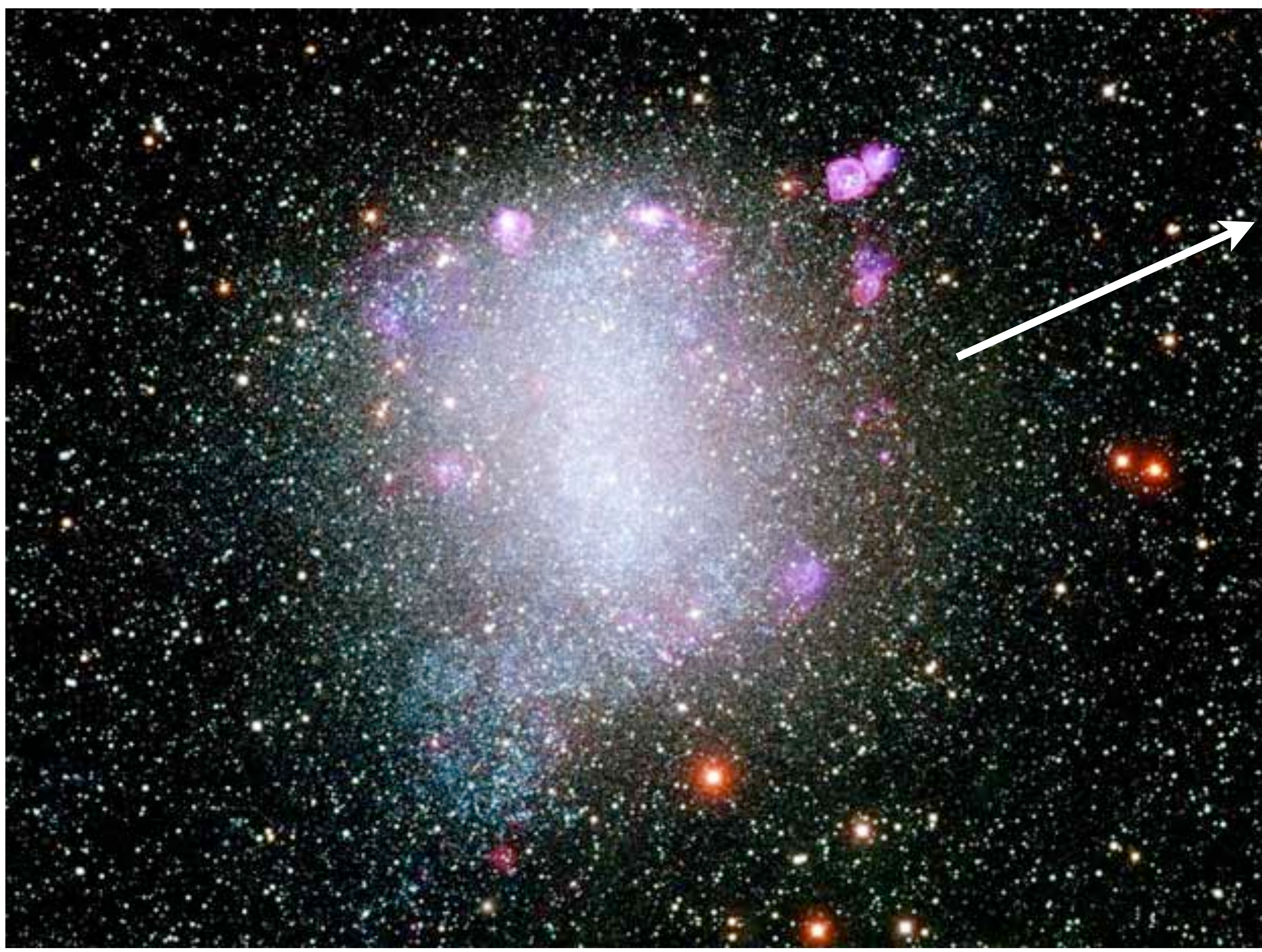


TABLE 1

NEBULAE WHOSE DISTANCES HAVE BEEN ESTIMATED FROM STARS INVOLVED OR FROM MEAN LUMINOSITIES IN A CLUSTER

OBJECT	m_s	r	v	m_t	M_t
S. Mag.	..	0.032	+ 170	1.5	−16.0
L. Mag.	..	0.034	+ 290	0.5	17.2
N. G. C. 6822	..	0.214	− 130	9.0	12.7
598	..	0.263	− 70	7.0	15.1
221	..	0.275	− 185	8.8	13.4
224	..	0.275	− 220	5.0	17.2
5457	17.0	0.45	+ 200	9.9	13.3
4736	17.3	0.5	+ 290	8.4	15.1
5194	17.3	0.5	+ 270	7.4	16.1
4449	17.8	0.63	+ 200	9.5	14.5
4214	18.3	0.8	+ 300	11.3	13.2
3031	18.5	0.9	− 30	8.3	16.4
3627	18.5	0.9	+ 650	9.1	15.7
4826	18.5	0.9	+ 150	9.0	15.7
5236	18.5	0.9	+ 500	10.4	14.4
1068	18.7	1.0	+ 920	9.1	15.9
5055	19.0	1.1	+ 450	9.6	15.6
7331	19.0	1.1	+ 500	10.4	14.8
4258	19.5	1.4	+ 500	8.7	17.0
4151	20.0	1.7	+ 960	12.0	14.2
4382	..	2.0	+ 500	10.0	16.5
4472	..	2.0	+ 850	8.8	17.7
4486	..	2.0	+ 800	9.7	16.8
4649	..	2.0	+1090	9.5	17.0

Hubble's Law

- Hubble used this equation to determine a linear relationship :

$$rK + X\cos\alpha\cos\delta + Y\sin\alpha\cos\delta + Z\sin\delta = v\ ,$$

Distance

Longitude

Latitude

Constants

Velocity

- Plotting the data :

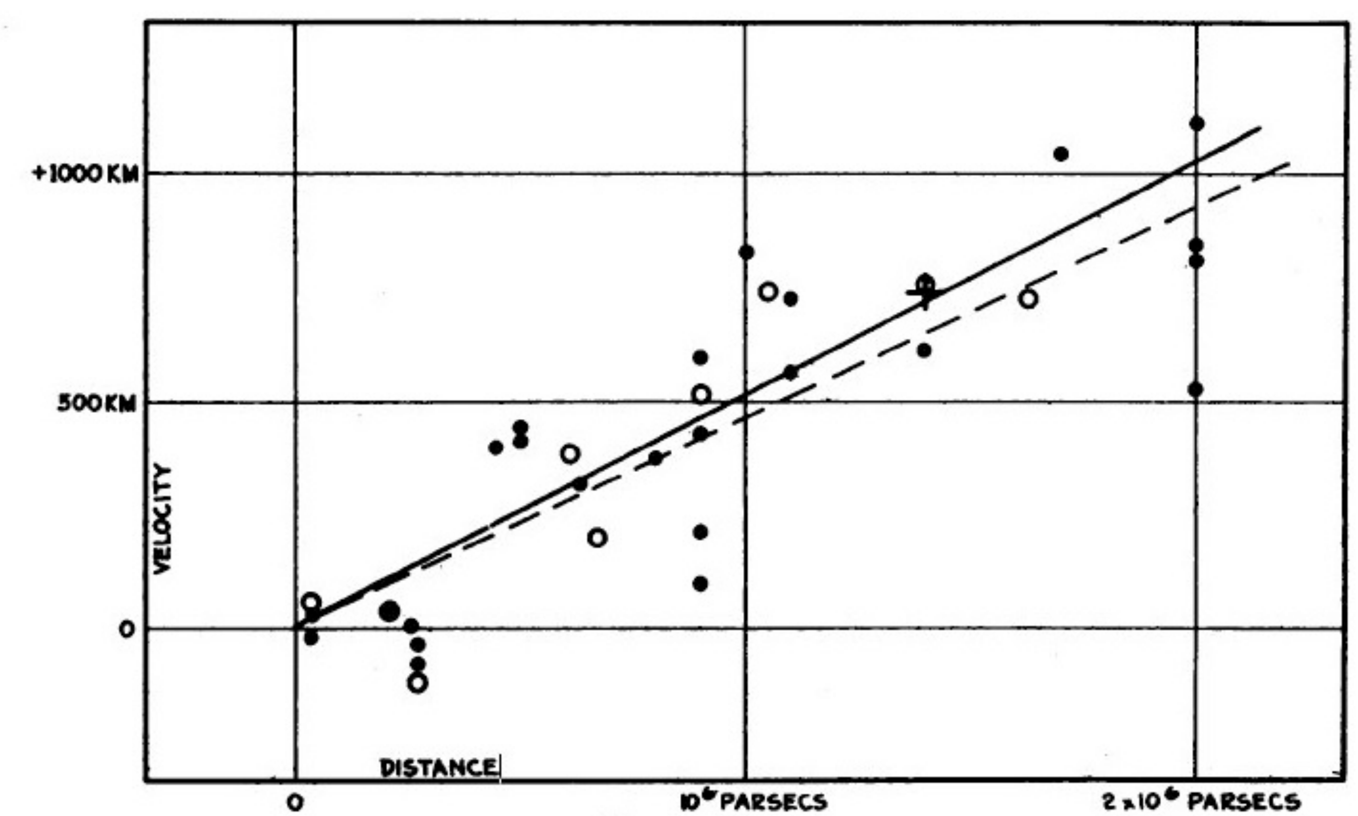
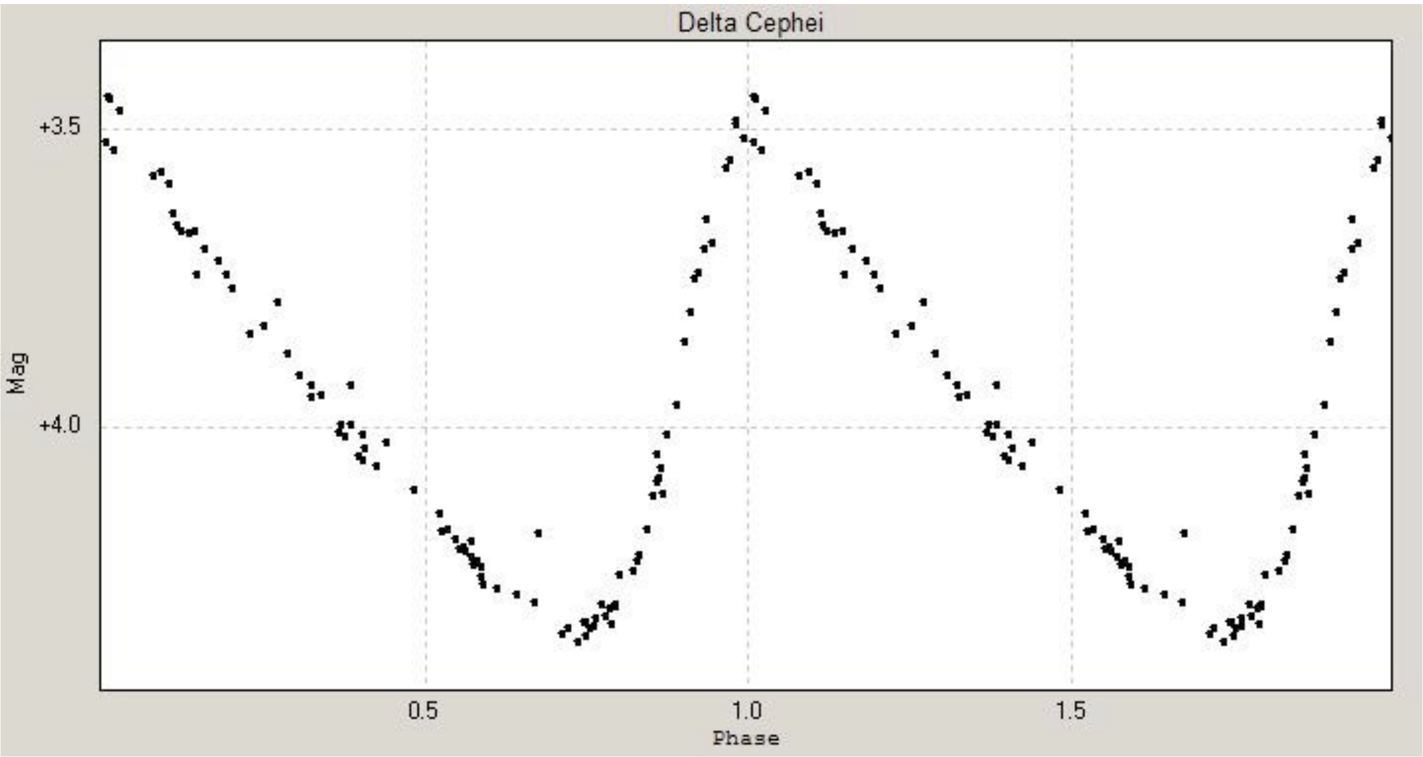


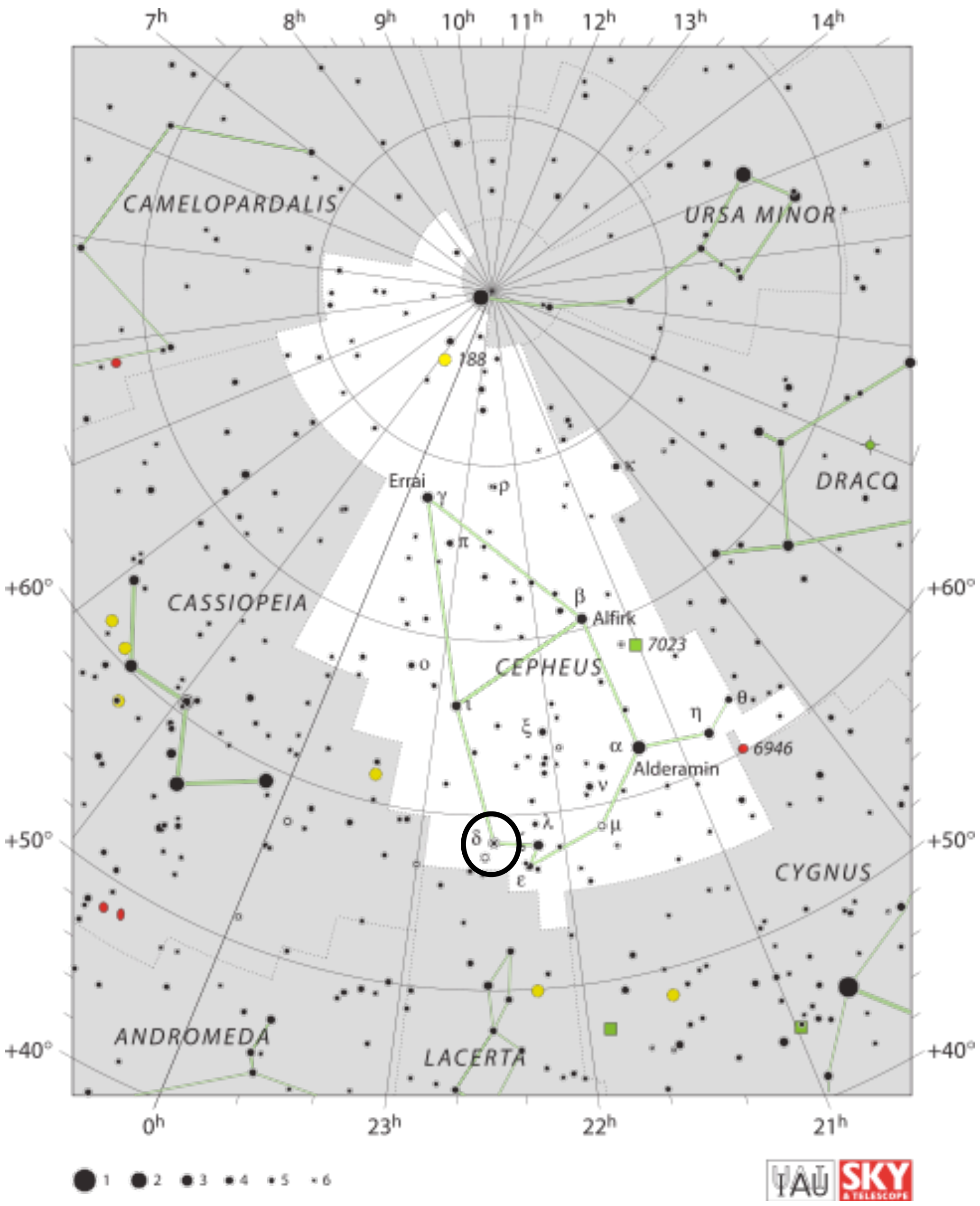
FIGURE 1

Velocity-Distance Relation among Extra-Galactic Nebulae.

Hubble's Law

- How do we get the luminosity for distant objects?
- Use Cepheid Variables!
  - http://en.wikipedia.org/wiki/Cepheid_variable
- Luminosity of the star can be estimated from its period!
  - Period is very easy to measure
  - Convert to luminosity
- For instance : Delta Cephei:
  - http://en.wikipedia.org/wiki/Delta_Cephei





Hubble's Law

- Why do we expect that the further the distance of the galaxy, the faster they should be moving away from us?

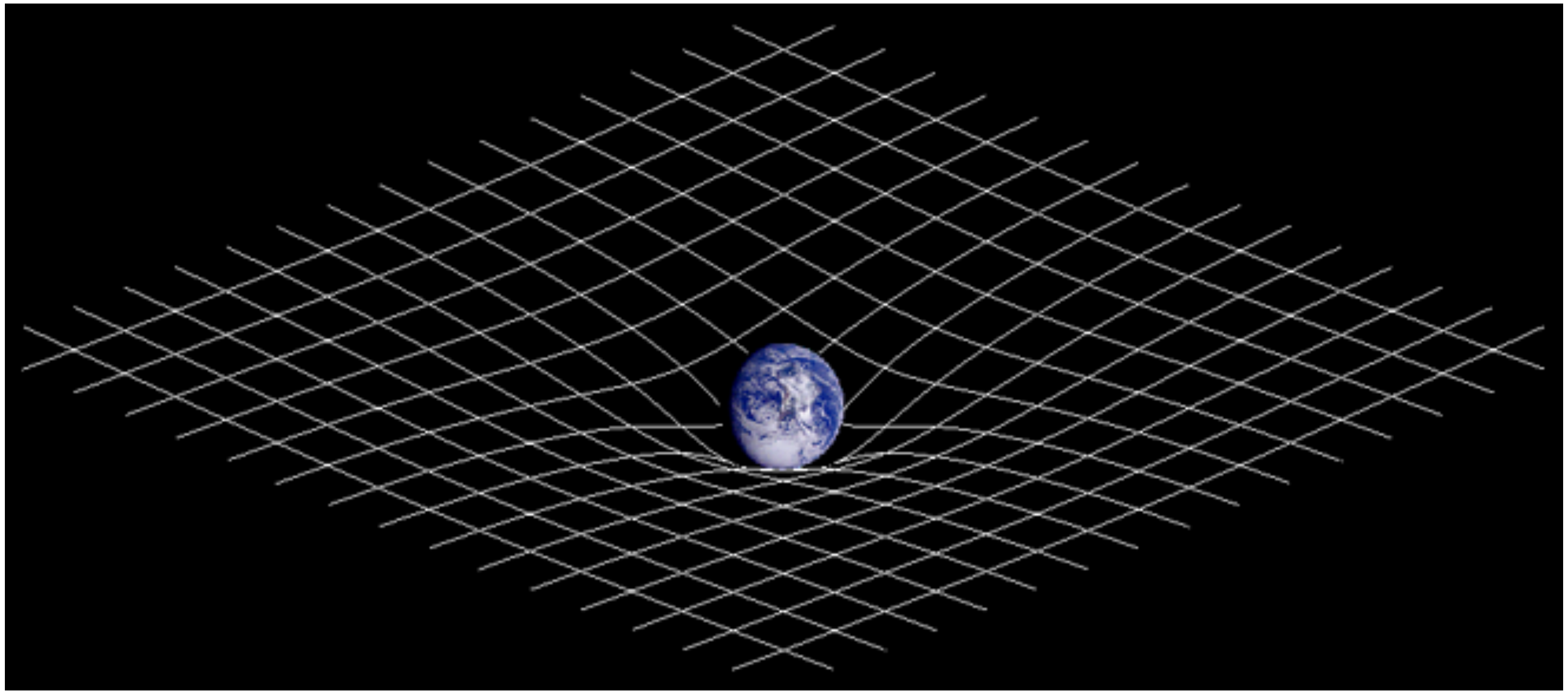
- A priori, no reason
- It just happens to be so in our universe!

- So, now for a bit of general relativity and cosmology

General Relativity

- http://en.wikipedia.org/wiki/General_relativity

- Relates gravity to the curvature of space-time!



- Objects with mass or energy distort space-time, and this induces a gravitational field

General Relativity

- Space-time is a tensor
- So gravity is a tensor
- Einstein's equations :

Newton's constant

Stress-energy tensor (energy and momentum density of matter + radiation)

$$\mathcal{R}_{\mu\nu} - \frac{1}{2} g_{\mu\nu} \mathcal{R} = \frac{8\pi G_{\mathrm{N}}}{c^4} T_{\mu\nu} - g_{\mu\nu} \Lambda$$

Ricci tensor (gravitational force)

Curvature scalar

Speed of light

Cosmological constant

Metric tensor

General Relativity

- That's a huge set of nonlinear partial differential equations, and can be arbitrarily complicated ($T_{\mu\nu}$ has no constraint to its format)
- A few simple cases can be derived :
  - If spacetime is homogeneous and isotropic, this is the Robertson-Walker metric :

Cosmological scale factor

$$ds^2 \equiv \sum_{\mu=0}^{3}\sum_{\nu=0}^{3} g_{\mu\nu}dx^\mu dx^\nu = c^2dt^2 - R^2(t)\left[\frac{dr^2}{1-kr^2} + r^2(d\theta^2 + \sin^2\theta d\phi^2)\right] ,$$

  - Assuming that the matter+radiation behave like a uniform perfect fluid with density ρ and pressure p, this is the Friedmann-Lamaitre equations:

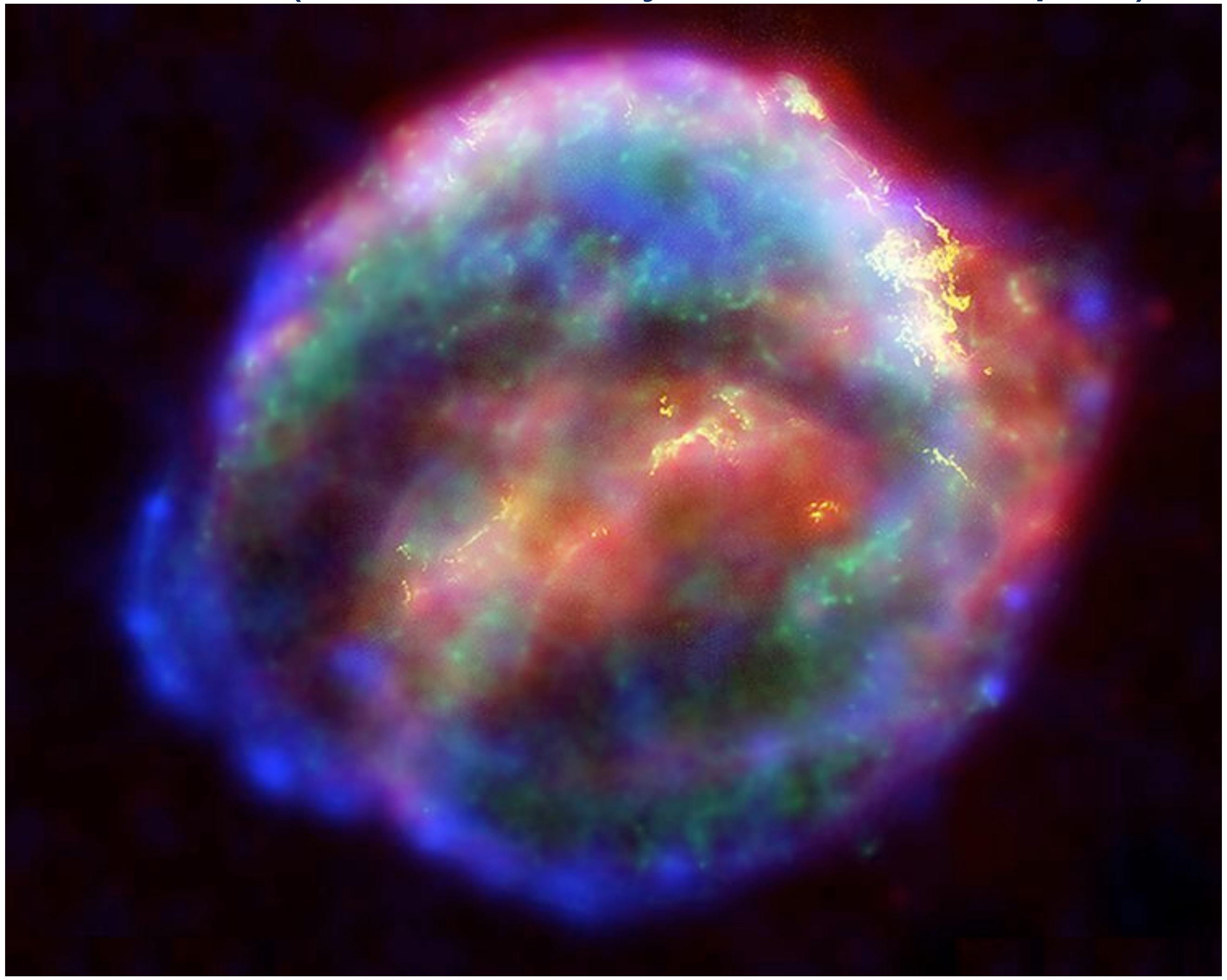
$$H^2 \equiv \left(\frac{\dot{R}}{R}\right)^2 = \frac{8\pi G_N \rho}{3} - \frac{kc^2}{R^2} + \frac{\Lambda c^2}{3} , \qquad \frac{\ddot{R}}{R} = -\frac{4\pi G_N}{3}(\rho + 3p) + \frac{\Lambda c^2}{3} .$$

Hubble parameter : H(t0) = 72 km/s/Mpc at present time

Supernovae

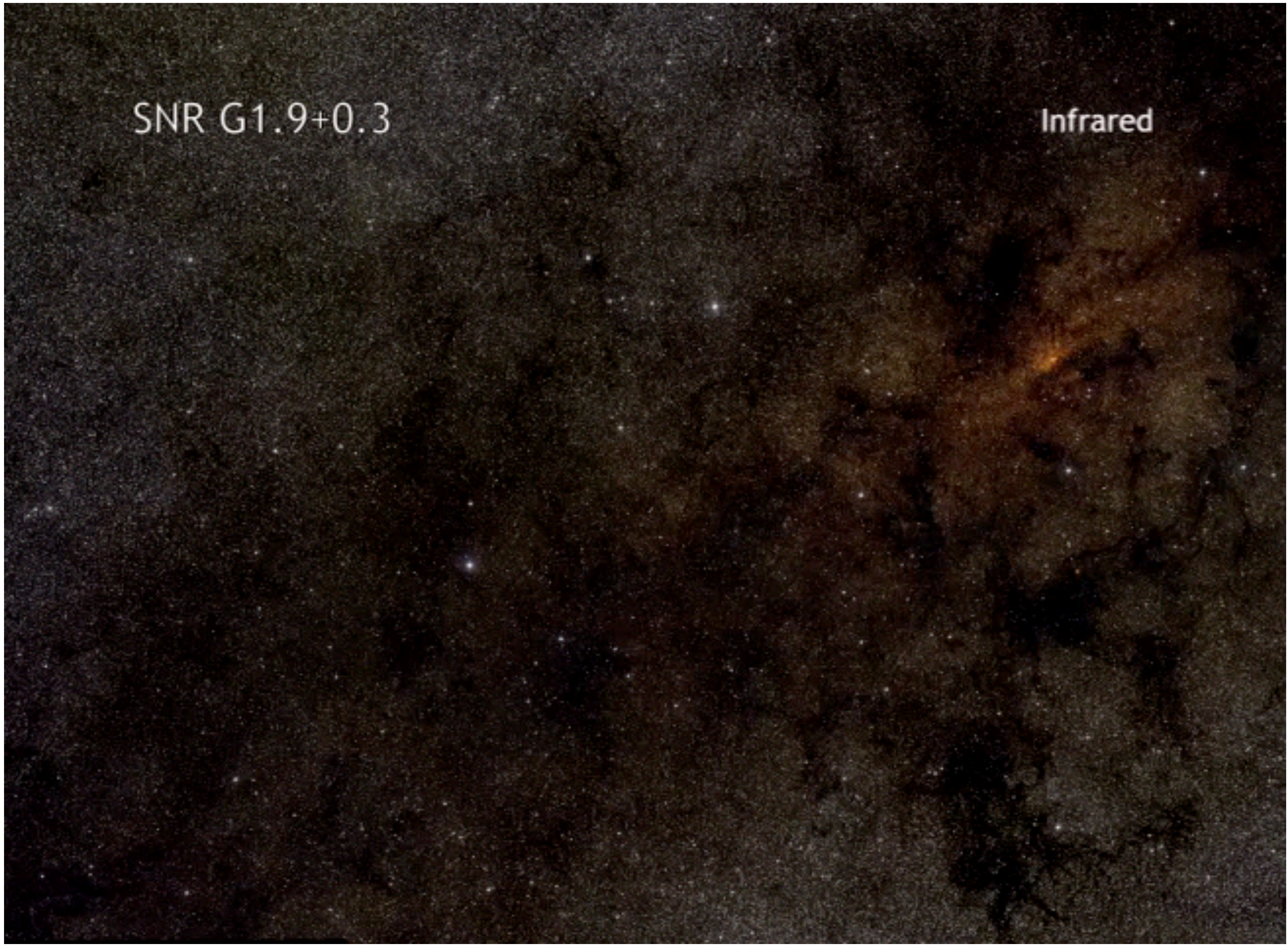
- Supernovae occur when a star exhausts its hydrogen fuel, and blows off the outer shell
- It reduces in size, but the Pauli exclusion principle prevents collapse
  - White dwarf
- White dwarf then accretes material from nearby stars
- The core explodes in a thermonuclear event
- That's the supernova!
- This emits light at specific frequencies, which can be used to estimate the distance!

SN 1604 (discovered by Johannes Kepler)



Supernovae

- Supernovae occur when a star exhausts its hydrogen fuel, and blows off the outer shell
- It reduces in size, but the Pauli exclusion principle prevents collapse
  - White dwarf



- White dwarf then accretes material from nearby stars
- The core explodes in a thermonuclear event
- That's the supernova!
- This emits light at specific frequencies, which can be used to estimate the distance!

http://chandra.harvard.edu/photo/2013/g19/g19_BU_sm_web.mov

Supernovae

- PDG's Review of big bang cosmology has a nice set of data :
- http://pdg.lbl.gov/2012/reviews/rpp2012-rev-bbang-cosmology.pdf
- Brightness is measured by absolute magnitude "M"
- Apparent magnitude is "m"
- M is equal to m at 10 pc
- r is distance in pc
- Distance modulus is :

$$\mu \equiv m - M = 5 \log_{10} r - 5 \,,$$

- Luminosity distance is :

$$D_L = 10^{\frac{(m-M)}{5}+1}$$

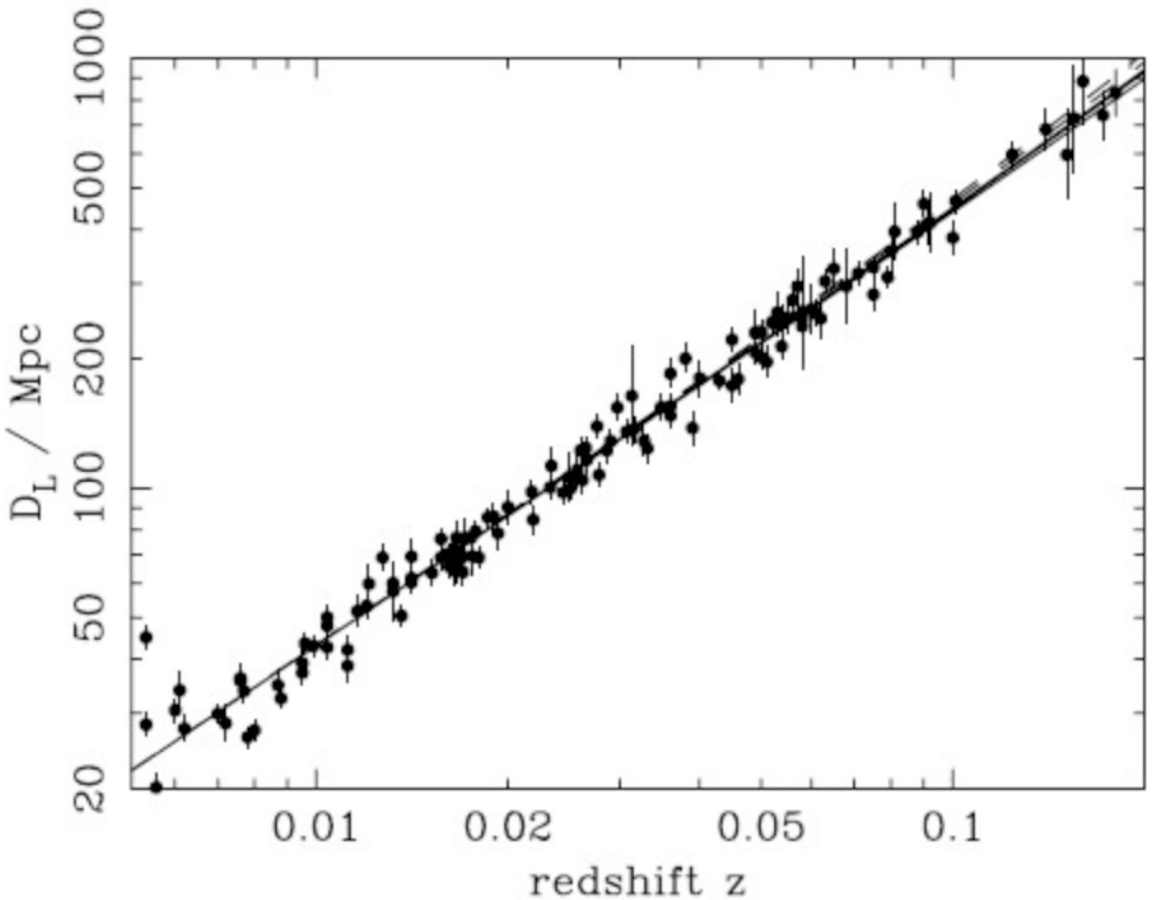
Supernovae

- The distance modulus is approximately (for distant SN's)

$$\mu = 25 + 5\log_{10}\left(\frac{cz}{H_0}\right) + 1.086(1 - q_0)z + \ldots$$

- Combine with G.R. doppler shift :

$$z + 1 = \frac{\nu_1}{\nu_2} = \frac{R_2}{R_1} \simeq 1 + \frac{v_{12}}{c}$$



http://dark.dark-cosmology.dk/~tamarad/SN/

- We conclude that faster objects have more redshift!
- There is a linear relationship between brightness and redshift for supernovae!

(Careful : we will use the distance modulus and not luminosity distance since that's what we have data for)

Supernovae

- Specifics don’t matter here, but we just want to state the relation of redshifts of galaxies to their velocities
- Obtained from G.R. doppler shift!

Frequency when emitted by the galaxy

Scale factor when detected by observer on earth

Redshift

Speed of the galaxy relative to observer

$$z + 1 = \frac{\nu_1}{\nu_2} = \frac{R_2}{R_1} \simeq 1 + \frac{v_{12}}{c}$$

Frequency when detected by observer on earth

Scale factor when emitted by the galaxy

I don’t expect you to be able to derive this, but we’ll just fit the data

Recall : Development

- Step 1 : Write the algorithm down on paper
- Step 2 :
  - If you don’t understand everything : goto step 1
  - else : continue
- Step 3 : write pseudocode
- Step 4 : continue
  - If you don’t understand everything : goto step 3
  - else : continue
- Step 5 : write code
- Step 6 : check code with unit tests
  - Check “pass” criterion
  - Check “fail” criterion
  - If unit test fails : goto Step 5
  - else : continue
- Step 7 : Publish!

Linear fits

- Want to fit a line to a bunch of points
- Let's think for a bit about what this means and how we should expect to implement it

Cepheid variables

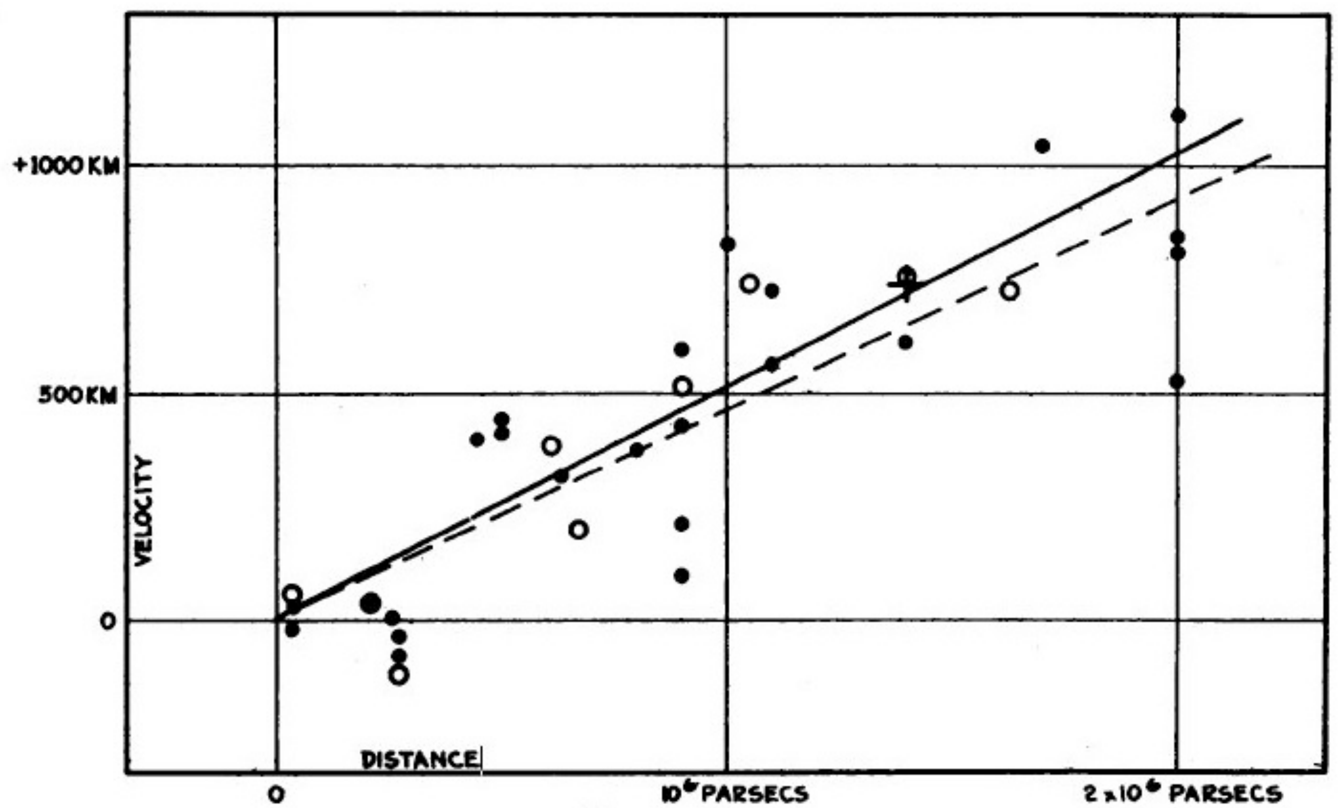
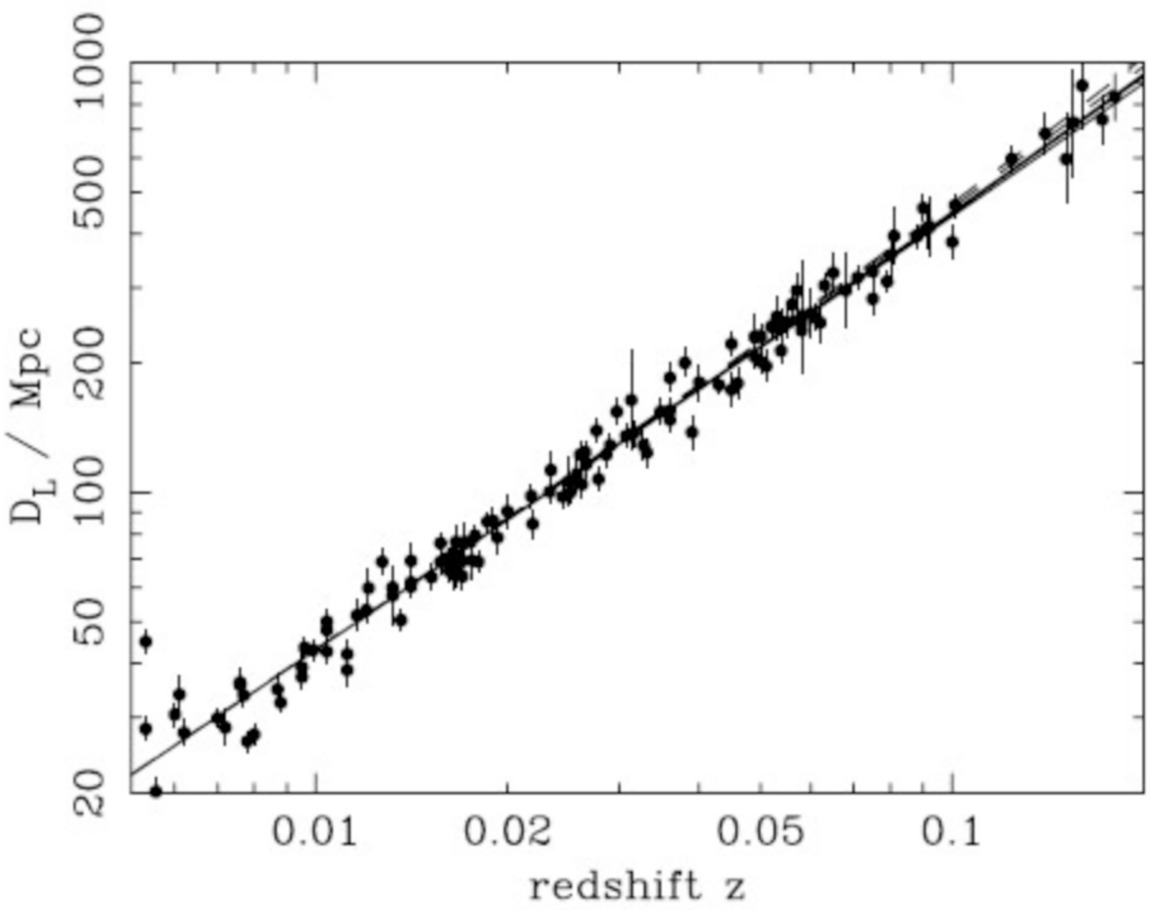


FIGURE 1
Velocity-Distance Relation among Extra-Galactic Nebulae.

Supernovae



Linear fits

- Think about the simplest case : 1 point.
  - What happens here?

Linear fits

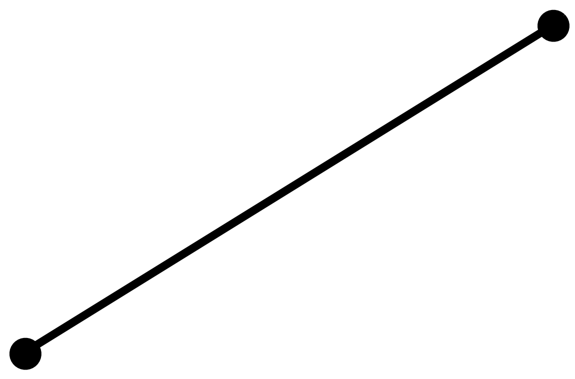
- Think about the simplest case : 1 point.
  - What happens here?
- Nothing! You can't fit a line to a point.
  - So, this should be checked before we attempt to fit anything

Linear fits

- OK, next simplest case : 2 points
  - What happens here?

Linear fits

- OK, next simplest case : 2 points
  - What happens here?
- That's easy too : there's no fit, you just draw a line

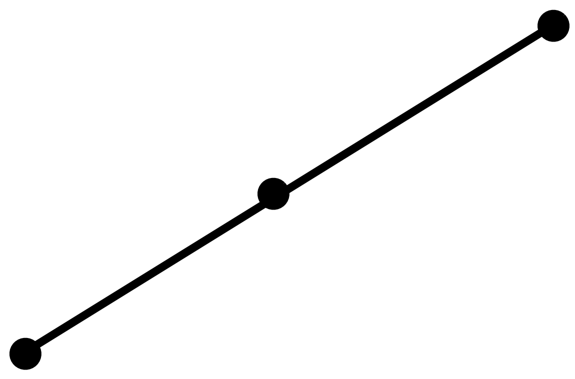


Linear fits

- What about 3 points?
  - Now this gets interesting!

- There's degeneracy that you can exploit

- If the three points are colinear, this works

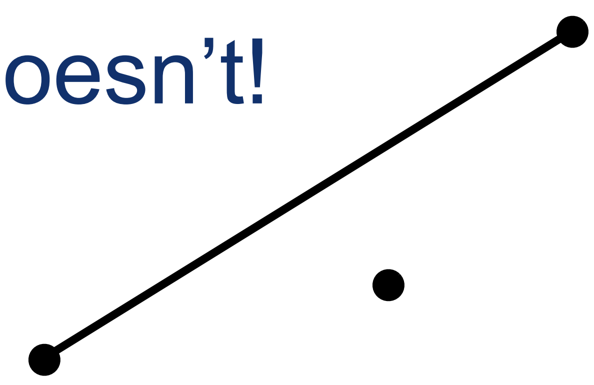


Linear fits

- What about 3 points?
  - Now this gets interesting!

- There's degeneracy that you can exploit

- If the three points are colinear, this works
- Otherwise, it doesn't!

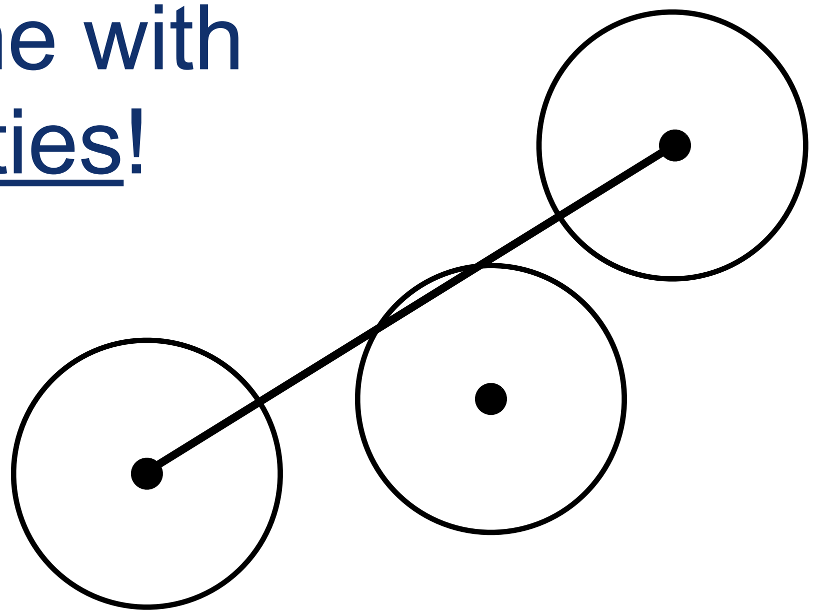


Cannot draw a line to connect these three!

Linear fits

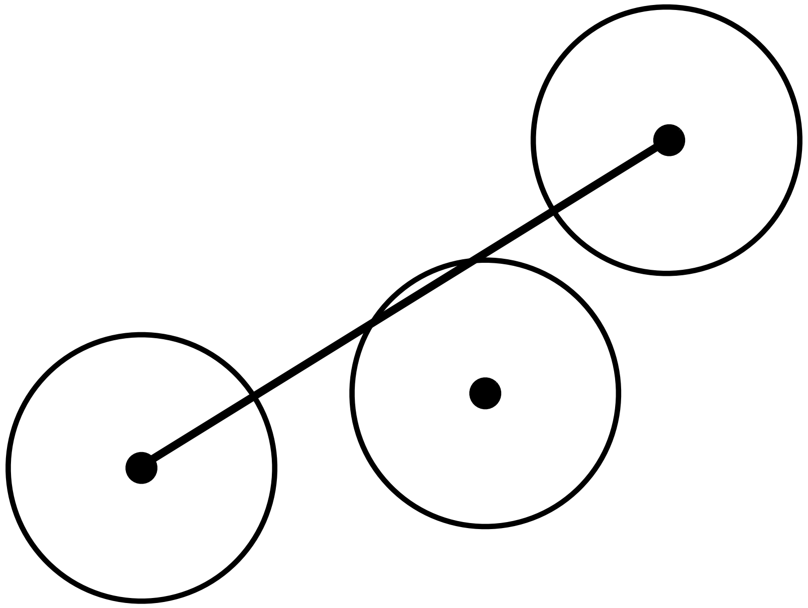
- So what do we do? We can't just give up!

- Very important aspect to remember :
  - These points are not points : they are actually **ellipses**!
  - They come with uncertainties!



Linear fits

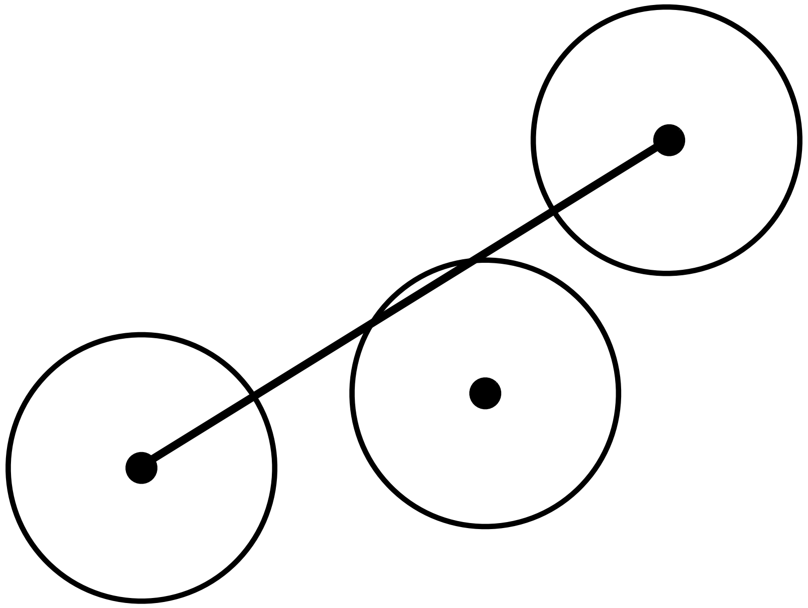
- The uncertainties can come from two sources :
  - Statistical sampling



  - Systematic effects

Uncertainty

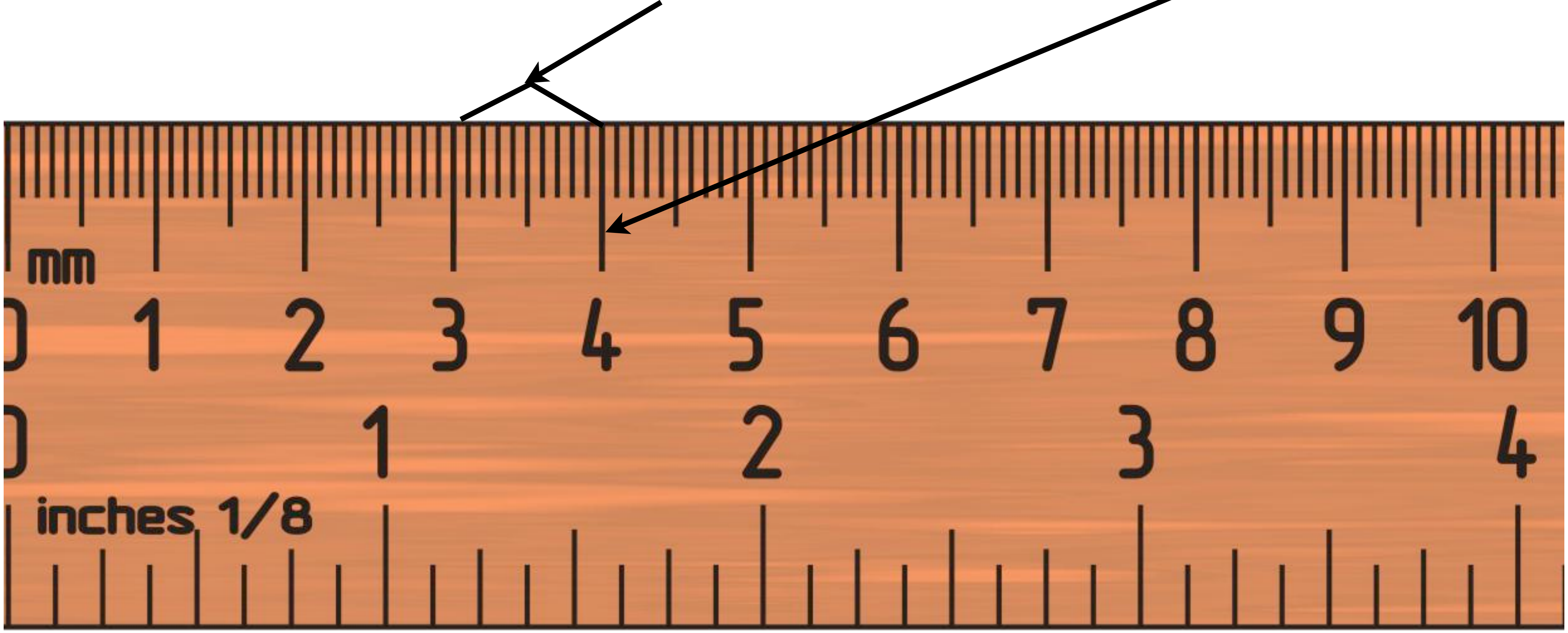
- The uncertainties can come from two sources :
  - Statistical sampling
    - From variations in repeated trials
    - Mathematically “well-behaved”
    - Easy to estimate



  - Systematic effects
    - Intrinsic uncertainty from non-deterministic sources
    - Not mathematically “well-behaved”
    - Difficult to estimate

Uncertainty

- Example : measuring distance with a ruler
  - Systematic limitation : ruler has finite width of the lines, and finite number of lines to measure!



  - Statistical variation : you can try to repeat the same measurement over and over to get a better estimate of the "true" value

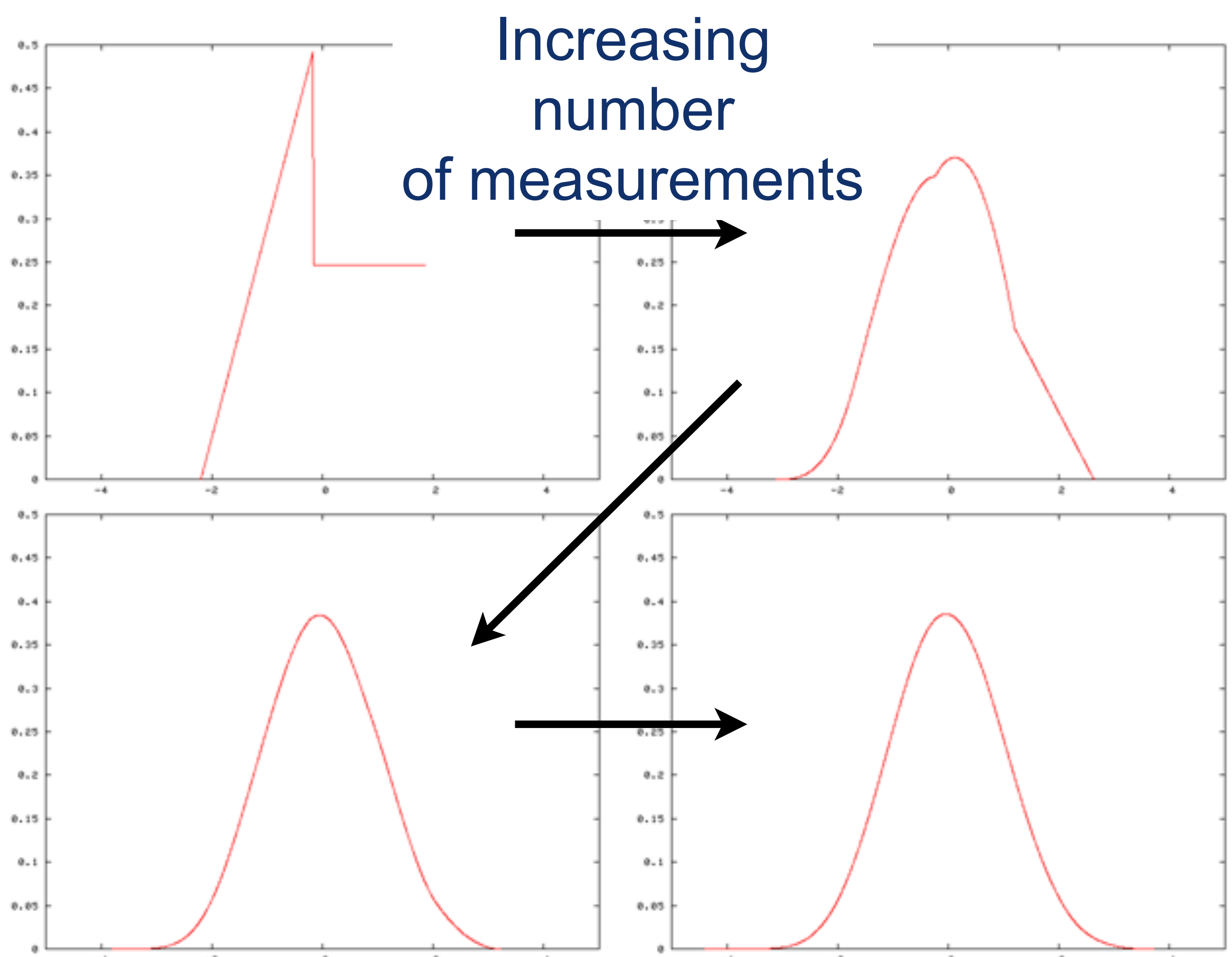
Uncertainty

- Easy one first : statistical uncertainties

- Problem stated :
  - We have a true value $\tilde{x}$
  - We have several measured values $x_0, x_1, \ldots x_{n-1}$
  - How well can we estimate $\tilde{x}$ given $x_0, x_1, \ldots x_{n-1}$?

Uncertainty

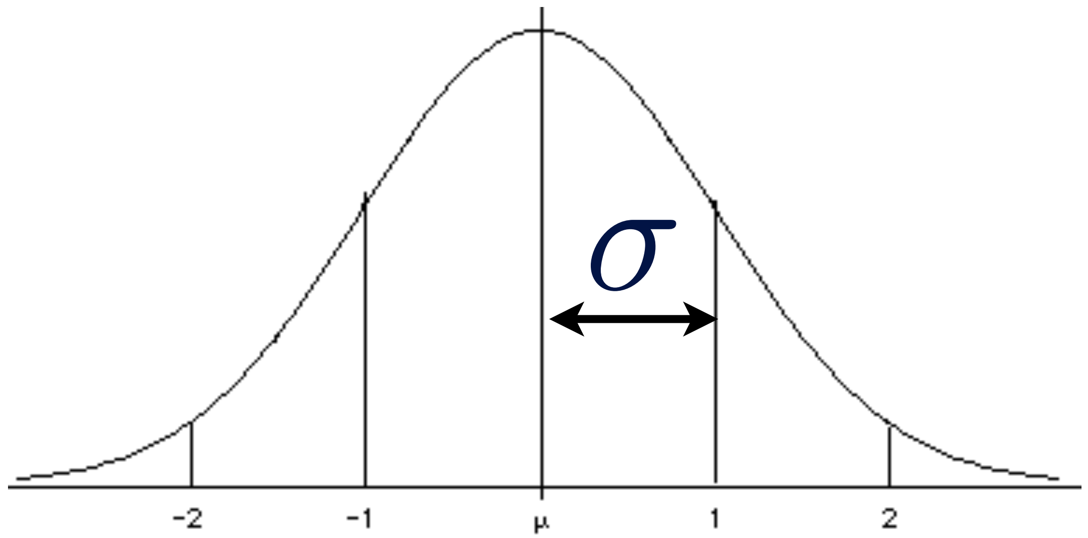
- Central limit theorem!
  - http://en.wikipedia.org/wiki/Central_limit_theorem
- If your measurements are uncorrelated :
  - As you make more measurements, they follow a Gaussian (or “normal”, or bell-shaped curve) distribution
  - http://en.wikipedia.org/wiki/Normal_distribution



Called a “probability distribution function” (or PDF)

Uncertainty

- So, the distribution of values will follow a Gaussian distribution for statistical uncertainties
- We usually quote the "sigma" (σ) of the Gaussian as the uncertainty band



- What about systematic effects?
  - If you repeat the trial again and again, what happens?
  - It's a systematic effect, so you actually get basically the same thing!

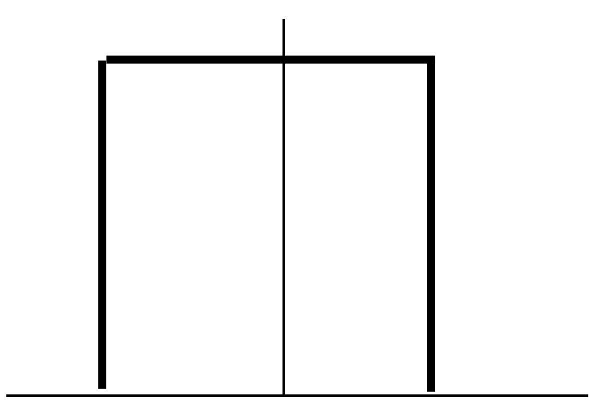
Uncertainty

- Systematic uncertainties are very hard to estimate

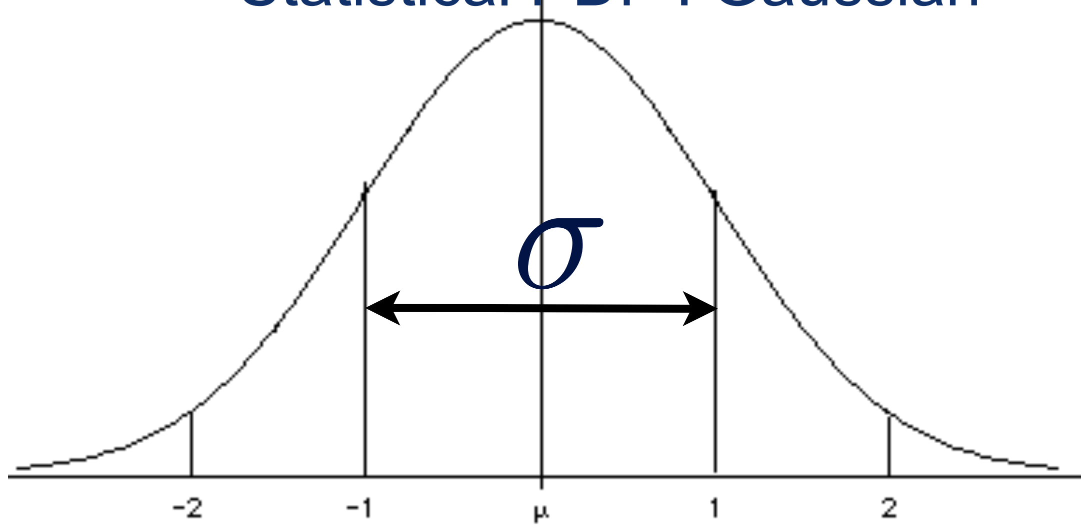
- Typically we (as scientists) “reckon” them in some way
  - Control samples
  - Minimum resolution of your device
  - And so on

- So, the probability distribution function for these are basically FLAT
  - You don’t know where it is, but it’s somewhere within that range

Systematic PDF : “Box”



Statistical PDF : Gaussian



Uncertainty

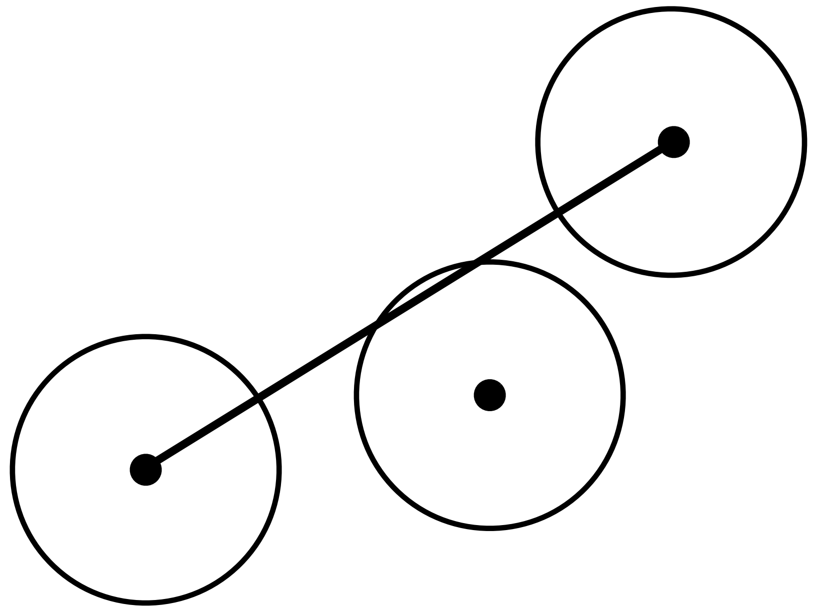
- You now say : “Sal, I thought you were supposed to teach me about programming? I haven’t seen a line of code yet!”
- “Ah, my good students,” I reply. “But remember we must understand what we’re doing!”

Uncertainty

- So why am I telling you all of this?
  - The systematic effects don't follow a mathematical formalism
  - The statistical effects do follow a mathematical formalism
  - So, we usually just pretend that systematic effects are like statistical effects, and assume Gaussian uncertainties too!

Linear fits

- So, back to linear fits!
- What the heck are these circles?
  - They're the uncertainties!

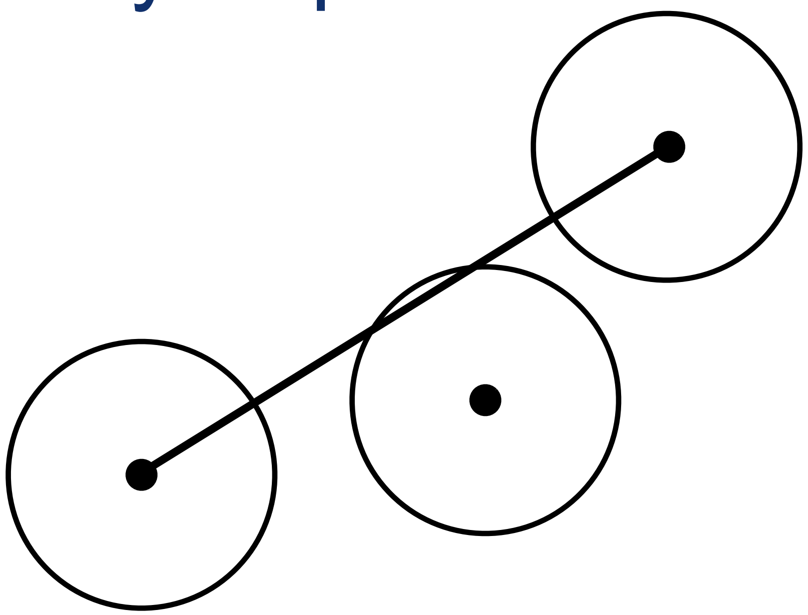


- We'll just pretend that they're statistical
  - Scientists usually pretend they're Gaussian anyway!

Linear fits

- So now, what do we actually want to do?

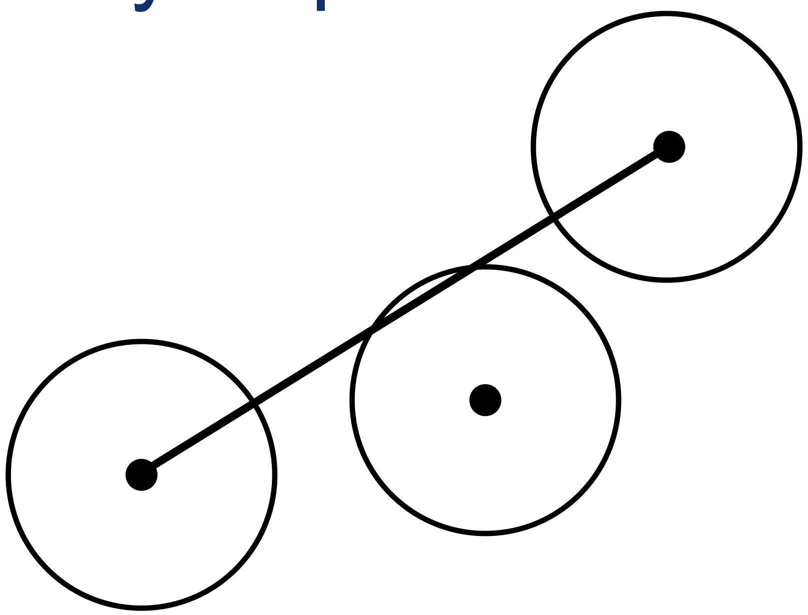
- We want to draw the line that intersects all of the possible uncertainty ellipses :



Linear fits

- So now, what do we actually want to do?

- We want to draw the line that intersects all of the possible uncertainty ellipses :

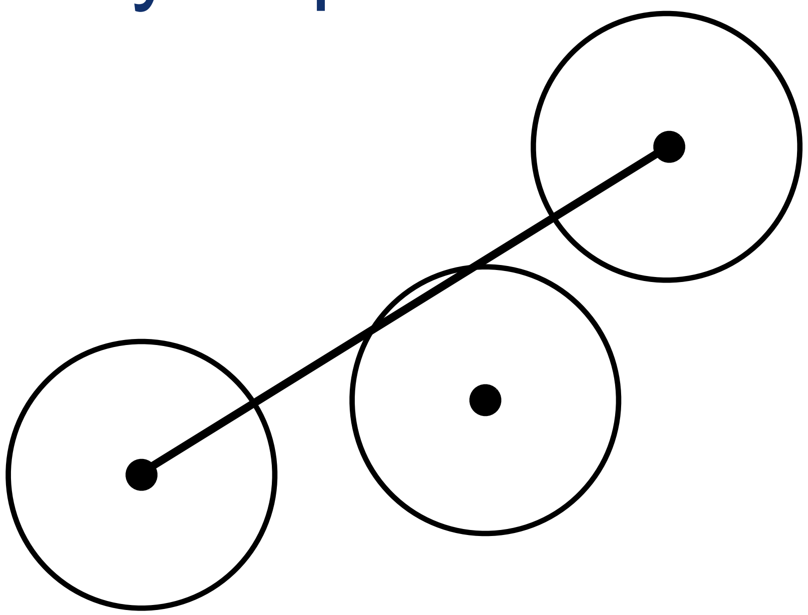


BAD : we're ignoring the third point entirely! So what to do?

Linear fits

- So now, what do we actually want to do?

- We want to draw the line that intersects all of the possible uncertainty ellipses :



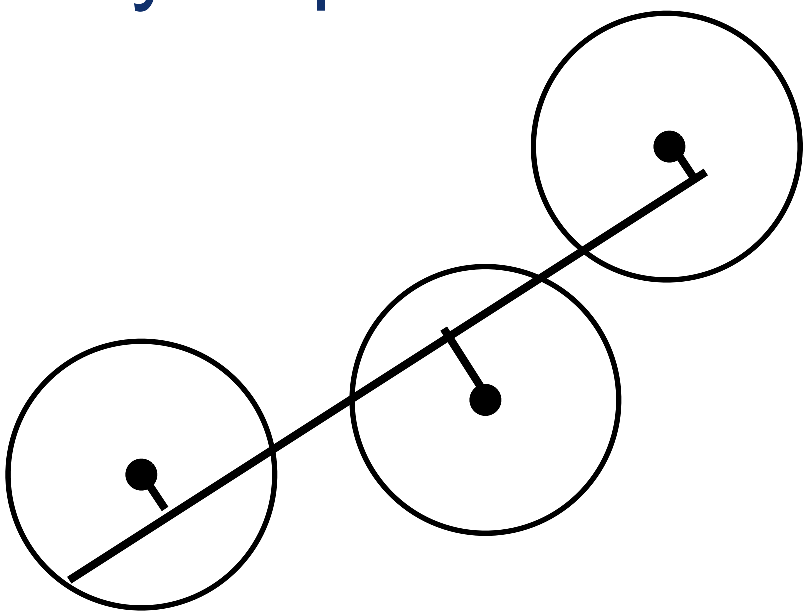
BAD : we're ignoring the third point entirely! So what to do?

Minimize the least-squares distance!

Linear fits

- So now, what do we actually want to do?

- We want to draw the line that intersects all of the possible uncertainty ellipses :



Minimize the least-squares distance!

Linear fits

- So what exactly do we want to compute, and what do we want to minimize?
- Assume the data are described by

$$y(x) = a + bx$$

- Presume first that all of the uncertainties are exactly the same
- Then you just have to minimize the distance :

$$f(a,b) \equiv \sum_{i=0}^{n-1} \left(y_i - a - bx_i\right)^2$$

Linear fits

- At the minimum, the derivatives are zero:

$$\frac{\partial f}{\partial a} = -2\sum_{i=0}^{n-1}(y_i - a - bx_i) = 0\ , \qquad \text{and} \qquad \frac{\partial f}{\partial b} = -2\sum_{i=0}^{n-1}x_i(y_i - a - bx_i) = 0$$

- Can solve these simultaneously for the two unknowns a and b
  - Two equations, two unknowns!
- Define the quantities :

$$s_x \equiv \sum_{i=0}^{n-1} x_i\ , \qquad s_y \equiv \sum_{i=0}^{n-1} y_i\ , \qquad s_{xx} \equiv \sum_{i=0}^{n-1} x_i^2\ , \qquad s_{xy} \equiv \sum_{i=0}^{n-1} x_i y_i\ .$$

Average of xi

Average of yi

Standard deviation of xi

cross-term

Linear fits

- With this definition, can compute a and b :

$$a = \frac{s_{xx}s_y - s_x s_{xy}}{n s_{xx} - s_x^2}, \qquad b = \frac{n s_{xy} - s_x s_y}{n s_{xx} - s_x^2}.$$

  – This algorithm is discussed in Section 15.2: Fitting data to a straight line of Numerical Recipes.

  – (you can access a certain number of pages per month for free... but in any case, you don't need this if you don't want to use it)

Linear fits

- But! We're not quite done.
- What are the uncertainties on a and b?
- Actually, what we want is the uncertainty per degree of freedom of the fit
- Degrees of freedom is :
  number of data points - number of constraints

- For a linear fit we have two constraints

- Variance is therefore :

$$\sigma^2 \equiv \frac{f(a,b)}{\nu} = \frac{1}{n-2}\sum_{i=0}^{n-1}\left(y_i - y(x_i)\right)^2 \ .$$

Linear fits

- What if the uncertainties are not all equal?

- The same principle applies, but instead of minimizing the mean-squared distance :

$$f(a,b) \equiv \sum_{i=0}^{n-1} \left(y_i - a - bx_i\right)^2$$

- You instead minimize the "chi-squared" which is the distance divided by the uncertainty :

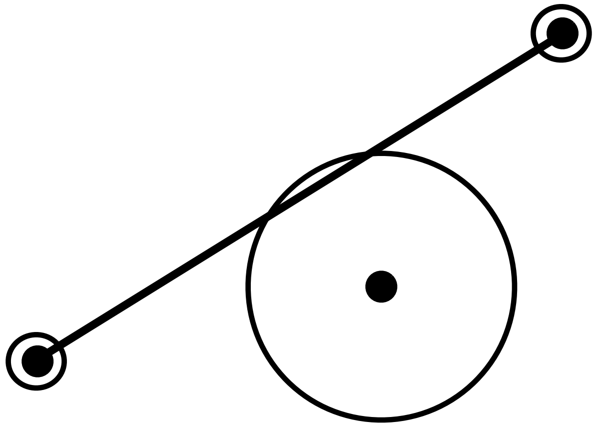
$$\chi^2(a,b) \equiv \sum_{i=0}^{n-1} \left(\frac{y_i - a - bx_i}{\sigma_i}\right)^2 .$$

Note : This is only strictly true for GAUSSIAN uncertainties!

Linear fits

- How does this help?

- It “ignores” values with large uncertainties :



- The two points on the ends are very precise
- The third point in the middle is not precise
- Good point to check for a unit test!

Linear fits

- So how does this get modified?
- Parameters and uncertainties are :

$$b = \frac{1}{S_{tt}} \sum_{i=0}^{n-1} \frac{t_i y_i}{\sigma_i} , \qquad a = \frac{S_y - S_x b}{S} \qquad \sigma_a^2 = \frac{1}{S} \left(1 + \frac{S_x^2}{S S_{tt}} \right) , \qquad \sigma_b^2 = \frac{1}{S_{tt}} .$$

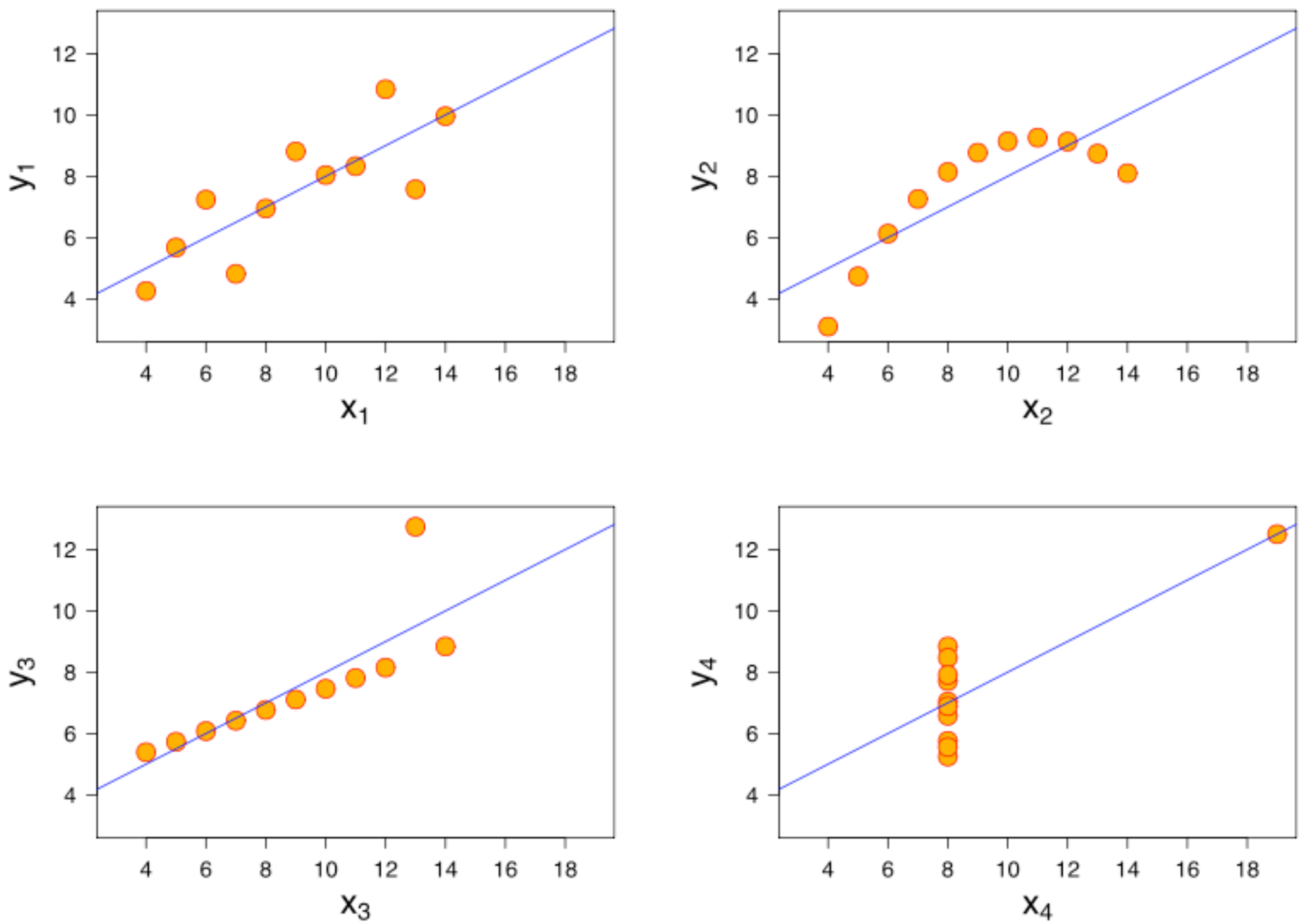
- Where :

$$t_i = \frac{1}{\sigma_i} \left(x_i - \frac{S_x}{S} \right) , \qquad S_{tt} = \sum_{i=0}^{n-1} t_i^2 ,$$

$$S = \sum_{i=0}^{n-1} \frac{1}{\sigma_i^2} , \qquad S_x = \sum_{i=0}^{n-1} \frac{x_i}{\sigma_i^2} , \qquad S_y = \sum_{i=0}^{n-1} \frac{y_i}{\sigma_i^2} .$$

Linear fits

- With uncertainties on the inputs, you can compute "goodness of fit"
- Why do you care?
- All of these have the same fit :



Obviously, several of these are bad!

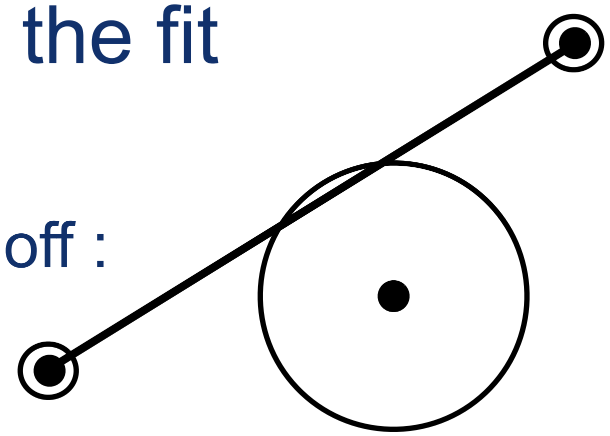
Linear fits

- The chi-squared per degree of freedom is what we're looking for here

$$\chi^2/\mathrm{d.o.f} \equiv \frac{1}{n-2}\sum_{i=0}^{n-1}\left(\frac{y_i - a - bx_i}{\sigma_i}\right)^2 \approx 1\ .$$

- Roughly speaking, it's the number of "standard deviations" that you're "off" in the fit

About 1 S.D. off :



- If you've estimated your uncertainties correctly, it should follow a distribution of values called the "chi-squared" distribution :

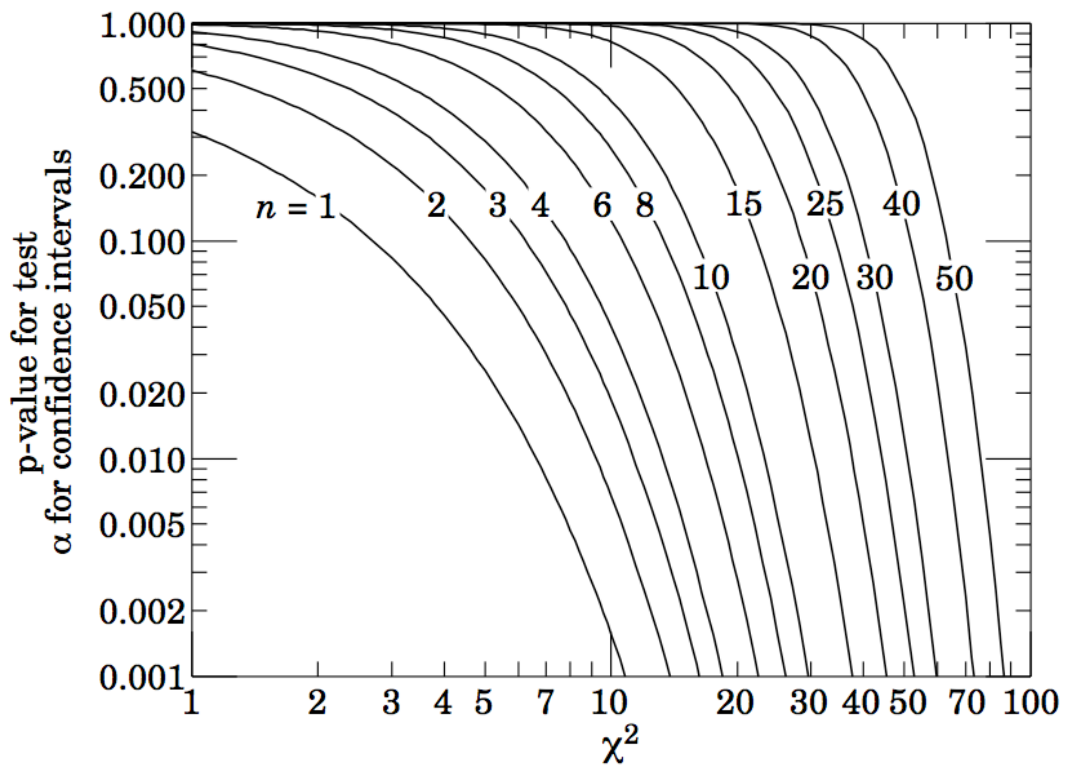


Figure 36.1: One minus the χ^2 cumulative distribution, $1 - F(\chi^2; n)$, for n degrees of freedom. This gives the p-value for the χ^2 goodness-of-fit test as well as one minus the coverage probability for confidence regions (see Sec. 36.3.2.4).

- http://en.wikipedia.org/wiki/Chi-squared_distribution
- http://pdg.lbl.gov/2013/reviews/rpp2012-rev-statistics.pdf

Linear fits

- OK : moment of truth
- We have two cases : with, and without uncertainties
- Let’s do without first, it’s easier

Recall : Development

- Step 1 : Write the algorithm down on paper
- Step 2 :
  - If you don’t understand everything : goto step 1
  - else : continue
- Step 3 : write pseudocode
- Step 4 : continue
  - If you don’t understand everything : goto step 3
  - else : continue
- Step 5 : write code
- Step 6 : check code with unit tests
  - Check “pass” criterion
  - Check “fail” criterion
  - If unit test fails : goto Step 5
  - else : continue
- Step 7 : Publish!

Pseudocode : No uncertainties

$$s_x \equiv \sum_{i=0}^{n-1} x_i \ , \qquad s_y \equiv \sum_{i=0}^{n-1} y_i \ ,$$

$$s_{xx} \equiv \sum_{i=0}^{n-1} x_i^2 \ , \qquad s_{xy} \equiv \sum_{i=0}^{n-1} x_i y_i \ .$$

$$a = \frac{s_{xx}s_y - s_x s_{xy}}{n s_{xx} - s_x^2}$$

$$b = \frac{n s_{xy} - s_x s_y}{n s_{xx} - s_x^2}$$

$$\sigma^2 \equiv \frac{f(a,b)}{\nu} = \frac{1}{n-2} \sum_{i=0}^{n-1} (y_i - y(x_i))^2 \ .$$

```
input data
n = size of data
if n < 2 :
   print 'Error! Not enough data!'
   return
for i = 0... N-1 :
   s_x += x_i
   s_y += y_i
   s_xx += x_i**2
   s_xy += x_i*y_i
den = n * s_xx - s_x*s_x
if abs( den ) < 0.000001 :
   print 'Error! Denominator is zero!'
   return
a = (s_xx * s_y - s_x * s_xy) / den
b = (n*s_xy - s_x * s_y) / den
for i = 0... N-1 :
  sigma2 += (y_i - (a*x_i+b))**2
sigma2 = sigma2 / (n-2)
```

Recall : Development

- Step 1 : Write the algorithm down on paper
- Step 2 :
  - If you don’t understand everything : goto step 1
  - else : continue
- Step 3 : write pseudocode
- Step 4 : continue
  - If you don’t understand everything : goto step 3
  - else : continue
- Step 5 : write code
- Step 6 : check code with unit tests
  - Check “pass” criterion
  - Check “fail” criterion
  - If unit test fails : goto Step 5
  - else : continue
- Step 7 : Publish!

Recall : Development

- Step 1 : Write the algorithm down on paper
- Step 2 :
  - If you don’t understand everything : goto step 1
  - else : continue
- Step 3 : write pseudocode
- Step 4 : continue
  - If you don’t understand everything : goto step 3
  - else : continue
- Step 5 : write code
- Step 6 : check code with unit tests
  - Check “pass” criterion
  - Check “fail” criterion
  - If unit test fails : goto Step 5
  - else : continue
- Step 7 : Publish!

Recall : Development

- Step 1 : Write the algorithm down on paper
- Step 2 :
  - If you don’t understand everything : goto step 1
  - else : continue
- Step 3 : write pseudocode
- Step 4 : continue
  - If you don’t understand everything : goto step 3
  - else : continue
- Step 5 : write code
- Step 6 : check code with unit tests
  - Check “pass” criterion
  - Check “fail” criterion
  - If unit test fails : goto Step 5
  - else : continue
- Step 7 : Publish!

Write code : No uncertainties

Pseudocode → python

```
input data
n = size of data
if n < 2 :
   print 'Error! Not enough data!'
   return
for i = 0... N-1 :
   s_x += x_i
   s_y += y_i
   s_xx += x_i**2
   s_xy += x_i*y_i
den = n * s_xx - s_x*s_x
if abs( den ) < 0.000001 :
   print 'Error! Denominator is zero!'
   return
a = (s_xx * s_y - s_x * s_xy) / den
b = (n*s_xy - s_x * s_y) / den
for i = 0... N-1 :
  sigma2 += (y_i - (a*x_i+b))**2
sigma2 = sigma2 / (n-2)
```

```python
n = len(x)    # number of galaxies

if n <= 2 :
    print ('Error! Need at least two data points!')
    exit()

# Compute all of the stat. variables we need
s_x = np.sum(x)
s_y = np.sum(y)
s_xx = np.sum( x**2 )
s_xy = np.sum( x*y )
denom = n * s_xx - s_x**2
if abs( denom ) < 0.000001 :
    print ('Error! Denomominator is zero!')
    exit()

# Compute y-intercept and slope
a = (s_xx * s_y - s_x * s_xy) / denom
b = (n*s_xy - s_x * s_y) / denom

# Compute uncertainties
if n > 2 :
      sigma = np.sqrt(np.sum((y - (a+b*x))**2 ) / (n-2))
      sigma_a = np.sqrt(sigma**2 * s_xx / denom)
      sigma_b = np.sqrt(sigma**2 * n / denom)
else :
      sigma = 0.
      sigma_a = 0.
      sigma_b = 0.

return [a, b, sigma, sigma_a, sigma_b]
```

Recall : Development

- Step 1 : Write the algorithm down on paper
- Step 2 :
  - If you don’t understand everything : goto step 1
  - else : continue
- Step 3 : write pseudocode
- Step 4 : continue
  - If you don’t understand everything : goto step 3
  - else : continue
- Step 5 : write code
- Step 6 : check code with unit tests
  - Check “pass” criterion
  - Check “fail” criterion
  - If unit test fails : goto Step 5
  - else : continue
- Step 7 : Publish!

Hands-on!

Pseudocode : With uncertainties

- Effectively the same as without uncertainties, with a few minor modifications

Pseudocode : With uncertainties

Compute S, Sx, Sy:

$$S = \sum_{i=0}^{n-1} \frac{1}{\sigma_i^2}\,, \qquad S_x = \sum_{i=0}^{n-1} \frac{x_i}{\sigma_i^2}\,, \qquad S_y = \sum_{i=0}^{n-1} \frac{y_i}{\sigma_i^2}\,.$$

Compute ti and S_tt

$$t_i = \frac{1}{\sigma_i}\left(x_i - \frac{S_x}{S}\right)\,, \qquad S_{tt} = \sum_{i=0}^{n-1} t_i^2\,,$$

Compute a and b, and uncertainties :

$$b = \frac{1}{S_{tt}} \sum_{i=0}^{n-1} \frac{t_i y_i}{\sigma_i}\,, \qquad a = \frac{S_y - S_x b}{S}$$

$$\sigma_a^2 = \frac{1}{S}\left(1 + \frac{S_x^2}{S S_{tt}}\right)\,, \qquad \sigma_b^2 = \frac{1}{S_{tt}}\,.$$

Compute chi-squared :

$$\chi^2(a,b) \equiv \sum_{i=0}^{n-1} \left(\frac{y_i - a - b x_i}{\sigma_i}\right)^2\,.$$

```
input data
n = size of data
if n < 2 :
   print 'Error! Not enough data!'
   return
for i = 0... N-1 :
   if abs( sigma_i ) < 0.00001 :
      return
   S += 1.0 / sigma_i**2
   s_x += x_i / sigma_i**2
   s_y += y_i / sigma_i**2
for i = 0... N-1 :
   t_i = 1.0 / sigma_i * (x_i-s_x/S)
   s_tt = t_i**2
   b += t_i * y_i / sigma_i
if abs( S ) < 0.000001 :
   return
a = (s_y - s_x * b) / S
b = b / s_tt
sigma_a2 = (1 + s_x**2/S*s_tt) / S
sigma_b2 = 1.0 / s_tt
for i = 0... N-1 :
   chi2 += ((y_i - a - b*x_i)/sigma_i)**2
```

Pseudocode : With uncertainties

Compute S, Sx, Sy:

$$S = \sum_{i=0}^{n-1} \frac{1}{\sigma_i^2} , \qquad S_x = \sum_{i=0}^{n-1} \frac{x_i}{\sigma_i^2} , \qquad S_y = \sum_{i=0}^{n-1} \frac{y_i}{\sigma_i^2} .$$

Compute ti and S_tt

$$t_i = \frac{1}{\sigma_i}\left(x_i - \frac{S_x}{S}\right) , \qquad S_{tt} = \sum_{i=0}^{n-1} t_i^2 ,$$

Compute a and b, and uncertainties :

$$b = \frac{1}{S_{tt}} \sum_{i=0}^{n-1} \frac{t_i y_i}{\sigma_i} , \qquad a = \frac{S_y - S_x b}{S}$$

$$\sigma_a^2 = \frac{1}{S}\left(1 + \frac{S_x^2}{S S_{tt}}\right) , \qquad \sigma_b^2 = \frac{1}{S_{tt}} .$$

Compute chi-squared :

$$\chi^2(a,b) \equiv \sum_{i=0}^{n-1} \left(\frac{y_i - a - b x_i}{\sigma_i}\right)^2 .$$

```
input data
n = size of data
if n < 2 :
   print 'Error! Not enough data!'
   return
for i = 0... N-1 :
   if abs( sigma_i ) < 0.00001 :
      return
   S += 1.0 / sigma_i**2
   s_x += x_i / sigma_i**2
   s_y += y_i / sigma_i**2
for
   t_i = 1.0 / sigma_i * (x_i-s_x/S)
   s_tt = t_i**2
   b += t_i * y_i / sigma_i
if abs( S ) < 0.000001 :
   return
a = (s_y - s_x * b) / S
b = b / s_tt
sigma_a2 = (1 + s_x**2/S*s_tt) / S
sigma_b2 = 1.0 / s_tt
for i = 0... N-1 :
   chi2 += ((y_i - a - b*x_i)/sigma_i)**2
```

Where did I mess up here!?!?!

Pseudocode : With uncertainties

Compute S, Sx, Sy:

$$S = \sum_{i=0}^{n-1} \frac{1}{\sigma_i^2} \, , \qquad S_x = \sum_{i=0}^{n-1} \frac{x_i}{\sigma_i^2} \, , \qquad S_y = \sum_{i=0}^{n-1} \frac{y_i}{\sigma_i^2} \, .$$

Compute ti and S_tt

$$t_i = \frac{1}{\sigma_i}\left(x_i - \frac{S_x}{S}\right) \, , \qquad S_{tt} = \sum_{i=0}^{n-1} t_i^2 \, ,$$

Compute a and b, and uncertainties :

$$b = \frac{1}{S_{tt}} \sum_{i=0}^{n-1} \frac{t_i y_i}{\sigma_i} \, , \qquad a = \frac{S_y - S_x b}{S}$$

$$\sigma_a^2 = \frac{1}{S}\left(1 + \frac{S_x^2}{S S_{tt}}\right) \, , \qquad \sigma_b^2 = \frac{1}{S_{tt}} \, .$$

Compute chi-squared :

$$\chi^2(a,b) \equiv \sum_{i=0}^{n-1} \left(\frac{y_i - a - b x_i}{\sigma_i}\right)^2 \, .$$

```
input data
n = size of data
if n < 2 :
   print 'Error! Not enough data!'
   return
for i = 0... N-1 :
   if abs( sigma_i ) < 0.00001 :
      return
   S += 1.0 / sigma_i**2
   s_x += x_i / sigma_i**2
   s_y += y_i / sigma_i**2
for i = 0... N-1 :
   t_i = 1.0 / sigma_i * (x_i-s_x/S)
   s_tt = t_i**2
   b += t_i * y_i / sigma_i
if abs( S ) < 0.000001 :
   return
a = (s_y - s_x * b) / S
b = b / s_tt
sigma_a2 = (1 + s_x**2/S*s_tt) / S
sigma_b2 = 1.0 / s_tt
for i = 0... N-1 :
   chi2 += ((y_i - a - b*x_i)/sigma_i)**2
```

Pseudocode : With uncertainties

Compute S, Sx, Sy:

$$S = \sum_{i=0}^{n-1} \frac{1}{\sigma_i^2}\,, \qquad S_x = \sum_{i=0}^{n-1} \frac{x_i}{\sigma_i^2}\,, \qquad S_y = \sum_{i=0}^{n-1} \frac{y_i}{\sigma_i^2}\,.$$

Compute ti and S_tt

$$t_i = \frac{1}{\sigma_i}\left(x_i - \frac{S_x}{S}\right)\,, \qquad S_{tt} = \sum_{i=0}^{n-1} t_i^2\,,$$

Compute a and b, and uncertainties :

$$b = \frac{1}{S_{tt}} \sum_{i=0}^{n-1} \frac{t_i y_i}{\sigma_i}\,, \qquad a = \frac{S_y - S_x b}{S}$$

$$\sigma_a^2 = \frac{1}{S}\left(1 + \frac{S_x^2}{S S_{tt}}\right)\,, \qquad \sigma_b^2 = \frac{1}{S_{tt}}\,.$$

Compute chi-squared :

$$\chi^2(a,b) \equiv \sum_{i=0}^{n-1} \left(\frac{y_i - a - b x_i}{\sigma_i}\right)^2\,.$$

```
input data
n = size of data
if n < 2 :
   print 'Error! Not enough data!'
   return
for i = 0... N-1 :
   if abs( sigma_i ) < 0.00001 :
      return
   S += 1.0 / sigma_i**2
   s_x += x_i / sigma_i**2
   s_y += y_i / sigma_i**2
for i = 0... N-1 :
   t_i = 1.0 / sigma_i * (x_i-s_x/S)
   s_tt = t_i**2
   b += t_i * y_i / sigma_i
if abs( S ) < 0.000001 :
   return
a = (s_y - s_x * b) / S
b = b / s_tt
sigma_a2 = (1 + s_x**2/S*s_tt) / S
sigma_b2 = 1.0 / s_tt
for i = 0... N-1 :
   chi2 += ((y_i - a - b*x_i)/sigma_i)**2
```

Unlikely to happen, but good to be paranoid anyway!

Write code : With uncertainties

Pseudocode → python

```
input data
n = size of data
if n < 2 :
   print 'Error! Not enough data!'
   return
for i = 0... N-1 :
   if abs( sigma_i ) < 0.00001 :
      return
   S += 1.0 / sigma_i**2
   s_x += x_i / sigma_i**2
   s_y += y_i / sigma_i**2
for i = 0... N-1 :
   t_i = 1.0 / sigma_i * (x_i-s_x/S)
   s_tt = t_i**2
   b += t_i * y_i / sigma_i
if abs( S ) < 0.000001 :
   return
a = (s_y - s_x * b) / S
b = b / s_tt
sigma_a2 = (1 + s_x**2/S*s_tt) / S
sigma_b2 = 1.0 / s_tt
for i = 0... N-1 :
   chi2 += ((y_i - a - b*x_i)/sigma_i)**2
```

```python
import numpy as np

def chi_square_fit(x, y, err):
    n = len(x)
    if n < 2 :
        print ('Error! Need at least 2 data points!')
        exit()
    S = np.sum(1/err**2)
    if abs(S) < 0.00001 :
        print ('Error! Denominator S is too small!')
        exit()
    S_x = np.sum(x/err**2)
    S_y = np.sum(y/err**2)
    t = (x - S_x/S) / err
    S_tt = np.sum(t**2)
    if abs(S_tt) < 0.00001 :
        print ('Error! Denominator S is too small!')
        exit()
    b = np.sum(t*y/err) / S_tt
    a = (S_y - S_x * b) / S
    sigma_a2 = (1 + S_x**2/S/S_tt) / S
    sigma_b2 = 1/S_tt
    if sigma_a2 < 0.0 or sigma_b2 < 0.0 :
        print ('Error! About to pass a negative to sqrt')
        exit()
    sigma_a = np.sqrt(sigma_a2)
    sigma_b = np.sqrt(sigma_b2)
    chi_square = np.sum(((y - a - b*x) / err)**2)
    return(a, b, sigma_a, sigma_b, chi_square)
```

Write code : With uncertainties

```
void chi_square_fit(          // makes a linear chi-square fit
  const vector<double>& x,    // vector of x values - input
  const vector<double>& y,    // vector of y values - input
  const vector<double>& err,  // vector of y error values - input
  double& a,                  // fitted intercept - output
  double& b,                  // fitted slope - output
  double& sigma_a,            // estimated error in intercept - output
  double& sigma_b,            // estimated error in slope - output
  double& chi_square)         // minimized value of chi-square sum - output
{
  int n = x.size();

  assert(n >= 2);

  double S = 0, S_x = 0, S_y = 0;
  for (int i = 0; i < n; i++) {
    assert ( fabs(err[i]) >= 0.000001 );
    S += 1 / err[i] / err[i];
    S_x += x[i] / err[i] / err[i];
    S_y += y[i] / err[i] / err[i];
  }

  vector<double> t(n);
  for (int i = 0; i < n; i++)
    t[i] = (x[i] - S_x/S) / err[i];

  double S_tt = 0;
  for (int i = 0; i < n; i++)
    S_tt += t[i] * t[i];

  b = 0;
  for (int i = 0; i < n; i++)
    b += t[i] * y[i] / err[i];
  assert( fabs(S_tt) > 0.00001);
  b /= S_tt;

  assert( fabs(S) > 0.00001);
  a = (S_y - S_x * b) / S;
  sigma_a = sqrt((1 + S_x * S_x / S / S_tt) / S);
  sigma_b = sqrt(1 / S_tt);

  chi_square = 0;
  for (int i = 0; i < n; i++) {
    double diff = (y[i] - a - b * x[i]) / err[i];
    chi_square += diff * diff;
  }
}
```

C++ tip : assert yourself!

- If you have a condition that your algorithm must fulfill, you can use a few C++ mechanisms to handle this.
- Python handles exceptions on its own.
- C++ behavior there is undefined, and compiler dependent!
- So, you can use a simple “assert(condition)” to make sure it’s true
- Alternatively you can use exception handling but that can get a little hairy. I won’t cover it in this class much (if at all).

Hands-on!

Recall : Development

- Step 1 : Write the algorithm down on paper
- Step 2 :
  - If you don’t understand everything : goto step 1
  - else : continue
- Step 3 : write pseudocode
- Step 4 : continue
  - If you don’t understand everything : goto step 3
  - else : continue
- Step 5 : write code
- Step 6 : check code with unit tests
  - Check “pass” criterion
  - Check “fail” criterion
  - If unit test fails : goto Step 5
  - else : continue
- Step 7 : Publish!

Recall : Development

- Step 1 : Write the algorithm down on paper
- Step 2 :
  - If you don’t understand everything : goto step 1
  - else : continue
- Step 3 : write pseudocode
- Step 4 : continue
  - If you don’t understand everything : goto step 3
  - else : continue
- Step 5 : write code
- Step 6 : check code with unit tests
  - Check “pass” criterion
  - Check “fail” criterion
  - If unit test fails : goto Step 5
  - else : continue
- Step 7 : Publish!

Recall : Development

- Step 1 : Write the algorithm down on paper
- Step 2 :
  - If you don’t understand everything : goto step 1
  - else : continue
- Step 3 : write pseudocode
- Step 4 : continue
  - If you don’t understand everything : goto step 3
  - else : continue
- Step 5 : write code
- Step 6 : check code with unit tests
  - Check “pass” criterion
  - Check “fail” criterion
  - If unit test fails : goto Step 5
  - else : continue
- Step 7 : Publish!

Fitting curves

- What if we want to fit something besides a line?

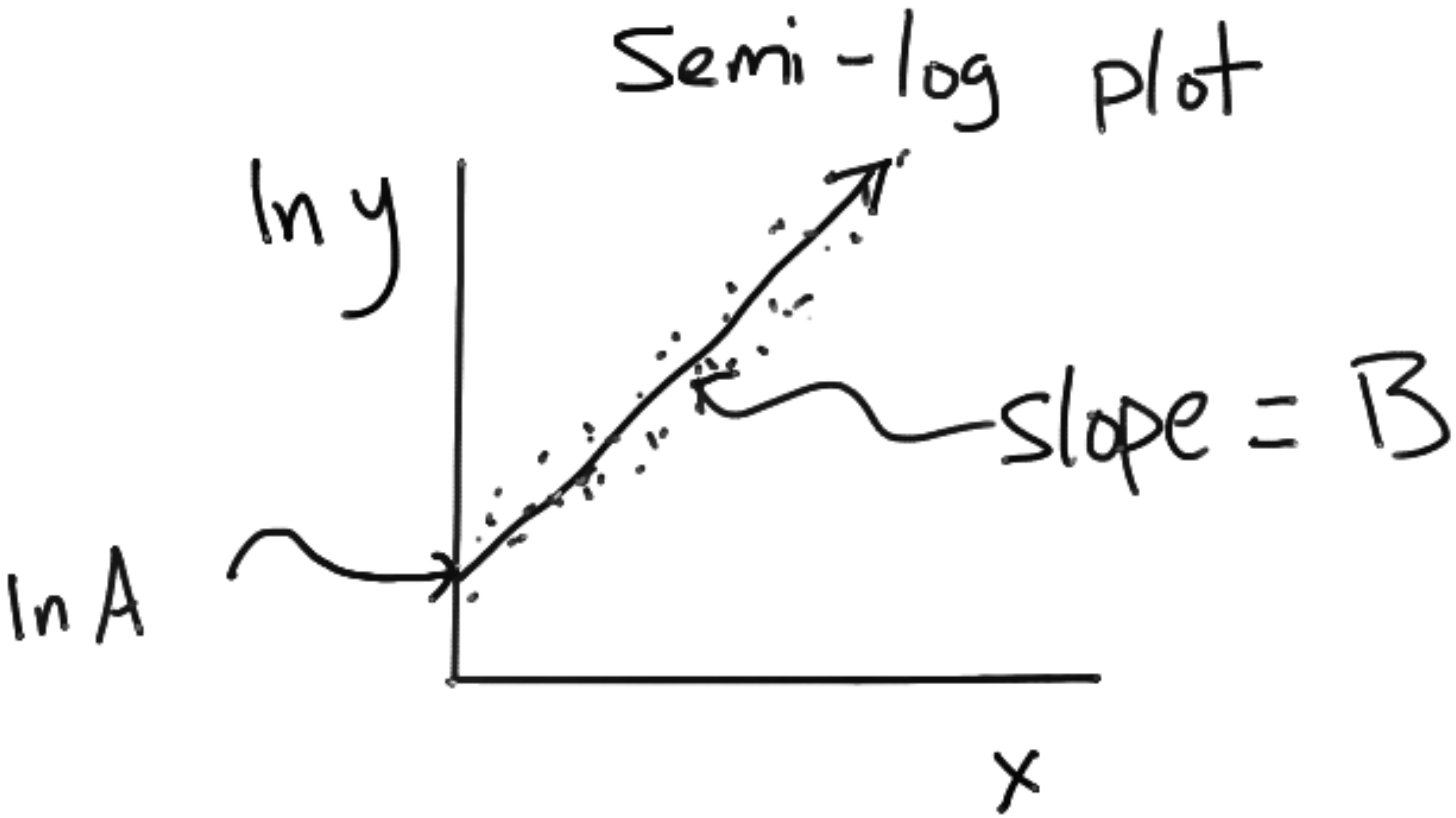
- Well, there are a few cases :
  1. Does change of variables in x and y make it a line?
  2. Everything else

- In the first case, it’s actually easiest to just fit a line again
  - In fact, you’re technically doing this in your homework example!

- In the second case, it’s also not much harder conceptually, but is more computationally intensive

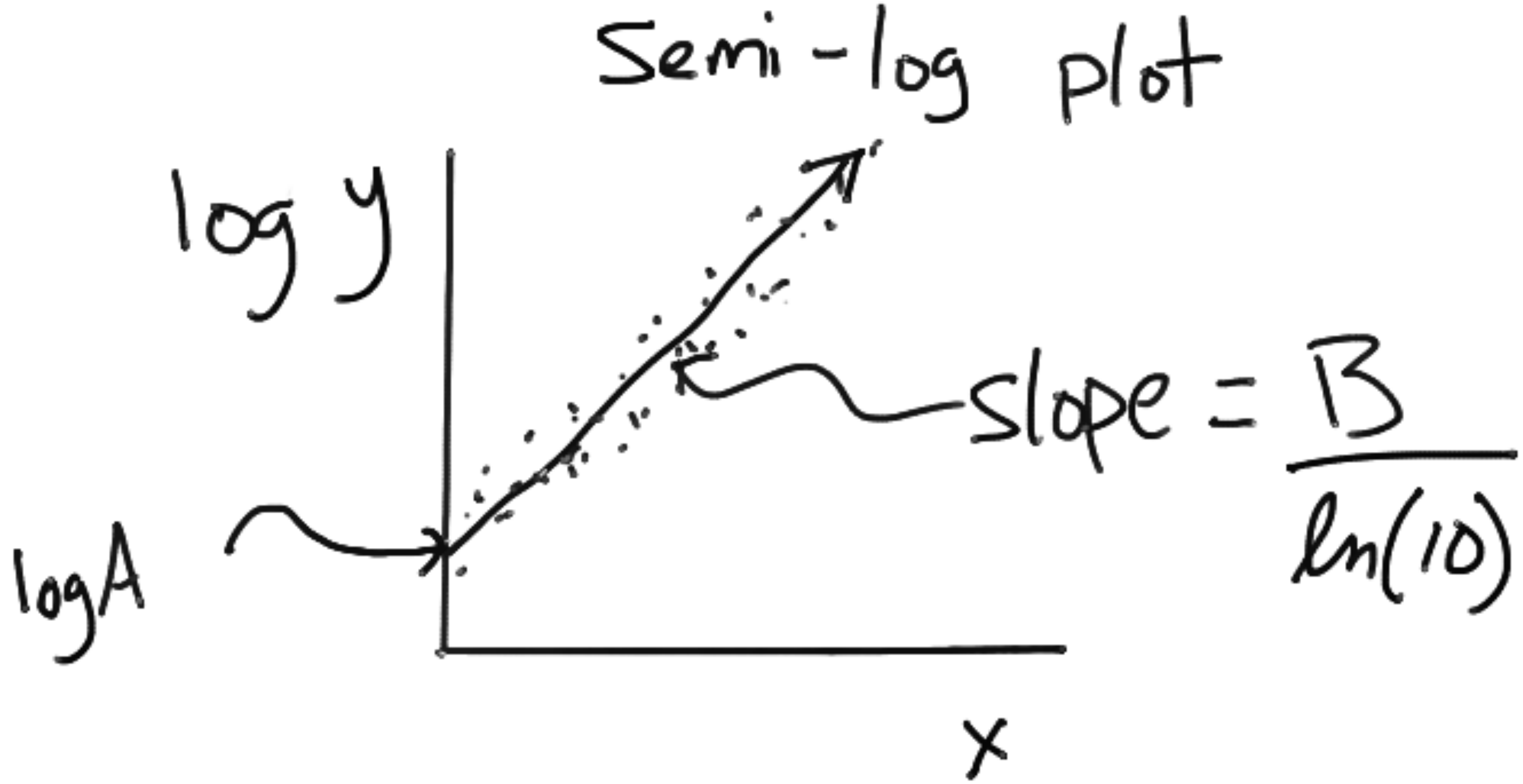
First case : transform to linear

- Say we have a model like $y = A\,e^{Bx}$
- What do we do?
- Take the logarithm of both sides! $\ln y = \ln A + Bx$



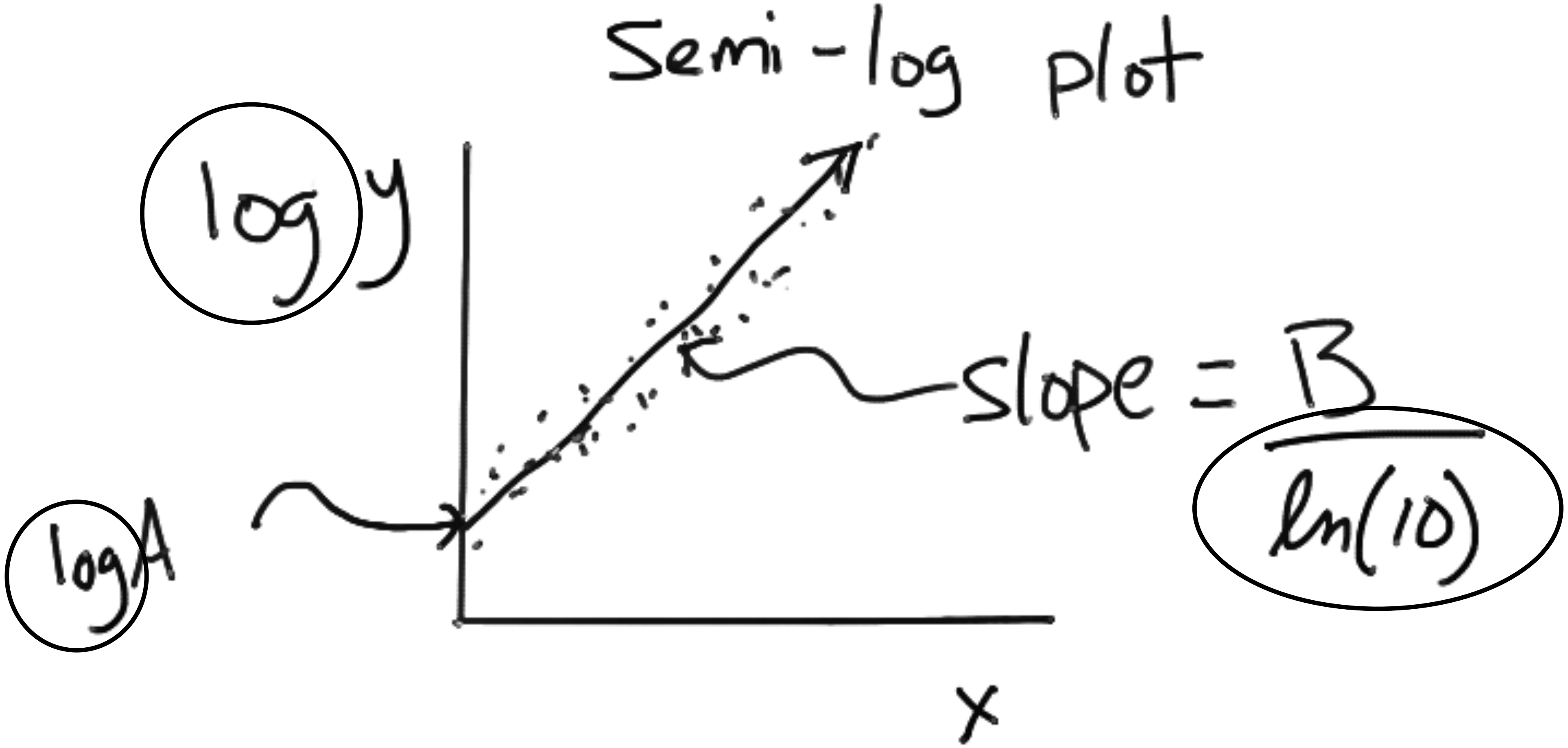
First case : transform to linear

- OK, but what if I do a base-10 logarithm instead of natural?
- Not a problem, just transform! $\log_b(x) = \dfrac{\log_d(x)}{\log_d(b)}$
- So in this case :



First case : transform to linear

- OK, but what if I do a base-10 logarithm instead of natural?
- Not a problem, just transform! $\log_b(x) = \dfrac{\log_d(x)}{\log_d(b)}$
- So in this case :

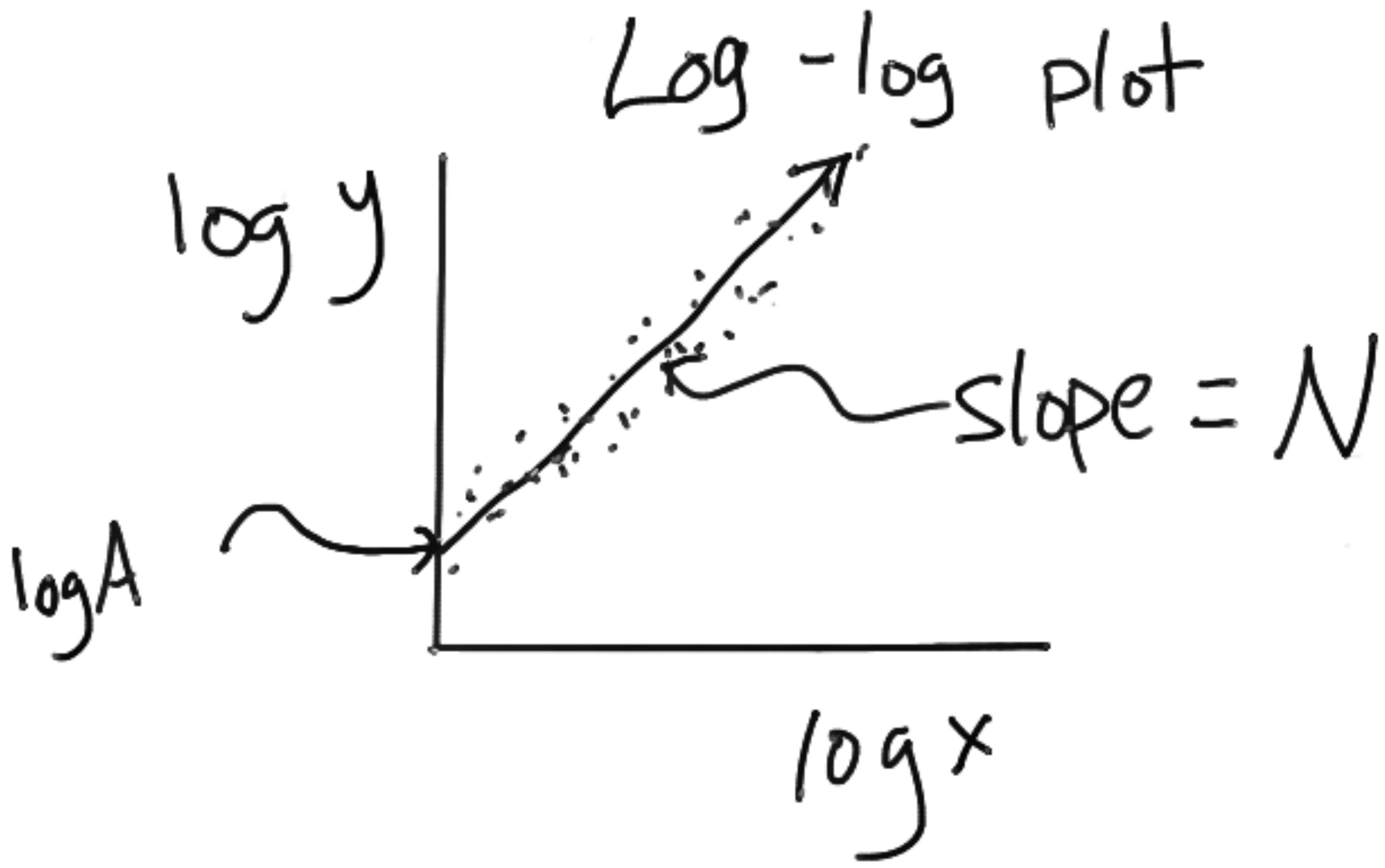


First case : transform to linear

- Now try $y = A\, x^N$

- Take the logarithm of both sides :

$$\log y = \log A + N \log x$$



First case : transform to linear

- What about the uncertainties?
- We did a change of variables :

$$y' = f(y)$$

- So, propagating the uncertainties, we get :

$$\sigma_{y'}^2 = \left(\frac{\partial f}{\partial y}\right)^2 \sigma_y^2$$

- So you just have to remember this in the chi-squared minimization
- Example :

$$y' = \ln y = \ln A + Bx \quad \Longrightarrow \quad \sigma_{y'} = \frac{\sigma_y}{|y|}$$

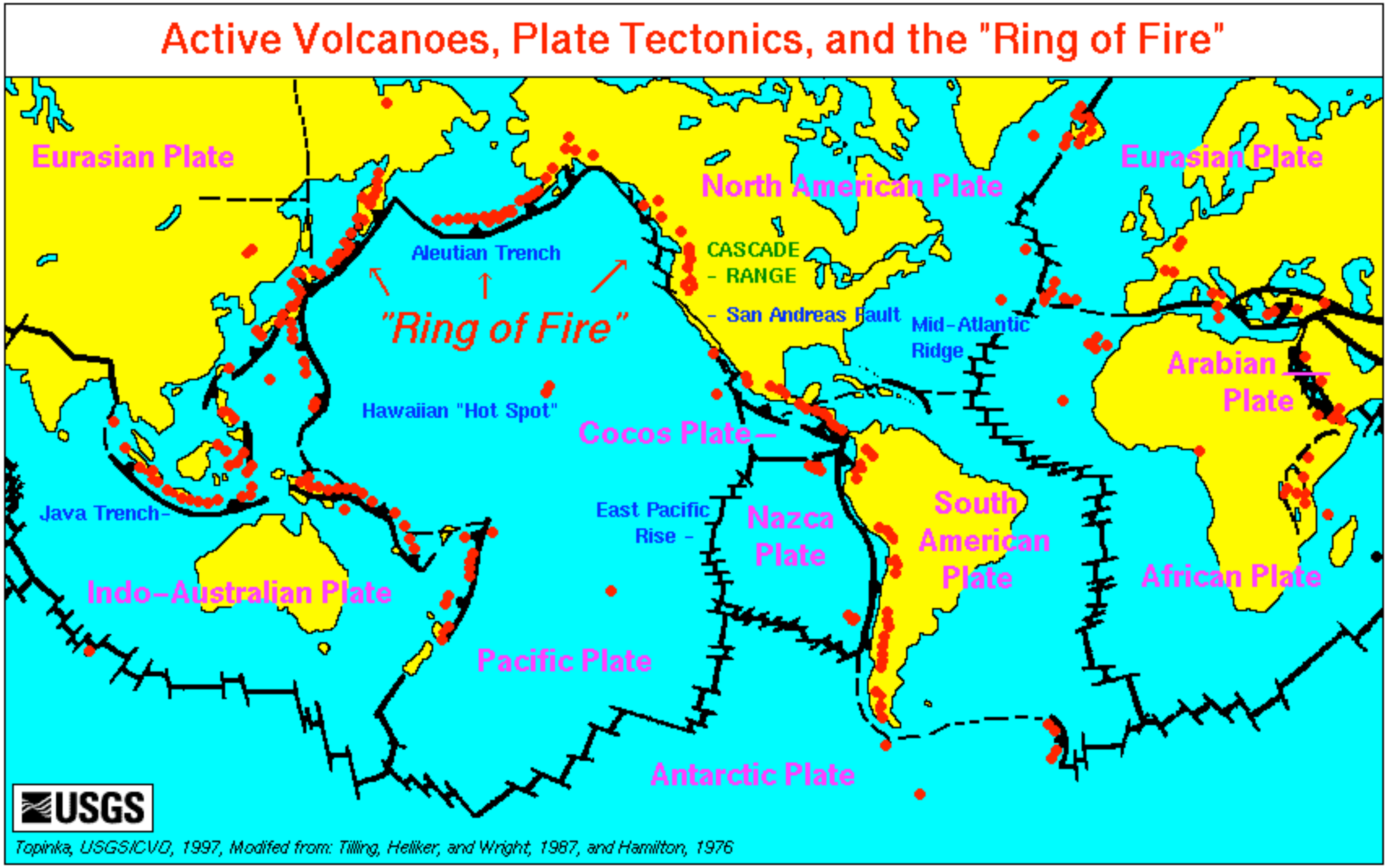
(Note : In Assignment 1 you're already given $\sigma_{y'}$ so you don't have to worry!

First case : transform to linear

- Modulo that, it's already a "solved problem"
- You should be doing this in your homeworks already!

Earthquakes

- Earthquakes occur when tectonic plates of the earth move relative to one another

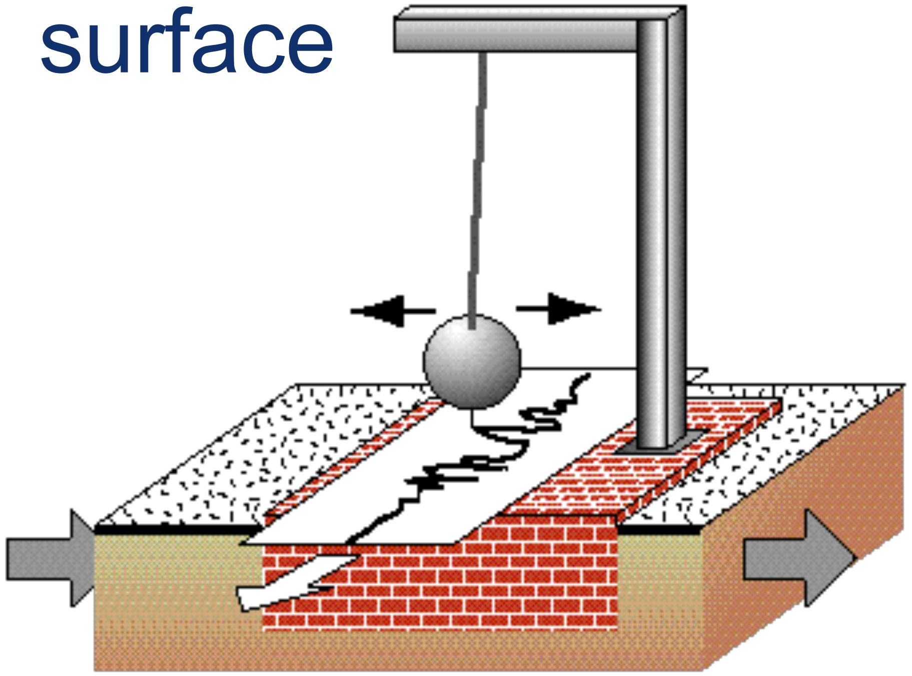


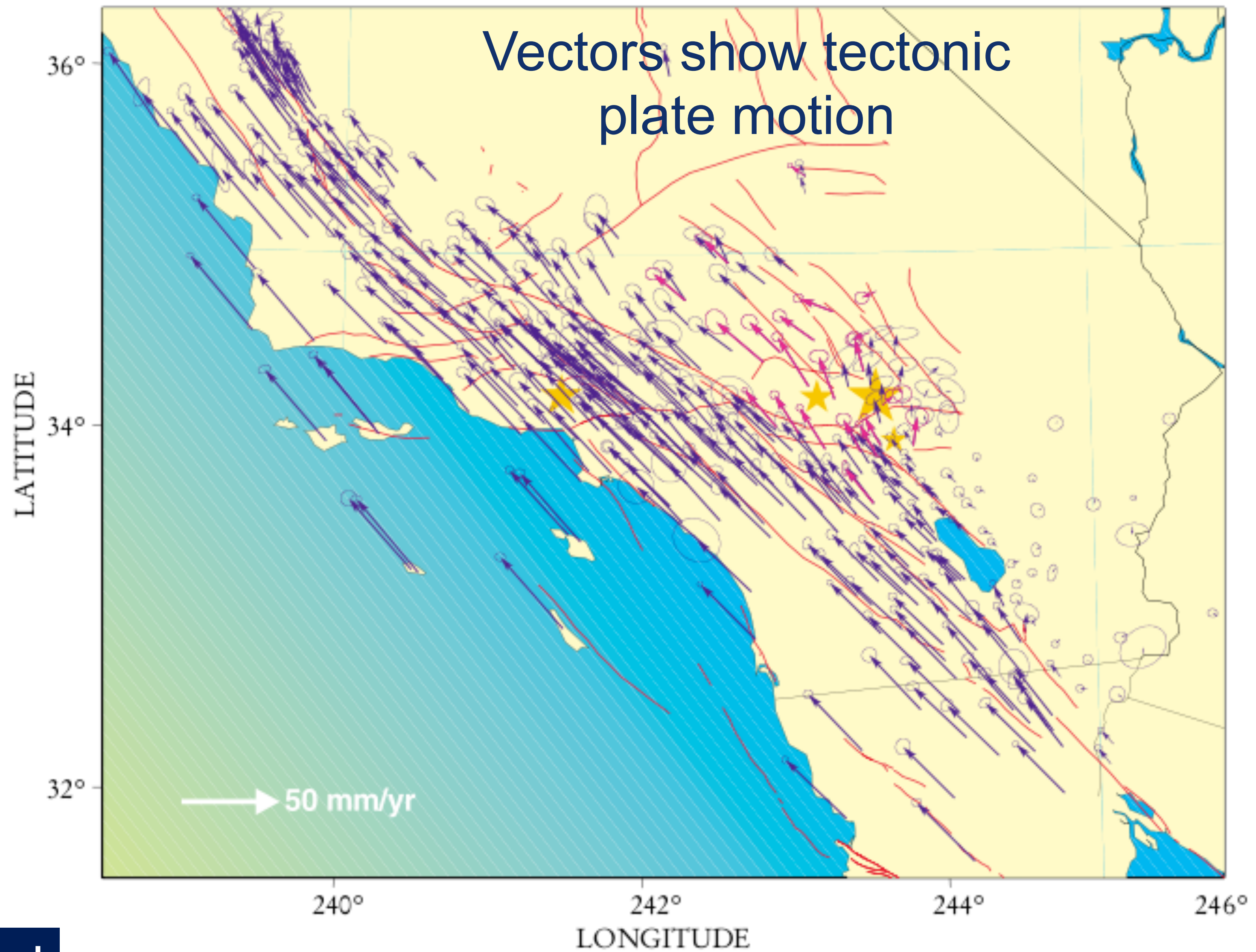
Earthquakes

- Earthquakes occur when tectonic plates of the earth move relative to one another

Earthquakes

- When they rub against each other, can get stuck!
- Builds pressure, then slips, releasing a lot of energy
- Seismographs can measure the vibrations on the surface





Magnitude is related to the squared amplitude over the event!

H. Kanamori and E.E. Brodsky, The Physics of Earthquakes, Physics Today **54**, 34-40 (2001)
http://www.colorado.edu/physics/phys2900/homepages/Marianne.Hogan/graphs.html

Earthquakes

- The “Richter scale” was developed by Richter in the 1930’s
- Relates the LOCAL magnitude scale M_L
  - Defined by the amount of amplitude variation on a seismograph
- Replaced in the 1970’s by the MOMENT magnitude scale

$$M_0 = \mu S D$$

rigidity (stiffness module)

area of fault

displacement of the plate

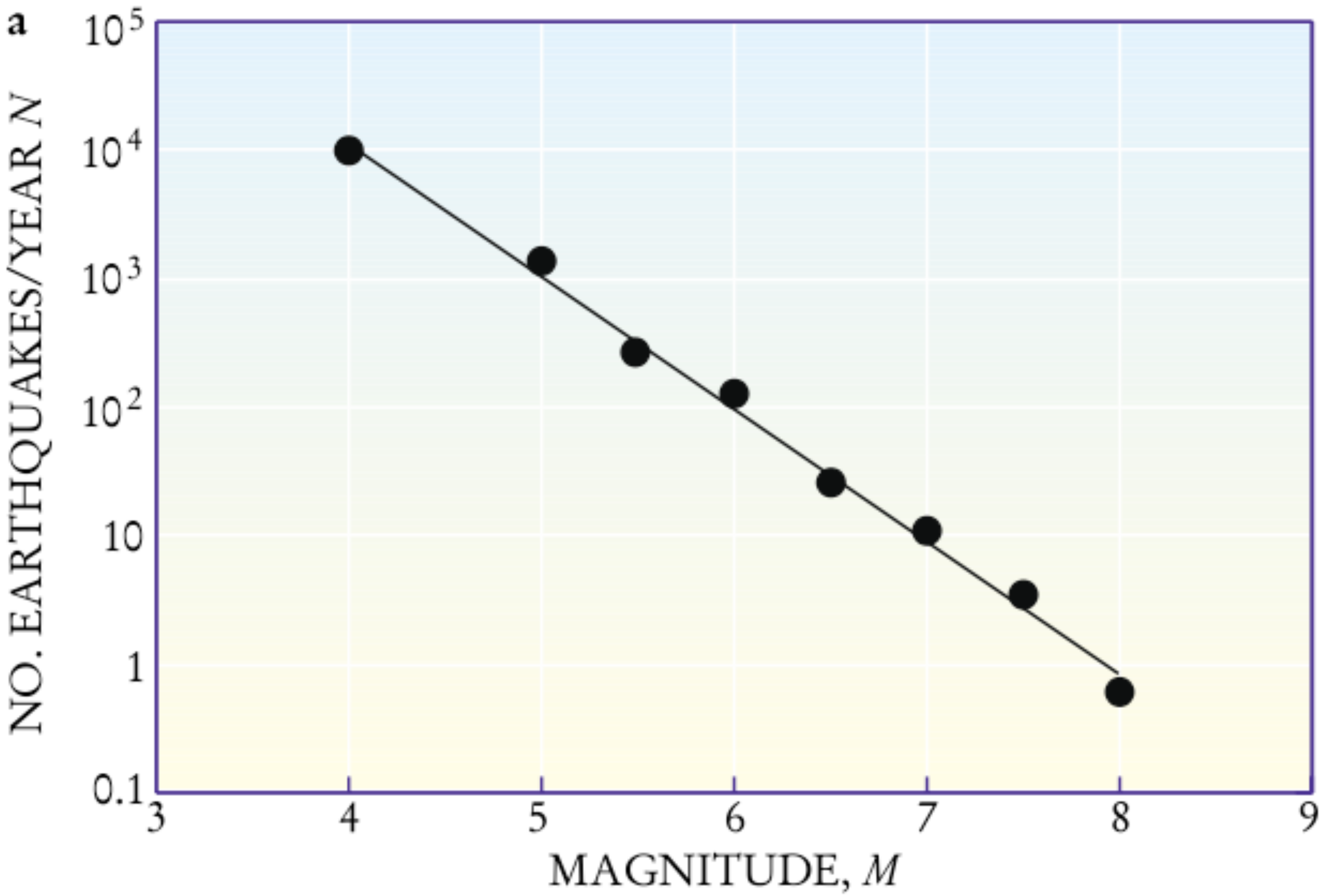
- Gutenberg-Richter ~~Law~~ Model :
  - Frequency (N) of earthquakes of magnitude (M) : defined as number of events with magnitude >= M
  - Empirical model :

$$\log N = a - bM$$

$$M \sim \log M_0$$

Earthquakes

- Frequency vs magnitude plot of earthquakes between 1904 and 2000 :



H. Kanamori and E.E. Brodsky, The Physics of Earthquakes, Physics Today **54**, 34-40 (2001)

Earthquakes

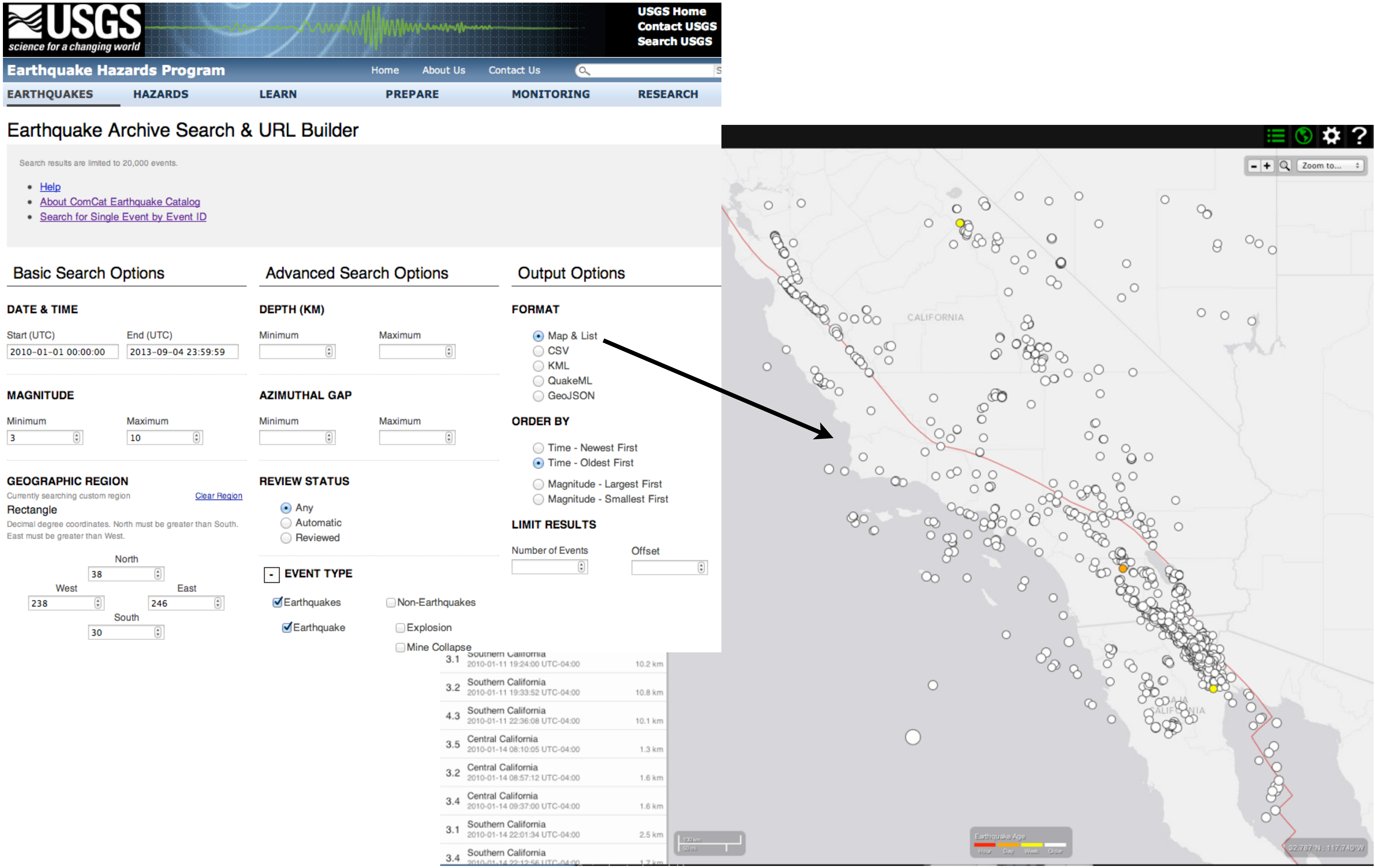
- Get data from :
  - http://earthquake.usgs.gov/earthquakes/eqarchives/epic/

- Many formats for the data file :
  - Map & List
  - CSV (comma-separated values)
  - KML (google-based geographical data representation)
  - QuakeML (XML for earthquakes)
  - GeoJSON (JSON for earthquakes)
- We'll go for CSV :


```
:time,latitude,longitude,depth,mag,magType,nst,gap,dmin,rms,net,id,updated,place,type
2010-01-01T02:33:42.590Z,32.476,-115.19,1.5,3.2,ml,22,175.8,,,pde,pde20100101023342590_1,2013-03-16T01:48:06.208Z,"Baja Californ
2010-01-01T02:55:04.280Z,35.979,-117.321,0.8,2.8,ml,17,51.2,,,pde,pde20100101025504280_0,2013-03-16T01:48:06.336Z,"Central Calif
2010-01-01T03:25:29.970Z,36.031,-117.784,3.2,2.9,ml,13,50.8,,,pde,pde20100101032529970_3,2013-03-16T01:48:06.354Z,"Central Calif
2010-01-01T14:06:45.100Z,32.474,-115.215,8,3.1,ml,7,176.3,,,pde,pde20100101140645100_8,2013-03-16T01:48:06.753Z,"Baja California
2010-01-02T03:15:45.200Z,33.576,-118.889,11.4,2.8,ml,7,252.5,,,pde,pde20100102031545200_11,2013-03-16T01:48:07.298Z,"Channel Isl
```

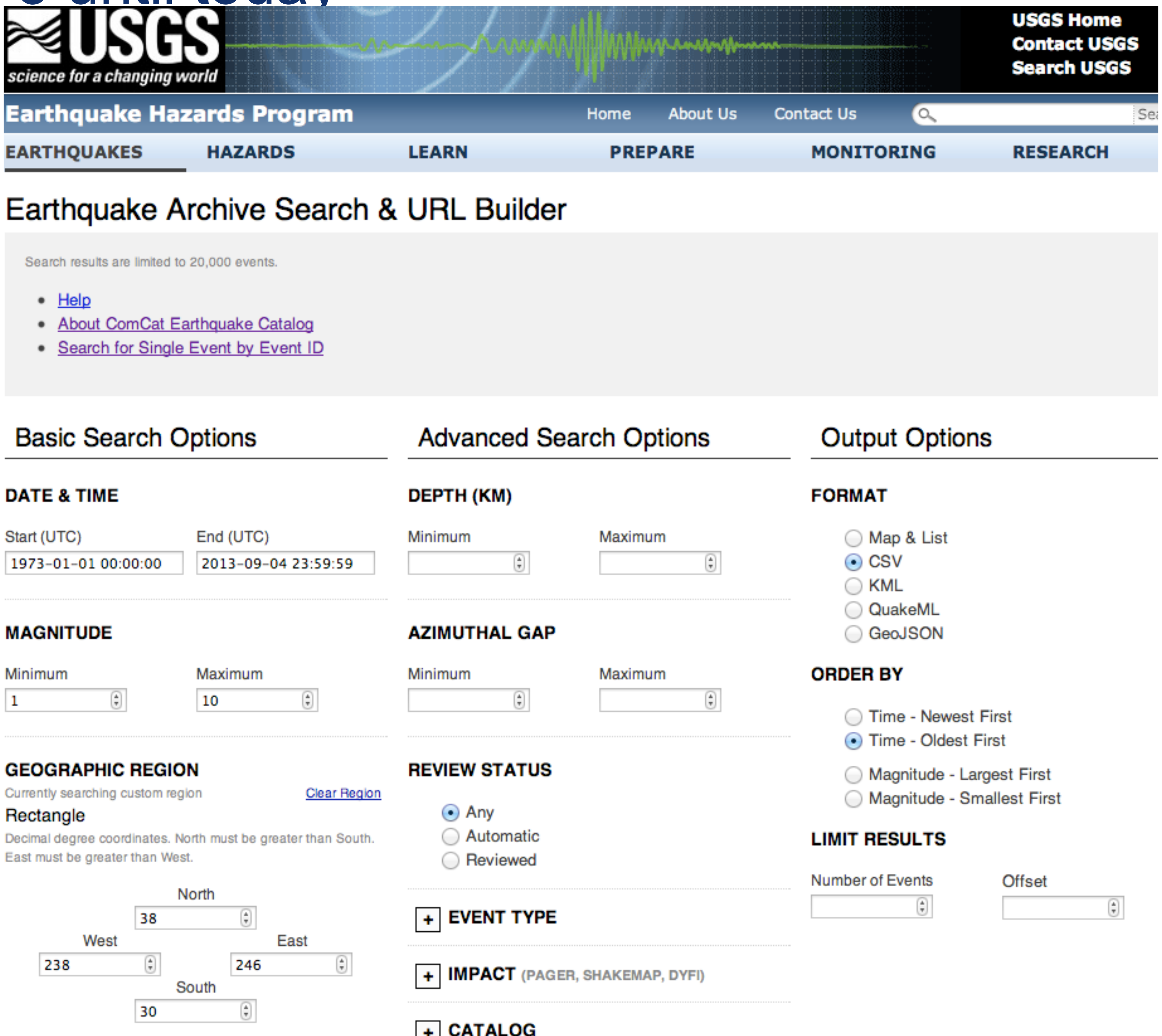
Earthquakes

- Example :



Earthquakes

- The data we'll fit : all earthquakes in southern California from 1973 until today



The big picture

- Read in earthquake data

```
open file
get lines
if error : exit
else :
   for line in lines :
      parse value
      record number, magnitude
```

- Compute N(>= M)

```
N(M) = number with magnitude >= M
```

- Transform to linear form

```
for each number,magnitude pair :
   y = log(number)
   x = magnitude
```

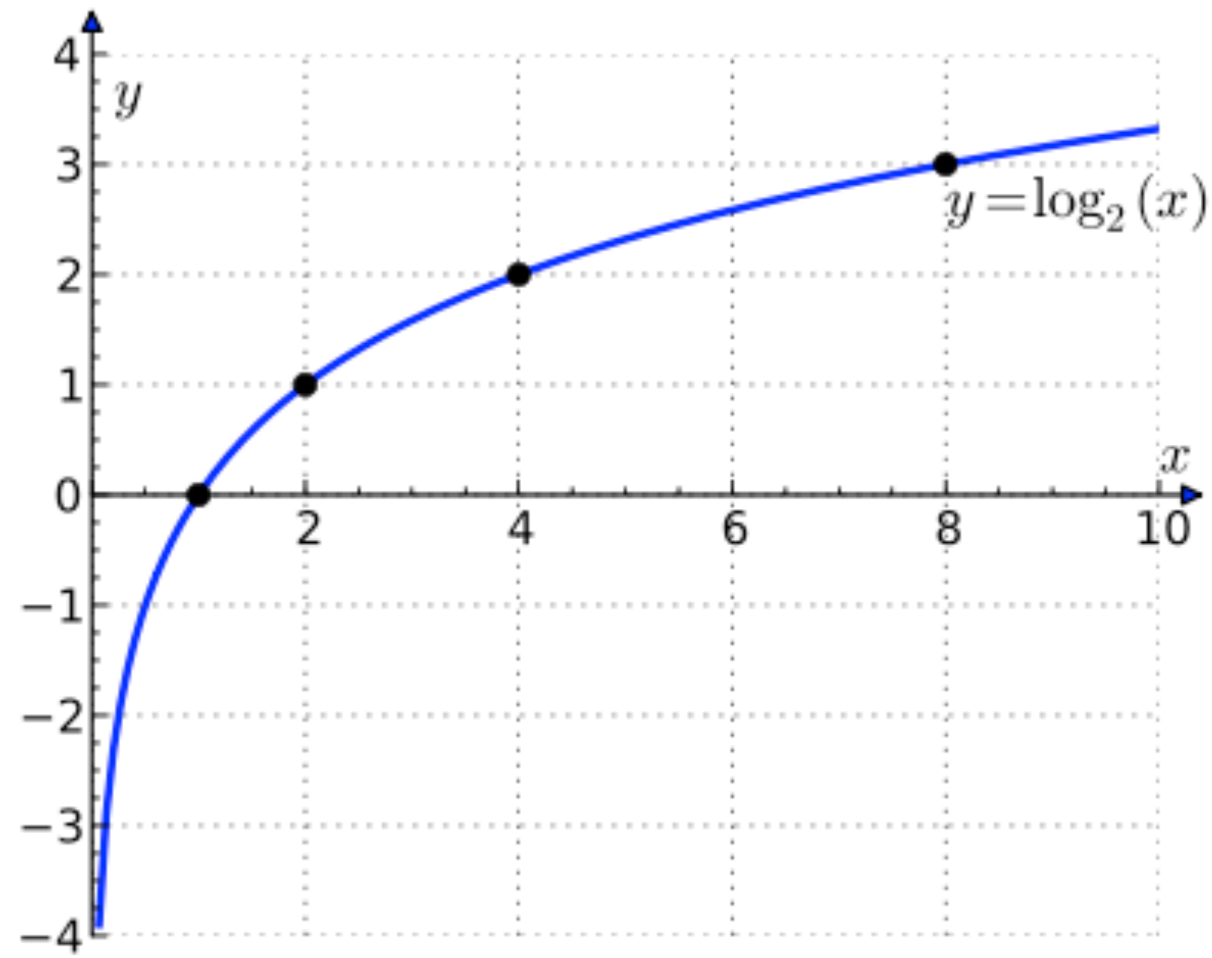
- Fit via least-squares

```
input data
n = size of data
if n < 2 :
   print 'Error! Not enough data!'
   return
for i = 0... N-1 :
   s_x += x_i
   s_y += y_i
   s_xx += x_i**2
   s_xy += x_i*y_i
den = n * s_xx - s_x*s_x
if abs( den ) < 0.000001 :
   print 'Error! Denominator is zero!'
   return
a = (s_xx * s_y - s_x * s_xy) / den
b = (n*s_xy - s_x * s_y) / den
for i = 0... N-1 :
  sigma2 += (y_i - (a*x_i+b))**2
sigma2 = sigma2 / (n-2)
```

Numerical issues : logarithms

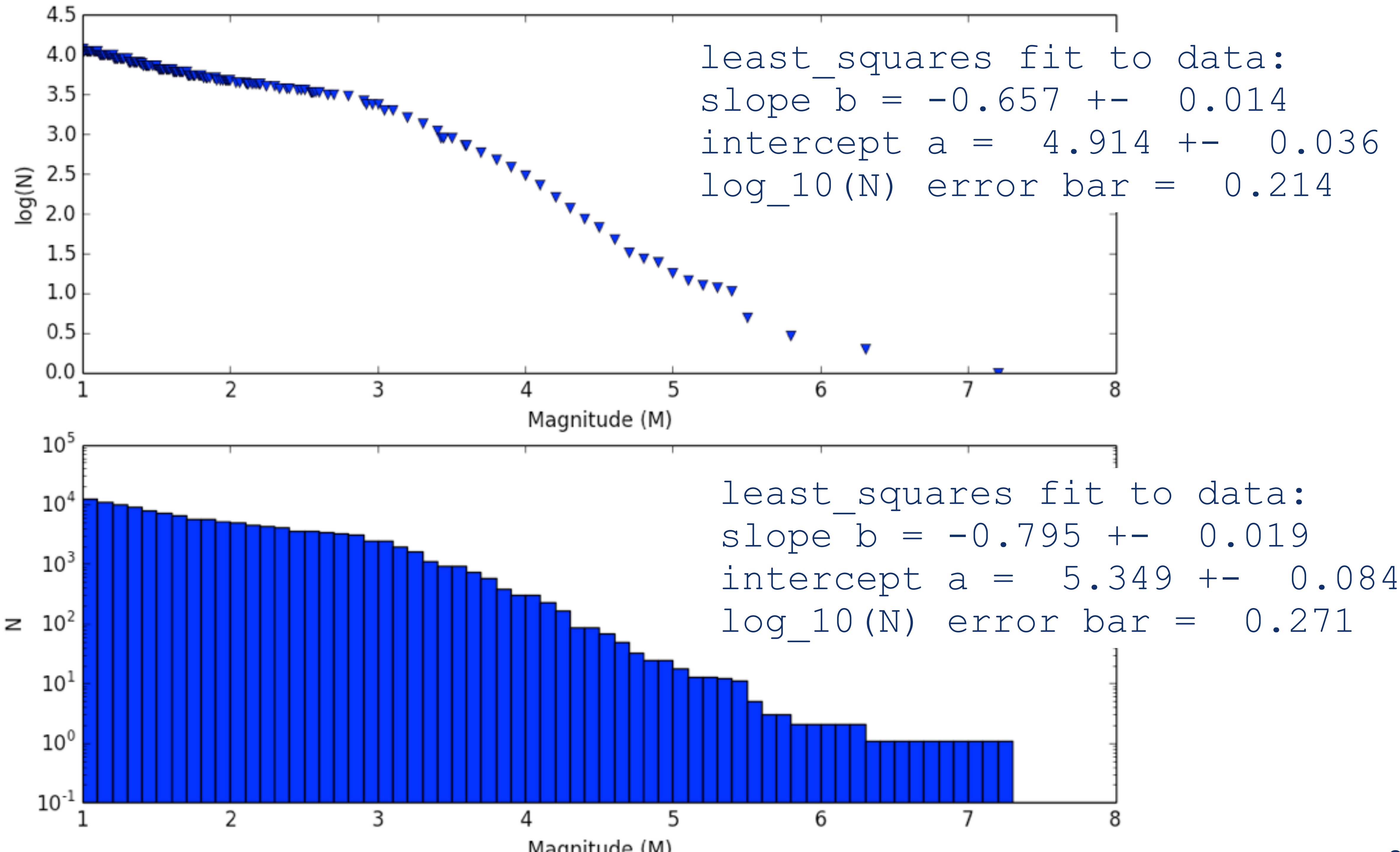
- Logarithms cannot be <= 0
- Always need to check this!
- Other than that, very nice because it transforms multiplication into addition!

$$\log x \times y = \log x + \log y$$



Binning!

- Be sure to be careful about fitting binned data!



General fitting of curves

- For general curve-fitting, it’s not conceptually more difficult

- However, it is computationally more difficult

General fitting of curves

- Define the function and its parameters as :

$$\{a, b\} \rightarrow \vec{a} \qquad\qquad y = y(x; \vec{a})$$

"y is a function of x, with parameters a"

- Rewrite our chi-squred expression :

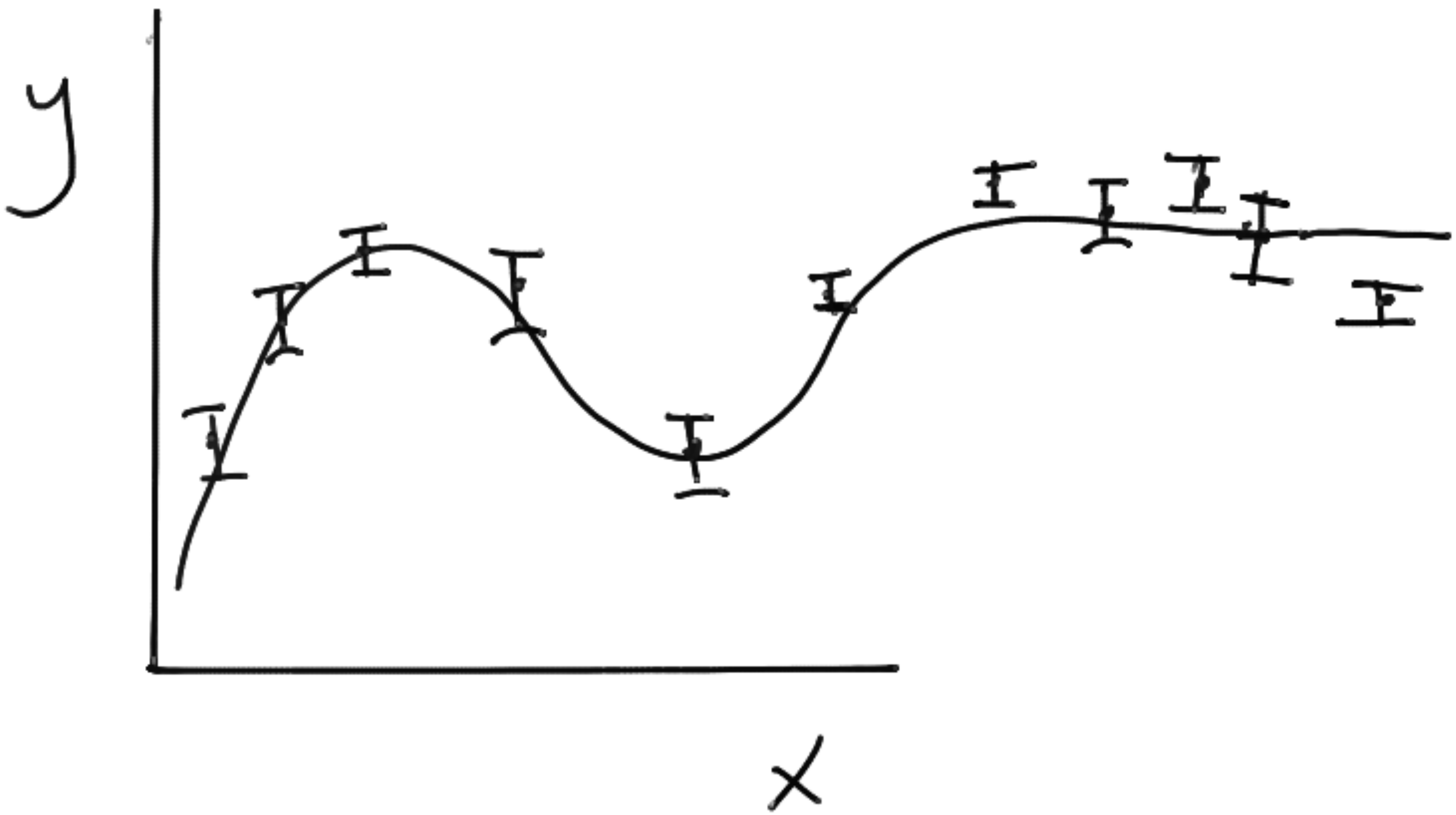
$$\chi^2(\vec{a}) \equiv \sum_{i=0}^{n-1} \left(\frac{y_i - y(x; \vec{a})}{\sigma_i} \right)^2$$

- Now this actually should be obvious!

- This completely generalizes to nonlinear y(x)

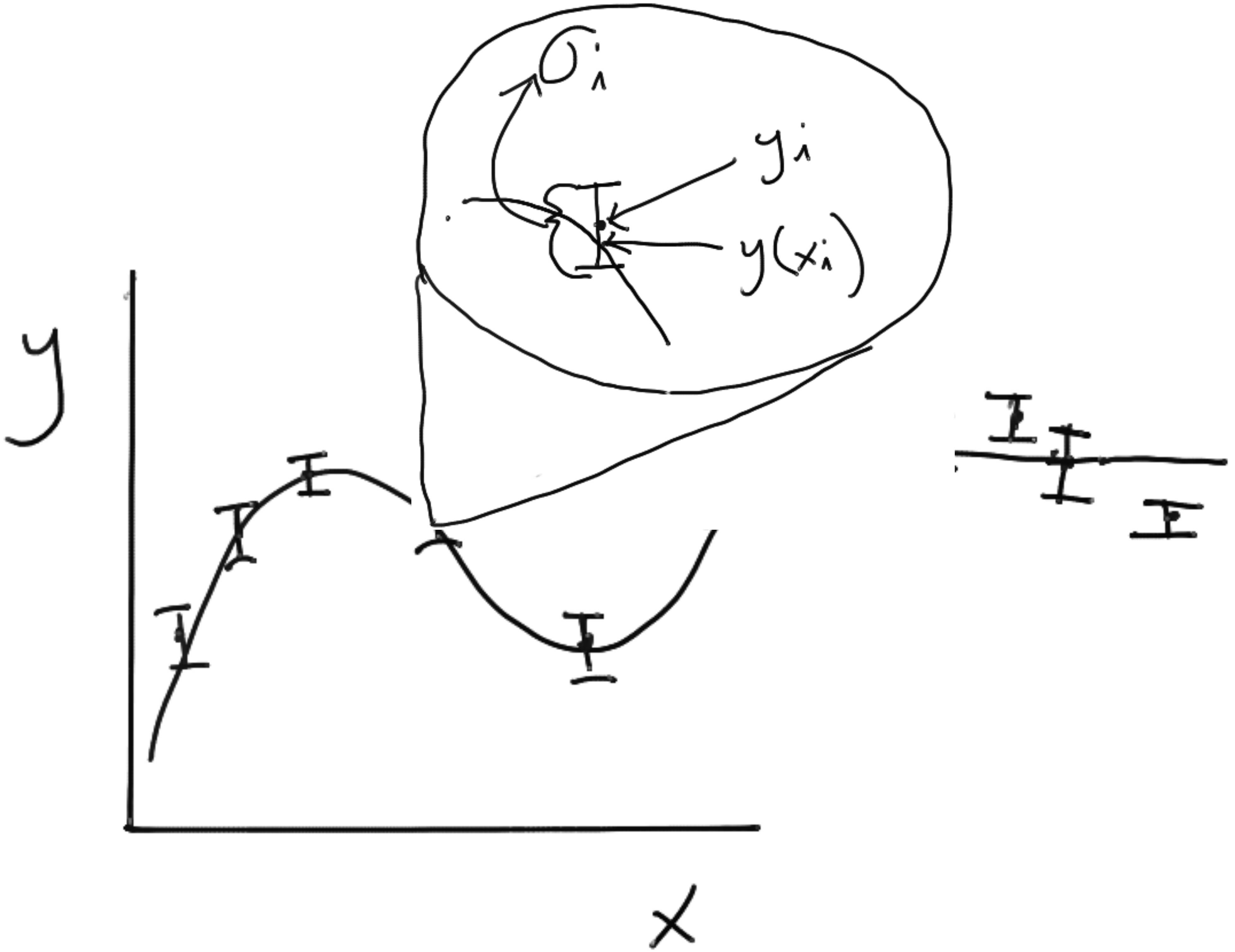
General fitting of curves

- Still just minimizing the distance within the uncertainties



General fitting of curves

- Still just minimizing the distance within the uncertainties



General fitting of curves

- Same strategy as before : minimize the chi2!

$$\chi^2(\vec{a}) \equiv \sum_{i=0}^{n-1} \left(\frac{y_i - y(x;\vec{a})}{\sigma_i} \right)^2$$

- So, let's say that y(x) is some expansion of functions $Y_k(x)$:

$$y(x;\vec{a}) = \sum_{k=0}^{m-1} a_k Y_k(x)$$

- Then the chi2 is :

$$\chi^2(x,\vec{a}) = \sum_{i=0}^{n-1} \frac{1}{\sigma_i^2} \left[y_i - \sum_{k=0}^{m-1} a_k Y_k(x) \right]^2$$

General fitting of curves

- We minimize :

$$\frac{\partial \chi^2}{\partial a_j} = \frac{\partial}{\partial a_j} \sum_{i=0}^{n-1} \frac{1}{\sigma_i^2} \left[y_i - \sum_{k=0}^{m-1} a_k Y_k(x) \right]^2 = 0$$

- Taking the derivative :

$$2a_j \sum_{i=0}^{n-1} \frac{1}{\sigma_i^2} Y_j(x) \left[y_i - \sum_{k=0}^{m-1} a_k Y_k(x) \right] = 0$$

- The 2*a_j cancels. Then we multiply the sum through, and bring over the second term, so we get :

$$\sum_{i=0}^{n-1} \sum_{k=0}^{m-1} \frac{Y_j(x_i) Y_k(x_i)}{\sigma_i^2} a_k = \sum_{i=0}^{n-1} \frac{Y_j(x_i) y_i}{\sigma_i^2}$$

General fitting of curves

- This is a matrix equation, so we define the “design matrix” :

$$A_{ij} = \frac{Y_j(x_i)}{\sigma_i}$$

$$\mathbf{A} = \left[\begin{array}{ccc} Y_1(x_1)/\sigma_1 & Y_2(x_1)/\sigma_1 & \dots \\ Y_1(x_2)/\sigma_2 & Y_2(x_2)/\sigma_2 & \dots \\ \dots & \dots & \dots \end{array} \right]$$

- Then our chi2 minimization becomes :

$$(\mathbf{A}^T \mathbf{A})\vec{a} = \mathbf{A}^T \vec{b}$$

- so :

$$\vec{a} = (\mathbf{A}^T \mathbf{A})^{-1} \mathbf{A}^T \vec{b}$$

General fitting of curves

- If we define the “correlation matrix” :

$$\mathbf{C} = (\mathbf{A}^T\mathbf{A})^{-1}$$

- Then the uncertainty on a_j is :

$$\sigma_{a_j} = \sqrt{C_{jj}}$$

General fitting of curves

- As a first example, let's look at polynomial fits

$$y = \sum_{k=0}^{m-1} a_k x^k \ .$$

- Slight generalization of the linear fit we did previously
- General solution is to minimize the chi2 :

$$\chi^2(\vec{a}) \equiv \sum_{i=0}^{n-1} \left(\frac{y_i - y(x;\vec{a})}{\sigma_i} \right)^2$$

- In this case :

$$\chi^2(\vec{a}) = \sum_{i=0}^{n-1} \left(\frac{y_i - \sum_{j=0}^{M} a_j x^j}{\sigma_i} \right)^2$$

General fitting of curves

- Our design matrix is therefore :

$$A_{ij} = x_i^j / \sigma_i$$

- Caveat : This oftentimes is ill-formed, so don't go too crazy here. Typically we do quadratic, cubic, quartic, but above that it strains credibility.

General Fitting of Curves

- Will return to this after we do some linear algebra!