PIBO final project: Linear and nonlinear Bayesian modeling for weather data

Maike F. Holthuijzen

December 11, 2017

Contents

List of Figures

1 Introduction

My objectives for this project were to use simulation data to investigate the relationships between temperature and precipitation based on elevation and latitude. These relationships were previously determined by Winter et al. (2016) for elevation and altitude adjustments for temperature and precipitation at weather stations. The equation for determining the temperature at a station, Z_{sta}, based on elevation (Z), and latitude (ϕ) is :

$$T_{sta} = \beta_0 + \beta\phi_{sta} + \gamma Z \tag{1}$$

, where $\beta = -4$ and $\gamma = -5.385$ (in $°Ckm^{-1}$).

The lapse rate equation for precipitation (P_{sta}), as a function of station elevation (Z), the reference elevation (Z_{ref}), and the reference precipitation (P_{ref}) is:

$$P_{sta} = P_{ref}\frac{1 + \chi(Z_{sta} - Z_{ref})}{1 - \chi(Z_{sta} - Z_{ref})} \tag{2}$$

, where $\chi = 0.00025, P_{ref} = 2cm$, and $Z_{ref} = 150m$ (Winter et al. (2016)).

2 Methods and results

2.1 Temperature

I simulated data for equation 1 with the following code:

```
#intercept is never given in paper,
#so I used 190, which gives reasonable values for y (station temperature).
T0 = 190

#station elevation
z = rlnorm(1000, meanlog = 6, sd=.5) / 1000

#beta is effect of latitude (from Blandford et al 2003)
Beta = -4

#gamma is effect of elevation (from Winter et al 2016)
gamma = -5.385


#phi = latitude range for D3 study area
phi = runif(1000, min = 44, max = 45.5)

#add error
err = rnorm(1000,0,1)

#y is surface temperature at weather station
y = T0 + Beta*phi + gamma*z + err
```

Next, I used the `lm` function in he R to see if I could recover the parameters above. The parameter estimates from `lm` are fairly close to the actual values:

```r
#make into a data frame
tempdata = as.data.frame(cbind(y, z, phi))
names(tempdata) = c("y", "z", "phi")
phi = tempdata$phi
z = tempdata$z
y = tempdata$y
N = length(y)

#use lm to see if coefficients can be estimated accurately
mod1 = lm(y ~ phi + z, data = tempdata)
summary(mod1)

##
## Call:
## lm(formula = y ~ phi + z, data = tempdata)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.2742 -0.6705  0.0094  0.6341  3.8056
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 192.84174    3.36231   57.35   <2e-16 ***
## phi          -4.06231    0.07508  -54.10   <2e-16 ***
## z            -5.39275    0.13422  -40.18   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.02 on 997 degrees of freedom
## Multiple R-squared:  0.8176,Adjusted R-squared:  0.8172
## F-statistic:  2235 on 2 and 997 DF,  p-value: < 2.2e-16
```

Next, I used the package `Rjags` to conduct a Bayesian regression analysis to recover the parameters for β, γ, and σ (which was set at 1 in the simulation data).

The following code executes a Bayesian regression analysis in the `Rjags` package. I used 4 chains and 100000 iterations. I also obtained predicted posterior distributions of temperature for elevations of 500 and 1000 and latitudes of 44.2 and 45.

```r
#not run
library(coda)
modelString<-"
model {
for (i in 1:N){
y[i] ~ dnorm(y.hat[i], tau)
y.hat[i] <- int + beta1 * phi[i] + gamma*z[i]
}
int ~ dnorm(0, .0001)
beta1 ~ dnorm(0, .0001)
gamma ~ dnorm(0,.001)
tau <- pow(sigma, -2)
```

```
sigma ~ dunif(0, 100)

#predicted values for elevation 500 lat 44.2
#and elevation 1000, lat 45
pred1 = int + beta1 * 44.2 + gamma*0.500
pred2 = int+ beta1 * 45 + gamma*1.000

pred1dist ~ dnorm(pred1, tau)
pred2dist ~ dnorm(pred2, tau)
}
"
writeLines(modelString, con='hcnmodel1.txt')
tempdata2 <- with(tempdata, list(y = y, z = z, phi = phi, N = length(y)))

jags = jags.model('hcnmodel1.txt',
                  data = list('z' = z,
                              'phi' = phi,
                              'y' = y,
                              'N' = N),
                  inits<-list(
                    list('int' = 100, 'beta1' = -4, 'gamma' = 2, 'sigma' = 5),
                    list('int' = 200, 'beta1' = 1, 'gamma' = -1, 'sigma' = 0.5),
                    list('int' = 100, 'beta1' = -.1, 'gamma' = 2, 'sigma' = 7),
                    list('int' = 500, 'beta1' = 2, 'gamma' = 5, 'sigma' = 1)),
                  n.chains = 4,
                  n.adapt = 100)
update(jags, 100000)
codaSamples = coda.samples(jags, c('int','beta1', 'gamma', 'sigma'), 1000, 1)
```

The posterior distribution for β and γ are also reasonably accurate. Figure 1 shows the posterior distribution for β_0, β , γ, and σ. The chains for β and β_0 ("int" in the figure) are not particularly well-mixed, but the density is still greatest around the original parameter values for β and β_0. The predicted values are reasonable. I expected to obtain lower predicted values for temperature as elevation and latitude increased from 500 to 1000 and 44.2 to 45, respectively (Figure 1). The predicted mean is 10.545°C for the lower elevation and latitude and 4.622°C for the higher elevation and latitude.

```
#not run
summary(codaSamples)

# Iterations = 50101:70100
# Thinning interval = 1
# Number of chains = 4
# Sample size per chain = 20000
#
# 1. Empirical mean and standard deviation for each variable,
#    plus standard error of the mean:
#
#              Mean      SD  Naive SE Time-series SE
# beta1      -4.097 0.07860 2.779e-04      1.165e-02
```

```
# gamma      -5.286 0.13535 4.785e-04      1.372e-03
# int       194.271 3.51509 1.243e-02      5.148e-01
# pred1dist  10.545 1.04279 3.687e-03      3.673e-03
# pred2dist   4.622 1.03816 3.670e-03      3.657e-03
# sigma       1.038 0.02331 8.242e-05      9.509e-05
#
# 2. Quantiles for each variable:
#
#               2.5%     25%     50%     75%   97.5%
# beta1      -4.2868  -4.141  -4.094  -4.050  -3.961
# gamma      -5.5506  -5.377  -5.286  -5.195  -5.021
# int       188.2098 192.175 194.156 196.264 202.755
# pred1dist   8.4976   9.842  10.547  11.254  12.581
# pred2dist   2.5804   3.923   4.623   5.320   6.658
# sigma       0.9932   1.022   1.037   1.053   1.085
```

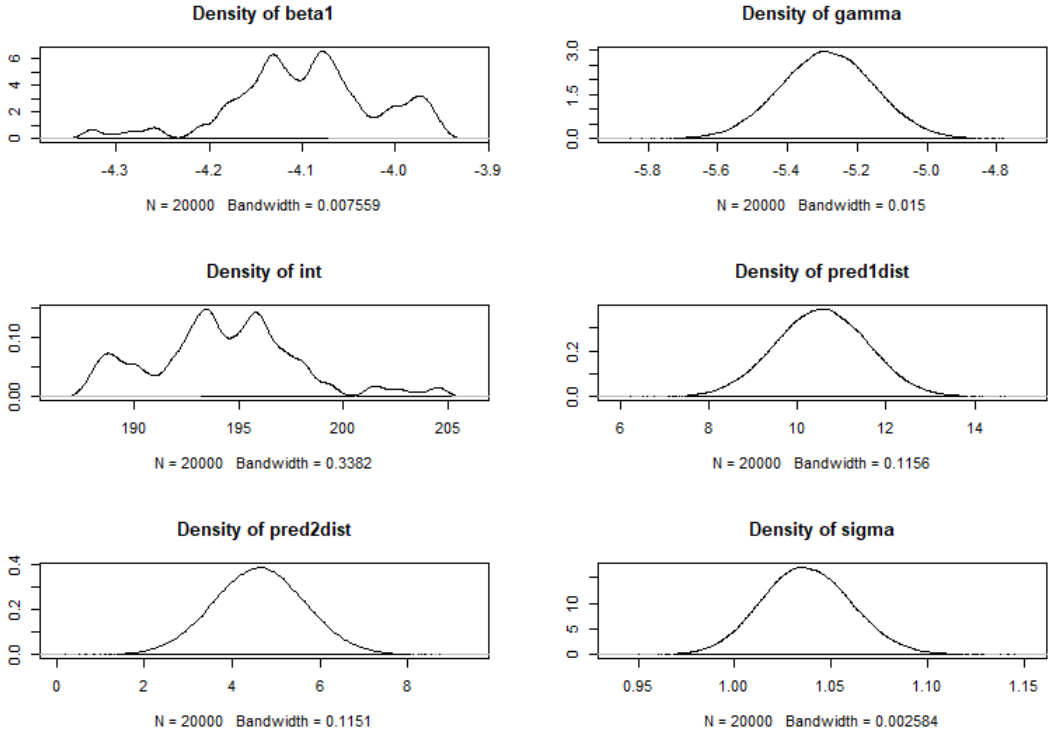

Figure 1: Posterior distributions for from jags model, including predicted distributions (pred1dist and pred2dist).

Finally, I conducted another Bayesian analysis of the data, this time using the R package `rstan`. The code for the model is similar to the jags code above:

```
fileName = "hcnstan.txt"
stan_code = readChar(fileName, file.info(fileName)$size)
cat(stan_code)

"
data {
```

```
   // Define variables in data
   // Number of observations (an integer)
   int<lower=0> N;
   // Number of parameters
   int<lower=0> p;
   real y[N];
//Variables
real<lower=0> z[N];
 real<lower=0> phi[N];
 }

parameters {
  // Define parameters to estimate
   real beta[p];

  // standard deviation (a positive real number)
   real<lower=0> sigma;
 }

 transformed parameters  {
   // Mean
   real mu[N];
   for (i in 1:N) {
     mu[i] = beta[1] + beta[2]*z[i] + beta[3]*phi[i];
   }
 }

 model {
   // Prior part of Bayesian inference (flat if unspecified)

   // Likelihood part of Bayesian inference
     y ~ normal(mu, sigma);
 }
"
```

The following lines execute the sampling for this model:

```
standat = list(N    = nrow(tempdata),
               p        = 3,
               y        = tempdata2$y,
               z        = tempdata2$z,
               phi      = tempdata2$phi)
resStan = stan(model_code = stan_code, data = standat,
                  chains = 3, iter = 3000, warmup = 500, thin = 10)
```

The stan model appeared to estimate the parameters more accurately with less iterations than the jags model. Results for the quantiles of the posterior distributions for parameters are shown below. Figures 2 and 3 show histograms for the intercept ("Beta1") and ϕ and γ ("beta2" and "beta3"), and σ, respectively. The trace plots in 4 do appear well-mixed, in contrast to those obtained from the jags model.

```
print(resStan, "beta")
# Inference for Stan model: 28f0102e61f8a3728c953224ef467974.
# 3 chains, each with iter=3000; warmup=500; thin=10;
# post-warmup draws per chain=250, total post-warmup draws=750.
#
#           mean se_mean   sd   2.5%    25%    50%    75%  97.5% n_eff Rhat
# beta[1] 193.76    0.12 3.33 186.83 191.57 193.71 196.01 199.82   750    1
# beta[2]  -5.40    0.00 0.13  -5.66  -5.49  -5.40  -5.31  -5.13   750    1
# beta[3]  -4.08    0.00 0.07  -4.22  -4.14  -4.08  -4.04  -3.93   750    1
#
# Samples were drawn using NUTS(diag_e) at Sun Dec 10 13:19:16 2017.
# For each parameter, n_eff is a crude measure of effective sample size,
# and Rhat is the potential scale reduction factor on split chains (at
# convergence, Rhat=1).
print(resStan, "sigma")


#       mean se_mean   sd 2.5%  25%  50%  75% 97.5% n_eff Rhat
# sigma 0.99       0 0.02 0.95 0.98 0.99 1.01  1.03   747    1
```

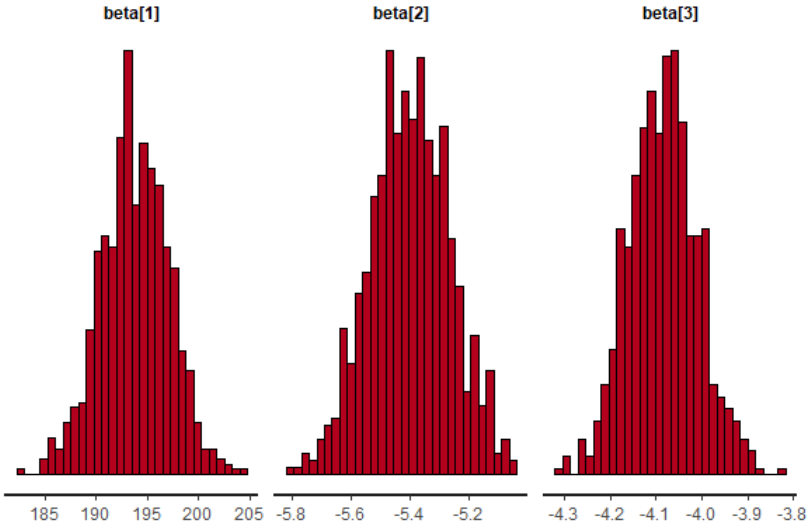

Figure 2: Posterior distributions for coefficients from stan model.

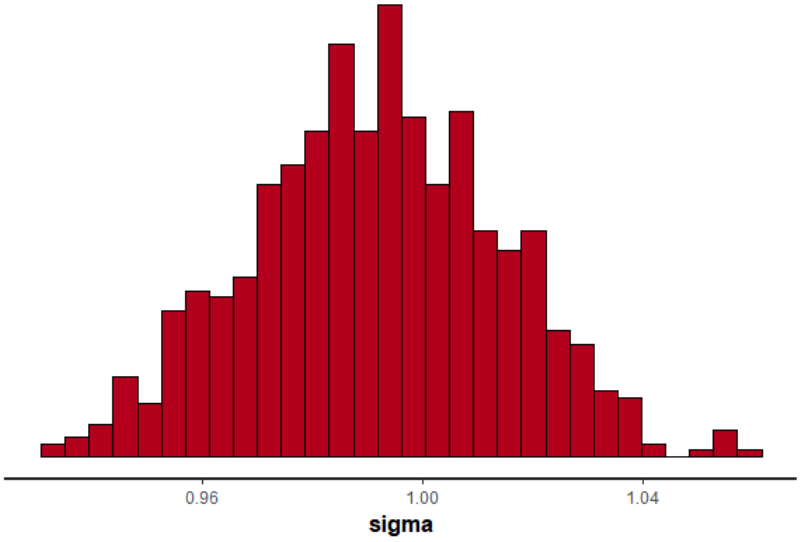

Figure 3: Posterior distribution for σ.

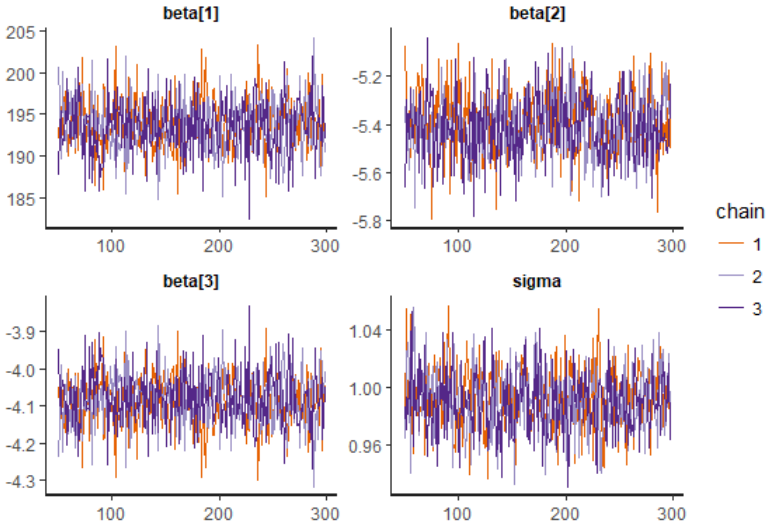

Figure 4: Traceplots from stan model.

2.2 Precipitation

I simulated data for equation 2 with the following code:

```
#zref is reference elevation
zref = 150

#chi (from Winter et al. 2016)
chi = 0.000250

#station elevations
zsta=rlnorm(1000, meanlog = 6, sd=.5)
#take log of station elevation for nls function
zsta = log(zsta)
```

```
x = zsta - zref

#Pref = 2 from Winter et al. 2016
pref = 2

#add normally distributed error
err = rnorm(1000, 0, .001)

#y is the estimated precipitation at a given station
y = pref*((1+chi*(x)) / (1 - chi*(x))) + err

#store data in a dataframe
dat = as.data.frame(cbind(x,y))
```

I wanted to determine if R's nonlinear modeling function `nls` could recover the parameters χ and P_{ref}. First, I took the natural log of station elevations to ensure normality and homogeneity of residuals. The `nls` function is able to recover the parameters with a very high degree of accuracy.

```
mymod=nls(y ~ pref*((1+chi*(x)) / (1 - chi*(x))),
    data = dat, start = list(pref = 1, chi=.001), control = list(maxiter=100000))
summary(mymod)

##
## Formula: y ~ pref * ((1 + chi * (x))/(1 - chi * (x)))
##
## Parameters:
##       Estimate Std. Error t value Pr(>|t|)
## pref 1.998e+00  9.665e-03  206.72   <2e-16 ***
## chi  2.463e-04  1.677e-05   14.68   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.001004 on 998 degrees of freedom
##
## Number of iterations to convergence: 5
## Achieved convergence tolerance: 8.379e-07
```

Next, I used a Bayesian approach with the R package `rjags`. In contrast to the previous nonlinear approach, I did not log-transform the station elevations. I updated the jags model with 40000 steps and used 4 chains.

```
#not run
#create the same dataset without log-transformed elevation values
#station elevations
zsta=rlnorm(1000, meanlog = 6, sd=.5)

x = zsta - zref

#Pref = 2 from Winter et al. 2016
```

```r
pref = 2

#add normally distributed error
err = rnorm(1000, 0, .001)

#y is the estimated precipitation at a given station
y = pref*((1+chi*(x)) / (1 - chi*(x))) + err

n = length(y)

mydata = list(x = x, y = y, n=n)

#jags model
jags.script <- "
model{
# likelihood
for( i in 1:n) {
y[i] ~ dnorm(mu[i], tau)
mu[i] <- pref*((1 + chi*(x[i])) / (1 - chi*(x[i])))
}
# priors
pref ~  dunif(0, 10)
chi  ~ dnorm(0, 0.0001)
tau  ~ dgamma(0.001, 0.001)
sigma <- 1 / sqrt(tau)
}
"

#run jags model with 40000 steps
mod1 = jags.model(textConnection(jags.script), data = mydata, n.chains = 4,
                  n.adapt = 100)
update(mod1, 40000)
mod1.samples = coda.samples(model = mod1, variable.names = c("pref", "chi", "sigma"), n.

summary(mod1.samples)
# Iterations = 42101:46100
# Thinning interval = 1
# Number of chains = 4
# Sample size per chain = 4000
#
# 1. Empirical mean and standard deviation for each variable,
#    plus standard error of the mean:
#
#             Mean        SD  Naive SE Time-series SE
# chi   0.0002499 1.461e-07 1.155e-09      2.045e-09
# pref  1.9998485 3.621e-04 2.862e-06      5.134e-06
# sigma 0.0097065 2.171e-04 1.717e-06      1.776e-06
#
# 2. Quantiles for each variable:
#
```

```
#             2.5%       25%       50%     75%     97.5%
# chi    0.0002496 0.0002498 0.0002499 0.00025 0.0002502
# pref   1.9991305 1.9996026 1.9998503 2.00010 2.0005409
# sigma 0.0092953 0.0095591 0.0097026 0.00985 0.0101411
```

The mean values below show that the jags model was able to recover the parameters very accurately. The posterior density estimates are close to the original parameter values, and the trace plots for the four chains appear well-mixed (5).

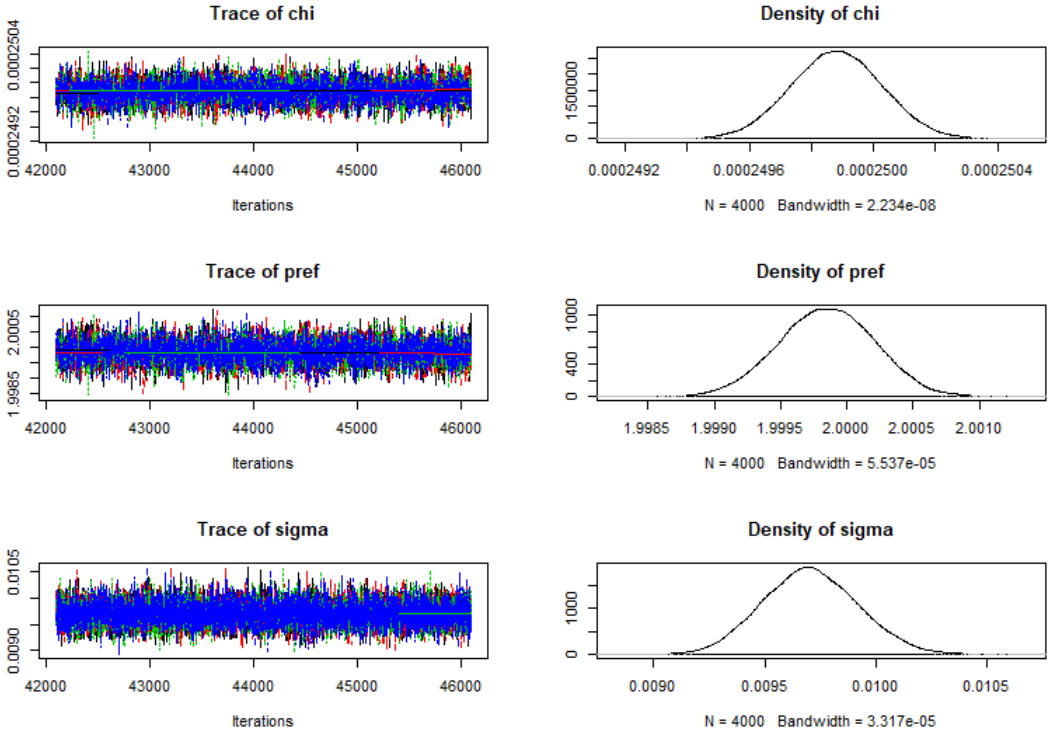

Figure 5: Trace plots and posterior densities from the jags model for precipitation.

Finally, I attempted to run the model in Stan for comparison. I received several warnings after compiling the model, and I attempted to resolve them by adjusting the control parameters of `max_treedepth` and `delta_adjust`. However, I still obtained unreasonable estimates for the parameters χ and P_{ref}.

```
#stan model
stanprecip = "
data {
    int <lower=1> N;
    vector [N] y;
    vector [N] x;
  }
  parameters {
    real <lower=0> sigma;
    real chi;
    real <lower=0> pref;
  }
  model {
```

```
    vector [N] denom;
    for (i in 1:N) denom[i] = 1 / (1 - chi*x[i]);
    y ~ normal(pref * (1 + chi*x) .* denom, sigma);
    sigma ~ normal(0, 10);
    chi ~ normal(0, .0001);
    pref ~ uniform(0, 10);
  }
"
```

3 References

References

J. M. Winter, B. Beckage, G. Bucini, R. M. Horton, and P. J. Clemins. Development and evaluation of high-resolution climate simulations over the mountainous northeastern united states. *Journal of Hydrometeorology*, 17(3):881–896, 2016.