Astronomical Data Analysis I

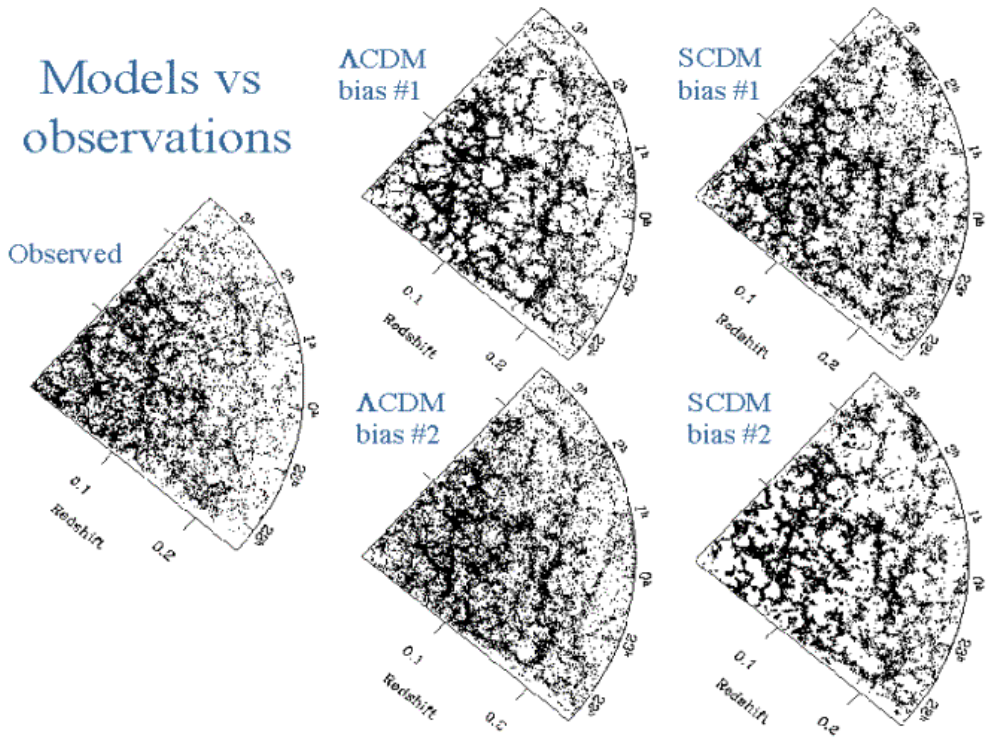
Dr Martin Hendry,
Dept of Physics and Astronomy
University of Glasgow, UK

10 lectures, beginning October 2006

4. Monte Carlo Methods

In many data analysis problems it is useful to create 'mock' datasets, in order to test models and explore possible systematic errors.

e.g. Mock galaxy catalogues:



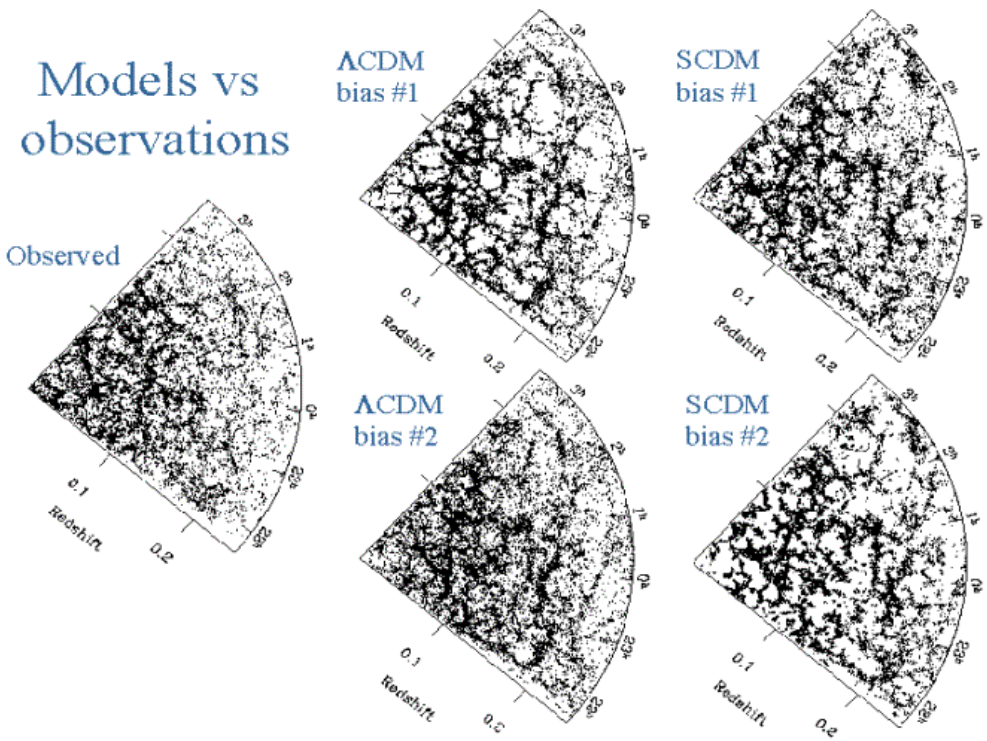
From Cole et al. (1998)

4. Monte Carlo Methods

In many data analysis problems it is useful to create 'mock' datasets, in order to test models and explore possible systematic errors.

e.g. Mock galaxy catalogues:

We need methods for generating **random variables** – i.e. samples of numbers which behave as if they are drawn from some particular pdf (e.g. uniform, Gaussian, Poisson etc).

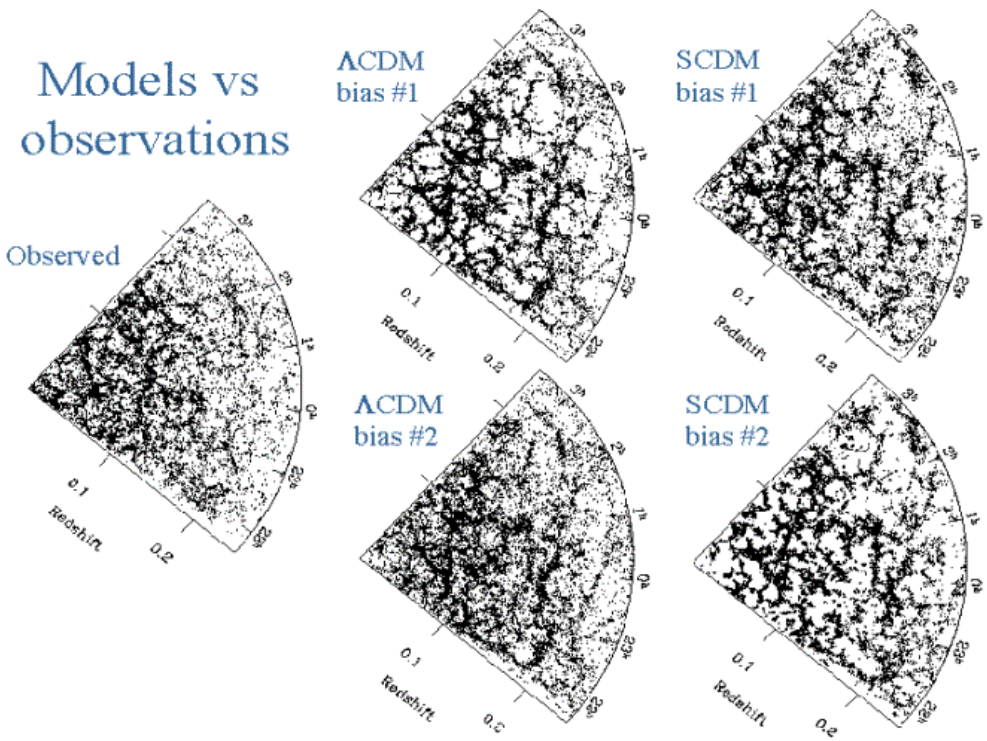


From Cole et al. (1998)

4. Monte Carlo Methods

In many data analysis problems it is useful to create 'mock' datasets, in order to test models and explore possible systematic errors.

e.g. Mock galaxy catalogues:



We need methods for generating **random variables** – i.e. samples of numbers which behave as if they are drawn from some particular pdf (e.g. uniform, Gaussian, Poisson etc).

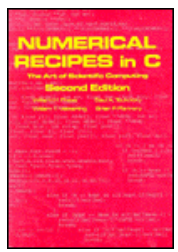
We call these **Monte Carlo methods**

From Cole et al. (1998)

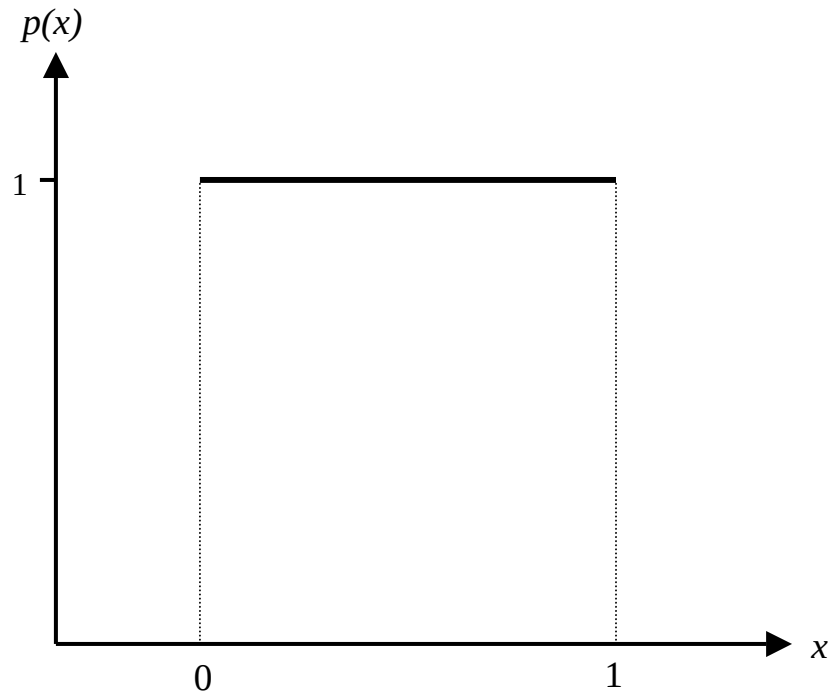
4.1 Uniform random numbers

Generating uniform random numbers, drawn from the pdf U[0,1], is fairly easy. Any scientific Calculator will have a RAN function…

Better examples of U[0,1] random number generators can be found in Numerical Recipes.

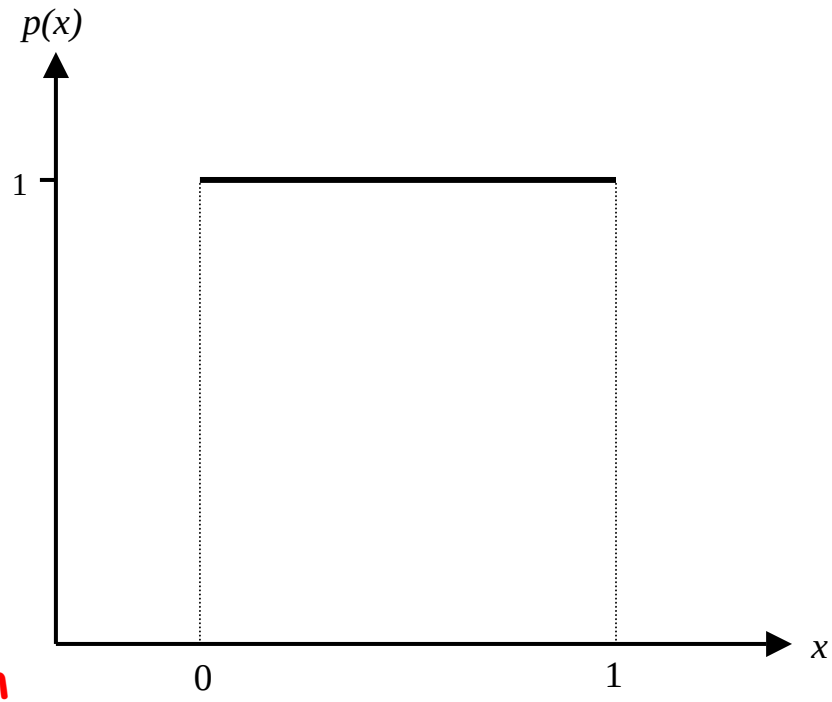


http://www.numerical-recipes.com/

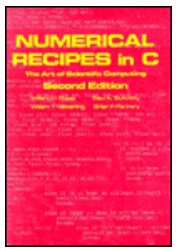


4.1 Uniform random numbers

Generating uniform random numbers, drawn from the pdf U[0,1], is fairly easy. Any scientific Calculator will have a RAN function…



Better examples of U[0,1] random number generators can be found in Numerical Recipes.



http://www.numerical-recipes.com/

In what sense are they better?…

Algorithms only generate **pseudo-random** numbers: very long (deterministic) sequences of numbers which are approximately random (i.e. no discernible pattern).

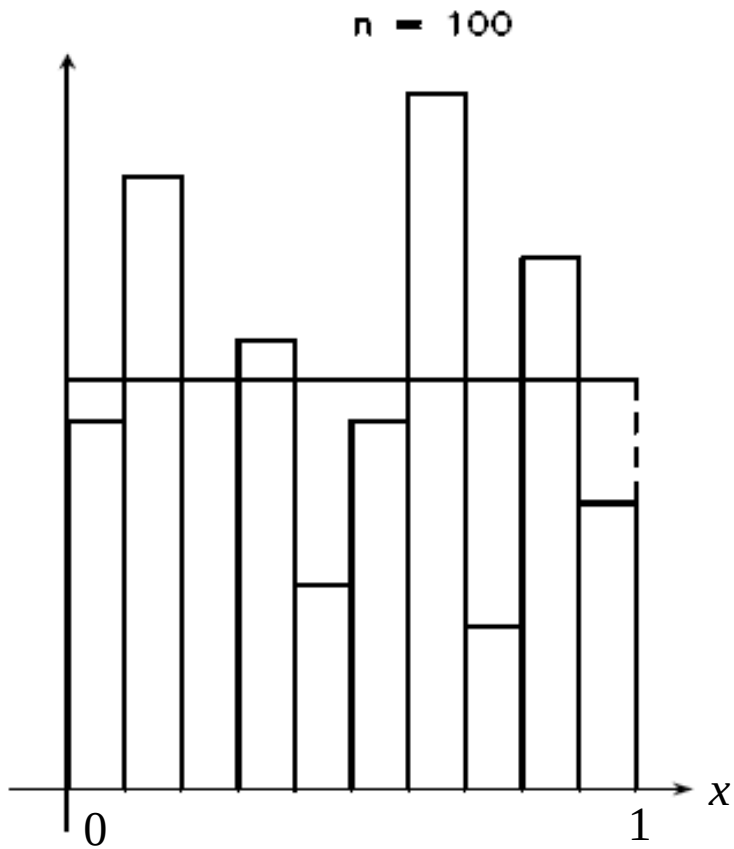
The better the RNG, the better it approximates U[0,1]

We can test pseudo-random numbers for randomness in several ways:

(a) Histogram of sampled values.

We can use hypothesis tests to see if the sample is consistent with the pdf we are trying to model.

e.g. Chi-squared test, applied to the
to the numbers in each histogram bin.



We can test pseudo-random numbers for randomness in several ways:

(a) Histogram of sampled values.

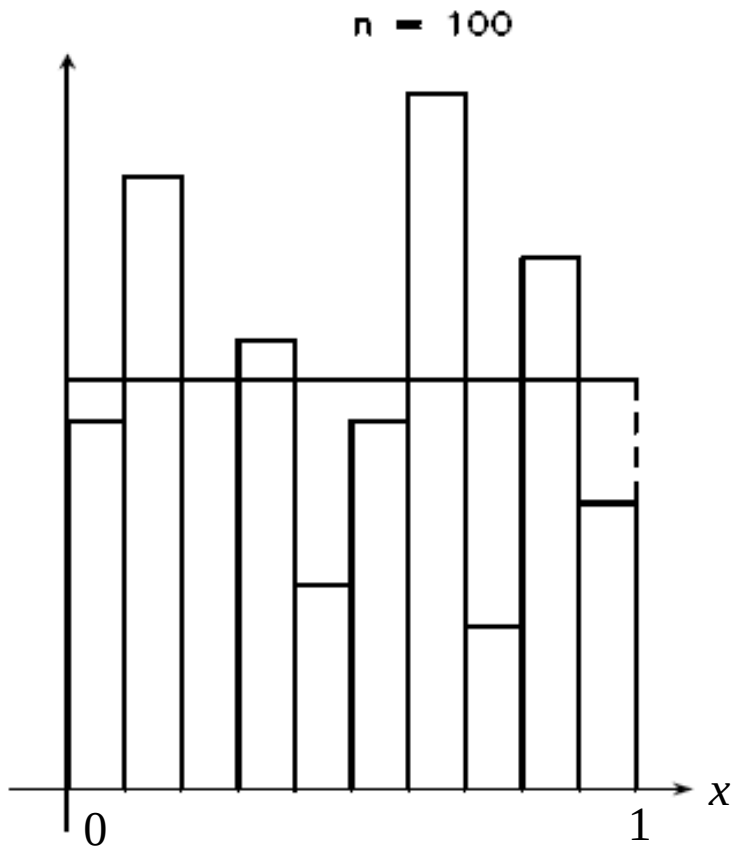
We can use hypothesis tests to see if the sample is consistent with the pdf we are trying to model.

e.g. Chi-squared test, applied to the to the numbers in each histogram bin.

$$\chi^2 = \sum_{i=1}^{n_{\text{bin}}} \left(\frac{n_i^{\text{obs}} - n_i^{\text{pred}}}{\sigma_i} \right)^2 \qquad (4.1)$$

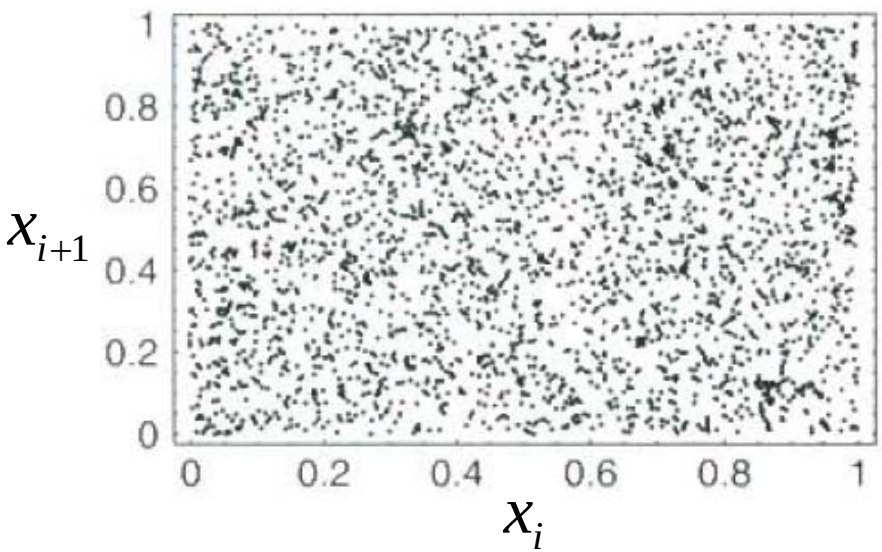
Assume the bin number counts are subject to Poisson fluctuations, so that $\sigma_i^2 = n_i^{\text{pred}}$

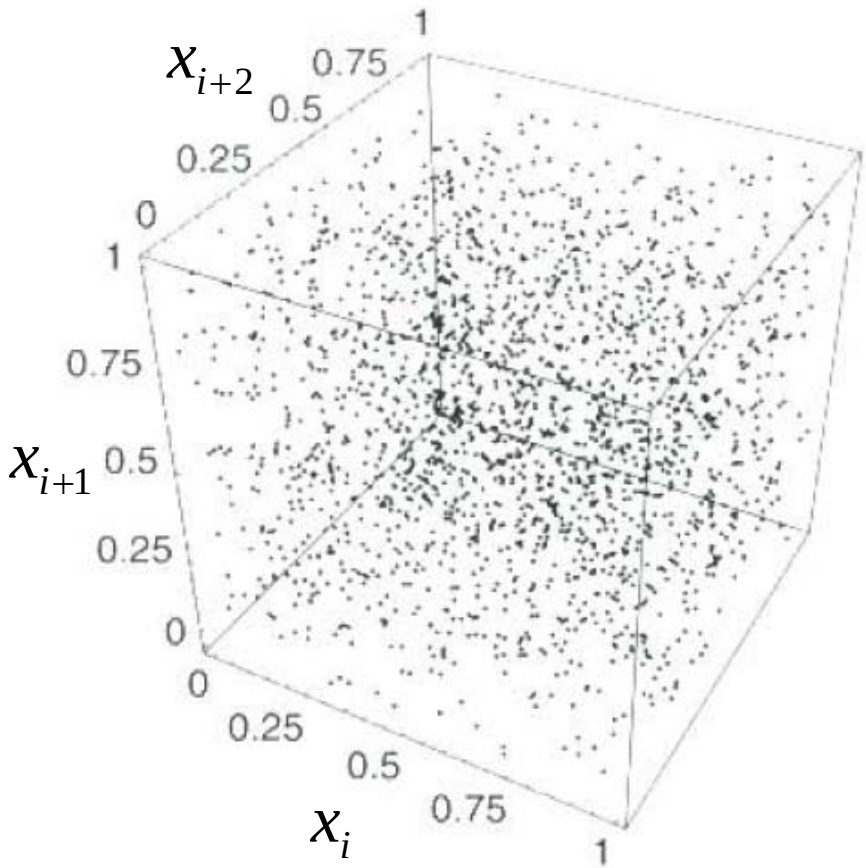
Note: no. of degrees of freedom $= n_{\text{bin}} - 1$ since we know the total sample size.



(b) Correlations between neighbouring pseudo-random numbers

Sequential patterns in the sampled values would show up as structure in the phase portraits – scatterplots of the i^{th} value against the $(i+1)^{th}$ value etc.





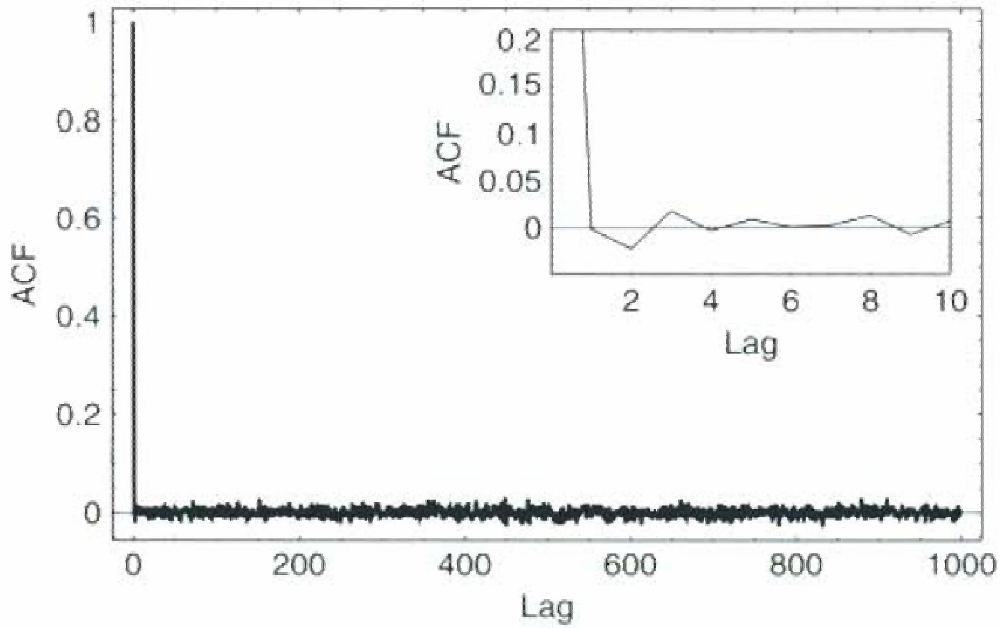
(b) Correlations between neighbouring pseudo-random numbers

Sequential patterns in the sampled values would show up as structure in the phase portraits – scatterplots of the i^{th} value against the $(i+1)^{th}$ value etc.

We can compute the Auto-correlation function

$$\rho(j) = \frac{\sum\left[(x_i - \bar{x})(x_{i+j} - \bar{x})\right]}{\sqrt{\sum(x_i - \bar{x})^2}\sqrt{\sum(x_{i+j} - \bar{x})^2}} \qquad (4.2)$$

j is known as the Lag



If the sequence is uniformly random, we expect

$$\begin{cases} \rho(j) = 1 & \text{for} \quad j = 0 \\ \rho(j) = 0 & \text{otherwise} \end{cases}$$

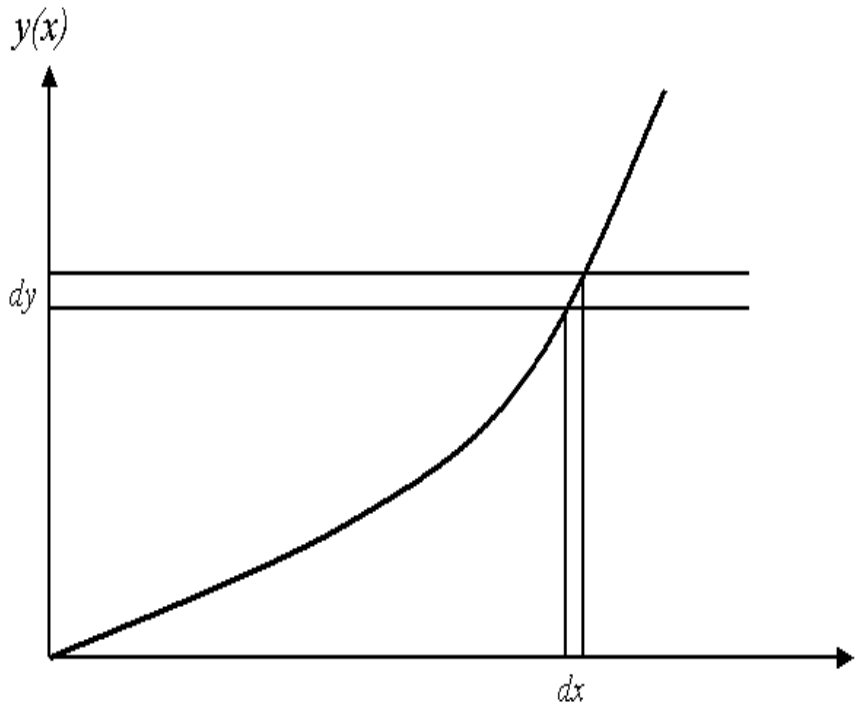
4.2 Variable transformations

Generating random numbers from other pdfs can be done by transforming random numbers drawn from simpler pdfs.

The procedure is similar to changing variables in integration.

Suppose, e.g. $x \sim p(x)$

Let $y = y(x)$ be **monotonic**



Then $$p(y)dy = p(x)dx \quad (4.3)$$

Probability of number between y and $y+dy$

Probability of number between x and $x+dx$

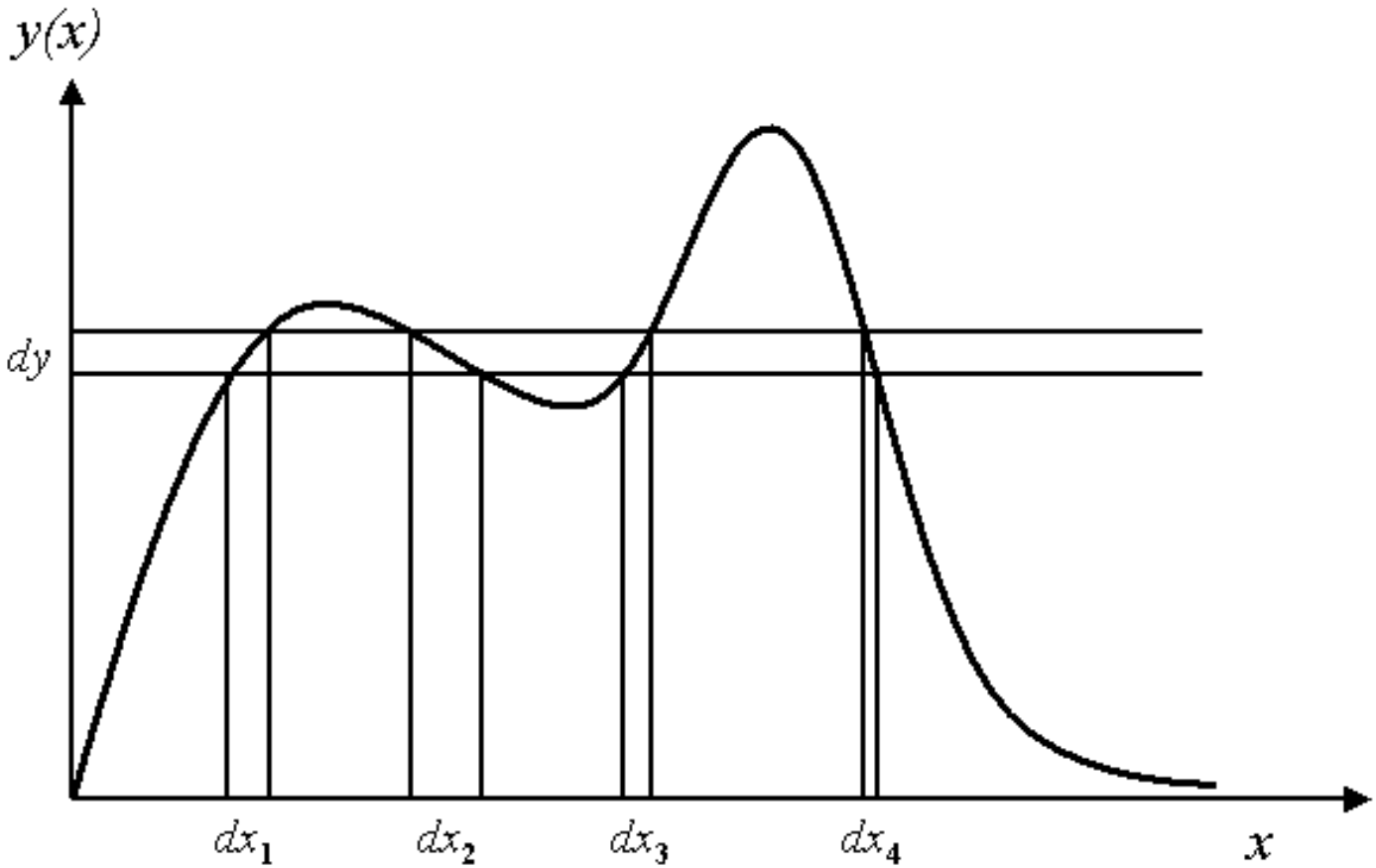
$$p(y) = \frac{p(x(y))}{\left|dy/dx\right|} \quad (4.4)$$

Because probability must be positive

We can extend the expression given in eq. (4.4) to the case where $y(x)$ is not monotonic, by calculating

$$p(y)dy = \sum_i p(x_i)dx_i \quad \text{so that}$$

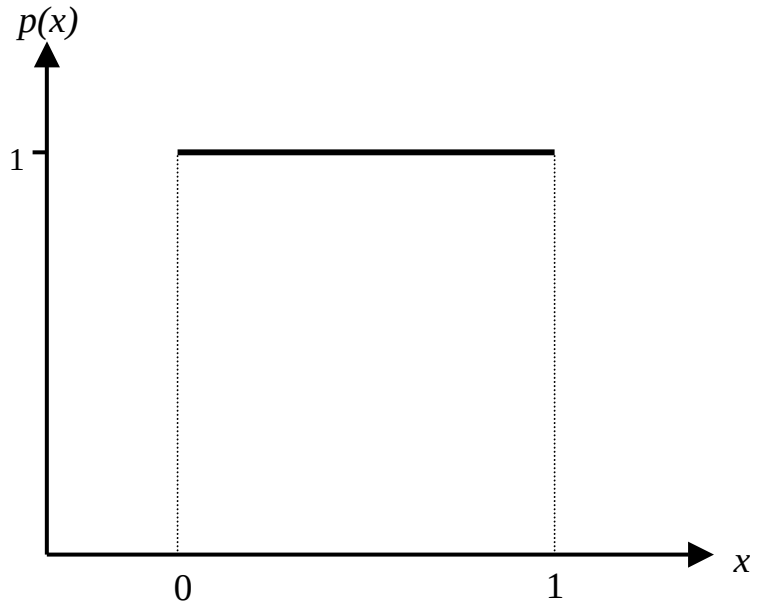
$$p(y) = \sum_i \frac{p(x_i(y))}{\left|dy/dx_i\right|} \tag{4.5}$$



Example 1

Suppose we have $x \sim U[0,1]$

Then $$p(x) = \begin{cases} 1 & \text{for} \quad 0 < x < 1 \\ 0 & \text{otherwise} \end{cases} \qquad (4.6)$$

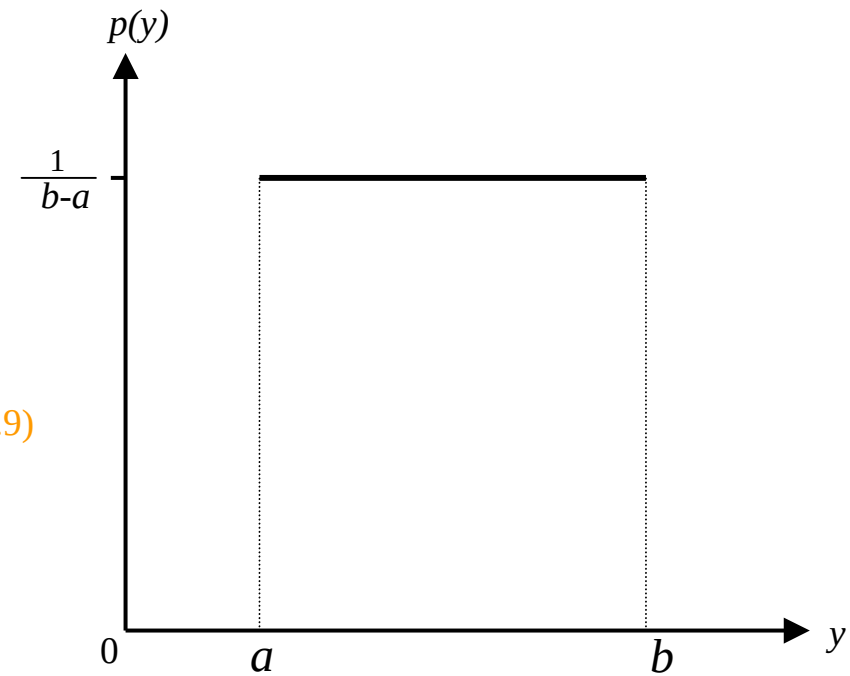


Define $$y = a + (b-a)x \qquad (4.7)$$

So $$\frac{dy}{dx} = (b-a) \qquad (4.8)$$

i.e. $$p(y) = \begin{cases} 1/b-a & \text{for} \quad a < y < b \\ 0 & \text{otherwise} \end{cases} \qquad (4.9)$$

or $y \sim U[a,b]$



Normal pdf with mean zero and standard deviation unity

Example 2

Numerical recipes provides a program to turn $x \sim U[0,1]$ into $y \sim N[0,1]$

Suppose we want $z \sim N[\mu, \sigma]$

We define $z = \mu + \sigma y$ (4.10) so that $\frac{dz}{dy} = \sigma$ (4.11)

Now $p(y) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2} y^2\right)$ (4.12)

so $p(z) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{1}{2}\left[\frac{z-\mu}{\sigma}\right]^2\right)$ (4.13)

Normal pdf with mean zero and standard deviation unity

Example 2

Numerical recipes provides a program to turn $x \sim U[0,1]$ into $y \sim N[0,1]$

Suppose we want $z \sim N[\mu, \sigma]$

We define $z = \mu + \sigma y$ (4.10) so that $\frac{dz}{dy} = \sigma$ (4.11)

Now $p(y) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2} y^2\right)$ (4.12)

so $p(z) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{1}{2}\left[\frac{z-\mu}{\sigma}\right]^2\right)$ (4.13)

Variable transformation formula also the basis for error propagation formulae we use in the lab. See example sheet 3 for more on this.

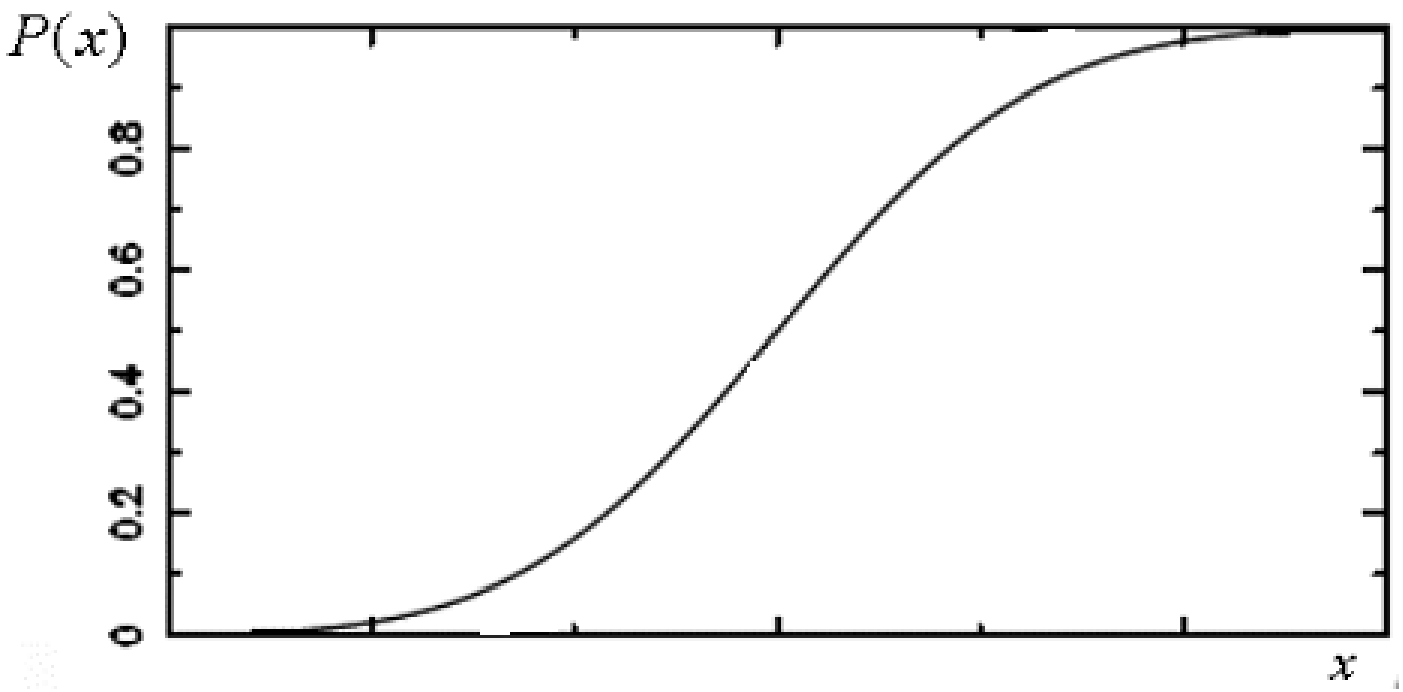
4.3 Probability integral transform

One particular variable transformation merits special attention.

Suppose we can compute the CDF of some desired random variable

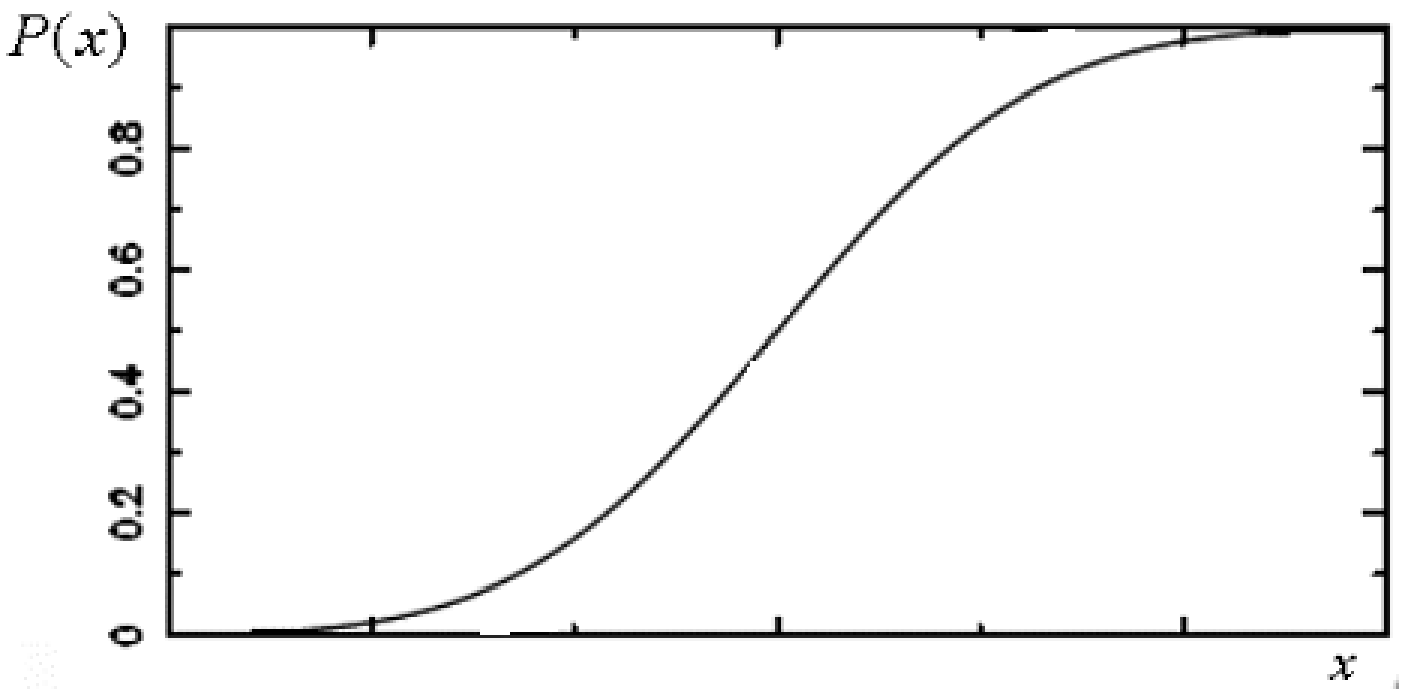
Cumulative distribution function (CDF)

$$P(a) = \int_{-\infty}^{a} p(x)dx = \text{Prob}(x < a)$$



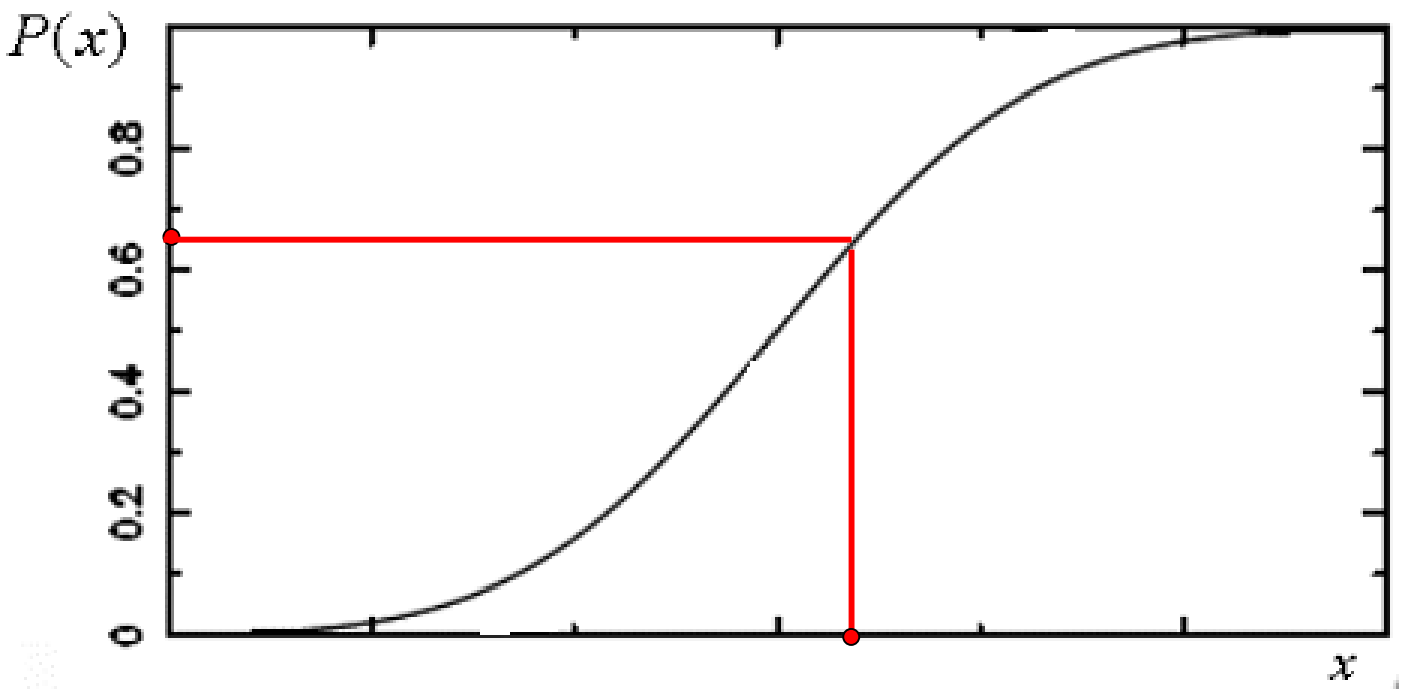
1) Sample a random variable $y \sim U[0,1]$

2) Compute x such that $y = P(x)$ i.e. $x = P^{-1}(y)$



1) Sample a random variable $y \sim U[0,1]$

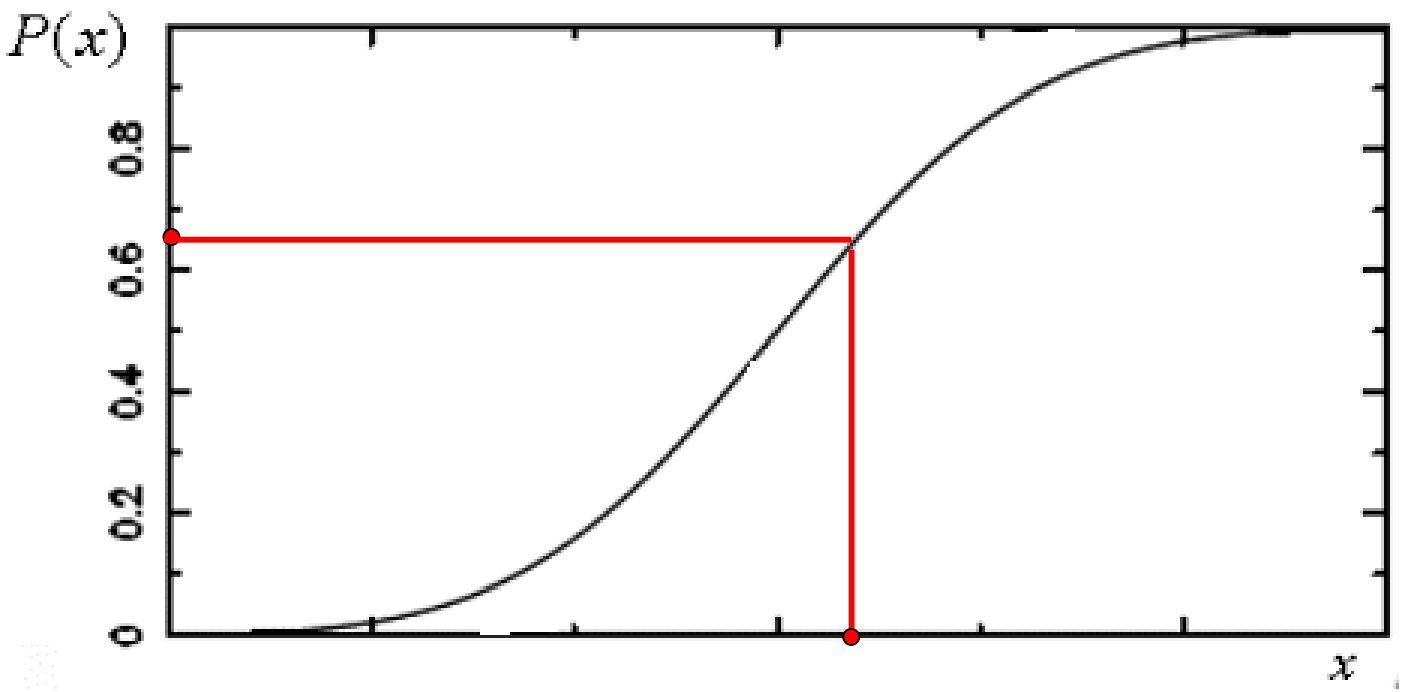
2) Compute x such that $y = P(x)$ i.e. $x = P^{-1}(y)$



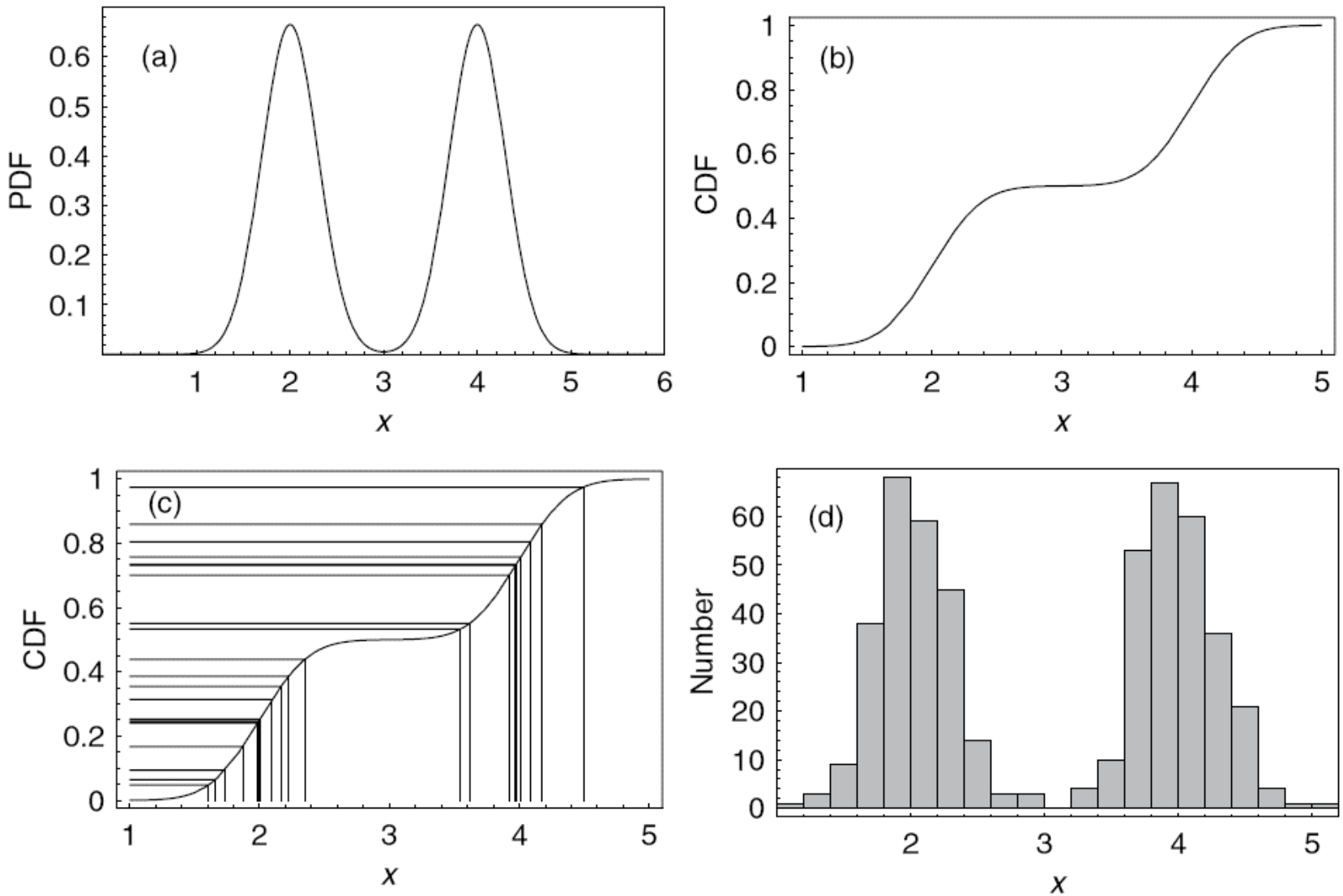
1) Sample a random variable $y \sim U[0,1]$

2) Compute x such that $y = P(x)$ i.e. $x = P^{-1}(y)$

3) Then $x \sim p(x)$ i.e. x is drawn from the pdf corresponding to the cdf $P(x)$



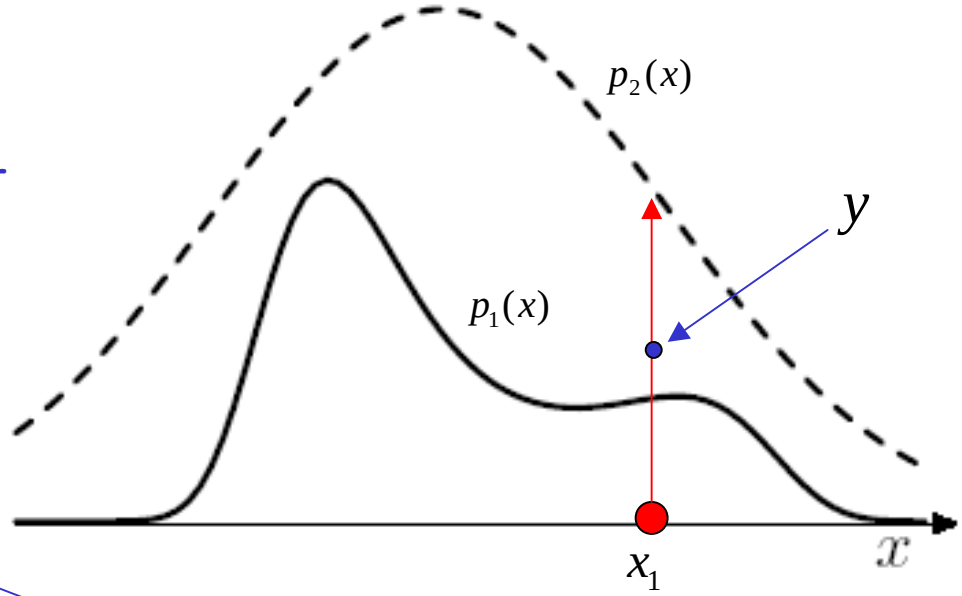
Example


(a)
PDF
0.6
0.5
0.4
0.3
0.2
0.1
1
2
3
4
5
6
x
(b)
CDF
1
0.8
0.6
0.4
0.2
0
1
2
3
4
5
x
(c)
CDF
1
0.8
0.6
0.4
0.2
0
1
2
3
4
5
x
(d)
Number
60
50
40
30
20
10
0
2
3
4
5
x

4.4 Rejection sampling

Suppose we want to sample from some pdf $p_1(x)$ and we know that

$$p_1(x) < p_2(x) \quad \forall x$$



1) Sample x_1 from $p_2(x)$

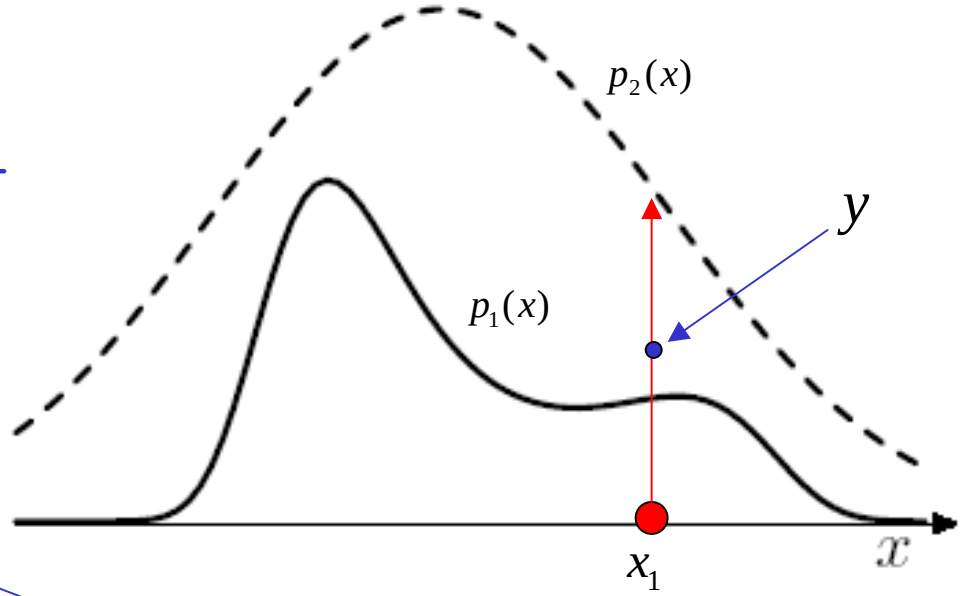
2) Sample $y \sim U[0, p_2(x_1)]$

(Suppose we have an 'easy' way to do this)

4.4 Rejection sampling

Suppose we want to sample from some pdf $p_1(x)$ and we know that

$$p_1(x) < p_2(x) \quad \forall x$$



1) Sample x_1 from $p_2(x)$

2) Sample $y \sim U[0, p_2(x_1)]$

(Suppose we have an 'easy' way to do this)

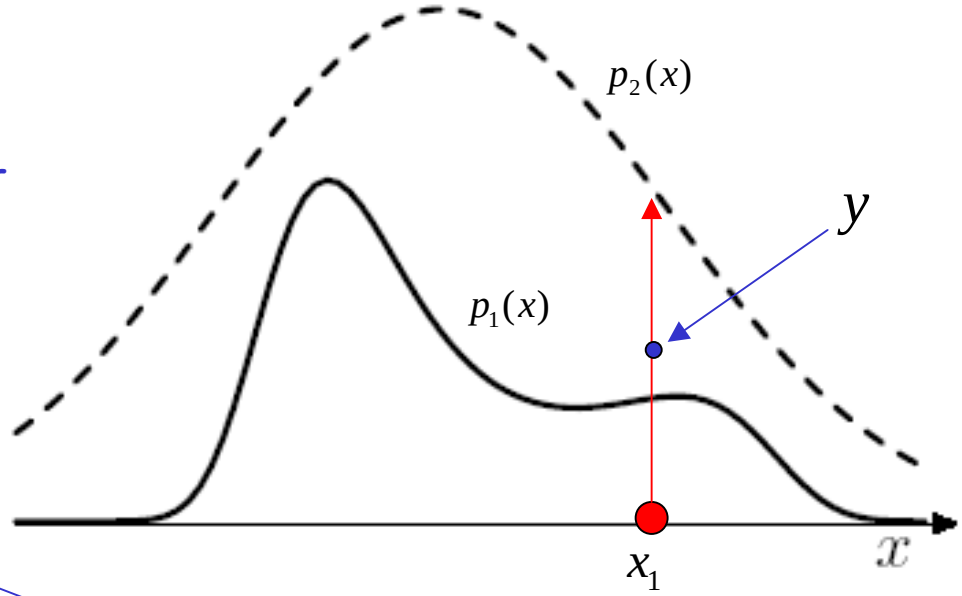
3) If $y < p_1(x)$ **ACCEPT**

otherwise **REJECT**

4.4 Rejection sampling

Suppose we want to sample from some pdf $p_1(x)$ and we know that

$$p_1(x) < p_2(x) \quad \forall x$$



1) Sample x_1 from $p_2(x)$

2) Sample $y \sim U[0, p_2(x_1)]$

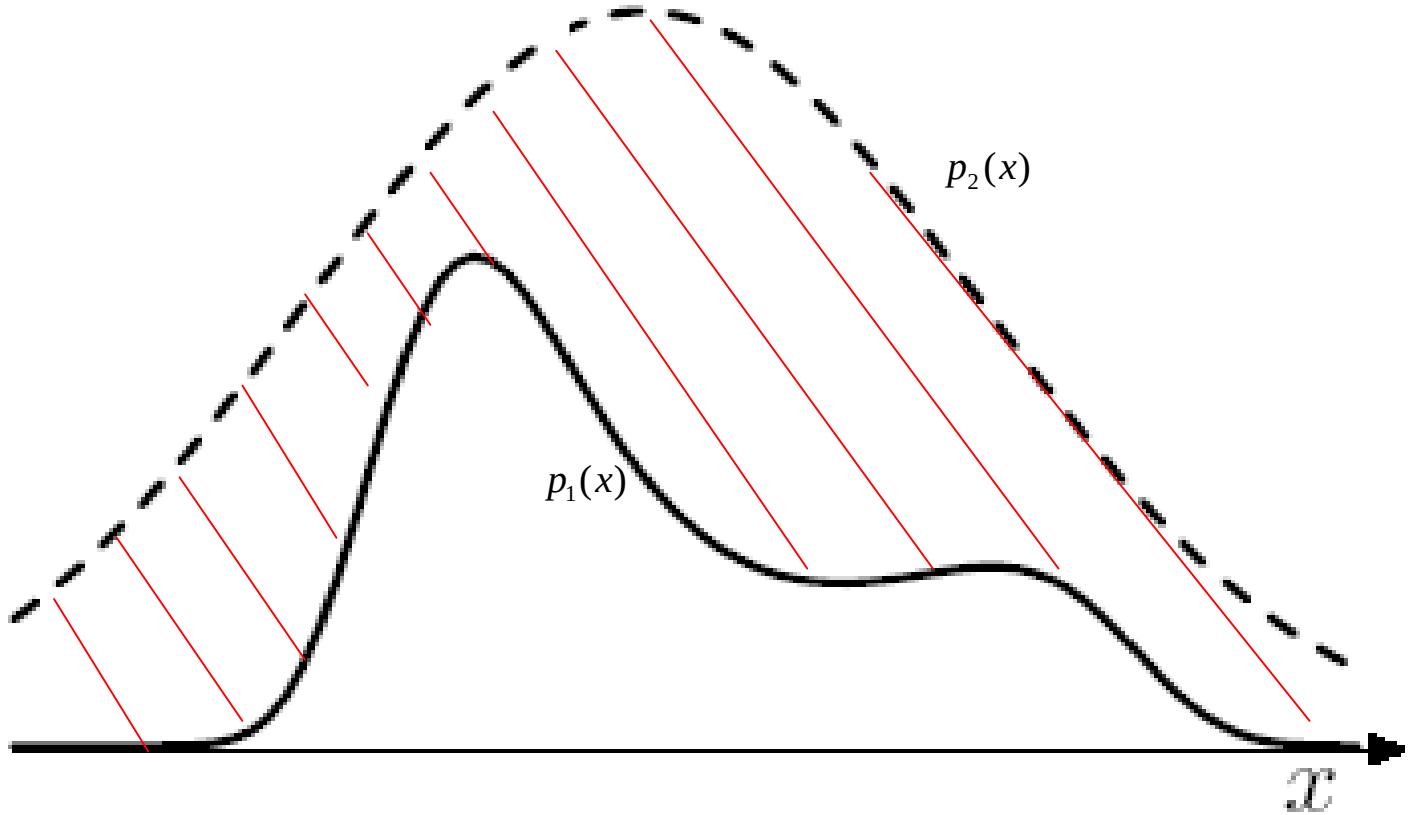
(Suppose we have an 'easy' way to do this)

3) If $y < p_1(x)$ **ACCEPT**

otherwise **REJECT**

Set of accepted values $\{x_i\}$ *are a sample from* $p_1(x)$

4.4 Rejection sampling



Method can be very slow if the shaded region is too large.

Ideally we want to find a pdf $p_2(x)$ that is: (a) easy to sample from

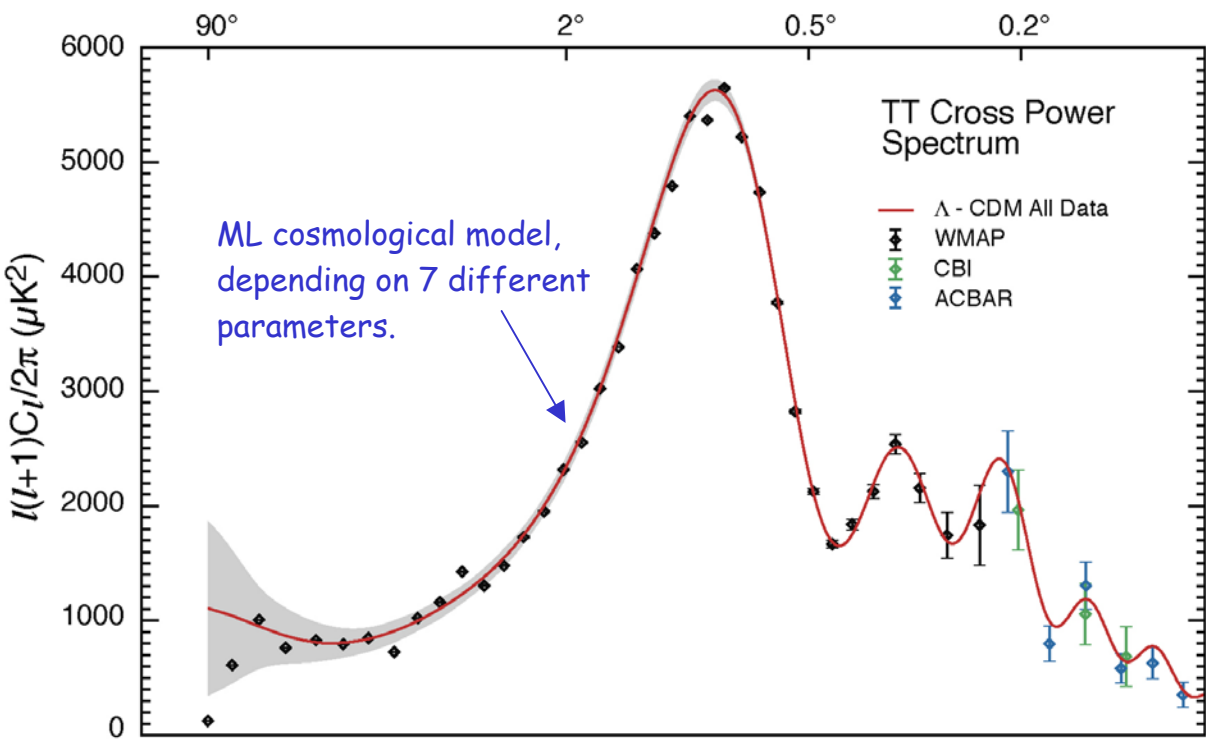
(b) close to $p_1(x)$

4.5 Markov Chain Monte Carlo

This is a very powerful, new (at least in astronomy!) method for sampling from pdfs. (These can be complicated and/or of high dimension).

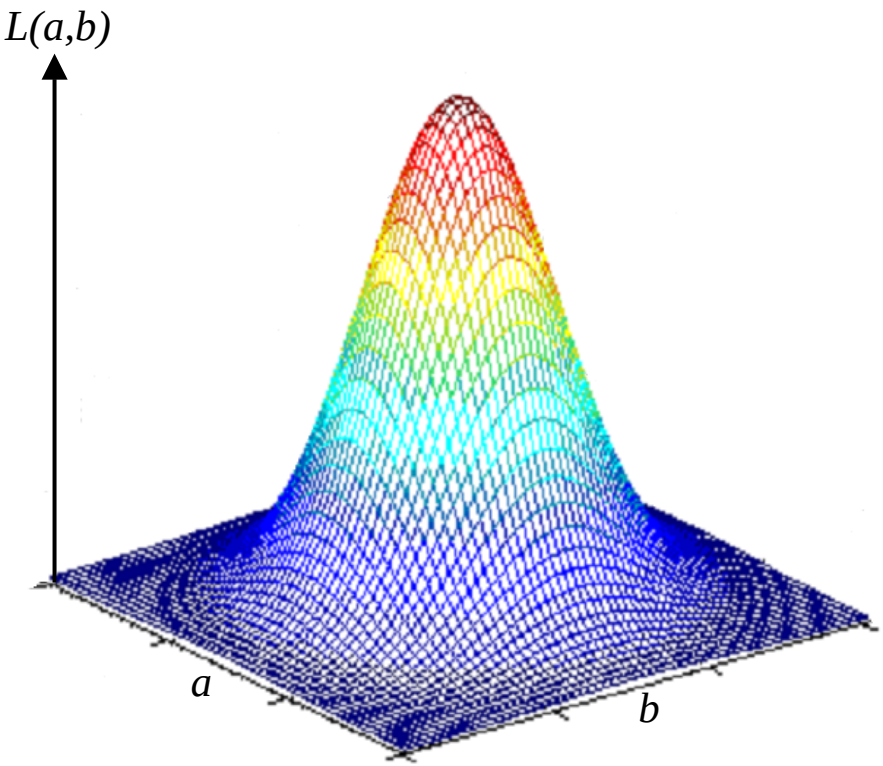
MCMC widely used e.g. in cosmology to determine 'maximum likelihood' model to CMBR data.

Angular power spectrum of CMBR temperature fluctuations



Consider a 2-D example (e.g. bivariate normal distribution);
Likelihood function depends on parameters a and b.

Suppose we are trying to find the maximum of $L(a,b)$



1) Start off at some randomly chosen value (a_1, b_1)

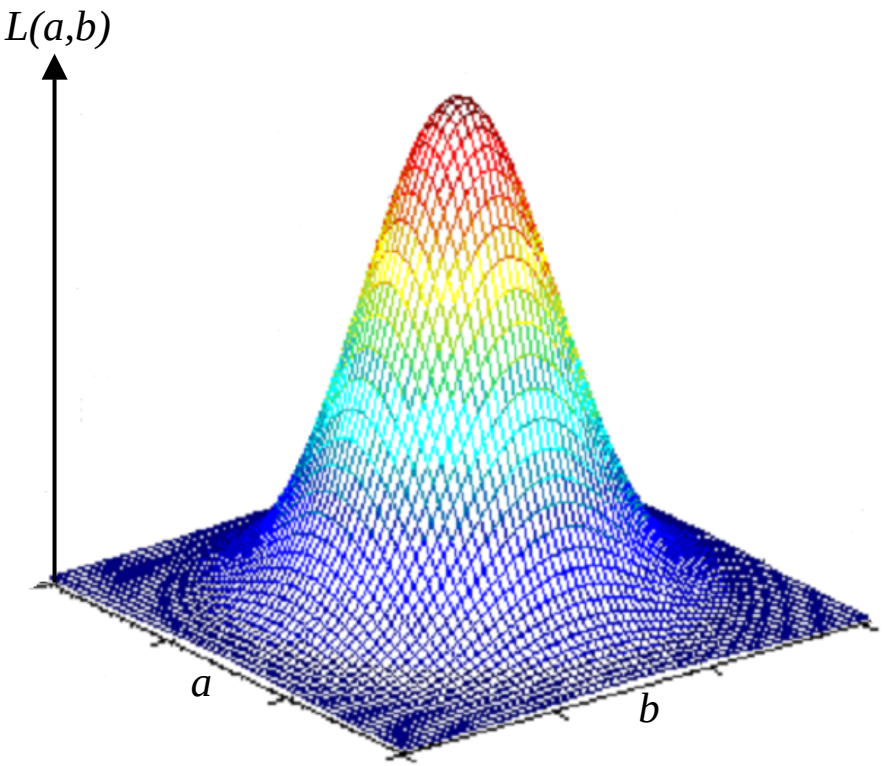
2) Compute $L(a_1, b_1)$ and gradient $\left(\frac{\partial L}{\partial a}, \frac{\partial L}{\partial b}\right)_{(a_1,b_1)}$

3) Move in direction of steepest +ve gradient – i.e. $L(a_1, b_1)$ is increasing fastest

4) Repeat from step 2 until (a_n, b_n) converges on maximum of likelihood

Consider a 2-D example (e.g. bivariate normal distribution);
Likelihood function depends on parameters a and b.

Suppose we are trying to find the maximum of $L(a,b)$



1) Start off at some randomly chosen value (a_1, b_1)

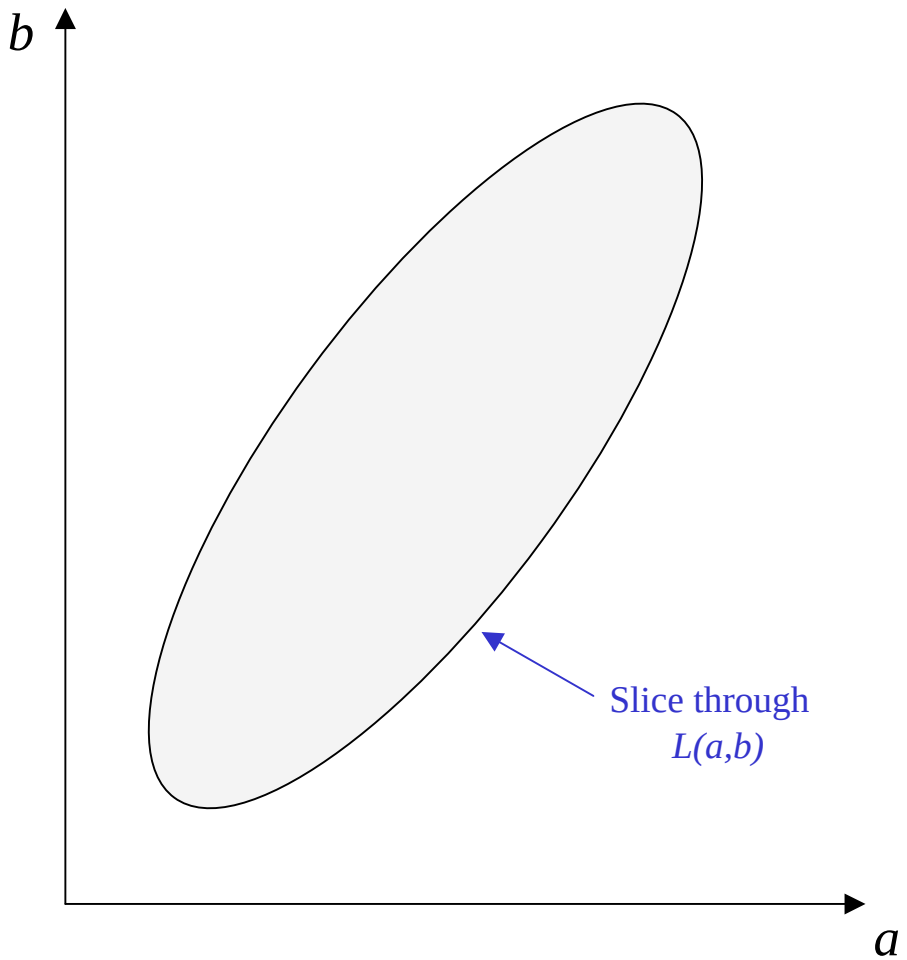
2) Compute $L(a_1, b_1)$ and gradient $\left(\frac{\partial L}{\partial a}, \frac{\partial L}{\partial b}\right)_{(a_1,b_1)}$

3) Move in direction of steepest +ve gradient – i.e. $L(a_1, b_1)$ is increasing fastest

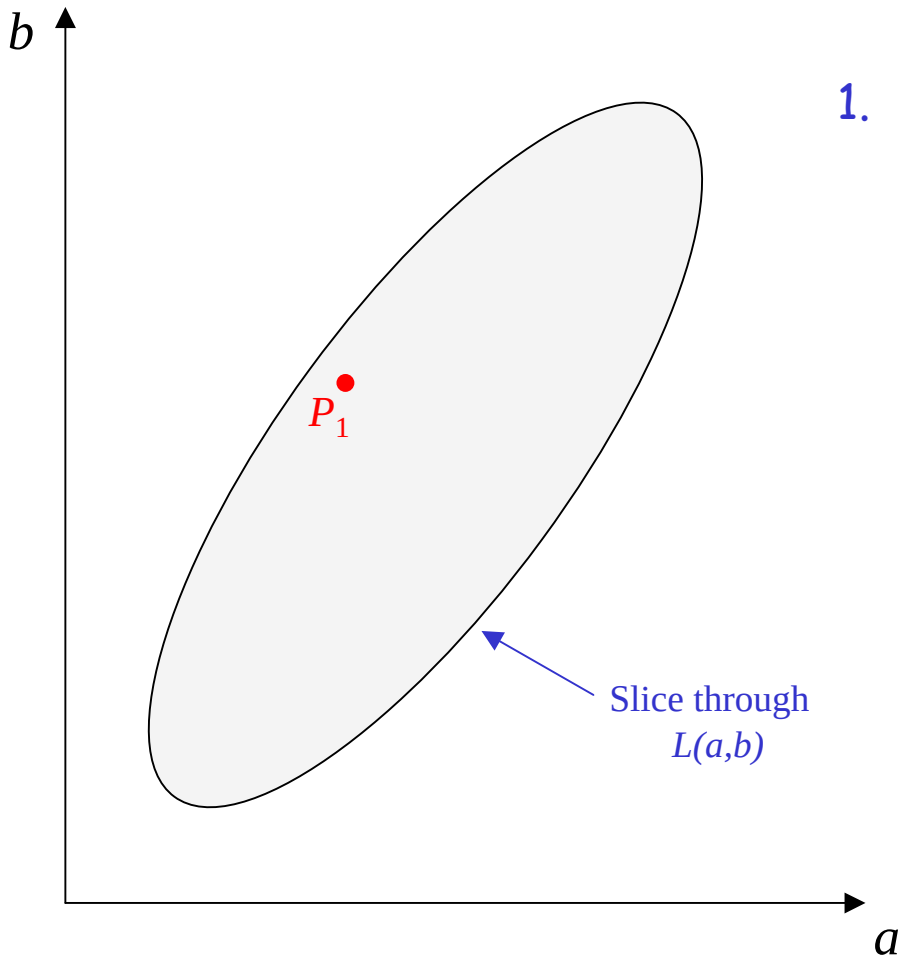
4) Repeat from step 2 until (a_n, b_n) converges on maximum of likelihood

OK for finding maximum, but not for generating a sample from $L(a,b)$ or for determining errors on the the ML parameter estimates.

MCMC provides a simple **Metropolis algorithm** for generating random samples of points from $L(a,b)$

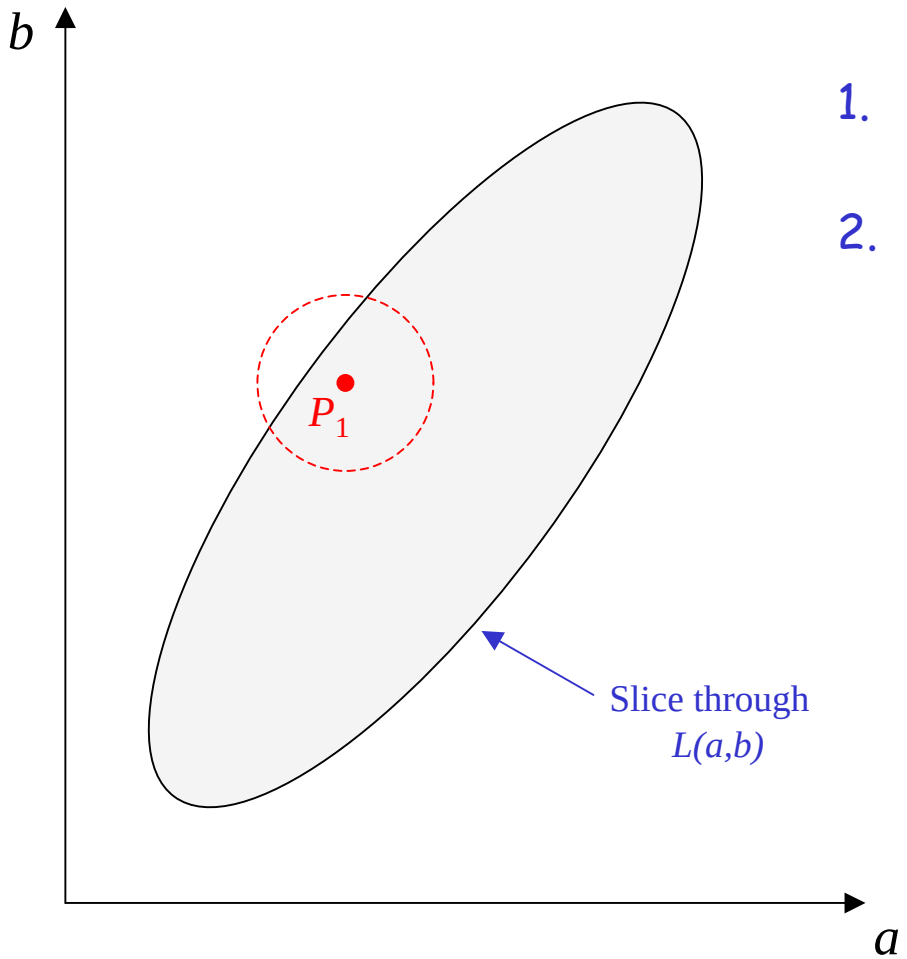


MCMC provides a simple **Metropolis algorithm** for generating random samples of points from $L(a,b)$



1. Sample random initial point $P_1 = (a_1, b_1)$

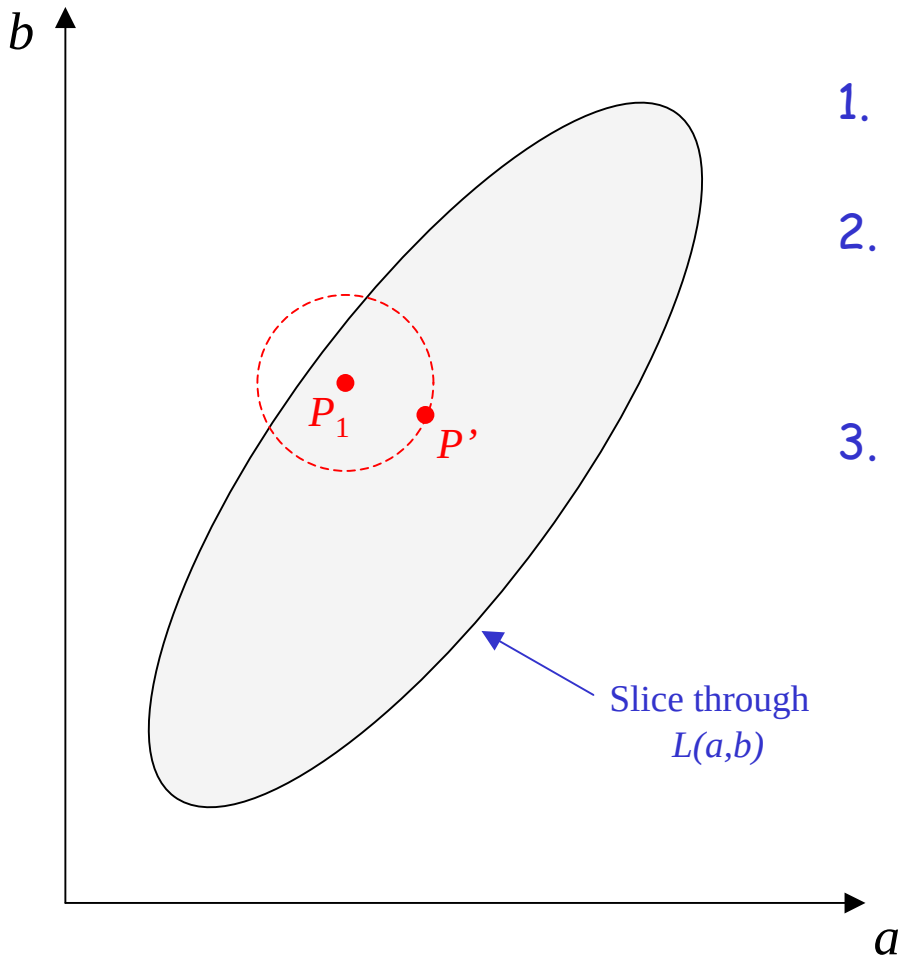
MCMC provides a simple **Metropolis algorithm** for generating random samples of points from $L(a,b)$



1. Sample random initial point $P_1 = (a_1, b_1)$

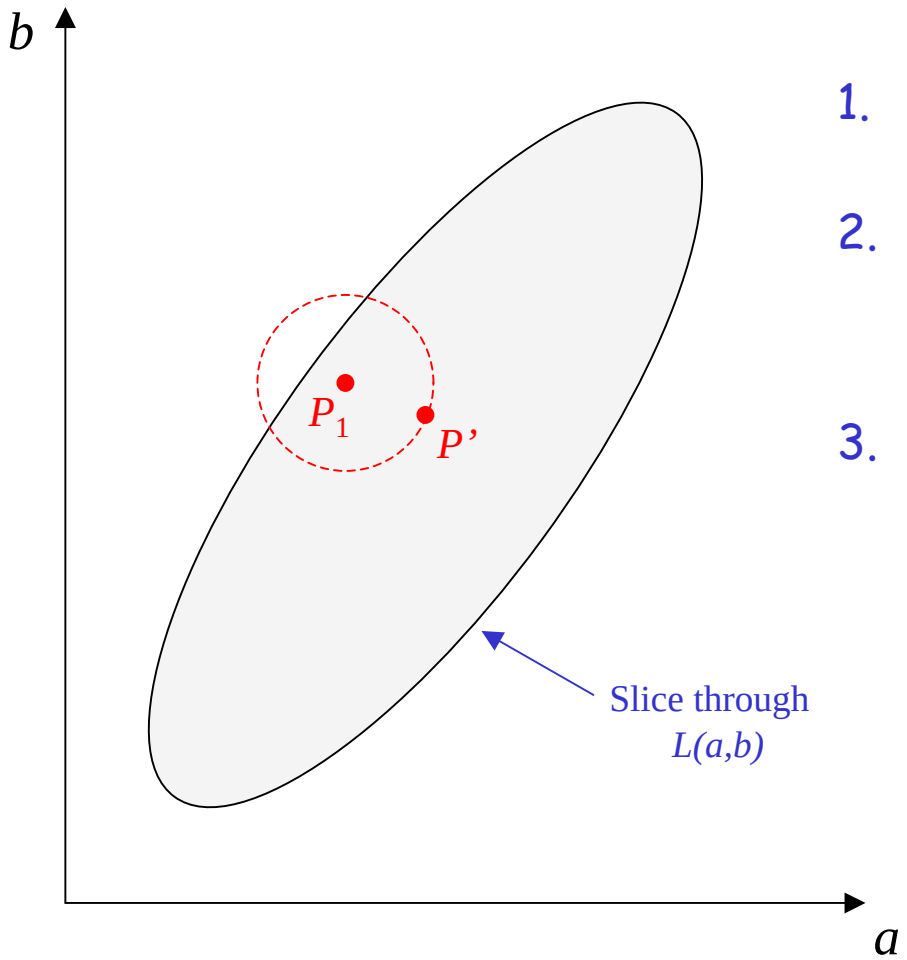
2. Centre a new pdf, Q, called the **proposal density**, on P_1

MCMC provides a simple **Metropolis algorithm** for generating random samples of points from $L(a,b)$



1. Sample random initial point $P_1 = (a_1, b_1)$
2. Centre a new pdf, Q, called the **proposal density**, on P_1
3. Sample tentative new point $P' = (a', b')$ from Q

MCMC provides a simple **Metropolis algorithm** for generating random samples of points from $L(a,b)$



1. Sample random initial point $P_1 = (a_1, b_1)$
2. Centre a new pdf, Q, called the **proposal density**, on P_1
3. Sample tentative new point $P' = (a', b')$ from Q
4. Compute $$R = \frac{L(a', b')}{L(a_1, b_1)}$$

5. If $R > 1$ this means P' is **uphill** from P_1.

We **accept** P' as the next point in our chain, i.e. $P_2 = P'$

6. If $R < 1$ this means P' is **downhill** from P_1.

In this case we ***may*** reject P' as our next point.

In fact, we accept P' with probability R.

How do we do this?...

(a) Generate a random number $x \sim U[0,1]$

(b) If $x < R$ then accept P' and set $P_2 = P'$

(c) If $x > R$ then reject P' and set $P_2 = P_1$

5. If $R > 1$ this means P' is **uphill** from P_1.

We **accept** P' as the next point in our chain, i.e. $P_2 = P'$

6. If $R < 1$ this means P' is **downhill** from P_1.

In this case we ***may*** reject P' as our next point.

In fact, we accept P' with probability R.

How do we do this?...

(a) Generate a random number $x \sim U[0,1]$

(b) If $x < R$ then accept P' and set $P_2 = P'$

(c) If $x > R$ then reject P' and set $P_2 = P_1$

Acceptance probability depends only on the previous point - ***Markov Chain***

So the Metropolis Algorithm generally (but not always) moves uphill, towards the peak of the Likelihood Function.

Remarkable facts

- Sequence of points $\{ P_1, P_2, P_3, P_4, P_5, \ldots \}$ represents a sample from the LF $L(a,b)$
- Sequence for each coordinate, e.g. $\{ a_1, a_2, a_3, a_4, a_5, \ldots \}$ samples the marginalised likelihood of a
- We can make a histogram of $\{ a_1, a_2, a_3, a_4, a_5, \ldots, a_n \}$ and use it to compute the mean and variance of a (i.e. to attach an error bar to a)

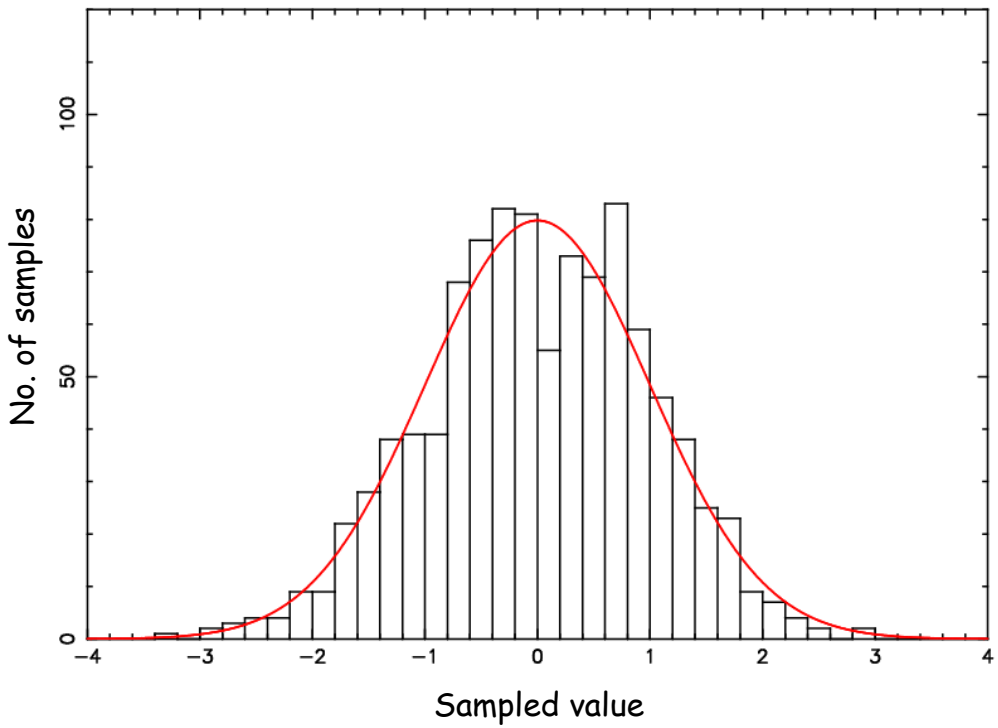
Why is this so useful?…

Suppose our LF was a 1-D Gaussian. We could estimate the mean and variance quite well from a histogram of e.g. 1000 samples.

But what if our problem is, e.g. 7 dimensional?

'Normal' sampling would need $(1000)^7$ samples!

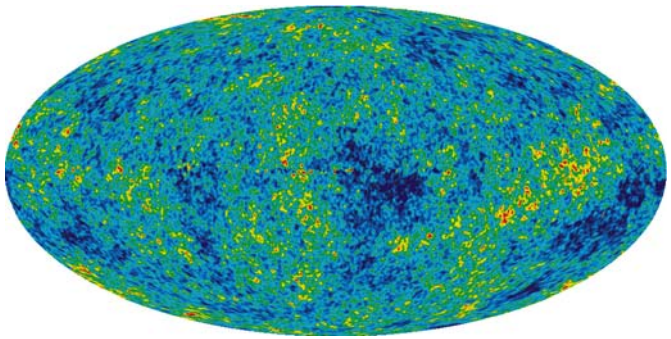
MCMC provides a short-cut.



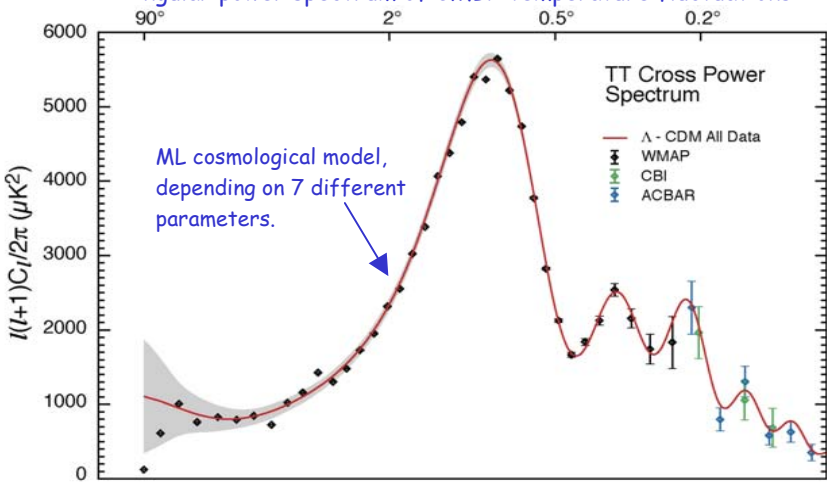
To compute a new point in our Markov Chain we need to compute the LF. But the computational cost does **not** grow so dramatically as we increase the number of dimensions of our problem.

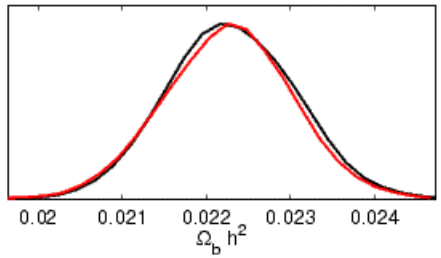
This lets us tackle problems that would be impossible by 'normal' sampling.

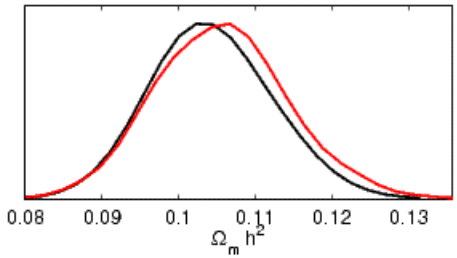
Example: CMBR constraints from WMAP 3 year data

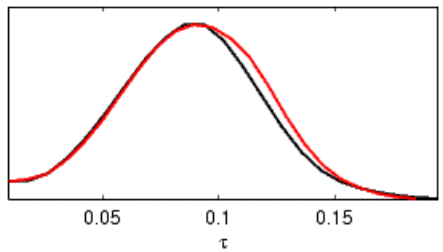


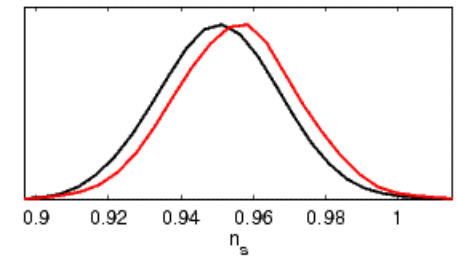
Angular power spectrum of CMBR temperature fluctuations

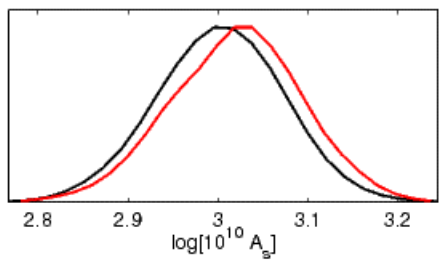


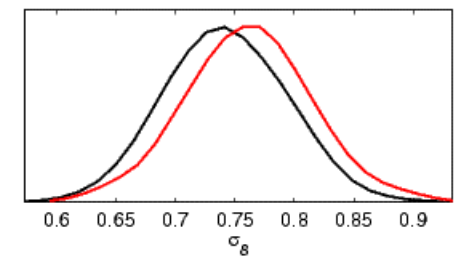


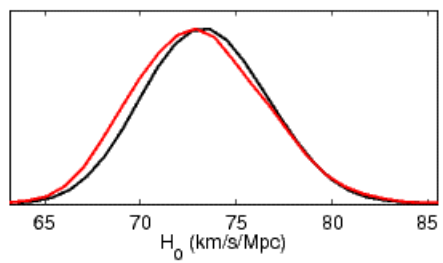












Possible Honours Lab Project!