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

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

Table of Contents

[1 Monte Carlo Integration 2](#_Toc444253262)

[1.1 Example: Approximating the integral 3](#_Toc444253263)

[1.2 Example: Approximating the expected value of the Beta distribution 4](#_Toc444253264)

[1.3 Monte Carlo Approximation for Optimization 5](#_Toc444253265)

[1.4 Example: Monte Carlo Optimization of *g(x) = e^(-(x-4)^2/2)* 6](#_Toc444253266)

[1.5 Summary on Monte Carlo Approximation 7](#_Toc444253267)

[2 Markov Chains 7](#_Toc444253268)

[2.1 Example: Predicting the weather with a finite state-space Markov chain 8](#_Toc444253269)

[2.2 Continuous state-space Markov chains 10](#_Toc444253270)

[2.3 Example: Sampling from a continuous distribution using continuous state-space Markov chains 10](#_Toc444253271)

[2.4 Markov Chain Summary 10](#_Toc444253272)

[3 MCMC: The Metropolis Sampler 11](#_Toc444253273)

[3.1 Metropolis Sampling 11](#_Toc444253274)

[3.2 Example: Using the Metropolis algorithm to sample from an unknown distribution 11](#_Toc444253275)

[3.3 Reversibility of the transition operator 13](#_Toc444253276)

[4 MCMC: The Metropolis-Hastings Sampler 13](#_Toc444253277)

[4.1 Example: Sampling from a Bayesian posterior with improper prior 14](#_Toc444253278)

[4.2 Wrapping Up 19](#_Toc444253279)

[5 Hellinger distance 19](#_Toc444253280)

[5.1 Properties 19](#_Toc444253281)

[5.2 Examples 20](#_Toc444253282)

[1 Appendix: MCMC Monte Carlo Approximation of Integral Code 21](#_Toc444253283)

[2 Appendix: MCMC Monte Carlo Beta Expectation Code 22](#_Toc444253284)

[3 Appendix: MCMC Monte Carlo Optimization of Exponential Function Code 23](#_Toc444253285)

[4 Appendix: MCMC Markov Chain Finite State Transitions Code 24](#_Toc444253286)

[5 Appendix: MCMC Markov Chain Continous State Transitions Code 25](#_Toc444253287)

[6 Appendix: MCMC Metropolis Markov Sampler Code 26](#_Toc444253288)

[7 Appendix: MCMC Metropolis Hastings Priors and Posterior Code 28](#_Toc444253289)

# Monte Carlo Integration

<https://theclevermachine.wordpress.com/2012/09/22/monte-carlo-approximations/>

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

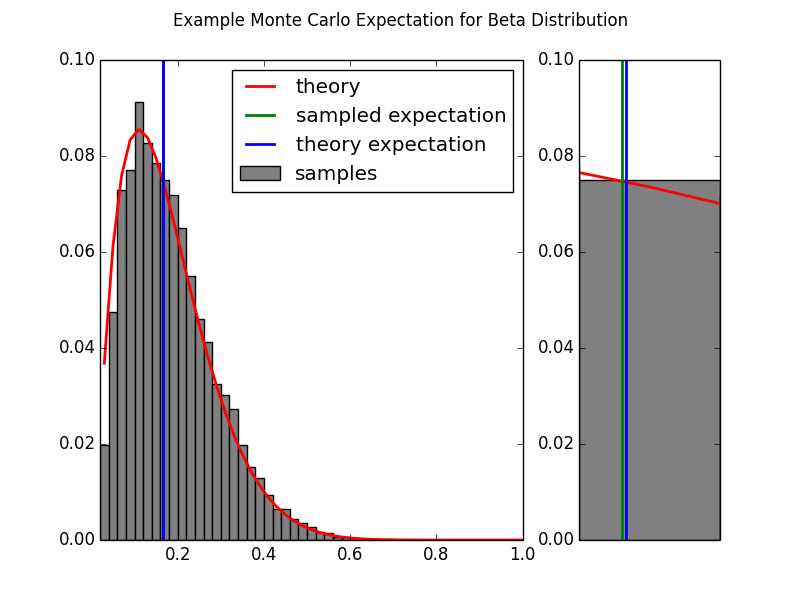
**Step 2:** Therefore = and:

**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

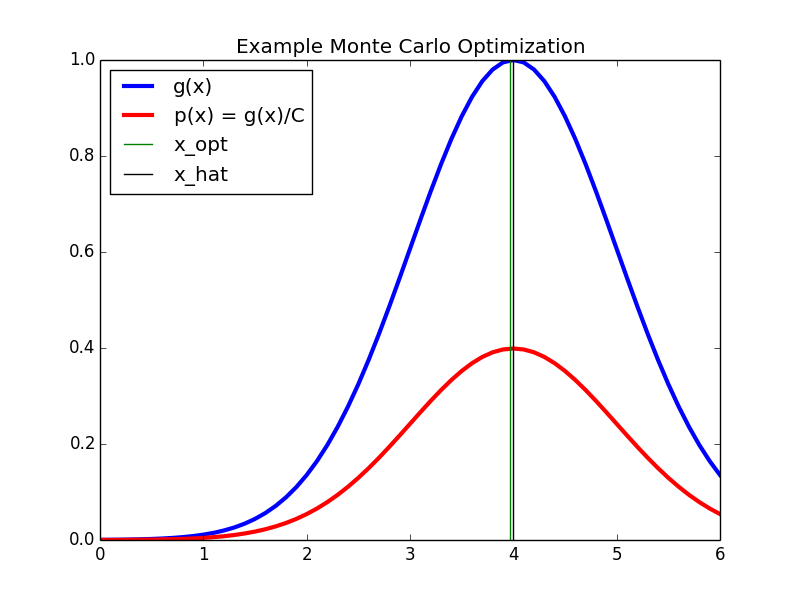
For which is defined as

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

<https://theclevermachine.wordpress.com/2012/09/24/a-brief-introduction-to-markov-chains/>

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

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 ***transisiton 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

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

…

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.

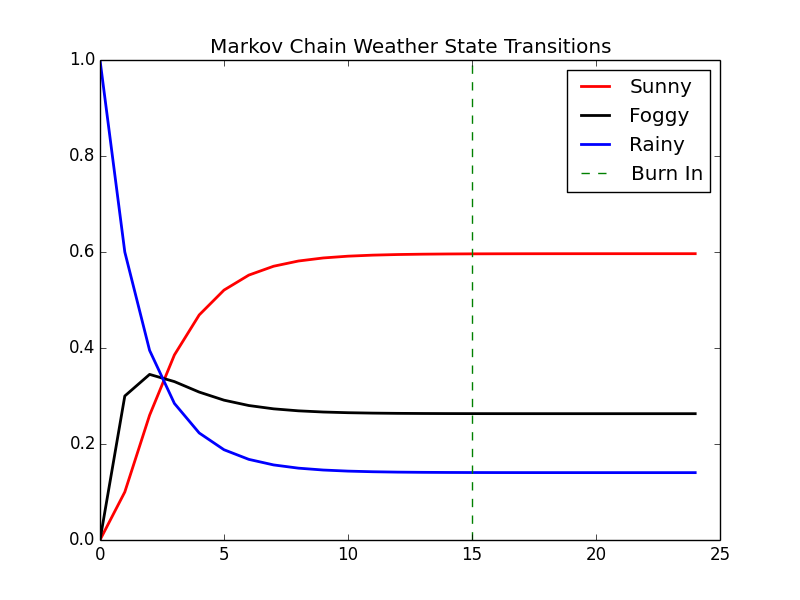
If it is *rainy* today then

* 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.

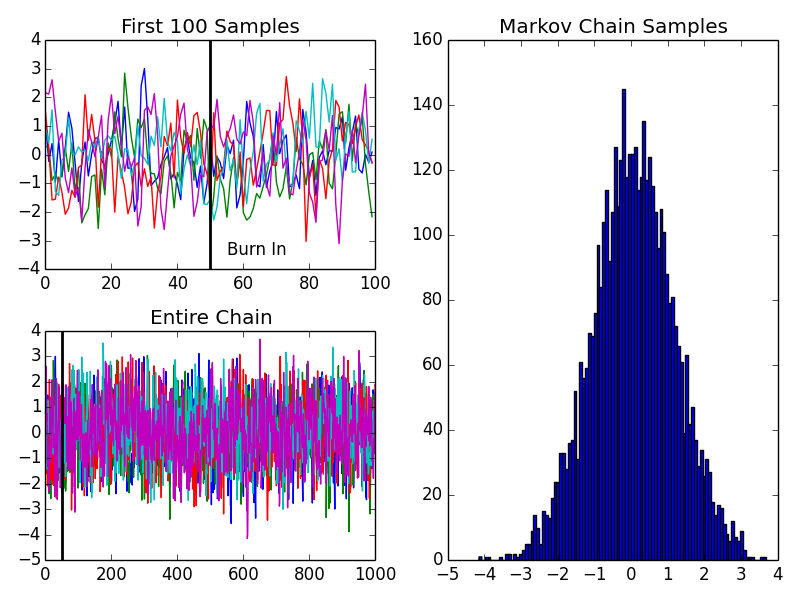
## Continuous state-space Markov chains

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: The Metropolis Sampler

<https://theclevermachine.wordpress.com/2012/10/05/mcmc-the-metropolis-sampler/>

…

## Metropolis Sampling

… heuristics:

1. If  , the proposed state is kept as a sample and is set as the next state in the chain (i.e. move the chain’s state to a location  where has equal or greater density).
2. If –indicating that has low density near –then the proposed state may still be accepted, but only randomly, and with a probability .

…. 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

…

p(x) = (1 + x^2)^{-1}

…

\pi^{(0)} \sim \mathcal N(0,1)

q(x | x^{(t-1)}) \sim \mathcal N(x^{(t-1)},1),

…

SEE Appendix: MCMC Metropolis Sampler

TODO: Output accept/reject examples

In the figure above, we visualize the first 50 iterations of the Metropolis sampler…

…

p^*(x) = \frac{p(x)}{Z}

…

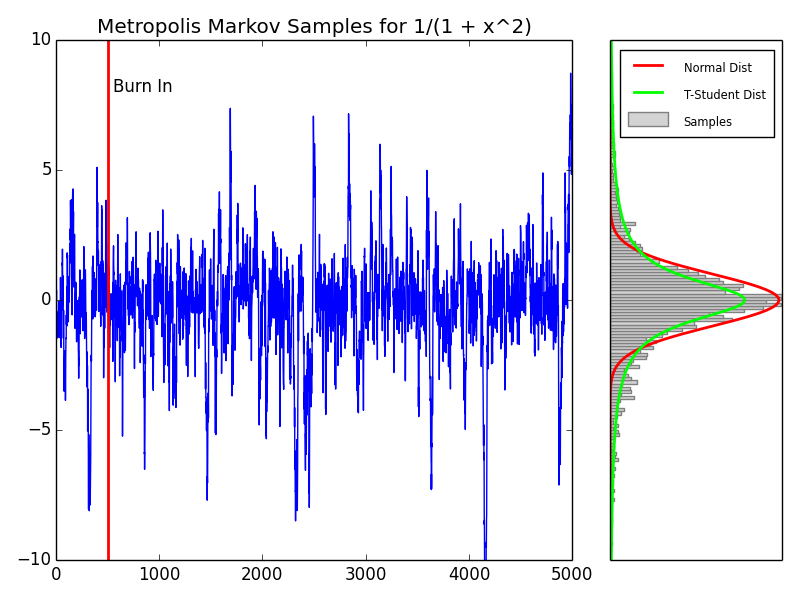
p(x) = Zp^*(x)

…

\frac{p(a)}{p(b)} = \frac{Zp^*(a)}{Zp^*(b)} = \frac{p^*(a)}{p^*(b)}

…

…



…

## Reversibility of the transition operator

…

# MCMC: The Metropolis-Hastings Sampler

<https://theclevermachine.wordpress.com/2012/10/20/mcmc-the-metropolis-hastings-sampler/>

*Note: Metropolis requires Pij = Pji (symmetric) but not the case for MH*

In an earlier [post](https://theclevermachine.wordpress.com/2012/10/05/mcmc-the-metropolis-sampler/) we discussed how the Metropolis sampling algorithm can draw samples from a complex and/or unnormalized target probability distributions using a Markov chain. The Metropolis algorithm first proposes a possible new state x^*in the Markov chain, based on a previous state x^{(t-1)}, according to the proposal distribution q(x^* | x^{(t-1)}). The algorithm accepts or rejects the proposed state based on the density of the the target distribution p(x)evaluated at x^*. (If any of this Markov-speak is gibberish to the reader, please refer to the previous posts on [Markov Chains](https://theclevermachine.wordpress.com/2012/09/24/a-brief-introduction-to-markov-chains/), MCMC, and the [Metropolis Algorithm](https://theclevermachine.wordpress.com/2012/10/05/mcmc-the-metropolis-sampler/) for some clarification).

One constraint of the Metropolis sampler is that the proposal distribution q(x^* | x^{(t-1)})must be symmetric. The constraint originates from using a Markov Chain to draw samples: a necessary condition for drawing from a Markov chain’s stationary distribution is that at any given point in time t, the probability of moving from x^{(t-1)} \rightarrow x^{(t)}must be equal to the probability of moving from x^{(t-1)} \rightarrow x^{(t)}, a condition known as ***reversibility*** or ***detailed balance***. However, a symmetric proposal distribution may be ill-fit for many problems, like when we want to sample from distributions that are bounded on semi infinite intervals (e.g. [0, \infty)).

In order to be able to use an asymmetric proposal distributions, the Metropolis-Hastings algorithm implements an additional correction factor c, defined from the proposal distribution as

c = \frac{q(x^{(t-1)} | x^*) }{q(x^* | x^{(t-1)})}

The correction factor adjusts the transition operator to ensure that the probability of moving from x^{(t-1)} \rightarrow x^{(t)}is equal to the probability of moving from x^{(t-1)} \rightarrow x^{(t)}, no matter the proposal distribution.

The Metropolis-Hastings algorithm is implemented with essentially the same procedure as the Metropolis sampler, except that the correction factor is used in the evaluation of acceptance probability \alpha.  Specifically, to draw Msamples using the Metropolis-Hastings sampler:

1. set t = 0
2. generate an initial state x^{(0)} \sim \pi^{(0)}
3. repeat until t = M

set t = t+1

generate a proposal state x^*from q(x | x^{(t-1)})

calculate the proposal correction factor c = \frac{q(x^{(t-1)} | x^*) }{q(x^*|x^{(t-1)})}

calculate the acceptance probability \alpha = \text{min} \left (1,\frac{p(x^*)}{p(x^{(t-1)})} \times c\right ) 

draw a random number ufrom \text{Unif}(0,1)

if u \leq \alphaaccept the proposal state x^*and set x^{(t)}=x^*

else set x^{(t)} = x^{(t-1)}

Many consider the Metropolis-Hastings algorithm to be a generalization of the Metropolis algorithm. This is because when the proposal distribution is symmetric, the correction factor is equal to one, giving the transition operator for the Metropolis sampler.

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

For a number of applications, including regression and density estimation, it is usually necessary to determine a set of parameters \thetato an assumed model p(y | \theta)such that the model can best account for some observed data y. The model function p(y | \theta)is often referred to as the likelihood function. In Bayesian methods there is often an explicit prior distribution p(\theta)that is placed on the model parameters and controls the values that the parameters can take.

The parameters are determined based on the posterior distribution p(\theta | y), which is a probability distribution over the possible parameters based on the observed data. The posterior can be determined using Bayes’ theorem:

p(\theta | y) = \frac{p(y | \theta) p(\theta)}{p(y)}

where, p(y)is a normalization constant that is often quite difficult to determine explicitly, as it involves computing sums over every possible value that the parameters and ycan take.

Let’s say that we assume the following model (likelihood function):

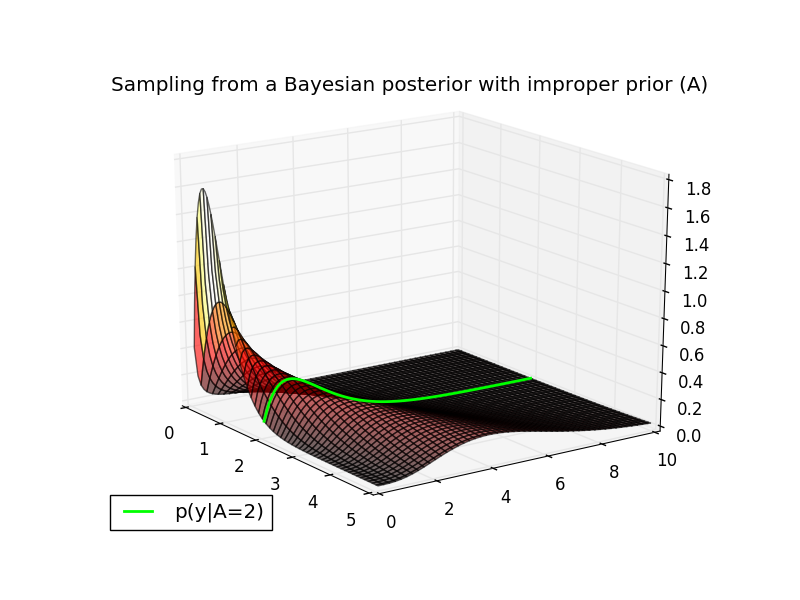
p(y | \theta) = \text{Gamma}(y;A,B), where

\text{Gamma}(y;A,B) = \frac{B^A}{\Gamma(A)} y^{A-1}e^{-By}, where

\Gamma( )is the [gamma function](http://en.wikipedia.org/wiki/Gamma_function). Thus, the model parameters are

\theta = [A,B]

The parameter Acontrols the shape of the distribution, and Bcontrols the scale. The likelihood surface for B = 1, and a number of values of Aranging from zero to five are shown below.



The conditional distribution p(y | A=2, B = 1)is plotted in green along the likelihood surface. You can verify this is a valid conditional in MATLAB with the following command:

|  |  |
| --- | --- |
| 1 | plot(0:.1:10,gampdf(0:.1:10,4,1)); % GAMMA(4,1) |

Now, let’s assume the following priors on the model parameters:

p(B = 1) = 1

and

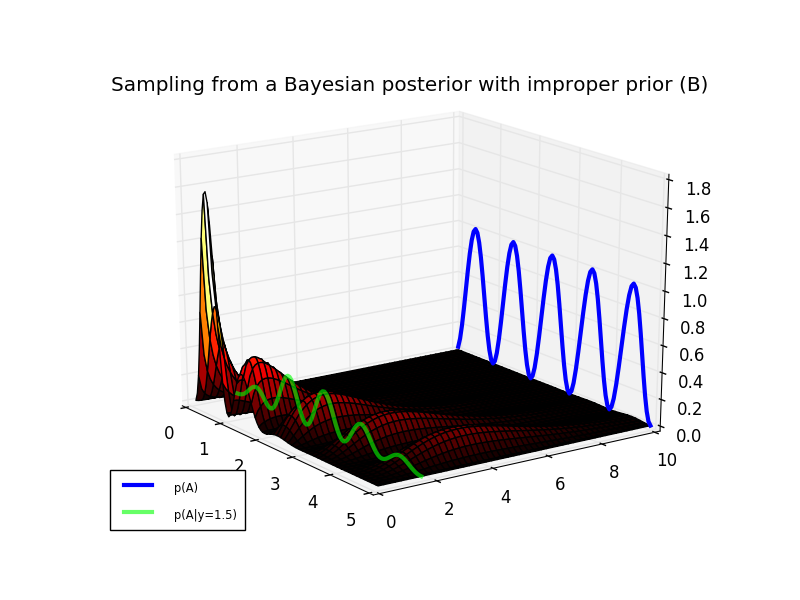
p(A) = \text{sin}(\pi A)^2

The first prior states that Bonly takes a single value (i.e. 1), therefore we can treat it as a constant. The second (rather non-conventional) prior states that the probability of Avaries as a sinusoidal function. (Note that both of these prior distributions are called ***improper priors*** because they do not integrate to one). Because Bis constant, we only need to estimate the value of A.

It turns out that even though the normalization constant p(y)may be difficult to compute, we can sample from p(A | y)without knowing p(x)using the Metropolis-Hastings algorithm. In particular, we can ignore the normalization constant p(x)and sample from the unnormalized posterior:

p(A | y) \propto p(y |A) p(A)

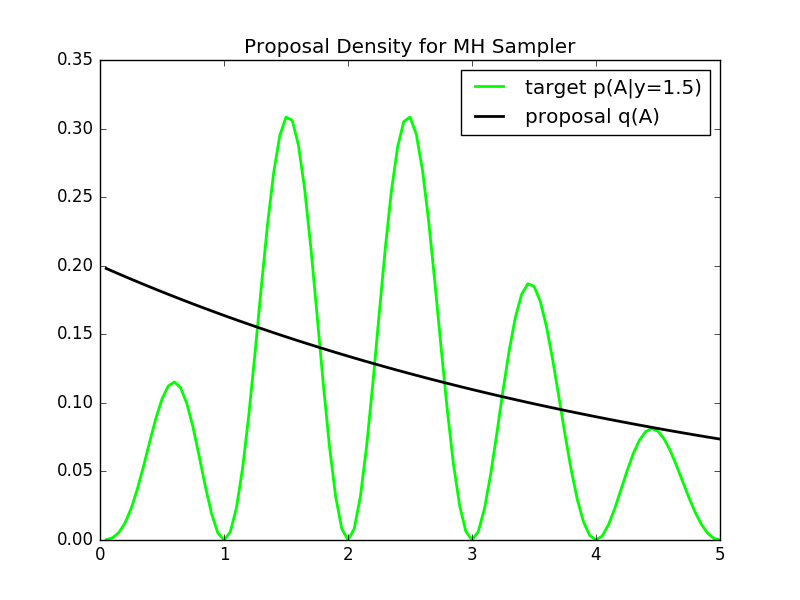
The surface of the (unnormalized) posterior for yranging from zero to ten are shown below. The prior p(A)is displayed in blue on the right of the plot. Let’s say that we have a datapoint y = 1.5and would like to estimate the posterior distribution p(A|y=1.5)using the Metropolis-Hastings algorithm. This particular target distribution is plotted in magenta in the plot below.



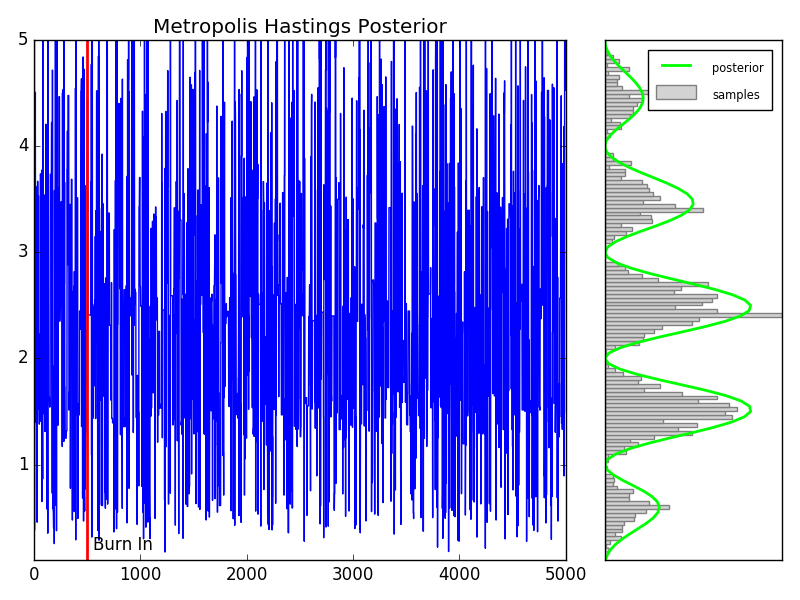
Using a symmetric proposal distribution like the Normal distribution is not efficient for sampling from p(A|y=1.5)due to the fact that the posterior only has support on the real positive numbers A \in [0 ,\infty). An asymmetric proposal distribution with the same support, would provide a better coverage of the posterior. One distribution that operates on the positive real numbers is the exponential distribution.

q(A) = \text{Exp}(\mu) = \mu e^{-\mu A},

This distribution is parameterized by a single variable \muthat controls the scale and location of the distribution probability mass. The target posterior and a proposal distribution (for \mu = 5) are shown in the plot below.



We see that the proposal has a fairly good coverage of the posterior distribution. We run the Metropolis-Hastings sampler in the block of MATLAB code at the bottom. The Markov chain path and the resulting samples are shown in plot below.



As an aside, note that the proposal distribution for this sampler does not depend on past samples, but only on the parameter \mu(see line 88 in the MATLAB code below). Each proposal states x^*is drawn independently of the previous state. Therefore this is an example of an **independence sampler**, a specific type of Metropolis-Hastings sampling algorithm. Independence samplers are notorious for being either very good or very poor sampling routines. The quality of the routine depends on the choice of the proposal distribution, and its coverage of the target distribution. Identifying such a proposal distribution is often difficult in practice.

The MATLAB  code for running the Metropolis-Hastings sampler is below. Use the copy icon in the upper right of the code block to copy it to your clipboard. Paste in a MATLAB terminal to output the figures above.

## Wrapping Up

Here we explored how the Metorpolis-Hastings sampling algorithm can be used to generalize the Metropolis algorithm in order to sample from complex (an unnormalized) probability distributions using asymmetric proposal distributions. One shortcoming of the Metropolis-Hastings algorithm is that not all of the proposed samples are accepted, wasting valuable computational resources. This becomes even more of an issue for sampling distributions in higher dimensions. This is where Gibbs sampling comes in. We’ll see in a later post that Gibbs sampling can be used to keep all proposal states in the Markov chain by taking advantage of conditional probabilities.

# Hellinger distance

## Properties

The Hellinger distance forms a [bounded](https://en.wikipedia.org/wiki/Bounded_function) [metric](https://en.wikipedia.org/wiki/Metric_%28mathematics%29) on the [space](https://en.wikipedia.org/wiki/Function_space) of probability distributions over a given [probability space](https://en.wikipedia.org/wiki/Probability_space).

The maximum distance 1 is achieved when *P* assigns probability zero to every set to which *Q* assigns a positive probability, and vice versa.

Sometimes the factor 1/2 in front of the integral is omitted, in which case the Hellinger distance ranges from zero to the square root of two.

The Hellinger distance is related to the [Bhattacharyya coefficient](https://en.wikipedia.org/wiki/Bhattacharyya_distance) BC(P,Q)as it can be defined as

H(P,Q) = \sqrt{1 - BC(P,Q)}.

Hellinger distances are used in the theory of [sequential](https://en.wikipedia.org/wiki/Sequential_analysis) and [asymptotic statistics](https://en.wikipedia.org/wiki/Asymptotic_statistics).[[4]](https://en.wikipedia.org/wiki/Hellinger_distance#cite_note-4)[[5]](https://en.wikipedia.org/wiki/Hellinger_distance#cite_note-5)

## Examples

The squared Hellinger distance between two [normal distributions](https://en.wikipedia.org/wiki/Normal_distribution) \scriptstyle P\,\sim\,\mathcal{N}(\mu_1,\sigma_1^2)and \scriptstyle Q\,\sim\,\mathcal{N}(\mu_2,\sigma_2^2)is:


  H^2(P, Q) = 1 - \sqrt{\frac{2\sigma_1\sigma_2}{\sigma_1^2+\sigma_2^2}} \,  e^{-\frac{1}{4}\frac{(\mu_1-\mu_2)^2}{\sigma_1^2+\sigma_2^2}}.
  

The squared Hellinger distance between two [exponential distributions](https://en.wikipedia.org/wiki/Exponential_distribution) \scriptstyle P\,\sim \,\rm{Exp}(\alpha)and \scriptstyle Q\,\sim\,\rm{Exp}(\beta)is:


  H^2(P, Q) = 1 - \frac{2 \sqrt{\alpha \beta}}{\alpha + \beta}.
  

The squared Hellinger distance between two [Weibull distributions](https://en.wikipedia.org/wiki/Weibull_distribution) \scriptstyle P\,\sim \,\rm{W}(k,\alpha)and \scriptstyle Q\,\sim\,\rm{W}(k,\beta)(where  k is a common shape parameter and  \alpha\, , \beta are the scale parameters respectively):


  H^2(P, Q) = 1 - \frac{2 (\alpha \beta)^{k/2}}{\alpha^k + \beta^k}.
  

The squared Hellinger distance between two [Poisson distributions](https://en.wikipedia.org/wiki/Poisson_distribution) with rate parameters \alphaand \beta, so that \scriptstyle P\,\sim \,\rm{Poisson}(\alpha)and \scriptstyle Q\,\sim\,\rm{Poisson}(\beta), is:


  H^2(P,Q) = 1-e^{-\frac{1}{2}(\sqrt{\alpha} - \sqrt{\beta})^2}.
  

The squared Hellinger distance between two [Beta distributions](https://en.wikipedia.org/wiki/Beta_distribution) \scriptstyle P\,\sim\,\text{Beta}(a_1,b_1)and \scriptstyle Q\,\sim\,\text{Beta}(a_2, b_2)is:


H^{2}(P,Q) =1-\frac{B\left(\frac{a_{1}+a_{2}}{2},\frac{b_{1}+b_{2}}{2}\right)}{\sqrt{B(a_{1},b_{1})B(a_{2},b_{2})}}
  

where Bis the [Beta function](https://en.wikipedia.org/wiki/Beta_function).

# 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')