Assignment I - Complexity And Noise

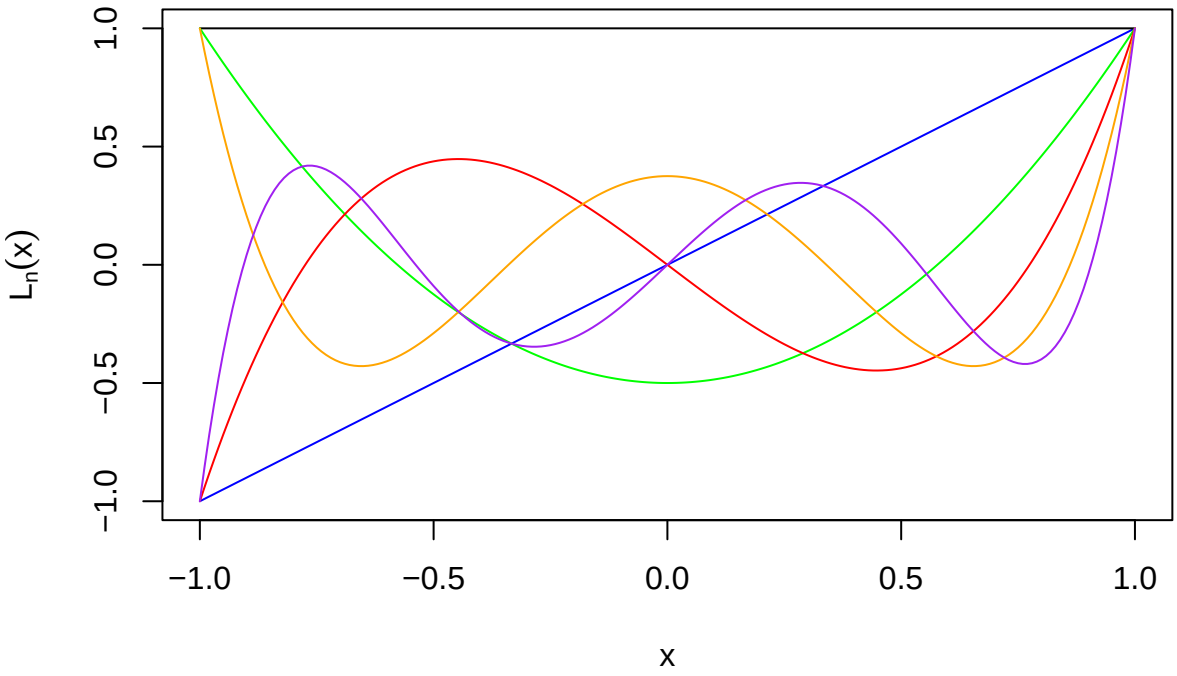
Roger Bukuru

2024-10-07

Question 1

(a)

```
# Legendre polynomials
Legendre=function(x,q){
    val=0
    for(i in 0:q){
        val=val+((x^i)*choose(q,i)*choose((q+i-1)/2,q))
    }
    return((2^q)*val)
}

x<-seq(-1,1,0.01)
plot(x,Legendre(x,1),type="l",ylab=expression(L[n](x)),col="blue")
lines(x,Legendre(x,0),type="l",col="black")
lines(x,Legendre(x,2),type="l",col="green")
lines(x,Legendre(x,3),type="l",col="red")
lines(x,Legendre(x,4),type="l",col="orange")
lines(x,Legendre(x,5),type="l",col="purple")
```

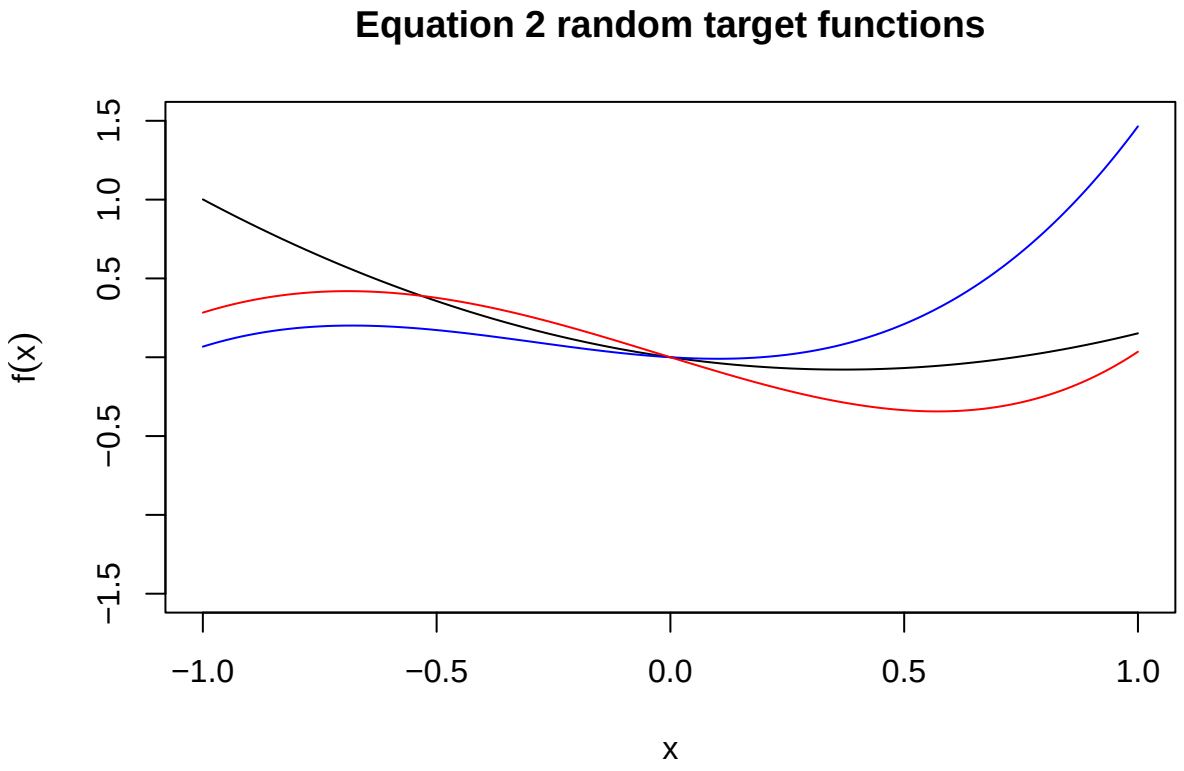


The behaviour that we observe is that as we increase q the complexity of the resulting function increases, it ranges from a function that is at the intercept to a 5-degree polynomial.

(b)

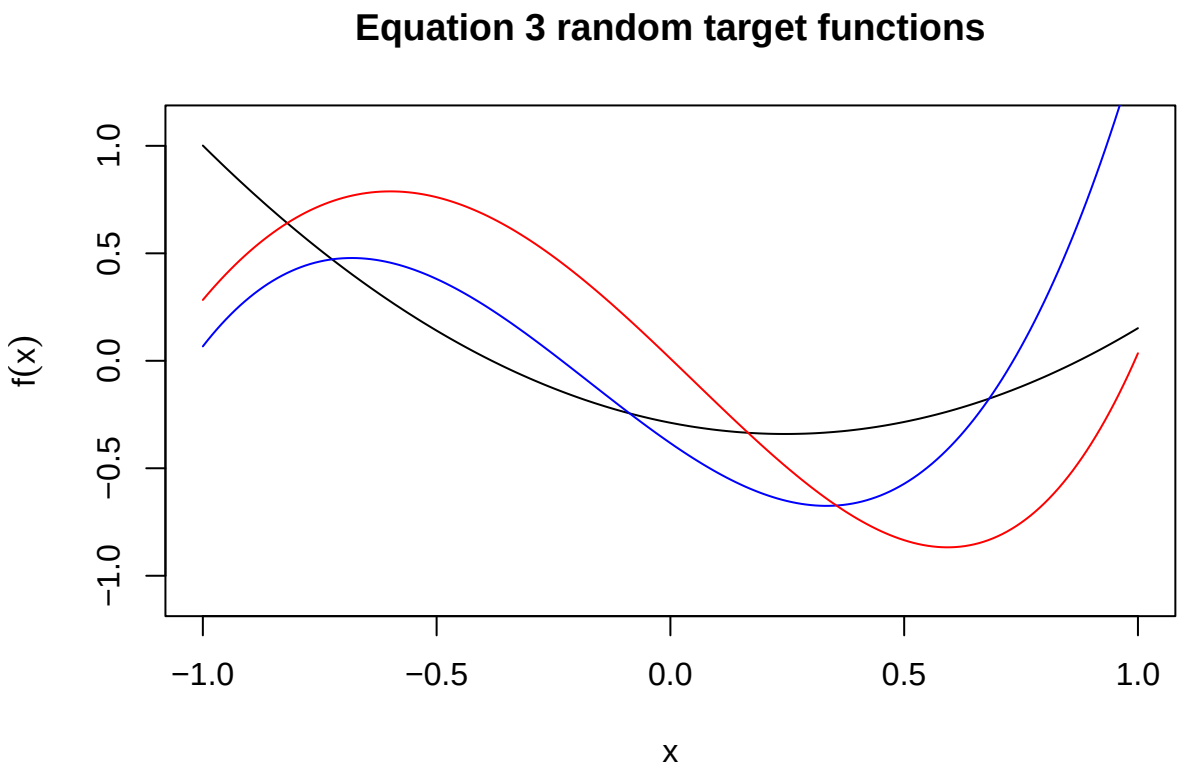
```
set.seed(123)
# creates random polynomial target functions
rPolyfunc=function(x,n){ # Equation 2
  val=0
  vec=runif(n,-1,1)
  for(i in 1:n){
    val=val+(x^i*vec[i])
  }
  return(val)
}

rLegfunc=function(x,n){ # Equation 3
    val=0
    vec=runif(n,-1,1)
    for(i in 1:n){
        val=val+(Legendre(x,i)*vec[i])
    }
    return(val)
}
```

```r
# Equation 2 randomly generated functions
x<-seq(-1,1,0.01)
plot(x,1.5*x,type="n",ylab=expression(f(x)), main= "Equation 2 random target functions")
lines(x,rPolyfunc(x,2),type="l")
lines(x,rPolyfunc(x,3),col="blue")
lines(x,rPolyfunc(x,4),col="red")
```




```r
#lines(x,rPolyfunc(x,5),col="green")
```

```r
# Equation 3 randomly generated functions
set.seed(123)
x<-seq(-1,1,0.01)
plot(x,1.1*x,type="n",ylab=expression(f(x)), main= "Equation 3 random target functions")
lines(x,rLegfunc(x,2),type="l")
lines(x,rLegfunc(x,3),col="blue")
lines(x,rLegfunc(x,4),col="red")
```



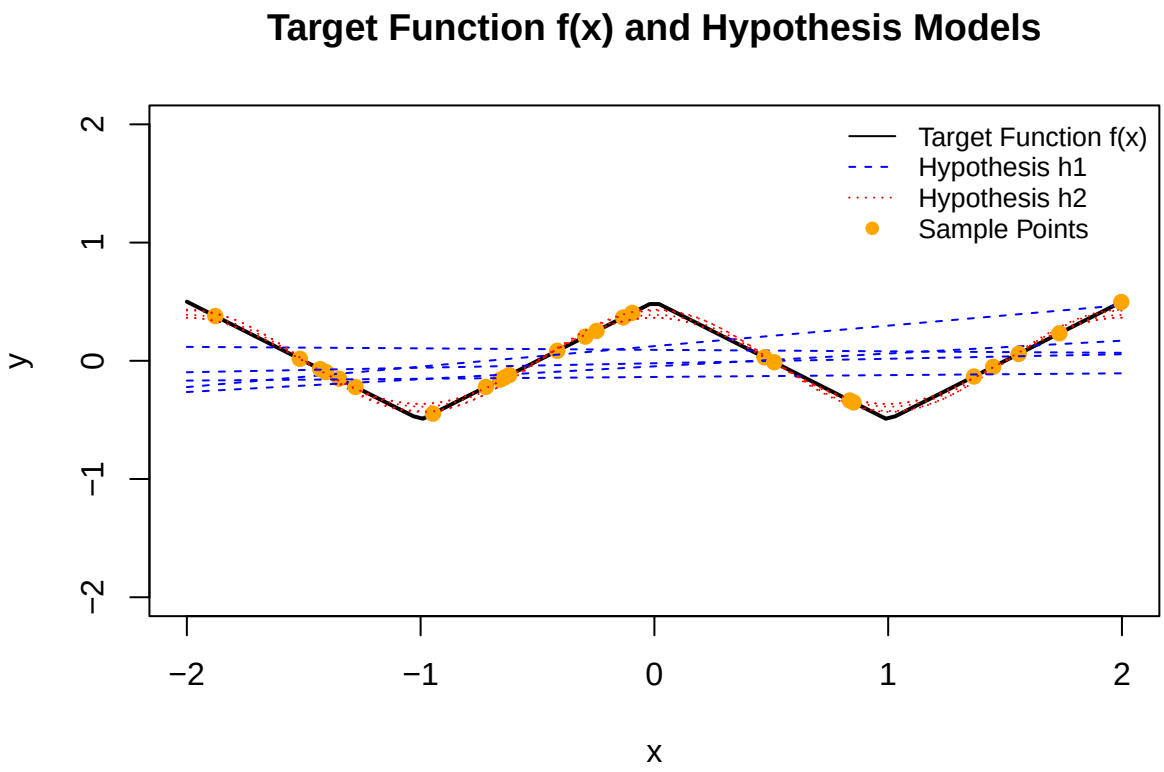
Reviewing the above two figures, we observe that for the target functions generated from equation 2, they have fewer degrees of freedom compared to the target functions generated from equation 3. As a result, they depict target functions with less model complexity compared to those from equation 3. The target functions from equation 3 are also smoother and can in theory better capture non-linear relationships, however if we push the q value we may end up with more complex functions similar to those from question 1.

Question 3

(a)

```
rm(list=ls())

set.seed(2023)
# Set up a target function/pattern:
f = function(x) {
  ifelse(x < rep(0, length(x)),  (abs(x + 1) - 0.5), (abs(x - 1) - 0.5))
}


# Data Generating Process
dgp = function(N, f, sig2=0)
{
   x = runif(N,-2,2)
```

```r
   e = rnorm(N,0,sqrt(sig2))
   y = f(x) + e
   return(list(y = y, x = x))
}


# Hypothesis models
h1 = function(x, w0, w1) {
  w0 + w1 * x
}

h2 = function(x, w1, w2) {
  w1 * sin(pi * x) + w2 * cos(pi * x)
}

# Plot f(x) over the input space
x_vals = seq(-2, 2, length.out = 100)
plot(x_vals, f(x_vals), type = "l", col = "black", ylim = c(-2, 2), lwd = 2,
     main = "Target Function f(x) and Hypothesis Models", xlab = "x", ylab = "y")

for( i in 1:5){
  sample_data = dgp(5, f)
  h1_fit      = lm(y~x, data=sample_data)
  h1_pred     = predict(h1_fit, newdata = data.frame(x = x_vals))

  sample_data$z1 = sin(pi * sample_data$x)
  sample_data$z2 = cos(pi * sample_data$x)
  h2_fit = lm(y ~ z1 + z2 - 1, data = sample_data)
  h2_pred = predict(h2_fit, newdata = data.frame(z1 = sin(pi * x_vals), z2 = cos(pi * x_vals)))
  # Plot the fitted models
  lines(x_vals, h1_pred, col = "blue", lty = 2)  # h1 hypothesis
  lines(x_vals, h2_pred, col = "red", lty = 3) # h2 hypothesis

  # Add points for each sample
  points(sample_data$x, sample_data$y, col = "orange", pch = 19)
}

legend("topright", legend = c("Target Function f(x)", "Hypothesis h1", "Hypothesis h2", "Sample Points"
       col = c("black", "blue", "red", "orange"), lty = c(1, 2, 3, NA), pch = c(NA, NA, NA, 19),
       bty = "n", cex = 0.8)
```



(b)

Based on the plot in (a) hypothesis $h2$ apriori shows that it would best capture the target function. However it could fail to generalize out-of-sample as it is overly fitting onto the target function. The model complexity indicates that it has low-bias however when attempting to learn i.e generalize we apriori expect it to product sub-optimal results.

(c)

```
library(knitr)
library(kableExtra)
bias_var = function(N = 5, order = 1, sig2 = 0, M = 1000, hypothesis = "h1", dx = 1/100)
{
    x_lat = seq(-2,2,dx)
    N_dx  = length(x_lat)
    g_bar = rep(0,N_dx)
    G_d   = matrix(0,M,N_dx)

    errors = rep(0,M)
    for(i in 1:M)
    {
```

```
        res_data = dgp(N,f,sig2)
        g_D = NULL
        if(hypothesis == "h1"){
           res_model = lm(y~x, data=res_data)
           g_D     = predict(res_model,data.frame(x = x_lat))
           g_bar   = g_bar + g_D # Calculating the sum part in the averaging
           G_d[i,] = g_D
           dat_OOS  = dgp(N,f,sig2)
           yhat_OOS = predict(res_model,data.frame(x=dat_OOS$x))
           errors[i] = mean((yhat_OOS-dat_OOS$y)^2)
        }
      if(hypothesis == "h2"){
           res_data$z1  = sin(pi * res_data$x)
       res_data$z2  = cos(pi * res_data$x)
       #print(res_data)
       res_model    = lm(y ~ z1 + z2 - 1, data = res_data)
       g_D          = predict(res_model, newdata = data.frame(z1 = sin(pi * x_lat), z2 = cos(pi * x_lat)
       g_bar        = g_bar + g_D # Calculating the sum part in the averaging
       G_d[i,]      = g_D
           dat_OOS      = dgp(N,f,sig2)
           dat_OOS$z1   = sin(pi * dat_OOS$x)
       dat_OOS$z2   = cos(pi * dat_OOS$x)
           yhat_OOS     =  predict(res_model, newdata = data.frame(z1 = sin(pi * dat_OOS$x), z2 = cos(pi
           errors[i]    = mean((yhat_OOS-dat_OOS$y)^2)
      }


    }
    g_bar = g_bar/M # Finally calculate the average

    test_error = mean(errors)
    phi_X = 1/2
    bias2 = sum((g_bar-f(x_lat))[-N_dx]^2*phi_X*dx)

    ones = matrix(1,M,1)
    var_at_x = colSums((G_d-ones%*%g_bar)^2)/M
    var      = sum(var_at_x[-N_dx]*phi_X*dx)

    return(list(bias2 = bias2,var = var, both = bias2+var, test_error = test_error, g_bar = g_bar))
}
h1_model_bias_variance = bias_var(N = 5, order = 1, sig2 = 0, M = 1000, hypothesis = "h1", dx = 1/100)
h2_model_bias_variance = bias_var(N = 5, order = 1, sig2 = 0, M = 1000, hypothesis = "h2", dx = 1/100)



results <- data.frame(
  Component = c("Bias", "Variance"),
  `H_0` = c(round(h1_model_bias_variance$bias2, 4), round(h1_model_bias_variance$var, 4)),
  `H_1` = c(round(h2_model_bias_variance$bias2, 4), round(h2_model_bias_variance$var, 4))
)

#kable(results, format = "html", caption = "Bias and Variance for H0 and H1") %>%
 # kable_styling(full_width = FALSE, position = "center")
```

Table 1: Bias and Variance for H0 and H1

Component	H_0	H_1
Bias	0.1675	0.0024
Variance	0.1728	0.0016