

Title: Computational Physics - Lecture 12

Date: Oct 19, 2018 01:00 PM

URL: <http://pirsa.org/18100053>

Abstract:

Markov Chain Monte Carlo

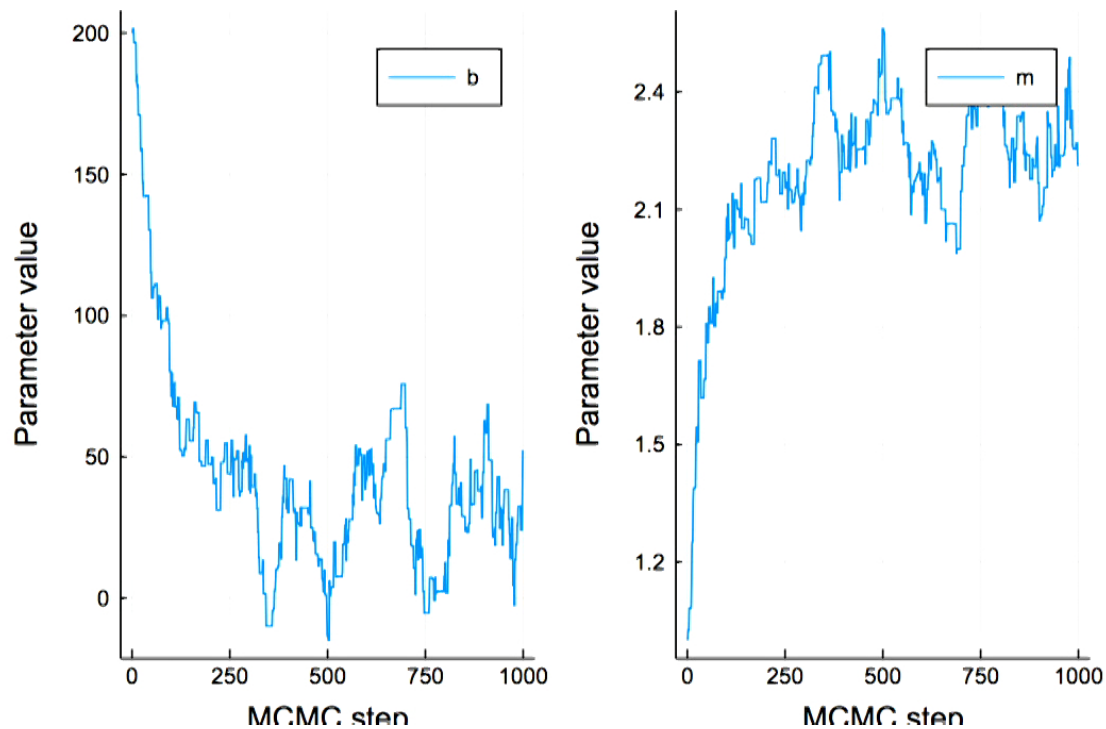
```
In [1]: 1 using Plots
        2 gr()
        3 using LinearAlgebra
        4 using Statistics
```

As before, we're going to use this funky function to read the data

```
In [2]: 1 function get_data()
        2     # \tablehead{ID & $x$ & $y$ & $\sigma_y$ & $\sigma_x$ & \multicolumn{2}{c}{$\rho_{xy}$}}
        3     latex = ""
        4     1 & 201 & 592 & 61 & 9 & -0 & 84\\
        5     2 & 244 & 401 & 25 & 4 & 0 & 31\\
        6     3 & 47 & 583 & 38 & 11 & 0 & 64\\
        7     4 & 287 & 402 & 15 & 7 & -0 & 27\\
        8     5 & 203 & 495 & 21 & 5 & -0 & 33\\
        9     6 & 58 & 173 & 15 & 9 & 0 & 67\\
       10     8 & 202 & 504 & 14 & 4 & -0 & 05\\
       11     9 & 198 & 510 & 30 & 11 & -0 & 84\\
       12     10 & 158 & 416 & 16 & 7 & -0 & 69\\
       13     11 & 165 & 393 & 14 & 5 & 0 & 30\\
       14     12 & 201 & 442 & 25 & 5 & -0 & 46\\
```

```
In [13]: 1 p1 = plot(chain[:,1], xlabel="MCMC step", ylabel="Parameter value", label="b")  
2 p2 = plot(chain[:,2], xlabel="MCMC step", ylabel="Parameter value", label="m")  
3 plot(p1, p2)
```

Out[13]:



Let's start writing the Metropolis-Hastings MCMC algorithm from the notes:

```
In [12]: 1 chain = []
2 # initial guess -- in this case, doesn't have to be very good!
3 params = [200., 1.]
4 jumpsizes = [10., 0.1]
5 logprob = good_line_logprob(params)
6 for i in 1:1_000
7     params_new = params .+ randn(2) .* jumpsizes
8     logprob_new = good_line_logprob(params_new)
9     if (exp(logprob_new - logprob) >= rand(Float64))
10         logprob = logprob_new
11         params = params_new
12     end
13     append!(chain, params)
14 end
15 # append! makes "chain" a 1-d vector; reshape to a matrix
16 chain = reshape(chain, (2,Int64(length(chain)/2)))';
```

Notice that we're taking 1,000 steps here, so the chain will have shape (1000 x 2).

```
In [13]: 1 p1 = plot(chain[:,1], xlabel="MCMC step", ylabel="Parameter value", label="b")
2 p2 = plot(chain[:,2], xlabel="MCMC step", ylabel="Parameter value", label="m")
3 plot(p1, p2)
```

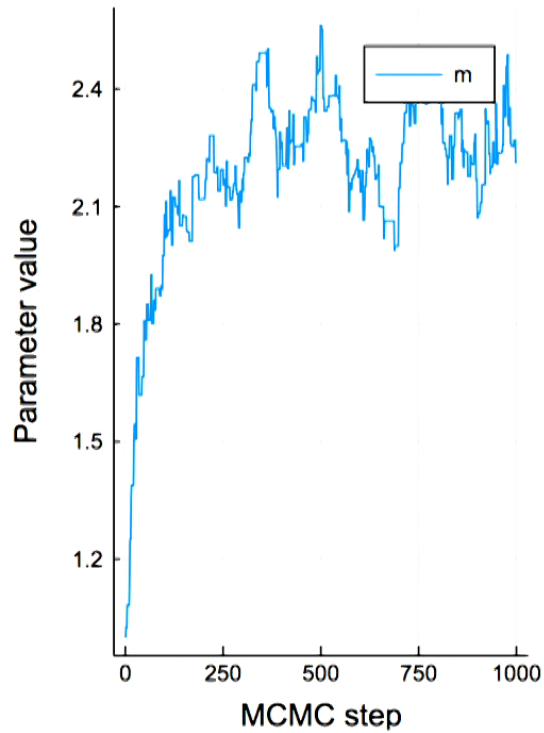
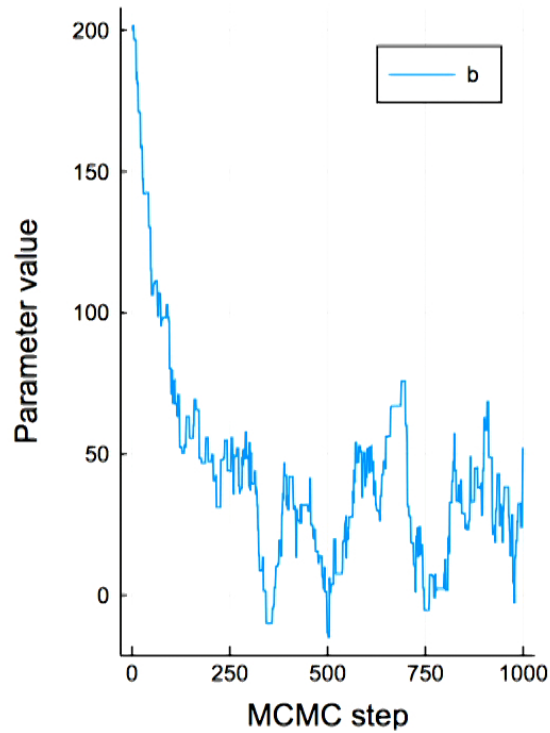
Out[13]:



Notice that we're taking 1,000 steps here, so the chain will have shape (1000 x 2).

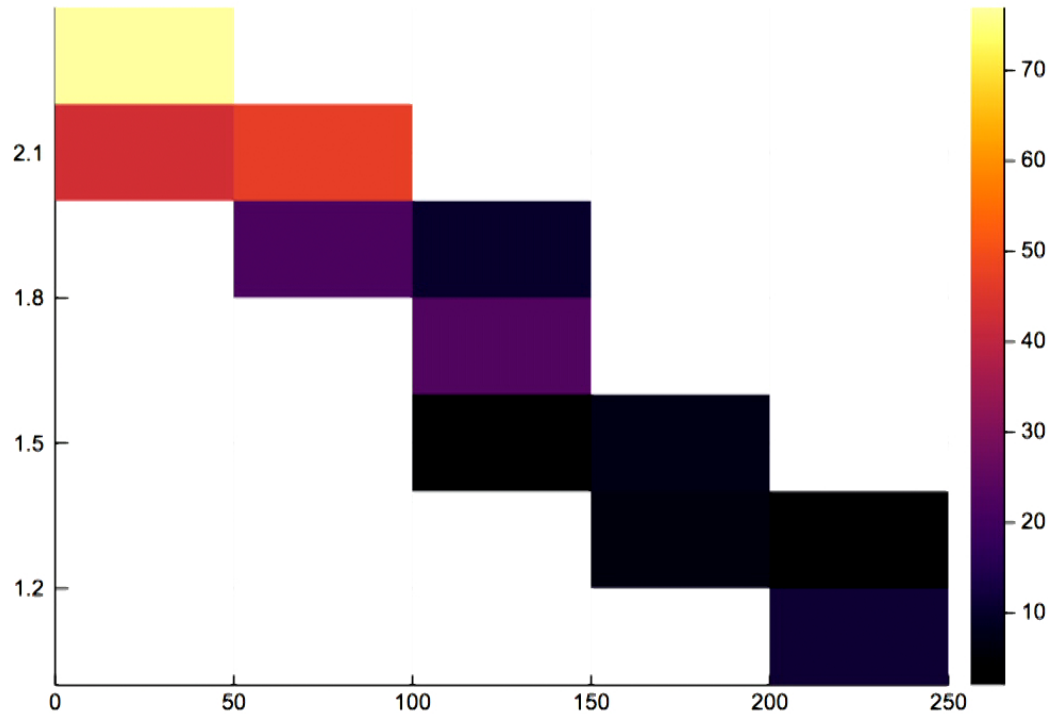
```
In [13]: 1 p1 = plot(chain[:,1], xlabel="MCMC step", ylabel="Parameter value", label="b")  
2 p2 = plot(chain[:,2], xlabel="MCMC step", ylabel="Parameter value", label="m")  
3 plot(p1, p2)
```

Out[13]:



```
In [14]: 1 histogram2d(chain[1:250, 1], chain[1:250, 2])
```

Out[14]:

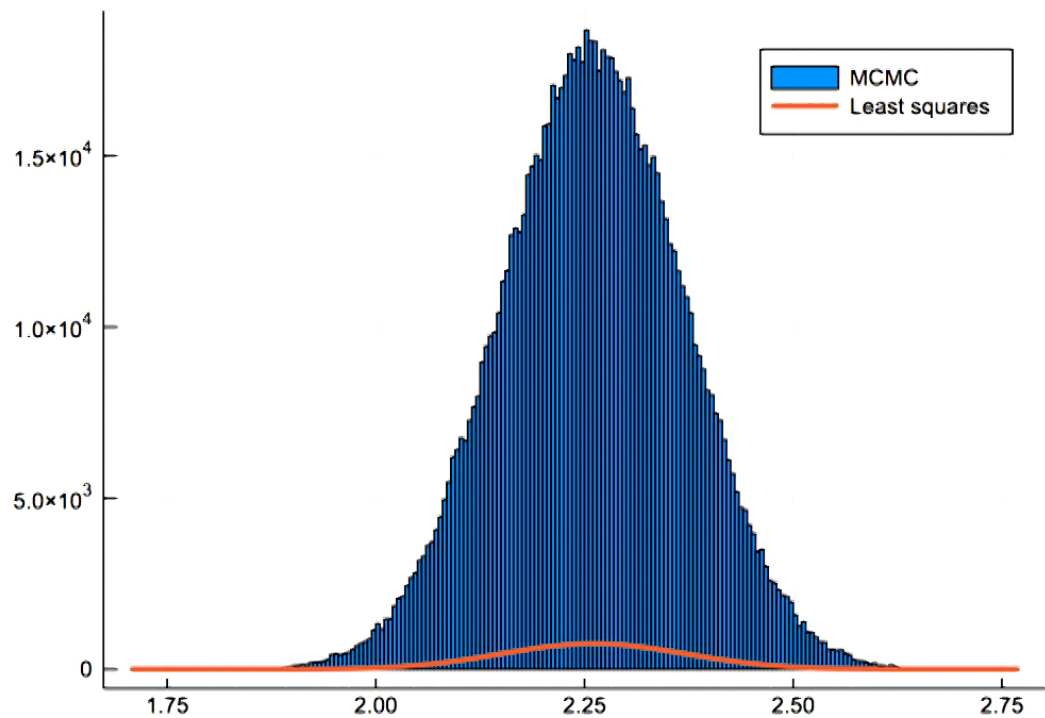


```
In [15]: 1 plot(chain[:,1], chain[:,2], color=:black, label="", xlabel="b", ylabel="m")  
2 scatter!(chain[:,1], chain[:,2], zcolor=1:size(chain)[1], ms=3, label="MCMC positions", col
```

b

```
In [21]: 1 histogram(chain[burnin:end,2], label="MCMC")  
2 v = C_ls[2,2]  
3 plot!(x->750 * exp(-0.5 * (x - m_ls)^2 / v), label="Least squares", linewidth=3)
```

Out[21]:



Now, time to play with this algorithm!

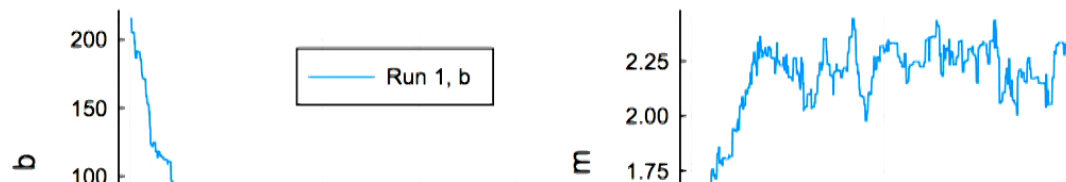
1. Notice how we had to set the "jump sizes". Try changing the jump sizes, run MCMC for 1,000 steps, and observe what happens with the traces. Also observe what happens to the "acceptance" fraction. Can you explain what is happening?
2. Think about the acceptance fraction. When the algorithm is working well, what do you think the acceptance fraction should be? What would you change if the acceptance fraction was too small?

```
In [22]: 1 initial = [200., 1.]  
2 jumps1 = [10., 0.1]  
3 chain1, logprobs1, acceptance1 = run_mcmc(good_line_logprob, initial, jumps1, 1_000);  
4 jumps2 = [30., 0.3]  
5 chain2, logprobs2, acceptance2 = run_mcmc(good_line_logprob, initial, jumps2, 1_000);  
6 acceptance1, acceptance2
```

Out[22]: (0.287, 0.083)

```
In [23]: 1 p1b = plot(chain1[:,1], ylabel="b", label="Run 1, b")  
2 p1m = plot(chain1[:,2], ylabel="m", label="Run 1, m", legend=:bottomright)  
3 p2b = plot(chain2[:,1], ylabel="b", label="Run 2, b")  
4 p2m = plot(chain2[:,2], ylabel="m", label="Run 2, m", legend=:bottomright)  
5 plot(p1b, p1m, p2b, p2m)
```

Out[23]:

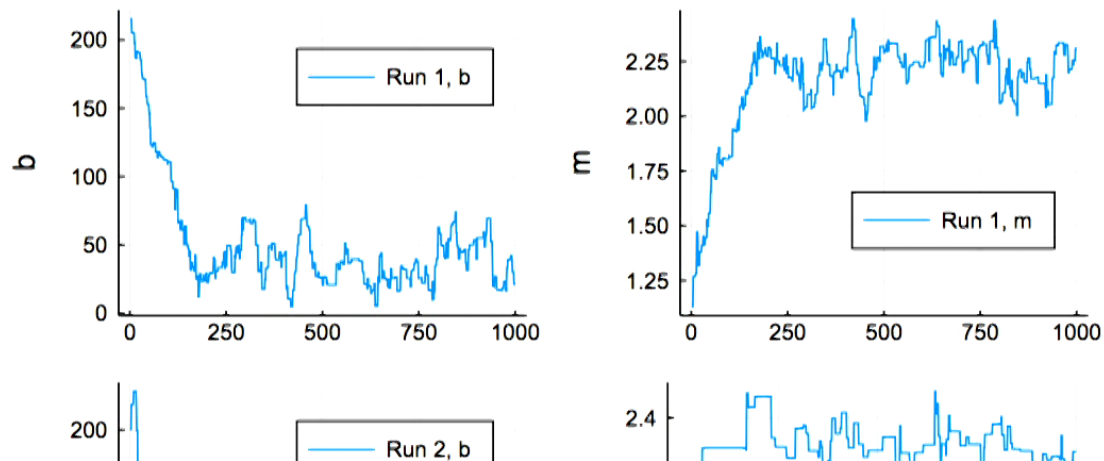



```
In [22]: 1 initial = [200., 1.]  
2 jumps1 = [10., 0.1]  
3 chain1, logprobs1, acceptance1 = run_mcmc(good_line_logprob, initial, jumps1, 1_000);  
4 jumps2 = [30., 0.3]  
5 chain2, logprobs2, acceptance2 = run_mcmc(good_line_logprob, initial, jumps2, 1_000);  
6 acceptance1, acceptance2
```

Out[22]: (0.287, 0.083)

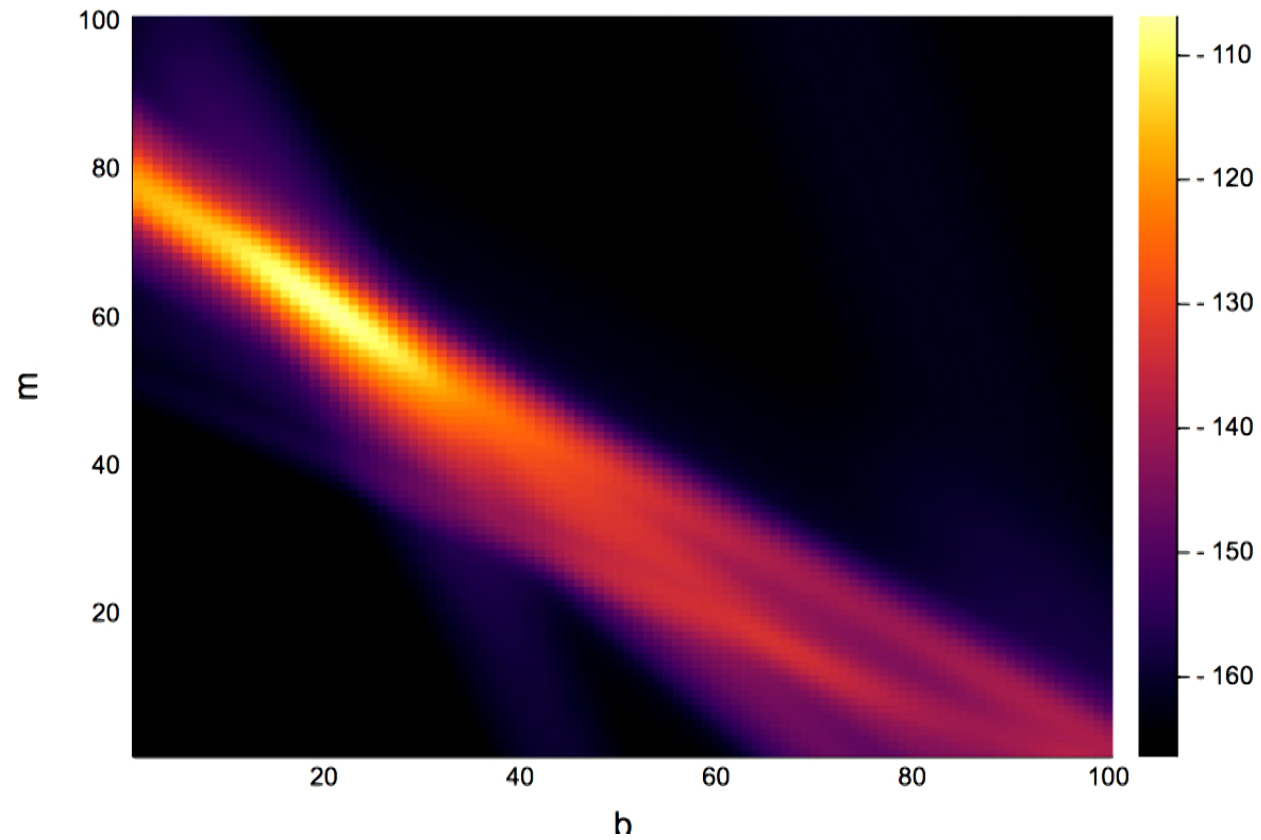
```
In [23]: 1 p1b = plot(chain1[:,1], ylabel="b", label="Run 1, b")  
2 p1m = plot(chain1[:,2], ylabel="m", label="Run 1, m", legend=:bottomright)  
3 p2b = plot(chain2[:,1], ylabel="b", label="Run 2, b")  
4 p2m = plot(chain2[:,2], ylabel="m", label="Run 2, m", legend=:bottomright)  
5 plot(p1b, p1m, p2b, p2m)
```

Out[23]:



```
13     end  
14 end  
15 heatmap(LL, xlabel="b", ylabel="m")
```

Out[41]:



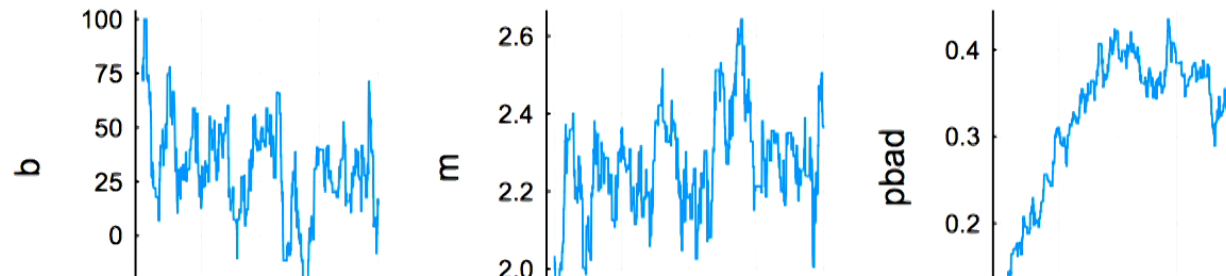
Let's run it with our foreground-background probability space for 1,000 steps and see what we get.

```
In [10]: 1 initial = [50., 2., 0.1, mean(y), var(y)]  
2 jumpsizes = [10., 0.1, 0.01, 10., 100.]  
3 chain,logprobs,acceptance = run_mcmc(fgbg_logprob, initial, jumpsizes,  
4 acceptance
```

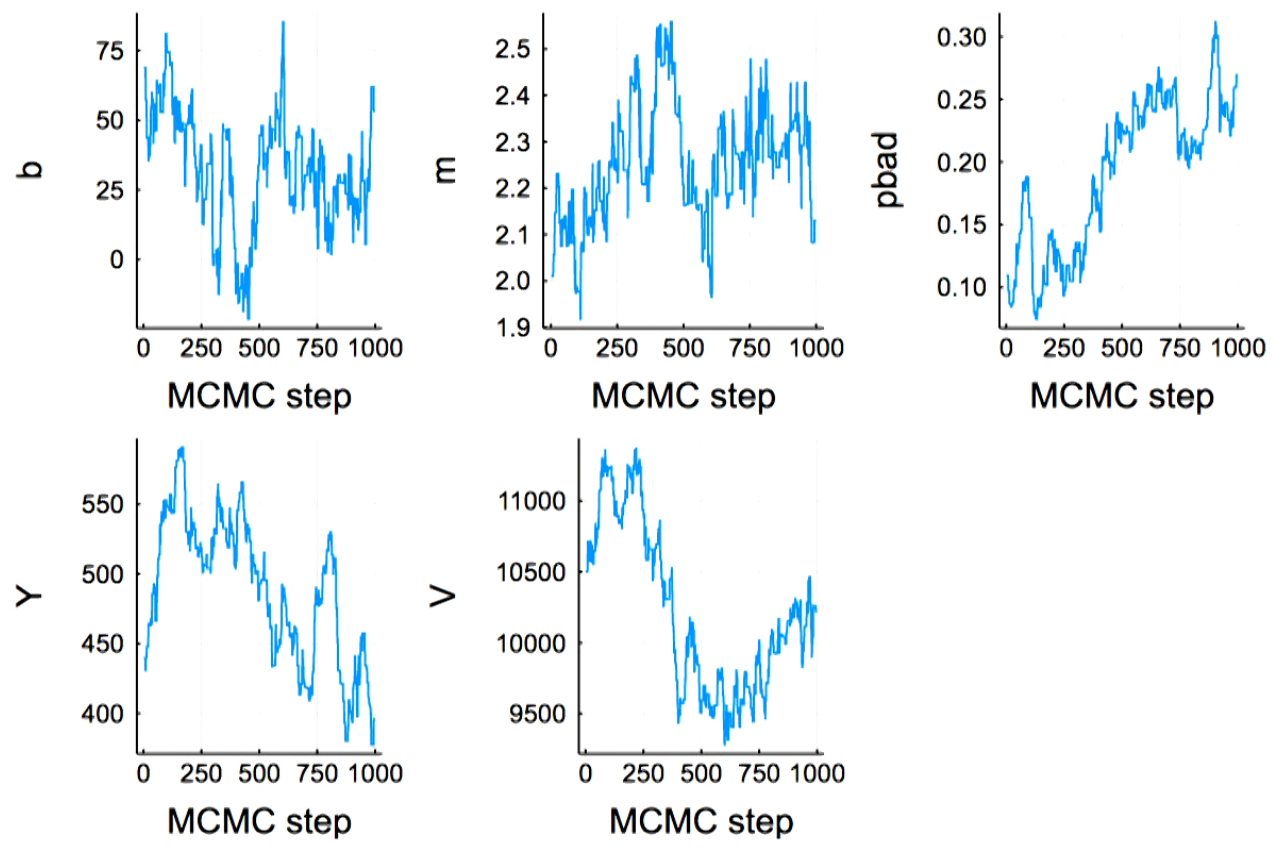
Out[10]: 0.345

```
In [11]: 1 names = ["b", "m", "pbad", "Y", "V"]  
2 plots = []  
3 for (i,n) in enumerate(names)  
4     p = plot(chain[:,i], xlabel="MCMC step", ylabel=n, label="") # "Para  
5     push!(plots, p)  
6 end  
7 plot(plots...)
```

Out[11]:



Out[43]:



That actually looks pretty good -- it hasn't quite settled down into a stable pattern yet but it looks

```
In [12]: 1 function run_gibbs_mcmc(logprob_function, initial, jumpsizes, nsteps)
2         nparams = length(initial)
3         chain = zeros((nsteps, nparams))
4         logprobs = zeros(nsteps)
5         params = initial
6         logprob = logprob_function(params)
7         accepts = zeros(nparams)
8         tries = zeros(nparams)
9         for i in 1:nsteps
10            # Choose which parameter to adjust this time
11            j = rand(1:nparams)
12            # Jump just that parameter.
13            params_new = copy(params)
14            params_new[j] += randn() * jumpsizes[j]
15            logprob_new = logprob_function(params_new)
16            if (exp(logprob_new - logprob) >= rand(Float64))
17                logprob = logprob_new
18                params = params_new
19                accepts[j] += 1
20            end
21            tries[j] += 1
22            chain[i,:] .= params
23            logprobs[i] = logprob
24        end
25        return chain, logprobs, accepts ./ tries
26    end;
```

```
23         logprobs[i] = logprob
24     end
25     return chain, logprobs, accepts ./ tries
26 end;
```

```
In [13]: 1 initial = [50., 2., 0.1, mean(y), var(y)]
2 jumpsizes = [10., 0.1, 0.01, 10., 100.]
3 chain,logprobs,acceptance = run_gibbs_mcmc(fgbg_logprob, initial, jumpsizes,
4 acceptance
```

```
Out[13]: 5-element Array{Float64,1}:
0.4729064039408867
0.41397849462365593
0.9526315789473684
0.9757281553398058
0.9953488372093023
```

Aha, so look, the acceptance rates for my "Y" and "V" parameters are nearly 1, so we can crank those ones up.

It is worth noting, also, that the acceptance rates can be quite different in different parts of the parameter space, depending on how narrow and steep the "ridge" of high probability is.

```
In [14]: 1 initial = [50., 2., 0.1, mean(y), var(y)]
```

```
23         logprobs[i] = logprob
24     end
25     return chain, logprobs, accepts ./ tries
26 end;
```

```
▶ In [44]: 1 initial = [50., 2., 0.1, mean(y), var(y)]
           2 jumpsizes = [10., 0.1, 0.02, 10., 100.]
           3 chain,logprobs,acceptance = run_gibbs_mcmc(fgbg_logprob, initial, jumpsizes,
           4 acceptance
```

Out[44]: 5-element Array{Float64,1}:
0.6201923076923077
0.3641304347826087
0.9396984924623115
0.9593908629441624
0.9952830188679245

Aha, so look, the acceptance rates for my "Y" and "V" parameters are nearly 1, so we can crank those ones up.

It is worth noting, also, that the acceptance rates can be quite different in different parts of the parameter space, depending on how narrow and steep the "ridge" of high probability is.

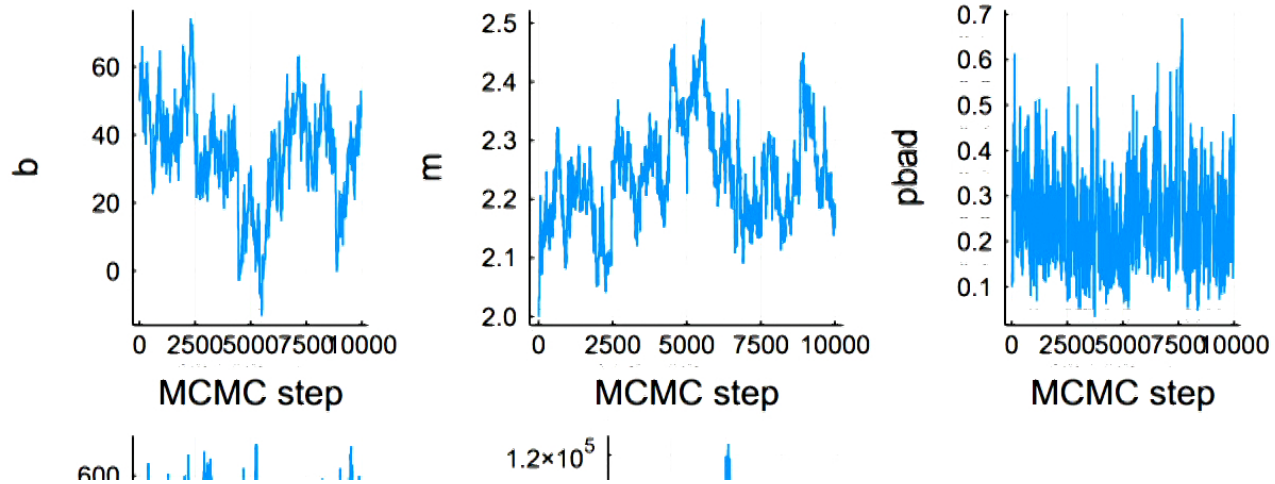
```
In [14]: 1 initial = [50., 2., 0.1, mean(y), var(y)]
```

0.739412057797708
0.7092337917485265
0.7620020429009193
0.8497281265447355

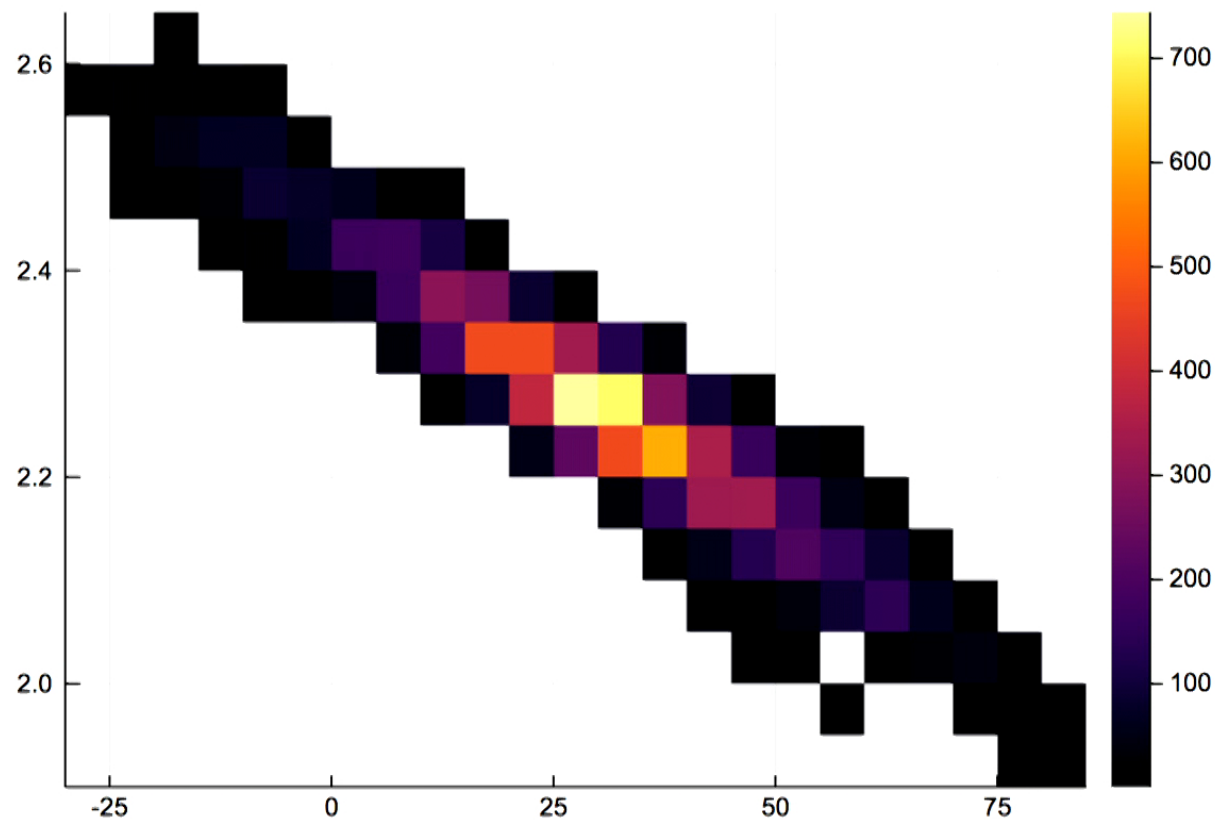
In [55]:

```
1 names = ["b", "m", "pbad", "Y", "V"]  
2 plots = []  
3 for (i,n) in enumerate(names)  
4     p = plot(chain[:,i], xlabel="MCMC step", ylabel=n, label="")  
5     push!(plots, p)  
6 end  
7 plot(plots...)
```

Out[55]:



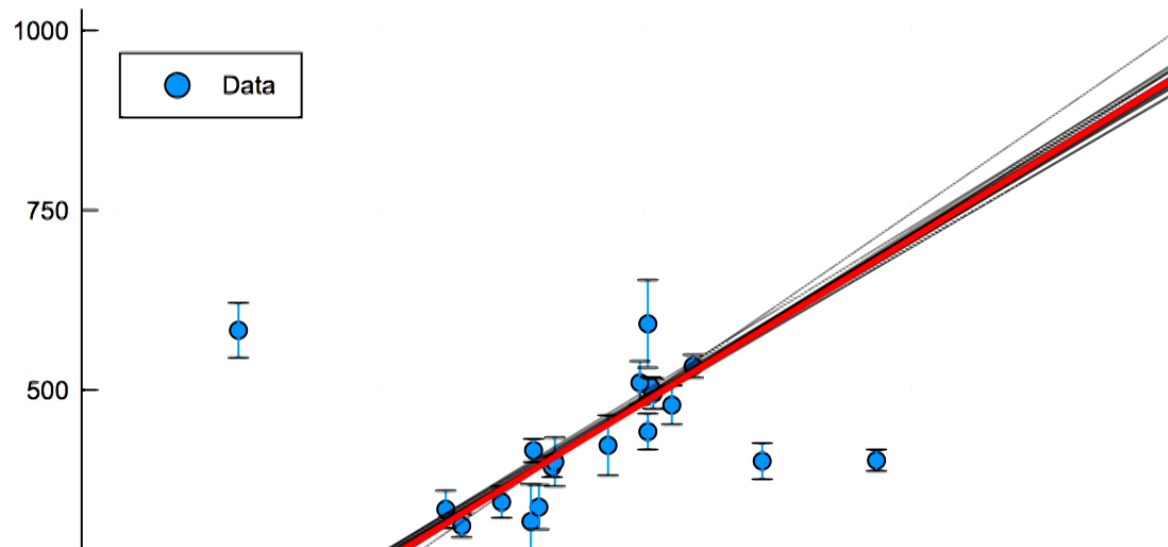
Out[58]:



```
In [19]: 1 samples = chain[rand(1000:(size(chain)[1]), 20), :]  
2 plot(x, y, yerr=yerr, seriestype=:scatter, legend=:topleft, label="Data")
```

```
In [59]: 1 samples = chain[rand(1000:(size(chain)[1]), 20), :]  
2 plot(x, y, yerr=yerr, seriestype=:scatter, legend=:topleft, label="Data")  
3 for i in 1:size(samples)[1]  
4     b = samples[i, 1]  
5     m = samples[i, 2]  
6     plot!([0,400], x->b+m*x, alpha=0.5, color=:black, label="")  
7 end  
8 meanb,meanm = mean(chain[1000:end, 1]), mean(chain[1000:end, 2])  
9 plot!([0,400], x->meanb+meanm*x, color=:red, label="", linewidth=3)
```

Out[59]:



Out[22]:

