Class 7: Variable selection and non-parametrics

Andrew Parnell
andrew.parnell@ucd.ie

Learning outcomes:

- Be able to specify complex prior distributions
- Understand the difference between shrinkage and variable selection
- Be able to fit shrinkage and selection models in JAGS/Stan
- Be able to fit some non-parametric Bayesian models

Some general lessons from the course

- Don't divide your data up, use all of it together in one model and put priors on the parameters to borrow strength between groups
- Move away from dichotomous hypothesis testing towards model development with the aim of quantifying your effects of interest
- Some steps to model development:
    1. Start with a simple, even stupid model
    2. Optionally simulate from it to define reasonable priors
    3. Fit the model in Stan or JAGS
    4. Get the DIC and perhaps a posterior predictive plot
    5. Try enriching the model by adding layers/variables

Really clever prior distributions

- Most of the models we have covered use the prior distribution in GLMs to vary the slopes and intercepts of a latent linear model
- In this class instead we will use prior distributions to perform some other tricks:
  - Selecting variables in a regression model
  - Non-linear regression models

Shrinkage and variable selection

- Occasionally we have data sets with lots of potential covariates
- If many of them are not linked (either causally or predictively) to the response variable we want to remove them from the model
- In machine learning this is often called *feature selection*
- In a Bayesian world we can remove variables from the model by using the prior to move them towards zero. This is called a *shrinkage prior*

Example data

This is the Prostate data, taken from *Elements of Statistical Learning* by Hastie et al

```
##       lcavol  lweight age      lbph svi       lcp gleason pgg45       lpsa
## 1 -0.5798185 2.769459  50 -1.386294   0 -1.386294       6     0 -0.4307829
## 2 -0.9942523 3.319626  58 -1.386294   0 -1.386294       6     0 -0.1625189
## 3 -0.5108256 2.691243  74 -1.386294   0 -1.386294       7    20 -0.1625189
## 4 -1.2039728 3.282789  58 -1.386294   0 -1.386294       6     0 -0.1625189
## 5  0.7514161 3.432373  62 -1.386294   0 -1.386294       6     0  0.3715636
## 6 -1.0498221 3.228826  50 -1.386294   0 -1.386294       6     0  0.7654678
##   train
## 1  TRUE
## 2  TRUE
## 3  TRUE
## 4  TRUE
## 5  TRUE
## 6  TRUE
```

- Goal is to predict `lpsa` from the other 8 variables

Bayesian variable selection

- We will temporarily ignore borrowing strength, and set up the models as default regression models:

$$y_i \sim N(\alpha + \beta_1 x_{1i} + \ldots + \beta_p x_{pi}, \sigma^2)$$

  so we have p covariates, some of which are possibly useless
- We will place weak prior distributions on the α and σ terms, e.g. normal/Cauchy
- The key prior distribution will be on the β parameters

Approach 1: ridge regression

- A first prior that is common is known as the *ridge* prior distribution. This is:

$$\beta_j \sim N(0, \sigma_b^2), \text{for } j = 1, \ldots, p$$

- When σ_b is small, most of the parameters will be close to zero. When it is big, then we will end up with the standard regression model
- We can learn about σ_b too by including a prior distribution on it
- We always must *standardise the covariates* when running a variable selection model otherwise these priors make no sense

Ridge regression applied to the prostate data

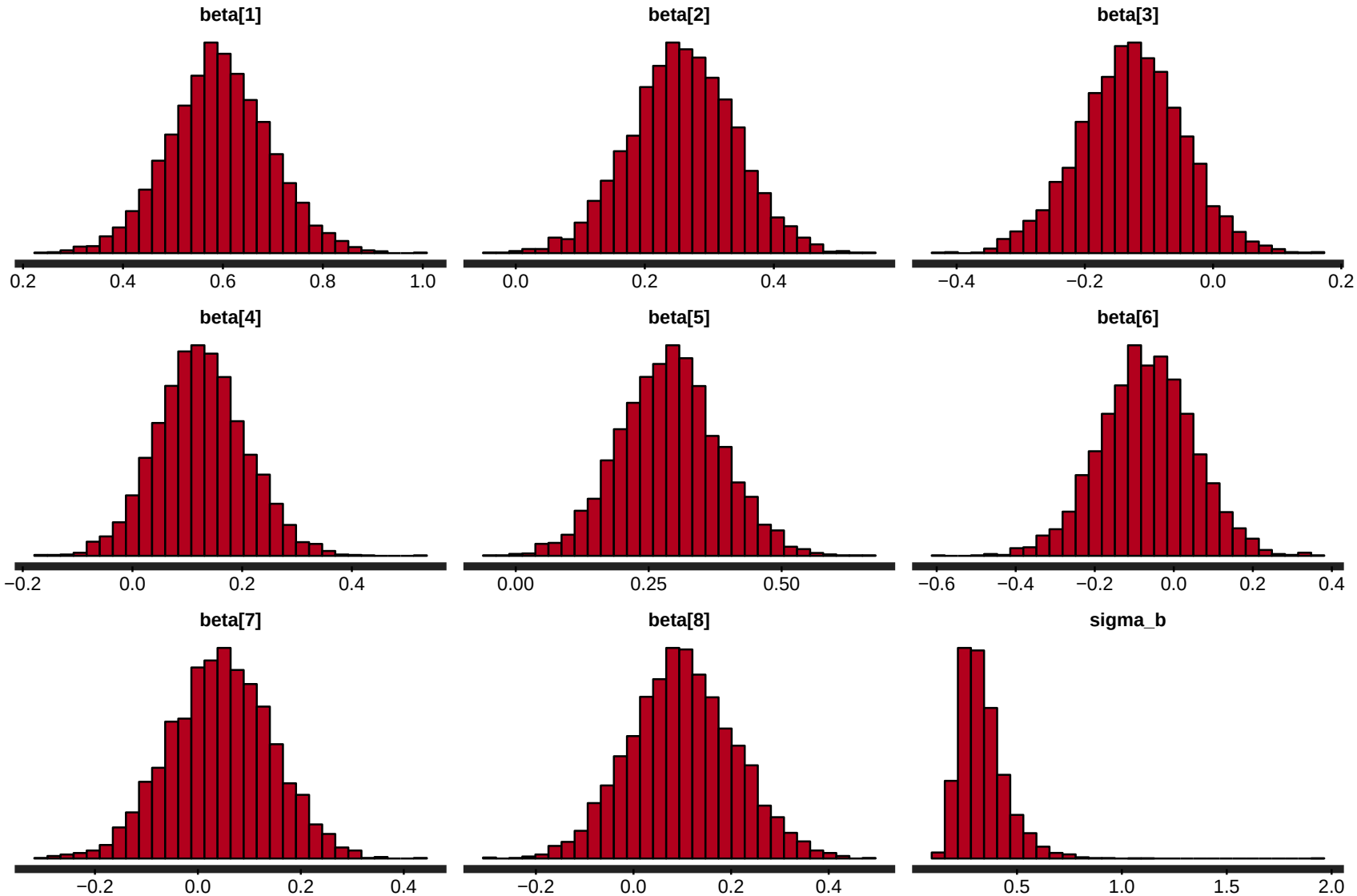
```
stan_code = '
data {
  int<lower=0> N;
  int<lower=0> p;
  vector[N] y;
  matrix[N,p] X;
}
parameters {
  real alpha;
  vector[p] beta;
  real<lower=0> sigma;
  real<lower=0> sigma_b;
}
model {
  y ~ normal(alpha + X * beta, sigma);
  for(j in 1:p)
    beta[j] ~ normal(0, sigma_b);
  alpha ~ normal(0, 10);
  sigma_b ~ cauchy(0, 10);
  sigma ~ cauchy(0, 10);
}
'
```

Running the Stan version

```
X = with(prostate, cbind(lcavol, lweight,
                         age, lbph, svi,
                         lcp, gleason, pgg45))
X_std = apply(X, 2, 'scale')
stan_run = stan(data = list(N = nrow(prostate),
                            p = ncol(X_std),
                            y = prostate$lpsa,
                            X = X_std),
                model_code = stan_code)
```

Stan output

```
stan_hist(stan_run, pars = c('beta','sigma_b'), bins = 30)
```



Lasso

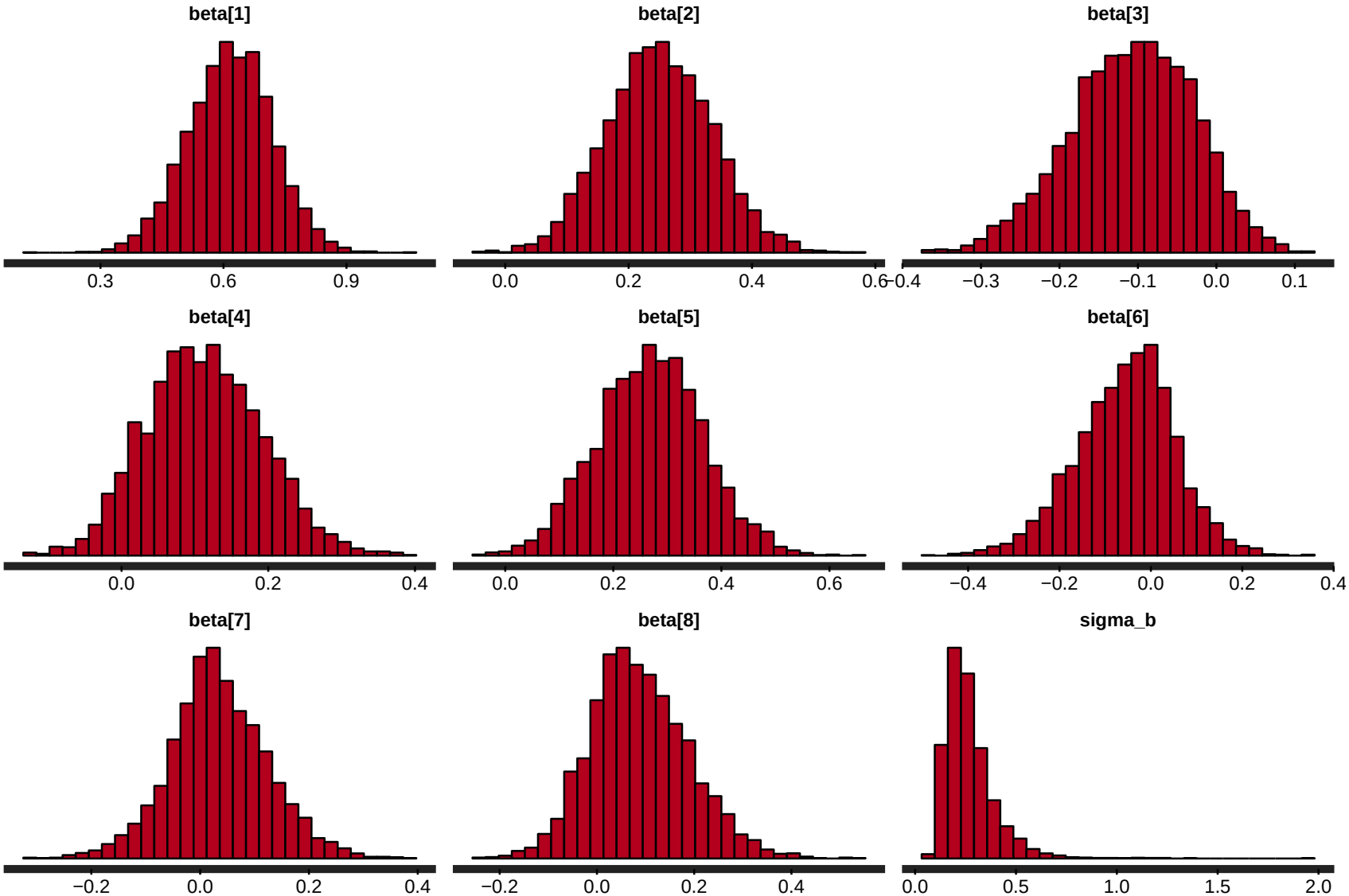
- An alternative shrinkage prior is the *Lasso*

$$\beta_j \sim dexp(0, \sigma_b), \text{for } j = 1, \ldots, p$$

  where *dexp* is the double exponential distribution (AKA the Laplace distribution)
- This distribution is slightly sharper than the ridge prior and so tends to shrink more of the variables to zero
- σ_b has the same effect as previously
- The code is almost identical to the ridge version with the only difference being: `beta[j] ~ double_exponential(0, sigma_b);`

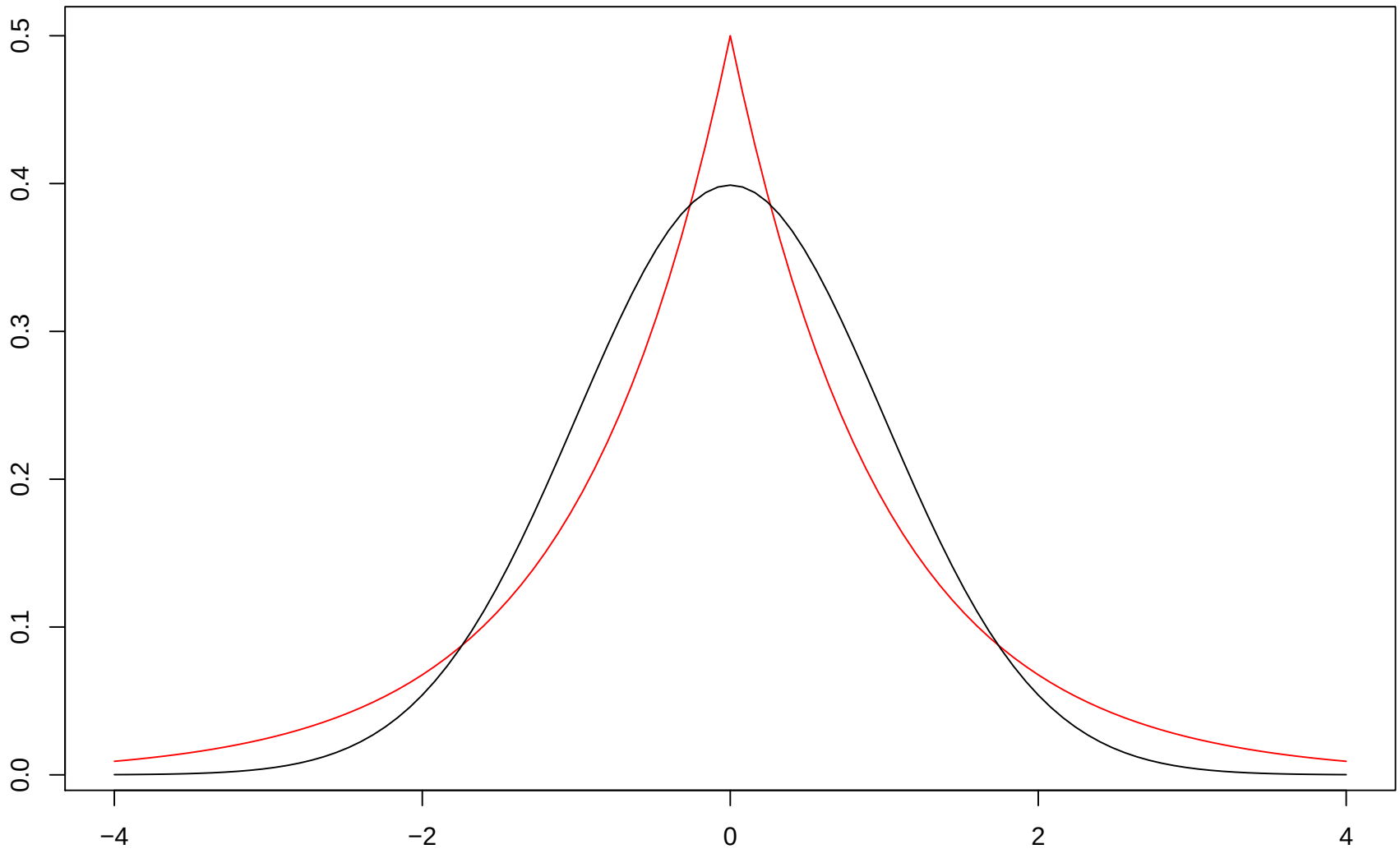
Stan output

```
stan_hist(stan_run, pars = c('beta','sigma_b'), bins = 30)
```



Comparing ridge and lasso

```
library(ExtDist)
curve(dLaplace, from = -4, to = 4, col = 'red', ylab = '')
curve(dnorm, from = -4, to = 4, add = TRUE)
```



A third alternative: Horseshoe

- The *horseshoe* prior aims to shrink even more than the Lasso:

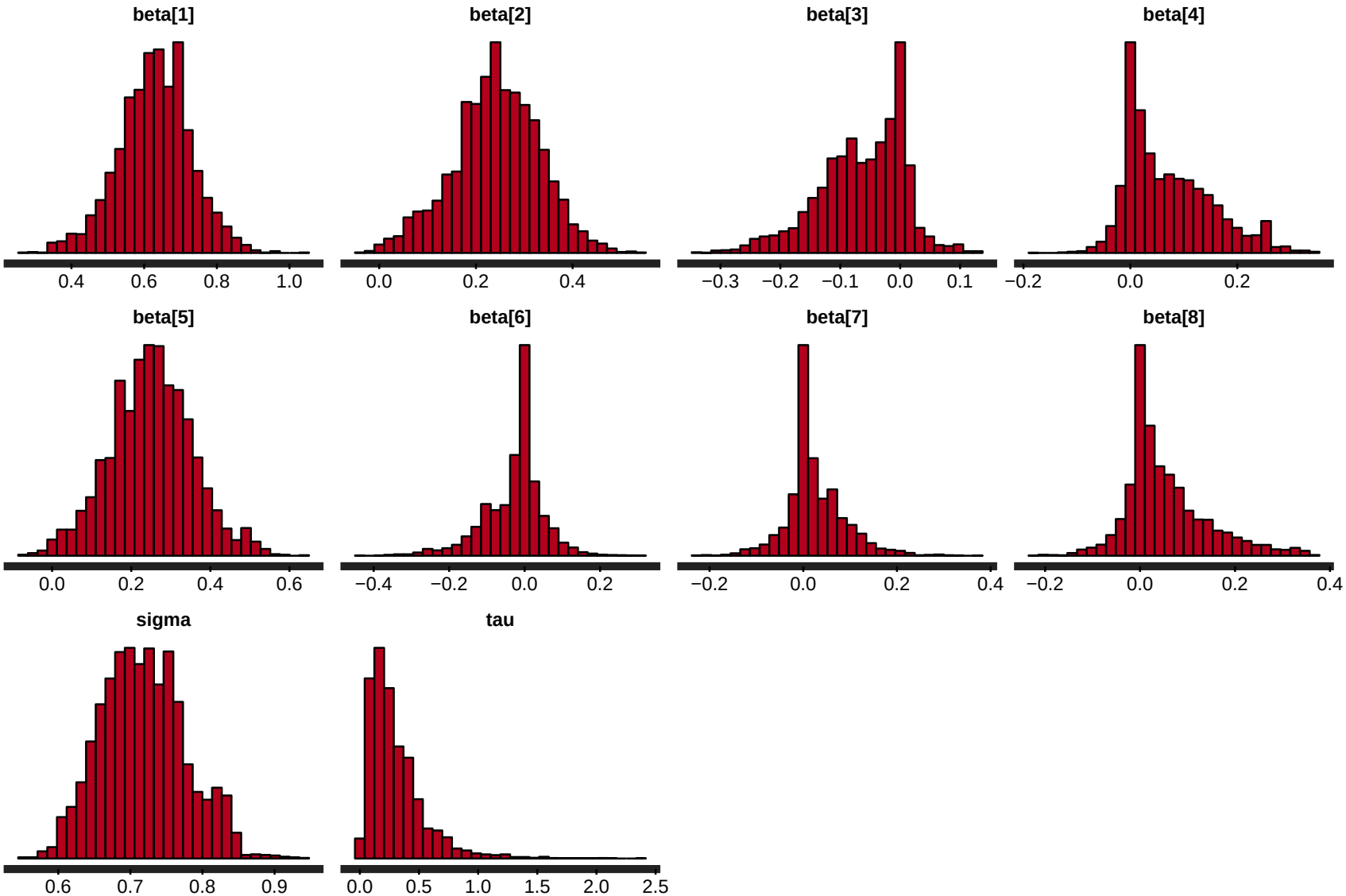
$$\beta_j \sim N(0, \lambda_j^2), \lambda_j \sim Cauchy^+(0, \tau\eta_j),$$

$$\eta_j \sim U(0,1), \tau \sim Cauchy^+(0,1)$$

  where $Cauchy^+$ are the half-Cauchy distributions we have been using already
- Whilst this looks complicated it's not too hard to simulate from this distribution to see how it works
- The name comes from the plot of the shrinkage behaviour which looks a bit like a horseshoe
- The plot of the density looks even more spiked than that of the Lasso

Stan output

```
stan_hist(stan_run, pars = c('beta','sigma','tau'), bins =
```



Final notes on Shrinkage

- If we're most concerned with predictive performance we can judge these using cross-validation, or compare them using DIC (or WAIC with Stan)
- The prostate data set has a column containing a training/test split so we could compare by fitting to the training and looking at performance on the test
- There is lots of current research on these topics and new shrinkage priors with supposedly better theoretical properties are appearing all the time

The spike and slab approach

- None of these methods actually sets the parameter values to zero. They just shrink the values towards zero
- The spike and slab approach however, does exactly this:

$$\beta_j = I_j \gamma_j, I_j \sim Binom(1, q_j), \gamma_j \sim N(0, \sigma_b)$$

- I_j here is the key parameter, a binary variable that either includes that variable in the model or removes it
- q_j is the probability of variable j being in the model. Often $q_j \sim U(0, 1)$ but more informative priors can be used
- Unfortunately you cannot fit this model in Stan, as it does not allow for discrete parameters I_j. You can fit it in JAGS though!

Spike and slab example

```
library(R2jags)
jags_code = '
model{
  for (i in 1:N) {
    y[i] ~ dnorm(alpha + inprod(X[i,], beta), sigma^-2)
  }
  for (j in 1:p) {
    beta[j] <- I[j] * gamma[j]
    I[j] ~ dbin(q[j], 1)
    q[j] ~ dunif(0, 1)
    gamma[j] ~ dnorm(0, 10^-2)
  }
  alpha ~ dnorm(0, 10^-2)
  sigma ~ dt(0, 10, 1)T(0,)
}
'
```

Output

```
print(jags_run)
```

```
## Inference for Bugs model at "5", fit using jags,
##  3 chains, each with 2000 iterations (first 1000 discarded)
##  n.sims = 3000 iterations saved
##          mu.vect sd.vect    2.5%     25%     50%     75%   97.5%  Rhat
## I[1]       1.000   0.000   1.000   1.000   1.000   1.000   1.000 1.000
## I[2]       0.839   0.368   0.000   1.000   1.000   1.000   1.000 1.164
## I[3]       0.019   0.138   0.000   0.000   0.000   0.000   0.000 1.037
## I[4]       0.070   0.255   0.000   0.000   0.000   0.000   1.000 1.041
## I[5]       0.516   0.500   0.000   0.000   1.000   1.000   1.000 1.055
## I[6]       0.015   0.122   0.000   0.000   0.000   0.000   0.000 1.127
## I[7]       0.014   0.118   0.000   0.000   0.000   0.000   0.000 1.163
## I[8]       0.048   0.213   0.000   0.000   0.000   0.000   1.000 1.082
## deviance 215.055   6.947 205.283 208.795 215.284 219.081 230.196 1.074
##          n.eff
## I[1]         1
## I[2]        24
## I[3]       660
## I[4]       180
## I[5]        41
## I[6]       220
## I[7]       180
## I[8]       120
## deviance    40
##
## For each parameter, n.eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor (at convergence, Rhat=1).
##
## DIC info (using the rule, pD = var(deviance)/2)
## pD = 22.9 and DIC = 238.0
## DIC is an estimate of expected predictive error (lower deviance is better).
```

Some notes on spike and slab

- It's called spike and slab because you have a spike probability at 0 (represented by q_0) and then a slab representing the β value being non-zero
- It's often quite hard to get the model to converge as the parameters γ_j don't always appear in the model
- Playing with some of the priors here can be quite good fun

A ridiculous word: non-parametric

- You may have come across the term *non-parametric* in the context of hypothesis tests which make fewer distributional assumptions about the data.
- Bayesian statistics has a branch of work called Bayesian non-parametrics. This is a complete misnomer though - *all of these models contain many parameters!*
- The reason they are useful is that they do not impost rigid assumptions about the shape of the data or the relationships between parameters

Regression with basis functions

- A common alternative to standard linear regression is to use polynomial regression:

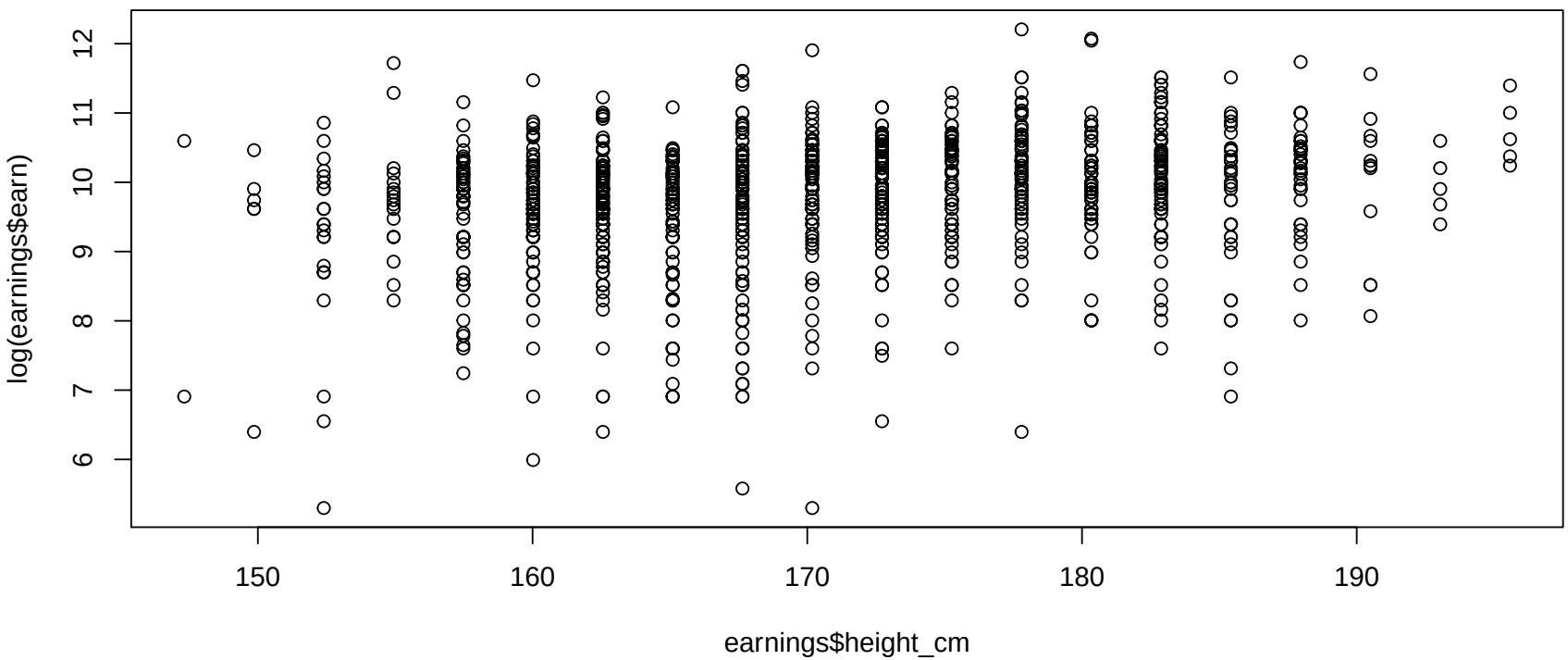
$$y_i \sim N(\beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \ldots + \beta_p x_i^p, \sigma^2)$$

- This is simple to fit in JAGS/Stan using much of the code we have already seen
- However, when p is large this becomes very unwieldy, numerically unstable, hard to converge, and has some odd properties

Example

- Let's go back to the earnings data and fit a non-linear regression model:

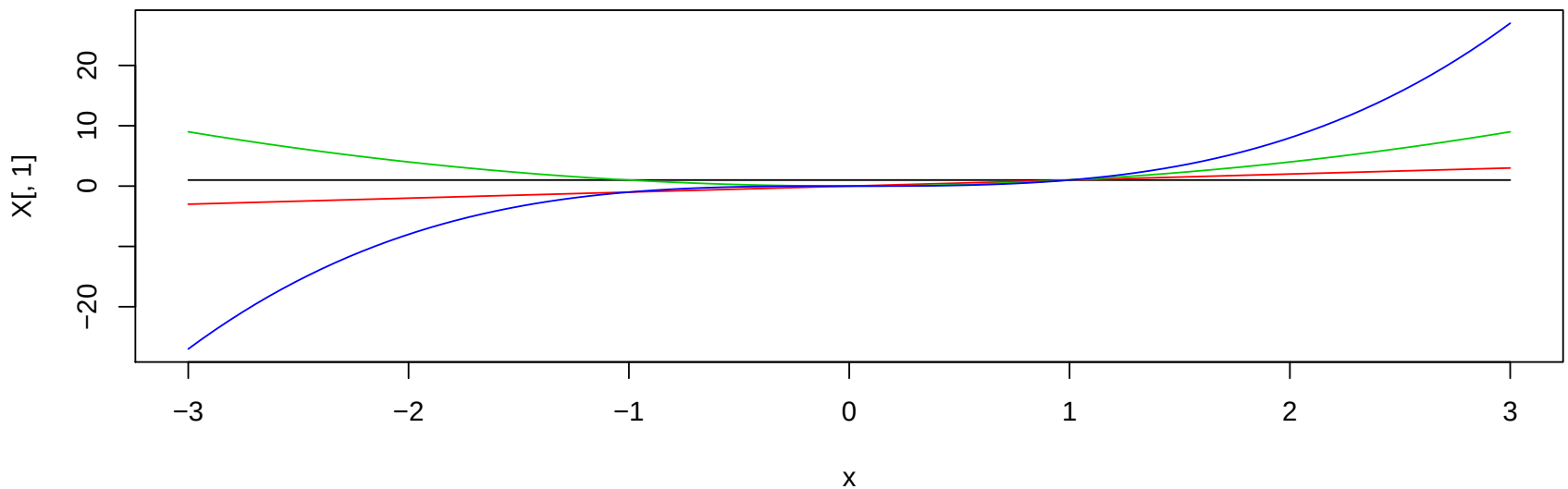
```
earnings = read.csv('../data/earnings.csv')
plot(earnings$height_cm, log(earnings$earn))
```



Basis functions

- When you have a matrix X used in a regression model, the columns are often called *basis functions*

```
x = seq(-3, 3, length = 100)
X = cbind(1, x, x^2, x^3)
plot(x,X[,1],ylim=range(X), type = 'l')
for(i in 2:ncol(X)) lines(x,X[,i], col = i)
```



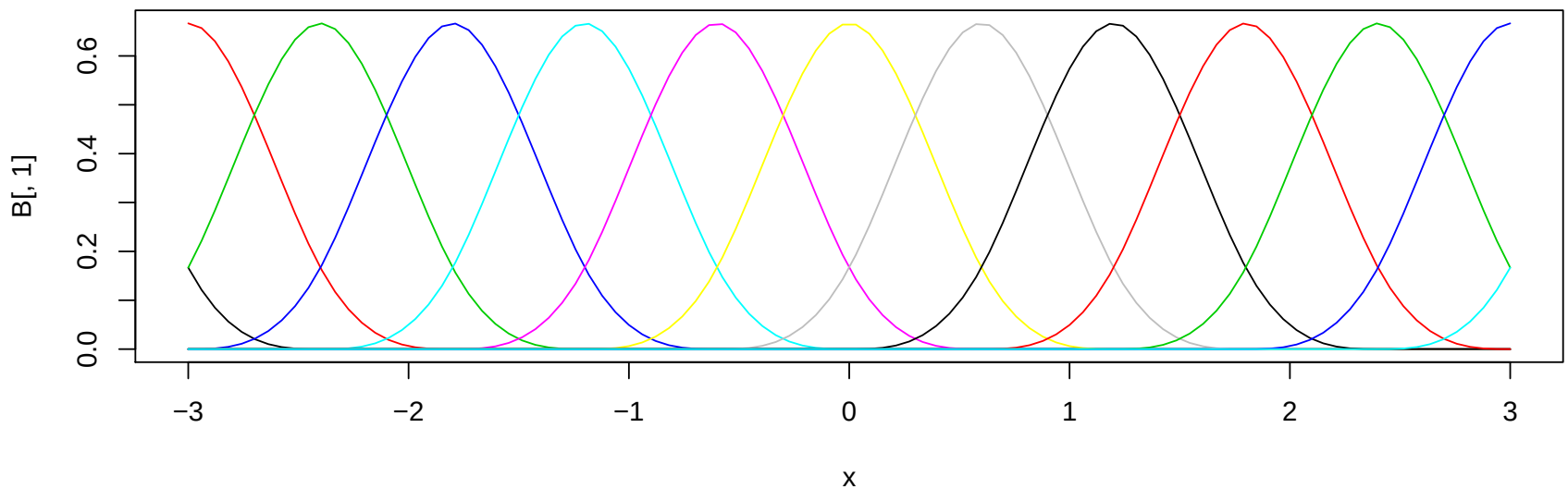
Creating new basis functions

- When we run a linear regression model using this matrix, we are multiplying each column by its associated β value, and forming an estimate of y
- As you can see, as we go up to powers of 3, 4, 5, etc, the values on the y-axis start to get really big
- Why not replace these *polynomial basis functions* with something better?

B-splines

- Here are some better, beautiful, basis functions called *B-spline* basis functions

```
B = bbase(x)
plot(x,B[,1],ylim=range(B), type = 'l')
for(i in 2:ncol(B)) lines(x,B[,i], col = i)
```



P-splines

- Now, instead of using a matrix X with polynomial basis functions, we create a matrix B of B-spline basis functions
- Each basis function gets its own weight β_j which determines the height of the curve
- A common way to make the curve smooth is make the β_j values similar to each other via a hierarchical model. Often:

$$\beta_j \sim N(\beta_{j-1}, \sigma_b^2)$$

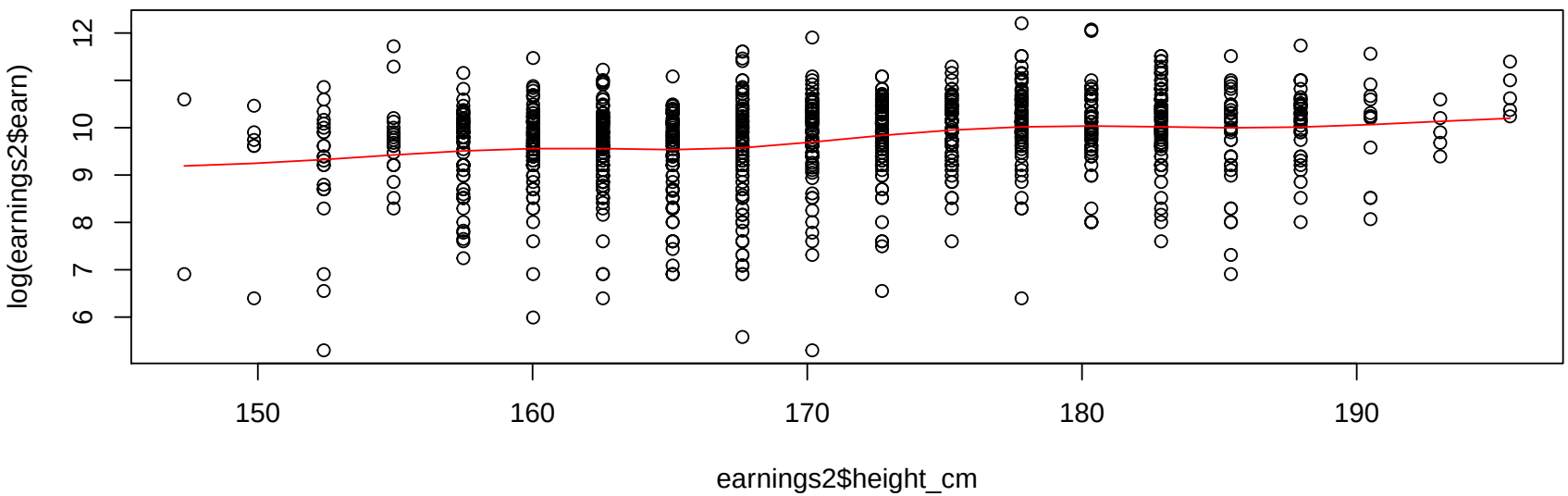
- This is known as a *penalised spline* or P-spline model since you are penalising the spline basis weights by making them similar to each other

Example code in Stan

```
stan_code = '
data {
  int<lower=0> N;
  int<lower=0> p;
  vector[N] y;
  matrix[N,p] B;
}
parameters {
  vector[p] beta;
  real<lower=0> sigma_b;
  real<lower=0> sigma;
}
model {
  y ~ normal(B * beta, sigma);
  beta[1] ~ normal(0, 10);
  for(j in 2:p)
    beta[j] ~ normal(beta[j-1], sigma_b);
  sigma_b ~ cauchy(0, 10);
  sigma ~ cauchy(0, 10);
}
'
```

Stan output

```
beta = apply(extract(stan_run, pars = 'beta')$beta,
             2,'mean')
plot(earnings2$height_cm, log(earnings2$earn))
lines(earnings2$height_cm, B%*%beta, col = 'red')
```



An alternative: Gaussian processes

- Whilst splines are great, they tend to use many parameters. You have to decide how many basis functions you want, and the advice is to use as many as possible
- The idea is that the penalty term induces the smoothness so it doesn't matter how many you used. But the model will be much slower to converge
- An alternative is to use a multivariate normal distribution, where you constrain the correlation between the parameters to be larger when they are close together. This is called a *Gaussian process*

A Gaussian process model:

- We write the Gaussian process model as:

$$y \sim MVN(\alpha, \Sigma + \sigma^2 I)$$

- Here α is a single parameter which represents the overall mean (but could also include regression covariates)
- Σ is a covariance matrix with terms:

$$\Sigma_{ij} = \tau^2 \exp\left(-\phi(x_i - x_j)^2\right)$$

  If you think about this, when x_i and x_j are close then you get a value of approximately τ^2. When they're far away you get a value of zero
- σ represents the residual standard deviation, as usual

A GP model in Stan

```
stan_code = '
data {
  int<lower=1> N;
  vector[N] x;
  vector[N] y;
}
transformed data {
  vector[N] alpha;
  for (i in 1:N) alpha[i] = 0;
}
parameters {
  real<lower=0> eta_sq;
  real<lower=0> inv_rho_sq;
  real<lower=0> sigma_sq;
}
transformed parameters {
  real<lower=0> rho_sq;
  rho_sq = inv(inv_rho_sq);
} model {
  matrix[N, N] Sigma;
  // off-diagonal elements
  for (i in 1:(N-1)) {
    for (j in (i+1):N) {
      Sigma[i, j] = eta_sq * exp(-rho_sq * pow(x[i] - x[j],2));
      Sigma[j, i] = Sigma[i, j];
    }
  }
  // diagonal elements
  for (k in 1:N)
    Sigma[k, k] = eta_sq + sigma_sq;
  eta_sq ~ cauchy(0, 5);
  inv_rho_sq ~ cauchy(0, 5);
  sigma_sq ~ cauchy(0, 5);
  y ~ multi_normal(alpha, Sigma);
}
```

Stan output

```
pred = apply(extract(stan_run, pars = 'y2')$y2,
             1, 'mean')
plot(earnings3$height_cm, log(earnings3$earn))
lines(earnings3$height_cm, pred, col = 'red')
```

Notes on GPs

- Whilst GPs have far fewer parameters than splines, they tend to be slower to fit because the calculation of the density for the multivariate normal involves a matrix inversion which is really slow
- There are lots of fun ways to fiddle with GP models, as you can change the function that controls the way the covariance decays, or add in extra information in the mean
- There is a very useful but quite fiddly formula that enables you to predict for new values of y from new values of x just like a regression

Summary

- We can put some clever priors on regression coefficients to induce shrinkage (or selection)
- We have covered the ridge, lasso, and horseshoe methods
- We have met some ways of running non-parametric regression models, including P-splines and Gaussian processes
- There is much more to be said on these methods, but they will not be relevant to everyone here

- Good luck using Bayesian models in your research!