Hierarchical Bayesian Modeling with Ensemble MCMC
Bayesian Computing for Astronomical Data Analysis 2014

Eric Ford

Preparation

For this lab, we'll use the Julia programming language. We *strongly* encourage you to use the workstation at your seat, as it will use a high-speed wired internet connection, rather than wifi. In this exercise, you'll be making plots with lots of points, so the enhanced bandwidth will be worth the minor inconvenience of using an unfamiliar computer.

First, login to the Windows workstation. From the Windows start button in the lower left, click "All Programs", then "Internet Applications", then "Communications", then "Open Text Exceed onDemand Client", then "Exceed onDemand Client". In the window that pops up, enter host: "hammer.rcc.psu.edu", your userid and password, and click ok. On the next screen leave "Xconfig" set to "Default Xconfig" but change "Xstart" to "Xterm.xs" and click run. A unix command line prompt will appear with a prompt like "[ebf11@hammer21 ~]$".

Next, login to the lionxv.rcc.psu.edu submit node via ssh

```
[user@hammerXX ~] ssh -X userid@lionxv.rcc.psu.edu
```

Copy the file `popmcmc.jl` into your working directory

```
cp /gpfs/home/ebf11/public/popmcmc.jl .
```

Next, adjust your git configuration settings (just once), so you can get around the firewall.

```
[user@lionxv ~] git config --global url."https://".insteadOf git://
```

Next, load the julia module, specifying the recent version

```
[user@lionxv ~] module load julia/060514
```

and start julia

```
[user@lionxv ~] julia
```

We'll want to plot, so we need to install one plotting package (just once)

```
julia> Pkg.add("Winston")
```

You'll wait a while as it installs a plotting package. To exit julia

```
julia> quit()
```

Now, let's get to work. First, we'll submit a request for an interactive job (with one processor on one node with a 2 hour time limit) to be run on a compute node.

```
[user@lionxv ~] qsub -X -I -l nodes=1:ppn=1,walltime=02:00:00 -q
astro-seminar
[user@lionxv ~] module load julia/060514
[user@lionxv ~] julia
```

In the instructions below, everything is inside julia, so we stop showing the `julia>` prompts so you can copy and paste more easily.

```
# Tell julia to load the Winston plotting package
using Winston
```

1. Comparing performance when sampling from a multivariate Gaussian

Copy and paste the code below, reading through the comments, so you understand what it's doing, even if you don't recognize julia's syntax.

```
# Create a function that returns target density to sample from
(initially a multivariable normal)
ndim = 2    # number of model parameters
srand(157)  # seed the random number generator
tmp = rand(ndim,ndim);
target_covar = tmp'*tmp
target_covar_fact = cholfact(target_covar)

function log_target_pdf(theta::Array{Float64,1}, beta::Float64 = 1.0)
  @assert(size(theta,1)== size(target_covar_fact,1) ==
size(target_covar_fact,2) )
  ll = -0.5*( dot(vec(theta),vec(target_covar_fact\theta)) +
2*logdet(target_covar_fact) + size(theta,1)*log(2pi) )
  return beta*ll
end

# Create an initial population of parameter values
popsize = max(16,ifloor(4*ndim))
offset = 0.0
scale = 1.0
pop_init = Array(Float64,ndim,popsize);
for i in 1:popsize
    pop_init[:,i] = offset .+ scale.* (target_covar_fact[:L] *
randn(ndim))
end

# Pause to plot initial population
plot(vec(pop_init[1,:]),vec(pop_init[2,:]),"r.")
xlabel("\\theta_1")
ylabel("\\theta_2")

# Load functions to perform population MCMC from a file
include("popmcmc.jl")

# Setup two possible proposal densities for use with Independent
Random Walk Metropolis Hasting (IRMMH) MCMC.
rwmh_prop_covar =
2.38^2/size(pop_init,1)*estimate_covariance(pop_init)
rwmh_prop_covar_diag = copy(rwmh_prop_covar)
for i in 1:ndim, j in 1:ndim
   if i!=j
```

```
     rwmh_prop_covar_diag[i,j] = 0.0
   end
end
rwmh_prop_covar_diag
```

1a. Assess the sampling efficiency of Independent Random Walk Metropolis-Hasting MCMC with a diagonal Proposal Density

Run IRMMH MCMC simulations with a diagonal proposal density and inspect plots showing the evolution of the model parameters in the first few chains, using the code below.

```
results_rwmh = run_indep_rwmh_mcmc(pop_init,  log_target_pdf,
rwmh_prop_covar_diag, num_gen= 500)
plot_trace(results_rwmh,1)

plot_trace(results_rwmh,2)
```

Questions: Is this MCMC efficiently sampling from the posterior density?

Try changing the proposal density (but keeping it diagonal) and note how the sampling efficiency is affected. For example, you might try:

```
rwmh_prop_covar_try = 10.*rwmh_prop_covar_diag
results_rwmh = run_indep_rwmh_mcmc(pop_init,  log_target_pdf,
rwmh_prop_covar_try, num_gen= 500);
plot_trace(results_rwmh,1)
plot_trace(results_rwmh,2)
# and
rwmh_prop_covar_try = 0.01*rwmh_prop_covar_diag
# followed by same code from above to run and plot
```

Using the intuition you've gained, try to improve the sampling by adjusting the proposal density (but keeping it diagonal). After a few tries, it's time to move on.

Questions: Did you improve the sampling significantly? Would you have been able to do that if you didn't know so much about your target posterior?

1b. Assess the sampling efficiency of Independent Random Walk Metropolis-Hasting MCMC with a non-diagonal Proposal Density

Next, run IRWMH MCMC simulations with a non-diagonal proposal density and inspect the results visually.

```
results_rwmh = run_indep_rwmh_mcmc(pop_init,  log_target_pdf,
rwmh_prop_covar, num_gen= 500);
plot_trace(results_rwmh,1)
plot_trace(results_rwmh,2)
```

Questions: How does the efficiency of sampling from the posterior density compare to when using a diagonal proposal matrix?

Optionally, try changing the proposal density and see if you can improve the sampling. For example, you might try

```
rwmh_prop_covar_try = 2.0*rwmh_prop_covar
results_rwmh = run_indep_rwmh_mcmc(pop_init,  log_target_pdf,
rwmh_prop_covar_try, num_gen= 500);
plot_trace(results_rwmh,1)
plot_trace(results_rwmh,2)
```

After a few tries, it's time to move on.

Questions: Were you able improve the sampling significantly? Would you have been able to do that if you didn't know so much about your target posterior? Why would it be harder to choose an efficient proposal density in a high-dimensional parameter space?

1c. Assess the sampling efficiency of Differential Evolution MCMC

Run a Differential Evolution MCMC (DE-MCMC) simulation and visually inspect the evolution of the model parameters.

```
results_demcmc = run_demcmc( pop_init,  log_target_pdf, num_gen= 500);
plot_trace(results_demcmc,1)
plot_trace(results_demcmc,2)
```

Questions: Is the DEMCMC algorithm sampling more or less efficiently than the IRWMH-MCMC? How easy/hard was it to get efficient sampling in high dimensions compared to IRMMH-MCMC? How would increasing the dimensionality of the parameter space affect the efficiency of DE-MCMC?

1d. Assess the sampling efficiency of an Affine Invariant MCMC

Run an Affine Invariant Population MCMC (AIP-MCMC) simulation and visually inspect the evolution of the model parameters.

```
results_affine = run_affine_pop_mcmc(pop_init,  log_target_pdf,
num_gen= 500); results_affine["theta_last"]
plot_trace(results_affine,1)
plot_trace(results_affine,2)
```

Questions: Is the AIP-MCMC algorithm sampling more or less efficiently than the IRWMH-MCMC? How easy/hard is it to get efficient sampling in high dimensions compared to IRMMH-MCMC? How does AIP-MCMC compare to DE-MCMC (for this target density and initial population)?

2. Assess Burn-in Time Required with Population MCMC Algorithm

For a real problem, we wouldn't be able to start with a population drawn from the target density. Therefore, we'd need to let the Markov chain run "burn-in" for enough iterations that the initial conditions no longer have a significant effect on the samples being generated. In this exercise, you are to compare the three sampling algorithms again, but choosing an initial population of parameter values that is offset from the mode of the target density and/or changed in size or shape. For example, you might try an initial population like:

```
popsize = max(16,ifloor(4*ndim))
offset = [10.0, 10.0]
scale = 0.01
pop_init_poor = Array(Float64,ndim,popsize);
for i in 1:popsize
    pop_init_poor[:,i] = offset .+ scale*randn(ndim)
end
```

Then rerun the three algorithms, each starting from that initial population:

```
rwmh_prop_covar =
2.38^2/size(pop_init,1)*estimate_covariance(pop_init_poor)
results_rwmh = run_indep_rwmh_mcmc(pop_init_poor,  log_target_pdf,
rwmh_prop_covar, num_gen= 500);
results_demcmc = run_demcmc( pop_init_poor,  log_target_pdf, num_gen=
500);
results_affine = run_affine_pop_mcmc(pop_init_poor,  log_target_pdf,
num_gen= 500);
```

Plot and inspect the evolution of the model parameters for a few chains from each simulation.

```
plot_trace(results_rwmh,1)
plot_trace(results_demcmc,1)
plot_trace(results_affine,1)
```

Plot the resulting samples from each simulation, using the `plot_sample` function. Overplot, samples taken just from the end of the chain. For example,

```
plot_sample(results_rwmh,style="r.")
plot_sample(results_rwmh,style="b.", overplot=true, gen_start=101,
gen_stop = 300)
plot_sample(results_rwmh,style="g.", overplot=true, gen_start=301)
```

Questions: How long did it takes for each of the different algorithms to "burn-in" enough, that the results are no longer sensitive to the choice of the initial population? What are the implications for the length of MCMC simulations needed for a real world problem?

3. Sampling from a Multi-modal Target Density

For real problems, there can be multiple significant modes in the posterior probability density. Even if there aren't, a scientists may not know that there isn't a second mode. In this exercise, we'll try sampling from a multi-modal target density. Replace the log_target_pdf function with a version designed for a multi-modal target density. For example,

```
tmp = reshape([0.1, 0.0, 0.0, 1.0], (2,2))
target2_covar = tmp'*tmp
target2_covar_fact = cholfact(target2_covar)

function log_target_pdf(theta::Array{Float64,1}, beta::Float64 = 1.0 )
  @assert(size(theta,1) == ndim)
  # This example calculates the density of a mixture of two Gaussian
distributions
  # with means loc1 and loc2 and a common covariance matrix.
  # The fraction of the density from each mixture component is given
by frac1 and frac2=1-frac1
  loc1 = [1.0, 0.0 ]
  loc2 = [-3.0, 3.0 ]
  frac1 = 1.0/2.0
  logfrac1 = log(frac1)
  logfrac2 = log(1.0 - frac1)
  delta = theta.-loc1
  # Calculate pdf of each component distribution
  logpdf1 = -0.5*( dot(delta,target_covar_fact\delta) +
2*logdet(target_covar_fact) + ndim*log(2pi) )
  delta = theta.-loc2
  logpdf2 = -0.5*( dot(delta,target2_covar_fact\delta) +
2*logdet(target2_covar_fact) + ndim*log(2pi) )
  # Weight each by the fraction it contributes to the mixture
  logpdf1 += logfrac1
  logpdf2 += logfrac2
  logpdf = log_sum_of_log_quantities(logpdf1,logpdf2)
  logpdf *= beta
  return logpdf
end
```

Our population MCMC algorithms need an initial population of parameter values. This time, initialize the population drawn from a multivariate normal which isn't centered on either of the modes of the target distribution. For example,

```
popsize = 16
offset = 0.0
scale = 1.0
pop_init = Array(Float64,ndim,popsize);
for i in 1:popsize
```

```
    pop_init[:,i] = offset .+ scale.* randn(ndim)
end
```

First, try a IRWMH MCMC simulation. Since we know our initial population isn't likely very good, we'll do two simulations. First, we do one simulation for the chains to "burn-in".

```
results_rwmh = run_indep_rwmh_mcmc(pop_init,  log_target_pdf,
rwmh_prop_covar_try, num_gen= 500);
```

Then, run the simulations some more, picking up where you left off, but otherwise discarding the samples from during the burn-in phase. We're *hoping* that this second set of chains will be usable for inference.

```
results_rwmh = run_indep_rwmh_mcmc(results_rwmh["theta_last"],
log_target_pdf, rwmh_prop_covar_try, num_gen= 500);
```

Similarly, run DE-MCMC simulations.

```
results_demcmc = run_demcmc( pop_init,  log_target_pdf, num_gen= 500);
results_demcmc = run_demcmc( results_demcmc["theta_last"],
log_target_pdf, num_gen= 500);
```

And finally, let's run AIP-MCMC simulations

```
results_affine = run_affine_pop_mcmc( pop_init,  log_target_pdf,
num_gen= 500);
results_affine = run_affine_pop_mcmc( results_affine["theta_last"],
log_target_pdf, num_gen= 500);
```

Plot the results posterior samples from each simulation, using the `plot_sample` function.

```
plot_sample(results_rwmh,style="g.")
plot_sample(results_demcmc,style="r.", overplot=true)
plot_sample(results_affine,style="b.", overplot=true)
```

In this case, we know that the target density has two modes. Inspecting the above plots, identify the slope (a) and intercept (b) for a straight line that would divide the two posterior samples. Set the variable a and b to these values and check that the line accurately separates points from the two modes.

```
a = 2.5
b = 1.0
linex = linspace(-4.0,4.0,11)
liney = a.*linex.+b
oplot(linex,liney,"b-")
```

For each algorithm, inspect the trace plots to look for any indications of non-convergence.

```
plot_trace(results_rwmh,1)
plot_trace(results_rwmh,2)
```

```
plot_trace(results_demcmc,1)
plot_trace(results_demcmc,2)

plot_trace(results_affine,1)
plot_trace(results_affine,2)
```

Questions: Qualitatively compare the posterior samples from the second simulation of each algorithm to the target density. Which algorithms performed well? Which performed poorly?

Using the coefficients a and ab identified above, calculate how many samples came from each mode (for each algorithm) using the function `calc_frac_above_line(x, y, a, b)` (in popmcmc.jl) that takes vectors of x and y coordinates and scalar coefficients a and b and returns the fraction of points for which y >= a*x+b.

```
calc_frac_above_line(vec(results_rwmh["theta_all"][1,:,:]),vec(results
_rwmh["theta_all"][2,:,:]),a,b)
calc_frac_above_line(vec(results_demcmc["theta_all"][1,:,:]),vec(resul
ts_demcmc["theta_all"][2,:,:]),a,b)
calc_frac_above_line(vec(results_affine["theta_all"][1,:,:]),vec(resul
ts_affine["theta_all"][2,:,:]),a,b)
```

You should also inspect what happens to several individual chains. For example,

```
chainid = 1
calc_frac_above_line(vec(results_rwmh["theta_all"][1,chainid,:]),vec(r
esults_rwmh["theta_all"][2,chainid,:]),a,b)
```

Questions: Does this affect your assessment of which algorithms performed well? If you didn't know the target density was multi-modal or the true fraction from each mode, could you have recognized the non-convergent simulations from the samples and/or trace plots?
Based on your results from this and the previous exercises, under what circumstances would you choose or avoid each algorithm?

---

4. Investigating the Computational Cost

The computational cost of a population MCMC simulation depends on both the population size and the number of generations. Perform new simulations to compare the sampling efficiency and burn-in time required as a function of the size of the population.
Remember you'll need to reset the target density. For example, you might return to a simple multivariate normal target density, e.g.,

```
function log_target_pdf(theta::Array{Float64,1}, beta::Float64 = 1.0)
  @assert(size(theta,1) == size(target_covar_fact,1) ==
size(target_covar_fact,2) )
```

```
  ll = -0.5*( dot(vec(theta),vec(target_covar_fact\theta)) +
2*logdet(target_covar_fact) + size(theta,1)*log(2pi) )
  return beta*ll
end

ndim = 32
tmp = rand(ndim,ndim);
target_covar = tmp'*tmp
target_covar_fact = cholfact(target_covar)
```

and create a new initial population of states. E.g.,

```
popsize = ifloor(2*ndim)
offset = randn(ndim)
scale = 0.5
pop_init = Array(Float64,ndim,popsize);
for i in 1:popsize
    pop_init[:,i] = offset .+ scale.* (target_covar_fact[:L] *
randn(ndim))
end
```

Then you can run simulations and plot the results as before.

```
results_demcmc = run_demcmc( pop_init,  log_target_pdf, num_gen=
2000);
# etc
```

When exploring higher dimensional target distributions, you'll want to inspect many trace plots, making sure to check each model parameter and more than just five chains, using options like `plot_trace(results_demcmc,1,chains=[6,7,8,9,10])`.

Remember that MCMC use pseudorandom numbers, so some runs will perform better than others, even with the exact same initial conditions.

Questions: How does the number of generations required for burn-in vary with the number of model parameters (ndim)? After burn in is complete, how does the convergence rate (in terms of either number of generations and/or model evaluations) change as you vary the population size?
Why is it important that the population size be larger than the number of model parameters?
Why might you use a larger popsize than needed?
Based on your previous results, what size population would you recommend for a serial computer (relative to the number of model parameters)? Would it make sense to use a larger size population if you could perform the calculations in parallel?

5 . Apply Population MCMC to a Hierarchical Bayesian Model (Optional, as time permits)
Update the function `log_target_pdf(theta)` to evaluate the posterior probability of a hierarchical model. Create simulated data set and an initial population of model parameters. For example, you could construct a target density equal to the posterior distribution for the hierarchical model from problems 2 & 3 of the Approximate Bayesian Computing lab earlier this afternoon.

```
# Set variables for "known" parameters.
mu_o = 0.0
sigma_o = 10.0
sigma_obs = 1.0 # same as "sigma" in the ABC lab

# Create a simulated data set y
nobs = 64
mu_true = mu_o + sigma_o.*randn()
y = mu_true .+ sigma_obs.*randn(nobs)

# log_posterior for Gaussian Example from ABC Lab questions #2 & #3
function log_target_pdf(theta::Array{Float64,1}, beta::Float64 = 1.0)
  @assert(size(theta,1) == 1 )
  mu = theta[1]
  n = length(y)
  logprior =  -0.5*( ((mu-mu_o)/sigma_o)^2 + 2*log(sigma_o) + log(2pi)
)
  ll = -0.5*( sum( ((y.-mu)./sigma_obs).^2 ) + 2n*log(sigma_obs) +
n*log(2pi) )
  return logprior+beta*ll
end
# Create an initial population for population MCMC
popsize = 16
pop_init = Array(Float64,1,popsize);
for i in 1:popsize
    pop_init[1,i] =  mu_o + 2*sqrt(sigma_o^2+sigma_obs^2) * randn()
end
```

Apply a population MCMC algorithms to analyze your data. E.g.,

```
results_demcmc = run_demcmc( pop_init, log_target_pdf, num_gen= 100);
results_demcmc = run_demcmc( results_demcmc["theta_last"],
log_target_pdf, num_gen= 1000);
plot_trace(results_demcmc,1)
```

Question: Check whether the posterior samples that you get out consistent with the parameters used to generate the data? How do your posterior samples compare to the true posterior? How does the efficiency compare to ABC?

```
## Example code for plotting histograms
```

```
# Plot histogram of posterior samples for mu
nbins = 50
edges = histrange(vec(results_demcmc["theta_all"][1,:,:]),nbins)
h_demcmc = hist(vec(results_demcmc["theta_all"][1,:,:]),edges)[2]
plot(midpoints([edges]),h_demcmc./(sum(h_demcmc)*(edges[2]-
edges[1])),"b*")

# Overplot true posterior for mu
true_pdf_x = edges
true_pdf_mu =
(mu_o/sigma_o^2+sum(y)/sigma_obs^2)/(1/sigma_o^2+length(y)/sigma_obs^2
)
true_pdf_sigma = sqrt(1/(1/sigma_o^2+length(y)/sigma_obs^2))
true_pdf_y = exp(-0.5*((true_pdf_x.-
true_pdf_mu)./true_pdf_sigma).^2)/(sqrt(2pi)*true_pdf_sigma);
oplot(true_pdf_x,true_pdf_y,"k-")
```