MCMC Lab

This is an INDIVIDUAL LAB, no working with others. This part is the full MCMC lab, it should work for the cases you ran for the previous lab too so I have included those descriptions here as well. As discussed in class, write a program to implement MCMC. Your code should use Metropolis Hastings (or just Metropolis)to handle continuous nodes. Note that you do not need to detect local conjugacy; just sample with Metropolis at each node. You should be able to handle the following node types.

- Normal
- Gamma
- Inverse Gamma
- Poisson
- Beta
- Bernoulli or Binomial

Please demonstrate your code on the networks given below AND on one you make up yourself. Share the network you make up by posting it (and a query) on the discussion page BUT DO NOT POST THE ANSWER. Run your code on a couple of networks that have be posted by others.

Because of numerical issues, we strongly recommend doing calculations on the log scale.

Please submit a report on your code that is complete enough that I can see what you have done without having to run your code, although you need to submit the code too. Include your answers from the tests below, and from the discussion page. Include some mixing plots and, of course, prior and posterior plots for each example (the point isn't to see how many trees you can kill) but again, you must demonstrate that your code worked and got the right distributions.

Example Networks

Faculty Evaluations

In an example in class, we modeled faculty evaluation scores as distributed normally with a normal prior over the mean parameter and an inverse gamma prior over the variance parameter. Therefore there are two parameter nodes and N observed nodes (one for each datum). Andrew's prior belief is represented by a Normal(5, 1/9) prior distribution for mean and an InvGamma(mean=1/4, var=1/144) for variance (alpha=11, beta=2.5). Each piece of data is distributed according to N(mean, var). Since you're implementing Gamma, note that an

inverse gamma random sample is 1 divided by a gamma random sample.

The humble MCMC Hammer code to represent this network and show posterior distributions for the mean and variance parameters is as follows (note that the starting values are tweaked to remove the need for burn):

```
#!/usr/bin/env python

from __future__ import division
from mcmchammer import *
from scipy import stats

datafilename = 'faculty.dat'
nsamples = 10000
burn = 0
mean_candsd = 0.2
var_candsd = 0.15

# Read in Data

data = [float(line) for line in open(datafilename)]
# Use point estimators from the data to come up with
starting values.
estimated_mean = stats.mean(data)
estimated_var = stats.var(data)

# Create Nodes and Links in Network

meannode = NormalNode(estimated_mean, name='Mean', \
        candsd=mean_candsd, mean=5, var=(1/3)**2)

varprior_mean = 1/4
varprior_stddev = 1/12
varprior_shape = MomentsInvGammaShape(varprior_mean,
varprior_stddev**2)
varprior_scale = MomentsInvGammaScale(varprior_mean,
varprior_stddev**2)
varnode = InvGammaNode(estimated_var, name='Variance', \
        candsd=var_candsd, shape=varprior_shape,
scale=varprior_scale)

for datum in data:
```

```
    NormalNode(datum, observed=True, mean=meannode,
var=varnode)

# Perform simulations and plot results

currentnetwork.simulate(nsamples, burn)

meannode.plotmixing()
varnode.plotmixing()
meannode.plotmarginal(min=4, max=6.5)
varnode.plotmarginal(min=0)
```

The faculty evaluation data are as follows (courtesy of the **Stats** department):

```
6.39
6.32
6.25
6.24
6.21
6.18
6.17
6.13
6.00
6.00
5.97
5.82
5.81
5.71
5.55
5.50
5.39
5.37
5.35
5.30
5.27
4.94
4.50
```

http://aml.cs.byu.edu/%7Ekseppi/cs677sp06wiki/images/7/75/Fac-mean.png http://aml.cs.byu.edu/%7Ekseppi/cs677sp06wiki/images/3/39/Fac-var.png

Professional Golfers

The file golfdataR.dat reports the scores at 42 professional golf tournaments involving 604 professional golfers. The Bayesian network for this example contains about 6000 nodes. With such a large network, it is difficult to tune mixing and burn parameters, and it takes a while to run simulations. The model is as follows:

An observation for Golfer i in Tournament j (obsnode[i][j]) is normally distributed with mean golfermean[i]+tournmean[j] and variance obsvar. The node tournmean[j] represents the difficulty (average score) of Tournament j, and the node golfermean[i] represents the skill of the golfer (how much better they score than the average). The prior distribution of the node obsvar is an Inverse Gamma with shape=83 and scale=1/.0014. Each golfermean[i] is Normal with mean 0 and variance hypergolfervar. Hypergolfervar is Inverse Gamma with shape=18 and scale=1/.015. Each tournmean[j] is Normal with mean=hypertournmean and var=hypertournvar. Hypertournmean is Normal with mean=72 and var=2 and hypertournvar is Inverse Gamma with shape=18 and scale=1/.015.

Note that the model and priors were created by a statistician who is very familiar with professional golf.

The nodes of interest are the golfermeans. It is insightful to compare the posterior distributions for different golfers. We took the 5th, 50th, and 95th percentiles of samples in the posterior distribution for each golfer and printed these out in ranked order. In Python, this looked like:

```
ability = []
for golfer in golfermean:
    samples = golfermean[golfer].samples[:]
    samples.sort()
    median = samples[nsamples//2]
    low = samples[int(.05 * nsamples)]
    high = samples[int(.95 * nsamples)]
    ability.append( (golfer, low, median, high) )

ability.sort(lambda x, y: cmp(x[2], y[2]) )
i = 1
for golfer, low, median, high in ability:
    print '%d: %s %f; 90%% interval: (%f, %f)' % (i,
golfer, median, low, high)
    i += 1
```

The final results looked like:

```
1: VijaySingh -3.840661; 90% interval: (-4.354664,
-3.288281)
```

```
2: TigerWoods -3.737515; 90% interval: (-4.568503,
-2.850548)
3: ErnieEls -3.430582; 90% interval: (-4.313688, -2.480963)
4: PhilMickelson -3.353783; 90% interval: (-3.961659,
-2.734419)
5: StewartCink -3.044881; 90% interval: (-3.647329,
-2.476737)
...
600: TommyAaron 4.883014; 90% interval: (2.666637,
7.017471)
601: DerekSanders 5.420959; 90% interval: (3.484806,
7.799434)
602: BobLohr 6.026131; 90% interval: (3.881693, 8.108147)
603: ArnoldPalmer 6.226227; 90% interval: (4.355256,
8.178954)
604: TimTims 7.978016; 90% interval: (5.905149, 10.156050)
```

If you're struggling with this one, here's the code that set up the network in one implementation:

```
hypertournmean = NormalNode(72.8, name='Tournament Hyper
Mean', \
        candsd=hypertournmean_candsd, mean=72, var=2)
hypertournvar = InvGammaNode(3, name='Tournament Hyper
Var', \
        candsd=hypervar_candsd, shape=18, scale=1/.015)
tournmean = {}
for tourn in tourns:
    tournmean[tourn] = NormalNode(est_avg,
name='Tournament %s'%tourn, \
            candsd=mean_candsd, mean=hypertournmean,
var=hypertournvar)
hypergolfervar = InvGammaNode(3.5, name='Golfer Hyper Var',
\
        candsd=hypervar_candsd, shape=18, scale=1/.015)
golfermean = {}
for golfer in golfers:
    golfermean[golfer] = NormalNode(est_avg, name=golfer, \
            candsd=mean_candsd, mean=0, var=hypergolfervar)
obsvar = InvGammaNode(3.1, name='Observation Var', \
        candsd=obsvar_candsd, shape=83, scale=1/.0014)
for (name, score, tourn) in data:
```

```
    NormalNode(score, observed=True, \
            mean=tournmean[tourn]+golfermean[name],
var=obsvar)
```

Wacky Network

A wacky network:

```
A = NormalNode(mean=20, var=1)
E = BetaNode(alpha=1, beta=1)
B = GammaNode(shape=A**pi, invscale=7)
D = BetaNode(alpha=A, beta=E)
C = BernoulliNode(D: True, (1-D): False)
F = PoissonNode(rate=D)
G = NormalNode(mean=E, var=F)
```

Plot the densities for each node with no observations and then again with G = 5.

Notes

- You shouldn't need to use much from any statistics libraries. Really all you need to be able to do is to sample from a normal and from a uniform. Since you want to be in the log space anyway, we recommend that you enter in all of your own log pdfs.
- Try to keep your code somewhat general. You really shouldn't have to have a separate Metropolis implementation for each different type of node.
- For Bernoulli/Binomial, sample directly instead of trying to use Metropolis.
- For any node, but particularly Gamma, remember to reuse the last value if the candidate is outside of support.
- You may either implement a separate Inverse Gamma node, or you can just invert the output of a Gamma before sending the value on to the child node.
- For Poisson, use Metropolis with a candidate distribution that rounds samples from a normal.
-