|  |
| --- |
| UCCS |
| Monte Carlo Markov Chains Hellinger Convergence Metrics |
| [Type the document subtitle] |

|  |
| --- |
| Broberg, Ronald  [Pick the date] |

Table of Contents

[1 Introduction 2](#_Toc447442953)

[2 Monte Carlo Integration 3](#_Toc447442954)

[2.1 Example: Approximating the integral 5](#_Toc447442955)

[2.2 Example: Approximating the expected value of the Beta distribution 6](#_Toc447442956)

[2.3 Monte Carlo Approximation for Optimization 7](#_Toc447442957)

[2.4 Example: Monte Carlo Optimization of *g(x) = e^(-(x-4)^2/2)* 8](#_Toc447442958)

[2.5 Summary on Monte Carlo Approximation 9](#_Toc447442959)

[3 Markov Chains 9](#_Toc447442960)

[3.1 Example: Predicting the weather with a finite state-space Markov chain 10](#_Toc447442961)

[3.2 Continuous state-space Markov chains 12](#_Toc447442962)

[3.3 Example: Sampling from a continuous distribution using continuous state-space Markov chains 12](#_Toc447442963)

[3.4 Markov Chain Summary 13](#_Toc447442964)

[4 MCMC: Metropolis Sampling 13](#_Toc447442965)

[4.1 Metropolis Sampling 13](#_Toc447442966)

[4.2 Example: Using the Metropolis algorithm to sample from an unknown distribution 14](#_Toc447442967)

[4.3 Reversibility of the transition operator 15](#_Toc447442968)

[5 MCMC: Metropolis-Hastings Sampling 15](#_Toc447442969)

[5.1 Example: Sampling from a Bayesian posterior with improper prior 16](#_Toc447442970)

[6 Hellinger Convergence 18](#_Toc447442971)

[1 Appendix: MCMC Monte Carlo Approximation of Integral Code 19](#_Toc447442972)

[2 Appendix: MCMC Monte Carlo Beta Expectation Code 20](#_Toc447442973)

[3 Appendix: MCMC Monte Carlo Optimization of Exponential Function Code 22](#_Toc447442974)

[4 Appendix: MCMC Markov Chain Finite State Transitions Code 23](#_Toc447442975)

[5 Appendix: MCMC Markov Chain Continous State Transitions Code 24](#_Toc447442976)

[6 Appendix: MCMC Metropolis Markov Sampler Code 25](#_Toc447442977)

[7 Appendix: MCMC Metropolis Hastings Priors and Posterior Code 27](#_Toc447442978)

# Introduction

This is the story of the evolution of MCMC methods. It begins with a single paper, one with no antecedent. The original idea required the right combination of place, people, and perspective. The place was Los Alamos right after World War II. The people included the familiar—von Neumann, Ulam, Teller—along with several less familiar. The perspective was that randomness and sampling could be used to circumvent insurmountable analytic roadblocks. There was also one last necessary ingredient present: a computer (Richey 2010)

This new approach first appeared in Ulam and Metropolis’s 1949 paper *The Monte*

*Carlo Method* [**49**]. (Richey 2010)

“The Monte Carlo method is an application of the laws of probability and statistics to the natural sciences. The essence of the method is to use various distributions of the random numbers, each distribution reflecting a particular process in a sequence of processes such as diffusion of neutrons in various materials, to calculate samples that approximate the real diffusion history. … The computer made the approach extremely useful for many physics problems.

Metopolis was also involved in the development of an importance-sampling scheme, called the Metropolis algorithm, that improves the effectiveness of the Monte Carlo method.” (Anderson, 1986)

*A recent survey places the Metropolis algorithm among the ten algorithms that have had the*

*greatest influence on the development and practice of science and engineering in the 20th*

*century (Beichl&Sullivan, 2000). This algorithm is an instance of a large class of sampling*

*algorithms, known as Markov chain Monte Carlo (MCMC). These algorithms have played*

*a significant role in statistics, econometrics, physics and computing science over the last*

*two decades. There are several high-dimensional problems, such as computing the volume*

*of a convex body in d dimensions, for which MCMC simulation is the only known general*

*approach for providing a solution within a reasonable time (polynomial in d) (Dyer, Frieze,*

*& Kannan, 1991; Jerrum & Sinclair, 1996).*(Andrieu 2003)

Eventually in 1949, [Metropolis] published the first public document on Monte Carlo simulation with

Stan Ulam (Metropolis & Ulam, 1949). … Soon after, he proposed the Metropolis algorithm with the Tellers and the Rosenbluths (Metropolis et al., 1953). (Andrieu 2003)

In the introduction to this special issue, we focus on describing algorithms that we feel are the main building blocks in modern MCMC programs. We should emphasize that in order to obtain the best results out of this class of algorithms, it is important that we do not treat them as black boxes, but instead try to incorporate as much domain specific knowledge as possible into their design. MCMC algorithms typically require the design of proposal mechanisms to generate candidate hypotheses. Many existing machine learning algorithms can be adapted to become proposal mechanisms (de Freitas et al., 2001). This is often essential to obtain MCMC algorithms that converge quickly. (Andrieu 2003)

To be more specific, the successful design of most practical Monte Carlo algorithms to sample from a target distribution, say π, in scenarios involving both high dimension and complex patterns of dependence relies on the appropriate choice of proposal distributions. As a rule of thumb, to lead to efficient algorithms, such distributions should both be easy to sample from and capture some of the important characteristics of π, such as its scale or dependence structure. Whereas the design of such efficient proposal distributions is often feasible in small dimensions, this proves to bemuch more difficult in larger scenarios. The classical solution that is exploited by both MCMC and SMC methods, albeit in differing ways, consists of breaking up the original sampling problem into smaller and simpler sampling problems by focusing on some of the subcomponents of π. This results in an easier design of proposal distributions. This relative ease of implementation comes at a price, however, as such local strategies inevitably ignore some of the global features of the target distribution π, resulting in potentially poor performance. The art of designing Monte Carlo algorithms mainly resides in the adoption of an adequate trade-off between simplicity of implementation and the often difficult incorporation of important characteristics of the target distribution. (Andrieu 2010)

A considerable amount of attention is now being devoted to the Metropolis-Hastings (M-H) algorithm, which was developed by Metropolis, Rosenbluth, Rosenbluth, Teller, and Teller (1953) and subsequently generalized by Hastings (1970). This algorithm is extremely versatile and gives rise to the Gibbs sampler as a special case, as pointed out by Gelman (1992). The M-H algorithm has been used extensively in physics, yet despite the paper by Hastings, it was little known to statisticians until recently. Papers by Muller (1993) and Tierney (1994) were instrumental in exposing the value of this algorithm and stimulating interest among statisticians in its use. (Chib 1995)

Properly defined and implemented, MCMC methods enable the user to successively sample values from a convergent Markov chain, the limiting distribution of which is the true joint posterior of the model observables. Important features of MCMC methods that enhance their applicability include their ability to reduce complex multidimensional problems to sequences of much lower-dimensional ones and their relative indifference to the presence or absence of conjugate structures between the likelihood and the prior distribution. (Cowles 1996)

Excellent tutorials on the methodology have recently been provided by Albert (1993) and Casella and George (1992); a more complete and advanced summary was given by Tierney (1995) (Cowles 1996)

Although the MCMC algorithms allow an enormous expansion of the class of candidate models for a given dataset, the also suffer from a well-known and potentially serious drawback: It is often difficult to decide when it is safe to terminate them and conclude their “convergence.” That is, at what point is it reasonable to believe that the samples are truly representative of the underlying stationary distribution of the Markov chain? It is immediately clear that is a more general notion of convergence than is usual for iterative procedures, because what is produced by the algorithm at convergence is not a single number or even a distribution, but rather a *sample* from a distribution. Worse yet, the Markov nature of the algorithm means that the members of this sample will generally be correlated with each other, slowing the algorithm it is attempt to sample from the entire stationary distribution and muddying the determination of appropriate Monte Carlo variances for estimates of model characteristics based on the output. (Cowles 1996)

Markov chain Monte Carlo (MCMC) methods allow exploration of intractable probability distributions by constructing a Markov chain whose stationary distribution equals the desired distribution. A major challenge for practitioners is determining how long to run an MCMC simulation. Many experiments employ a \_xed-time rule to terminate the simulation; that is, the procedure terminates after n iterations, where n is determined heuristically. Indeed, some simulations are so complex that this is the only practical approach, but that is not so for most experiments.

Alternatively, many practitioners use convergence diagnostics to determine if n is sufficiently large (for a review see Cowles and Carlin, 1996). Although practical, these methods are mute about the quality of the resulting estimates (Flegal et al., 2008). Moreover, they can introduce bias directly in to the estimates (Cowles et al., 1999). (Flegal 2013)

Geyers 1992: multiple runs, (witch’s hat counterexample); long run (test stationary); mixing; central limit theorem; estimating variance; burnin related to autocovarience

Markov chain Monte Carlo (MCMC) is a set of methods for drawing samples from a distribution, *pi(\*)*, defined on a measurable space (*X; B*), whose density is only known up to some proportionality

constant. Although the *i*-th sample is dependent on the *(i - 1)*-th, the Ergodic Theorem ensures that

for an appropriately constructed Markov chain with invariant distribution *pi(\*)*, long-run averages are

consistent estimators for expectations under *pi(\*).* As a result, MCMC methods have proven useful

in Bayesian statistical inference, where often, the posterior density pi(xjy) / f(yjx)\_0(x) for some

parameter, x (where f(yjx) denotes the likelihood for data y and \_0(x) the prior density), is only known

up to a constant [7]. (Livingstone 2014)

Law of large numbers, if mean exists and variance is bounded

Central limit theorem is asymptotic distributed as a normal distribution with mean and standard deviation .

Let denote the observations or data, and let denote the parameter or set of parameters by which the data are to be summarised. Bayesian methods combine prior evidence on the parameters contained in the density with the likelihood to produce the entire posterior density of . From the posterior density one may extract any information not simply "the most likely value" of a parameter, as with maximum likelihood (ML) estimators. However, until the advent of Monte Carlo Markov Chain methods it was not straightforward to sample from the posterior density, except in cases where it was analytically de.ned. Monte Carlo Markov Chain (MCMC) methods are iterative sampling methods that allow sampling from :

The rest of this paper is organized as follow: Section 2 Monte Carlo Integration. Section 3 Markov Chains. Section 4 MCMC Metropolis Sampler. Section 5 MCMC Metropolis Hastings. Section 6 Hellinger Convergence.

# Monte Carlo Integration

A common integral form is as follows

Expected value of some function of a random variable

And the marginal likelihood

Is also an integral of this form.

Such integrals often cannot be evaluated analytically. When such solutions are not possible, numerical methods may be applied. But these can become inappropriate due to computational costs at high dimensions. A third approach is to use Monte Carlo approximation. In Monte Carlo approximation, the integration problem is resolved through a procedure where averages are taken of values sampled from a computable probability distribution reflective of the original integral.

Two criteria:

One, the function is positive on the interval (a,b)

Two, the integral of the function is finite.

Then we can define the a probability distribution by normalizing the function

Restate the original integration as

Where samples are drawn independently from

A four step procedure for performing Monte Carlo approximation to the integral :

**Step 1:** Identify

**Step 2:** Identify and from it determine and .

**Step 3:** Draw independent samples from

**Step 4:** Evaluate

## Example: Approximating the integral

Integrate the following integral

Analytic solution using integration by parts where

Leads to

A Monte Carlo approximation follows

**Step 1:** From the integral, identify

**Step 2:** Therefore and therefore .

According to the definition expression for given above we determine to be:

**Step 3:** The expression on the right is the definition for the uniform distribution

**Step 4:** we calculate the Monte Carlo approximation as

SEE Appendix MCMC Monte Carlo Estimate of Integral

## Example: Approximating the expected value of the Beta distribution

The expectation function for x is

Where and is the Beta function.

Again, the four step Monte Carlo approximation follows.

**Step 1:** Identify

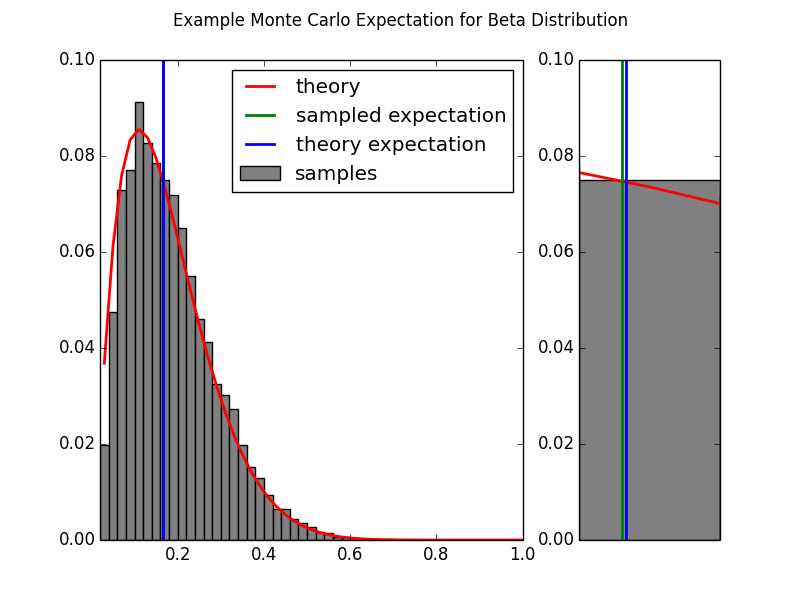
**Step 2:** Therefore = and:

**Step 3** Draw independent samples from the Beta distribution

**Step 4** Approximate the expectation with the expression

For and , the analytical solution is calculuated

SEE Appendix MCMC Monte Carlo Beta Expectation



On the left we see the overall Monte Carlo approximation of the given function. On the right is a zoomed in comparison of the estimated expectation and the calculated expectation. The Monte Carlo approximation is quite close to the analytic solution.

## Monte Carlo Approximation for Optimization

Monte Carlo Approximation can also be used to solve optimization problems of the form:

If fulfills the same criteria described above (namely that it is a scaled version of a probability distribution), then (as above) we can define the probability function

From this, it follows that

For which can be sampled easily, is found at the highest density of the samples. An example follows.

## Example: Monte Carlo Optimization of *g(x) = e^(-(x-4)^2/2)*

Let

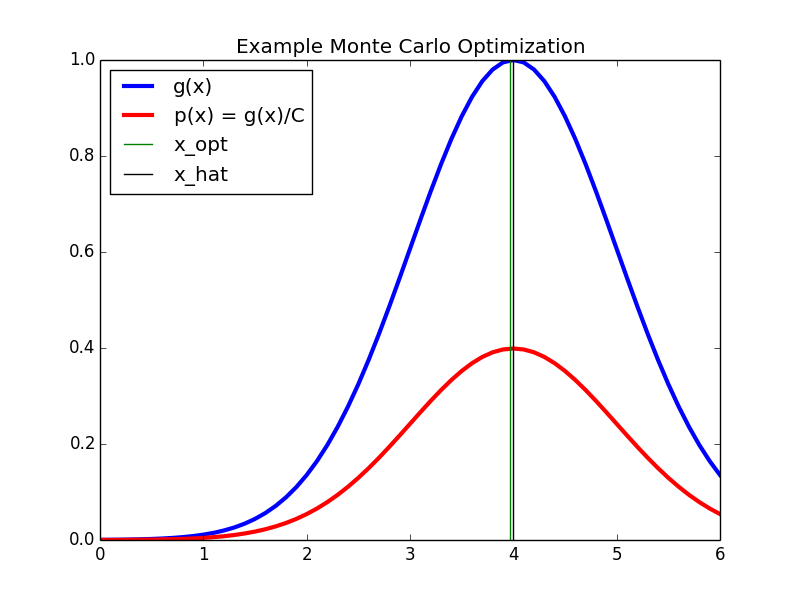
For which is defined as

Note that , is  a scaled version of a Normal distribution with mean equal to 4 and unit variance:

Where is the Normal distribution and .

Thus we can solve for by drawing samples from the normal distribution and determining where those samples have the highest density.

SEE Appendix Monte Carlo Optimization of Exponential Function Code



The Monte Carlo method provides a good approximation (green) to the real solution (black).

## Summary on Monte Carlo Approximation

The example used above were both easy to calculate an exact solution and easy to sample from the probability function . However, there are many practical problems for which an analytic solution does not exist and in which it is not easy to sample from the probability function . One method to attack such problems is to use Markov Chain Monte Carlo methods such as the Metropolis Hasting algorithm. An introduction to Markov Chains follows.

# Markov Chains

A Markov chain is a random process with the Markov property that transitions from one state to another on a state space.

Given a sequence of random variables x(i) such that

a Markov Chain is a stochastic process defined by x(i) such that

Such that the probability of being in state only on the state

A Markov chain is defined by three elements:

1. A ***state space*** , which is a set of values that the chain is allowed to take
2. A ***transition operator*** that defines the probability of moving from state to . The transition operator is often expressed as a ***transition matrix***.
3. An ***initial distribution*** which defines the probability of being in any one of the possible states at the initial iteration .

The Markov property is that the transition to a new state depends only on the prior state without and is independent of all states prior to .

The Markov propery is also commonly called “*memorylessness*.”

Time-homogenous Markov chains are those which converge over time so that

This condition is also called the “*equilibrium distribution”* or the *“stationary distribution.”*

**Finite state-space (time homogenous) Markov chain**

*If the state space is* [*finite*](https://en.wikipedia.org/wiki/Finite_set)*,* the transition probability distribution can be represented by a [matrix](https://en.wikipedia.org/wiki/Matrix_%28mathematics%29), called the transition matrix, with the [element](https://en.wikipedia.org/wiki/Element_%28mathematics%29) of ***P*** equal to

(Hastings, 1970)

## Example: Predicting the weather with a finite state-space Markov chain

Take a city in which there is only 3 types of weather: sunny, foggy, or rainy. In this city, the weather transitions from one state to another in a statistically predictable pattern.

If it is *sunny* today, then

* it is highly likely that it will be *sunny*next week
* it is very unlikely that it will be *raining*next week
* and somewhat likely that it will *foggy*next week

If it is *foggy* today then

* it is somewhat likely that it will be *sunny*next week
* but slightly less likely that it will be *foggy*next week
* and fairly unlikely that it will be *raining*next week.

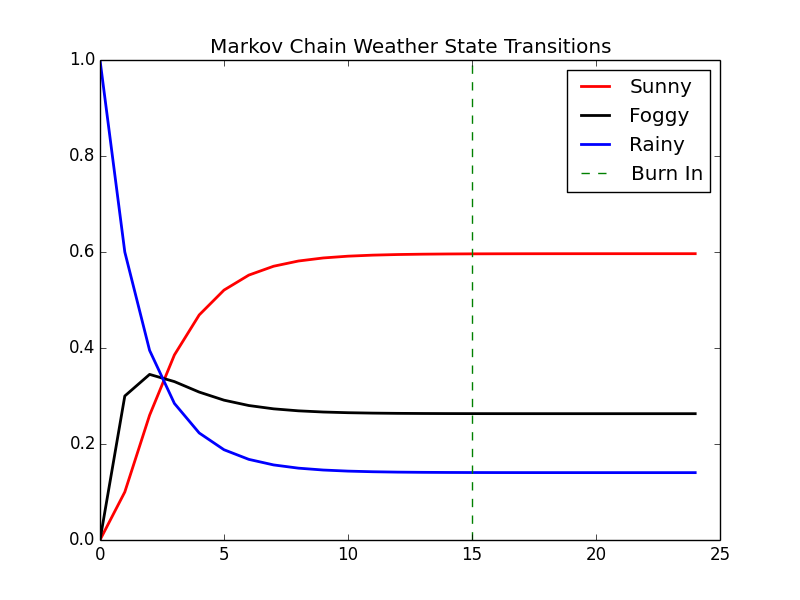
If it is *rainy* today then

* it is unlikely that it will be *sunny*next week
* it is somewhat likely that it will be *foggy*next week
* and it is fairly likely that it will be *rainy*next week

The transition matrix for this set of conditions is

Where each row of P corresponds to the weather at time and each column corresponds to the weather at time .

SEE Appendix Markov Chain Finite State Transitions Code



For time-homogeneous Markov chains, the transition matrix **P** is the same after each step, so the k-step transition probability can be computed as the kth power of the transition matrix, **P**k.

For irreducible and aperiodic Markov chains, there is a unique stationary distribution **π**.

np.dot(X[0,:],matpow(P,2)) = array([ 0.26 , 0.345, 0.395])

and in six months:

np.dot(X[0,:],matpow(P,24)) = array([ 0.59648855, 0.26315895, 0.1403525 ])

We obtain the same results by iterating the Markov Chain from the initial state through the desired number of steps.

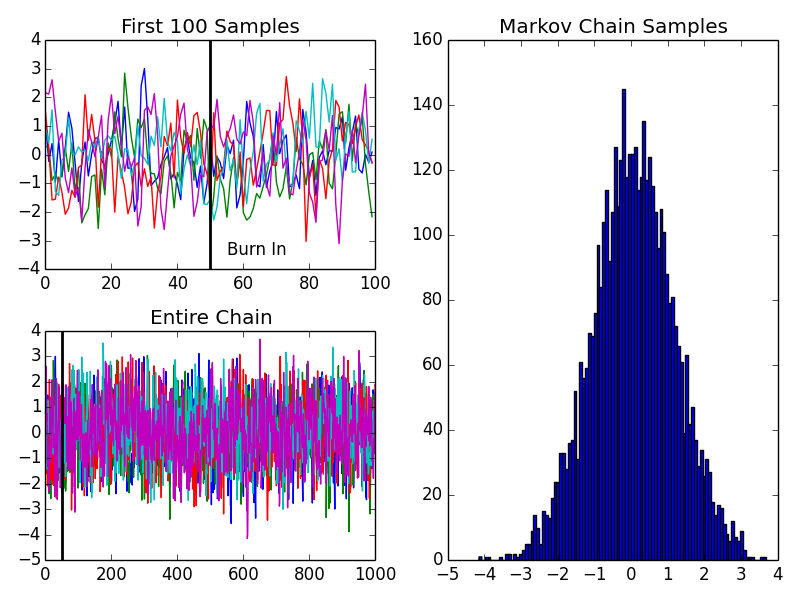
## Continuous state-space Markov chains

A Markov chain can also have a continuous state space that exists in the real numbers . It is a [continuous-time stochastic process](https://en.wikipedia.org/wiki/Continuous-time_stochastic_process) with the [Markov property](https://en.wikipedia.org/wiki/Markov_property).In this case the transition operator cannot be instantiated simply as a matrix, but is instead some continuous function on the real numbers.

## Example: Sampling from a continuous distribution using continuous state-space Markov chains

A continuous state-space Markov chain is presented in this example. The transition operator is a Normal distribution with a mean that is one half the distance between zero and the previous state. The variance is set to one. The initial state is set with a mean of zero and a variance of one. The model is run with a ‘burn in’ of 50 transitions and five chains are run simultaneously.

SEE MCMC Markov Chain Continuous State Transitions Code



The upper left panel shows the first 100 transitions of five different runs of the defined Markov chain. The lower left panel shows the full 1000 transitions for each of the five runs. The ‘burn in’ time is marked on both. The right hand panel shows the stationary distribution derived from the samples of the five runs (minus the “burn in” samples) is a near normal distribution with the mean equal to zero and a variance of 1.3.

## Markov Chain Summary

In the example above, the stationary distribution of the Markov chain is deduced from the samples generated by the chain after a specified burn-in period. Using Markov chains to sample from a specific target distribution, however, requires a transition operator for which the chain converges to a stationary distribution that matches the target distribution. Markov chain samplers such as the Metropolis sample and Metropolis Hastings sample enable us to choose such an operator.

# MCMC: Metropolis Sampling

## Metropolis Sampling

Propose distribution

Acceptance criteria ….

…

1. Set t = 0
2. generate an initial state from a prior distribution over initial states
3. repeat until

set

generate a proposal state from

calculate the acceptance probability

draw a random number from a uniform distribution

if , accept the proposal and set

else  set

## Example: Using the Metropolis algorithm to sample from an unknown distribution

Given a function

From which we would like to draw samples

Define

1. Prior distribution over the initial state of the Markov Chain
2. Proposal distribution

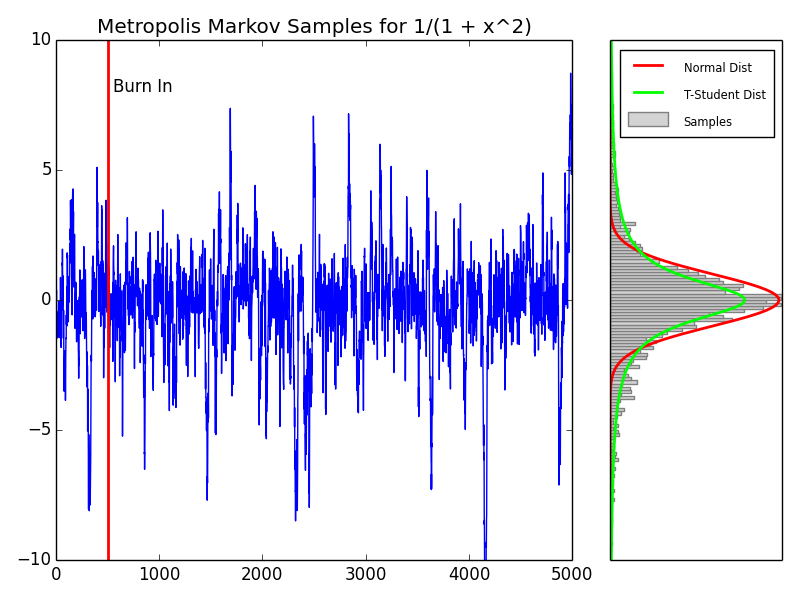
For this example, define



SEE Appendix: MCMC Metropolis Sampler

TODO: Output accept/reject examples

Additional output from the code above showing that the samples from Metropolis sampler draws samples that follow a normalized Student’s-t distribution, even though p(x) is not normalized.



On the left, the progression of the Markov chain’s state evolution is shown. The burn in period is chosen to be 500 transitions. On the right, the distribution of the samples is displayed. A normal distribution is overlain in red. A student-t distribution is overlean in green. It is apparent that the model values follow the student-t, even though the function p(x) was not a formally normalized distribution.

## Reversibility of the transition operator

# MCMC: Metropolis-Hastings Sampling

Unlike in the original Metropolis algorithm, *it does not need to be symmetric*.

Define the transition matrix by

Where is given by

The values si j can be quite general, so long as (i) si j = s j i for all i, j and (ii) αi j ∈

[0, 1]. For any such choice of si j , it is easy to verify that π is the unique stable distribution

for P. For a symmetric Q, a simple choice of si j recovers the originalMetropolis

algorithm.

For a given distribution π, different choices of the si j lead to qualitatively different

Metropolis-like algorithms, all of which produce a Markov chain stable on π. Why

does only the original Metropolis(-Hasting) algorithm live on? The reason was provided

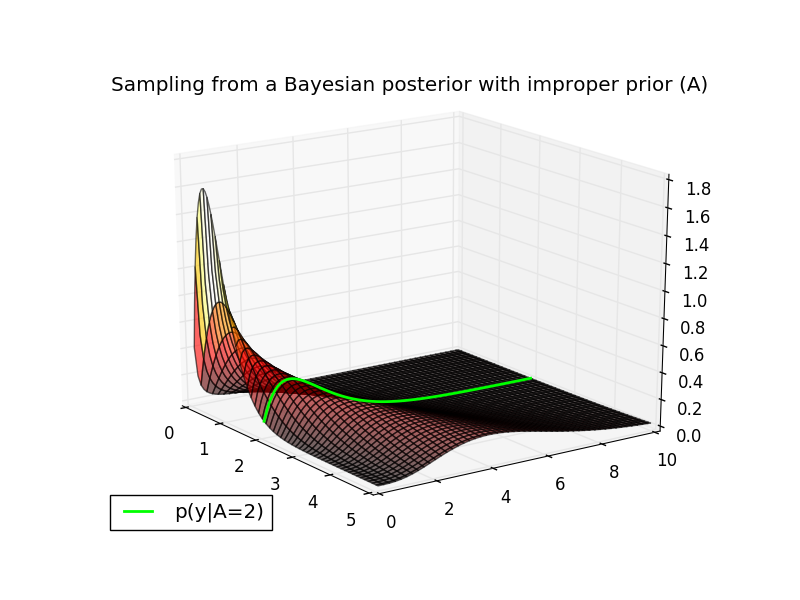
by Hastings’s student, P. H. Peskun. Peskun [52] showed that among all choices

of the si j , the variance of the estimate given in (2) is asymptotically minimal for the

choice that leads to the Metropolis algorithm. Whether by luck or intuition, the first

example of a Markov chain Monte Carlo method proved to be the best. (Richey 2010)

## Example: Sampling from a Bayesian posterior with improper prior



The conditional distribution p(y|A-=2,B=1) is plotted in green along the likelihood surface.

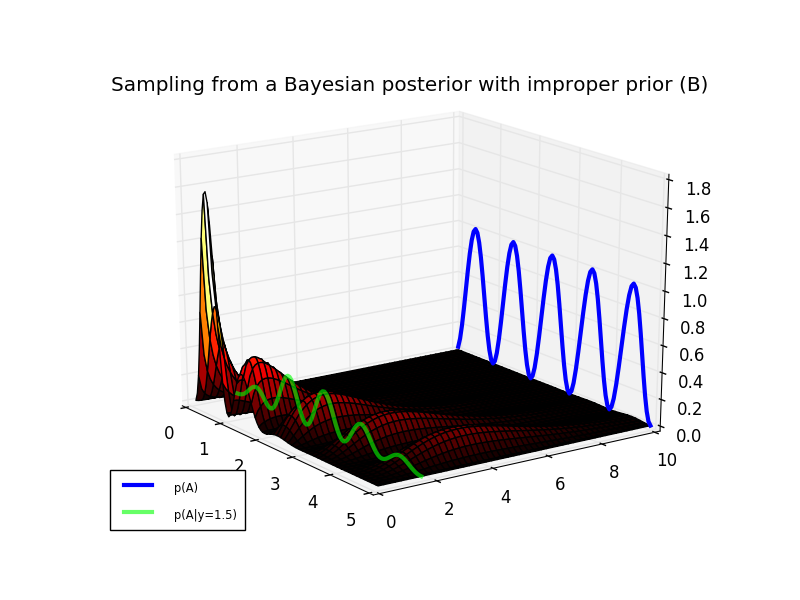
Assume the following priors on the model parameters:

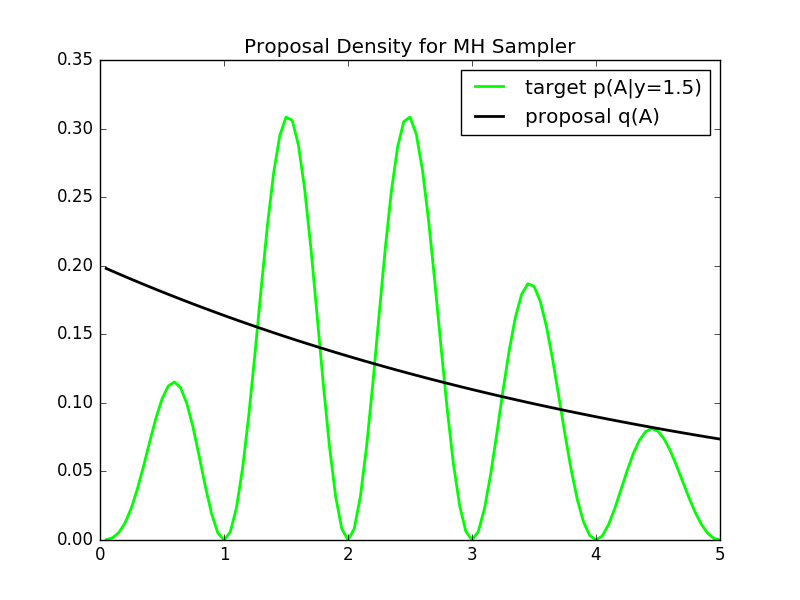
p(A) = sin(pi\*A)^2

and

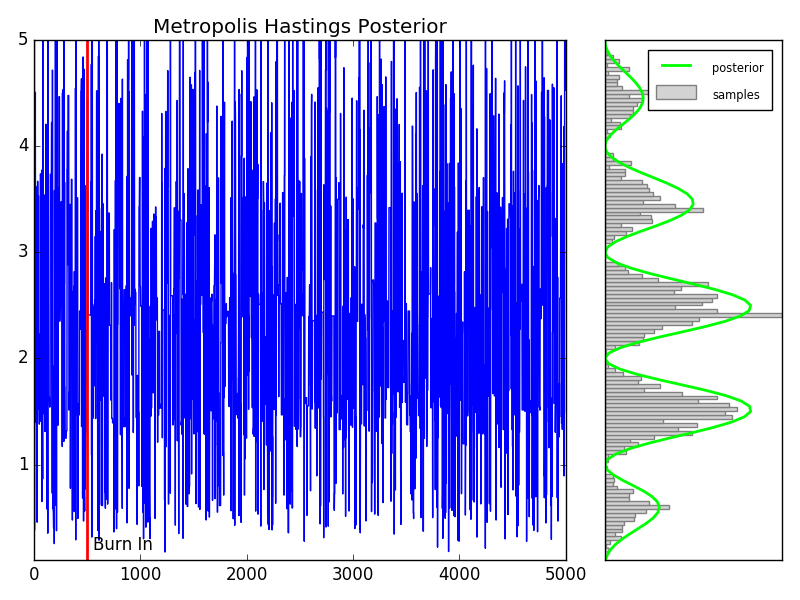
p(B=1) =1

Let’s say that we have a datapoint y = 1.5 and would like to estimate the posterior distribution p(A|y=1.5) using the Metropolis-Hastings algorithm. This particular target distribution is plotted in green below.





The proposal has a fairly good coverage of the posterior distribution. The Markov chain progession and the sample results are shown below.



# Hellinger Convergence

A basic problem of Markov chain theory concerns the rate of convergence in . How long must the chain run to be suitably close to ? It is customary to measure distances between two probabilities by total variation distance:

This yields the math problem: Given *K*,, *x,* and , how large *n* so ?

Sadly, there are very few practical problems where this question can be answered. (Diaconis 2009)

Metrics such as Hellinger distance and negative exponential disparity have a long history in robust estimation in frequentist inference (Hooker,?)

In the i.i.d. context, MHDE estimators are defined by minimizing the Hellinger distance between a postulated parametric density f\_(\_) and a non-parametric estimate gn(\_) over the p-dimensional parameter space \_; that is,

Typically, for continuous data, gn(\_) is taken to be a kernel density estimate; if the probability model is supported on discrete values, the empirical distribution is used. More generally, Lindsay (1994) introduced the concept of a minimum disparity procedure; developing a class of divergence measures that have similar properties to minimum Hellinger distance estimates. These have been further developed in Basu et al. (1997) and Park and Basu (2004). Recently, Hooker and idyashankar (2011a) have extended these methods to a non-linear regression framework. (Hooker ?)

Our analysis and simulations demonstrate that while the use of MCMC signi\_cantly increases computational costs, the additional cost of the use of disparities is on the order of a factor between 2 and 10, remaining implementable for many applications. (Hooker ?)

Convergence of distributions; rates of convergenc; limiting behavior of averages and LLN (Tierney 1994)

If the series can be approximated by a first-order autoregressive process, then the asymptotic standard deviation of the sample mean is

Where again is the posterior standard deviation of and is the autocorrelation of the series . A rough guess for can thus be used to adjust the sample size for dependence in the series. (Tierney 1994)

# Appendix: MCMC Monte Carlo Approximation of Integral Code

# MONTE CARLO APPROXIMATION OF INT(xexp(x))dx

# FOR TWO DIFFERENT SAMPLE SIZES

import numpy as np

np.random.seed(271828)

runs=1000

# THE FIRST APPROXIMATION USING N1 = 100 SAMPLES

N1 = 100;

x = np.random.uniform(size=N1);

I\_hat\_1 = sum(x\*np.exp(x))/N1

I\_hat\_1 # 0.95254390652501486

# A SECOND APPROXIMATION USING N2 = 5000 SAMPLES

N2 = 5000;

x = np.random.uniform(size=N2);

I\_hat\_2 = sum(x\*np.exp(x))/N2

I\_hat\_2 # 1.0209454002831784

# Estimate variance for N1=100

est=[]

for r in range(runs):

x = np.random.uniform(size=N1);

est.append(sum(x\*np.exp(x))/N1)

np.var(est) # 0.0063657238770784439

# Estimate variance for N1=5000

est=[]

for r in range(runs):

x = np.random.uniform(size=N2);

est.append(sum(x\*np.exp(x))/N2)

np.var(est) # 0.00012223069067703404

# variance decreases linearly with number of runs

0.0063657238770784439/0.00012223069067703404

# 52.079586900955817

# Appendix: MCMC Monte Carlo Beta Expectation Code

# MONTE CARLO EXPECTATION

import numpy as np

import scipy.stats as stats

import matplotlib.pyplot as plt

from matplotlib import gridspec

import pylab

np.random.seed(271828)

alpha1 = 2;

alpha2 = 10;

N = 10000;

#x = betarnd(alpha1,alpha2,1,N);

x = stats.beta.rvs(alpha1,alpha2, size=N)

# MONTE CARLO EXPECTATION

expectMC = np.mean(x);

# ANALYTIC EXPRESSION FOR BETA MEAN

expectAnalytic = 1.\*alpha1/(alpha1 + alpha2);

plt.figure(figsize=(8, 6))

plt.hist(x[:,0])

plt.show()

# DISPLAY

steps=0.02

bins = np.arange(0,1.+steps,steps)

h=np.histogram(x,bins)

counts=h[0]

probSampled = 1.\*counts/sum(counts);

probTheory = stats.beta.pdf(bins,alpha1,alpha2);

fig = plt.figure(figsize=(8, 6))

gs = gridspec.GridSpec(1, 2, width\_ratios=[3, 1])

ax0 = plt.subplot(gs[0])

ax0.bar(bins[1:],probSampled,width=steps,color='grey',label="samples")

ax0.plot(bins[1:]+0.5\*steps,probTheory[1:]/sum(probTheory[1:]),'r',linewidth=2,label="theory")

ax0.axvline(x=expectMC,color='g',linewidth=2,label="sampled expectation")

ax0.axvline(x=expectAnalytic,color='b',linewidth=2,label="theory expectation")

ax0.set\_xlim([0.02,1.00])

leg=ax0.legend(loc='upper right')

ltext = leg.get\_texts()

llines = leg.get\_lines()

frame = leg.get\_frame()plt.setp(ltext, fontsize='x-small')

fig.suptitle("Example Monte Carlo Expectation for Beta Distribution")

ax1 = plt.subplot(gs[1])

ax1.bar(bins[1:],probSampled,width=steps,color='grey')

ax1.plot(bins[1:]+0.5\*steps,probTheory[1:]/sum(probTheory[1:]),'r',linewidth=2)

ax1.axvline(x=expectMC,color='g',linewidth=2)

ax1.axvline(x=expectAnalytic,color='b',linewidth=2)

ax1.set\_xlim([0.16,0.18])

ax1.xaxis.set\_major\_locator(pylab.NullLocator())

#ax1.yaxis.set\_major\_locator(pylab.NullLocator())

# plt.show()

plt.savefig('mcmc-monte-carlo-beta-expectation.png')

# Appendix: MCMC Monte Carlo Optimization of Exponential Function Code

# MONTE CARLO OPTIMIZATION OF exp(x-4)^2

import numpy as np

import scipy.stats as stats

import matplotlib.pyplot as plt

from matplotlib import gridspec

import pylab

np.random.seed(271828)

def g(x):

return np.exp(-0.5\*(x-4)\*\*2)

# INITIALZIE

N = 100000

step=0.1

x = np.arange(0,6+step,step)

C = np.sqrt(2\*np.pi)

y = stats.norm.pdf(4,1,x)

# CALCULATE MONTE CARLO APPROXIMATION

#x = normrnd(4,1,1,N);

n = np.random.normal(4,1,size=N)

h=np.histogram(n,100)

counts=h[0]

bins=h[1]

optIdx = np.argmax(counts)

x\_hat = bins[optIdx];

# OPTIMA AND ESTIMATED OPTIMA

# ph = plot(x,g(x)/C,'r','Linewidth',3); hold on

# gh = plot(x,g(x),'b','Linewidth',2); hold on;

# oh = plot([4 4],[0,1],'k');

# hh = plot([xHat,xHat],[0,1],'g');

plt.plot(x,g(x),color='blue',linewidth=3,label="g(x)")

plt.plot(x,g(x)/C,color='red',linewidth=3,label="p(x) = g(x)/C")

plt.axvline(x\_hat, color='green',linewidth=1,label="x\_opt")

plt.axvline(4, color='black',linewidth=1,label="x\_hat")

leg=plt.legend(loc='upper left')

ltext = leg.get\_texts()

llines = leg.get\_lines()

frame = leg.get\_frame()

plt.title("Example Monte Carlo Optimization")

#plt.show()

plt.savefig('mcmc-monte-carlo-optimization-exp.png')

# Appendix: MCMC Markov Chain Finite State Transitions Code

# FINITE STATE-SPACE MARKOV CHAIN EXAMPLE

import numpy as np

import scipy.stats as stats

import matplotlib.pyplot as plt

from matplotlib import gridspec

import pylab

# TRANSITION OPERATOR

# S = Sunny

# F = Foggy

# R = Rainy

# S->S F->S R->S

# S->F F->F R->F

# S->R F->R R->R

P = np.array([[ 0.8 , 0.15, 0.05],

[ 0.4 , 0.5 , 0.1 ],

[ 0.1 , 0.3 , 0.6 ]])

nWeeks = 25

# time series of state vectors

X = np.zeros((nWeeks,3))

#INITIAL STATE IS RAINY

X[0,:] = [0,0,1];

# RUN MARKOV CHAIN

for iB in range(nWeeks-1):

X[iB+1,:] = np.dot(X[iB,:],P)

# DISPLAY

plt.plot(X[:,0],color='r',linewidth=2,label='Sunny')

plt.plot(X[:,1],color='k',linewidth=2,label='Foggy')

plt.plot(X[:,2],color='b',linewidth=2,label='Rainy')

plt.axvline(15, color='g',ls='dashed',label='Burn In')

leg=plt.legend(loc='upper right')

ltext = leg.get\_texts()

llines = leg.get\_lines()

frame = leg.get\_frame()

plt.title("Markov Chain Weather State Transitions")

#plt.show()

plt.savefig('mcmc-markov-chain-finite-state.png')

# Appendix: MCMC Markov Chain Continous State Transitions Code

# EXAMPLE OF CONTINUOUS STATE-SPACE MARKOV CHAIN

import numpy as np

import scipy.stats as stats

import matplotlib.pyplot as plt

from matplotlib import gridspec

import pylab

# INITIALIZE

np.random.seed(271828)

nBurnin = 50; # BURNIN

nChains = 5; # MARKOV CHAINS

# DEFINE TRANSITION OPERATOR

def P(x,n):

return np.random.normal(0.5\*x,1,size=n)

nTransitions = 1000;

x = np.zeros((nTransitions,nChains));

x[0,:] = np.random.normal(1,1,size=nChains);

# RUN THE CHAINS

for iT in range(nTransitions-1):

x[iT+1,:] = P(x[iT],nChains);

plt.close('all')

fig = plt.figure()

ax1 = plt.subplot(221) # Burn In

ax2 = plt.subplot(223) # Full Chains for 5 runs

ax3 = plt.subplot(122) # Histogram of Full without Burnin

ax1.plot(x[0:100,:])

ax1.axvline(50,color='k',linewidth=2,label="Burn In")

ax1.text(50+5,np.floor(np.min(x[:100,:])) + 0.5, r'Burn In')

ax1.set\_title("First 100 Samples")

ax2.plot(x[:,:])

ax2.axvline(50,color='k',linewidth=2,label="Burn In")

ax2.set\_title("Entire Chain")

h=ax3.hist(np.ndarray.flatten(x[100:,:]),bins=100)

ax3.set\_title("Markov Chain Samples")

plt.tight\_layout()

#plt.show()

plt.savefig('mcmc-markov-chain-continuous-state.png')

# Appendix: MCMC Metropolis Markov Sampler Code

# METROPOLIS SAMPLING EXAMPLE

import numpy as np

import scipy.stats as stats

import matplotlib.pyplot as plt

from matplotlib import gridspec

import pylab

np.random.seed(271828)

# DEFINE THE TARGET DISTRIBUTION

def p(x):

return 1./(1.+x\*\*2)

# INITIALIZE CONSTANTS

nSamples = 5000;

burnIn = 500;

nDisplay = 30;

sigma = 1;

minn = -20;

maxx = 20;

step=0.1

xx = np.arange(3.\*minn,3.\*maxx+step,step);

target = p(xx);

pauseDur = .8;

# INITIALZE SAMPLER

x = np.zeros((1,nSamples));

x[0,0]= np.random.normal()

# RUN SAMPLER

for t in range(nSamples-1):

# SAMPLE FROM PROPOSAL

xStar = np.random.normal(x[0,t],sigma);

proposal = stats.norm.pdf(xx,x[0,t],sigma);

# CALCULATE THE ACCEPTANCE PROBABILITY

alpha = min([1., p(xStar)/p(x[0,t])]);

# ACCEPT OR REJECT?

u = np.random.uniform()

if u < alpha:

x[0,t+1] = xStar;

str = 'Accepted';

else:

x[0,t+1] = x[0,t];

str = 'Rejected';

#end

# DISPLAY SAMPLING DYNAMICS

# to do

# DISPLAY RESULTS

# generate some data

a = np.arange(1,nSamples+1,1)

# plot it

fig = plt.figure(figsize=(8, 6))

gs = gridspec.GridSpec(1, 2, width\_ratios=[3, 1])

# DISPLAY MARKOV CHAIN

ax0 = plt.subplot(gs[0])

ax0.set\_ylim([-10,10])

ax0.plot(a, x[0,:])

ax0.axvline(x=burnIn,color='r',linewidth=2)

ax0.text(burnIn+50, 8, r'Burn In')

plt.title('Metropolis Markov Samples for 1/(1 + x^2)')

# DISPLAY SAMPLES

ax1 = plt.subplot(gs[1])

ax1.set\_ylim([-10,10])

ax1.xaxis.set\_major\_locator(pylab.NullLocator())

ax1.yaxis.set\_major\_locator(pylab.NullLocator())

h=ax1.hist(x[0,burnIn:],bins=200,orientation="horizontal",color='lightgrey',edgecolor = 'grey',label="Samples")

b=np.arange(-10,10,0.1)

n=stats.norm.pdf(b)

t=stats.t.pdf(b,1)

plt.plot(n\*nSamples/sum(n),b,color='r',linewidth=2,label="Normal Dist")

plt.plot(t\*nSamples/sum(n),b,color='lime',linewidth=2,label="T-Student Dist")

#plt.ylabel('samples')

leg=ax1.legend(loc='upper left')

ltext = leg.get\_texts()

llines = leg.get\_lines()

frame = leg.get\_frame()

plt.setp(ltext, fontsize='x-small')

plt.tight\_layout()

#plt.show()

plt.savefig('mcmc-metropolis-sampler.png')

# Appendix: MCMC Metropolis Hastings Priors and Posterior Code

# METROPOLIS-HASTINGS BAYESIAN POSTERIOR

import numpy as np

#import scipy.stats as stats

import matplotlib.pyplot as plt

from matplotlib import gridspec

import pylab

from math import gamma

# INITIALIZE

np.random.seed(271828)

# PRIOR OVER SCALE PARAMETERS

B = 1.;

# DEFINE LIKELIHOOD

# likelihood = inline('(B.^A/gamma(A)).\*y.^(A-1).\*exp(-(B.\*y))','y','A','B');

def likelihood(y,A,B):

return (B\*\*A/gamma(A))\*y\*\*(A-1.)\*np.exp(-(B\*y))

# CALCULATE AND VISUALIZE THE LIKELIHOOD SURFACE

# yy = linspace(0,10,100);

# AA = linspace(0.1,5,100);

# avoid infinite edges

yy=np.arange(0.1,10.1,0.1)

AA=np.arange(0.05,5.05,.05)

likeSurf = np.zeros((yy.size,AA.size));

for iA in range(AA.size):

likeSurf[:,iA]=likelihood(yy[:],AA[iA],B)

from mpl\_toolkits.mplot3d import axes3d

from matplotlib import cm

#PLOT

fig = plt.figure()

ax = fig.add\_subplot(111, projection='3d')

axs=ax.get\_axes()

axs.azim=-35.

axs.elev=20.

#axs.dist=

Ys, As = np.meshgrid(yy, AA)

ax.plot\_surface(As.T,Ys.T,likeSurf, rstride=2, cstride=2, alpha=0.6,cmap=cm.hot)

ax.set

# plot A=2

ax.plot(list(As[39]),list(yy), list(likeSurf[:,39]), color='lime', linewidth=2,label='p(y|A=2)')

leg=plt.legend(loc='lower left')

ltext = leg.get\_texts()

llines = leg.get\_lines()

frame = leg.get\_frame()

plt.title("Sampling from a Bayesian posterior with improper prior (A)")

#plt.show()

plt.savefig('mcmc-metropolis-hastings-improper-prior-A.png')

# DEFINE PRIOR OVER SHAPE PARAMETERS

#prior = inline('sin(pi\*A).^2','A');

def prior(A):

return np.sin(np.pi\*A)\*\*2

# DEFINE THE POSTERIOR

# p = inline('(B.^A/gamma(A)).\*y.^(A-1).\*exp(-(B.\*y)).\*sin(pi\*A).^2','y','A','B');

def p(y,A,B):

return (B\*\*A/gamma(A))\*y\*\*(A-1.)\*np.exp(-(B\*y))\*np.sin(np.pi\*A)\*\*2

# CALCULATE AND DISPLAY THE POSTERIOR SURFACE

postSurf = np.zeros(likeSurf.shape);

for iA in range(AA.size):

postSurf[:,iA]=p(yy[:],AA[iA],B)

fig = plt.figure()

ax = fig.add\_subplot(111, projection='3d')

axs=ax.get\_axes()

axs.azim=-35.

axs.elev=20.

Ys, As = np.meshgrid(yy, AA)

ax.plot\_surface(As.T,Ys.T,postSurf, rstride=2, cstride=2, alpha=1.0,cmap=cm.hot)

ax.set

# prior

ax.plot(AA,np.ones((1,AA.size)).T\*10,prior(AA),color="blue",linewidth=3,label="p(A)")

# posterior (not shadowed properly)

ax.plot(As[:,14],np.ones((1,AA.size)).T\*1.6,postSurf[14,:],color="lime",alpha=0.6,linewidth=3,label="p(A|y=1.5)")

leg=plt.legend(loc='lower left')

ltext = leg.get\_texts()

llines = leg.get\_lines()

frame = leg.get\_frame()

plt.setp(ltext, fontsize='x-small')

plt.title("Sampling from a Bayesian posterior with improper prior (B)")

#plt.show()

plt.savefig('mcmc-metropolis-hastings-improper-prior-B.png')

# INITIALIZE THE METROPOLIS-HASTINGS SAMPLER

# DEFINE PROPOSAL DENSITY

# q = inline('exppdf(x,mu)','x','mu');

def exppdf(x,a=1.0):

return 1./a \* np.exp(-x/a)

def q(x,mu):

return exppdf(x,mu)

# MEAN FOR PROPOSAL DENSITY

mu = 5.;

fig = plt.figure()

plt.plot(AA,postSurf[14,:],color='lime',linewidth=2,label='target p(A|y=1.5)')

plt.plot(AA,q(AA,mu),color='black',linewidth=2,label='proposal q(A)')

leg=plt.legend(loc='upper right')

ltext = leg.get\_texts()

llines = leg.get\_lines()

frame = leg.get\_frame()

plt.title("Proposal Density for MH Sampler")

#plt.show()

plt.savefig('mcmc-metropolis-hastings-proposal-density.png')

# DISPLAY TARGET AND PROPOSAL

# SOME CONSTANTS

nSamples = 5000;

burnIn = 500;

minn = 0.1; maxx = 5.;

# INTIIALZE SAMPLER

x = np.zeros((1 ,nSamples));

x[0,0] = mu;

t = 0;

y=1.5

# RUN METROPOLIS-HASTINGS SAMPLER

for t in range(nSamples-1):

# SAMPLE FROM PROPOSAL

xStar = np.random.exponential(mu);

# CORRECTION FACTOR

c = q(x[0,t],mu)/q(xStar,mu);

# CALCULATE THE (CORRECTED) ACCEPTANCE RATIO

alpha = np.min([1., p(y,xStar,B)/p(y,x[0,t],B)\*c]);

# ACCEPT OR REJECT?

u = np.random.rand();

if u < alpha:

x[0,t+1] = xStar;

else:

x[0,t+1] = x[0,t];

# xStar = np.random.exponential(mu); c = q(x[0,t],mu)/q(xStar,mu); p(y,xStar,B)/p(y,x[0,t],B)\*c;

# DISPLAY RESULTS

# x-axis steps (t)

step=1

a = np.arange(1,nSamples+step,step)

# plot it

fig = plt.figure(figsize=(8, 6))

gs = gridspec.GridSpec(1, 2, width\_ratios=[3, 1])

# DISPLAY MARKOV CHAIN

ax0 = plt.subplot(gs[0])

ax0.set\_ylim([minn,maxx])

ax0.plot(a, x[0,:])

ax0.axvline(x=burnIn,color='r',linewidth=2)

ax0.text(burnIn+50, 0.2, r'Burn In')

plt.title('Metropolis Hastings Posterior')

# DISPLAY SAMPLES

ax1 = plt.subplot(gs[1])

ax1.set\_ylim([minn,maxx])

ax1.xaxis.set\_major\_locator(pylab.NullLocator())

ax1.yaxis.set\_major\_locator(pylab.NullLocator())

h=ax1.hist(x[0,burnIn:],bins=200,orientation="horizontal",color='lightgrey',edgecolor = 'grey',label="samples")

#b=np.arange(-10,10,0.1)

#n=stats.norm.pdf(b)

#t=stats.t.pdf(b,1)

#plt.plot(n\*nSamples/sum(n),b,color='r',linewidth=2,label="Normal Dist")

#plt.plot(t\*nSamples/sum(n),b,color='lime',linewidth=2,label="T-Student Dist")

plt.plot(postSurf[14,:]\*(nSamples-burnIn)/sum(postSurf[14,:]),AA, color='lime', linewidth=2,label="posterior");

#plt.ylabel('samples')

leg=ax1.legend(loc='upper right')

ltext = leg.get\_texts()

llines = leg.get\_lines()

frame = leg.get\_frame()

plt.setp(ltext, fontsize='x-small')

plt.tight\_layout()

#plt.show()

plt.savefig('mcmc-metropolis-hastings-posterior.png')