Stochastic Methods (Chapter 11)

In Chapter 5 we briefly discussed various techniques for determining the 'uncertainties' in calculations, including:

_______________method for deriving uncertainty: Generate *N* synthetic versions of the original data set. Each synthetic version may incorporate random tweaks to individual data values via their quoted uncertainties, or may randomly (sub)sample if the uncertainties are unknown or suspect. Carry out the computation(s) for each of the *N* iterations—the dispersion in the *N* values is the uncertainty.

Example

Find the mean of 15 numbers, and compute a confidence interval for the mean: *bootstrap.f and MonteCarlo.f.*

More generally, *Monte Carlo* techniques[☆] are utilized for solving physical and mathematical problems that don't necessarily lend themselves to easy analytic solutions (multi-dimensional integrals, data sampling). They are used for simulating systems with many degrees of freedom (e.g., ___________________ in physics). Homework #11.22 requires Monte Carlo methods to solve for the pressure of a gas with a Maxwell-Boltzmann velocity distribution.

A **general algorithm**[†] looks like

    Define the possible range for inputs
    Generate random values of inputs
    Execute the desired calculations
    Repeat many times to get a statistical sense of the average/median/variation
    Summarize results (statistically)

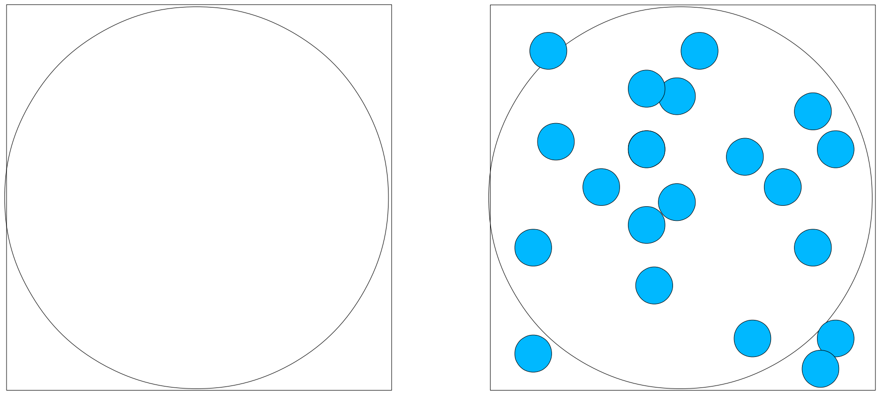
[☆]Named after the Monte Carlo Casino in Monaco [†]see wikipedia.org/wiki/Monte_Carlo_method

Next several pages: Examples of using Monte Carlo simulations

Basic Example

On your way to the World Darts Championship, you crash land on a remote island. In the process of building your MacGyver-esque signaling beacon, you need the ratio of areas for a square and a circle of the same width. However, the collision knocked the value of π from your memory.

Algorithm: throw a dart 100 times at a picture of a square with a circle inscribed. Ratio the number that fall within the square to the number that fall within the circle. *Repeat this process 100,000 times to get a good statistical understanding of your result!*



This example: 20/16 = 1.25

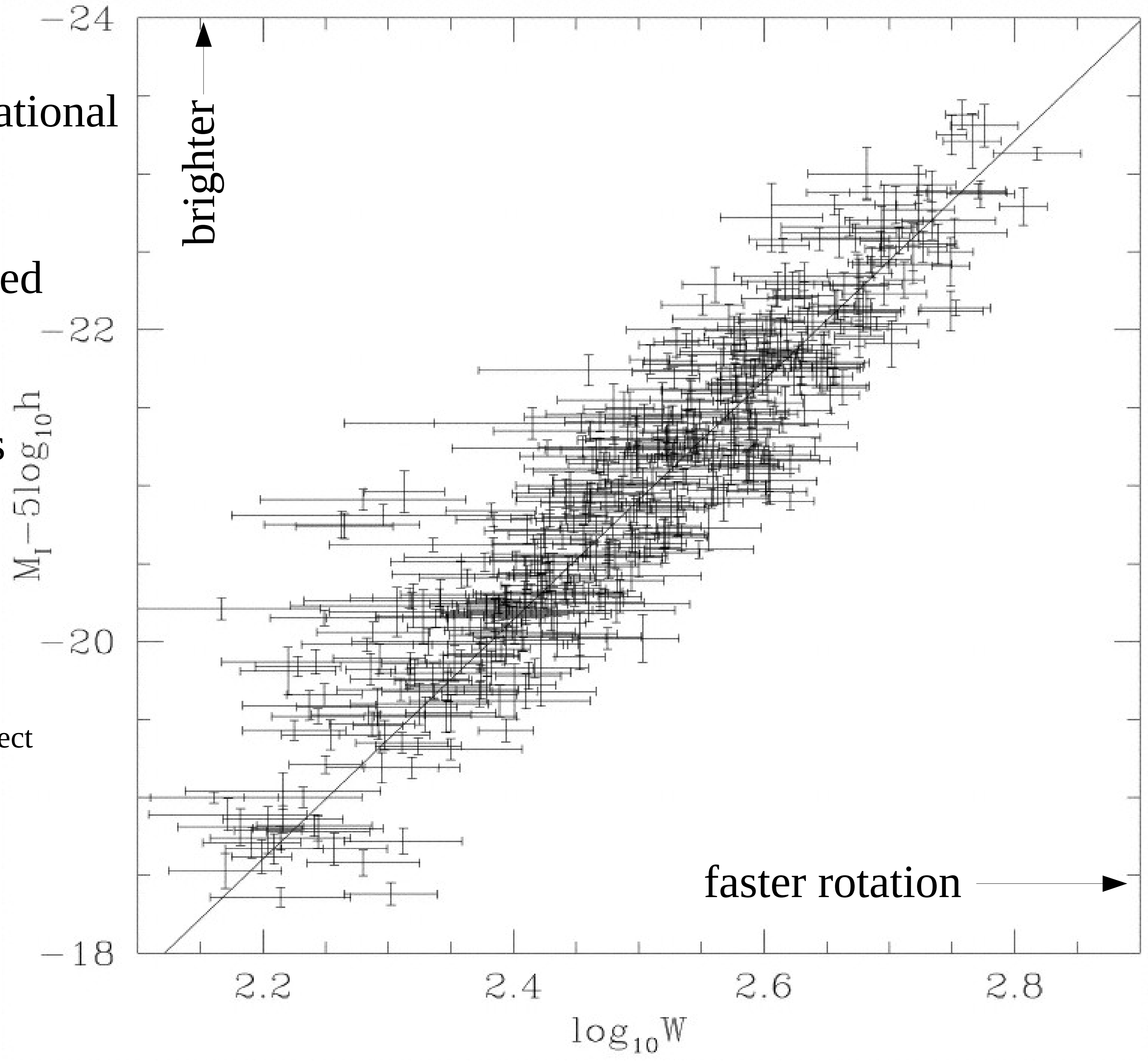
(versus the expected ________________)

Measurements of brightness and rotational speed

A brightness-limited survey will systematically miss faint galaxies

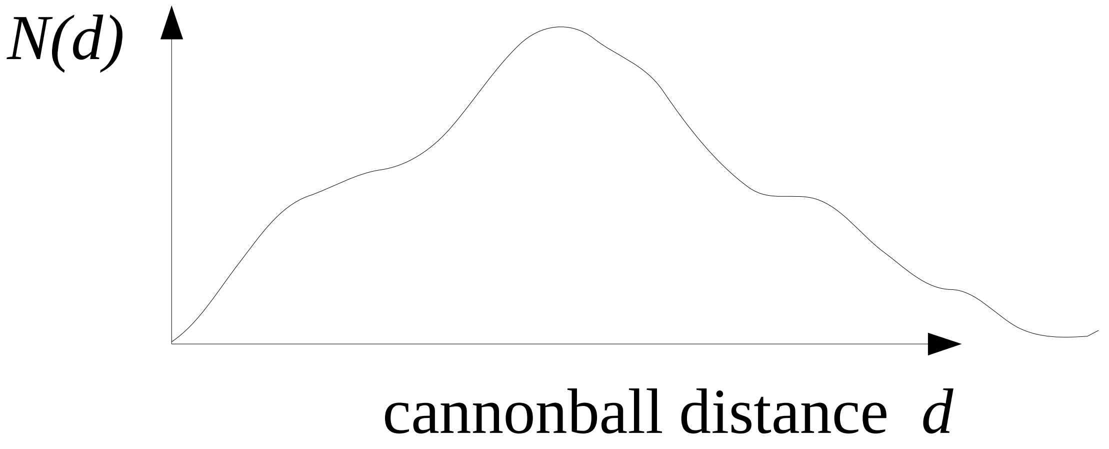
How would systematically missing objects “below” –18 on the *y*-axis affect linear fits to these data?

One can simulate (and correct for) such a bias.



How do you simulate a data set that follows a known distribution?
⇨ Acceptance-Rejection!

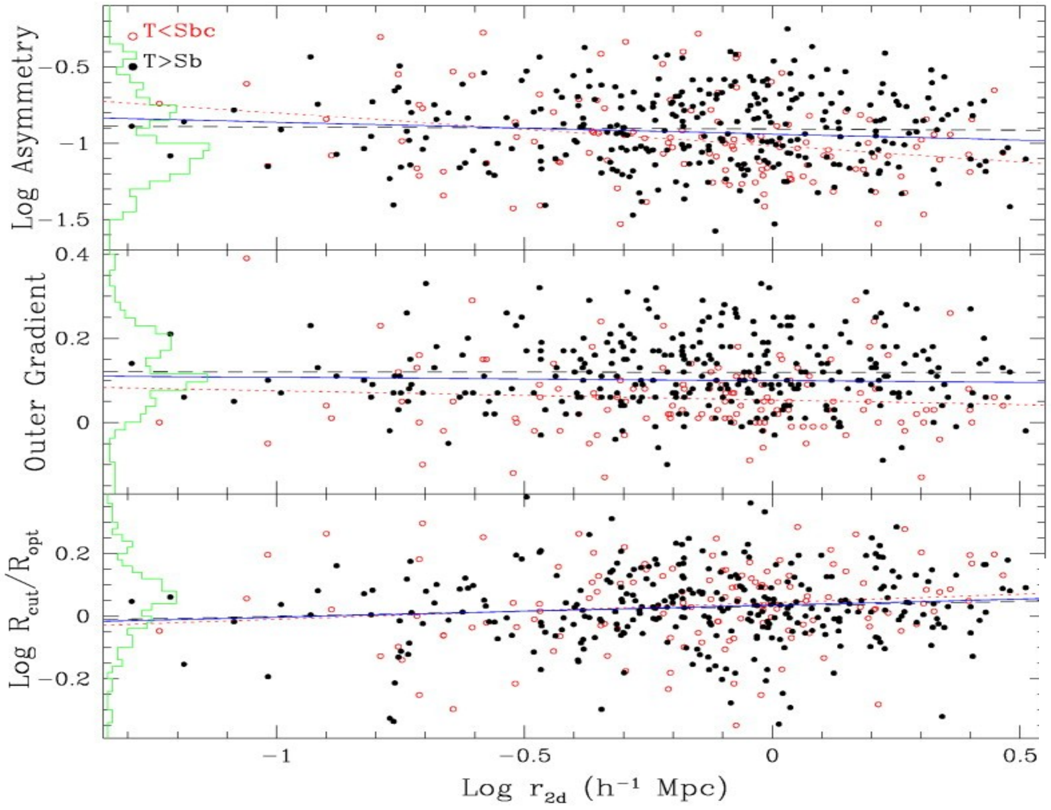
Imagine you need to simulate a cannon fight carried out on a medieval battlefield. To properly predict Team A's fighting effectiveness with a cannon, you first need a known distribution of cannon ball trajectories. Your faithful squire Longbeard reports the following distribution from previous battles:

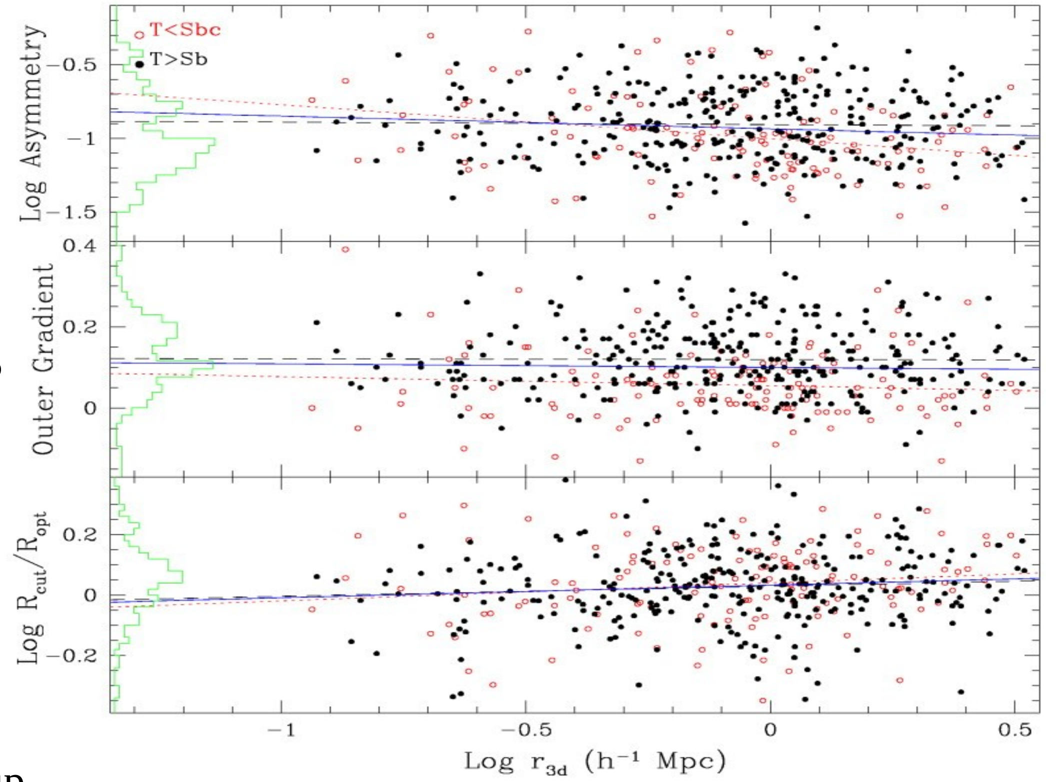


Convert to a probability distribution $P(d)$ by normalizing $N(d)$ to range from 0 to 1 (i.e., divide by the max $N(d)$).
Randomly select a trial distance d_{trial}; record that distance's probability $P(d_{\text{trial}})$

Is this trial distance reasonable?

```
    Randomly select a number $\eta$ from 0 to 1
    if $\eta < P(d)$ then *accept* this trial distance
    if $\eta > P(d)$ then *reject* this trial distance
Repeat many, many times
```





Deprojecting two-dimensional distances to three-dimensional distances

Generate a random line-of-sight distance $D_{\text{l.o.s.}}$

Compute trial 3D distance: $r_{3d}^{\ 2} = r_{2d}^{\ 2} + D_{\text{l.o.s.}}^{\ 2}$

Compute expected density ratio at this location:

$$\eta(r_{3d}) = n(r_{3d})/n(0)=(1+r_{3d}^{\ 2}/r_{core}^{\ 2})^{-3/2}$$

Generate random number γ between 0 and 1

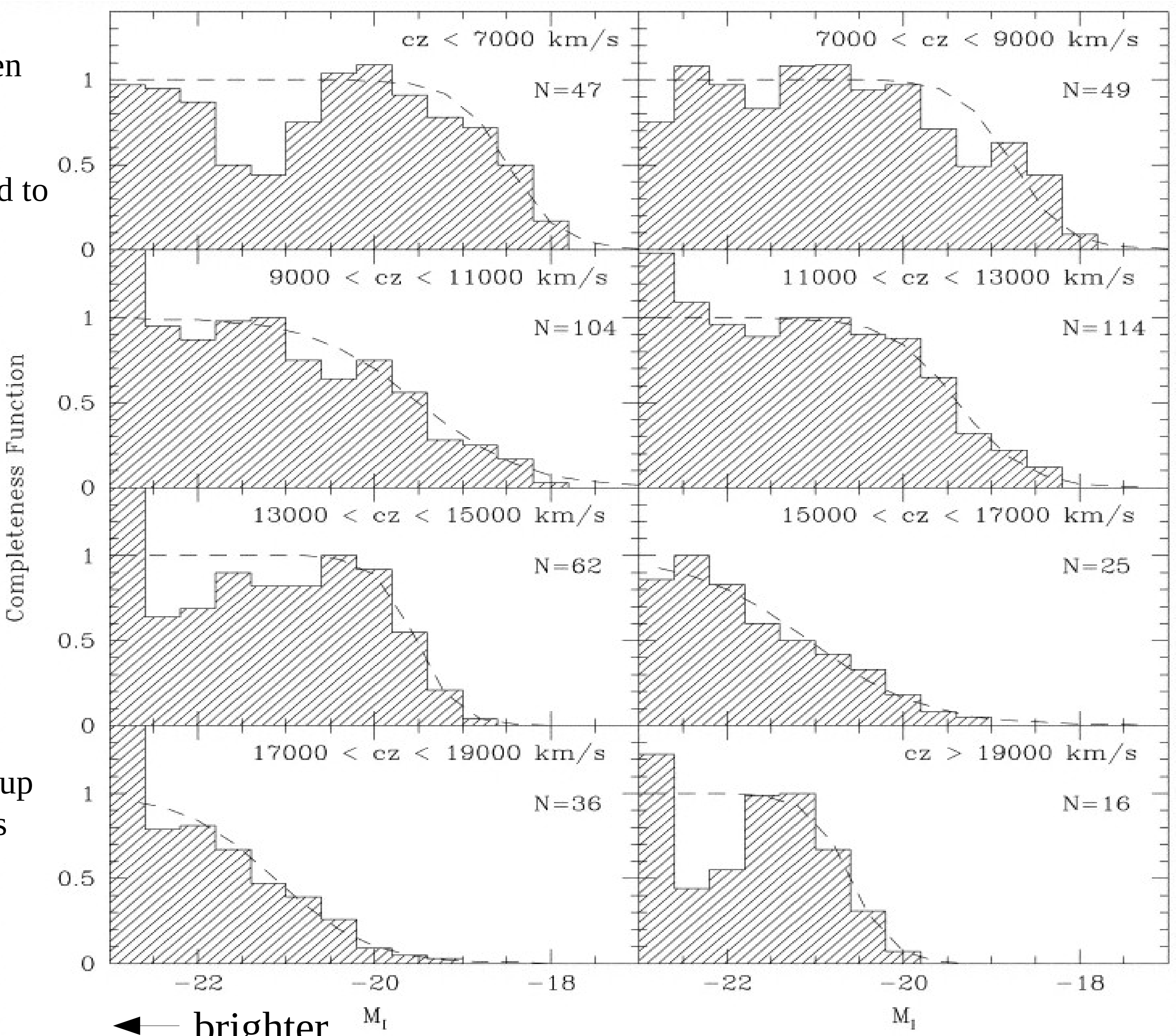
If γ is less than η, then accept this r_{3d}; otherwise reject

Repeat several thousand times until r_{3d} is statistically built up

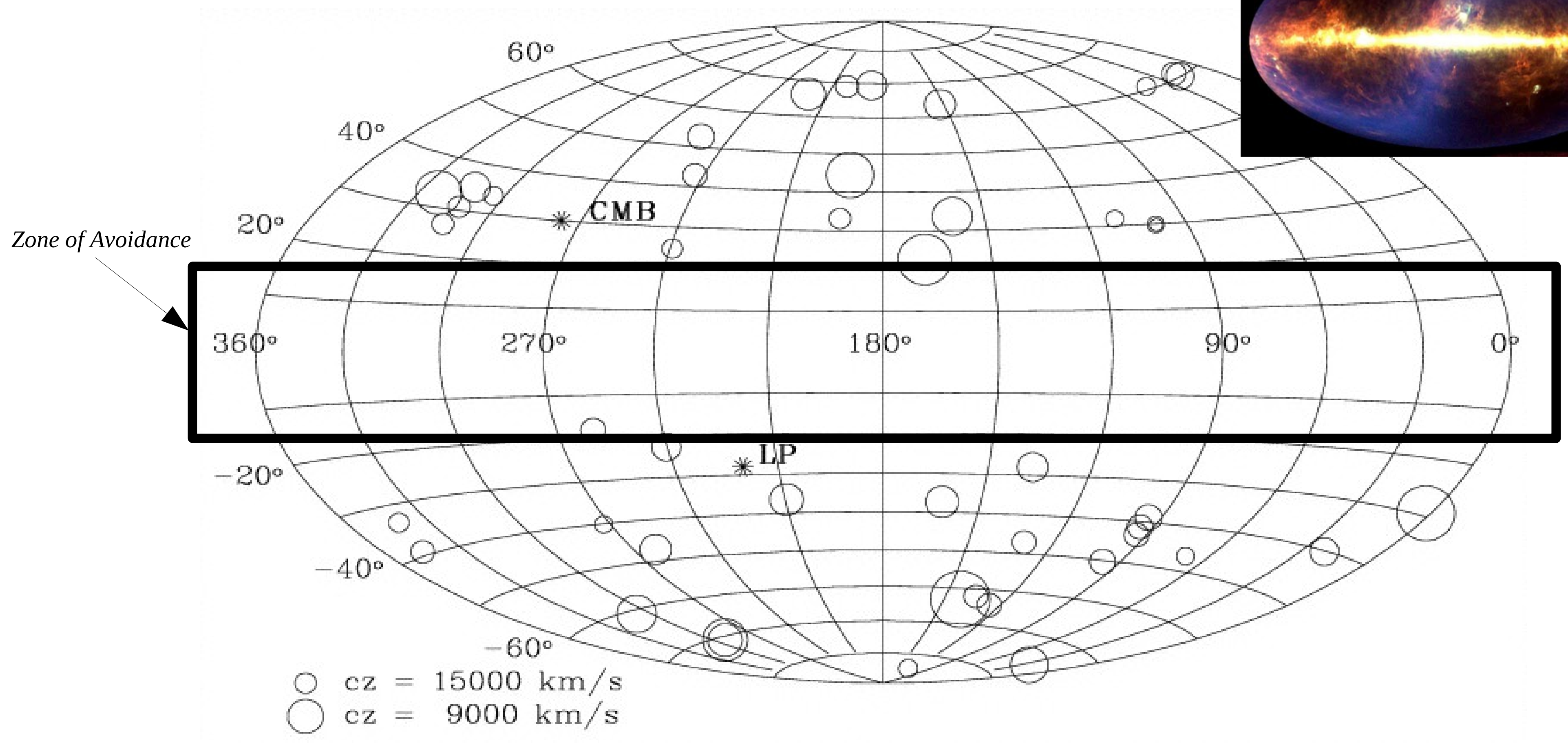
The distributions of *observed* galaxy luminosities.

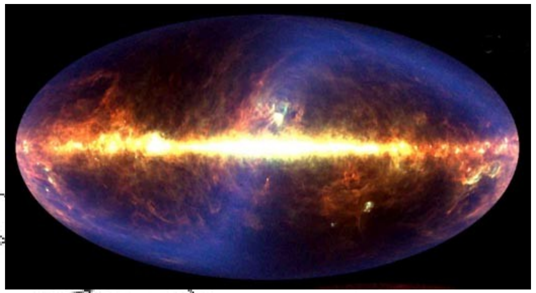
Using the known distribution of galaxies, one can simulate how often faint galaxies should appear in a survey. At the right are known luminosity distributions, normalized to have a maximum value of 1. In generating simulated galaxies, one needs to rely on these distributions.

Select a trial galaxy (luminosity)
Use the plots on the right to
    determine the likelihood η of
    its luminosity
Select a random number γ between
    0 and 1
If γ is less than η, then accept the
    trial; otherwise reject
Repeat thousands of times to build up
    a library of simulated galaxies



← brighter



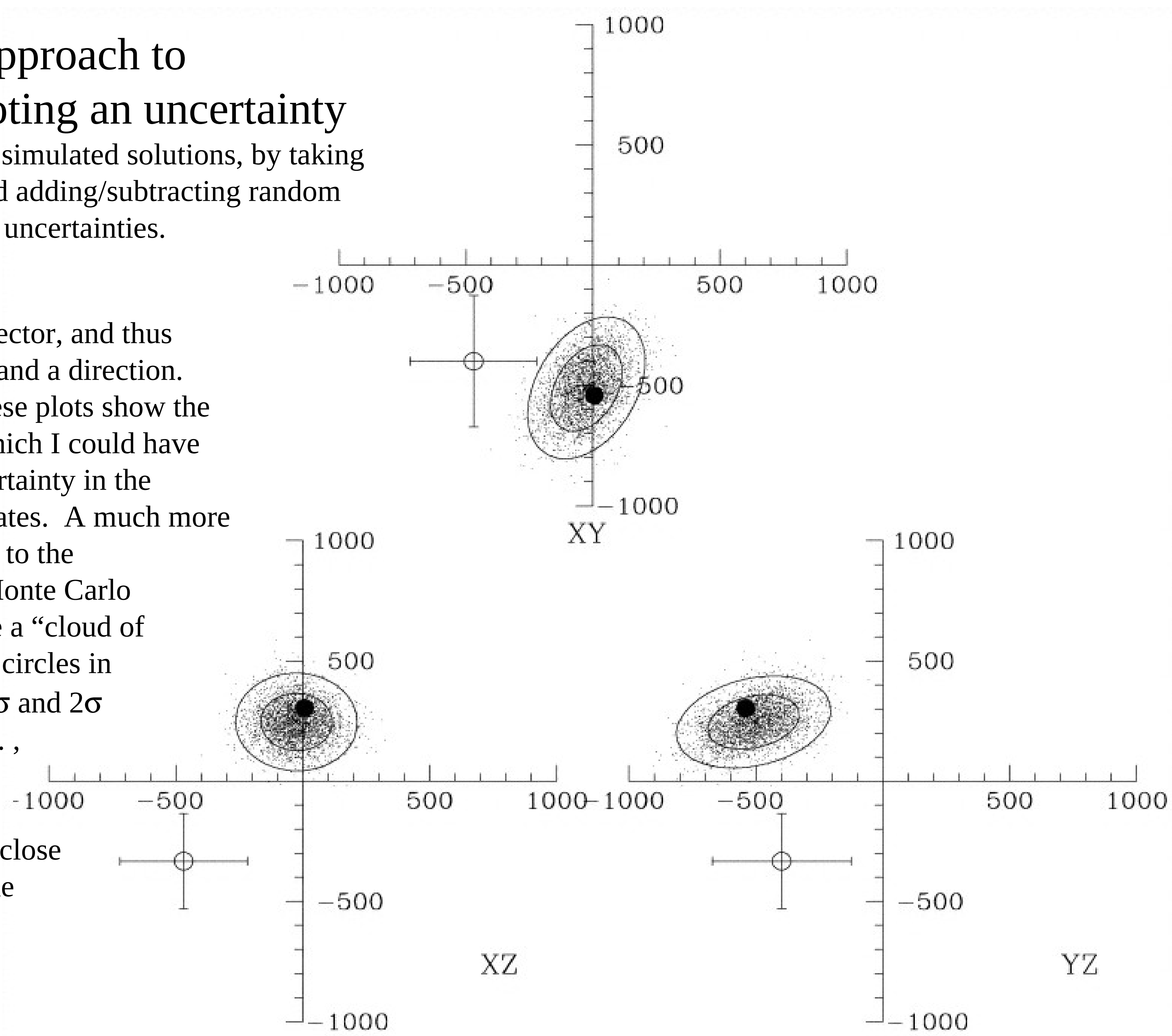


Project: Measuring the bulk motion of galaxy clusters in the Local Universe

Each circle represents a cluster of galaxies. For my project I measured each cluster's *peculiar motion* through space. No clusters are seen in the *Zone of Avoidance*, due to obscuration of background objects by the foreground Milky Way. What is the impact of the *ZoA* on all-sky surveys? → Quantify with simulations!

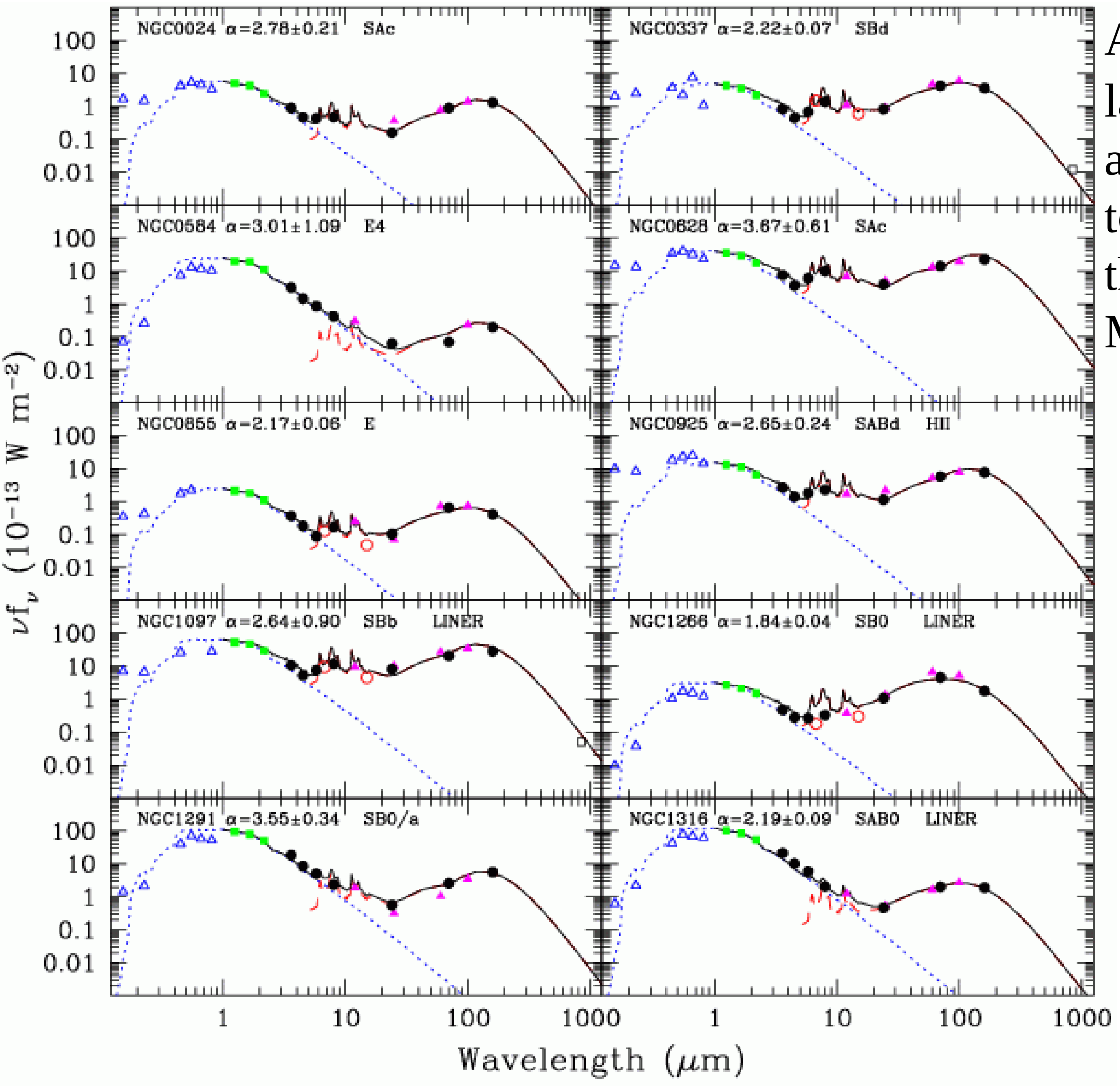
An improved approach to statistically quoting an uncertainty

Generate thousands of simulated solutions, by taking the overall solution and adding/subtracting random fractions of the known uncertainties.



The bulk motion is a vector, and thus has both an amplitude and a direction. The filled circles in these plots show the overall solution, for which I could have merely quoted an uncertainty in the amplitude and coordinates. A much more sophisticated approach to the uncertainty utilitizes Monte Carlo simulations to generate a "cloud of uncertainty". The two circles in each panel represent 1σ and 2σ confidence ellipses, i.e. , confident at the 68.3% and 95.4% levels, because they enclose 68.3% and 95.4% of the simulated solutions.

Quantifying uncertainties in fitting models to galaxy spectra.



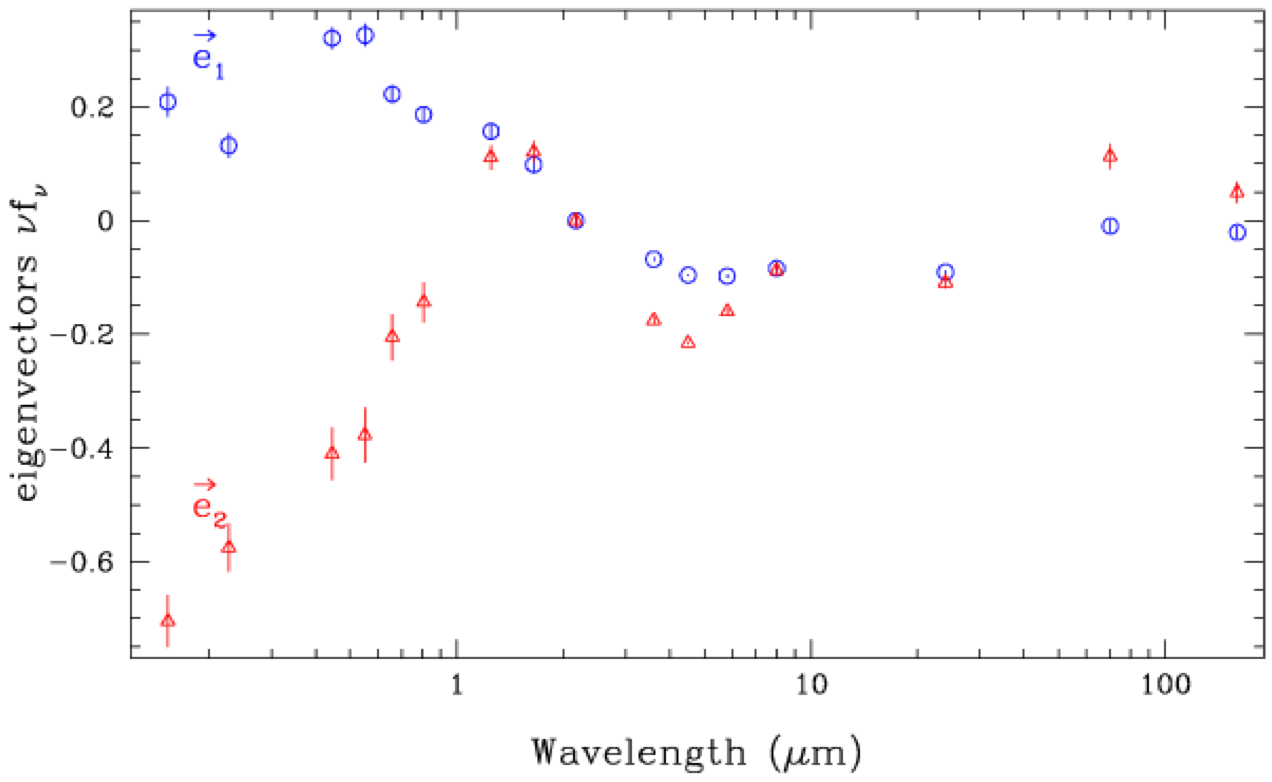
A series of spectral templates, labeled with a parameter α, are used to fit data. One way to quantify the uncertainty in the template fitting is to use Monte Carlo simulations:

- -tweak each data point according to its error
- -redo the fit
- -repeat thousands of times
- -uncertainty in α estimated from standard deviation in the fits

Quantifying uncertainties in a Principal Component Analysis.

A principal component analysis yields 'eigenspectra' that indicate underlying components to spectral data. Think of eigenvectors in quantum mechanics and linear algebra, and expressing data in terms of linear combinations of the eigenspectra.

Eigenspectra are computed from a diagonalization of a covariance matrix defined by the data. It would be complicated to formally compute the uncertainties in the eigenspectra. Monte Carlo simulations make it easy.



Redo the eigenspectra calculation thousands of times, with each simulation involving slightly-tweaked data according to the data errors. The plotted error bars stem from the standard deviations in the simulations.

Let's do an example

Suppose we are interested in computing the total mass of all aliens on Planet X, but that our instruments can only detect aliens with mass of 30 kg or larger. In our sampling of aliens with masses m=[35,45,55,65,75,85,95] kg we find N=[66,54,46,34,26,14,5] aliens. The uncertainty on the jth count is ε_{Nj}=____________ and thus
ε_N=[, , , , , ,].

The observed trend from 35 to 95 kg is well fit with a first-order polynomial,

$N(m) = -m/\mathrm{kg} + 100$,

where m is the mass in kg. Thus the total mass between 100 and 0 kg can be computed after extrapolating this trend to zero mass:

What is the uncertainty on this estimate? We can compute it by repeating the above exercise thousands of times, and taking the standard deviation in the results as the uncertainty. *(ch11/aliens.f)*

Just-in-Time Question: Random Number Generators

Suppose you generate 100,000 simulated numbers using the '*rand*' function of MATLAB. What are the expected ratios $N(0.1\text{-}0.2)/N(0.75\text{-}0.9)$ and $N(1.3\text{-}1.4)/N(0.1\text{-}0.2)$, where $N(x\text{-}y)$ is the number of times '*rand*' yields a value between x and y?

General hint for coding

Beware that Homework #7 can take awhile, especially when doing 10^6 trials for the acceptance-rejection problems. We are not using supercomputers in PS 227, so carefully try to streamline your code for efficiency. e.g., make all initial calculations such as P_{max}=3/4 outside of your loops, and then only use P_{max} in the loop. This will avoid doing things like calculating 3/4 again and again and again...