MCMC Project

This mini project is designed to get you learning the ropes of the python programming language, as well as teach you the basic principles behind the analysis tool, Markov Chain Monte Carlo sampling, you will be using this summer.

The below tasks are intentionally going to require a lot of searching and reading online. Because we will largely be advising you remotely, it will be critical for you to talk with teammates and search online for solutions to minor problems you encounter. Of course We will help you as much as we can, but we won't be able to help you every step of the way, and you will likely find it faster to solve many problems without our help.

1. Work through this basic introduction to python: `http://learnpythonthehardway.org/book/`. Python is a high-level scripting language that is an excellent first programming language to learn, if you don't know one already.

2. Learn how to import packages, and read introductions to `numpy` and `matplotlib`. The `numpy` package for python includes many useful numerical functions, and `matplotlib` is a plotting package that makes nice figures. `numpy` has many functions for helping to draw random numbers from various distributions (e.g. uniform, Gaussian, etc.). Use `numpy` to generate 10,000 numbers randomly drawn from a uniform distribution between 0 and 1. Use `matplotlib` to make a histogram of these numbers (i.e. samples), similar to Figure 1.

3. Use `numpy` to draw 10000 samples from a one-dimensional Gaussian (i.e. normal) distribution, with a mean of 0 and standard deviation of 1. Make a histogram of these numbers with 50 bins, like Figure 2.

Now it's time to learn a bit about the tool you will be using: the Markov Chain Monte Carlo (MCMC) sampler. When you plot a function on your graphing calculator, say a parabola, you give the calculator a function, e.g. $f(x) = x^2$, and tell it to plot that function across a range, say $x = -1$ to $x = +1$. The calculator then evaluates that function at say 10 different x values between -1 and $+1$, plots those points, and connects them with lines to produce the parabola you seen on the screen. Now say you have a function that depends on 15 different parameters, instead of the single x parameter in this example. If you try to use the same method for plotting the function in 15-dimensional space, it would require you to evaluate the function across a grid of 10^15 different points in order to test 10 different values of each parameter. For large-dimension functions, we need a more efficient method.

For the summer project you will be figuring out what the 15-dimensional probability distribution function (PDF) looks like for the parameters describing the merger of two black holes or neutron stars. The PDF itself is too expensive to compute 10^15 times. It is also very highly peaked in a very tiny region of the parameter space. If you were to set up this grid of 10 points in each parameter, the entire peak would likely be missed in between your grid points; the resolution of your grid would be too coarse. The goal of

MCMC sampling is to smartly pick points in parameter space to evaluate the function at, tracing out the peak(s) of the PDF without prior knowledge of where they are. By using the Metropolis-Hastings algorithm, an "MCMC chain" is able to randomly wander the parameter space and print out points in parameter space with a density equal to the PDF we're trying to determine.

The Markov Chain Monte Carlo algorithm is actually fairly simple:

Algorithm 1 MCMC

$i = 0$
Choose random starting point for $x[0]$, compute $p(x[0])$
while $i < N$ **do**
    Draw random guess y
    Calculate Hasting's ratio $H = p(y)/p(x[i])$
    Draw α from $U[0, 1]$
    if $H \geq \alpha$ **then**
        $x[i + 1] = y$
    else
        $x[i + 1] = x[i]$
    end if
    $i = i + 1$
end while

But all of the magic is in the implementation.

MCMC Assignment 1

(1) Write a simple MCMC routine (in python) to produce $N = 10000$ draws $\{x_i\}$ from the Gaussian distribution

$$p(x|\sigma,\mu) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

Note that the parameters σ, μ are fixed here, and they define the properties of the distribution. As a concrete example, set $\mu = 1, \sigma = 1$. For the proposal distribution, use

$$y = x + N[0, \alpha^2]$$

where $N[0, \alpha^2]$ is a draw from a gaussian distribution with mean $\mu = 0$ and variance $\sigma^2 = \alpha^2$. You can use `numpy.random.normal()` to draw random numbers. Google knows all about it.

(a) Run your code trying four different jump size scalings, $\alpha = 0.01$, $\alpha = 0.1$, $\alpha = 1.0$ and $\alpha = 10.0$. Visually inspect the chains. Which one appears to be exploring the distribution most effectively?

(b) Have your code compute the acceptance fraction for the proposed jumps (*i.e.* the fraction of the proposed jumps that are accepted). Does a high acceptance rate necessarily mean efficient exploration of the posterior distribution?

(c) Using the same 4 sets of jumps sizes, run your code for $N = 1000$ iterations and use the output to produce histograms of the $\{x_i\}$. Plot them against the target distribution $\pi(x|\sigma,\mu)$. Which distribution looks the best?

MCMC Assignment 2

(1) Use the best mixing version of the code from Assignment 1 to produce a large number of independent samples from the Gaussian with $\mu = 1, \sigma = 1$. The goal here is going to be trying to infer the parameters μ, σ that produce the data, and to look at model selection. (a) In this case, the samples $\{x_i\}$ are the data, d, and the model parameters are μ, σ. We wish to produce posterior distribution functions for these parameters. The likelihood is given by

$$p(d|\mu, \sigma) = \prod_{i=1}^{N} \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x_i-\mu)^2}{2\sigma^2}}$$

Experiment with different proposal distributions (Uniform draws between $[-1, 1]$, Gaussian draws for each parameter, anything else you can think of.) Once again, keep track of the acceptance rates and the auto-correlation length. Produce marginalized posterior distribution functions for the parameters. Also look at 2-d scatter plots of μ versus σ to get a sense of the parameter correlations. Try running with $N = 100$ and $N = 1000$ data samples.

MCMC Assignment 3

(1) This assignment deals with multi-modal posterior distributions where we will learn about *p*arallel tempering.

The target distribution that we wish to sample from has the form

$$p(\mu,\nu) = \frac{16}{3\pi}\left(e^{-\mu^2-(9+4\mu^2+8\nu)^2} + \frac{1}{2}e^{-8\mu^2-8(\nu-2)^2}\right)$$

The challenge here is that the target distribution is multi-modal, and in addition, one of the modes is "banana shaped."

Parallel Tempering

Exploring multi-modal posteriors can be a serious challenge for many MCMC algorithms, which tend to get stuck exploring a single mode of the posterior. While the Metropolis-Hastings transition probability allows for "downhill steps", it is highly unlikely that a standard MCMC chain will "walk" all the way down from one mode, cross the valley and climb up a second mode. There are a host of techniques that have been developed to tackle this type of problem. One that works fairly well is parallel tempering, which is a form of population MCMC. The idea is that while exploring a multi-modal posterior $p(\vec{x}|d) = p(d|\vec{x})p(\vec{x})$ may be challenging when the likelihood $p(d|\vec{x})$ has multiple peaks, it is much easier to explore the modified posterior $p_T(\vec{x}|d) = p(d|\vec{x})^{1/T}p(\vec{x})$ in the limit of large "temperature" T. The idea is to run multiple chains in parallel, each with a different temperature T, and by allowing exchanges between the chains we effectively end up using the high temperature chains as proposal densities for the low temperature chains. The high temperature chains freely explore the entire space and locate all the modes, while the low temperature chains map out the peaks in greater detail. Only samples from the $T = 1$ chain correspond to samples from the target distribution. Samples from the other chains are discarded. If properly implemented, a PTMCMC algorithm with M parallel chains run for N iterations each will converge to the target distribution far more quickly than a single chain run for $M \times N$ iterations.

A good choice of temperature spacing for most applications is given by the progression $T_{i+1} = T_i^c$ with $c \sim 1.2 \to 2$. Once the maximum temperature and temperature spacing have been chosen the total number of chains M is determined, and we are ready to proceed. The PTMCMC algorithm works by independently evolving each of the M chains using some standard MCMC algorithm to explore the $= p_{T_i}(\vec{x}|d)$, and then once every so often "swaps" are proposed with an inter-chain transition probability

$$H = \frac{p_{T_i}(\vec{x}_{i+1}|d)p_{T_{i+1}}(\vec{x}_i|d)}{p_{T_i}(\vec{x}_i|d)p_{T_{i+1}}(\vec{x}_{i+1}|d)} \tag{1}$$

If the swap is accepted, the parameters of the chain at temperature T_i get swapped with those at temperature T_{i+1} (not the entire past history, just the current location). The regular MCMC moves at each temperature then proceed for a while until another inter-chain

swap is proposed. Selecting the right ratio of within temperature proposals to chain swap proposals is more art than science. Typically a 3:1 ratio works pretty well.

(b) For the second part of this assignment, compare the performance of a single chain MCMC sampler with a multi-chain PTMCMC sampler. Start the chains out at the locations $(\mu, \nu) = (0, 0)$, $(\mu, \nu) = (1, 2)$ and $(\mu, \nu) = (0, -2)$ and produce plots showing the path taken by the chain(s) for the first 50 iterations for each sampler. For the PTMCMC sampler show the paths for both the $T = 1$ and the $T_{\max}$ chains. What did you choose for $T_{\max}$? Run the PTMCMC sampler for a large number of iterations. Produce 2-d histograms of the recovered posterior distributions for the $T = 1$ and the $T_{\max}$ chains. Play around a little with the choice of the geometric scaling parameter c and the maximum temperature $T_{\max}$. Produce a plot of the inter-chain swap acceptance fraction as a function of c with c between 1 and 2 in increments of 0.1.