Non-Linear Regression

36-600

The Setting

- Assume that we have n data tuples,

$$(x_i, y_i, e_i)\ ,$$

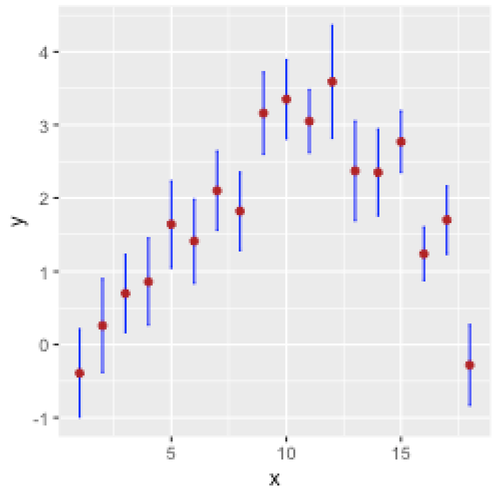
  - x_i is the x-coordinate for datum i
  - y_i is the y-coordinate for datum i
  - e_i is the estimated uncertainty for datum i
- Our goal: to draw a "curve" through the points
  - *but wait, isn't that just regression?*
  - it is!…it is not necessarily *linear* regression, but it is a (weighted) regression
- Unlike a typical linear regression setting, we have estimated uncertainties here (the e_i's)
  - a data point with a smaller uncertainty should have more weight in the fitting process than one with a larger uncertainty
  - *inverse-variance* weighting is often used:

$$w_i = \frac{1}{e_i^2}$$

Example

- Here is a dataset that you have been given:

x	1.00	2.00	3.00	4.00	5.00	6.00	7.00	8.00	9.00	10.00	11.00	12.00	13.00	14.00	15.00	16.00	17.00	18.00
y	-0.39	0.26	0.70	0.86	1.64	1.41	2.10	1.82	3.16	3.35	3.05	3.59	2.37	2.35	2.77	1.24	1.70	-0.28
e	0.60	0.64	0.53	0.59	0.59	0.57	0.54	0.53	0.56	0.54	0.43	0.77	0.68	0.59	0.42	0.36	0.46	0.55



- Your advisor tells you to "fit a model" to these data, and you dutifully agree
  - (note that for simplicity, we are not splitting the data into training and tests sets)

Global Polynomial Regression

- It is readily apparent that a simple linear regression model is *not* a good representation of the data-generating process in this particular instance
- A first variant on simple linear regression that we can examine is (global) polynomial regression:

$$Y|x = \beta_0 + \beta_1 x + \cdots + \beta_d x^d + \epsilon$$

Global Polynomial Regression

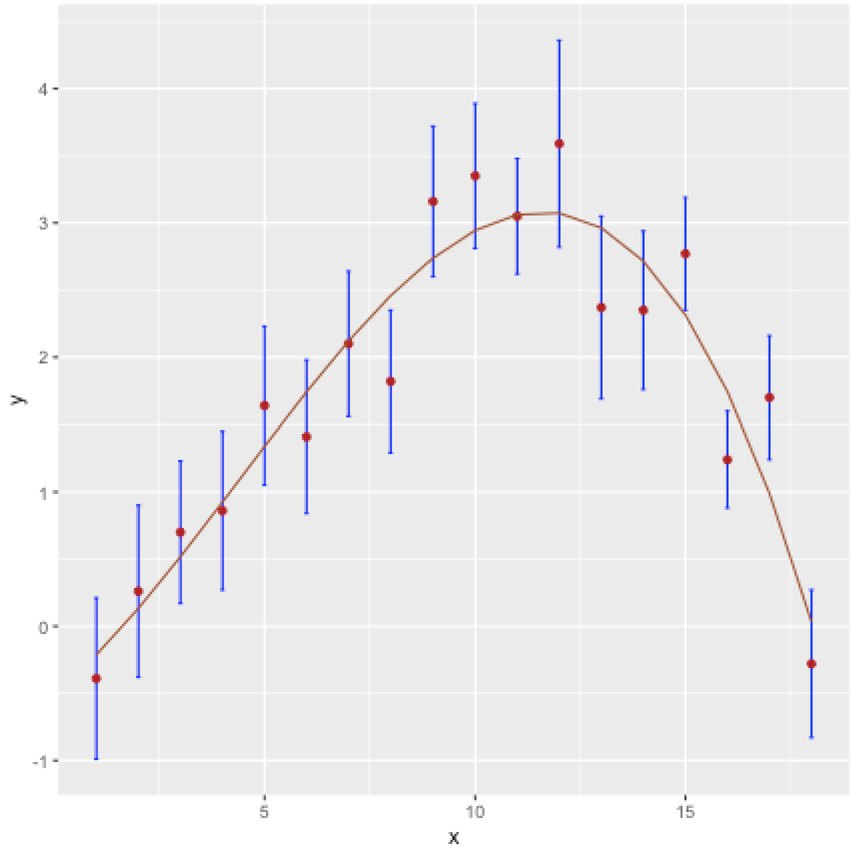
- Let's try $d = 3$ (while incorporating the provided uncertainties)

```
pr.out  <- lm(y~poly(x,3,raw=TRUE),data=df,
              weights=1/(e^2))
pr.pred <- predict(pr.out)
mean((y-pr.pred)^2)
```

```
## [1] 0.1585677
```

```
data.frame("Estimate"=
             summary(pr.out)$coefficients[,1])
```

```
##                             Estimate
## (Intercept)             -0.503930198
## poly(x, 3, raw = TRUE)1  0.258283863
## poly(x, 3, raw = TRUE)2  0.035351480
## poly(x, 3, raw = TRUE)3 -0.002669271
```



Regression Splines

- "Global" models have what can be viewed as an inherent disadvantage: because they are defined for all x, then can be insufficiently flexible
- An example of a "local" model is a spline model
  - the range of values of x is divided into K non-overlapping segments
  - the boundaries between segments are dubbed *knots*
  - as an example, we can express a quadratic polynomial model with one knot as

$$Y_i|x_i = \begin{cases} \beta_{0,1} + \beta_{1,1}x_i + \beta_{2,1}x_i^2 + \epsilon_i & x_i < c \\ \beta_{0,2} + \beta_{1,2}x_i + \beta_{2,2}x_i^2 + \epsilon_i & x_i \geq c \end{cases}$$

  - the β_i's are typically chosen such that the model pieces "match up" at the knots, but that doesn't explicitly need to be the case (e.g., step functions)
  - there are various varieties of splines: basis (or B) splines, natural splines, etc.
  - if d is the degree of the polynomial and $K-1$ is the number of knots, the overall number of estimated quantities is $(d+1)+K$

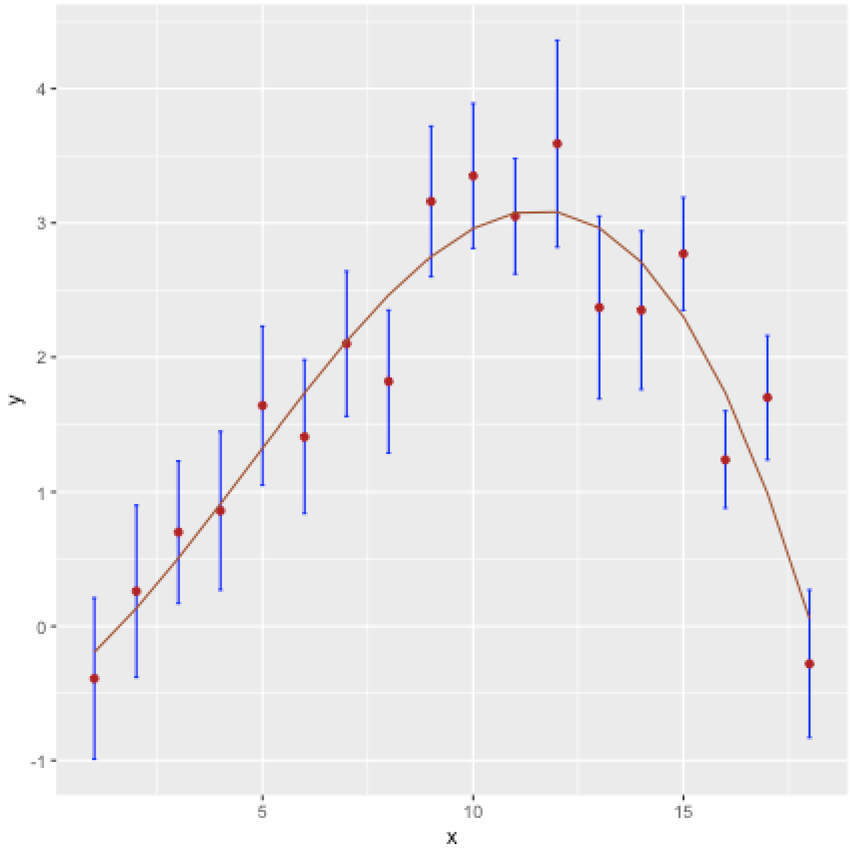
Regression Splines

```r
library(splines)
# default: cubic spline; K+3 basis functions
s.out <- lm(y~bs(x,knots=c(11)),data=df,
            weights=1/(e^2))
s.pred <- predict(s.out)
mean((y-s.pred)^2)
```

```
## [1] 0.1585325
```

```r
data.frame("Estimate"=
             summary(s.out)$coefficients[,1])
```

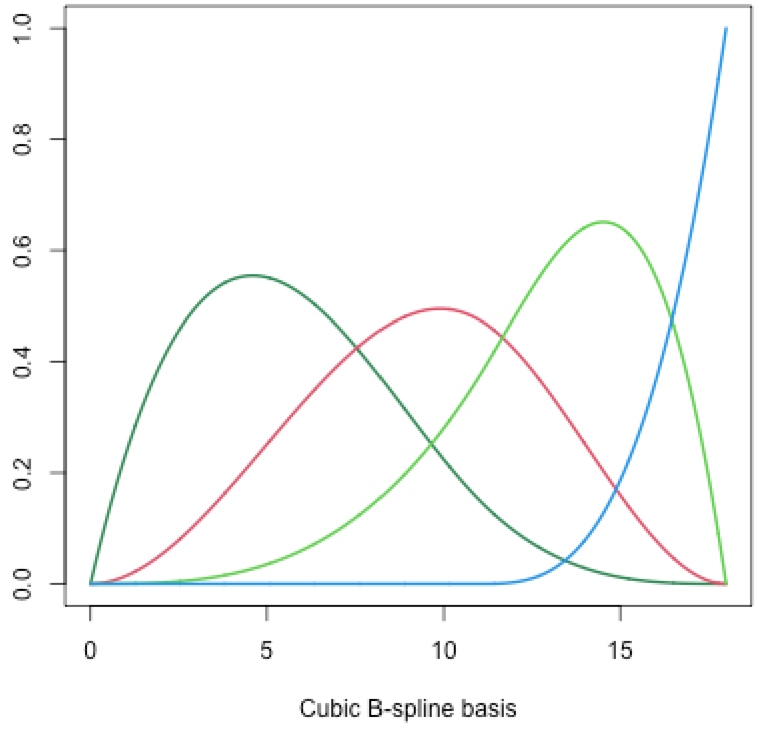
```
##                         Estimate
## (Intercept)           -0.1943431
## bs(x, knots = c(11))1  0.9950228
## bs(x, knots = c(11))2  4.5003507
## bs(x, knots = c(11))3  2.6608156
## bs(x, knots = c(11))4  0.2497895
```



Regression Splines: What Do the Coefficients Mean?

- A cubic B-spline model with one knot is built up using $K + d = 4$ separate basis functions
- Take each function, multiply by the coefficient, add in the intercept, and…voila, you have the overall model shape

```r
bs.x <- seq(0, 18, by=0.01)
spl  <- bs(bs.x,knots=c(11))
plot(spl[,1]~bs.x,ylim=c(0,max(spl)),type='l',lwd=2,
     col="seagreen",xlab="Cubic B-spline basis",
     ylab="")
for ( ii in 2:ncol(spl) ) {
  lines(spl[,ii]~bs.x,lwd=2,col=ii)
}
```



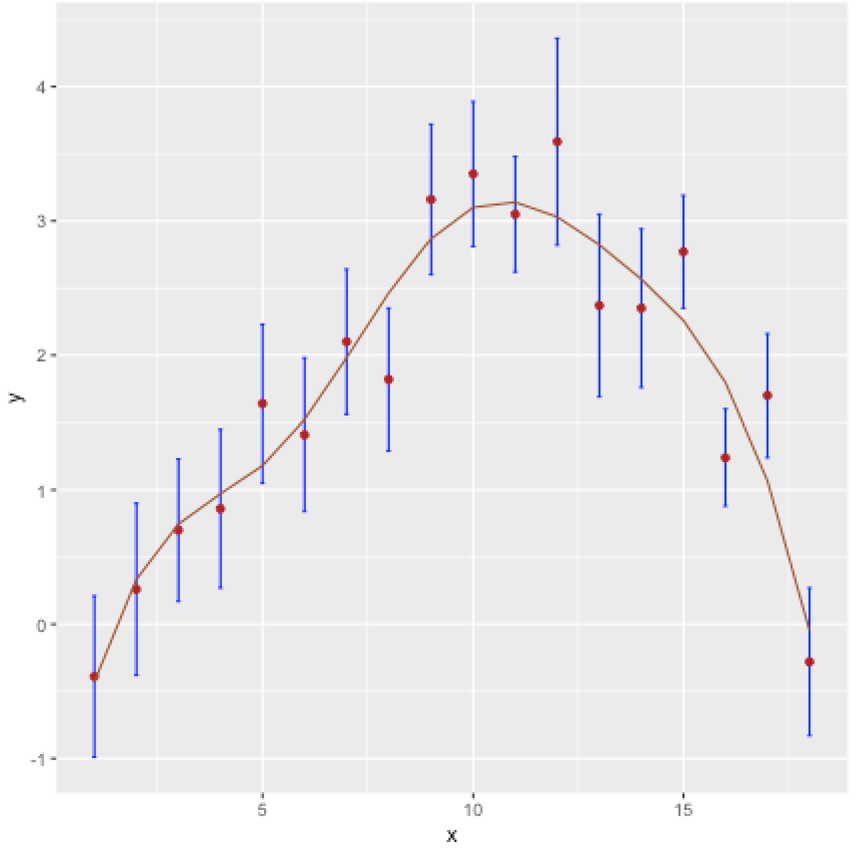
Regression Splines

```r
# don't know where to put knot(s)?
# specify dof instead
s.out <- lm(y~bs(x,df=6),data=df,
            weights=1/(e^2))
s.pred <- predict(s.out)
mean((y-s.pred)^2)
```

```
## [1] 0.1343997
```

```r
data.frame("Estimate"=
             summary(s.out)$coefficients[,1])
```

```
##                   Estimate
## (Intercept)     -0.4304463
## bs(x, df = 6)1   1.4121224
## bs(x, df = 6)2   1.0998492
## bs(x, df = 6)3   4.1123438
## bs(x, df = 6)4   3.0951492
## bs(x, df = 6)5   2.2892663
## bs(x, df = 6)6   0.3834174
```



Smoothing Splines

- A variation on the regression spline model is the *smoothing spline* model
- A smoothing spline model penalizes excessive model wiggliness, which is measured via the second derivative of the model function
- The objective function that we wish to minimize is

$$\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2 + \lambda \int f''(x)^2 dx$$

  - λ is a tuning parameter: smaller values lead to "wigglier" models
- Note that there is no need to place knots: in theory, they are placed at the coordinates of every datum (although in practice a reduced set is sets)
- When learning a smoothing-spline model, we can
  - choose the effective number of degrees of freedom by setting λ; or
  - use cross-validation to determine the optimal value of λ

Smoothing Splines

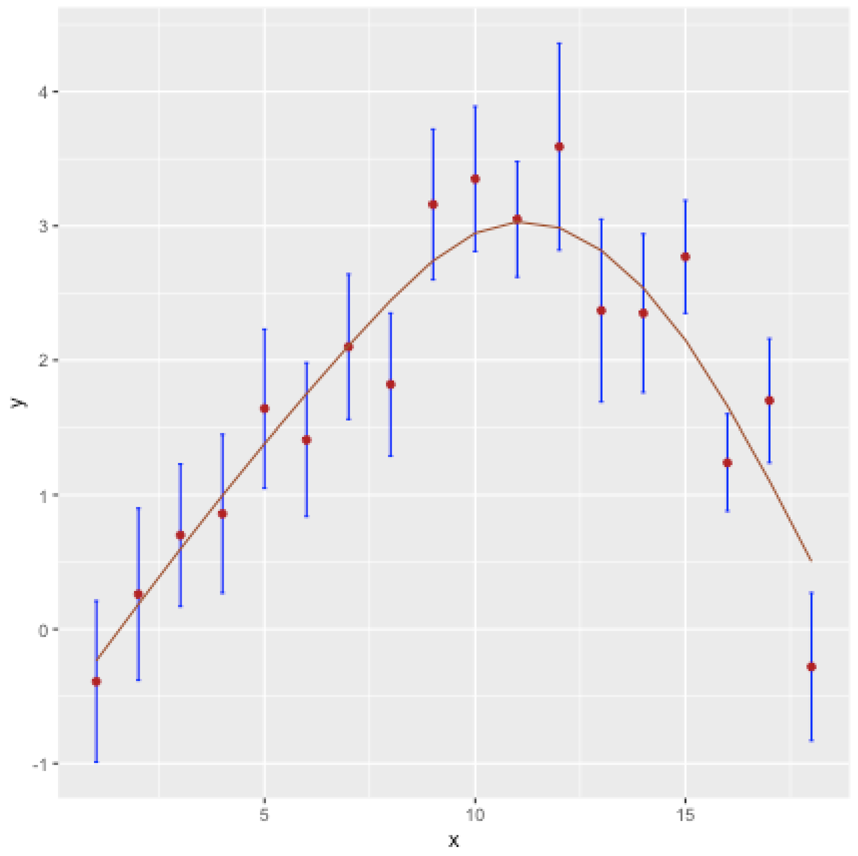
```
ss.out <- suppressWarnings(
  smooth.spline(df$x,y=df$y,w=1/(df$e^2),cv=TRUE)
)
ss.out$lambda
```

```
## [1] 0.002613839
```

```
ss.pred <- predict(ss.out)
mean((y-ss.pred$y)^2)
```

```
## [1] 0.1721536
```

- The number of coefficients is large and the values are not easily interpreted



Local Polynomial Regression

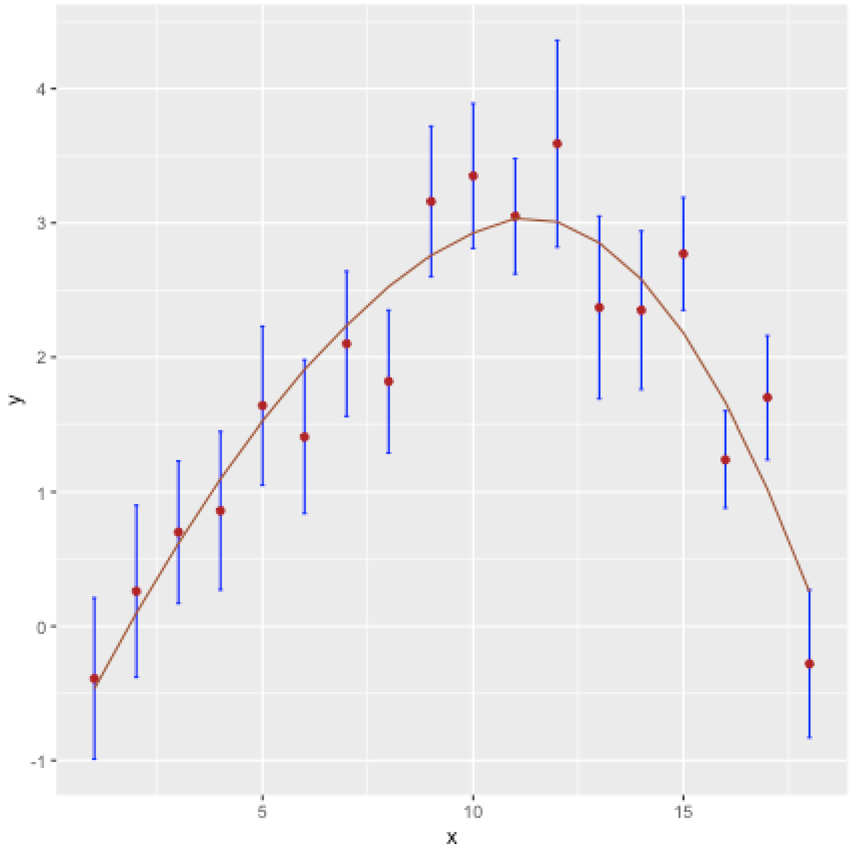
- This is the "localized" version of global polynomial regression
    - at each point, a polynomial is fit, with more weight being given to nearby data points and less to those farther away

Local Polynomial Regression

```r
lpr.out <- loess(y~x,data=df,weights=1/(df$e)^2,span
lpr.pred <- predict(lpr.out)
mean((y-lpr.pred)^2)
```

```
## [1] 0.1728908
```

- An important argument is `span` (default value: 0.75)
  - a larger value `span` means more neighboring data weigh in on the estimate made at x_o
  - therefore, large `span` values produce smoother functions

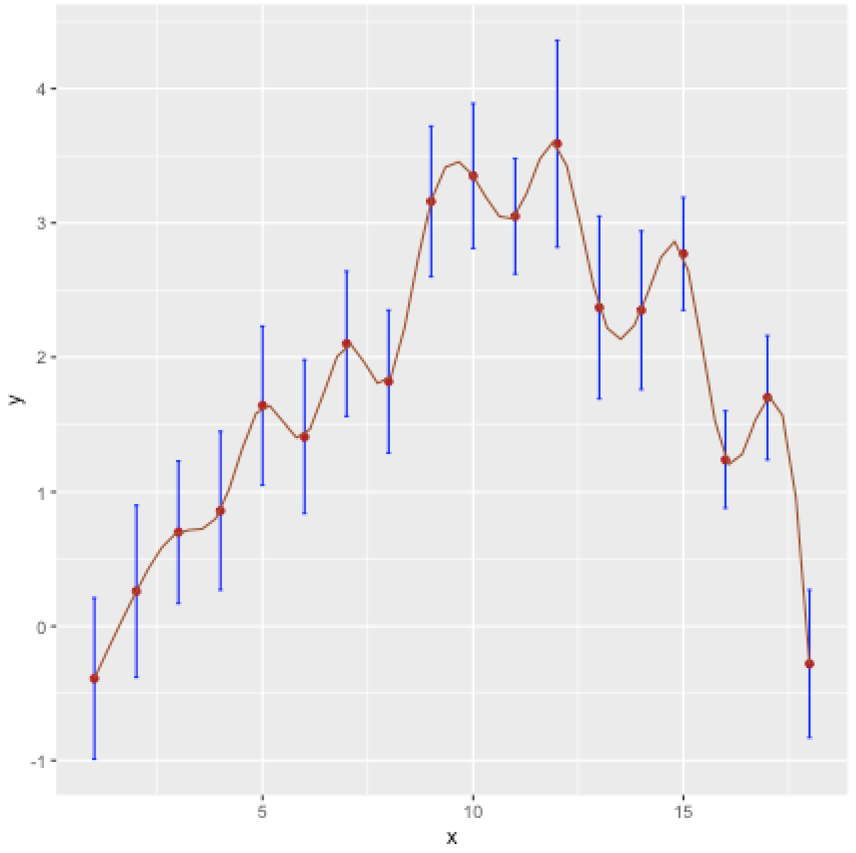


Digression: What About (Cubic) Splines?

- A confusing aspect to using splines is that some functions use them in the context of regression (i.e., they incorporate the idea of randomness) and some use them *deterministically* (i.e., as interpolators that thread functions through every point)

```
# note: uncertainties not taken into account
spl.out <- spline(df$x,df$y)
```

- If your output looks like this, you have not implemented a regression model!



Generalized Additive Models

- We can extend the models that we have discussed above to multiple regression settings
- The *generalized additive model* or *GAM* is

$$Y_i|x_i = \beta_0 + f_1(x_{1,i}) + \cdots + f_p(x_{p,i})$$

  - $x_{j,i}$ is the value of the j^{th} predictor for the i^{th} object
  - $f_j(\cdot)$ is a function applied to the data in the $j^{\backslash\mathrm{th}rm}$ predictor column only...so this could be B-splines, or global polynomials, etc.
- Why would we use GAMs?
  - they provide flexible nonlinear modeling with output that we can (possibly) use to make inferential statements...i.e., GAMs are not black-box models
- Why would we not use GAMs?
  - the "phase space" of model possibilities is *huge*: where do we use B-splines (and where are the knots)? where do we use polynomial regression (and what is the degree)?
- In the end...if model flexibility is needed and inference is not of utmost importance, working with simpler-to-implement nonlinear models like random forest is generally preferable to working with GAMs