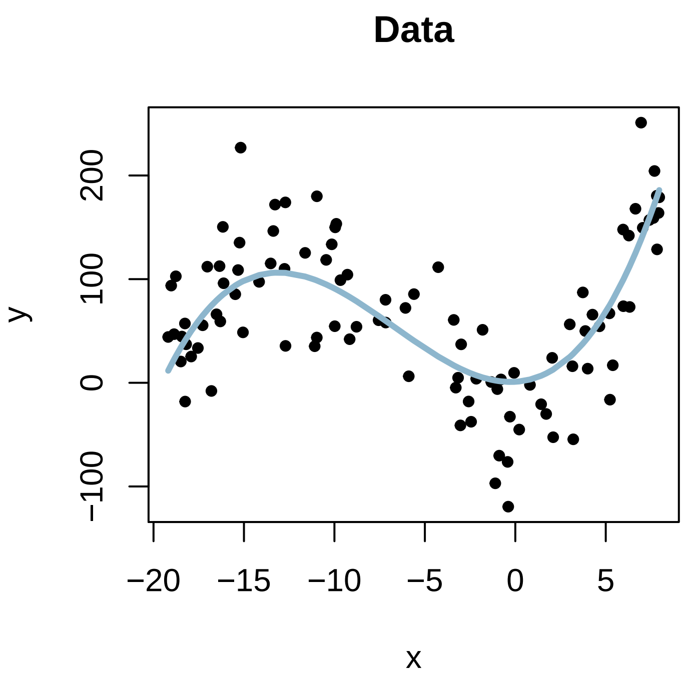
SL: Foundations

We illustrate the bias-variance trade-off on artificial data. Input values are univariate, and uniformly generated on the interval $[-20, 8]$. The input-output relationship is assumed to be cubic, with an additional normal error, so that $y = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \epsilon$, with $\epsilon \sim \mathcal{N}(0, \sigma = 50)$ and $\beta_0 = 1, \beta_1 = 1, \beta_2 = 2, \beta_3 = 0.1$. The learning sample, of size n, is denoted $\mathcal{L}_n := \{(x_1, y_1), ..., (x_n, y_n)\}$. The true input-output relationship is plotted in lightskyblue3 below.

```r
n = 100 # Generate n=100 data points
x = runif(n, -20, 8); x = sort(x);
beta0=1; beta1=1; beta2=2; beta3=.1;
y= beta0 + beta1*x + beta2*x^2 + beta3*x^3 + rnorm(n, 0, 50);
plot(x, y, pch=20)
lines(x, beta0 + beta1*x + beta2*x^2 + beta3*x^3, col="lightskyblue3", lwd=3)
title('Data')
```



On this dataset, we fit polynomials of increasing orders. Consider the set $\mathcal{F}_d$ of polynomial functions of order d. An element $f \in \mathcal{F}_d$ has the form

$$f(x) := \beta_0 + \beta_1 x + ... + \beta_d x^d .$$

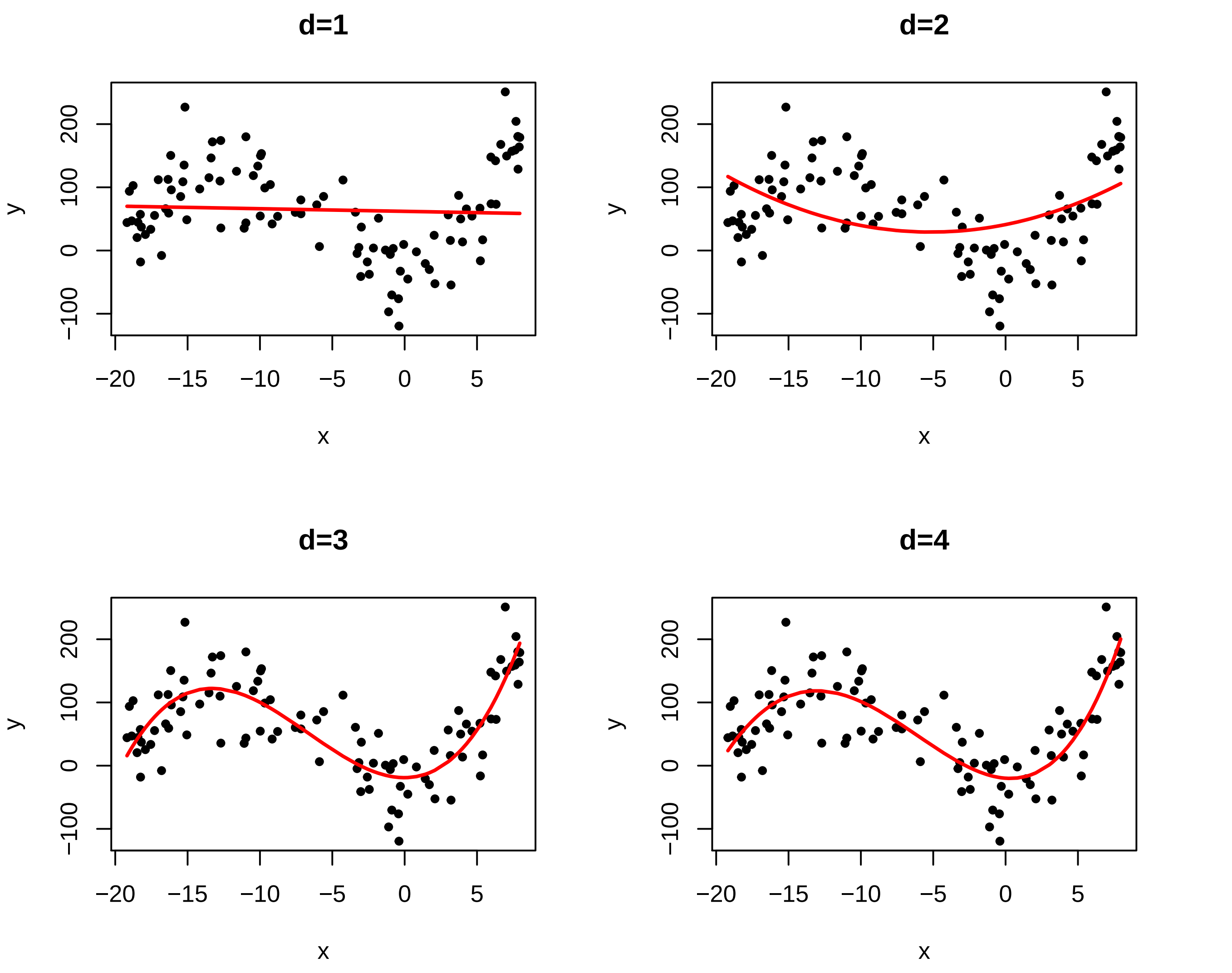
The coefficients $(\beta_0, \beta_1, ..., \beta_d)$ are estimated using a least squares procedure (for more information regarding least squares and its implementation, see chapter SL: LINEAR REGRESSION). The least squares estimate $(\hat{\beta}_0, \hat{\beta}_1, ..., \hat{\beta}_d)$ satisfy

$$\begin{aligned}(\hat{\beta}_0, \hat{\beta}_1, ..., \hat{\beta}_d) &:= \arg\min_{f \in \mathcal{F}_d} \sum_{i=1}^{n} \ell(y_i,\ f(x_i)) \\ &= \arg\min_{(\beta_0, \beta_1, ..., \beta_d)} \sum_{i=1}^{n} (y_i - \beta_0 - \beta_1 x_i - ... - \beta_d x_i^d)^2 ,\end{aligned}$$

where $\ell(y,\ f(x)) := (y - f(x))^2$ denotes the square loss. We consider polynomial fits of order $d = 1, ..., 4$.

```
poly.fit1 = lm(y~poly(x, degree=1, raw=TRUE))
poly.fit2 = lm(y~poly(x, degree=2, raw=TRUE))
poly.fit3 = lm(y~poly(x, degree=3, raw=TRUE))
poly.fit4 = lm(y~poly(x, degree=4, raw=TRUE))
xpower1= matrix(x^(rep(0:1, each=length(x))), ncol=2, byrow=FALSE)
xpower2= matrix(x^(rep(0:2, each=length(x))), ncol=3, byrow=FALSE)
xpower3= matrix(x^(rep(0:3, each=length(x))), ncol=4, byrow=FALSE)
xpower4= matrix(x^(rep(0:4, each=length(x))), ncol=5, byrow=FALSE)

par(mfrow=c(2,2))
plot(x, y, pch=20); lines(x, colSums(t(xpower1)*poly.fit1$coef), col="red", lwd=2); title("d=1")
plot(x, y, pch=20); lines(x, colSums(t(xpower2)*poly.fit2$coef), col="red", lwd=2); title("d=2")
plot(x, y, pch=20); lines(x, colSums(t(xpower3)*poly.fit3$coef), col="red", lwd=2); title("d=3")
plot(x, y, pch=20); lines(x, colSums(t(xpower4)*poly.fit4$coef), col="red", lwd=2); title("d=4")
```

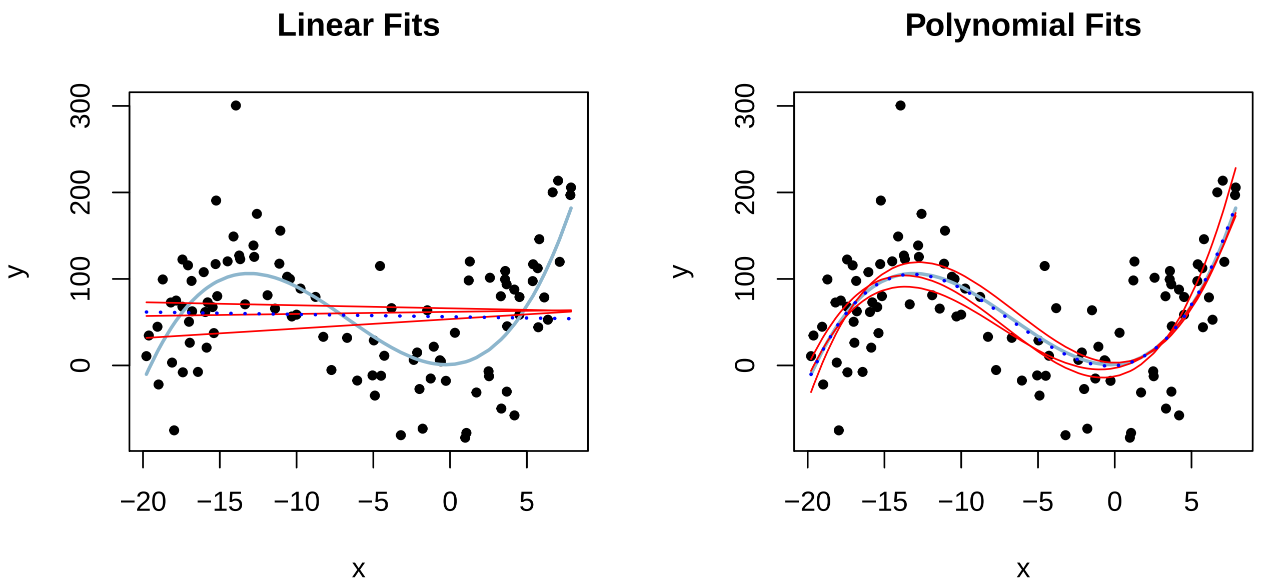


We show using Monte Carlo simulations that the class of linear predictors has high bias and small variance, while the class of polynomials of order $d = 4$ has low bias but high variance. We generate $n_{mc} = 100$ data sets, and compute in each case the least squares fit in $\mathcal{F}_1$ (linear regression) and $\mathcal{F}_4$ (order 4 polynomial). The coefficient estimates are stored in `mc.lin.fit` and `mc.poly.fit`. We plot three fits obtained for three realizations, the average fit obtained over n_{mc} simulations, and the true input-output relationship.

```r
nmc = 100
mc.lin.fit = matrix(rep(0, 2*nmc), nrow=nmc)
mc.poly.fit = matrix(rep(0, 5*nmc), nrow=nmc)
for (i in 1:nmc){
  x = runif(n, -20, 8); x = sort(x);
  y = beta0+ beta1*x+ beta2*x^2+ beta3*x^3+ rnorm(n, 0, 50);
  mc.lin.fit[i, ] = lm(y~poly(x, degree=1, raw=TRUE))$coef
  mc.poly.fit[i, ] = lm(y~poly(x, degree=4, raw=TRUE))$coef
}

par(mfrow=c(1,2))
# We plot only one data set
plot(x, y, pch=20)
xpower1= matrix(x^(rep(0:1, each=length(x))), ncol=2, byrow=FALSE)
lines(x, beta0 + beta1*x + beta2*x^2 + beta3*x^3, col="lightskyblue3", lwd=2)
lines(x, colSums(t(xpower1)*mc.lin.fit[1,]), col="red", lwd=1)
lines(x, colSums(t(xpower1)*mc.lin.fit[2,]), col="red", lwd=1)
lines(x, colSums(t(xpower1)*mc.lin.fit[3,]), col="red", lwd=1)
lines(x, colSums(t(xpower1)*colMeans(mc.lin.fit)), col="blue", lwd=2, lty="dotted")
title("Linear Fits")

plot(x, y, pch=20)
xpower4= matrix(x^(rep(0:4, each=length(x))), ncol=5, byrow=FALSE)
lines(x, beta0 + beta1*x + beta2*x^2 + beta3*x^3, col="lightskyblue3", lwd=2)
lines(x, colSums(t(xpower4)*mc.poly.fit[1,]), col="red", lwd=1)
lines(x, colSums(t(xpower4)*mc.poly.fit[2,]), col="red", lwd=1)
lines(x, colSums(t(xpower4)*mc.poly.fit[3,]), col="red", lwd=1)
lines(x, colSums(t(xpower4)*colMeans(mc.poly.fit)), col="blue", lwd=2, lty="dotted")
title("Polynomial Fits")
```



We see that that the linear fits (in red) do not change much from one realization to another (small variance), while the average fit (in blue, dotted line) is relatively far from the true (in light blue) input-output relationship (high bias). In contrast, the polynomial fit displays high variability from one realization to another (high variance), while the average fit is close to the true one (low bias). The variability in the coefficient estimates can be seen as well by looking directly at the estimates themselves:

```r
head(mc.lin.fit)
```

```
##          [,1]       [,2]
## [1,] 61.95035  0.2411561
## [2,] 53.56417  1.0977741
## [3,] 66.00820 -0.3528587
## [4,] 42.70502 -1.9869397
## [5,] 46.98786 -2.2177309
## [6,] 54.39514 -0.4234781
```

```r
head(mc.poly.fit)
```

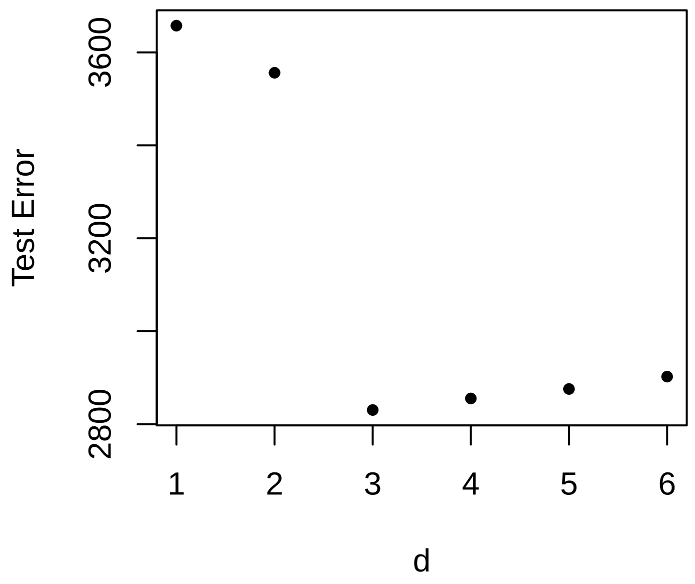
```
##            [,1]       [,2]     [,3]       [,4]          [,5]
## [1,] -12.720195  3.5467925 2.457154 0.12065028  0.0004081893
## [2,]  -3.106087  3.3365168 1.876559 0.07322976 -0.0006831384
## [3,]   3.146115 -0.2170363 2.045969 0.10088411 -0.0003790370
## [4,]   5.689992 -0.7638915 1.807555 0.08912410 -0.0003786707
## [5,]  -9.504370  4.6252414 2.291621 0.06891746 -0.0017535248
## [6,]   4.037586  0.8213981 1.905893 0.09368348 -0.0001404595
```

In particular, we see that the intercept estimate (first column) is highly variable in the case of polynomial regression; its sign even differs from one estimate to another. In future chapters, we investigate ways the reduce the variability of the fit, by penalising fits with large values of the coefficients, thus reducing the overall variance. Such techniques are called regularization techniques, see chapter SL: RIDGE REGRESSION AND LASSO).

We illustrate the bias-variance trade-off in the presence of a training and test sample. For each Monte Carlo simulation, we fit the training data, whose performance is then evaluated on an independent test sample. The Mean Square Error (MSE) is then averaged over the n_{mc} simulations. We condider polynomial fits of up to order $d = 6$.

```r
dmax=6
MSE= matrix(0, nrow=nmc, ncol=dmax)
for (i in 1:nmc){
  # 1. Generate the training and test sample
  x.train= runif(n, -20, 5); x.train= sort(x.train);
  y.train= beta0+ beta1*x.train+ beta2*x.train^2+ beta3*x.train^3+ rnorm(n, 0, 50);
  x.test= runif(n, -20, 5); x.test= sort(x.test);
  y.test= beta0+ beta1*x.test+ beta2*x.test^2+ beta3*x.test^3+ rnorm(n, 0, 50);

  # 2. Fit a polynomial of degree d (=1,...,6) and compute the test error
  for (d in 1:dmax){
    poly.fit= lm(y.train~poly(x.train, degree=d, raw=TRUE))
    y.hat= predict(poly.fit, data.frame(x.test))
    MSE[i, d]= mean((y.hat-y.test)^2)
  }
}
# Plot the test error as a function of polynomial degree
plot(1:dmax, colMeans(MSE), pch=20, xlab="d", ylab="Test Error")
```



The minimum MSE is obtained for $d = 3$, as expected.