Homework 6

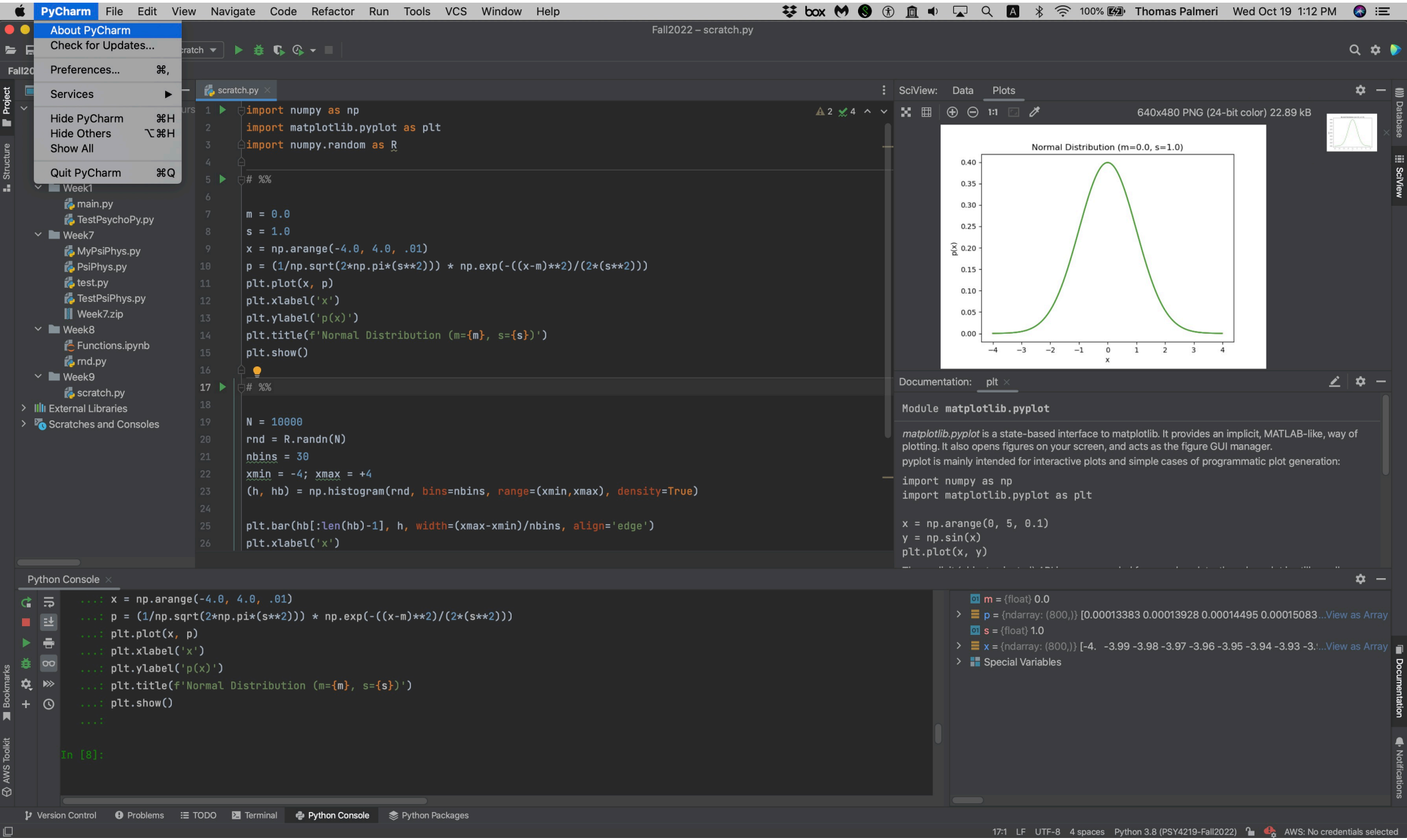
due Wed Oct 26 at start of class

NOTE that I have added a required question for graduate students (extra credit for undergraduates)

`Homework6.pdf`

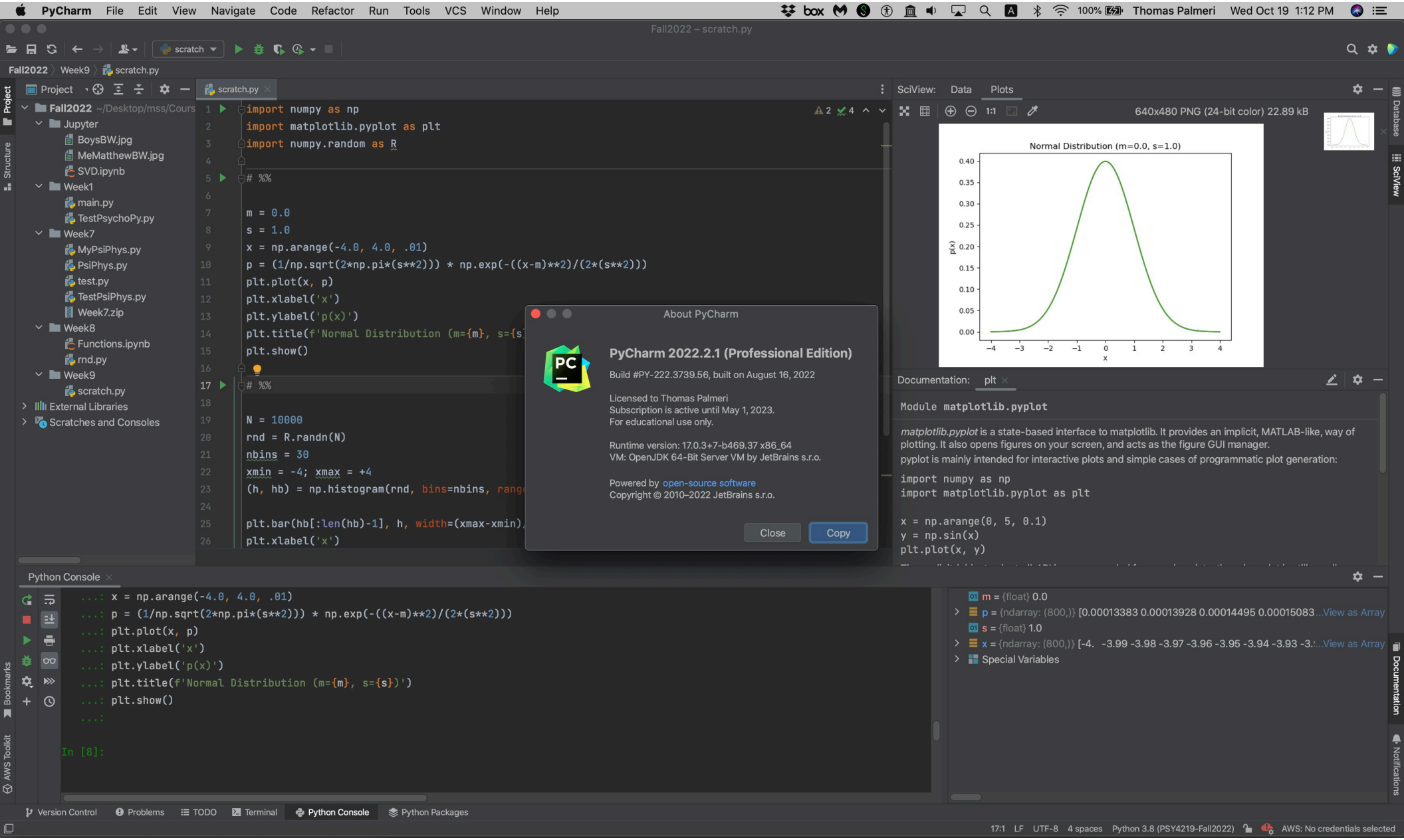
(free with a vanderbilt.edu email address - see instructions from early in the course)

use "Professional" version of PyCharm

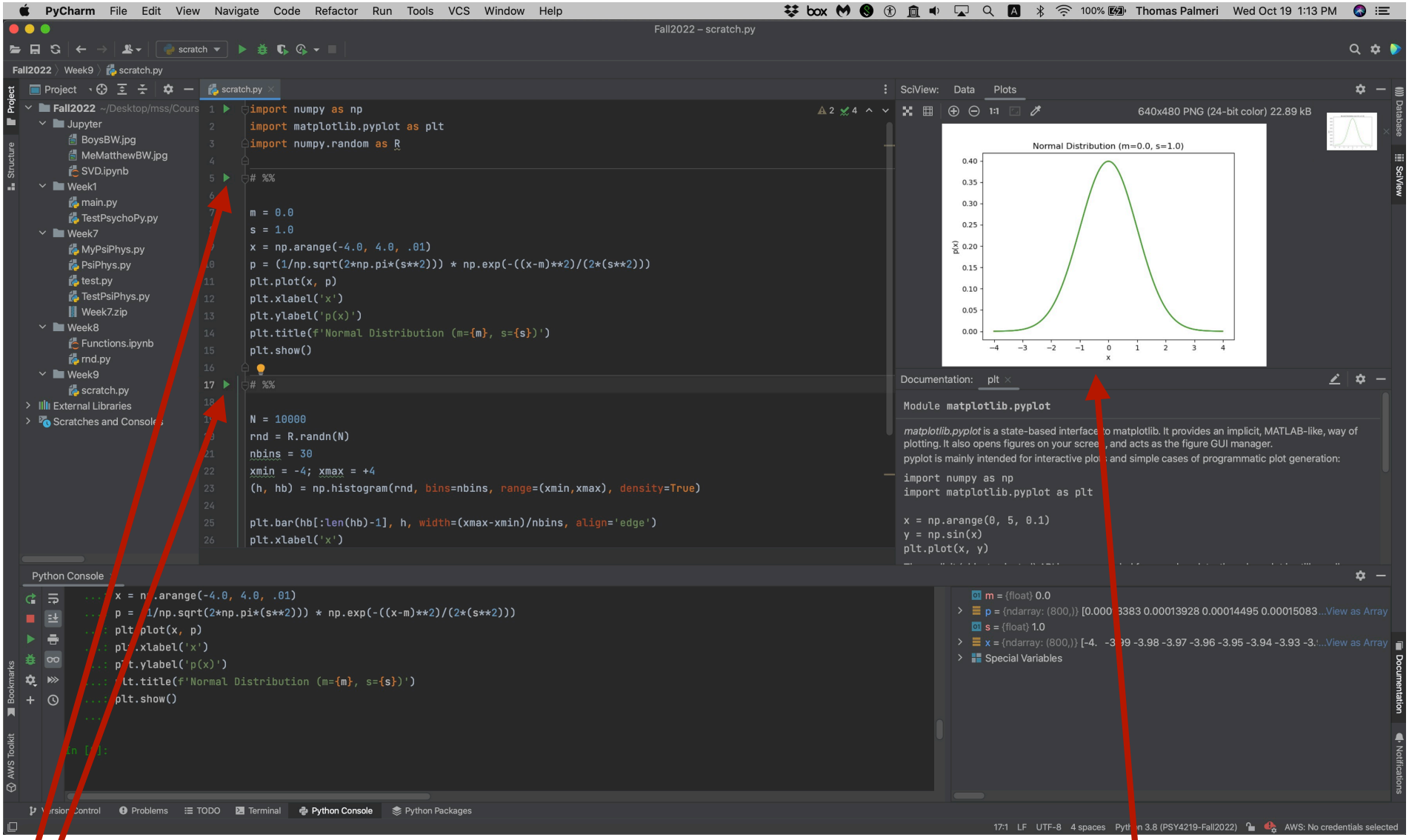


(free with a vanderbilt.edu email address - see instructions from early in the course)

use "Professional" version of PyCharm



running the free "Community" version of PyCharm means missing features

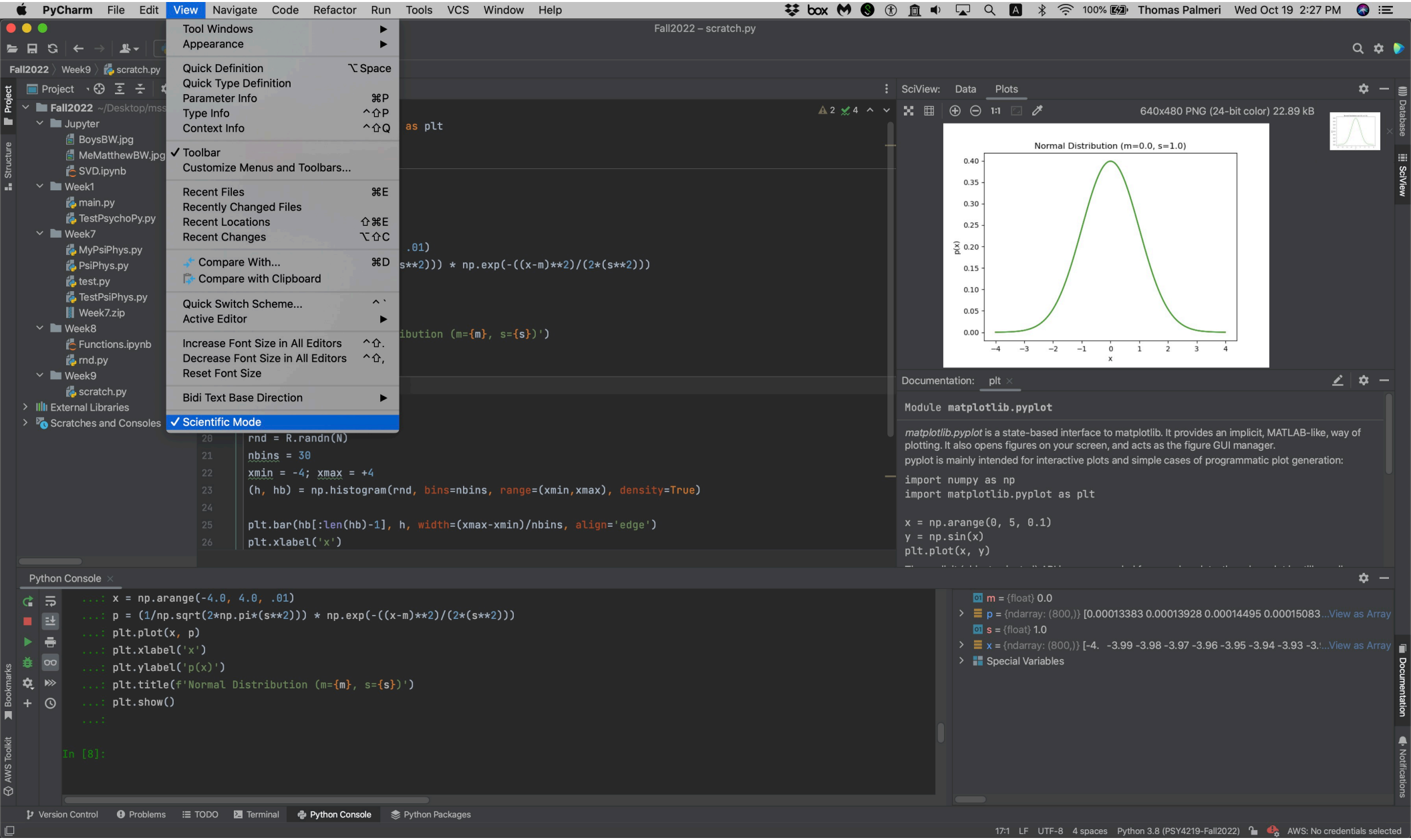


cell execution

SciView

(as well as lots of advanced features you may never use)

or check that you're running in "Scientific Mode"



download from Brightspace

`Random.ipynb`
`Visualization.ipynb`

generate random sample from distributions other than the uniform

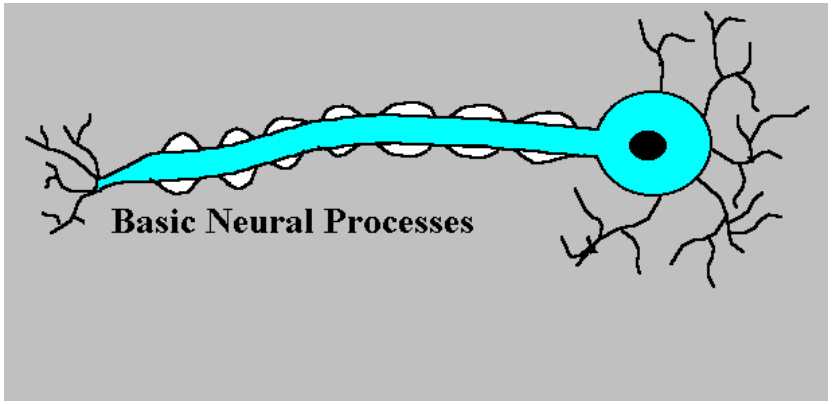
- **inverse transform sampling -** mathematically transform random numbers from a uniform to random numbers from another distribution (appropriate for mathematically simple probability distributions, like the exponential distribution)

- **other transformation methods -** specialized transform algorithm (e.g., Box-Muller transform for using samples from a uniform distribution to generate samples from a normal distribution)

- **rejection sampling algorithms -** Monte Carlo methods that can (in principle) be applied to any arbitrarily complex probability distribution (using random numbers to generate random numbers)

- **Markov Chain Monte Carlo methods -** simplest example is the Metropolis-Hastings algorithm (such methods are often associated with Bayesian statistics, but they are more general random number generation techniques beyond applications to Bayes)

Exponential Distribution



If you wanted to simulate some situation where the times between events (release of alpha particle, spike of a neuron, arrival of a person at a bus stop) you might want to sample random numbers from an exponential distribution.

It would make little sense to assume a uniform distribution (events do not happen uniformly in a fixed time interval) nor to assume a normal distribution (no negative time, not peaked like a normal)

random numbers from other probability distributions

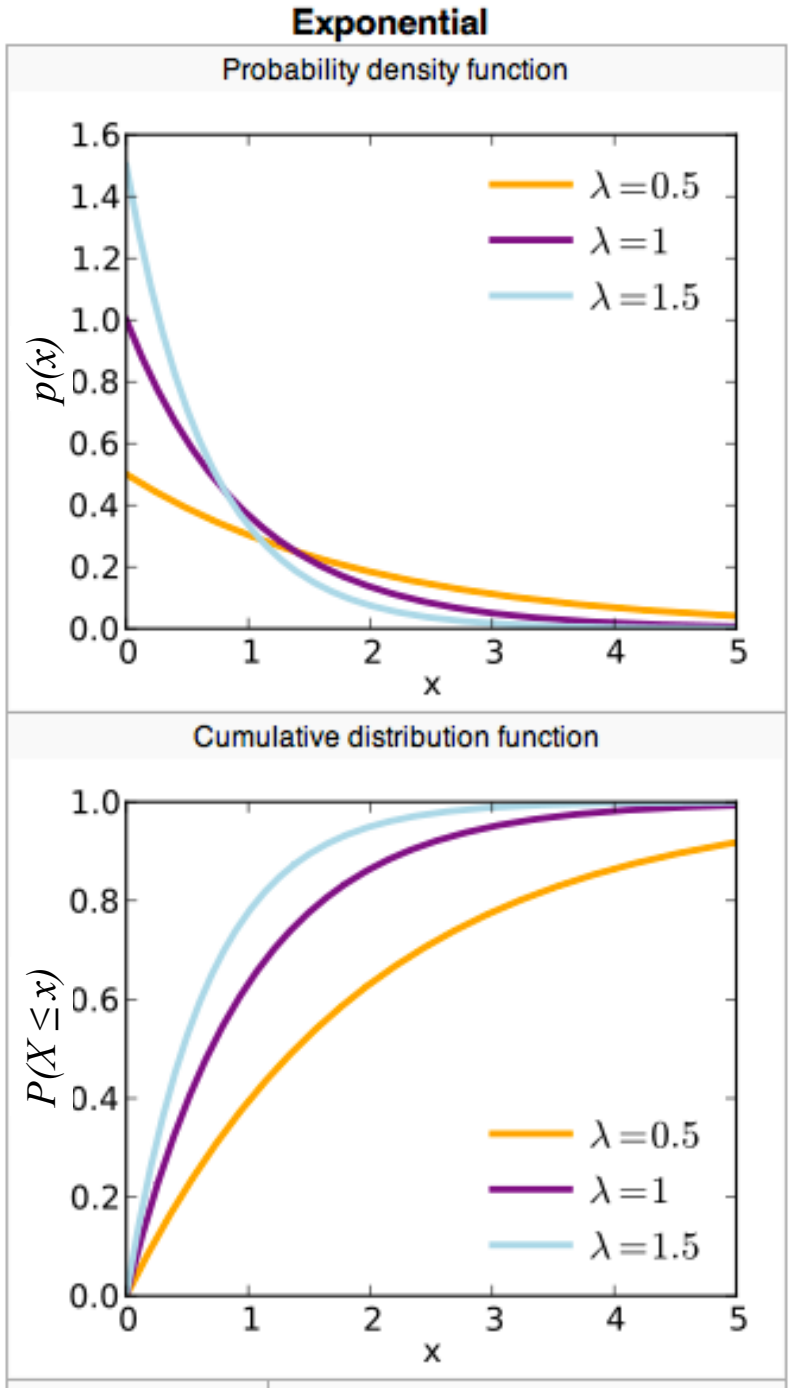
$$p(x) = \lambda \exp(-\lambda x)$$

Exponential Distribution:

time between events in what is called a “Poisson Process”

a Poisson distribution is the number of events observed within a certain time period where the time between events is exponentially distributed

Exponential Distribution (times between events) ⟷ Poisson Distribution (number of events)



random numbers from other probability distributions

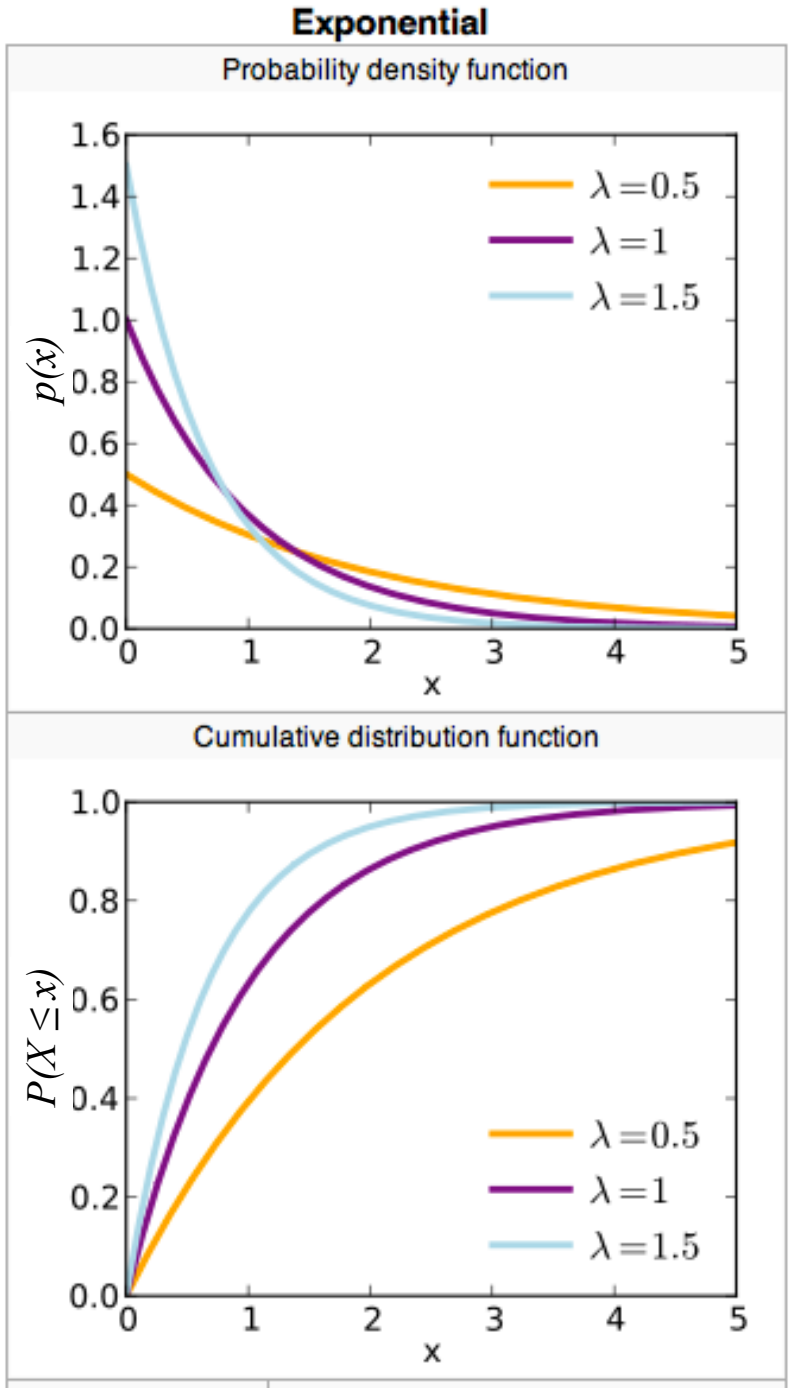
$$p(x) = \lambda \exp(-\lambda x)$$

Exponential Distribution:

time between events in what is called a “Poisson Process”

the only “memoryless” process in that the probability of an event occurring in the next infinitesimal time instant is the same irrespective of how much time has elapsed since the last event occurred ...

... used to model lots of natural processes for this reason



random numbers from other probability distributions

$$p(x) = \lambda \exp(-\lambda x)$$

let's create a PRNG that outputs random numbers from an exponential probability distribution

random numbers from other probability distributions

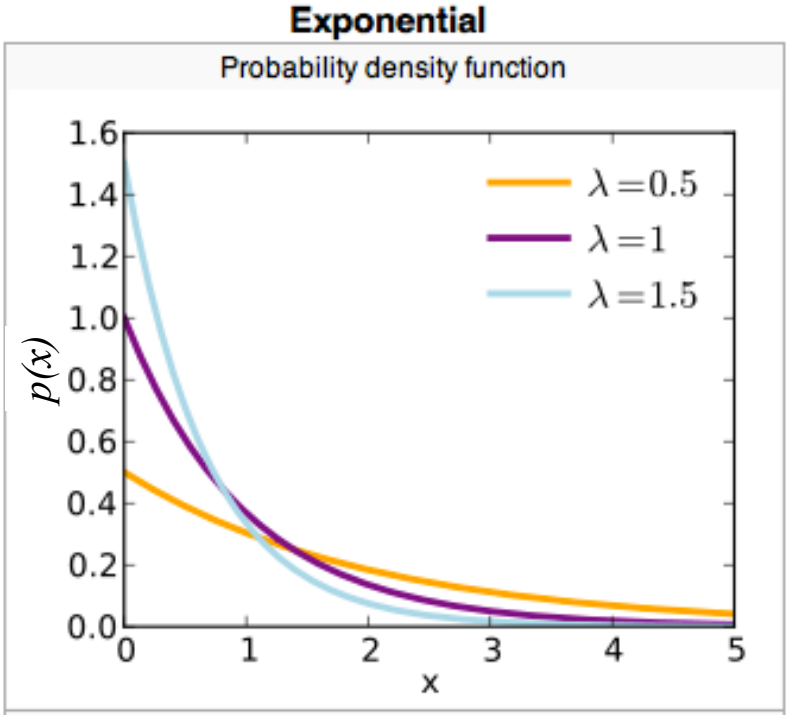
$$p(x) = \lambda \exp(-\lambda x)$$

let's create a PRNG that outputs random numbers from an exponential probability distribution

if you sample thousands and thousands of random numbers and create a histogram distribution it will look like the density function

Inverse Transform Sampling Method

$$p(x) = \lambda \exp(-\lambda x)$$



Inverse Transform Sampling Method

$$p(x) = \lambda \exp(-\lambda x)$$

$$\int_{-\infty}^{\infty} p(x)\, dx = 1$$

total area under a probability distribution equals 1

Inverse Transform Sampling Method

$$p(x) = \lambda \exp(-\lambda x)$$

$$\int_0^\infty p(x)\,dx = 1$$

exponential distribution is only positive values

Inverse Transform Sampling Method

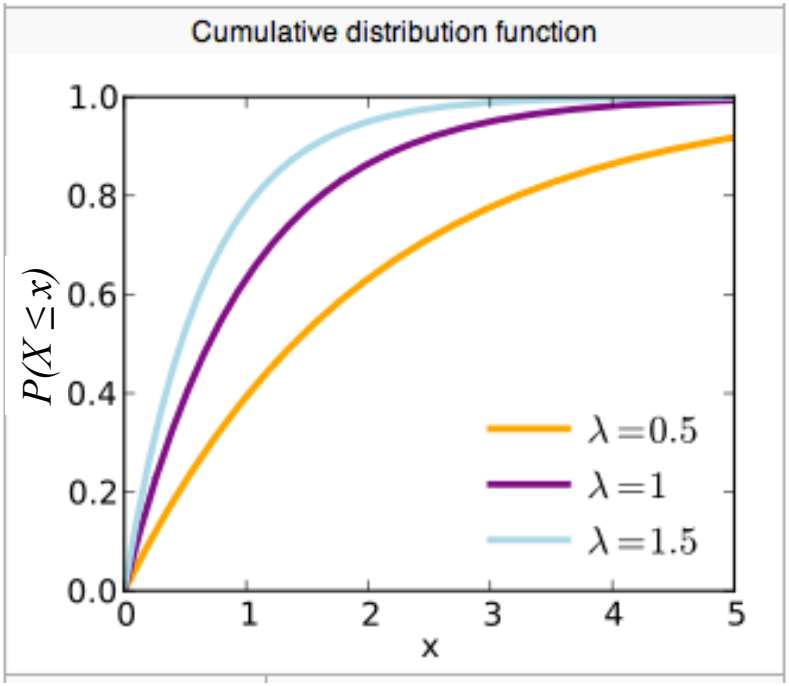
$$p(x) = \lambda \exp(-\lambda x)$$

$$\int_0^\infty p(x)\,dx = 1$$

$$\int_0^x p(x)\,dx = P(x)$$

exponential distribution is only positive values

definition of cumulative distribution function



Inverse Transform Sampling Method

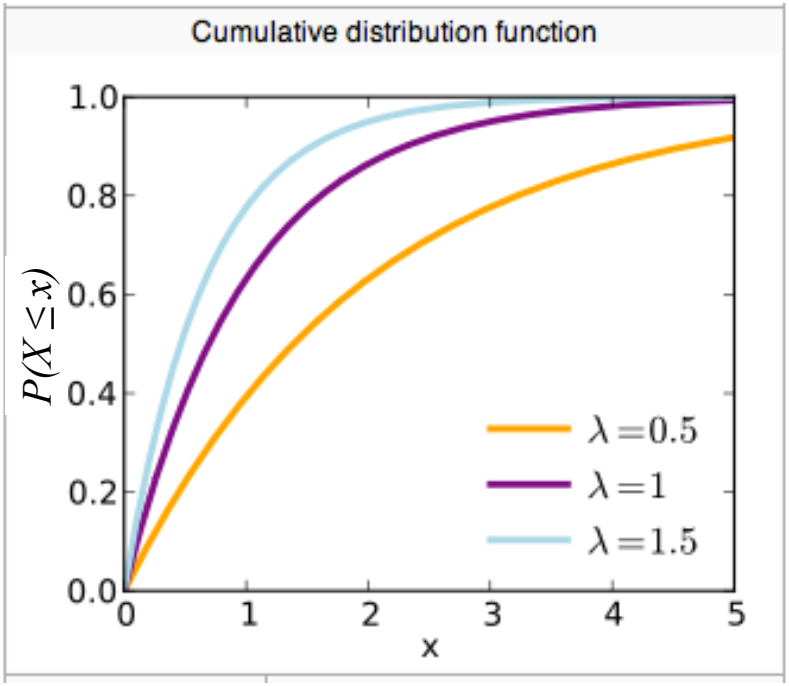
$$p(x) = \lambda \exp(-\lambda x)$$

$$\int_0^\infty p(x)\,dx = 1$$

$$\int_0^x p(x)\,dx = P(x)$$

exponential distribution is only positive values

therefore, we know that $P(x)$ has values [0,1]



Inverse Transform Sampling Method

$$p(x) = \lambda \exp(-\lambda x)$$

$$\int_0^{\infty} p(x)\,dx = 1$$

exponential distribution is only positive values

$$\int_0^{x} p(x)\,dx = P(x)$$

therefore, we know that $P(x)$ has values [0,1]

$$\int_0^{x} \lambda \exp(-\lambda x)\,dx = P(x)$$

Inverse Transform Sampling Method

$$p(x) = \lambda \exp(-\lambda x)$$

$$\int_0^\infty p(x)\,dx = 1$$

exponential distribution is only positive values

$$\int_0^x p(x)\,dx = P(x)$$

therefore, we know that $P(x)$ has values [0,1]

$$\int_0^x \lambda \exp(-\lambda x)\,dx = P(x)$$

plug in for $p(x)$

Inverse Transform Sampling Method

$$p(x) = \lambda \exp(-\lambda x)$$

$$\int_0^\infty p(x)\,dx = 1$$

exponential distribution is only positive values

$$\int_0^x p(x)\,dx = P(x)$$

therefore, we know that $P(x)$ has values [0,1]

$$\int_0^x \lambda \exp(-\lambda x)\,dx = P(x)$$

plug in for $p(x)$

$$1 - \lambda \exp(-\lambda x) = P(x)$$

solve integral

Inverse Transform Sampling Method

$$p(x) = \lambda \exp(-\lambda x)$$

$$\int_0^\infty p(x)\,dx = 1$$

exponential distribution is only positive values

$$\int_0^x p(x)\,dx = P(x)$$

therefore, we know that $P(x)$ has values [0,1]

$$\int_0^x \lambda \exp(-\lambda x)\,dx = P(x)$$

plug in for $p(x)$

$$1 - \lambda \exp(-\lambda x) = P(x)$$

solve integral

$$\exp(-\lambda x) = 1 - P(x)$$

rearrange terms

Inverse Transform Sampling Method

$$p(x) = \lambda \exp(-\lambda x)$$

$$\int_0^\infty p(x)\,dx = 1$$

exponential distribution is only positive values

$$\int_0^x p(x)\,dx = P(x)$$

therefore, we know that $P(x)$ has values [0,1]

$$\int_0^x \lambda \exp(-\lambda x)\,dx = P(x)$$

plug in for $p(x)$

$$1 - \lambda \exp(-\lambda x) = P(x)$$

solve integral

$$\exp(-\lambda x) = 1 - P(x)$$

rearrange terms

$$x = -\frac{1}{\lambda} \ln(1 - P(x))$$

solve for x

Inverse Transform Sampling Method

$$p(x) = \lambda \exp(-\lambda x)$$

$$\int_0^\infty p(x)\,dx = 1$$

exponential distribution is only positive values

$$\int_0^x p(x)\,dx = P(x)$$

therefore, we know that $P(x)$ has values [0,1]

$$\int_0^x \lambda \exp(-\lambda x)\,dx = P(x)$$

plug in for $p(x)$

$$1 - \lambda \exp(-\lambda x) = P(x)$$

solve integral

$$\exp(-\lambda x) = 1 - P(x)$$

rearrange terms

$$x = -\frac{1}{\lambda} \ln(1 - P(x))$$

remember, we know that $P(x)$ has values [0,1]

Inverse Transform Sampling Method

$$p(x) = \lambda \exp(-\lambda x)$$

$$\int_0^\infty p(x)\,dx = 1$$

$$\int_0^x p(x)\,dx = P(x)$$

$$\int_0^x \lambda \exp(-\lambda x)\,dx = P(x)$$

$$1 - \lambda \exp(-\lambda x) = P(x)$$

$$\exp(-\lambda x) = 1 - P(x)$$

$$x = -\frac{1}{\lambda} \ln(1 - U_{[0,1)})$$

$$x = -\frac{1}{\lambda} \ln U_{[0,1)}$$

Inverse Transform Method

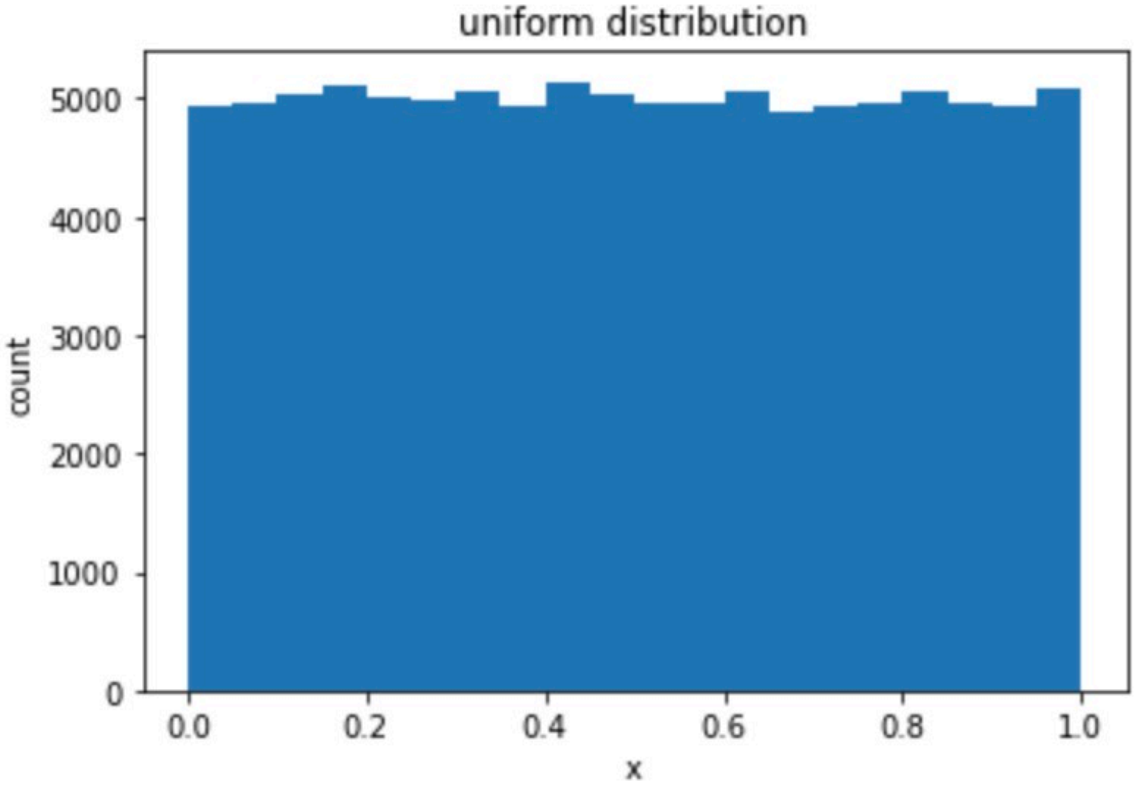
$U_{[0,1)}$ is between 0 and 1

imagine drawing a random number using `R.uniform()`

then x will be exponentially distributed with rate λ

continuous uniform distribution $U_{[0,1)}$

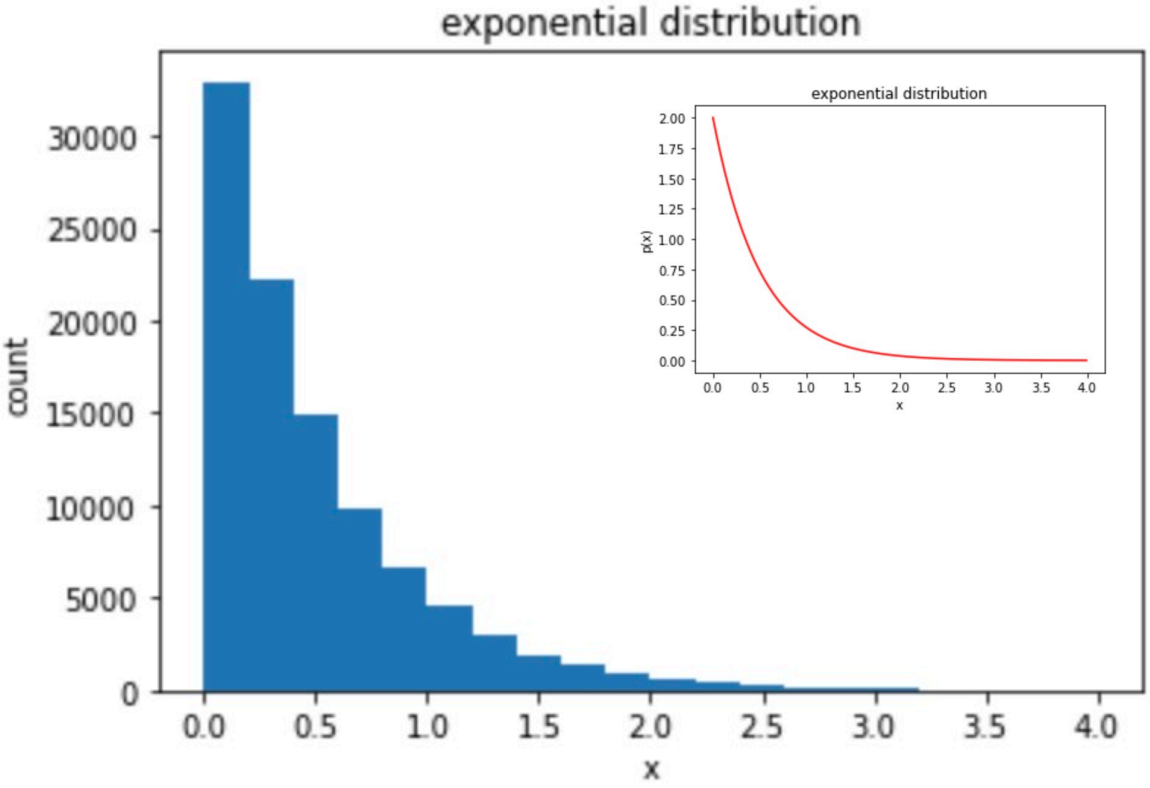
```
N = 100000
rnd = R.uniform(N, 1)
```



exponential via transformation method $x = -\frac{1}{\lambda} \ln U_{[0,1)}$

```
N = 100000
rnd = R.uniform(N, 1)

lam = 2
x = -(1/lam)*np.log(rnd)
```



Transformation Method

some built-in python random number generators use the transformation method, e.g., R.exponential(scale=2, size=(2,3))

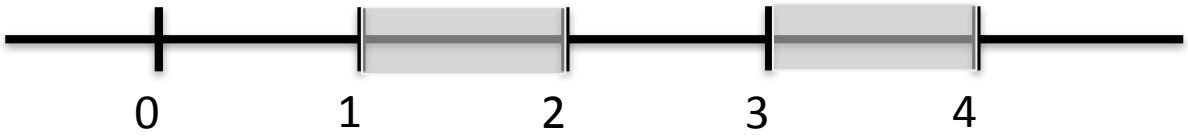
most distributions do not easily permit the transformation method

some use specialized algorithms (e.g., for the normal and related distributions), some use Monte Carlo methods (coming up next)

Rejection Sampling

Rejection Sampling Method

intuition …
imagine you had to create a random number generator that gave you a number uniformly distributed between 1 and 2 and uniformly distributed between 3 and 4, but does not exist anywhere else



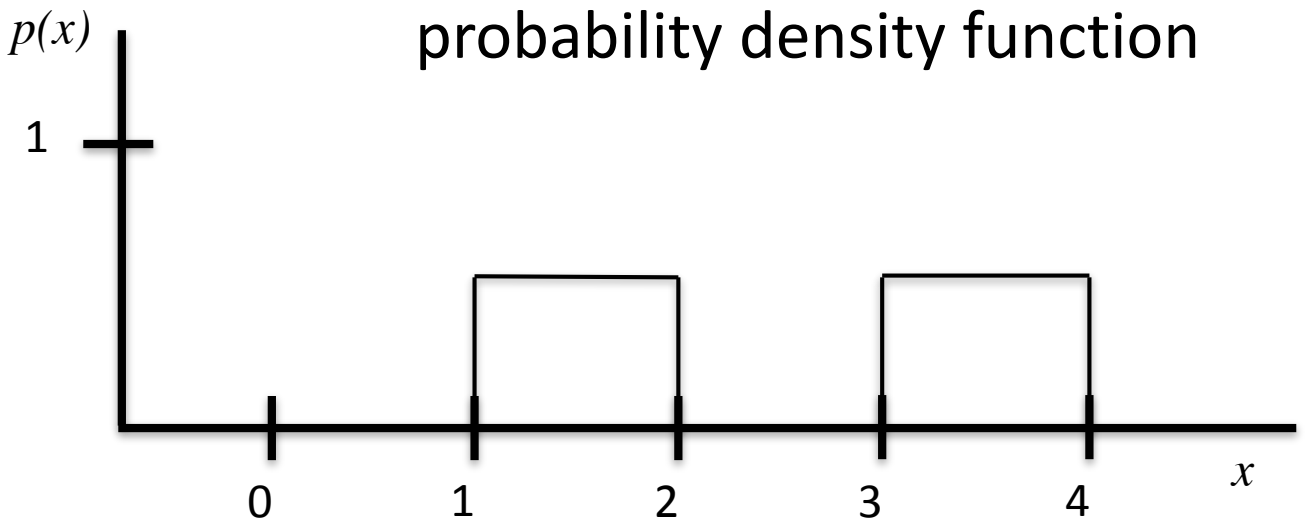
there is no built in Python program that does this …
nor for lots of other even more interesting distributions

Rejection Sampling

Rejection Sampling Method

intuition …

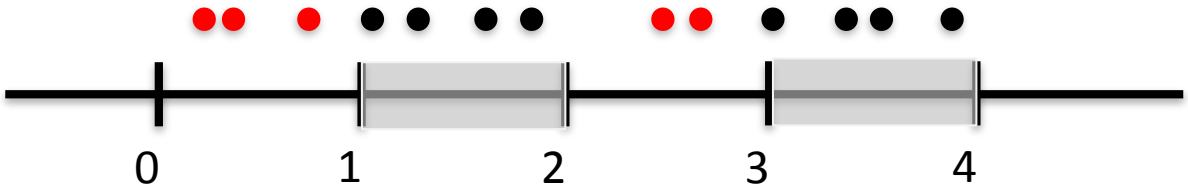
imagine you had to create a random number generator that gave you a number uniformly distributed between 1 and 2 and uniformly distributed between 3 and 4, but does not exist anywhere else



Rejection Sampling

Rejection Sampling Method

intuition …
imagine you had to create a random number generator that gave you a number uniformly distributed between 1 and 2 and uniformly distributed between 3 and 4, but does not exist anywhere else

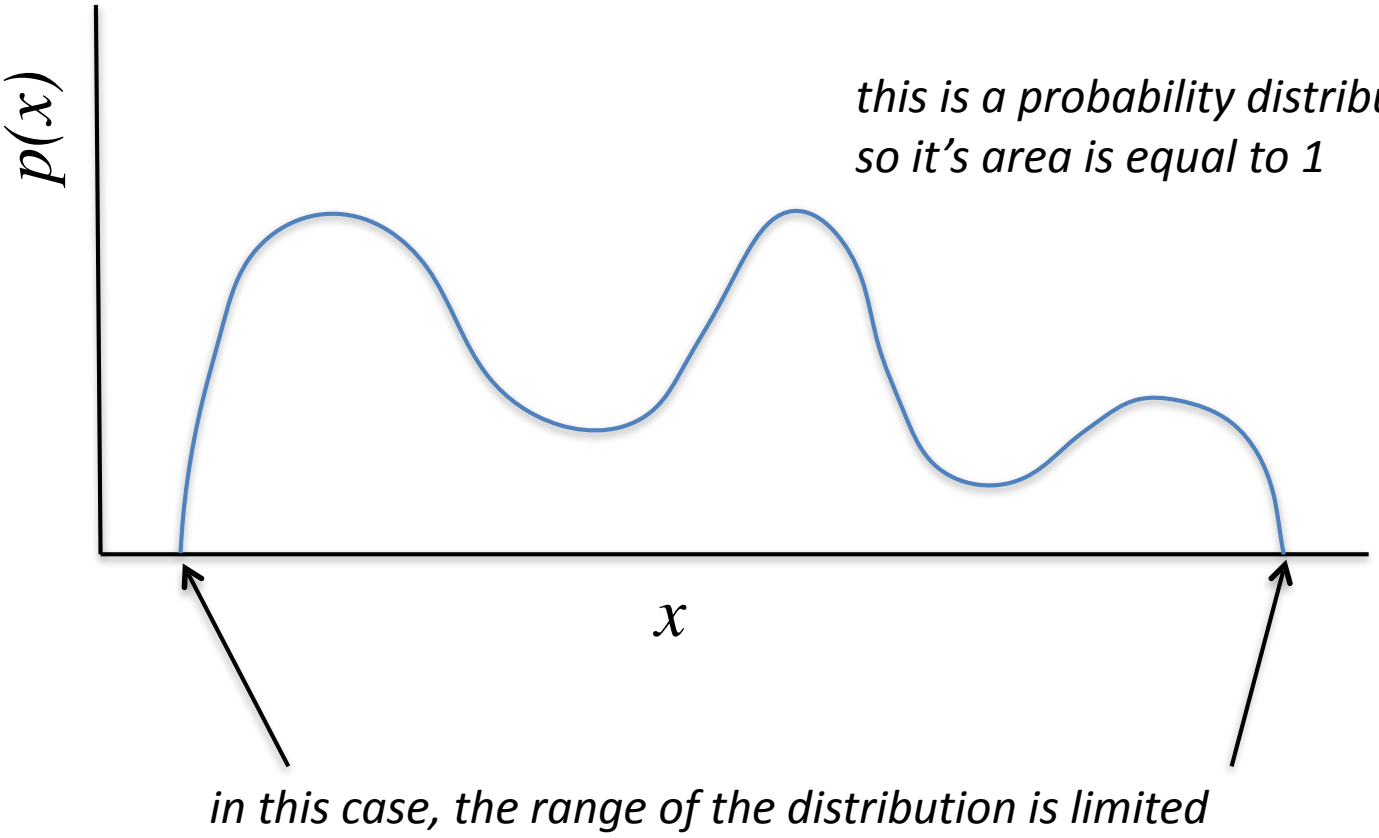


one possibility … generate uniform random numbers in [0,4) and reject any random numbers that don't fall in the required range … this is the beginning of rejection sampling

Rejection Sampling

Rejection Sampling Method

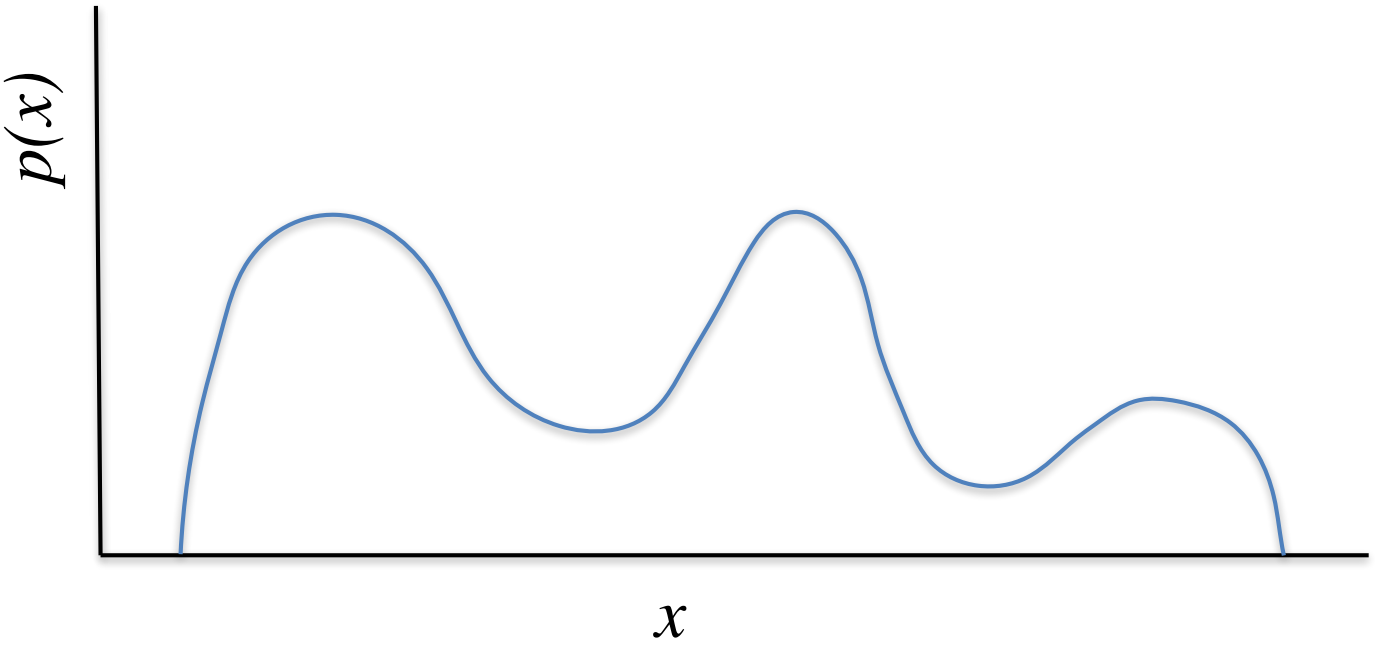
Now imagine you had some arbitrarily complex probability distribution that you wanted to generate random samples from …

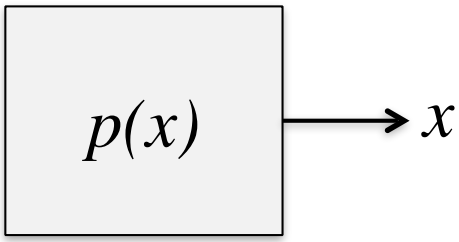


there are generalizations of rejection sampling that work with distributions going to negative and/or positive infinity

Rejection Sampling Method

Now imagine you had some arbitrarily complex probability distribution that you wanted to generate random samples from …

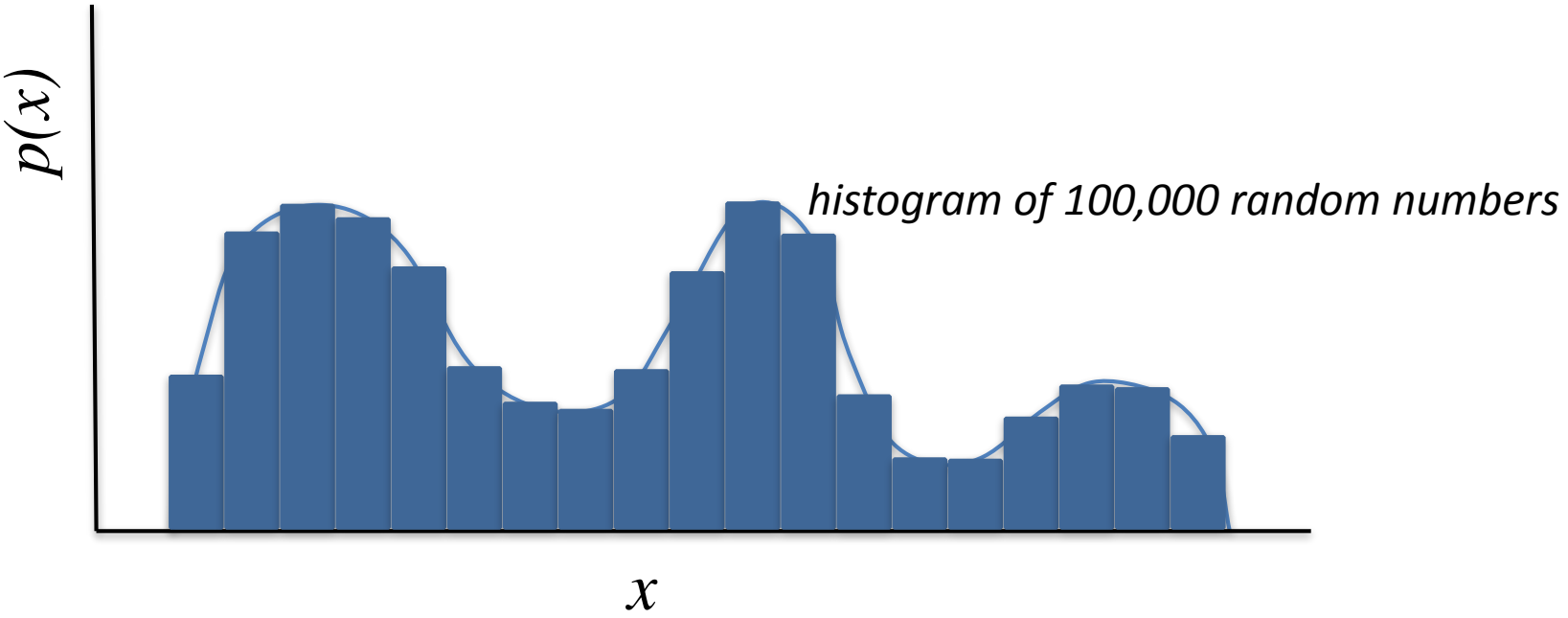




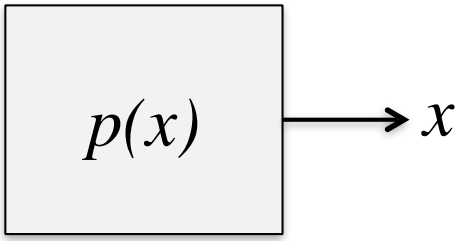
you want a machine that spits out random x values from a distribution $p(x)$

Rejection Sampling Method

Now imagine you had some arbitrarily complex probability distribution that you wanted to generate random samples from …



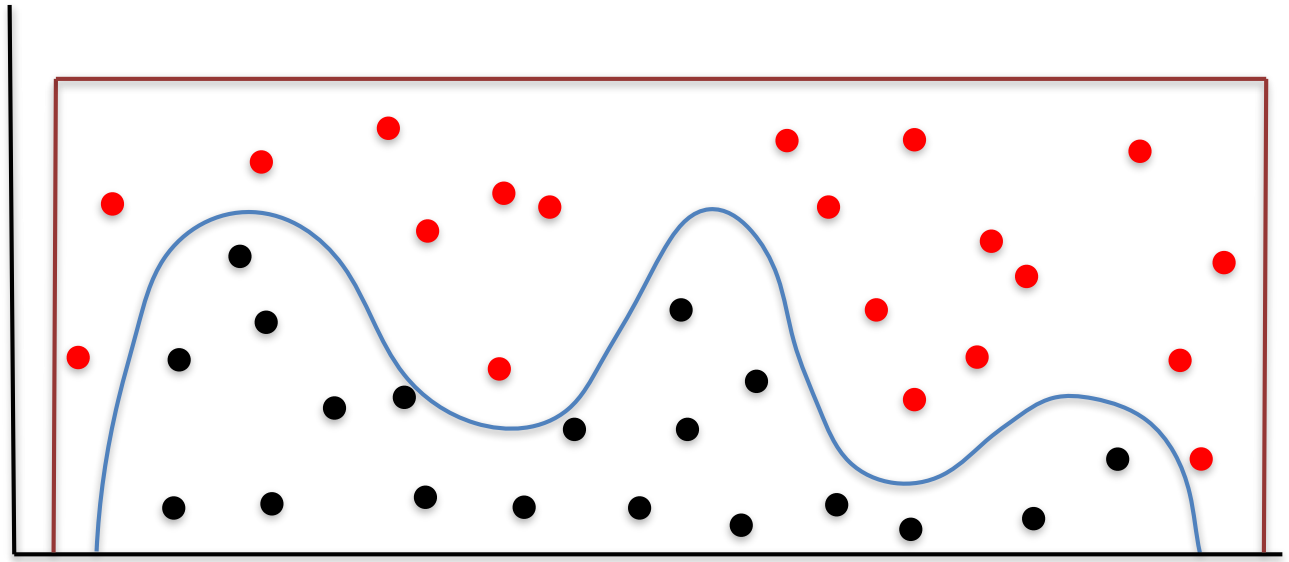
how do we create this?



you want a machine that spits out random x values from a distribution $p(x)$

Rejection Sampling

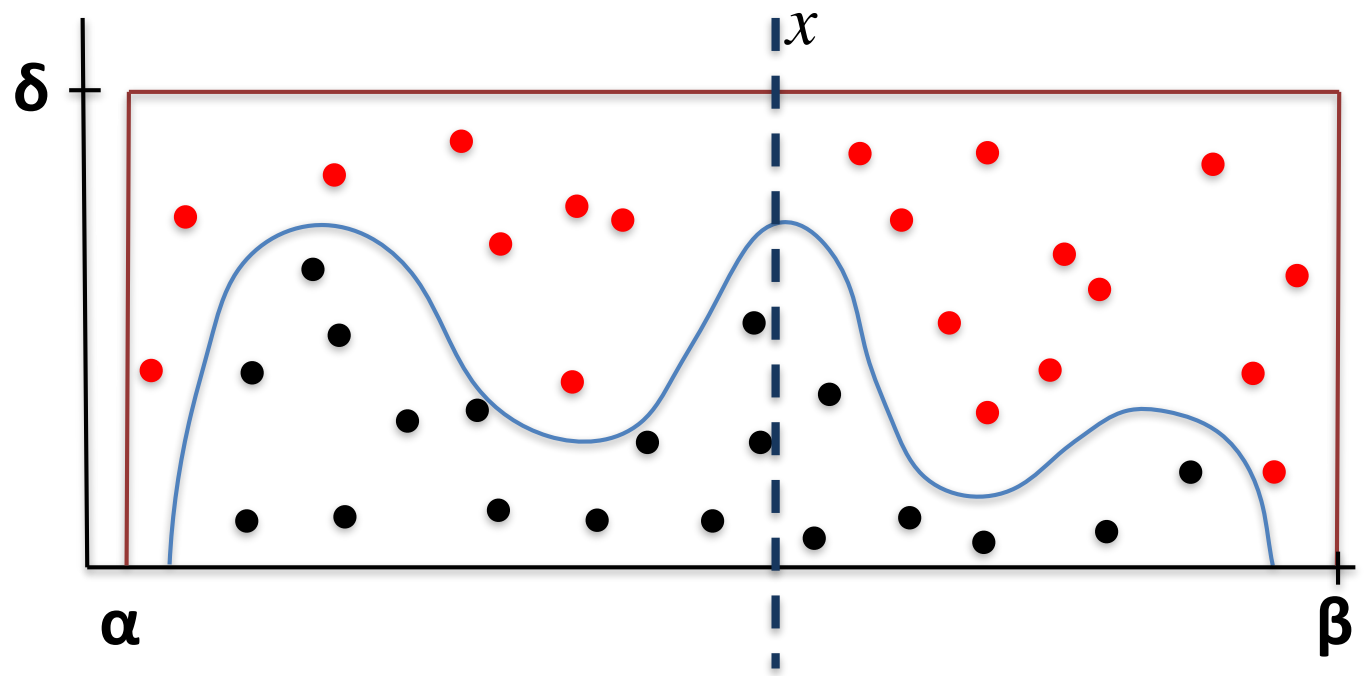
Rejection Sampling Method (intuition)



if we could come up with a distribution that entirely encompassed the target distribution and that allowed us to easily generate uniformly distributed deviates, then we could accept points that fell within the distribution and reject points that fell outside

Rejection Sampling

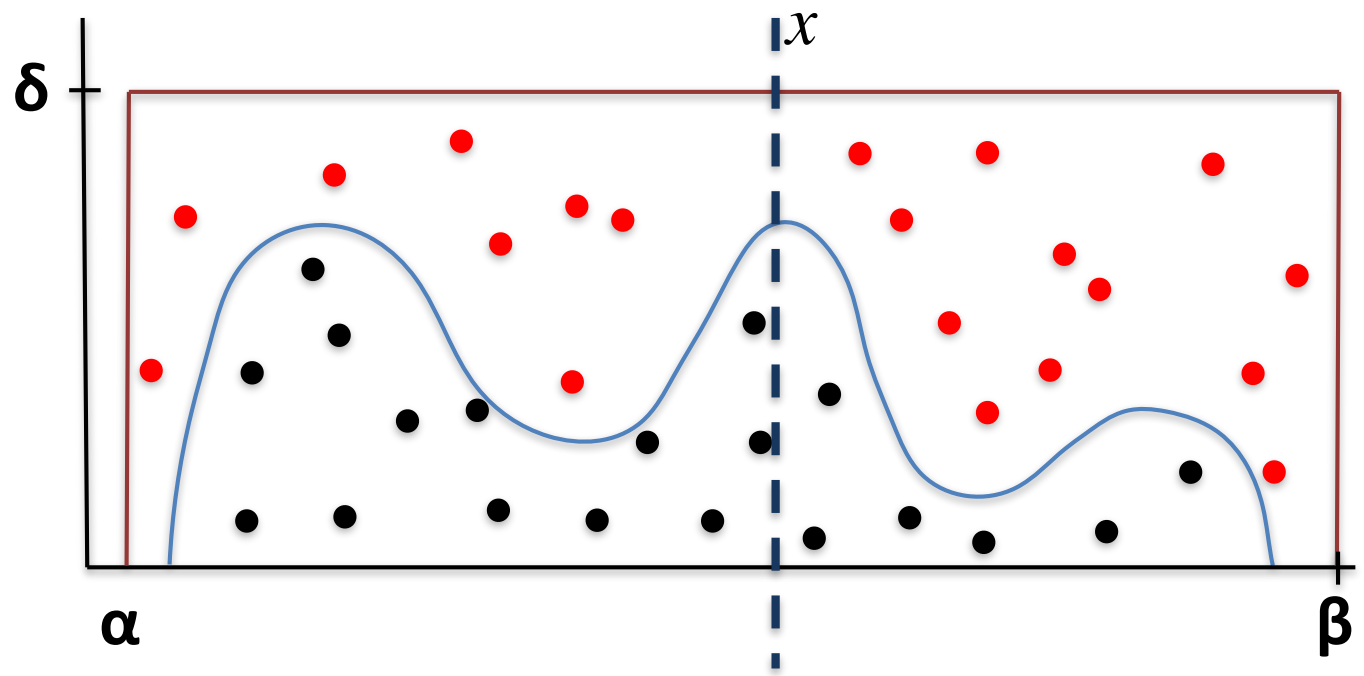
Rejection Sampling Method (intuition)



- first generate a uniformly distributed random number in the interval [**α**, **β**), and call it x

Rejection Sampling

Rejection Sampling Method (intuition)

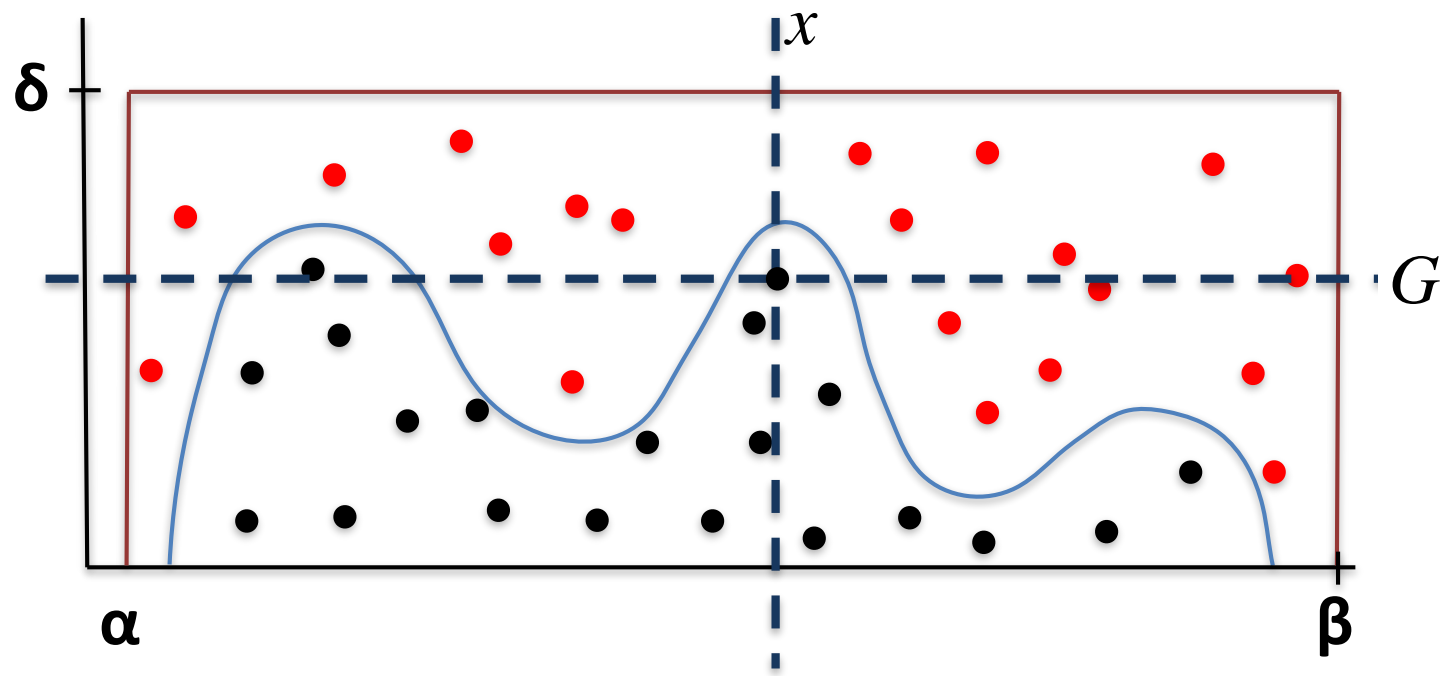


- first generate a uniformly distributed random number in the interval [**α**, **β**), and call it x

  Important: this assumes that the probability distribution has finite bounds and fits within [**α**, **β**)

Rejection Sampling

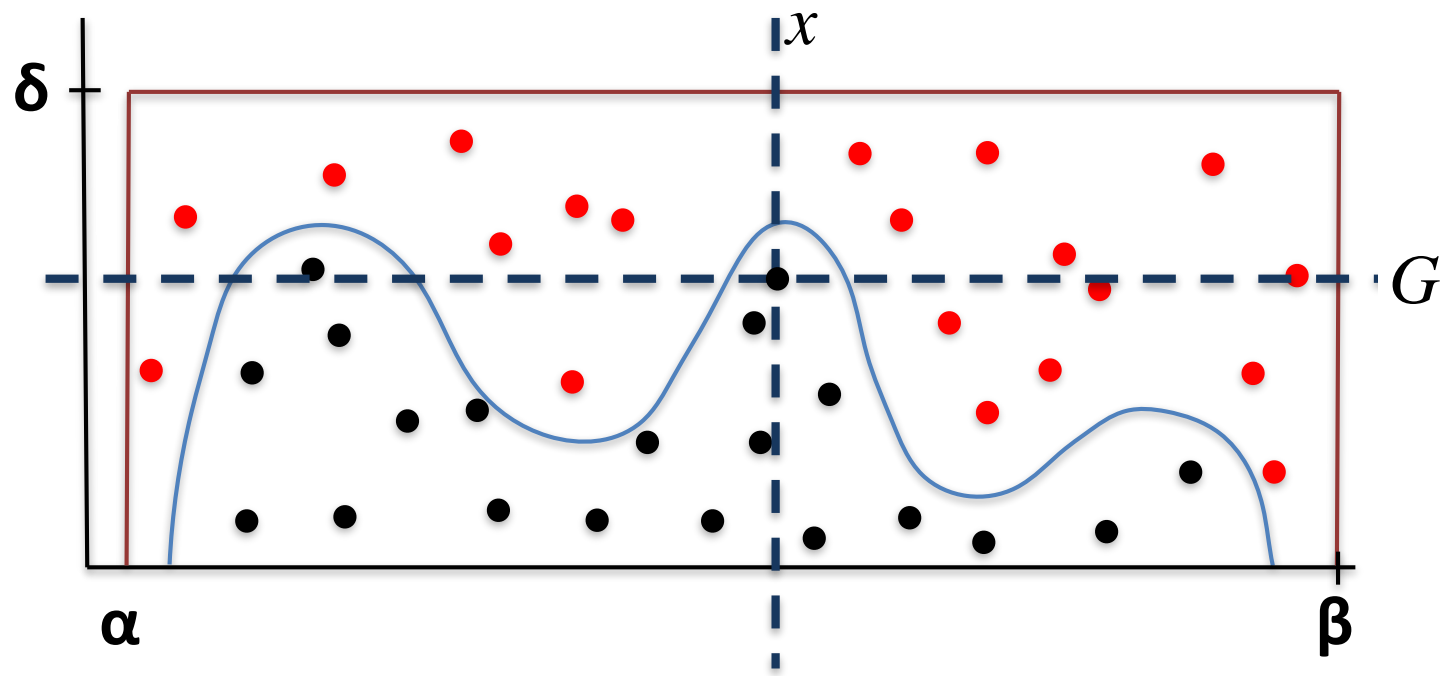
Rejection Sampling Method (intuition)



- first generate a uniformly distributed random number in the interval [**α**, **β**), and call it x
- then generate a uniformly distributed number in the interval [0, **δ**) and call it G

Rejection Sampling

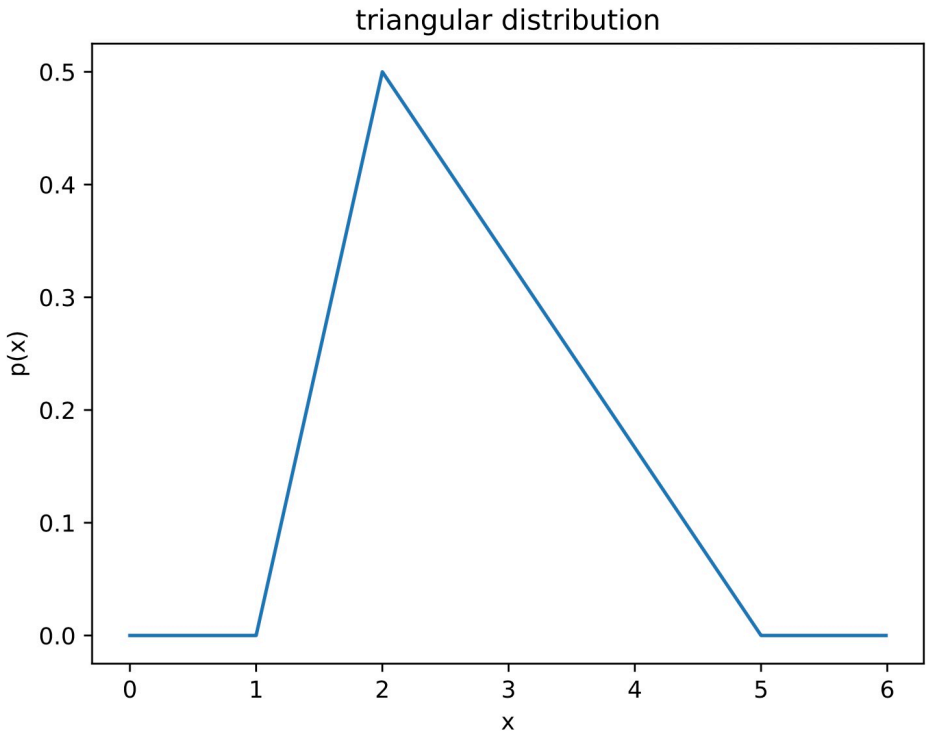
Rejection Sampling Method (intuition)



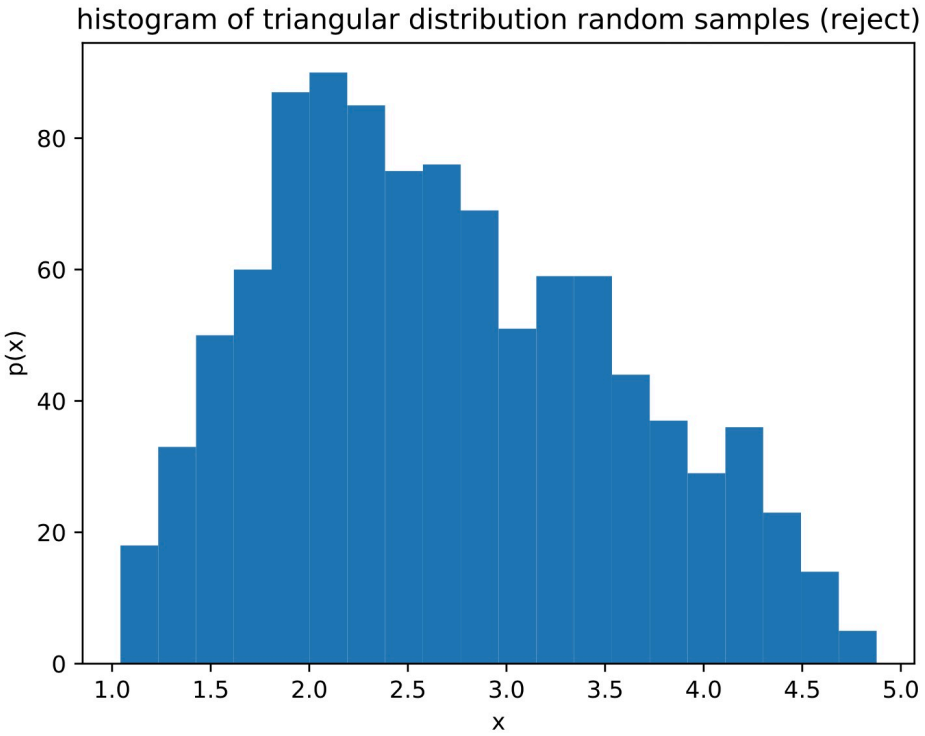
- first generate a uniformly distributed random number in the interval [**α**, **β**), and call it x
- then generate a uniformly distributed number in the interval [0, **δ**) and call it G
- if $G < p(x)$, then return x as the random sample from $p(x)$
- otherwise, reject and resample

Triangular Distribution

plot probability distribution



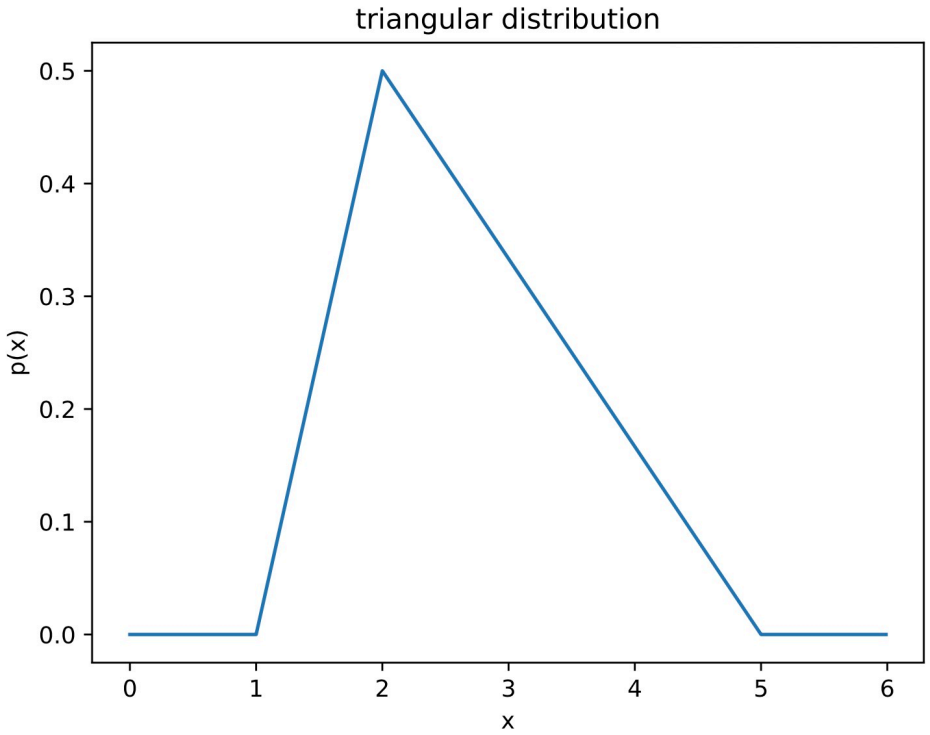
generate random numbers



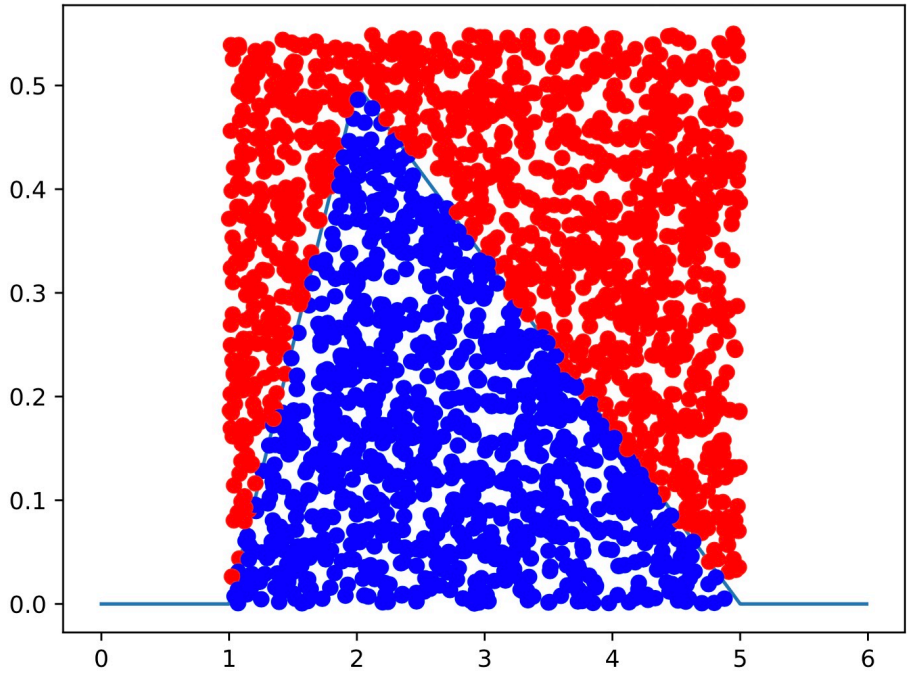
using rejection sampling

Triangular Distribution

plot probability distribution



generate random numbers



using rejection sampling

generate random sample from distributions other than the uniform

- **inverse transform sampling** - mathematically transform random numbers from a uniform to random numbers from another distribution (appropriate for mathematically simple probability distributions, like the exponential distribution)

- **other transformation methods** - specialized transform algorithm (e.g., Box-Muller transform for using samples from a uniform distribution to generate samples from a normal distribution)

- **rejection sampling algorithms** - Monte Carlo methods that can (in principle) be applied to any arbitrarily complex probability distribution (using random numbers to generate random numbers)

- **Markov Chain Monte Carlo methods** - simplest example is the Metropolis-Hastings algorithm (such methods are often associated with Bayesian statistics, but they are more general random number generation techniques beyond applications to Bayes)

Bayes' Rule

likelihood prior

posterior

$$p(\theta \mid D) = \frac{p(D \mid \theta)\, p(\theta)}{\int p(D \mid \theta')\, d\theta'}$$

marginal

Bayes' Rule

$$\underbrace{p(\theta \mid D)}_{\text{posterior}} = \frac{\overbrace{p(D \mid \theta)}^{\text{likelihood}}\ \overbrace{p(\theta)}^{\text{prior}}}{\underbrace{p(D)}_{\text{marginal}}}$$

Bayes' Rule

$$p(\theta_1, \theta_2, \theta_3, \ldots \mid D) = \frac{p(D \mid \theta_1, \theta_2, \theta_3, \ldots)\, p(\theta_1, \theta_2, \theta_3, \ldots)}{\int\int\int \ldots p(D \mid \theta_1', \theta_2', \theta_3', \ldots)\, d\theta_1'\, d\theta_2'\, d\theta_3' \ldots}$$

Bayes' Rule

$$p(\theta_1, \theta_2, \theta_3, \ldots \mid D) = \frac{p(D \mid \theta_1, \theta_2, \theta_3, \ldots)\, p(\theta_1, \theta_2, \theta_3, \ldots)}{\int\int\int \ldots p(D \mid \theta_1', \theta_2', \theta_3', \ldots)\, d\theta_1'\, d\theta_2'\, d\theta_3' \ldots}$$

this makes Bayesian statistics hard
(these integrals cannot be solved analytically or numerically)

Bayes' Rule

create a random number generator
that samples from this distribution

$$p(\theta_1, \theta_2, \theta_3, \ldots \mid D) = \frac{p(D \mid \theta_1, \theta_2, \theta_3, \ldots)\, p(\theta_1, \theta_2, \theta_3, \ldots)}{\int\int\int \ldots p(D \mid \theta_1', \theta_2', \theta_3', \ldots)\, d\theta_1'\, d\theta_2'\, d\theta_3' \ldots}$$

- **Markov Chain Monte Carlo methods** - simplest example is the Metropolis-Hastings algorithm (such methods are often associated with Bayesian statistics, but they are more general random number generation techniques beyond applications to Bayes)

Markov Chain Monte Carlo (MCMC)

Markov Chain **Monte Carlo** (MCMC)

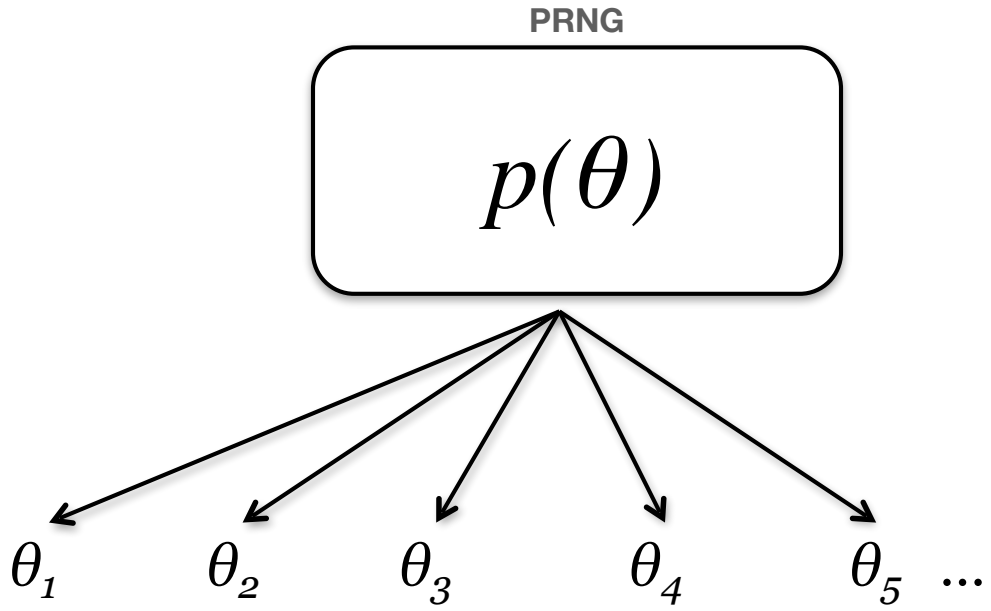
random numbers

Markov Chain Monte Carlo (MCMC)

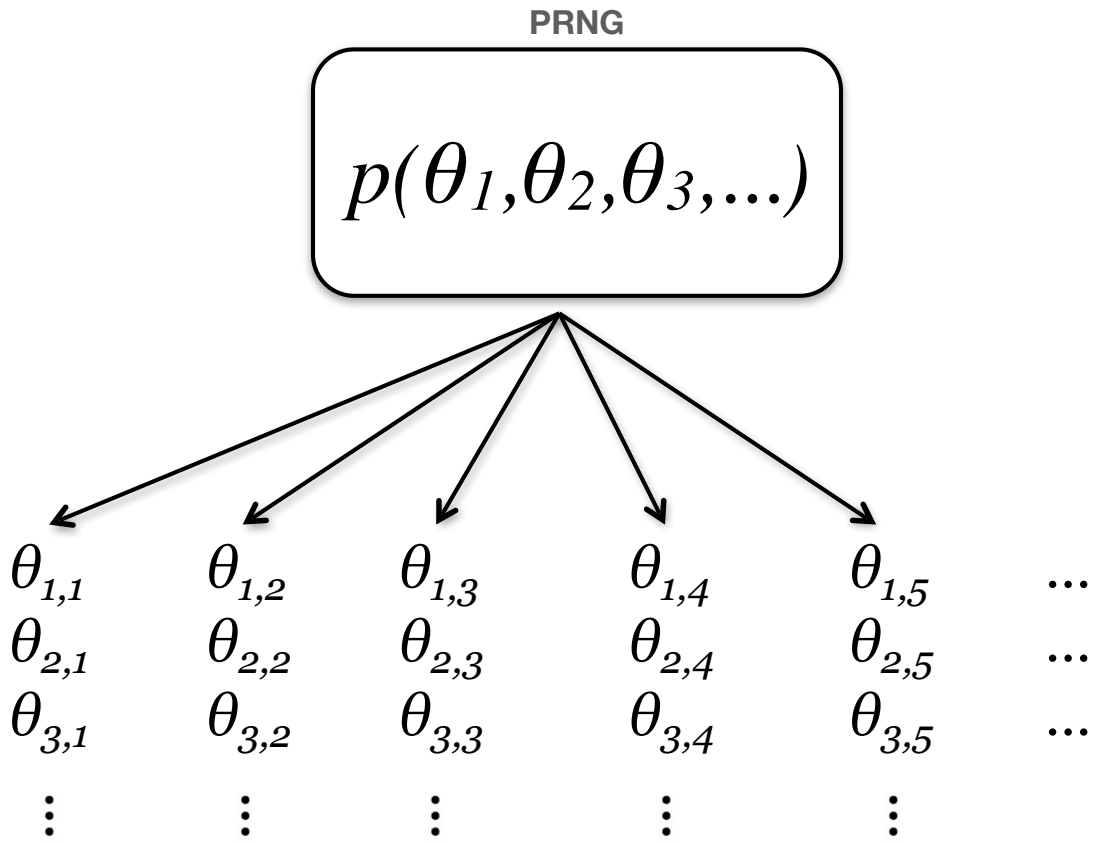
random sequence
obeying certain
rules

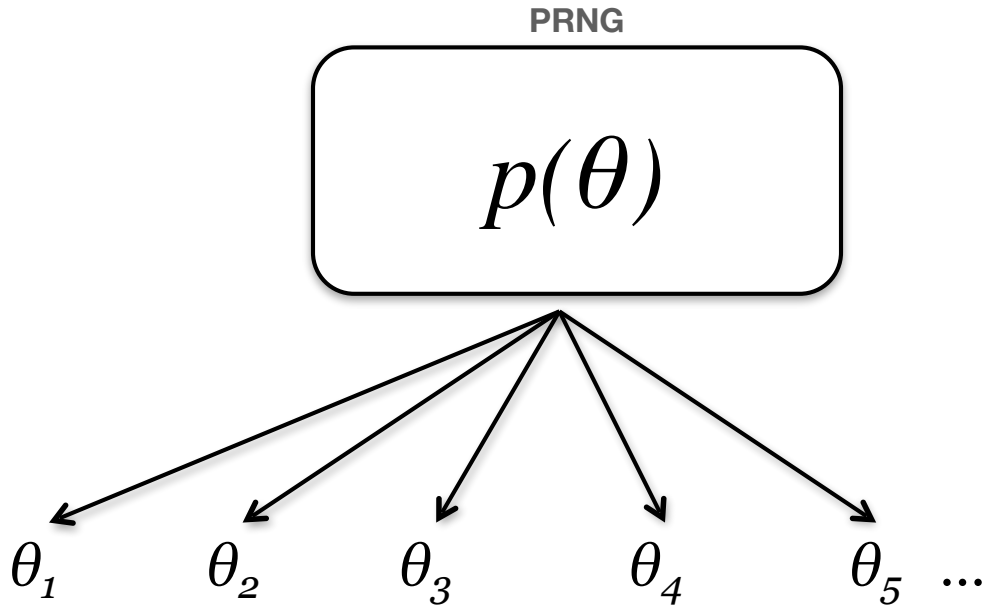
Markov Chain Monte Carlo (MCMC)

another method for generating random numbers sampled from an arbitrary probability distribution $p(\theta)$



notational comment: these θs are samples
from a random number generator
(the previous θs were different parameters)

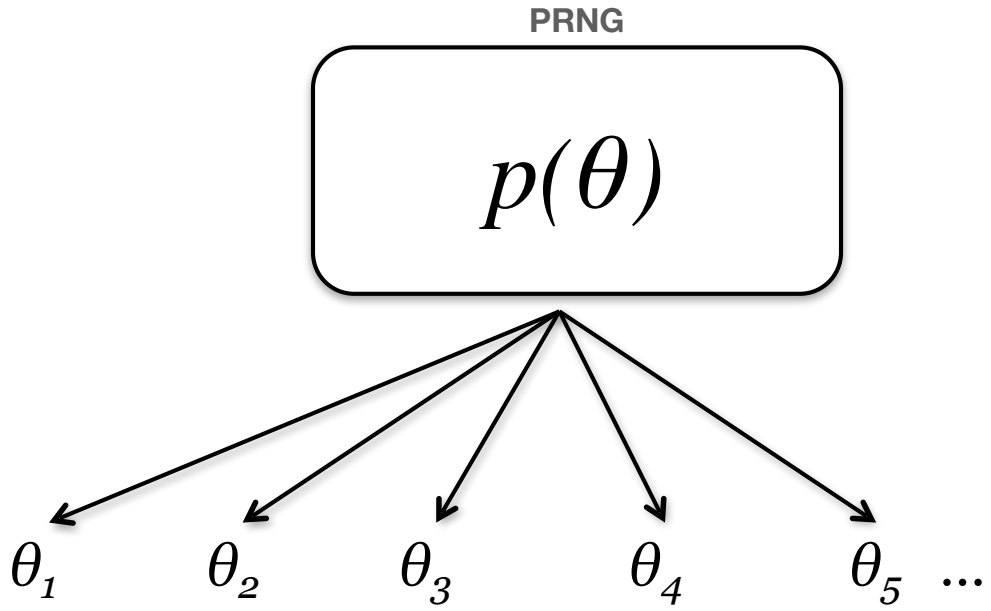

PRNG
$p(\theta_1,\theta_2,\theta_3,\ldots)$
$\theta_{1,1}$ $\theta_{1,2}$ $\theta_{1,3}$ $\theta_{1,4}$ $\theta_{1,5}$ …
$\theta_{2,1}$ $\theta_{2,2}$ $\theta_{2,3}$ $\theta_{2,4}$ $\theta_{2,5}$ …
$\theta_{3,1}$ $\theta_{3,2}$ $\theta_{3,3}$ $\theta_{3,4}$ $\theta_{3,5}$ …
⋮ ⋮ ⋮ ⋮ ⋮



Independent Sampling from a PRNG

$$p(\theta_t / \theta_1 \ldots \theta_{t-1}) = p(\theta_t)$$

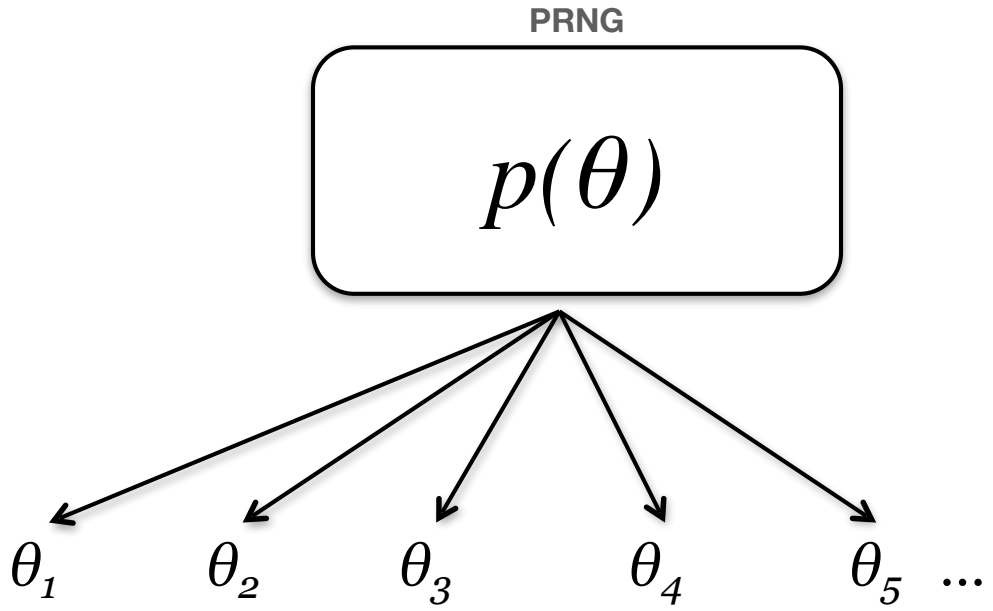
because the samples are independent, smaller sample sizes are needed to approximate distributions



Independent Sampling from a PRNG

$$p(\theta_t / \theta_1 \ldots \theta_{t-1}) = p(\theta_t)$$

because the samples are independent, smaller sample sizes are needed to approximate distributions

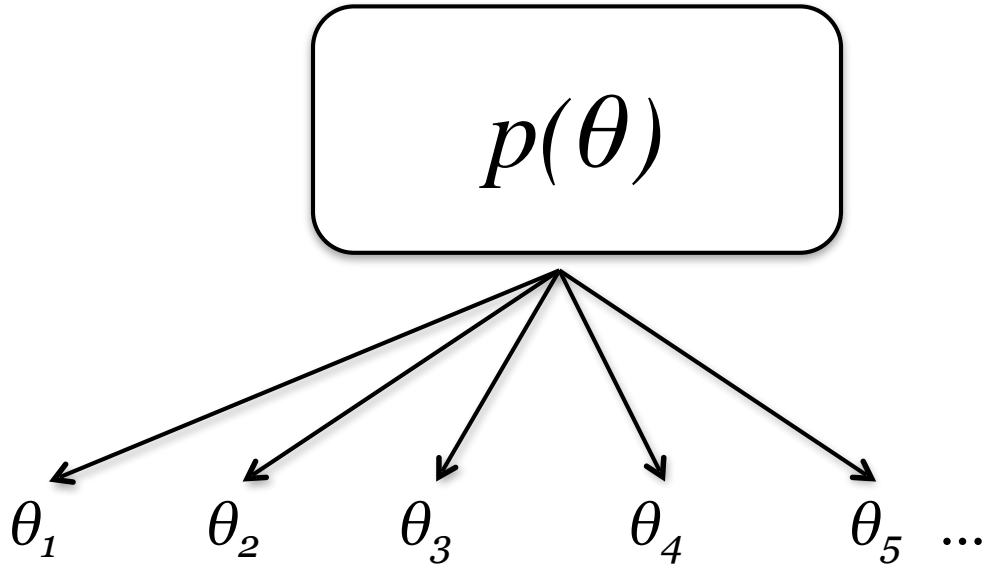


statistically independent while algorithmically fully dependent

Independent Sampling from a PRNG

$$p(\theta_t / \theta_1 \ldots \theta_{t-1}) = p(\theta_t)$$

because the samples are independent, smaller sample sizes are needed to approximate distributions



Independent Sampling from a PRNG

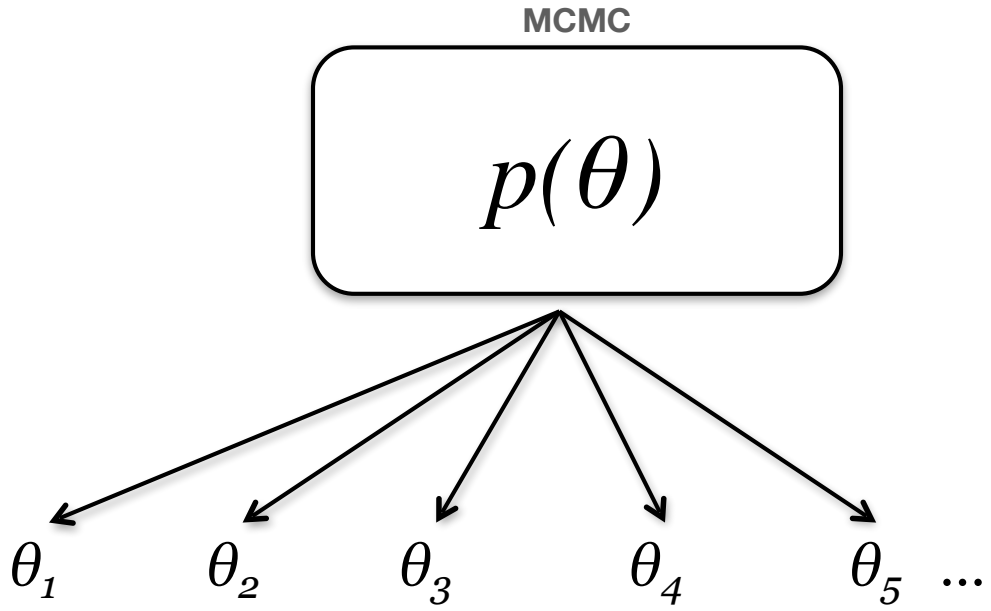
$p(\theta_t / \theta_1 \ldots \theta_{t-1}) = p(\theta_t)$

because the samples are independent, smaller sample sizes are needed to approximate distributions

Sampling from a Markov Chain Process

$p(\theta_t \mid \theta_1 \ldots \theta_{t-1}) = p(\theta_t \mid \theta_{t-1})$

because the samples are dependent, far larger sample sizes are needed to approximate distributions



Independent Sampling from a PRNG

$p(\theta_t \mid \theta_1 \ldots \theta_{t-1}) = p(\theta_t)$

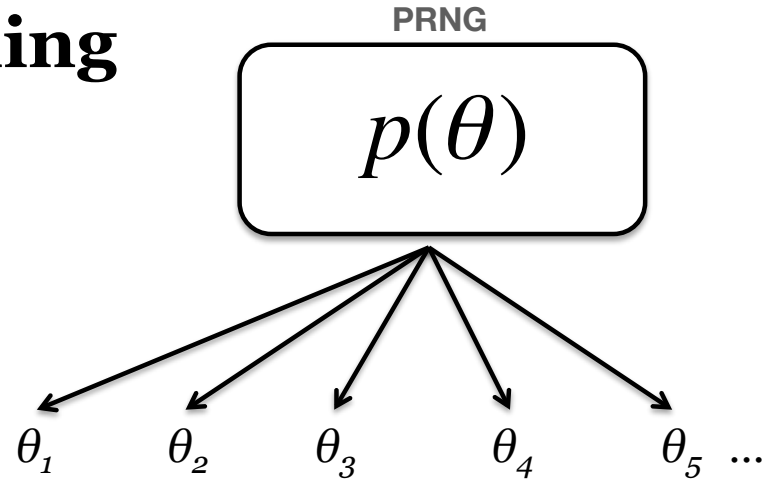
because the samples are independent, smaller sample sizes are needed to approximate distributions

Sampling from a **Markov Chain** Process

$\boldsymbol{p(\theta_t \mid \theta_1 \ldots \theta_{t-1}) = p(\theta_t \mid \theta_{t-1})}$

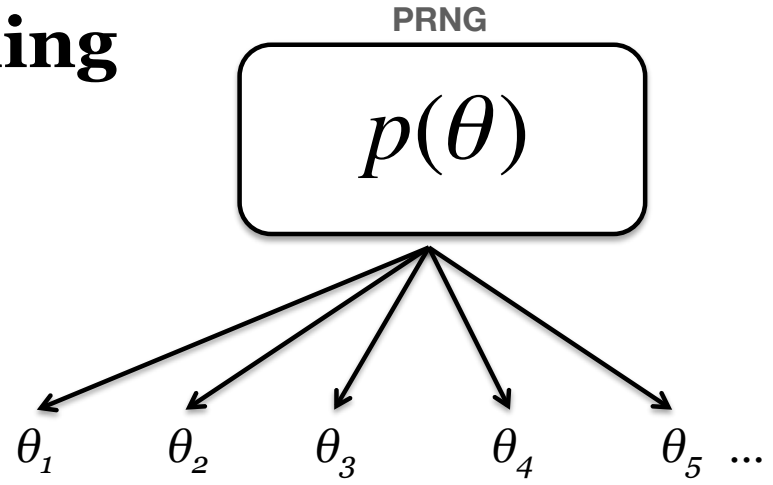
because the samples are dependent, far larger sample sizes are needed to approximate distributions

Independent Sampling

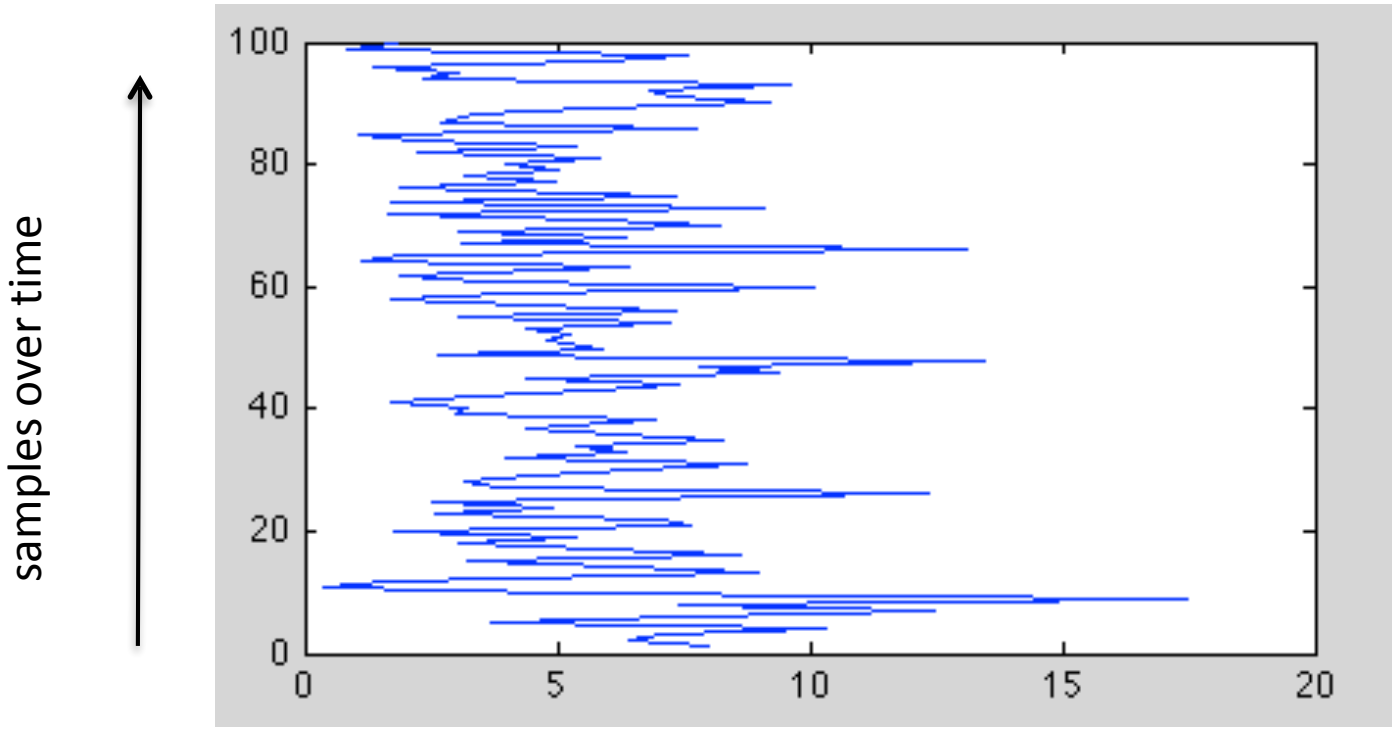


let's imagine drawing 100 random sample generated by a PRNG … we're just plotting these random samples along the x axis, with the count (or time) along the y axis …

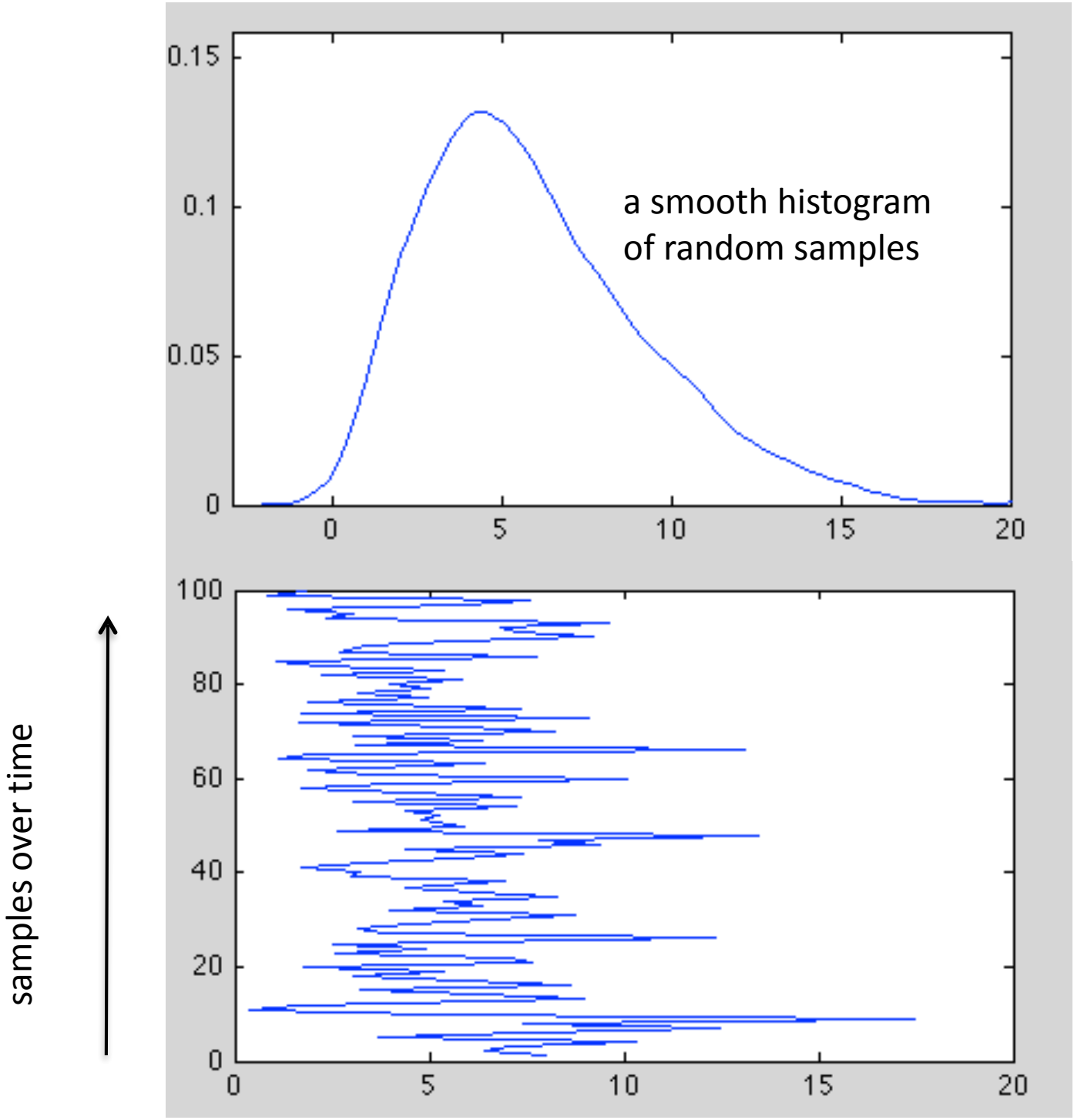
Independent Sampling



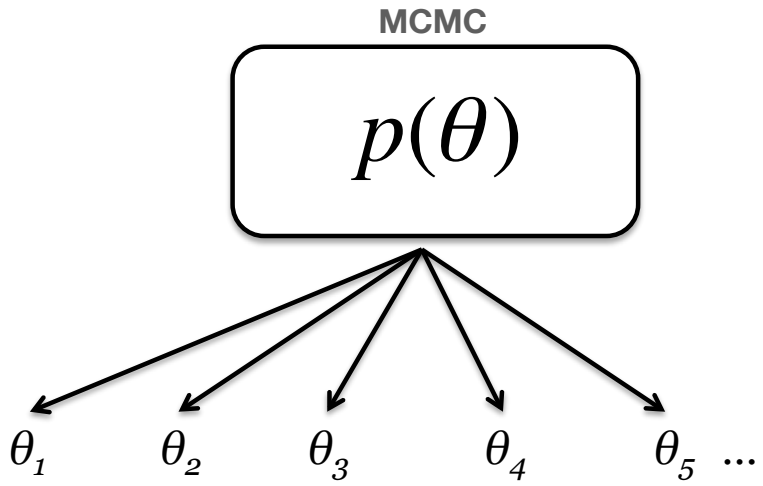
let's imagine drawing 100 random sample generated by a PRNG … we're just plotting these random samples along the x axis, with the count (or time) along the y axis …



Independent Sampling

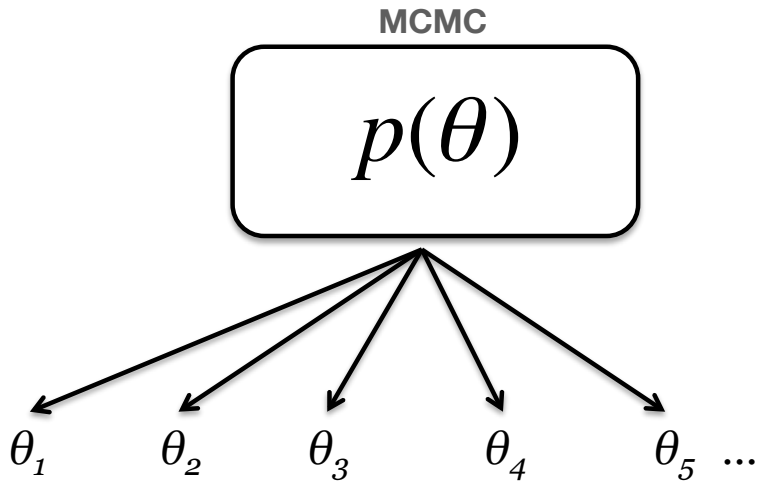


MCMC Sampling

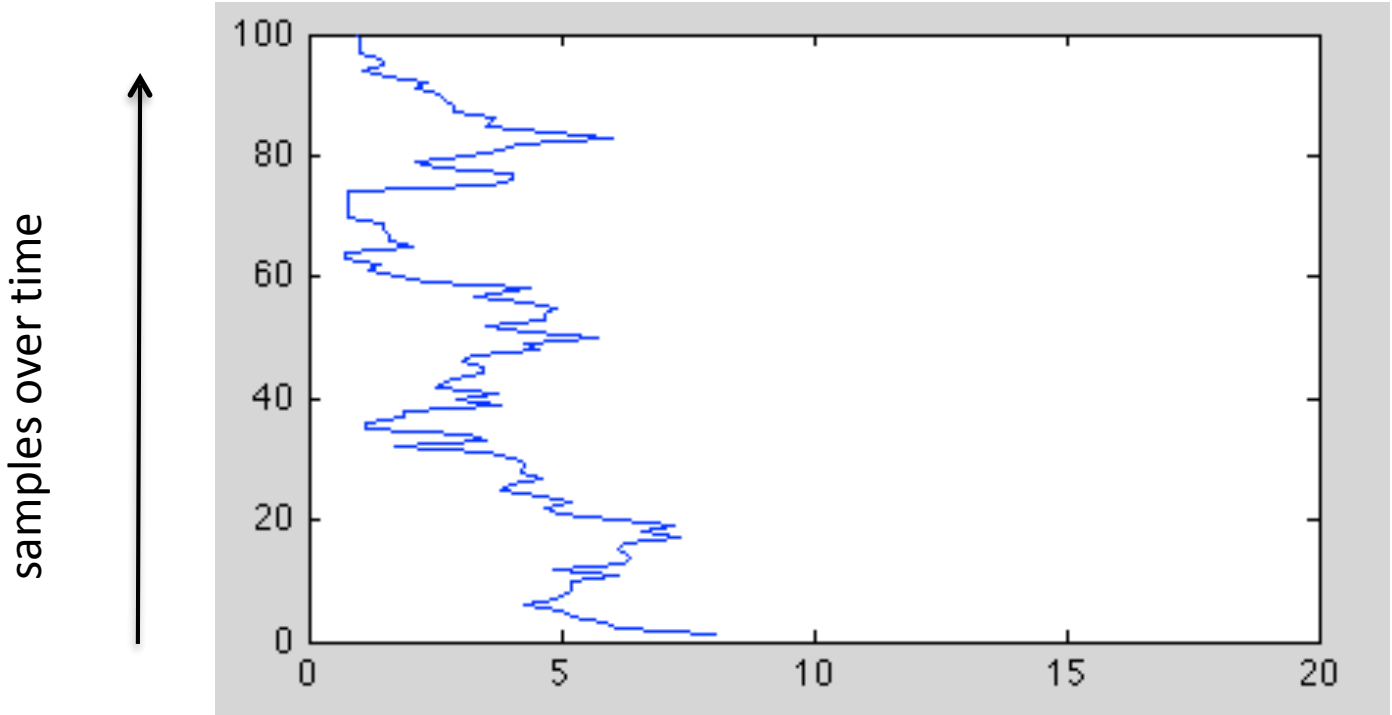


in an MCMC random sampling scheme

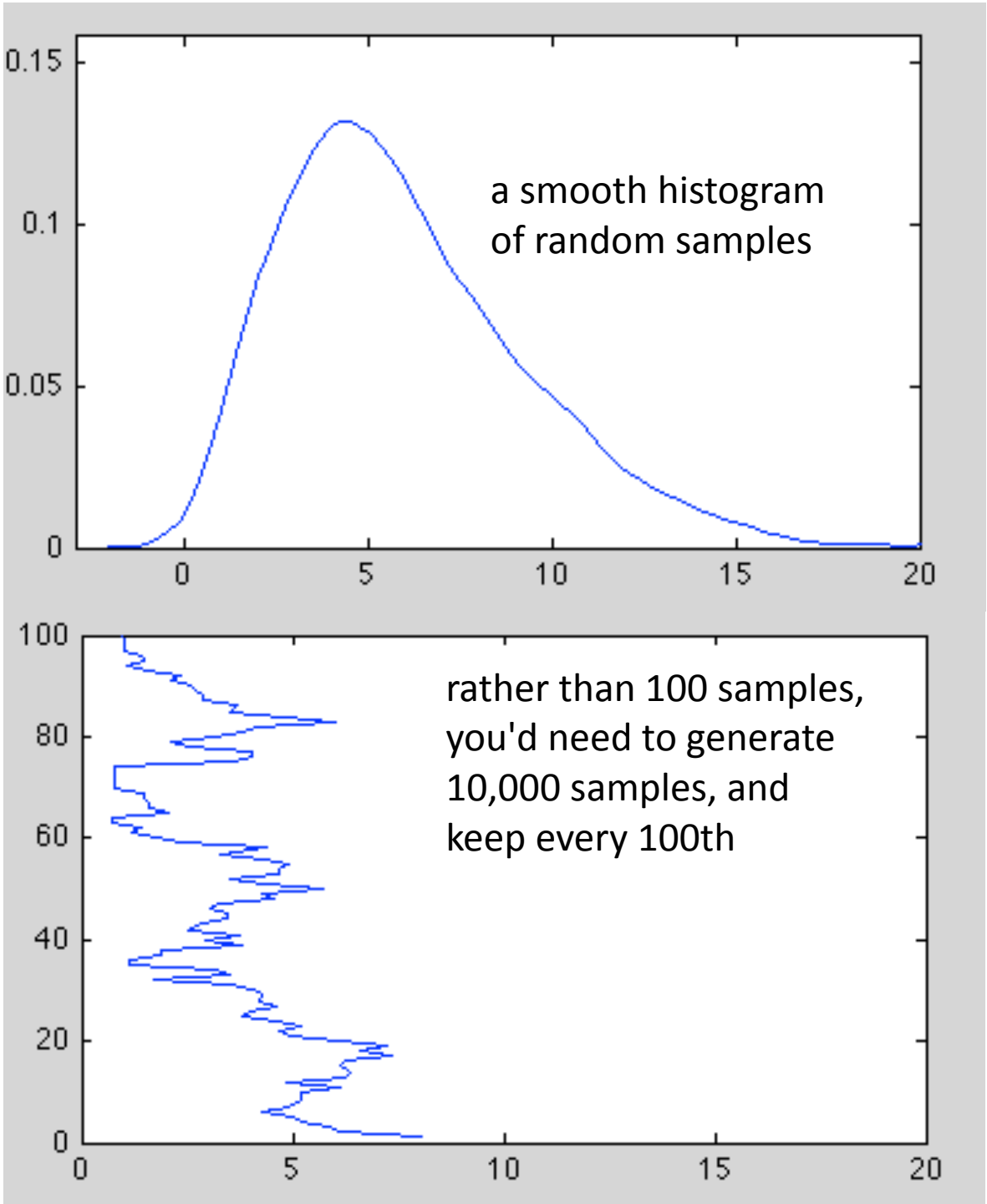
MCMC Sampling



in an MCMC random sampling scheme, you can see that there is some relationship between successive samples … in fact, this is a random walk (discrete version of a diffusion)

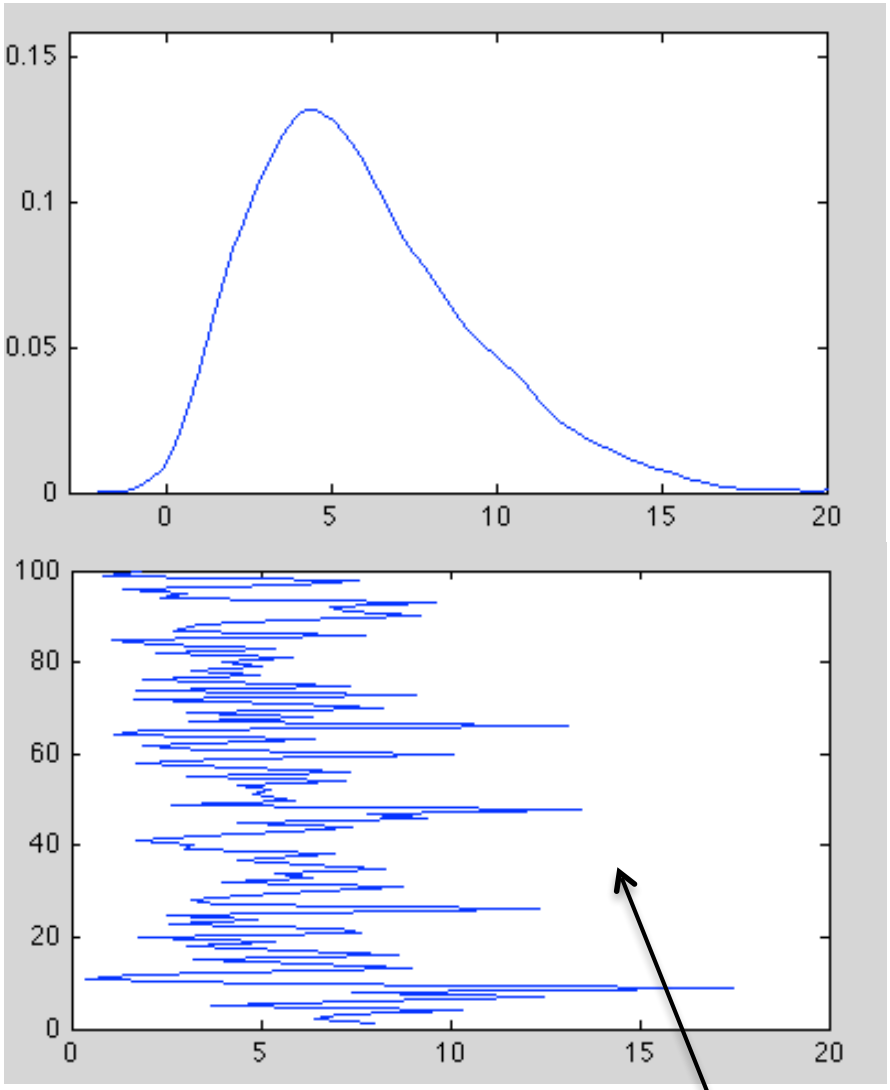


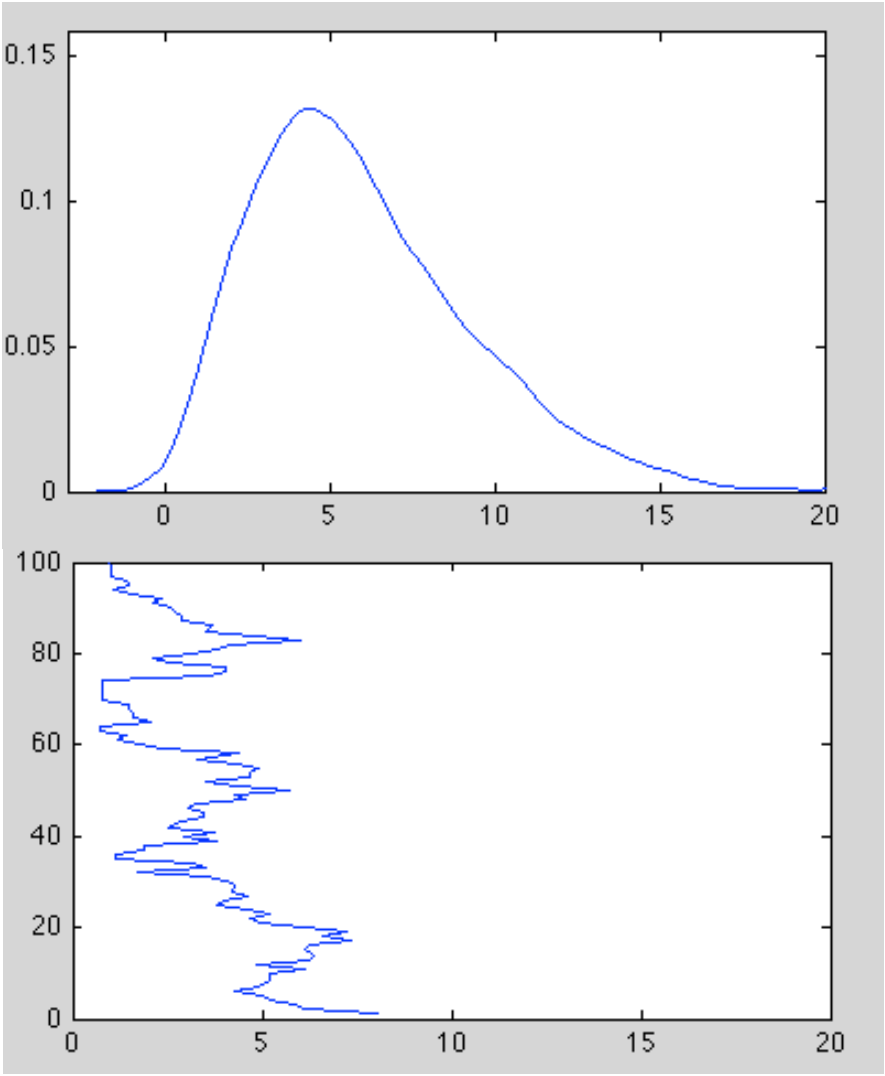
MCMC Sampling



Independent

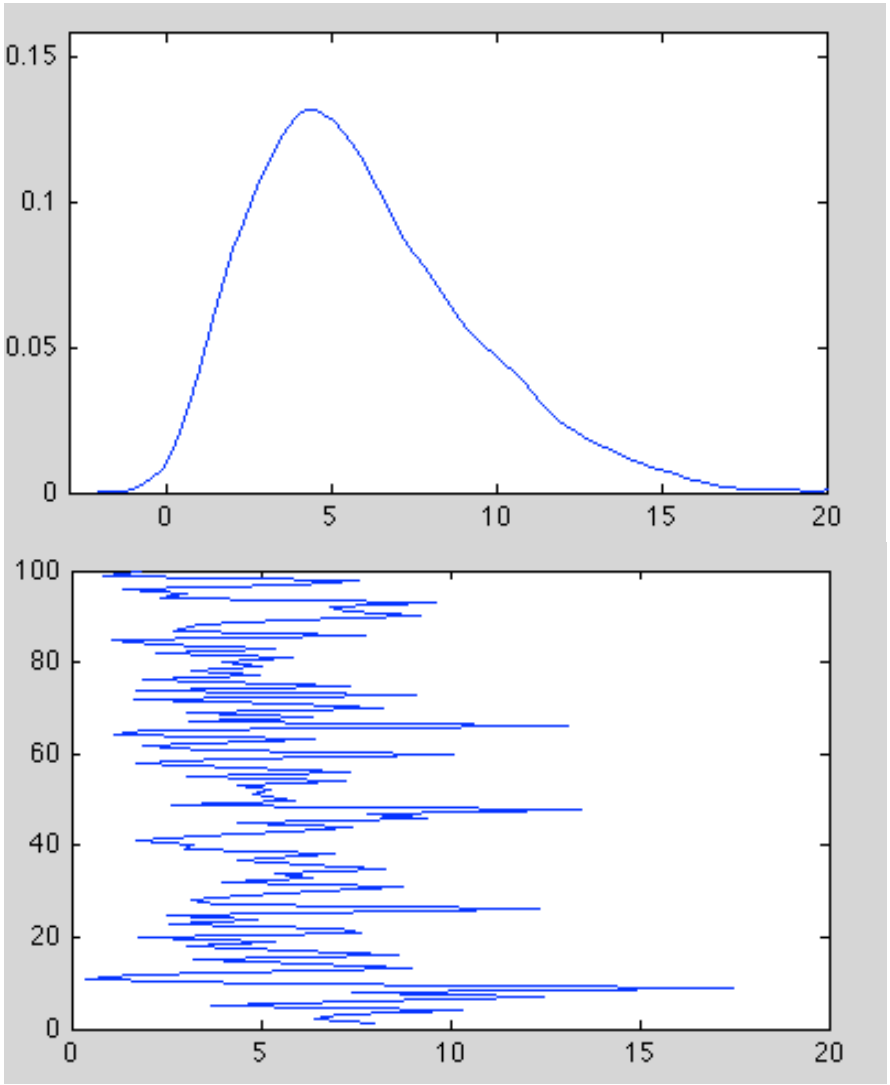
MCMC



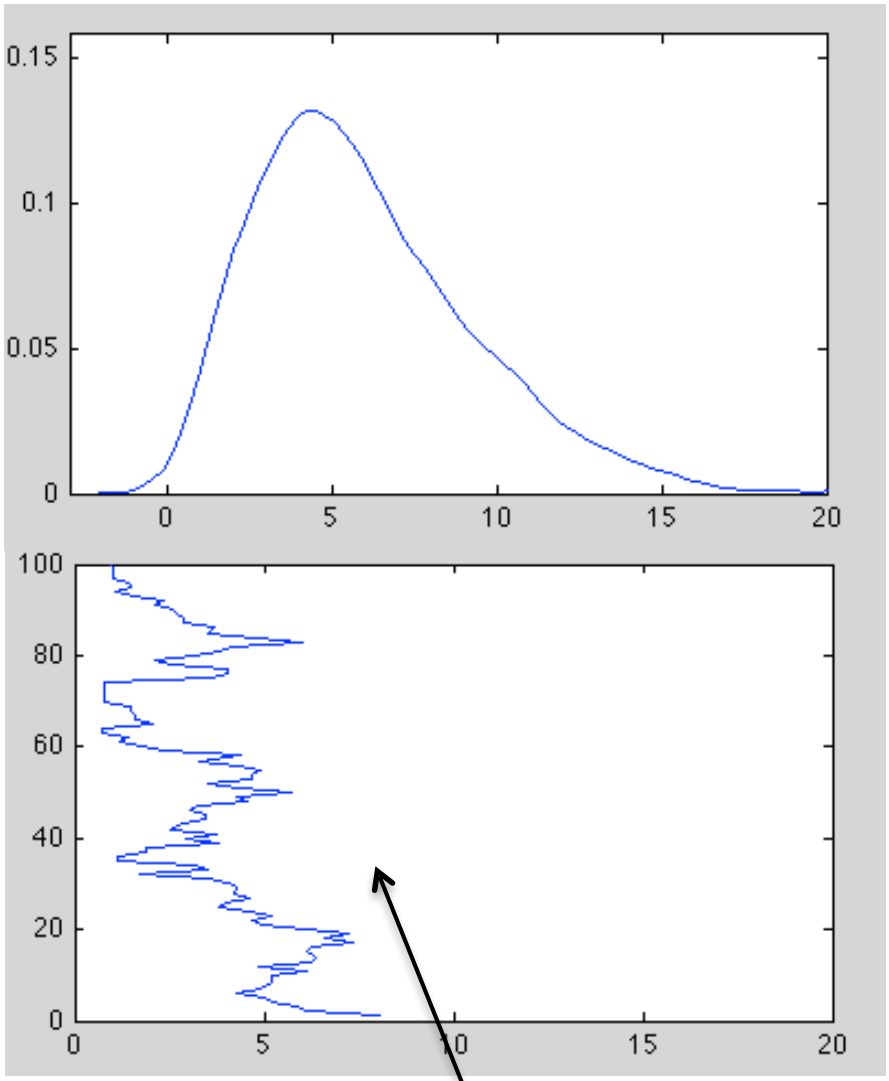


this comes from a numpy routine or something you wrote

Independent



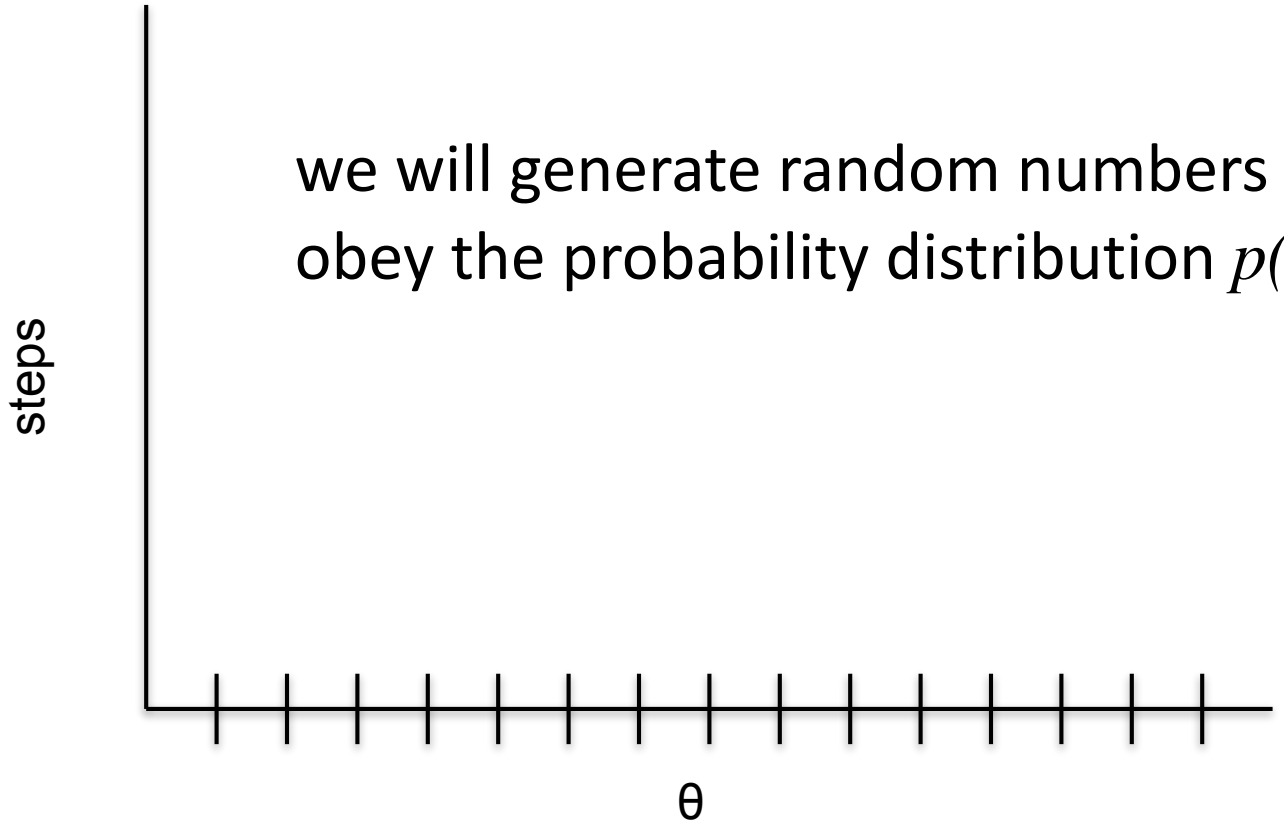
MCMC

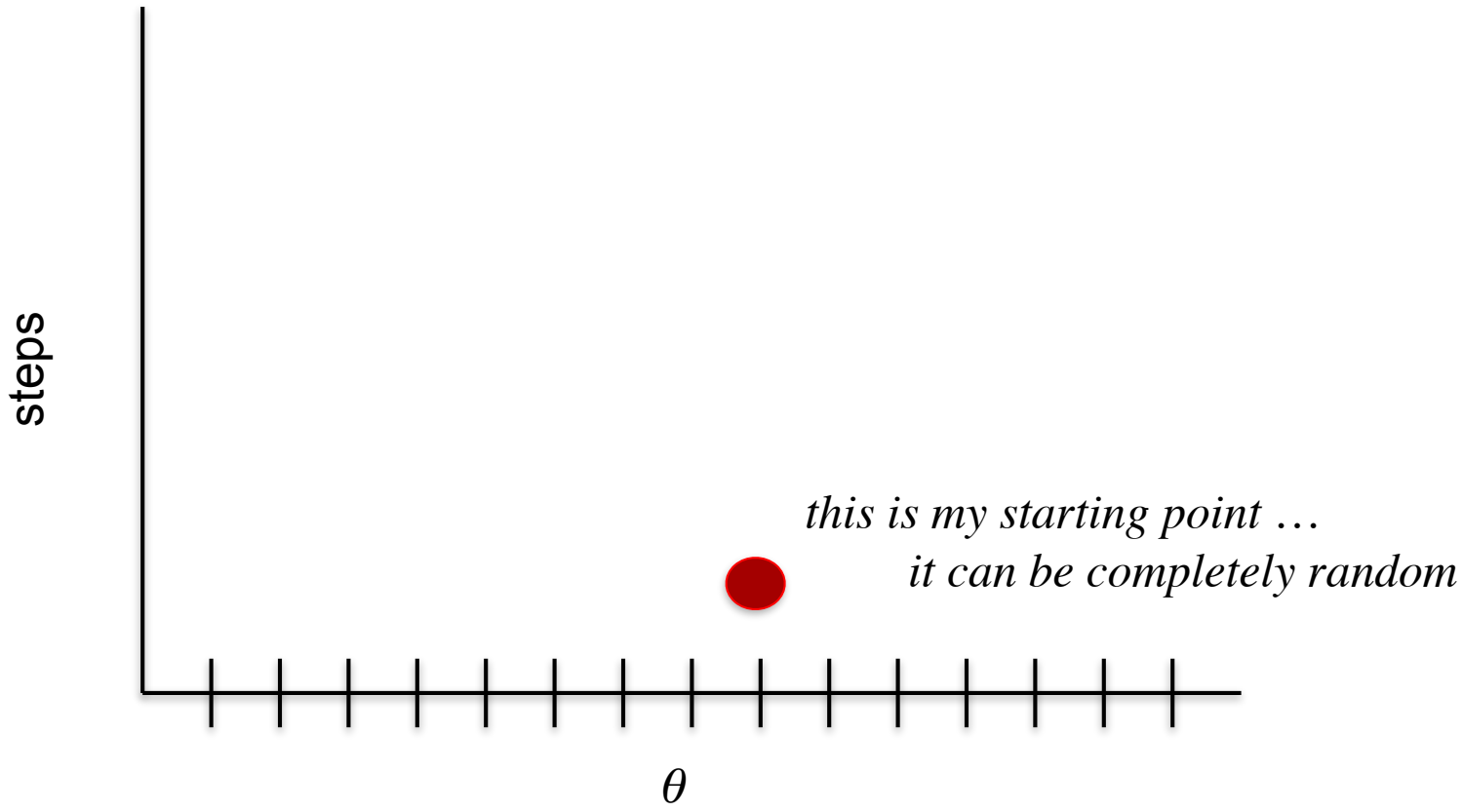


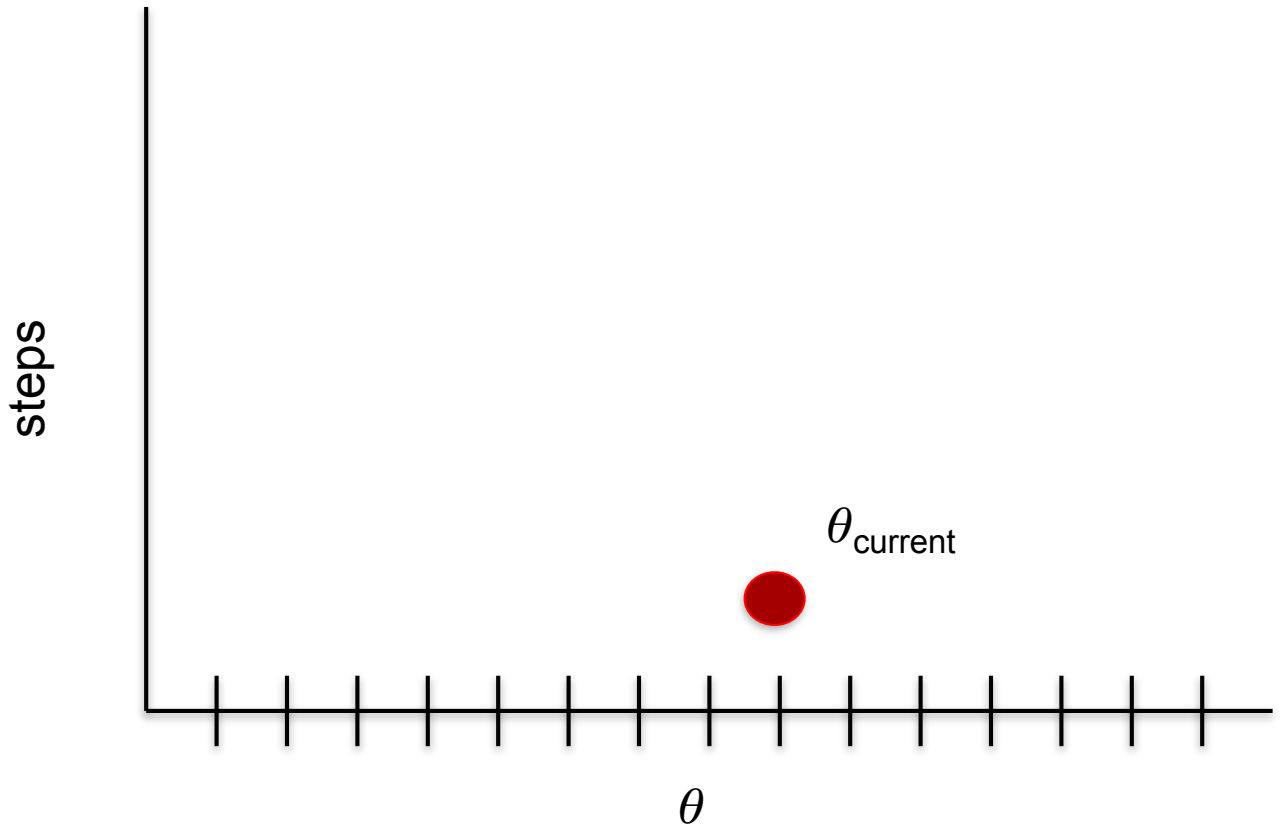
where does this come from?

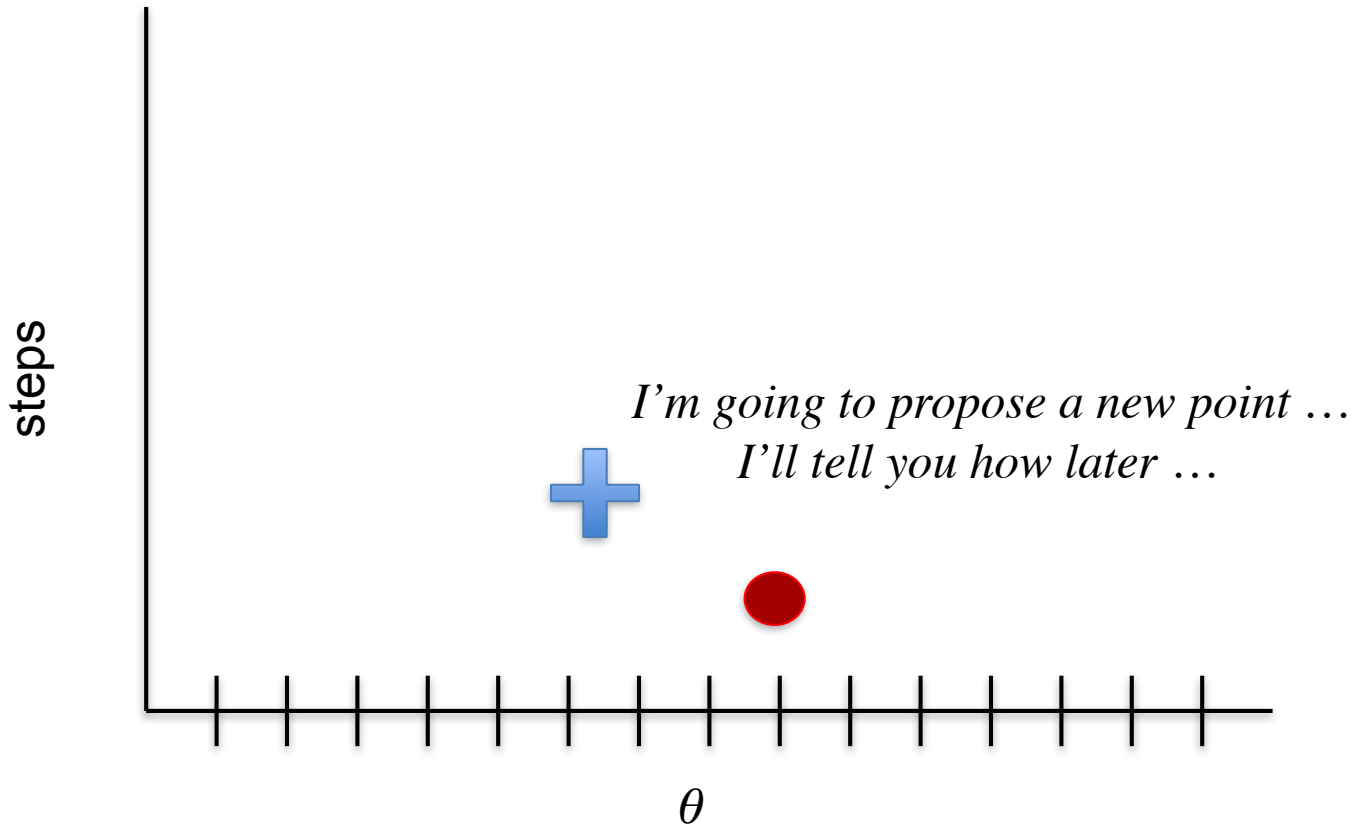
I'm going to show how you produce MCMC samples …

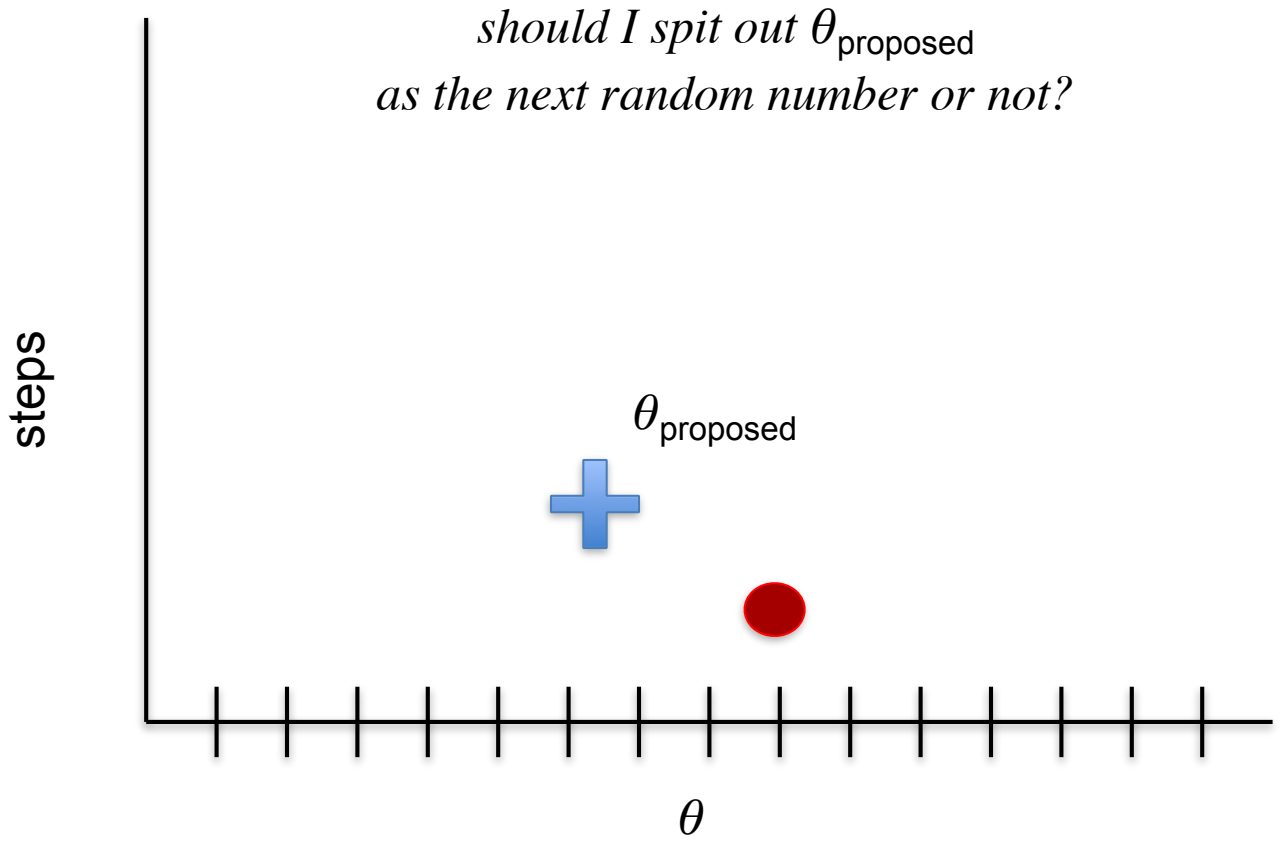
like before, the samples (random draws from the probability density function) are shown along the x axis … the steps (over time) are shown along the y axis …



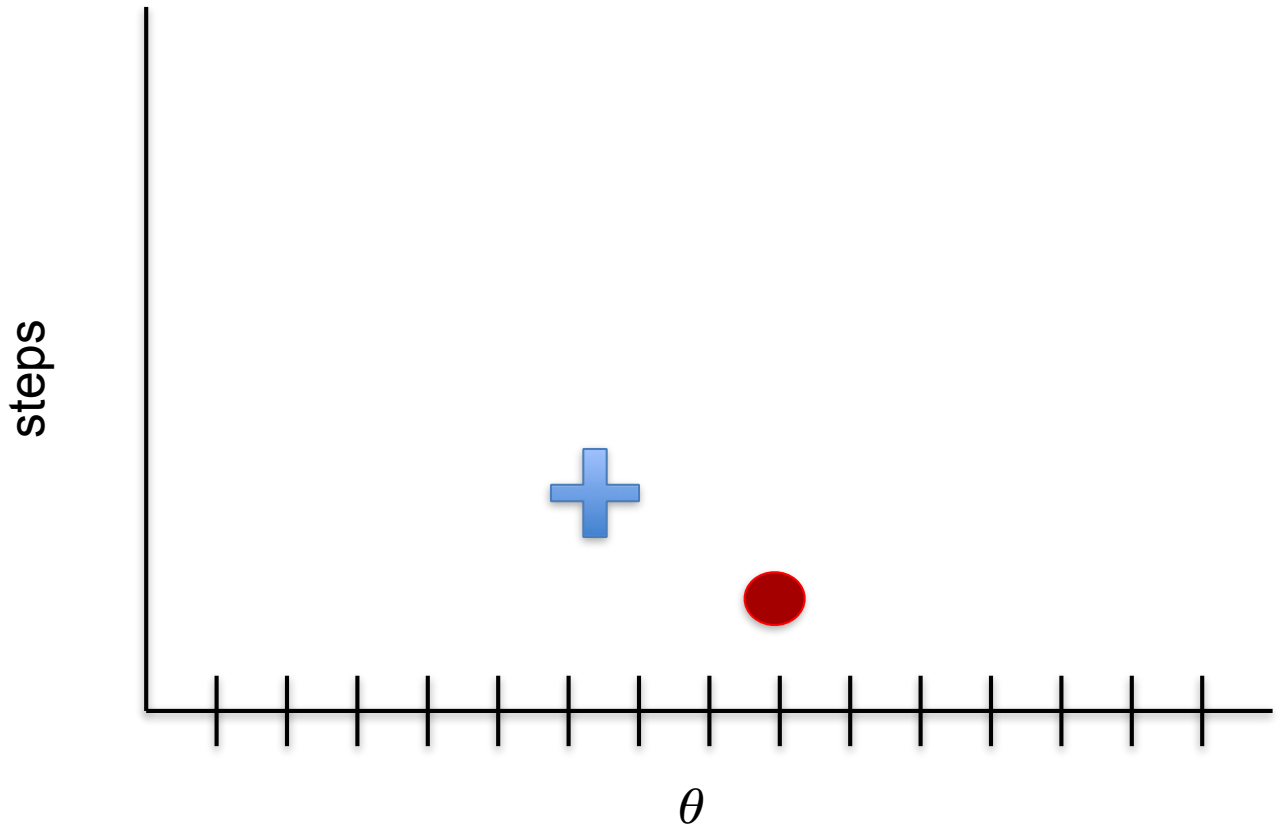

steps
this is my starting point …
it can be completely random
θ


steps
θ_{current}
θ


steps
I'm going to propose a new point …
I'll tell you how later …
θ


should I spit out θ_{proposed}
as the next random number or not?
steps
θ_{proposed}
θ

I'm going to move to (spit out) that new point probabilistically …

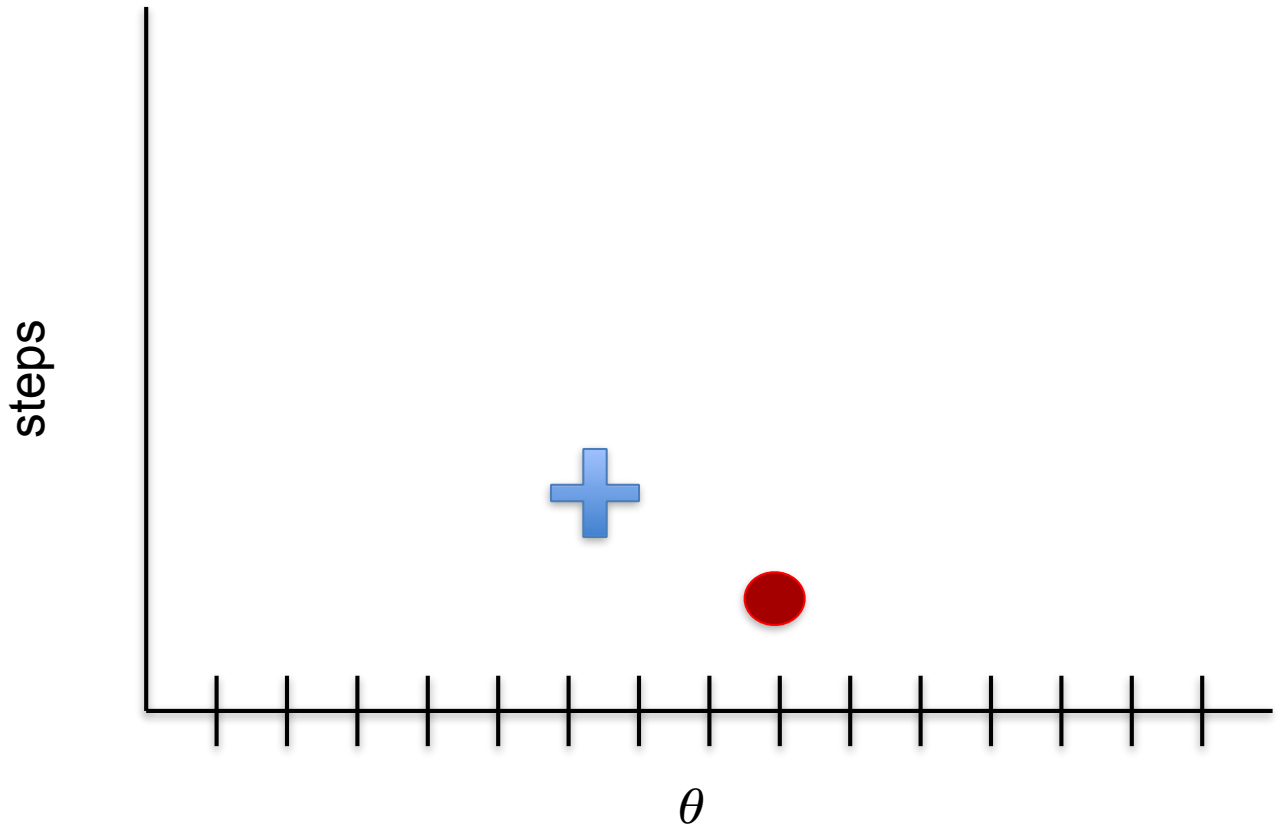


I'm going to move to (spit out) that new point probabilistically …

if $p(\theta_{\text{proposed}}) > p(\theta_{\text{current}})$

then move to the proposed point with probability 1

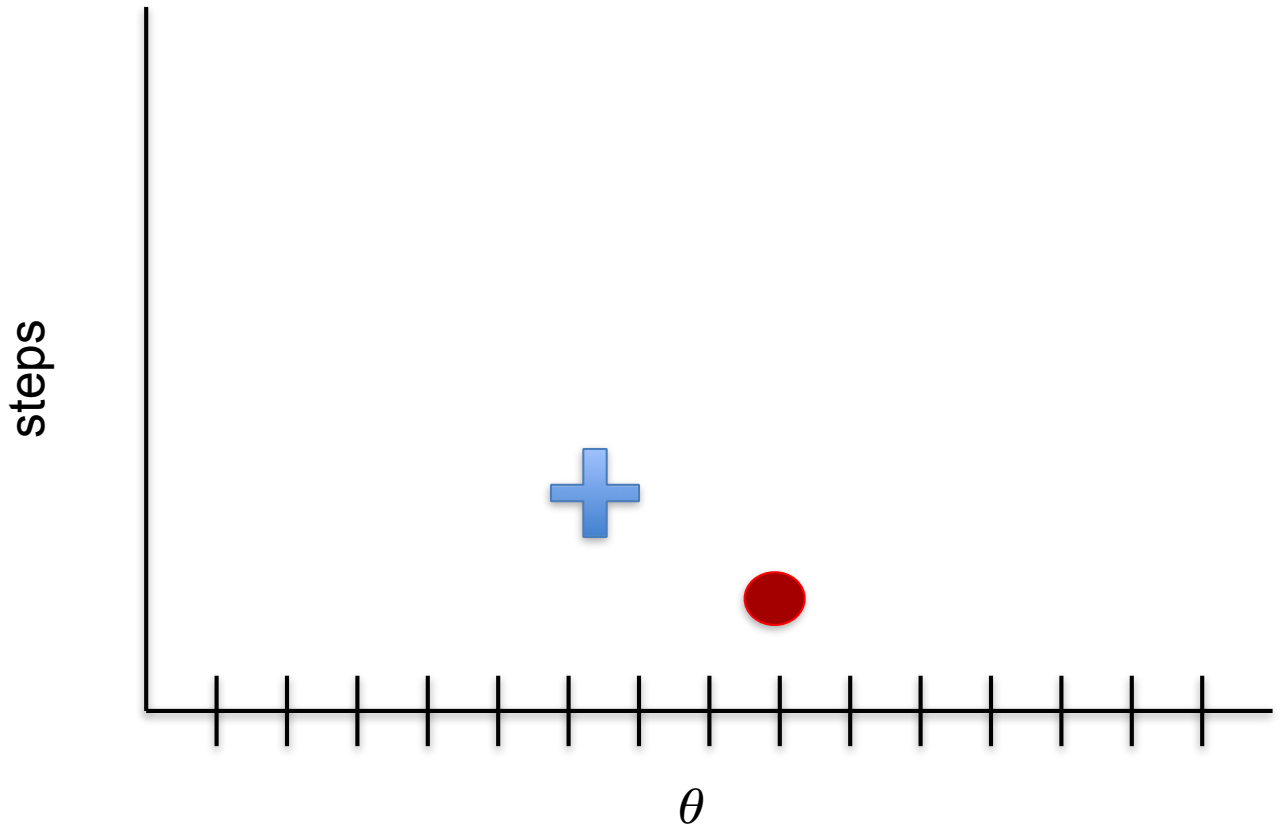
Note: I told you what p(θ) is - so you can calculate it for any value of θ.
You need MCMC because you want to generate random θ's from p(θ)



I'm going to move to (spit out) that new point probabilistically …

if $p(\theta_{proposed}) / p(\theta_{current}) > 1$

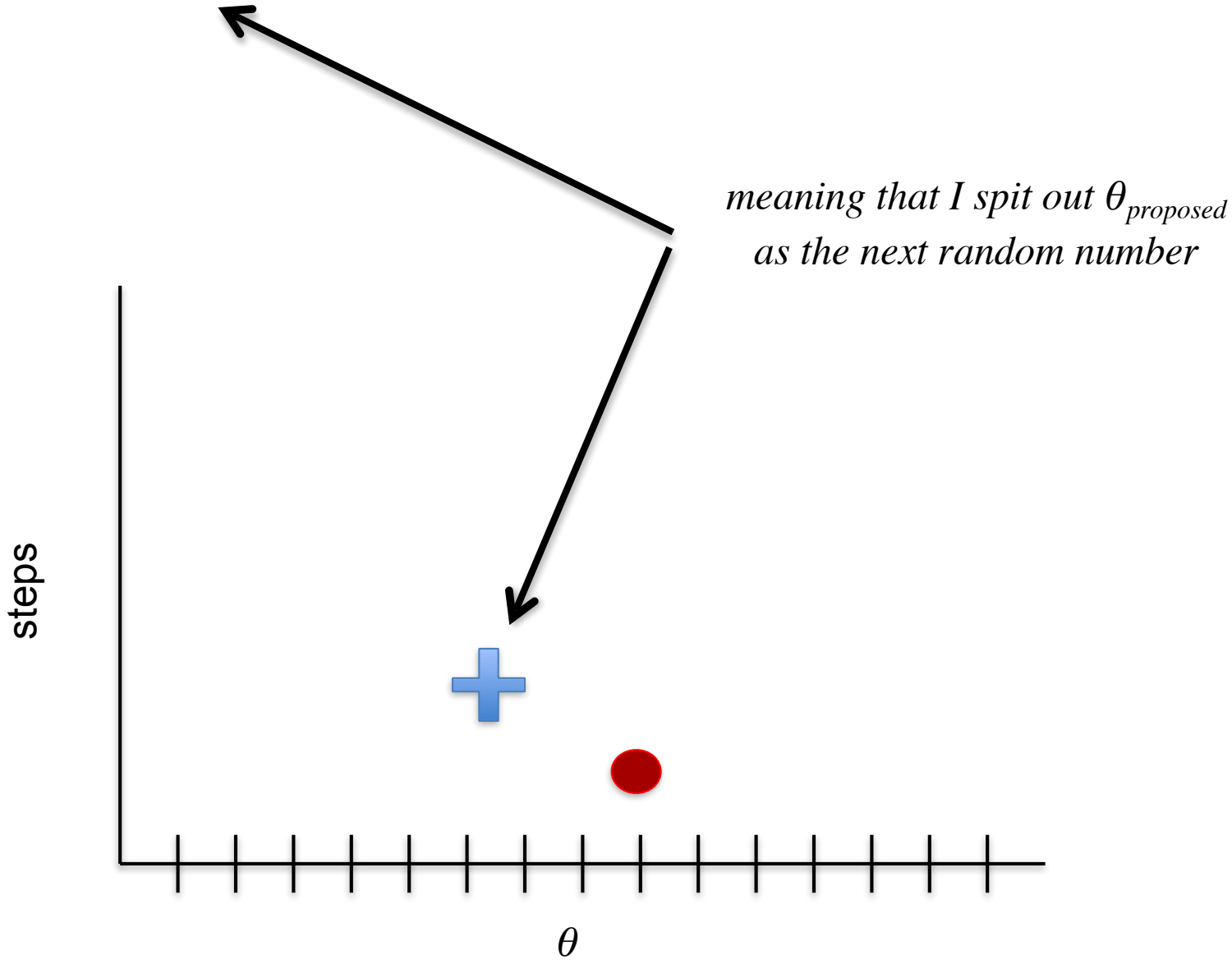
then move to the proposed point with probability 1



I'm going to move to (spit out) that new point probabilistically …

if $p(\theta_{proposed}) / p(\theta_{current}) > 1$

then move to the proposed point with probability 1



I'm going to move to (spit out) that new point probabilistically …

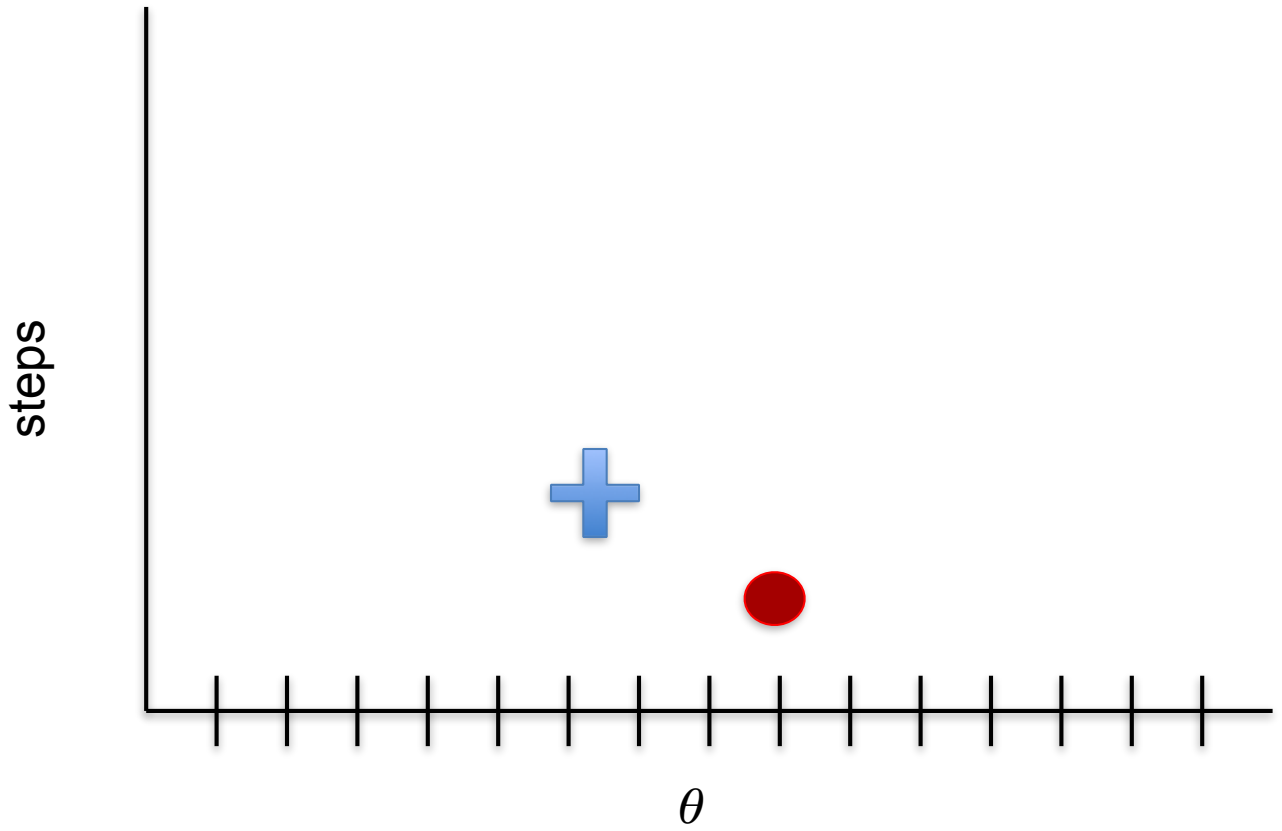
if $p(\theta_{proposed}) / p(\theta_{current}) > 1$

then move to the proposed point with probability 1

elseif $p(\theta_{proposed}) / p(\theta_{current}) \leq 1$

then move to the proposed point with probability $p(\theta_{proposed})/p(\theta_{current})$

otherwise stay where you are



I'm going to move to (spit out) that new point probabilistically …

if $p(\theta_{proposed}) / p(\theta_{current}) > 1$

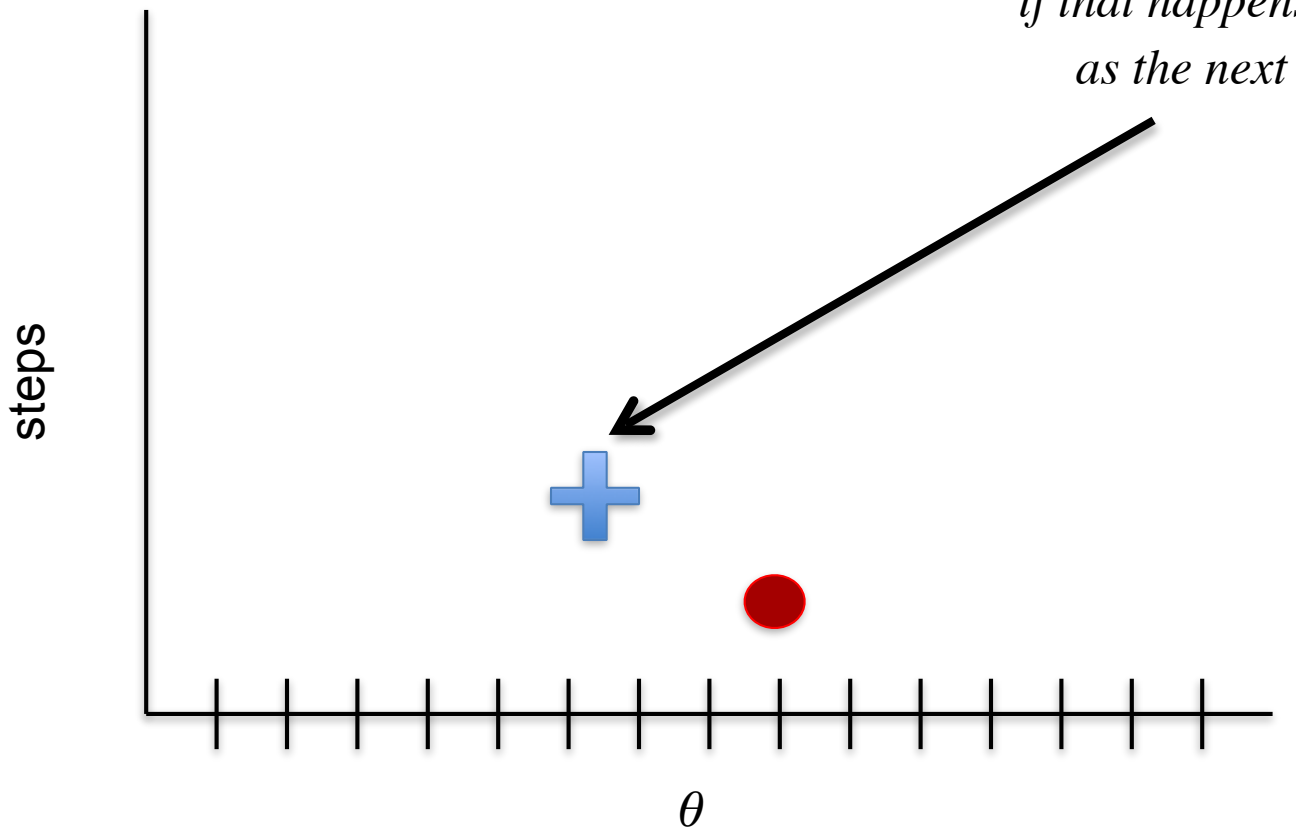
then move to the proposed point with probability 1

elseif $p(\theta_{proposed}) / p(\theta_{current}) \leq 1$

then move to the proposed point with probability $p(\theta_{proposed})/p(\theta_{current})$

otherwise stay where you are

if that happens, I spit out $\theta_{proposed}$ as the next random number



I'm going to move to (spit out) that new point probabilistically …

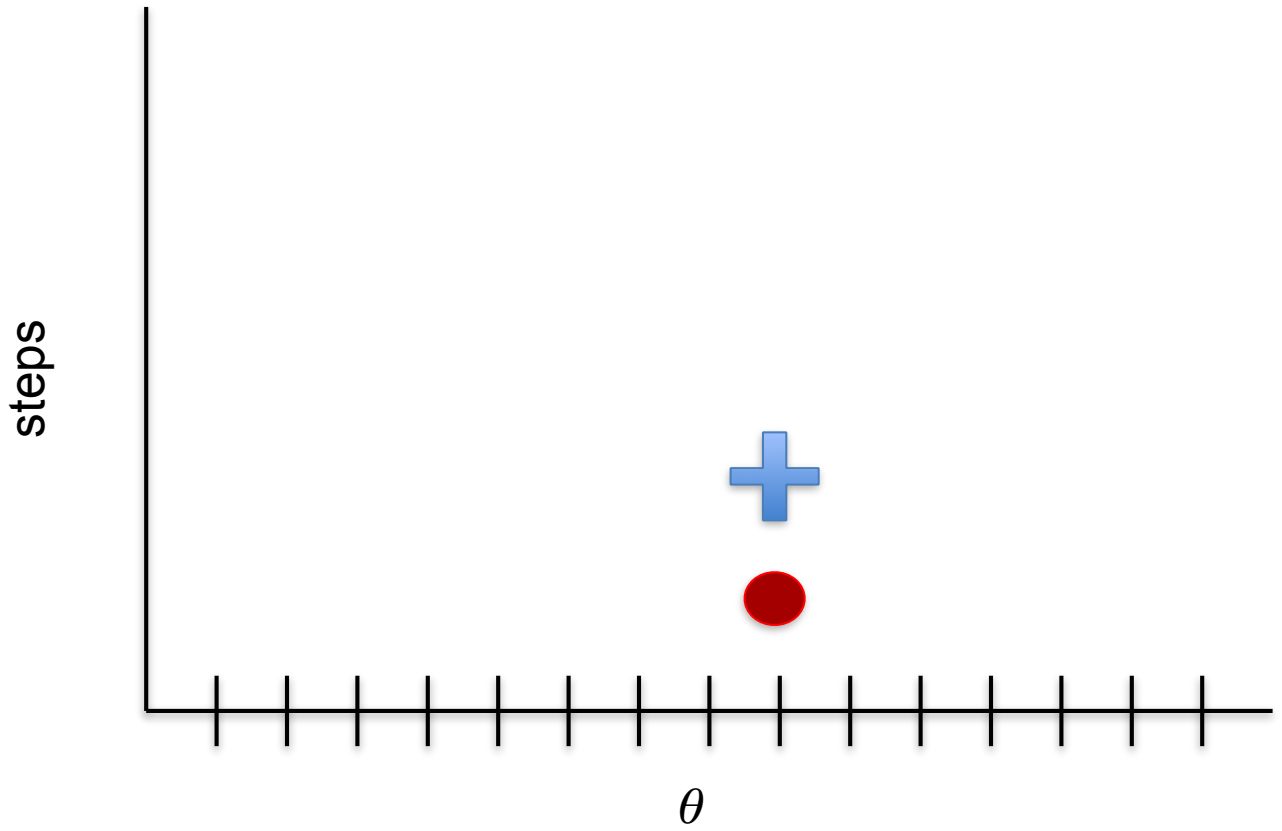
if $p(\theta_{proposed}) / p(\theta_{current}) > 1$

then move to the proposed point with probability 1

elseif $p(\theta_{proposed}) / p(\theta_{current}) \leq 1$

then move to the proposed point with probability $p(\theta_{proposed})/p(\theta_{current})$

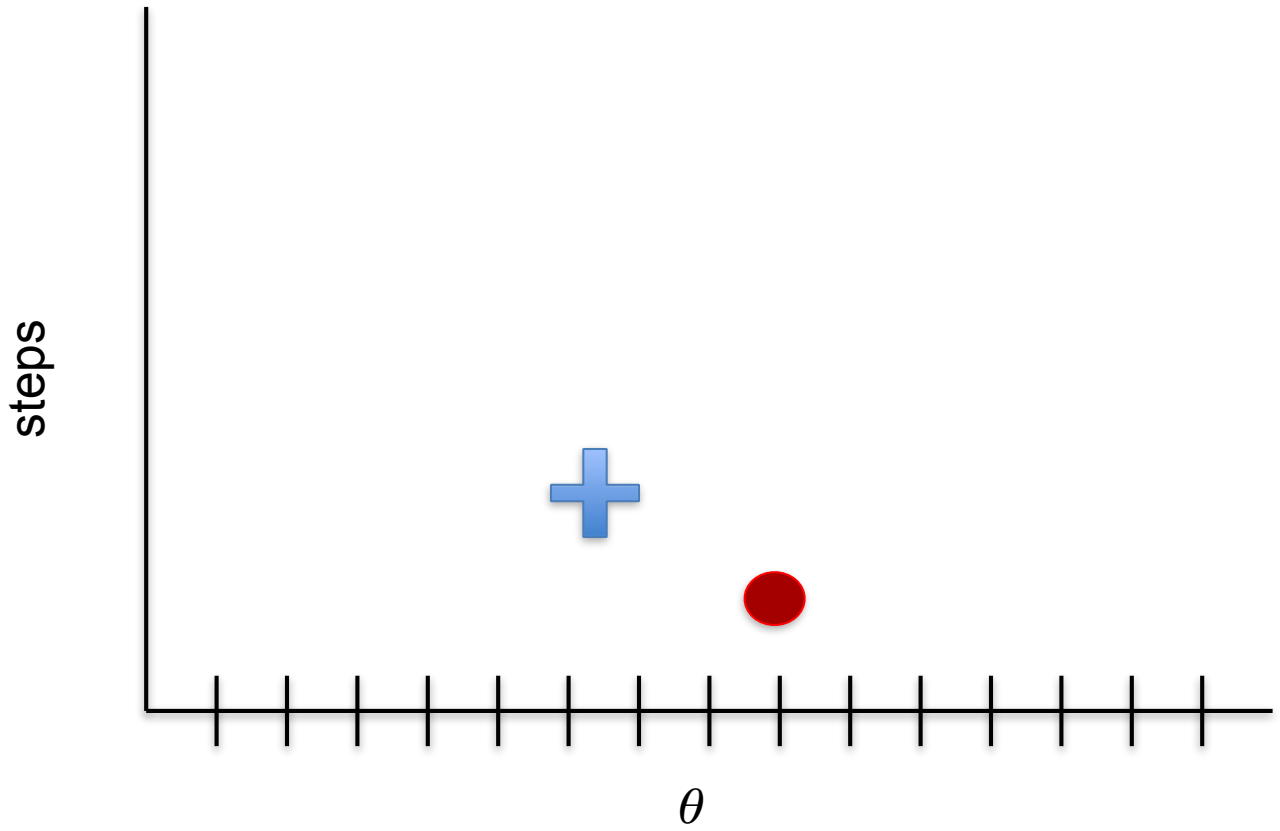
otherwise stay where you are and try again



I'm going to move to (spit out) that new point probabilistically …

$$P_{move} = \min\left(\frac{p(\theta_{proposed})}{p(\theta_{current})}, 1\right)$$

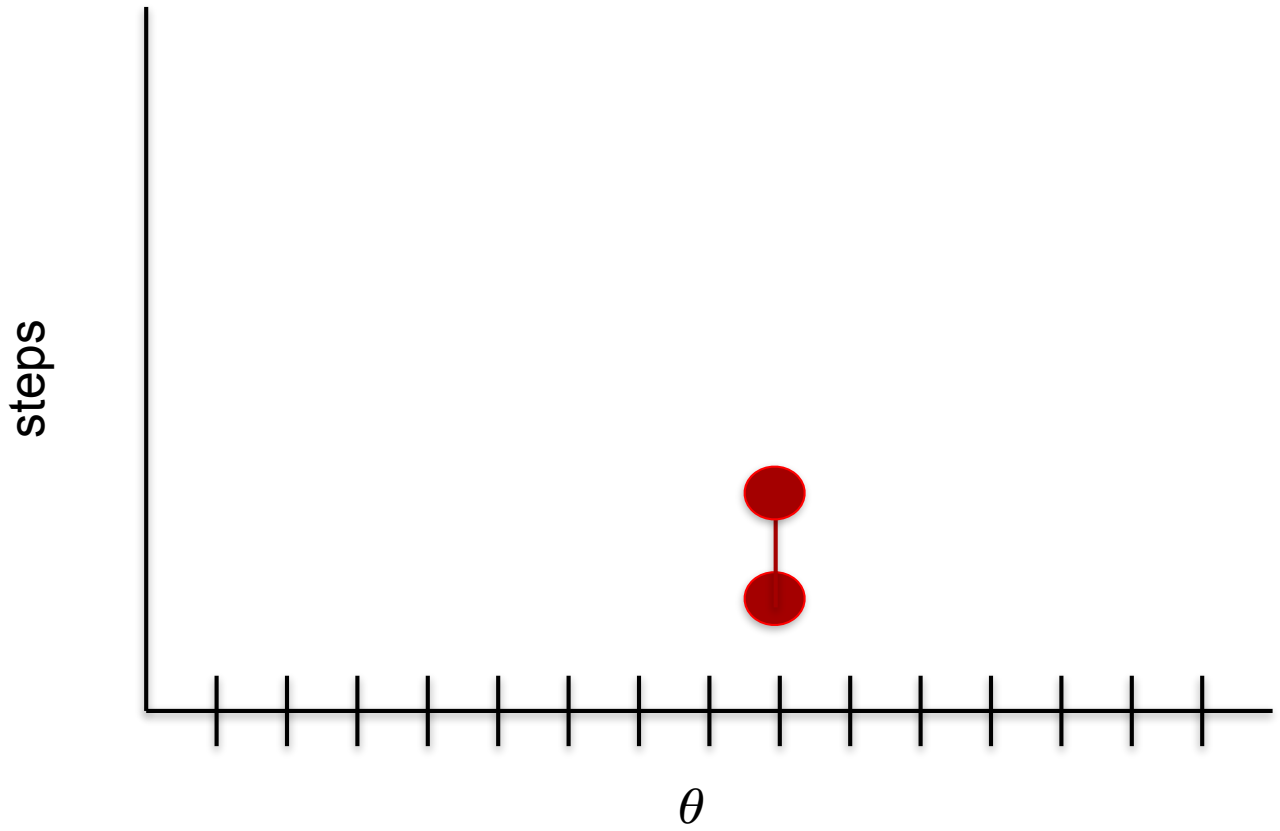
Metropolis Algorithm (Metropolis et al., 1953)



I'm going to move to (spit out) that new point probabilistically …

$$P_{move} = \min\left(\frac{p(\theta_{proposed})}{p(\theta_{current})}, 1\right)$$

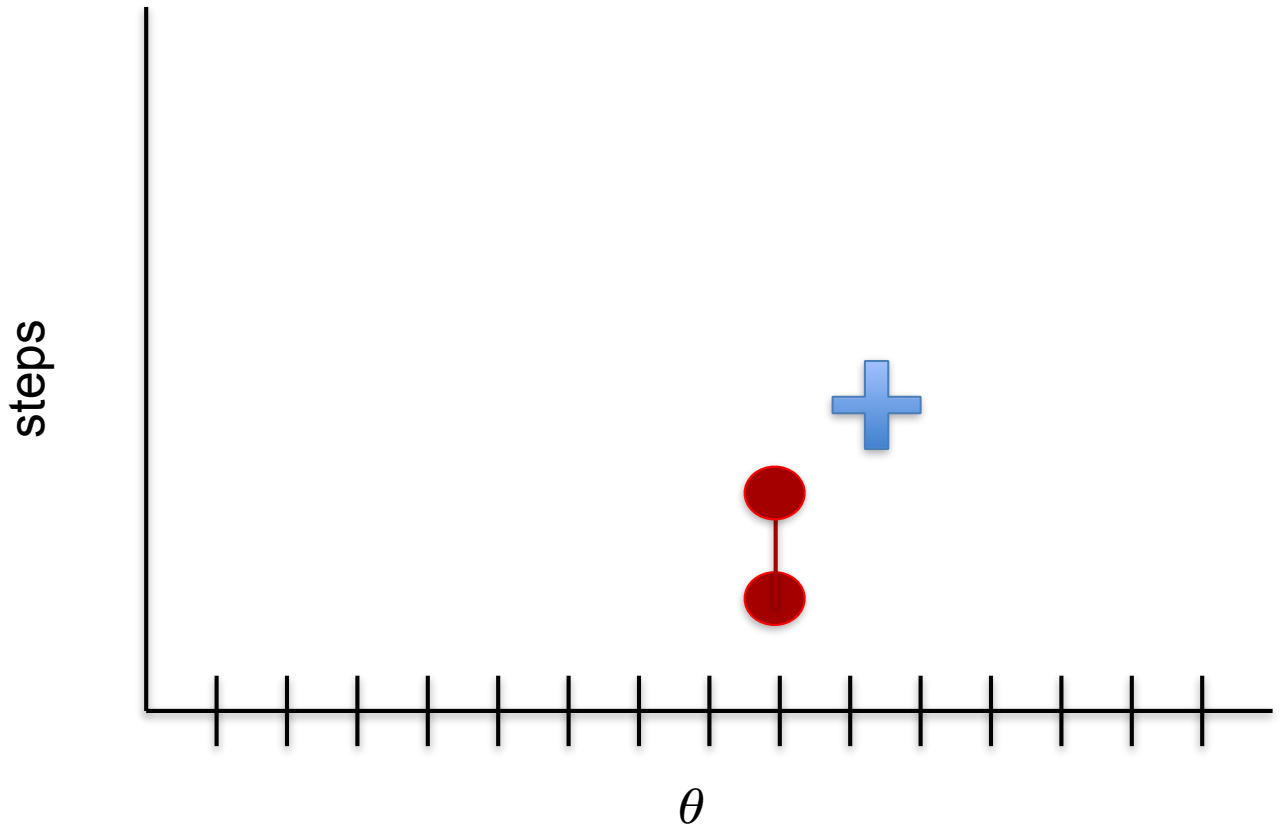
Metropolis Algorithm (Metropolis et al., 1953)



I'm going to move to (spit out) that new point probabilistically …

$$P_{move} = \min\left(\frac{p(\theta_{proposed})}{p(\theta_{current})}, 1\right)$$

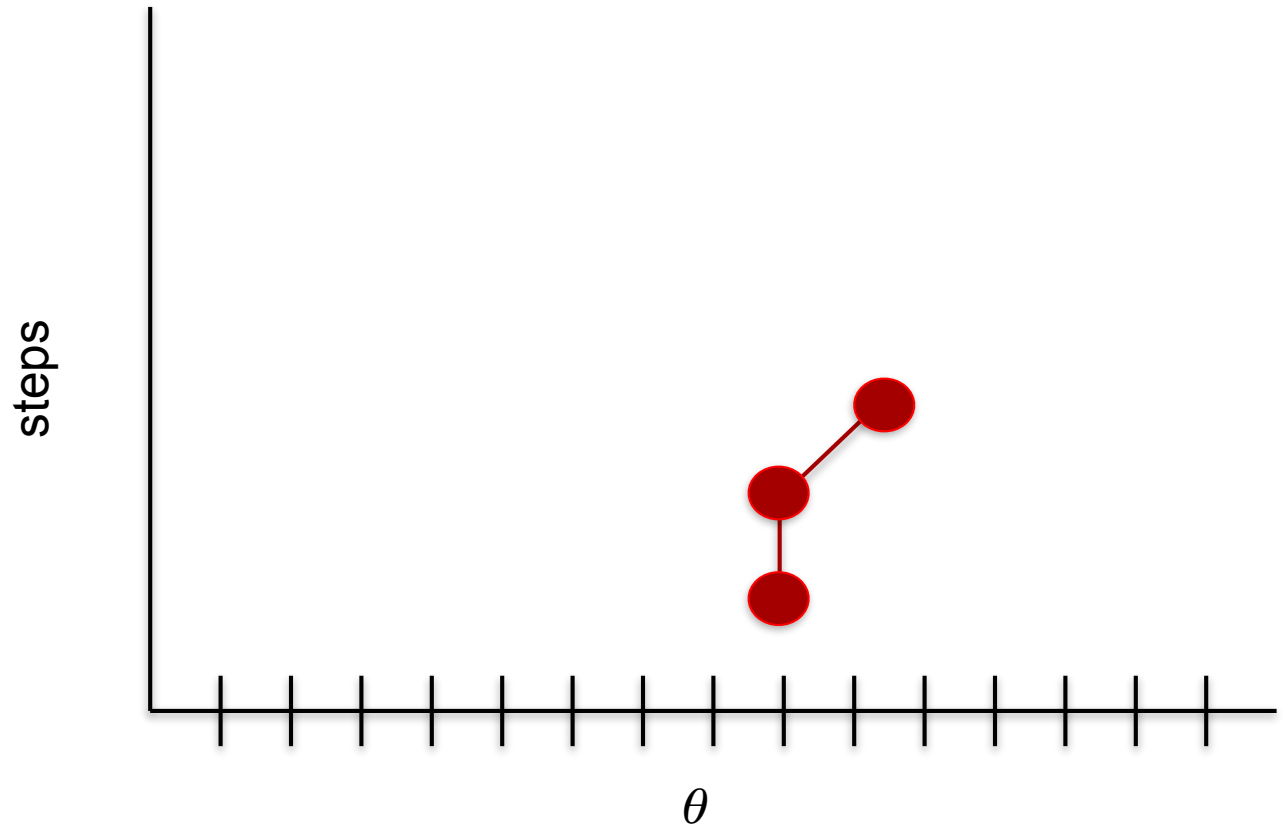
Metropolis Algorithm (Metropolis et al., 1953)



I'm going to move to (spit out) that new point probabilistically …

$$P_{move} = \min\left(\frac{p(\theta_{proposed})}{p(\theta_{current})}, 1\right)$$

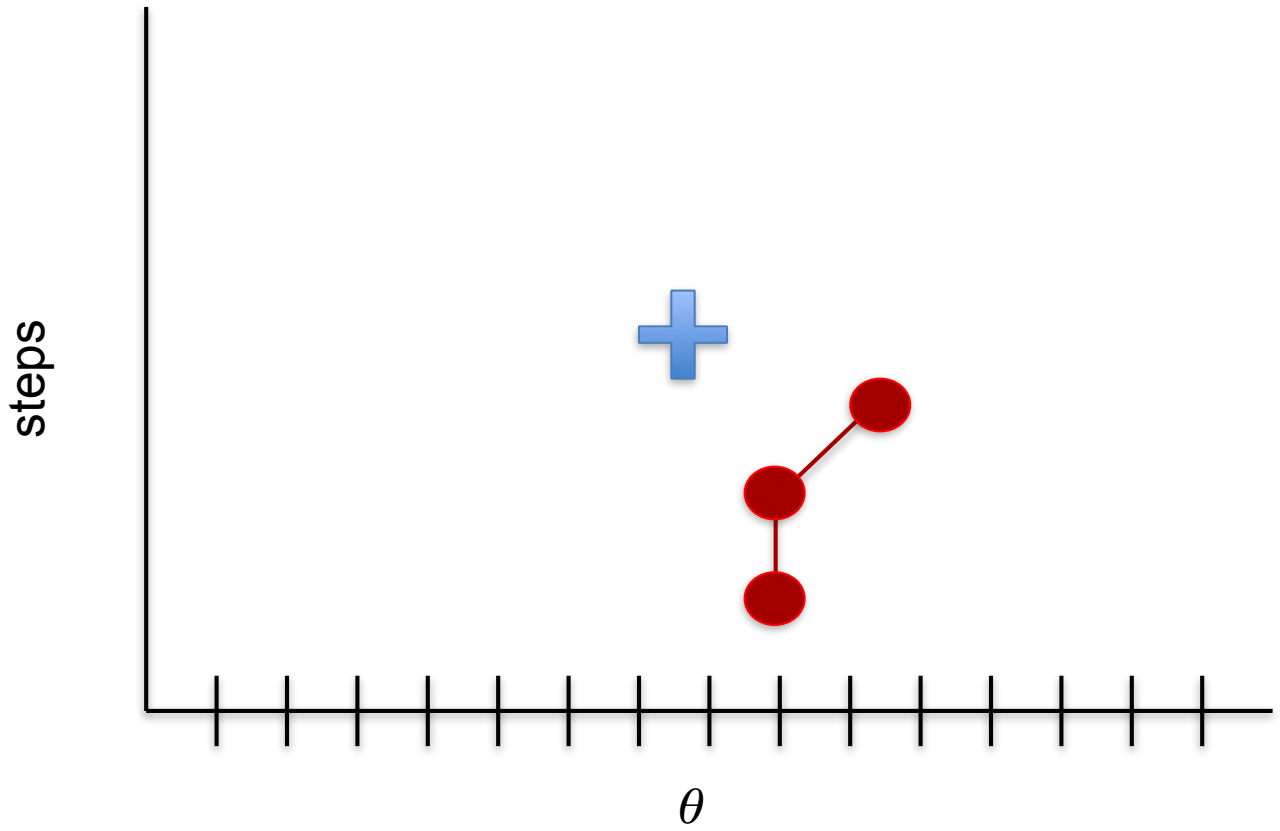
Metropolis Algorithm (Metropolis et al., 1953)



I'm going to move to (spit out) that new point probabilistically …

$$P_{move} = \min\left(\frac{p(\theta_{proposed})}{p(\theta_{current})}, 1\right)$$

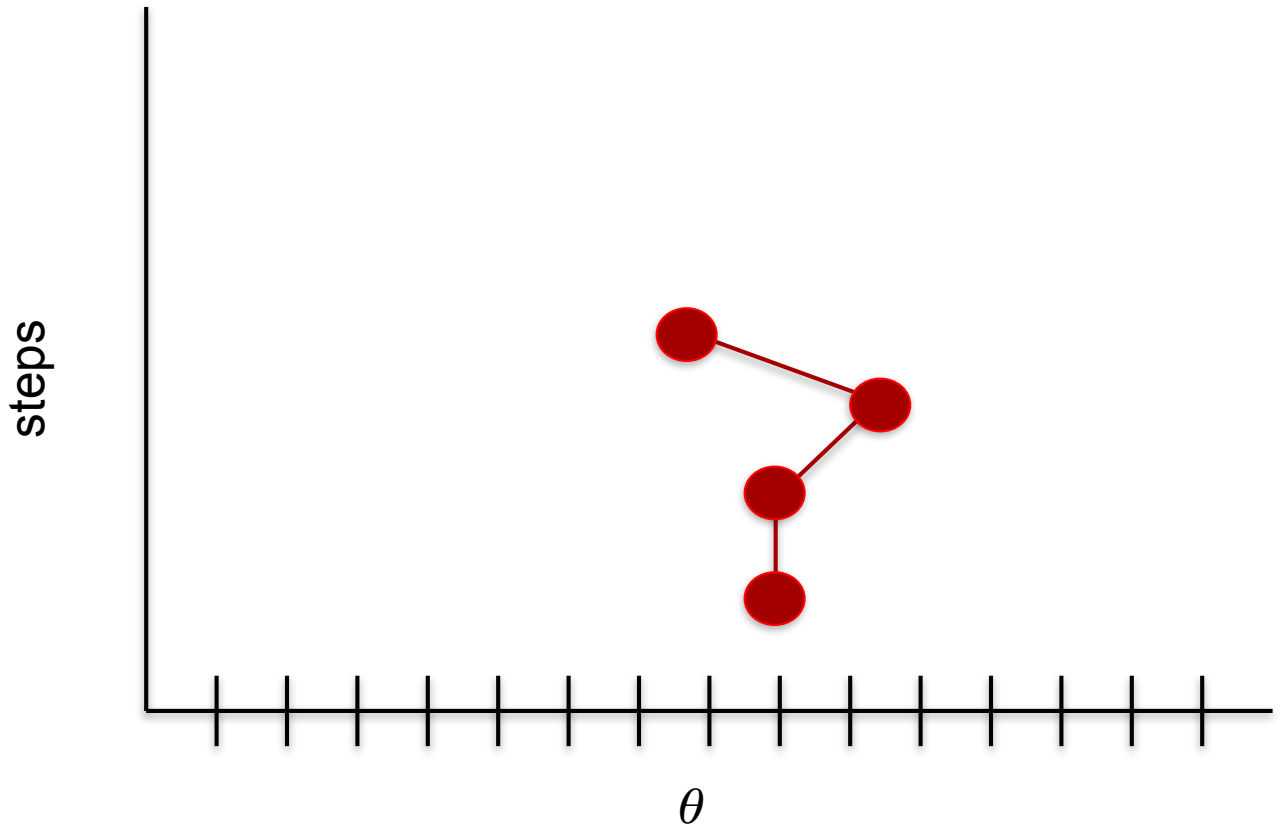
Metropolis Algorithm (Metropolis et al., 1953)



I'm going to move to (spit out) that new point probabilistically …

$$P_{move} = \min\left(\frac{p(\theta_{proposed})}{p(\theta_{current})}, 1\right)$$

Metropolis Algorithm (Metropolis et al., 1953)

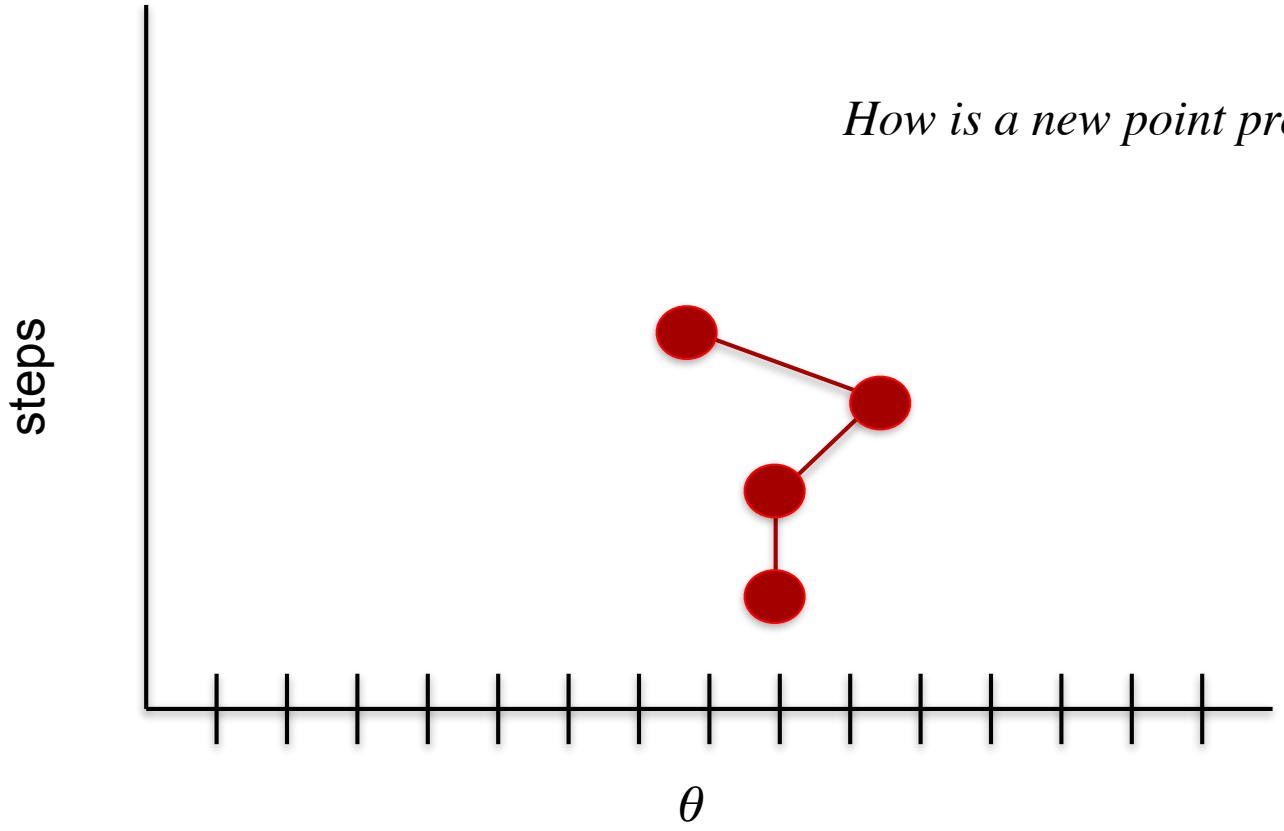


I'm going to move to (spit out) that new point probabilistically …

$$P_{move} = \min\left(\frac{p(\theta_{proposed})}{p(\theta_{current})}, 1\right)$$

Metropolis Algorithm (Metropolis et al., 1953)

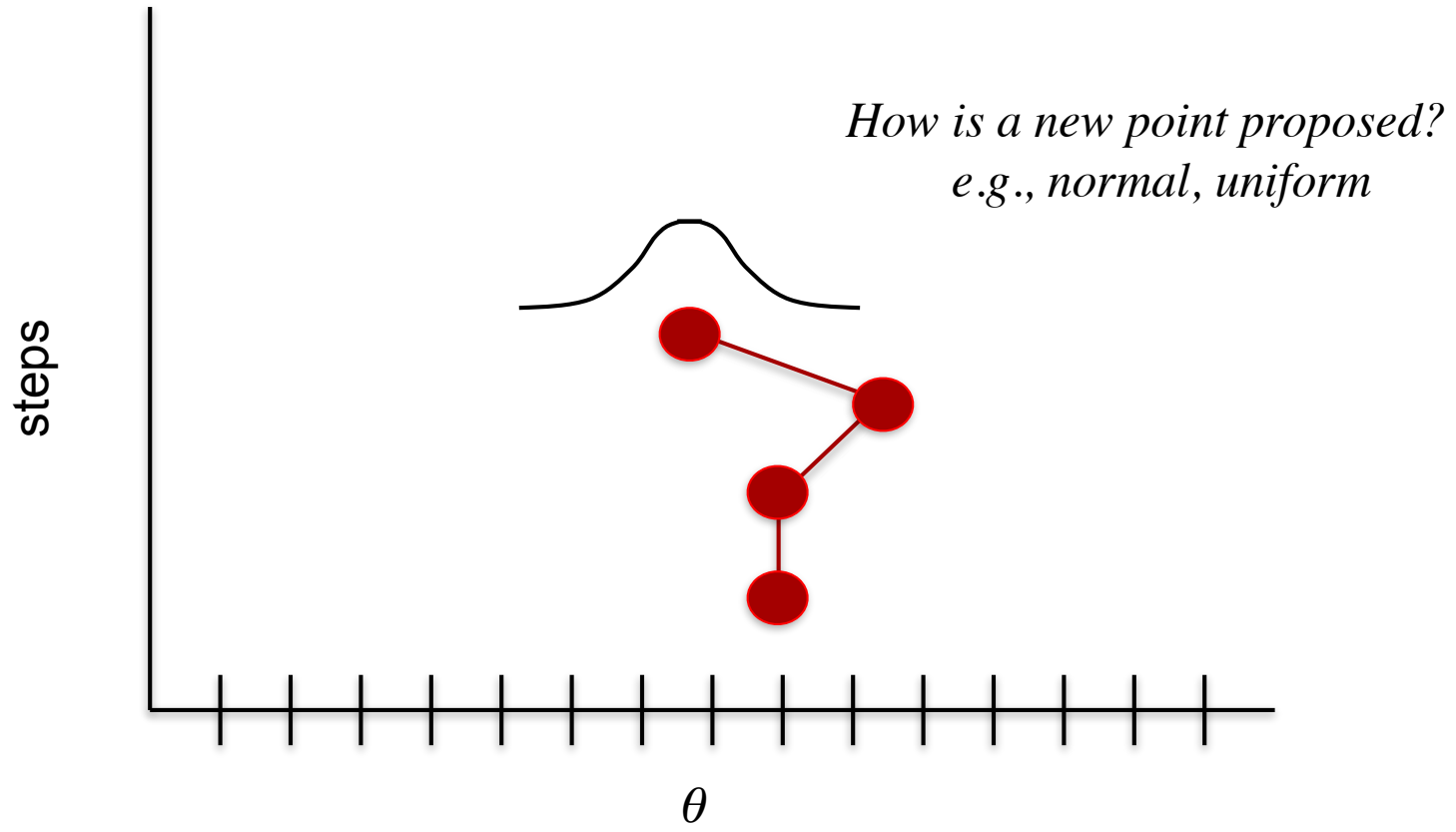
How is a new point proposed?



I'm going to move to (spit out) that new point probabilistically …

$$P_{move} = \min\left(\frac{p(\theta_{proposed})}{p(\theta_{current})}, 1\right)$$

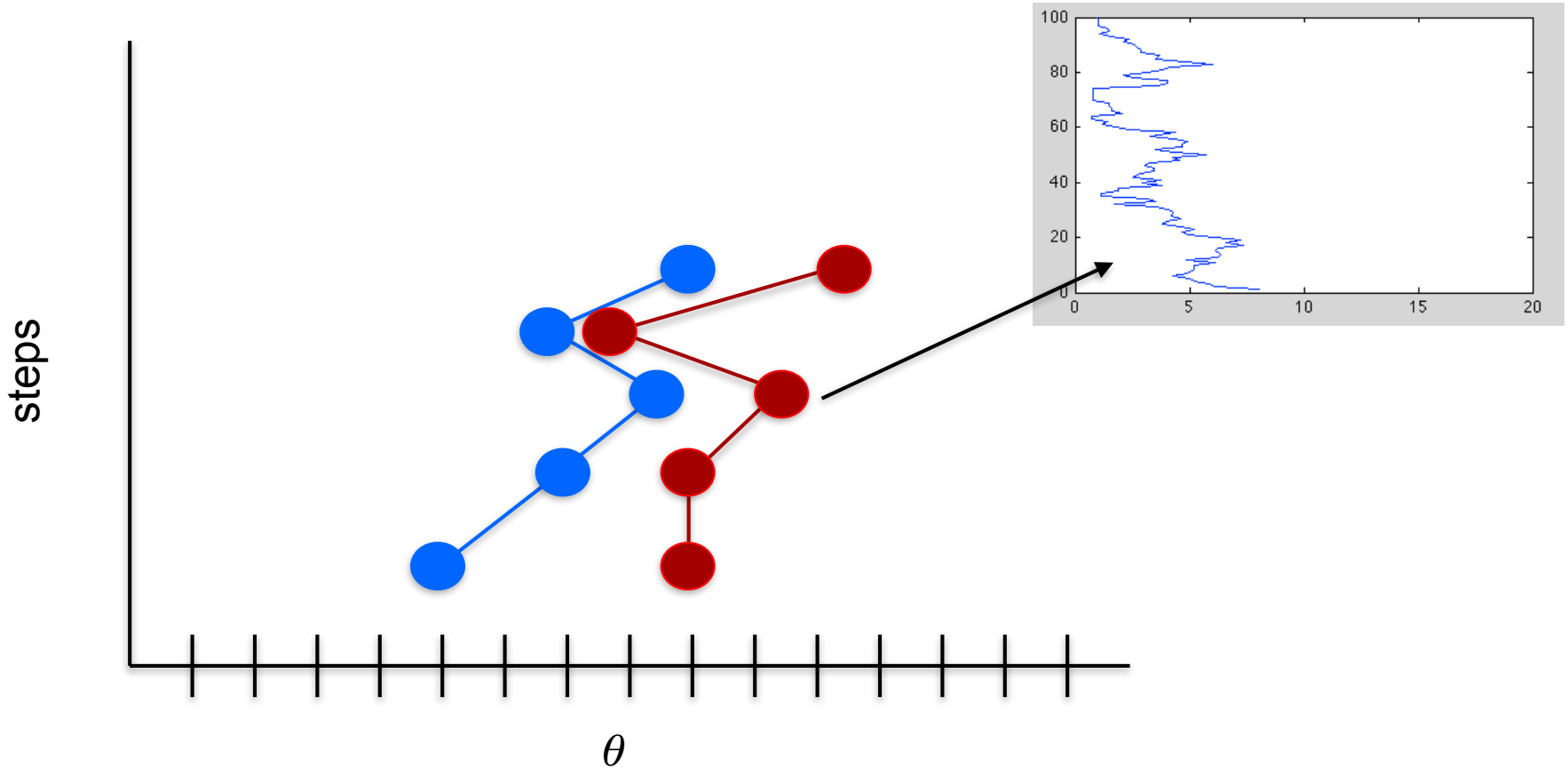
Metropolis Algorithm (Metropolis et al., 1953)



sometimes need to do 10's or even 100's of thousands of steps …
and keep only every 100th step

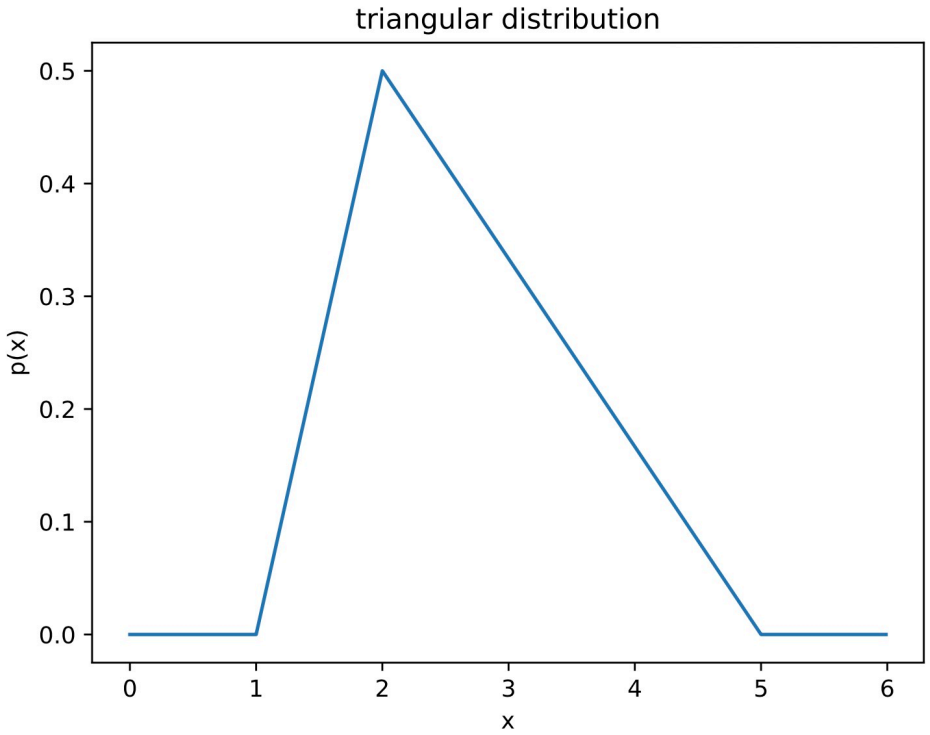
And allow a "burn-in" period that's discarded (the initial walk)

Multiple chains can be used to judge convergence …
e.g., if 3 chains with 3 different starting points all converge
to the same underlying distribution (after deleting the burn-in)

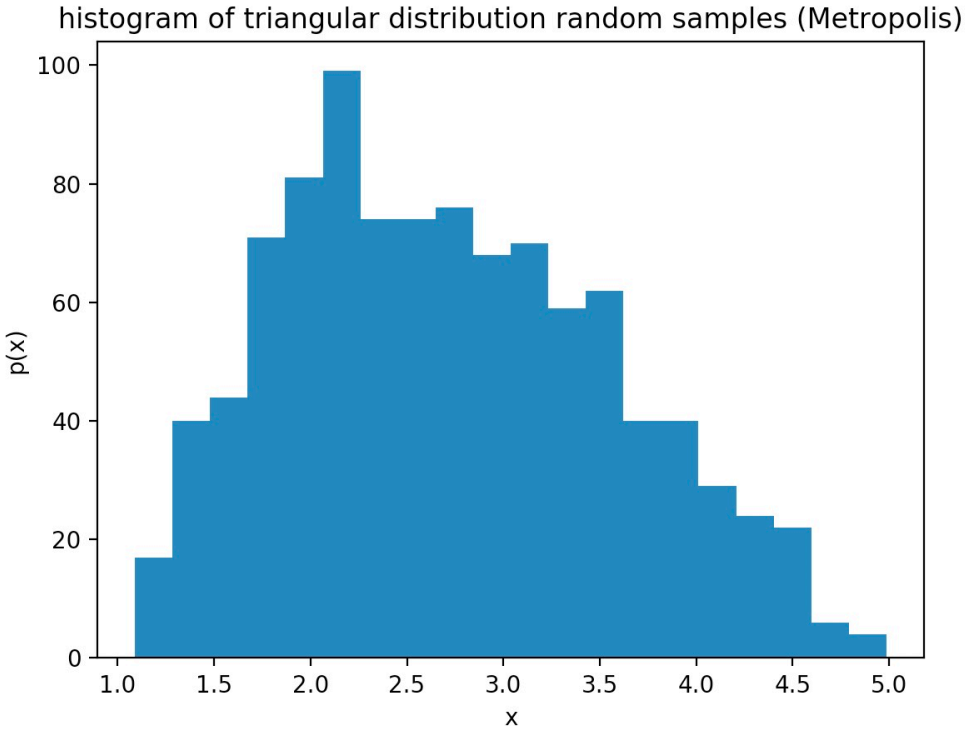


Triangular Distribution

plot probability distribution



generate random numbers



using Metropolis algorithm

Metropolis algorithm (symmetric proposal distribution)

Let $p(x)$ be the desired probability distribution, (for example, a triangular distribution); note that the Wikipedia page formalizes using $f(x)$, a function proportional to $p(x)$, which is the approach used, for example, to generate random numbers for a Bayesian posterior distribution.

1. Initialization:

- Choose an arbitrary point $x0$ to be the first sample. The first point in the chain xt will be equal to $x0$. If you know the range of the distribution, it makes sense to pick a first sample $x0$ that is within the range of the distribution.
- Choose a proposal distribution $g(xp|xt)$, where xt is the current point in the chain and xp is the proposed point in the chain. For the Metropolis algorithm, the proposal distribution needs to be symmetric (Metropolis-Hastings generalizes to allow for non-symmetric proposal distributions). The simplest choice for $g(xp|xt)$ is to assume that xp is drawn from a normal distribution with mean xt and standard deviation σ; σ should be small, but not too small. In python, to generate a random number with mean xt and standard deviation σ, you would use the following code:

```
xp = R.normal(xt, sigma)
```

2. For each iteration:

- Generate: Generate a candidate xp from $g(xp|xt)$ using the above.
- Calculate: Calculate the acceptance ratio $A = p(xp)/p(xt)$, which is used to decide whether to accept or reject the proposed candidate.
- Accept or Reject:
  - Generate a uniform random number U on [0,1) using `R.uniform()`.
  - If $U <= A$ accept the proposed candidate by setting $xt = xp$.
  - If $U > A$ reject the proposed candidate and keep xt at the same number.

Each new accepted xt produced on an iteration is a candidate random number produced by the Metropolis algorithm.

https://en.wikipedia.org/wiki/Metropolis–Hastings_algorithm

$$P_{move} = \min\left(\frac{p(\theta_{proposed})}{p(\theta_{current})}, 1\right)$$

Metropolis Algorithm (Metropolis et al., 1953)

One important use of MCMC techniques
is Bayesian statistical analysis.

Bayes is foundational to data science, statistics, biostatistics, machine learning, and lots of other applications.

$$p(\theta \mid D) = \frac{p(D \mid \theta)\, p(\theta)}{p(D)}$$

$$p(\theta \mid D) = \frac{p(D \mid \theta)\, p(\theta)}{\sum_i p(D \mid \theta_i)\, p(\theta_i)}$$

Bayes' Rule