Topics in Astronomical Research
Computational Astrophysics

PHYS 913

Dr. Mark Richardson
ZoomID: 990 2595 7333: Pass: PHYS913
Mark.Richardson@queensu.ca

Last time …

- Overview of observing data formats
- Phase I and II proposals
- Data reduction

Today:

- Creating mock observation data
- Data fitting
- Optimization
- MCMC
- Bayesian Inference

Mock Observations from Simulations

- What is the point of an astrophysics simulation?
- How is nature studied or seen?
- So how can we do the former given the latter?

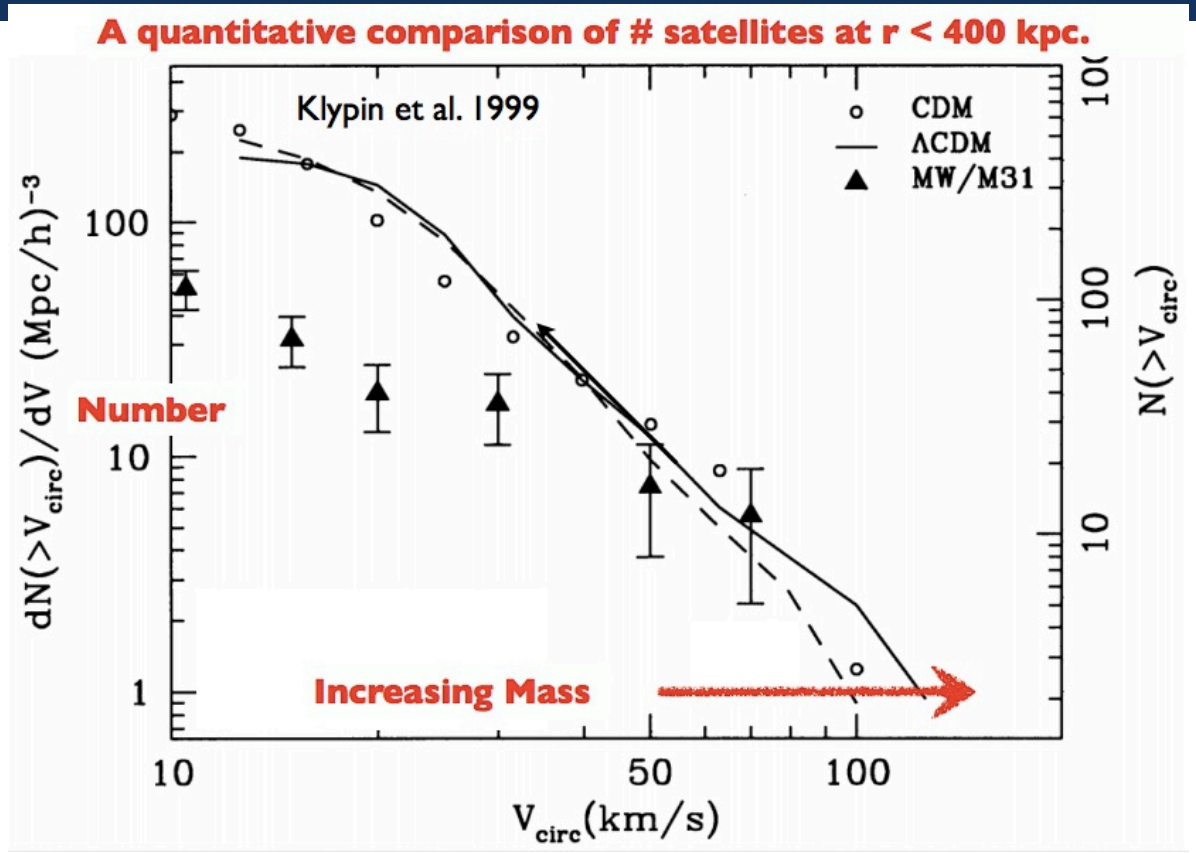
Mock Observations from Simulations

- Ultimately comparing a simulation output with observations is not an apples-to-apples comparison.

- What would be ideal is to account for the observational impact of studying an object and convolve that into your conclusions from the simulations, and the comparisons you make with observations.

Possible Example

- The Missing Satellite problem
- Observations only see so many satellites around the Milky Way and other galaxies.
- Simulations predict far more satellite galaxies.
- This is still a topic of much debate: is this effect real?



From Klypin et al. 1999, astrobites (Khullar, 2017)

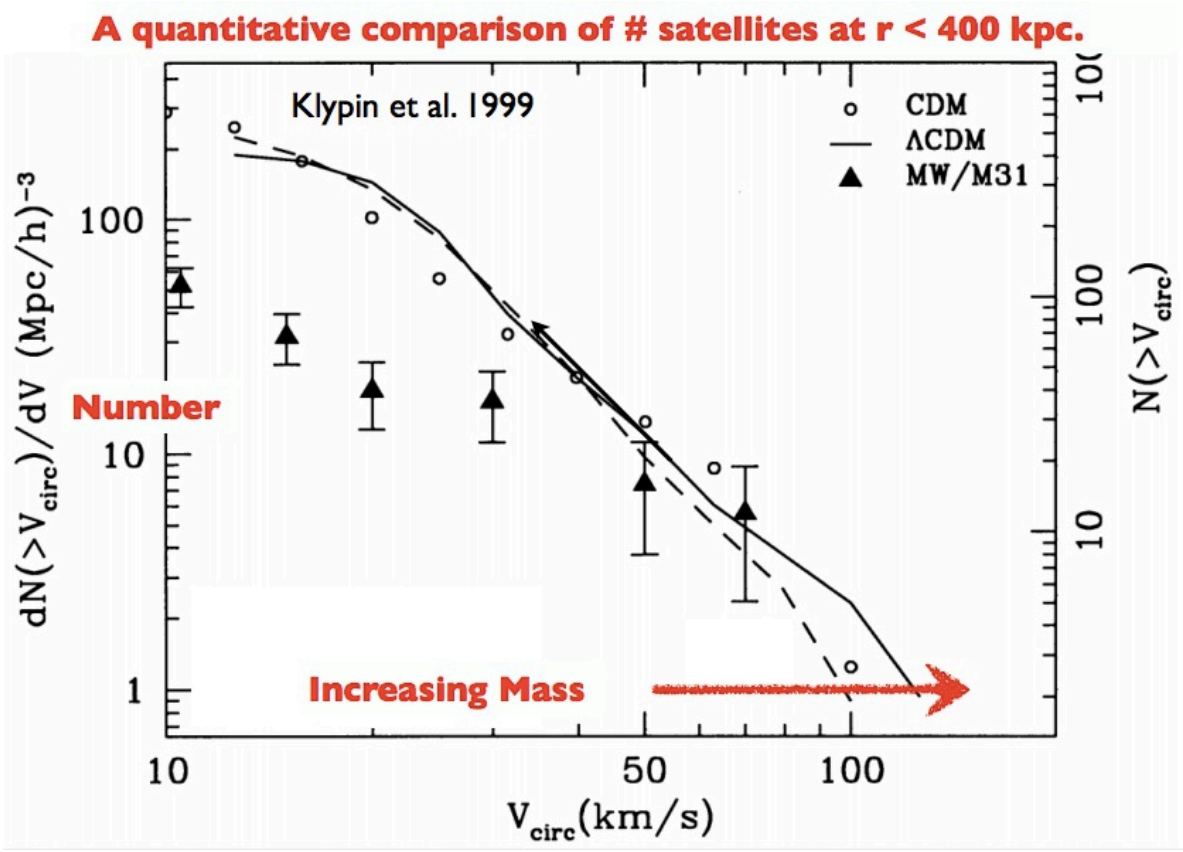
Making Mock (Simulated) Observations

- First step: What are the observations you are comparing to?
- Second step: What is the *in situ* data that those observations are measuring?
- Third step: Use your simulations to predict what that same observed data would be, *in situ*.
- Fourth step: Recreate the observing pipeline but applied to your "mock" observation data.

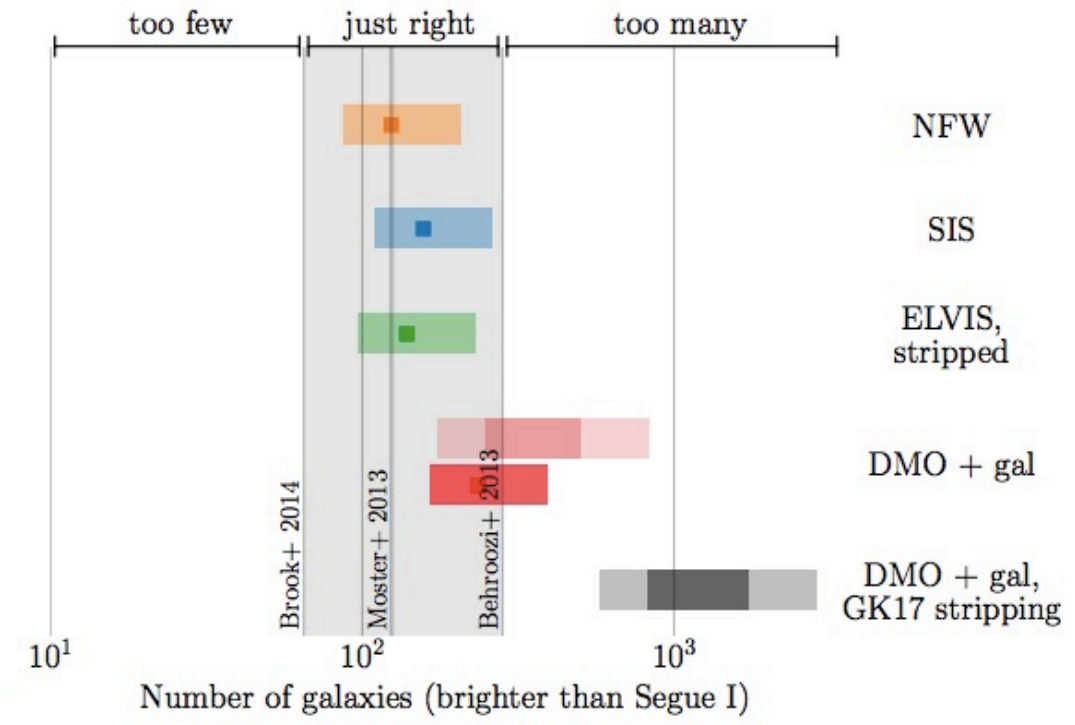
First example: Missing Satellite Problem

- Making mock galaxy satellite candidates.
- Observations use SDSS data of galaxies: *g, r, i* data sensitive to starlight, with some minimum surface brightness sensitivity.
- Need to convert number of satellites in a simulation (DM only, or with gas and stars) to predicted starlight and thus surface brightness.
- Then model how many satellites would be seen using the same observation techniques.

First Example: Missing Satellite Problem



From Klypin et al. 1999, astrobites (Khullar, 2017)

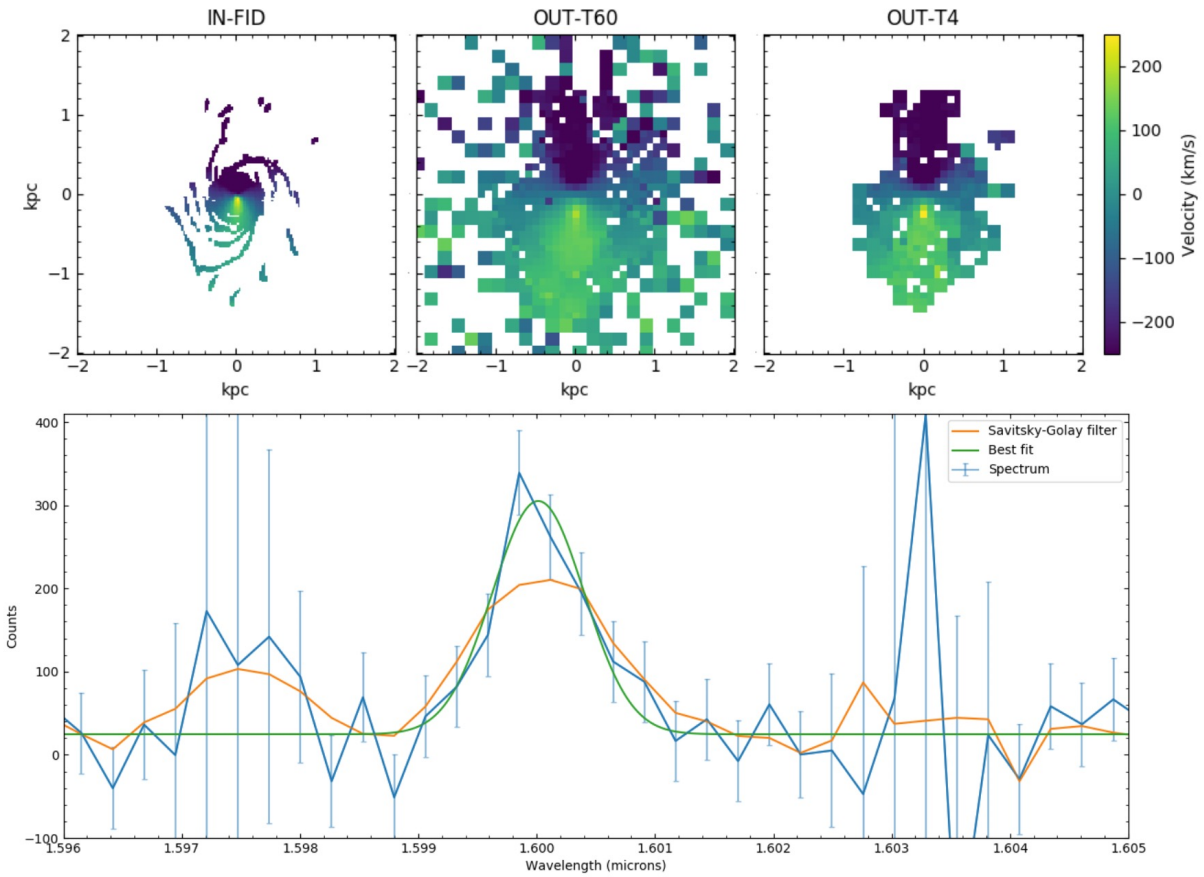


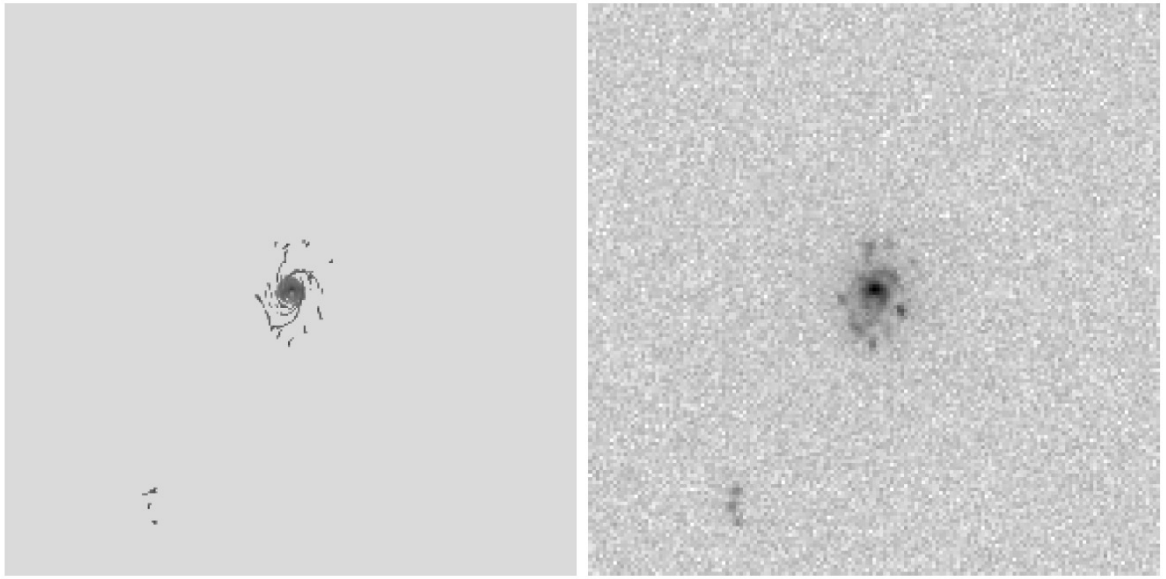
Kim et al. 2018

Second Example: Resolved kinematics at z=1.4

- New telescopes like the ELT will permit resolved study of the gas kinematics in z~1-3 disks, possible revealing information about the dark matter kinematics.
- Observation: gas emission (e.g., Hα).
- Simulations with gas (AMR): Predict the Hα from each cell (see Richardson et al. 2020)
- Model the observation itself (see HSIM: Zieleniewski 2015): Simulates how light is processed through the atmosphere, AO system, dish, optics, etc.
- Can predict expect SNR for a given galaxy mass and observing time.

Second Example: Gas Kinematics





- Richardson et al. 2020

Caveat:

- Additional assumptions go into converting your simulation data into the predicted observation data (e.g., mass-to-light, IMF of your star particles, optical depth of your cell, etc.)
- You can consider different models of this physics, and consider them in the same way you incorporated your physics into the simulations to begin with.
- Also many assumptions about the impact of observing as well.

- Can also learn about the real observational biases by comparing the input and output datasets.

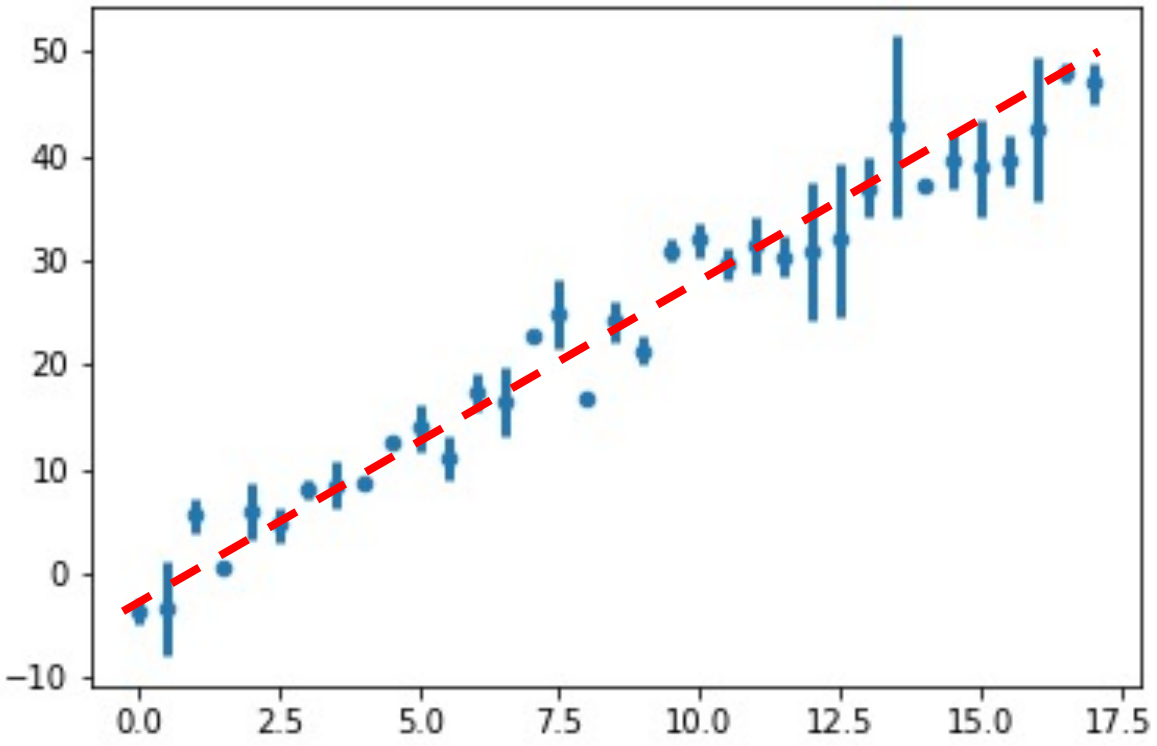
Fitting data

- We've seen when discussing interpolation how to fit a polynomial data exactly to data.

- However, if the data is noisy, or not represented by a high-order polynomial, a lower order polynomial that approximated the data is a better approach.

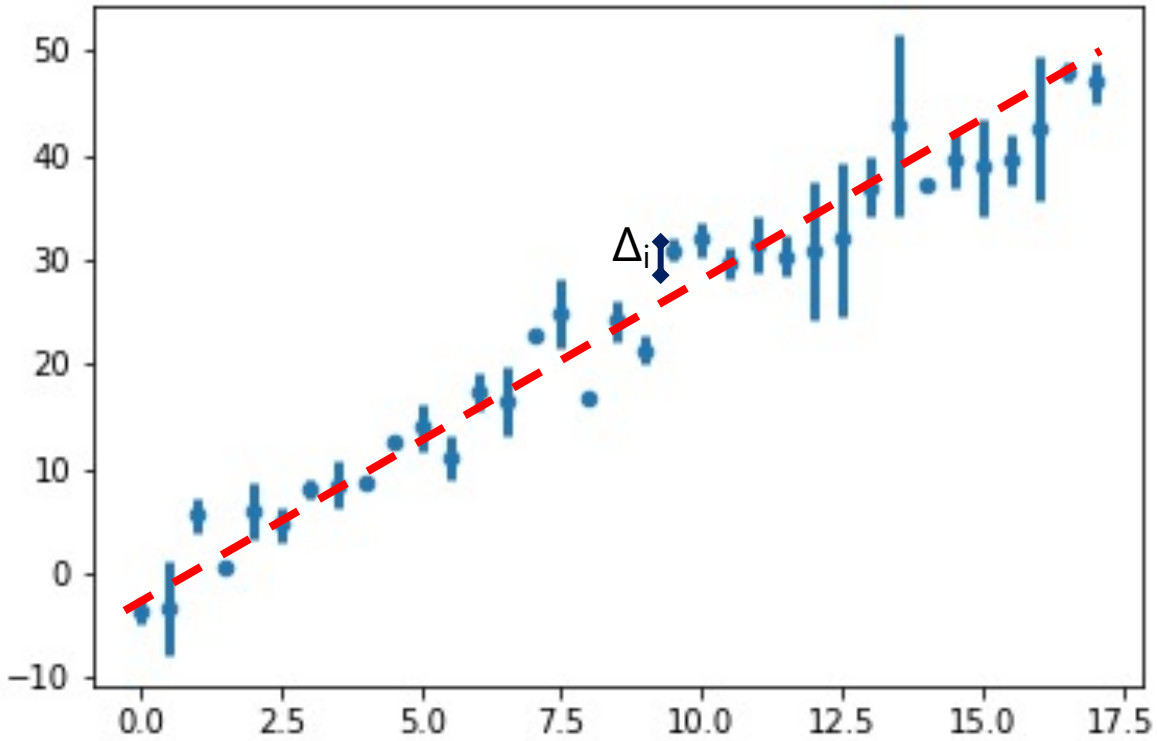
Fitting data

- Consider *N* data points (x_i, y_i)
- We often may also have errors/uncertainties/noise associated with each point: (σ_i).
- We may want to understand just the trend in the data, or maybe there is a physically motivated functional form for the data.
- This model can be charactized as some function, $Y(x,\{a_j\})$, defined by parameters a_j.



Fitting data

- We're likely familiar with the approach: χ^2 minimization:
- Let $\Delta_i = Y(x_i,\{a_j\}) - y_i$
- Then $\chi^2(\{a_j\}) = \sum_{i=1}^{N} \left(\frac{\Delta_i}{\sigma_i}\right)^2$
- We are looking for the set of parameters, a_j, that minimizes χ^2.



Fitting data: Linear Regression

- Consider the model Y(x) = a_0 + a_1x, with two parameters.
- So we want to minimize:
- $\chi^2(a_0, a_1) = \sum_{i=1}^{N} \left(\frac{a_0 + a_1 x_i - y_i}{\sigma_i}\right)^2$.
- Thus: $\frac{\partial \chi^2}{\partial a_0} = 2\sum_{i=1}^{N} \frac{a_0 + a_1 x_i - y_i}{\sigma_i} = 0$; $\frac{\partial \chi^2}{\partial a_1} = 2\sum_{i=1}^{N} \frac{a_0 + a_1 x_i - y_i}{\sigma_i} x_i = 0$
- Convenient to define the following:
- $S \equiv \sum_{i=1}^{N} \frac{1}{{\sigma_i}^2}$; $\xi_1 \equiv \sum_{i=1}^{N} \frac{x_i}{{\sigma_i}^2}$; $\xi_2 \equiv \sum_{i=1}^{N} \frac{{x_i}^2}{{\sigma_i}^2}$.
- $\eta_0 \equiv \sum_{i=1}^{N} \frac{y_i}{{\sigma_i}^2}$; $\eta_1 \equiv \sum_{i=1}^{N} \frac{x_i y_i}{{\sigma_i}^2}$.
- If σ_i=0 or constant, you can divide it out and then S=N.

Fitting data: Linear Regression

- We can then rewrite the two equations as:
- $a_0 S + a_1 \xi_1 - \eta_0 = 0$
- $a_0 \xi_1 + a_1 \xi_2 - \eta_1 = 0$

- This is trivial to solve for a_0 and a_1:
- $a_0 = \frac{\eta_0 \xi_2 - \eta_1 \xi_1}{S\xi_2 - \xi_1^2}$; $a_1 = \frac{S\eta_1 - \eta_0 \xi_1}{S\xi_2 - \xi_1^2}$

- A good fit will have $|\Delta_i| \sim \sigma_i$.

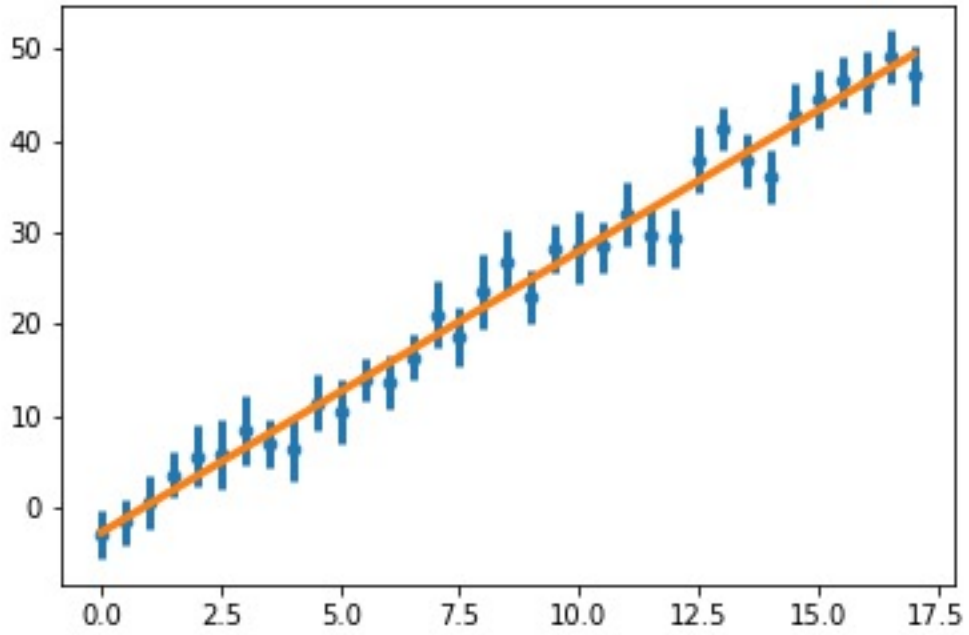
Fitting data: linear Regression

- A given fit will have *M* parameters (e.g., linear fit has 2), and you want N >> M so your not over-fitting.
- Number of degrees of freedom is N-M.

- Specifically: $\frac{\chi^2}{N-M} < 1$ is a good fit. If it's way less than 1 you likely have too many parameters, or your noise is too large.

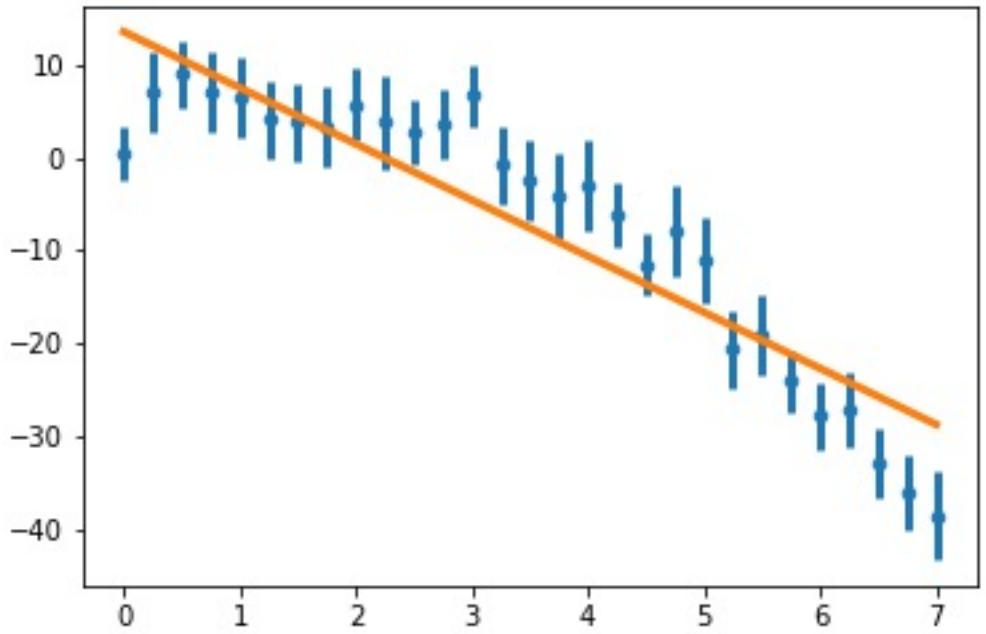
Examples

- Data: y=3x-2 with gaussian noise (σ=2)
- Fit: Y=3.065x-2.67, χ^2=0.48



Examples

- Data: $y=-1.2x^2 + 2x + 5$ with gaussian noise ($\sigma=3$)
- Fit: $Y=-6.02x+13.48$, $\chi^2=2.65$



Higher order polynomials

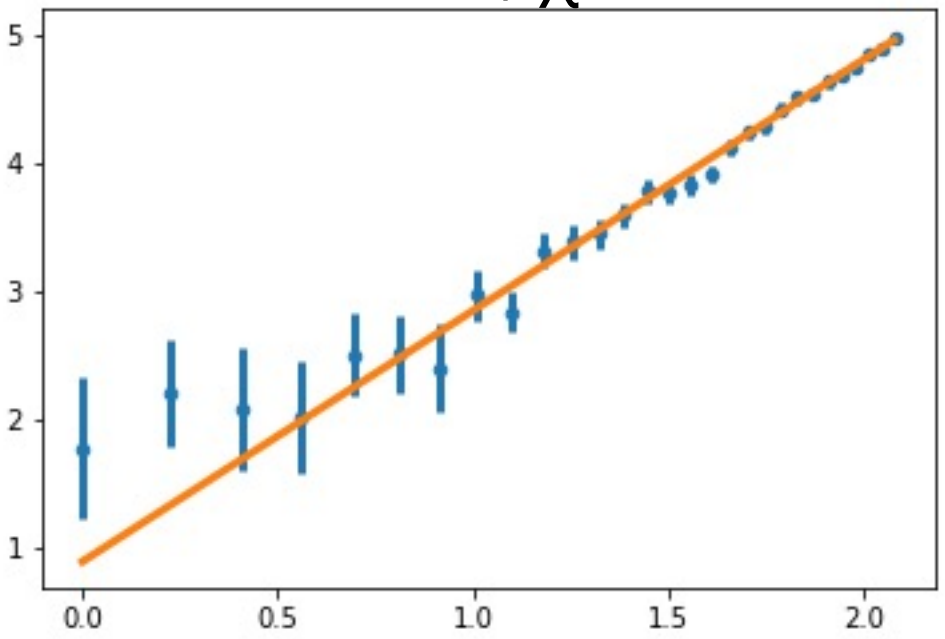
- In general it is not as easy to solve for higher order polynomials, but the system of equations reduce to:
- $Y(x;\{a_j\}) = \sum_{j=0}^{M-1} a_j\, x^j$
- $S \equiv \sum_{i=1}^{N} \frac{1}{{\sigma_i}^2};\ \xi_p \equiv \sum_{i=1}^{N} \frac{{x_i}^p}{{\sigma_i}^2};\ \eta_p \equiv \sum_{i=1}^{N} \frac{{x_i}^p y_i}{{\sigma_i}^2}$
- Then:
- $$\begin{bmatrix} S & \xi_1 & \xi_2 & \cdots & \xi_M \\ \xi_1 & \xi_2 & \xi_3 & \cdots & \xi_{M+1} \\ \xi_2 & \xi_3 & \xi_4 & & \xi_{M+2} \\ \vdots & & & \ddots & \vdots \\ \xi_M & \xi_{M+1} & \xi_{M+2} & \cdots & \xi_{2M} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{M-1} \end{bmatrix} = \begin{bmatrix} \eta_0 \\ \eta_1 \\ \eta_2 \\ \vdots \\ \eta_{M-1} \end{bmatrix}$$
- This can be solved with LU decomposition using existing packages. Not as M get's larger the determinant of the left tends to zero: very ill-behaved.

Fitting: Log-Log plot.

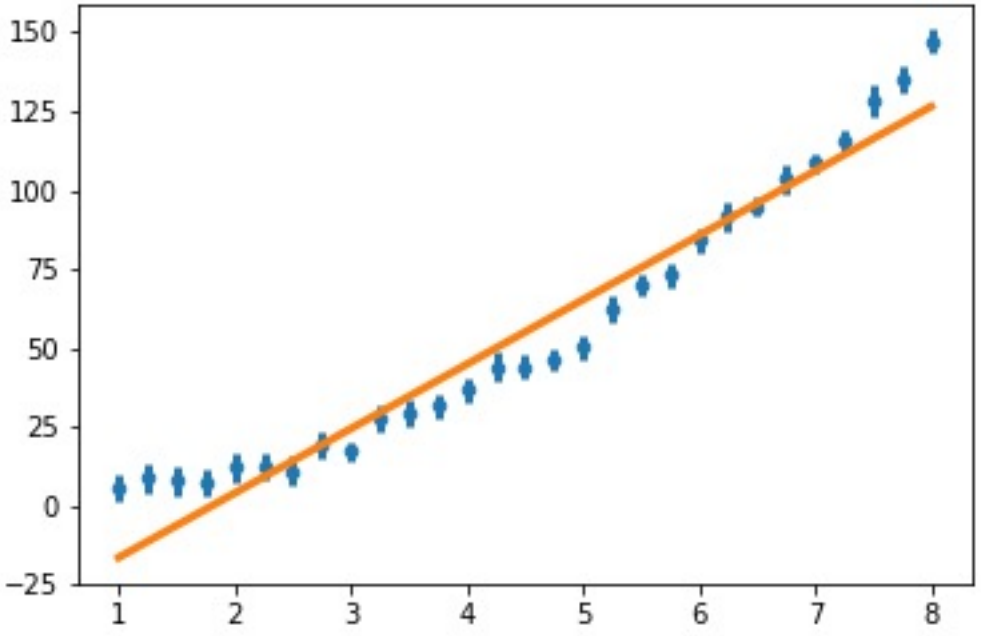
- Instead of using a higher order polynomial, we could convert our data (x_i,y_i) into (t_i,z_i) where $t=\ln x$, and $z=\ln y$.
- Be careful with your uncertainties, $\sigma_z \sim \ln (y+\sigma_y) - z$
- Then fit $Z=a_0 + a_1 t \Rightarrow Y=\alpha x^{\beta}$, $a_0=\ln \alpha$, $a_1= \beta$.

Example

- Data: $y=2.2x^2 + 0.2x + 2$ with gaussian noise ($\sigma=3$)
- Fit: $Y=2.46x^{1.96}$, $\chi^2=0.79$



- Vs: Fit: $Y=20.4x-36.6$, $\chi^2=6.64$



Even more general approach

- Linear combination of (basis) functions (Y_j could be x^{j-1}):
- $Y(x;\{a_j\}) = \sum_{j=1}^{M} a_j Y_j(x)$
- Again minimizing χ^2, get the M equations:
- $\sum_{i=1}^{N} \sum_{k=1}^{M} \frac{Y_j(x_i)}{\sigma_i} \frac{Y_k(x_i)}{\sigma_i} a_k = \sum_{i=1}^{N} \frac{Y_j(x_i)}{\sigma_i} \frac{y_i}{\sigma_i}$

- Defining $A_{ij} = \frac{Y_j(x_i)}{\sigma_i}$, we can rewrite the above as: $(\mathbf{A}^T\mathbf{A})\mathbf{a}=\mathbf{A}^T\mathbf{b}$, where $\mathbf{b} = \frac{y_i}{\sigma_i}$.
- This is only well behaved for large M when the Y_j make an orthonormal basis system on the interval (e.g., Legendre Polynomials)

Optimization

- At the end of the day, we are trying to minimize χ^2 in parameter space.
- You could use gradient descent similar to our root solving technique in multiple dimensions (somewhat equivalent to Newton's method).
  - Take a guess for **a**.
  - Evaluate $\nabla_{\boldsymbol{a}}\chi^2|_{\boldsymbol{a}}$
  - Take a step in the direction of maximum gradient (γ is a small number > 0):
  - $\mathbf{a}_{n+1} = \mathbf{a}_n - \gamma \nabla_{\boldsymbol{a}}\chi^2|_{\boldsymbol{a}^n}$
- However, while this will always find a minimum, it may be a local minimum.
- We need another way to find the minimum: Naively: what if we randomly sampled the function, and just use whatever yields the smallest value?

Revisit: Monte Carlo Techniques

- Recall: Monte Carlo techniques.
- We saw this previously for integration: sampling a function to determine it's mean value.
- $I = \int_a^b f(x)dx = (b-a)\langle f(x_i)\rangle$.
- We also saw the importance of sampling where the function varies more rapidly: "Importance Sampling"
- $I = \int_a^b f(x)dx = \int_a^b w(x)dx \left\langle \frac{f(x_i)}{w(x_i)} \right\rangle_w$

Other Monte Carlo uses

- Monte Carlo techniques can let you do determine probabilities of certain things happening.
- Consider a dice roll: Any given number is a 1/6 chance.

```
def get_roll_frac(Nrolls,ndice):
    Rolls = np.zeros([Nrolls])
    for i in range(ndice):
        Rolls += np.floor(np.random.random(Nrolls)*6+1)
    N = []
    for i in range(ndice-1,6*ndice):
        N.append(np.sum( (i+1<=Rolls)&(Rolls<i+2)))
    N = np.array(N)/Nrolls
    return N
```

- For one dice this is trivial, and will clearly tend to 1/6:

Rolling Dice

- 1 Die:
  - 100 rolls: [0.17 0.11 0.19 0.23 0.16 0.14]
  - 1,000 rolls: [0.169 0.184 0.167 0.162 0.161 0.157]
  - 10,000 rolls: [0.1728 0.1678 0.1664 0.1644 0.1656 0.163]
  - 10,000,000 rolls: [0.1666863 0.1666007 0.1666326 0.1666778 0.1666603 0.1667423]

- But what about 3 dice? Could get between 3-18-roll:
- [3: 0.46% 4: 1.39% 5: 2.78% 6: 4.63% 7: 6.96% 8: 9.72%
  9: 11.55% 10: 12.48% 11: 12.51% 12: 11.59% 13: 9.72% 14: 6.95%
  15: 4.63% 16: 2.78% 17: 1.39% 18: 0.47%]

Monte Carlo applications

- You could have done the math, but a quick program is easier if you don't need much precision.
- You could also look at poker hands … at cribbage hands … at Covid spread … etc. (see Ryan Martin's IGnite talk from December).
- Re optimization: Maybe we can sample our parameter space in the same way to minimize χ^2.
- One issue: idea of importance sampling. How likely are you to actually sample where χ^2 is small? Especially if it's a small region. We need to overcome this issue.

Markov Chain Monte Carlo

- (Mostly following Newman)
- Let's consider a system in equilibrium with temperature, T.
- Probability of occupying a state i with energy E_i is:
- $P(E_i) = \frac{e^{-E_i\ /kT}}{Z}$, Z is the partition function, or normalization:
- $Z = \sum_i e^{-E_i\ /kT}$
- The sum can be really large!!! Consider the number atoms in a mole.
- Then the expectation value for some quantity X where state i has value X_i is:
- $\langle X \rangle = \sum_i X_i P(E_i)$

Markov Chain Monte Carlo

- Let's approximate this like we did for integrals, where we use N random values in the sum.
- $\langle X \rangle \approx \frac{\sum_{k=1}^{N} X_{i(k)} P\left(E_{i(k)}\right)}{\sum_{k=1}^{N} P\left(E_{i(k)}\right)} \equiv \frac{\sum_{k=1}^{N} X_k P(E_k)}{\sum_{k=1}^{N} P(E_k)}$
- Note the need for the denominator since we aren't using all states, so we need to normalize again.
- Recall if $E_i >> kT$, then $P(E_i) << 1$.
- We need importance so we look at more probable states that contribute more to the sum.

Markov Chain Monte Carlo

- We follow the same approach as with integration.
- Define some weighting function *w* and sum over all states to get a weighted average of some quantity *g*:
- $\langle g \rangle_w = \frac{\sum_i w_i g_i}{\sum_i w_i}$
- We can let *g=XP/w*:
- $\left\langle \frac{XP(E)}{w} \right\rangle_w = \frac{\sum_i w_i (X_i P(E_i)/w_i)}{\sum_i w_i} = \frac{\sum_i X_i P(E_i)}{\sum_i w_i} = \frac{\langle X \rangle}{\sum_i w_i}$
- So: $\langle X \rangle = \left\langle \frac{XP(E)}{w} \right\rangle_w \sum_i w_i$, and remember this is over all states.

Markov Chain Monte Carlo

- Let's sample the states following the distribution defined by *w*.
- So $p_i = w_i / \sum_i w_i$

- So: $\left\langle \frac{XP(E)}{w} \right\rangle_w = \frac{1}{N} \sum_k \frac{X_k P(E_k)}{w_k}$
- So: $\langle X \rangle = \frac{1}{N} \sum_k \frac{X_k P(E_k)}{w_k} \sum_i w_i$

- Contrast with importance sampling with integration:
- $I = \int_a^b f(x)dx = \frac{1}{N} \sum_{i=1}^{N} \frac{f(x_i)}{w(x_i)} \int_a^b w(x)dx$

Markov Chain Monte Carlo

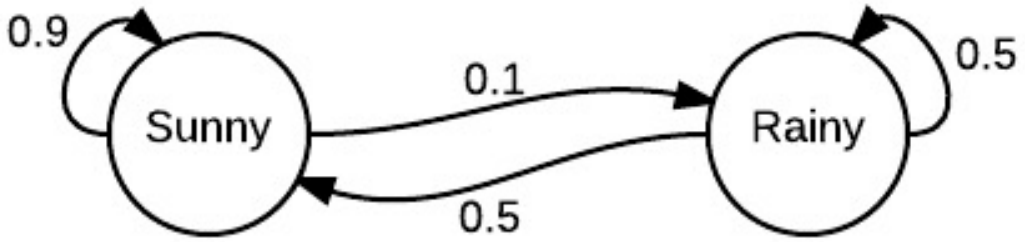
- So what are *w*? We chose the probabilities themselves:
- Recall: $\langle X \rangle = \frac{1}{N} \sum_k \frac{X_k P(E_k)}{w_k} \sum_i w_i$
- $w_i = P(E_i)$ → So $\sum_i w_i = 1$ by definition.
- And we get: $\langle X \rangle = \frac{1}{N} \sum_k X_k$

- Last difficulty: How do we sample according to the probability distribution?

Markov Chain Monte Carlo (MCMC)

- Markov Chain: A sequence of states where the next state only depends on the current state.

- Wikipedia has a cool example:
- Consider a weather prediction:
  - This is a directed graph: probability of going to one state from another.



  - States are (Sunny, Rainy) with transition matrix: $\boldsymbol{T} = \begin{pmatrix} 0.9 & 0.1 \\ 0.5 & 0.5 \end{pmatrix}$
  - Then given a state **S**, the probability of the state tomorrow is **ST**.
  - Example: If it's sunny today, **S=**(1,0), then tomorrow it's (0.9 0.1), so 90% chance of being sunny.
  - In general $\mathbf{S}^{(n)} = \mathbf{S}^{(n-1)}\mathbf{T} = \mathbf{S}^{(0)}\mathbf{T}^{n}$

Markov Chain Monte Carlo

- A Markov Chain allows us to sample from the probability distribution, not by drawing random numbers, by by going from one state to the next.

- Let's start with state *i*, and generate a proposal for the next state by making a small change (e.g,. Moving one atom a small distance).
- The transition probability then tells you whether you accept the new state: T_{ij} is the probability of going to state *j* from state *i*.
- We require: $\sum_j T_{ij} = 1$ (we are guaranteed to go somewhere, where *j*=*i* is permitted.

Markov Chain Monte Carlo

- We also require "detailed balance":
- $\frac{T_{ij}}{T_{ji}} = \frac{P(E_j)}{P(E_i)} = \frac{e^{-E_j/kT}/Z}{e^{-E_i/kT}/Z} = e^{-(E_j - E_i)/kT}$
- This is the relative probability, which can be rewritten as:
- $\Delta P = e^{-\Delta E/kT}$
- Metropolis Algorithm:
  - If ΔE < 0 (so ΔP>1): Accept the proposed change.
  - If ΔE > 0 (so ΔP<1): The change would give a higher energy state:
    - Get a random number *r* on [0:1) and accept the proposed state if *r*< ΔP
    - Otherwise: Your NEW state is *i* again.

Markov Chain Monte Carlo

- A few important points:
  - Using the Markov chain, if you are sampling the Boltzmann probability distribution already, you will continue to do so (it is a fixed point).
  - The Markov chain will (eventually) converge to the Boltzmann distribution.

- For this approach to work, every state must be accessible as defined by how you choose possible states (this is called *ergodicity)*.

- You track your entire chain and save its values.

Markov Chain Monte Carlo

- Note, this will sample the Energy space according to importance sampling.
- So what does this have to do with minimizing χ^2?
- Minimizing the Energy in the Boltzman equation is equivalent to minimizing any function.
  - For some choice of parameter space for your fit, you get a χ_i^2 value.
  - You can propose a new parameter set nearby and calculate the new χ_j^2. If it's lower accept it, if it's higher accept it with a probability given by:
  - $\exp(\chi_i^2 - \chi_j^2)$.
- Will find the global minimum if given enough time ….

Markov Chains and Probabilities

- Given some complicated parameter space and corresponding probability informed by χ_i^2, it can be quite complicated to characterize this probability space.

- A fit alone doesn't tell you how much uncertainty or variance there is in the parameters: how sensitive is χ^2 to small changes in parameters?

- Markov chains get you this information, and with it the power to understand how to incorporate new data and update your parameters.

Bayesian Inference

- Bayesian inference: How we can update our confidence in a model given out previous confidence in it, new data, and how well that model predicted the data:
- $p(\theta|D) = \frac{p(D|\theta)p(\theta)}{p(D)}$
- θ is the model defined by the set of parameters.
- $p(\theta)$ is the "prior" confidence in the model.
- $p(\theta|D)$ is the posterior, how confident are you in the model given new data, *D*.
- $p(D|\theta)$ is the probability of measuring the data given the model.
- $p(D)$ is the "model evidence", essentially saying what is the probability of measuring this data given all possible models.
- MCMC gives us the tools to understand these quantities for a given model and data.

Facilitation Time

Next time:

- More MC methods: Simulated Annealing
- More Bayesian inference
- If time: Start of machine learning.