Lab 1: Splines – Model answers

```
load(url("http://www.stats.gla.ac.uk/~levers/aptslab1.RData"))
# To make the solution generic we use either of the following three commands:
data <- data.frame(x=divorce[,1], y=divorce[,2])
```

1. Something simple like

   ```
   plot(x, y)
   ```

   should be fine.

2. —

3. Using the built-in functions …

   ```
   model <- lm(y~poly(x,10), data=data)
   yhat <- predict(model)
   plot(x, y)
   lines(x, yhat)
   ```

   Coding from first principles …

   ```
   r <- 4
   B <- outer(x, 0:r, "^")
   beta <- solve(crossprod(B), t(B)%*%y)
   yhat <- B%*%beta
   ```

   Plotting code is same as above. Higher power might not work because of numerical instability. The code using `lm` and `poly` is more stable because `lm` uses a QR decomposition, which is numerically more stable, and because `poly` creates orthogonal polynomials.

4. Simply define `B` using

   ```
   B <- bbase(x, nseg=10, deg=3)
   ```

5. We need to create the basis by hand (which we will do in a function called `tbase`.

   ```
   tbase <- function(x, xl = min(x), xr = max(x), n.knots = 10, deg = 3) {
     nseg <- n.knots - 1
     dx <- (xr - xl) / nseg
     knots <- seq(xl, xr,  by = dx)
     B <- cbind(outer(x-xl,0:(deg-1),"^"),
                outer(x,knots[-length(knots)],function(x,y) pmax(x-y,0))^deg)
     B
   }
   B  <- tbase(x, n.knots=10, deg=3)
   ```

   Both bases are equivalent, so the fitted values are exactly the same.

6. After creating `B` we insert

   ```
   D <- diff(diag(ncol(B)), diff=2)
   lambda <- 1 # or whatever value you like
   beta <-  solve(crossprod(B)+lambda*crossprod(D), t(B)%*%y)
   yhat <- B%*%beta
   ```

7. For the unpenalised models use

   ```r
   beta <- qr.coef(qr(B), y)
   ```

   For P-splines use

   ```r
   beta <- qr.coef(rbind(qr(B), sqrt(lambda)*D), c(y, numeric(nrow(D))))
   ```