Lab 3

Alex Ziyu Jiang

Today's agenda

- We will go through a quick review of the linear regression model under Bayesian paradigm
- We will introduce the spline models as a technique to model nonlinearity in your regression function.

Part 1: Linear regression review

The linear regression setting (at least, one of the most classical settings) is based on the framework of Gaussian distribution. Say Y is the **observed data** that we are interested in and want to build a statistical model to describe its distribution. A logical choice would be to use the Gaussian distribution, that brings two parameters (that are unobservable), the mean μ and the standard deviation σ. So this gives

$$Y_i \sim \mathcal{N}\left(\mu, \sigma^2\right), i = 1, ..., N$$

Since this is a Bayesian models so we need to place priors on the unknown parameters for the model to be complete. The unknown parameters in this model are μ and σ. One thing we can do is to place an independent prior on μ and σ: $\Pr(\mu, \sigma) = \Pr(\mu)\Pr(\sigma)$. Furthermore, since μ is also on the real line like Y_i, we can place a normal prior on it. Finally, notice that the standard deviation is positive, so we should place a prior that only has support on positive values. A uniform distribution between 0 and 50 could be a choice. (There are other alternatives to this prior introduced in class, say an exponential prior.)

$$\begin{aligned} &Y_i \sim \mathcal{N}\left(\mu, \sigma^2\right), i = 1, ..., N \\ &\mu \sim \mathcal{N}(0, 10) \\ &\sigma \sim \text{Uniform}(0, 50) \end{aligned}$$

Okay so far this is only a normal distribution which is a little far away from regression, in which we usually study the **statistical associations** between variables. Say now in addition to the outcome variable $\{Y_i\}$, we also have a predictor variable (that is also already observable) $\{X_i\}$ for all of the observations $i = 1, ..., n$. We want to study the association between Y_i and X_i by incorporating the latter into our model.

The way to do this is to re-express the mean of the Gaussian distribution μ (here because the expected mean for each observation will be different, so we use μ_i instead of μ in the model) as a linear function of the predictior variable X_i:

$$\mu_i = \alpha + \beta X_i$$

Here we can see the introduced parameters α and β collectively defined a deterministic relationship between X_i, our predictor variable and μ_i, the expected outcome given X_i. We see that if we know α, β and we already observed X_i, we can directly computed μ_i, so there is no need to model μ_i again in this model. We can interpret them in the following way:

- α is the expected height when $X_i = 0$.
- β is the change in expected outcome, when X_i changes by 1 unit.

- In the textbook they used an alternative representation $\mu_i = \alpha + \beta(X_i - \bar{X})$. This slightly changes the interpretation of α, but it does not affect β.

Thus the overall model is

$$
\begin{aligned}
Y_i &\sim \mathcal{N}(\mu, \sigma^2) \\
\mu_i &= \alpha + \beta X_i \\
\beta &\sim \mathcal{N}(0, 10) \\
\alpha &\sim \mathcal{N}(0, 10) \\
\sigma &\sim \text{Uniform}(0, 50)
\end{aligned}
$$

In class we've introduced how to code this in R so you can refer to the class materialas for more details.

Part 2: Curvilinear relationships: Polynomial regression and Splines

So far we assumed that the expected outcome μ_i and the predictor variable X_i has a linear relationship: $\mu_i = \alpha + \beta X_i$. But we can also generalize this to broader settings, where the relationship is nonlinear (i.e. a 'curve'). There are two ways to do it: polynomial regression and spline models. We will look at them respectively:

(1) Polynomial regression

An easy way to build curve relationships is to take powers (i.e. squares and cubes) from a certain variable. Now we will be looking at a dataset from the package `rethinking` that included height and weight data for both adult and non-adult participants. So here the height is the outcome (h_i) and the weight is the predictor variable (X_i). Because the dataset adult and non-adult participants, the scatterplot indicates a very nonlinear relationship:

```
library(rethinking)
```

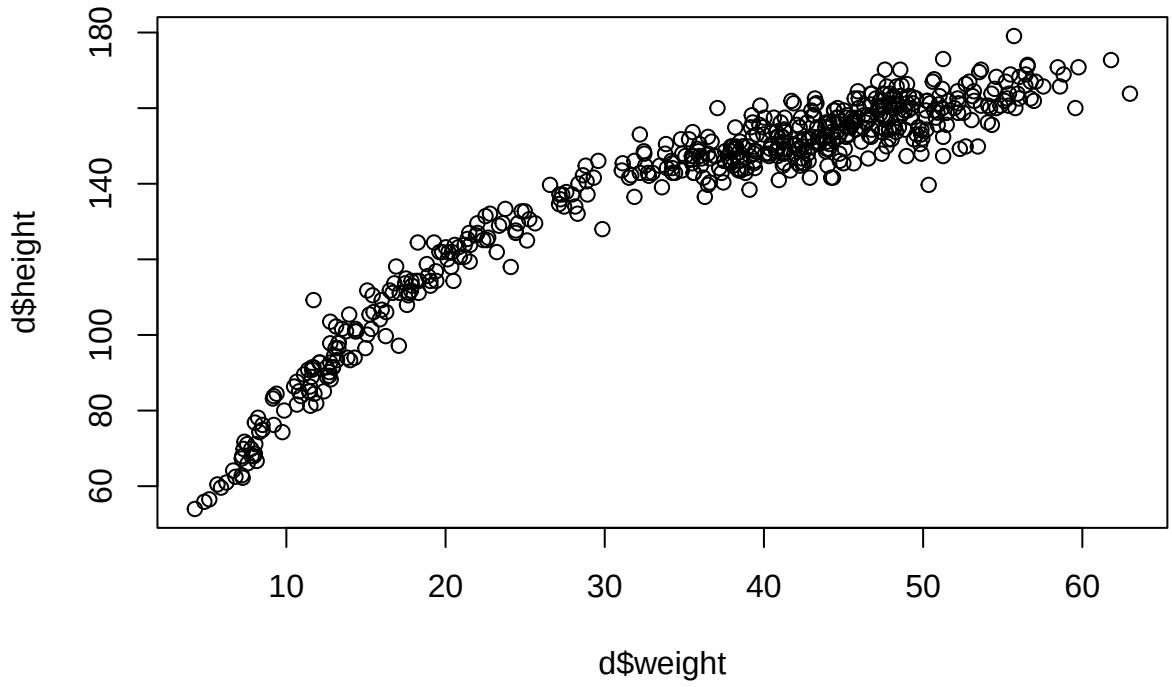
```
## Loading required package: rstan

## Loading required package: StanHeaders

## Loading required package: ggplot2

## rstan (Version 2.21.3, GitRev: 2e1f913d3ca3)

## For execution on a local, multicore CPU with excess RAM we recommend calling
## options(mc.cores = parallel::detectCores()).
## To avoid recompilation of unchanged Stan programs, we recommend calling
## rstan_options(auto_write = TRUE)

## Loading required package: cmdstanr

## This is cmdstanr version 0.5.0

## - CmdStanR documentation and vignettes: mc-stan.org/cmdstanr

## - CmdStan path: /Users/alexziyujiang/.cmdstan/cmdstan-2.29.2

## - CmdStan version: 2.29.2

## Loading required package: parallel

## rethinking (Version 2.21)
```

```
##
## Attaching package: 'rethinking'
```

```
## The following object is masked from 'package:rstan':
##
##     stan
```

```
## The following object is masked from 'package:stats':
##
##     rstudent
```

```
data(Howell1)
d <- Howell1
plot(d$height ~ d$weight) # plot height against weight
```




```
# a strong nonlinear relationship
```

Looking at the data, it makes sense that we want to describe the nonlinear relationship between the height and weight using a parabola. The mean can be broken down into two parts: the linear part $\alpha + \beta_1 x_i$ and the curvilinear part $\beta_2 x_i^2$; essentially, β_2 describes the curvature of the nonlinear relationship between the expected height and weight.

$$\mu_i = \alpha + \beta_1 x_i + \beta_2 x_i^2$$

According to the textbook the whole model has the following form:

$$\begin{aligned}
h_i &\sim \text{Normal}\left(\mu_i, \sigma\right) \\
\mu_i &= \alpha + \beta_1 x_i + \beta_2 x_i^2 \\
\alpha &\sim \text{Normal}(178, 20) \\
\beta_1 &\sim \log - \text{Normal}(0, 1) \\
\beta_2 &\sim \text{Normal}(0, 1) \\
\sigma &\sim \text{Uniform}(0, 50)
\end{aligned}$$

One thing you may notice is that the β_1 coefficient has a log-normal prior. This is due to the general belief (which, is also indicated by the scatter plot) that average height increases with average weight (at least up to a point). This gives us prior belief that the 'slope' coefficient should at least be positive, which encourages us to place a prior that has positive support. Log-normal distribution seems a good fit for β_1 (because otherwise we would have fitted a normal distribution, here we are just conduct a log-transformation such that range be stricted onto the positive real line).

We fit this model in R:

```
# Important! You need to standardize the data
d$weight_s <- ( d$weight - mean(d$weight) )/sd(d$weight)
d$weight_s2 <- d$weight_s^2
m4.5 <- quap(alist(
    height ~ dnorm( mu , sigma ) ,
    mu <- a + b1*weight_s + b2*weight_s2 ,
    a ~ dnorm( 178 , 20 ) , # sd = 20
    b1 ~ dlnorm( 0 , 1 ) , # log(b1) ~ dnorm(0,1)
    b2 ~ dnorm( 0 , 1 ) ,
    sigma ~ dunif( 0 , 50 )
), data=d )
precis( m4.5 )
```

```
##             mean        sd       5.5%      94.5%
## a     146.057626 0.3689751 145.467933 146.647319
## b1     21.732934 0.2888890  21.271233  22.194634
## b2     -7.803542 0.2741823  -8.241738  -7.365346
## sigma   5.774484 0.1764656   5.492458   6.056510
```

One important thing to gleam from the code is that **you need to standardize your predicted variable first** before you fit the model. This is very important especially in polynomial models, because x^2, x^3 can be outrageously large if x is large, and sometimes that makes your coefficients numerically very hard to interpret. Also, have coefficients on a more 'normal' scale make the computation more stable, and it is more convenient to place priors on. You can try this with your example by replacing \texttt{weight_s} with `weight` in your model.

Generally speaking, it is encouraged to center the variables first before you run a Bayesian regression model. Mathematically it wouldn't make a difference, but it improves the efficiency of the markov chain so you need less samples to get the desired results. (See more information here if you're interested: https://stats.stackexchange.com/questions/273123/mcmc-bayesian-approach-centering-and-standardizing)

We now take a stab at interpreting these coefficients:

- a ells us the expected value of height when weight is at its mean value
- β_1 and β_2 parameters are the linear and square components of the curve, respectively

```
# generate a series of weight values to do predictions on
weight.seq <- seq( from=-2.2 , to=2 , length.out=30 )
# data used for prediction
pred_dat <- list( weight_s=weight.seq , weight_s2=weight.seq^2 )
# sample posterior draws of the expected outcome for each (standardized) weight value
mu <- link( m4.5 , data=pred_dat )
# calculate posterior mean
mu.mean <- apply( mu , 2 , mean )
# posterior credible intervals
mu.PI <- apply( mu , 2 , PI , prob=0.89 )
# sample posterior draws of the predicted outcomes
sim.height <- sim( m4.5 , data=pred_dat )
# posterior credible intervals
```

```
height.PI <- apply( sim.height , 2 , PI , prob=0.89 )
```

We can plot the result based on our posterior draws. The blue dots are the actual observations from the dataset. The black solid line is the posterior mean of the expected height, its 89% credible interval is given by the dark band, and the 89% credible interval for the predicted outcome is given by the light band. Empirically for an appropriate model we would want about 89% of the observed data to fall into the light band, and it does seem like our model is doing a fairly good job, especially compared to the result based on a linear model.

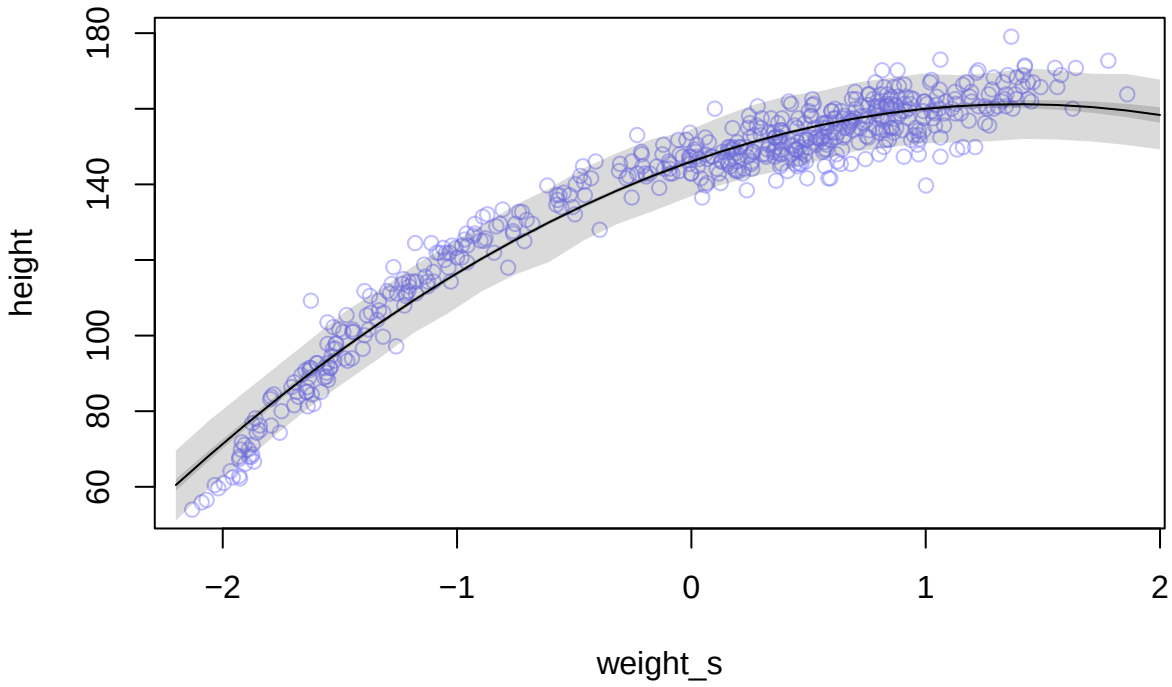
What if we think that this current model is not 'flexible' enough? We can add an extra cubic term $\beta_3 x_i^3$ into the regression function: $\mu_i = \alpha + \beta_1 x_i + \beta_2 x_i^2 + \beta_3 x_i^3$.

$$
\begin{aligned}
h_i &\sim \text{Normal}\left(\mu_i, \sigma\right) \\
\mu_i &= \alpha + \beta_1 x_i + \beta_2 x_i^2 + \beta_3 x_i^3 \\
\alpha &\sim \text{Normal}(178, 20) \\
\beta_1 &\sim \log - \text{Normal}(0, 1) \\
\beta_2 &\sim \text{Normal}(0, 1) \\
\beta_3 &\sim \text{Normal}(0, 1) \\
\sigma &\sim \text{Uniform}(0, 50)
\end{aligned}
$$

```
d$weight_s3 <- d$weight_s^3
m4.6 <- quap(
    alist(
        height ~ dnorm( mu , sigma ) ,
        mu <- a + b1*weight_s + b2*weight_s2 + b3*weight_s3 ,
        a ~ dnorm( 178 , 20 ) ,
        b1 ~ dlnorm( 0 , 1 ) ,
        b2 ~ dnorm( 0 , 10 ) ,
        b3 ~ dnorm( 0 , 10 ) ,
        sigma ~ dunif( 0 , 50 )
        ), data=d )
```

Ideally, we could make this model even more flexible by adding terms of higher order. But on the same time we need to consider whether doing this makes a model better at all. There are two points worth mentioning: - First, a better fit to the sample might not actually be a better model - The model contains no biological information, so we aren't learning anything about any causal relationship between height and weight.

```
plot( height ~ weight_s , d , col=col.alpha(rangi2,0.5) )
lines( weight.seq , mu.mean )
shade( mu.PI , weight.seq )
shade( height.PI , weight.seq )
```



(2) Splines

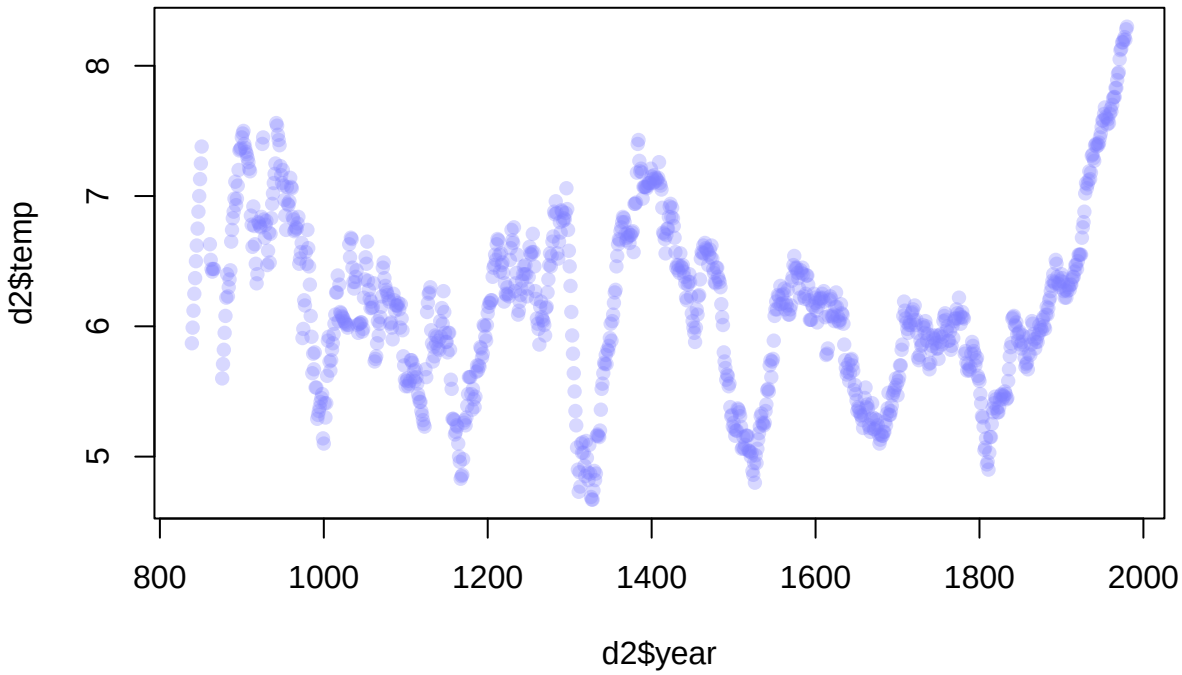
An alternative way to model this is to use the spline functions. **Spline** is a smooth function built out of smaller, component functions. The example we are using here is called **B-spline**, which stands for the basis spline functions: essentially, B-splines build up functions that are very wiggly from smaller component functions that are less wiggly. **By taking mixtures of some simple functions, we get the complicated, wiggly function that we use to model the nonlinear relationship**. We will see how to do this in a minute.

To get a better sense, lets first look at an example when the outcome variable you want to model has a very wiggly form. The \texttt{cherry_blossoms} dataset from the course package docuemnts the cherry blossom dates (usually in March) from year 839 to year 1980. We can see that the records in these years varies dramatically and it would be a little impractical to model this using a polynomial regression model:

```
#load the course package
library(rethinking)
data("cherry_blossoms")
d <- cherry_blossoms
precis(d)
```

```
##                  mean          sd      5.5%      94.5%       histogram
## year       1408.000000 350.8845964 867.77000 1948.23000
## doy         104.540508   6.4070362  94.43000  115.00000
## temp          6.141886   0.6636479   5.15000    7.29470
## temp_upper    7.185151   0.9929206   5.89765    8.90235
## temp_lower    5.098941   0.8503496   3.78765    6.37000
```

```
d2<-d[complete.cases(d$temp),] #complete cases on temp, remove NA
plot(d2$year,d2$temp,col=col.alpha(rangi2,0.3),pch=16)# march temperature over a very long timespan
```



How does spline functions work?

[Draw an example here]

Knots and Basis functions The first step in modeling splines is to choose the range in which you want to model the wiggly function and the number of **knots** in that range. Essentially, knots determine how many basis functions you have in this range. **If you have K knots, you split the range into (usually identical) $(K-1)$ parts, and each knot will correspond to a few basis function. Given the range and the number of knots, the number of basis functions is known and all the basis functions will be known and computable. For example, if we have 5 knots in the range $[0,1]$ for a linear spline model, for $\forall x \in [0,1]$, the Basis functions $B_1(x), ..., B_5(x)$ can be evaluated at any point.

An important observation you may get from this model is that if you are closer to a knot, you tend to get higher values in the Basis functions related to that knot, on the other hand if you move away from it, those Basis functions tends to shrink to zero. This corresponds to the intuition that **we use spline models to do local approximations**. You can think of this in the way that the Basis function that are close to your data has larger values, and therefore they have a heavier influence in determining the regression function near this point. As for the basis functions away from the point, a lot of them are zero so they wouldn't even affect the estimation result.

Finally, we can determine how 'smooth' these basis function can be by setting the **polynomial degree**. Essentially, if the polynomial degree is k, $(k+1)$ basis functions will collectively determine a given point (barring some special cases in the boundary). Higher orders lead to basis functions that are more 'smooth'.

Weights The next step in building our spline functions is to determine the weights for each of the basis functions. As our expected outcome can be expressed as a linear combination of the basis functions, we can re-express the mean function $\mu(x)$ as follows (similar to what we did before in the polynomial regression example, we introduce the parameter α as an intercept term):

$$\mu(x) = \alpha + \sum_{k=1}^{K} w_k B_k(x)$$

Apparently, we do not have data for every x on this range. To fit this model, we evaluate the Basis functions $\{B_k(x)\}_{k=1}^K$ for the each of the predictor variables $x_1, ..., x_n$. We define $B_{i,k} := B_k(x_i)$. Thus we have

$$\mu_i = \alpha + \sum_{k=1}^{K} w_k B_{i,k} = \alpha + \mathbf{B}_{n\times K} w_{K\times 1}$$

(On a side note, we can rephrase this in the matrix form and put all the B's in a matrix. The numbers of rows will be the number of observations/years, and the columns will be the number of basis functions. The weights for the K basis functions are stored into the vector.)

You'll discover that the equation that we have above is very similar to the regression function we had in the polynomial regression model. The only difference is that instead of taking the squares or cubes of x_i, we transform them through the basis functions to get $B_{i1}, ..., B_{ik}$ (these are called 'synthetic variables' in the textbook).

Fitting the model

At this point, we see that this model is essentially a linear regression model, if you treat each column of the $\mathbf{B}$ matrix as a separate predictor variable. We let T_i be the temperature observed

$$\begin{aligned}
T_i &\sim \text{Normal}\,(\mu_i, \sigma) \\
\mu_i &= \alpha + \sum_{k=1}^{K} w_k B_{k,i} \\
\alpha &\sim \text{Normal}(6, 10) \\
w_j &\sim \text{Normal}(0, 1) \\
\sigma &\sim \text{Exponential}(1)
\end{aligned}$$

We see that this model is very similar to the polynomial regression model.

We then see how to fit this model in R. First we need to do some preprocessing of the data to calculate the B matrix. To do this we will learn how to properly define the spline functions. We pull out the complete cases in the dataset due to data missingness, and choose 15 knots over the range $[839, 1980]$ with equal spaces between these knots. Considering the data is not evenly spaced due to missingness, we can also choose knots so that there are equal number of data points between each knot. This is done by the `quantile()` function:

```
d2 <- d[ complete.cases(d$temp) , ] # complete cases on temp
num_knots <- 15
knot_list <- quantile( d2$year , probs=seq(0,1,length.out=num_knots) )
knot_list
```

```
##        0% 7.142857% 14.28571% 21.42857% 28.57143% 35.71429% 42.85714%       50%
##  839.0000  937.2143 1017.4286 1097.6429 1177.8571 1258.0714 1338.2857 1418.5000
## 57.14286% 64.28571% 71.42857% 78.57143% 85.71429% 92.85714%      100%
## 1498.7143 1578.9286 1659.1429 1739.3571 1819.5714 1899.7857 1980.0000
```

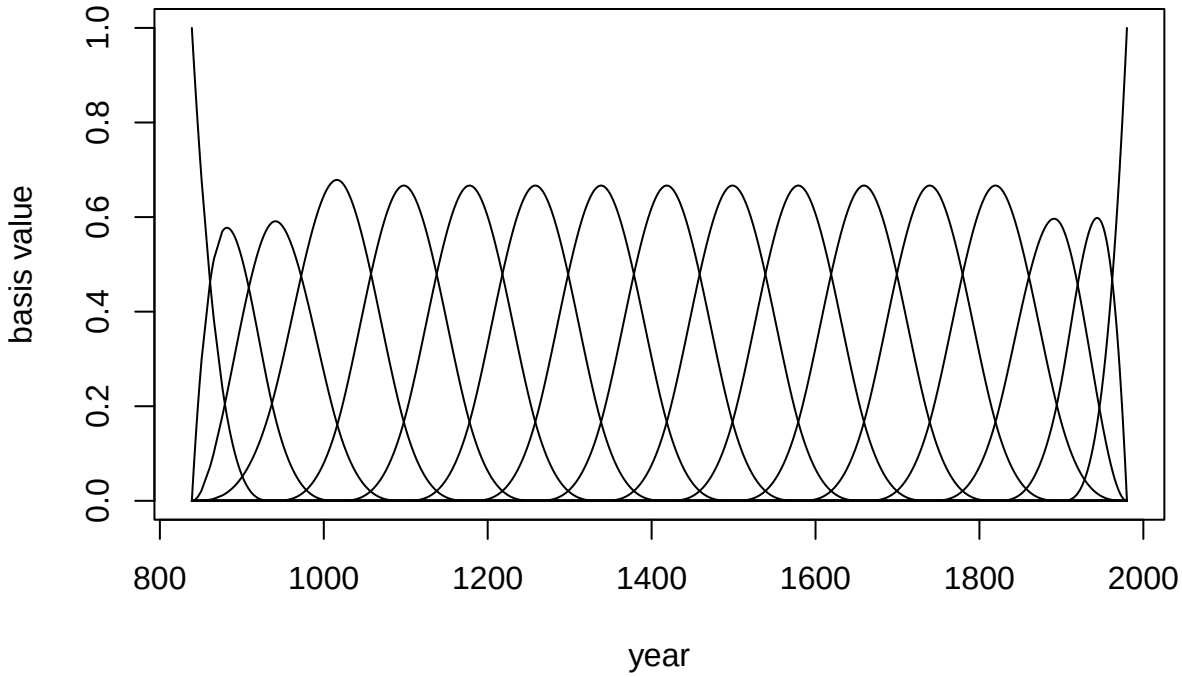
We then learn how to compute the basis functions using the `splines` package. We want an intercept term α in our regression function, so we set `intercept=TRUE`, and we set the polynomial degree to be 3. (Note that the syntax is a little different for `bs()` since the end points are set as default knots, so if)

```
library(splines)
B <- bs(d2$year,
    knots=knot_list[-c(1,num_knots)] ,
    degree=3 , intercept=TRUE )
dim(B) # check dimensions of this matrix
```

```
## [1] 1124   17
```

We can plot these 17 basis functions over the range and see how they look:

```
plot( NULL , xlim=range(d2$year) , ylim=c(0,1) , xlab="year" , ylab="basis value")
for ( i in 1:ncol(B) ) lines( d2$year , B[,i] )
```



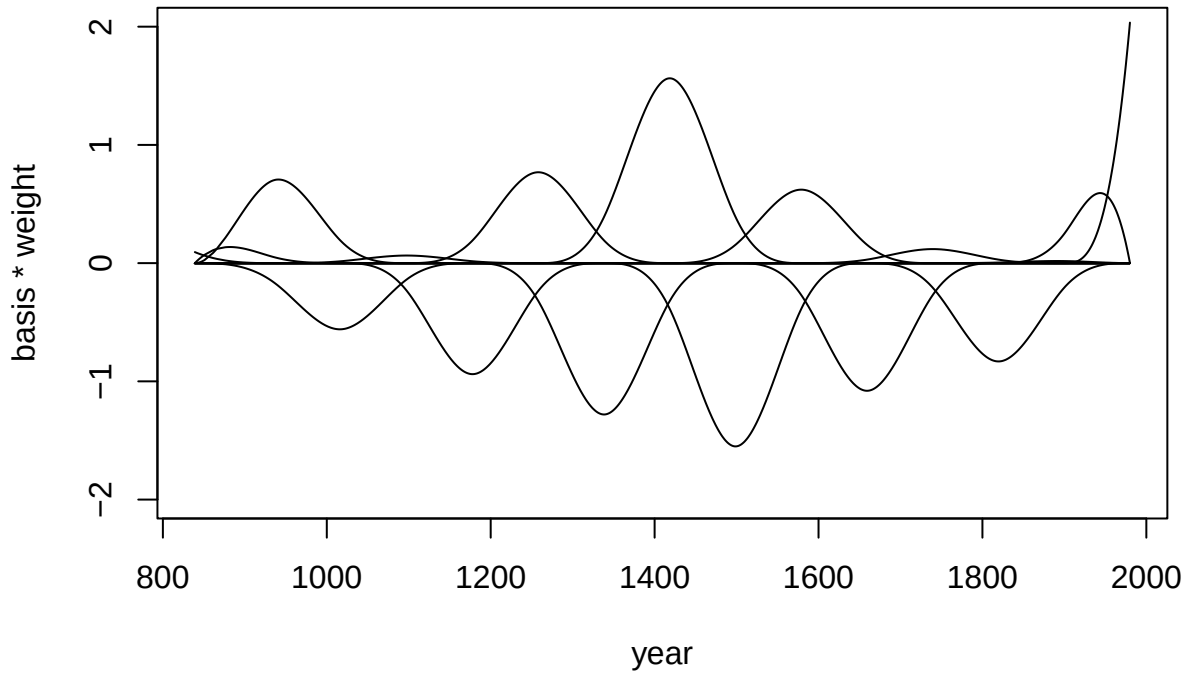
Finally, we fit this model using R, given that we have all the pieces.

```
m4.7 <- quap(
    alist(
        T ~ dnorm( mu , sigma ) ,
        mu <- a + B %*% w , # matrix expressions of B and w
        a ~ dnorm(6,10),
        w ~ dnorm(0,1),
        sigma ~ dexp(1)
    ),
    data=list( T=d2$temp , B=B ) ,
    start=list( w=rep( 0 , ncol(B)))) # we havent defined w before, so we let quap() know that w is a v
```

The results

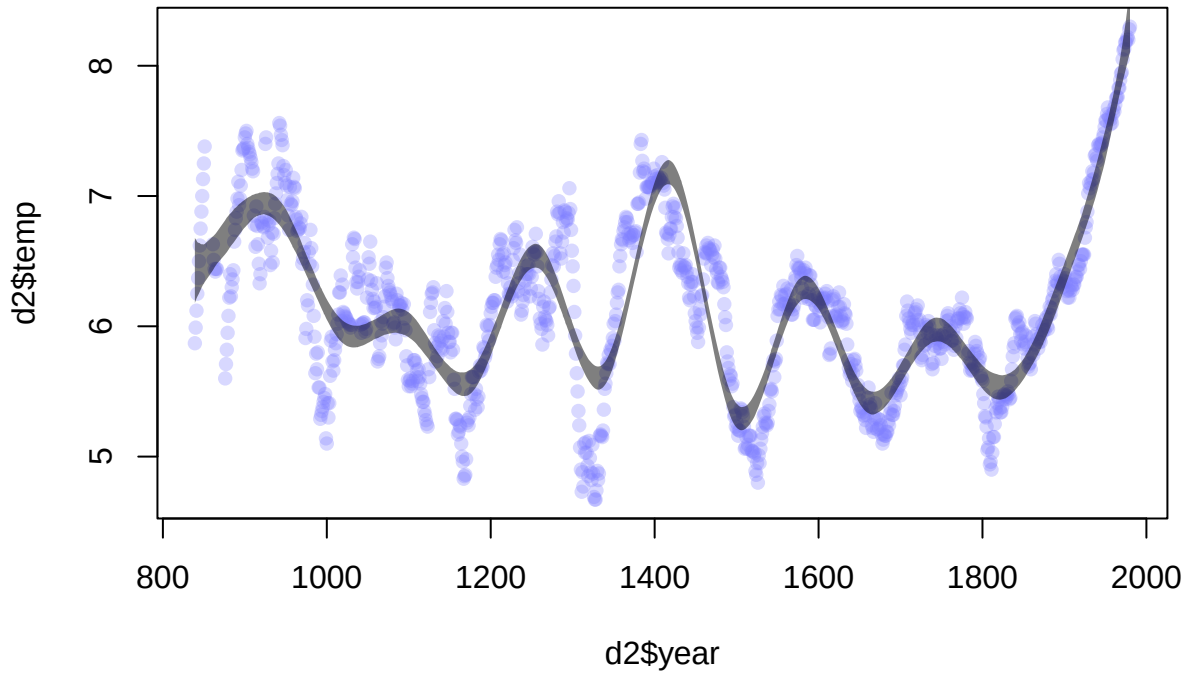
Finally, we can visualize the results. We can see the posterior mean of the weighted basis functions, $w_k B_k(x)$:

```
post <- extract.samples(m4.7)
w <- apply( post$w , 2 , mean )
plot( NULL , xlim=range(d2$year) , ylim=c(-2,2) ,
    xlab="year" , ylab="basis * weight" )
for ( i in 1:ncol(B) ) lines( d2$year , w[i]*B[,i] )
```



And finally we plot the 97% posterior credible interval for the expected temperature, given the temperature observed:

```
mu <- link( m4.7 )
mu_PI <- apply(mu,2,PI,0.97)
plot( d2$year , d2$temp , col=col.alpha(rangi2,0.3) , pch=16 )
shade( mu_PI , d2$year , col=col.alpha("black",0.5) )
```



Some final comments

- We see that the expected outcome given by the spline models is very 'wiggly' and it is pretty close to the actual data. You can even customize your results by tweaking the number of knots and polynomial orders in the model.
- Again, high polynomial order and more knots may lead to more 'flexible' results but that's not necessarily a good thing, as you may risk overfitting the data and end up learning the noise – more of this in future lectures.

Summary

- We start our model with a Gaussian distribution with the same mean for each observation: $\mu_i = \mu$
- Given a predictor variable X_i, we can build a **linear relationship** with the predicted mean and the predictor: $\mu_i = \alpha + \beta x_i$
- We can add more flexibility to the previous model by adding **polynomial order terms**: $\mu_i = \alpha + \beta_1 x_i + \beta_2 x_i^2$
- Finally, we can use **splines** to model really wiggly data: $\mu_i = \alpha + \sum_{k=1}^{K} w_k B_{i,k}(x_i)$