MA589 - Computational Statistics Project 2

Megha Pandit

October 9, 2018

1.(a) Explain why horner works; start by understanding how a small polynomial, say c(3, -2, 1), is evaluated, and then provide a mathematical expression that summarizes how the computations are performed.

```
horner <- function (coef)
function (x) {
s <- 0
for (i in seq(length(coef), 1, -1)){
s <- s * x + coef[i]
}
s
}
coef <- c(3,-2,1)
x <- 3
horner(coef)(x)
```

```
## [1] 6
```

Evaluating the polynomial c(3,-2,1) at x = 3, the above function computes it iteratively for i = 3,2 and 1, as $s_2 = s_2x + coef_3$, $s_1 = s_1x + coef_2$ ***and*** $s_0 = s_0x + coef_1$. ***Mathematically, if*** $p(x)$ ***is a polynomial*** $p(x) = b_0 + b_1x + b_2x^2 + b_3x^3 + ... + b_nx^n$, ***then, we can evaluate*** $p(x)$ ***at*** $x = x_0$ ***as follows: We can write the polynomial as***

$$p(x_0) = b_0 + x_0(b_1 + x_0(b_2 + x_0(....x_0(b_{n-1} + b_nx_0))))$$

We can define constants $s_n, s_{n-1}, ...$ ***such that***

$$s_n = s_nx_0 + b_n$$

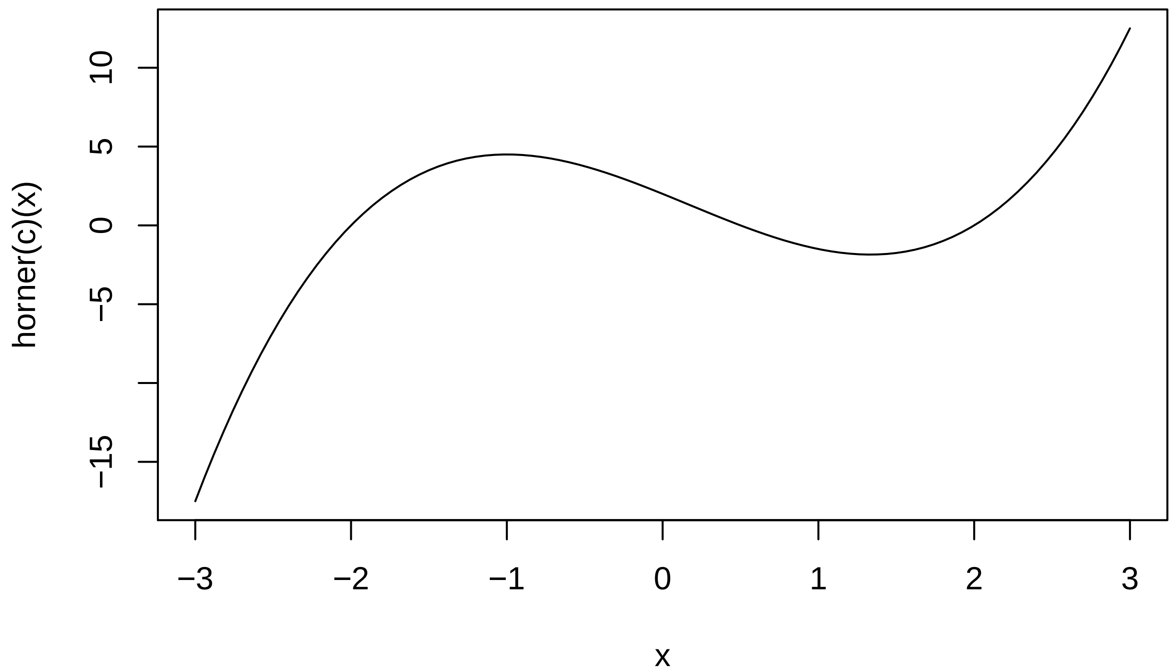
Since we are starting with s = 0,

$$s_n = 0 + b_n = b_ns_{n-1} = s_nx_0 + b_{n-1}s_{n-2} = s_{n-1}x_0 + b_{n-2}...s_0 = s_1x_0 + b_0$$

Iteratively substituting $s_i, i = n, (n-1), (n-2), ..., 0$,***in the p(x) equation, we get*** $p(x_0) = s_0$. ***In short, we can write the mathematical expression as: for*** $i = n, (n-1), (n-2), ..., 0$,
$s_i = b_i$ ***and*** $s_{i-1} = s_ix + b_{i-1}$.

1.(b) Use horner to plot $p(x) = 2 - 4x - x^2/2 + x^3$ for $x \in [-3, 3]$. (Hint: check the code in the next problem, or see curve in R.)

```
c <- c(2,-4,-1/2,1)
curve(horner(c)(x), from = -3, to = 3)
```



1.(c) Implement Newton's method to find the roots of p from the previous item.

```r
#horner function for evaluating polynomial
horner <- function (coef)
function (x) {
s <- 0
for (i in seq(length(coef), 1, -1)){
s <- s * x + coef[i]
}
s
}

#horner function for evaluating the derivative of polynomial
dhorner <- function(coef)
function(x){
s <- 0
for (i in seq(length(coef), 2, -1)){
  s <- s*x + (i-1)*coef[i]
  }
  s
}

#Newton's method for finding the roots of the polynomial
newton <- function(x){
  p <- x
  m <- p - horner(coef)(p)/dhorner(coef)(p)
  while (abs(p-m) > 1e-12) {
```

```r
    p <- m
    m <- p - horner(coef)(p)/dhorner(coef)(p)
  }
  print(p)
}
coef <- c(2,-4,-1/2,1)
newton(-1.5)
```

```
## [1] -2
```

```r
newton(0)
```

```
## [1] 0.5
```

```r
newton(1.5)
```

```
## [1] 2
```

Starting from x = -1 gives an error.

1.(d) Legendre polynomial

2.(a)creating the tableau T

```r
#creating a vandermonde matrix of predictors
T <- matrix(0,6,6)
x <- c(-3.0, -2.5, -2.0, -1.5, -1.0, -0.5, 0.0, 0.5, 1.0, 1.5, 2.0, 2.5, 3.0)
X <- outer(x, 0:4, FUN = "^")

#creating the tableau T
y <- c(-17.0, -7.0, 0.5, 3.0, 5.0, 4.0, 2.5, -0.5, -2.0, -2.0, 0.5, 5.0, 12.0)
T[1:5,1:5] <- crossprod(X,X)
T[1:5,6] <- crossprod(X,y)
T[6,1:5] <- crossprod(y,X)
T[6,6] <- crossprod(y,y)
T
```

```
##          [,1]    [,2]     [,3]      [,4]       [,5]      [,6]
## [1,]  13.000   0.000   45.500    0.0000   284.3750    4.0000
## [2,]   0.000  45.500    0.000  284.3750     0.0000  100.2500
## [3,]  45.500   0.000  284.375    0.0000  2099.0938  -47.3750
## [4,]   0.000 284.375    0.000 2099.0938     0.0000  946.0625
## [5,] 284.375   0.000 2099.094    0.0000 16739.0234 -458.8438
## [6,]   4.000 100.250  -47.375  946.0625  -458.8438  572.0000
```

2.(b)

```r
SWEEP <- function(T, k){
  n <- nrow(T)
  D <- T[k,k]
  T[k,] <- T[k,]/D
  for (i in 1:n) {
    if (i != k){
      B <- T[i,k]
      T[i,] <- T[i,] - B*T[k,]
      T[i,k] <- (-1)*B/D
    }
  }
  T[k,k] <- 1/D
```

```r
  return(T)
}

for (i in 1:5) {
  T <- SWEEP(T,i)
}
T
```

```
##             [,1]        [,2]        [,3]         [,4]         [,5]
## [1,]  0.27848622  0.00000000 -0.12957631  0.000000000  0.011517894
## [2,]  0.00000000  0.14338439  0.00000000 -0.019425019  0.000000000
## [3,] -0.12957631  0.00000000  0.10758033  0.000000000 -0.011289364
## [4,]  0.00000000 -0.01942502  0.00000000  0.003108003  0.000000000
## [5,]  0.01151789  0.00000000 -0.01128936  0.000000000  0.001279766
## [6,] -1.96770876  4.00299700  0.43486905 -0.993006993  0.006307418
##              [,6]
## [1,]  1.967708762
## [2,] -4.002997003
## [3,] -0.434869053
## [4,]  0.993006993
## [5,] -0.006307418
## [6,]  2.486895457
```

#Quick Check: SWEEP(SWEEP(T, k), k) returns the original tableau T.

```r
#Quick Check:
for (i in 1:5) {
SWEEP(SWEEP(T, i), i)
}
T
```

```
##             [,1]        [,2]        [,3]         [,4]         [,5]
## [1,]  0.27848622  0.00000000 -0.12957631  0.000000000  0.011517894
## [2,]  0.00000000  0.14338439  0.00000000 -0.019425019  0.000000000
## [3,] -0.12957631  0.00000000  0.10758033  0.000000000 -0.011289364
## [4,]  0.00000000 -0.01942502  0.00000000  0.003108003  0.000000000
## [5,]  0.01151789  0.00000000 -0.01128936  0.000000000  0.001279766
## [6,] -1.96770876  4.00299700  0.43486905 -0.993006993  0.006307418
##              [,6]
## [1,]  1.967708762
## [2,] -4.002997003
## [3,] -0.434869053
## [4,]  0.993006993
## [5,] -0.006307418
## [6,]  2.486895457
```

2.(c)

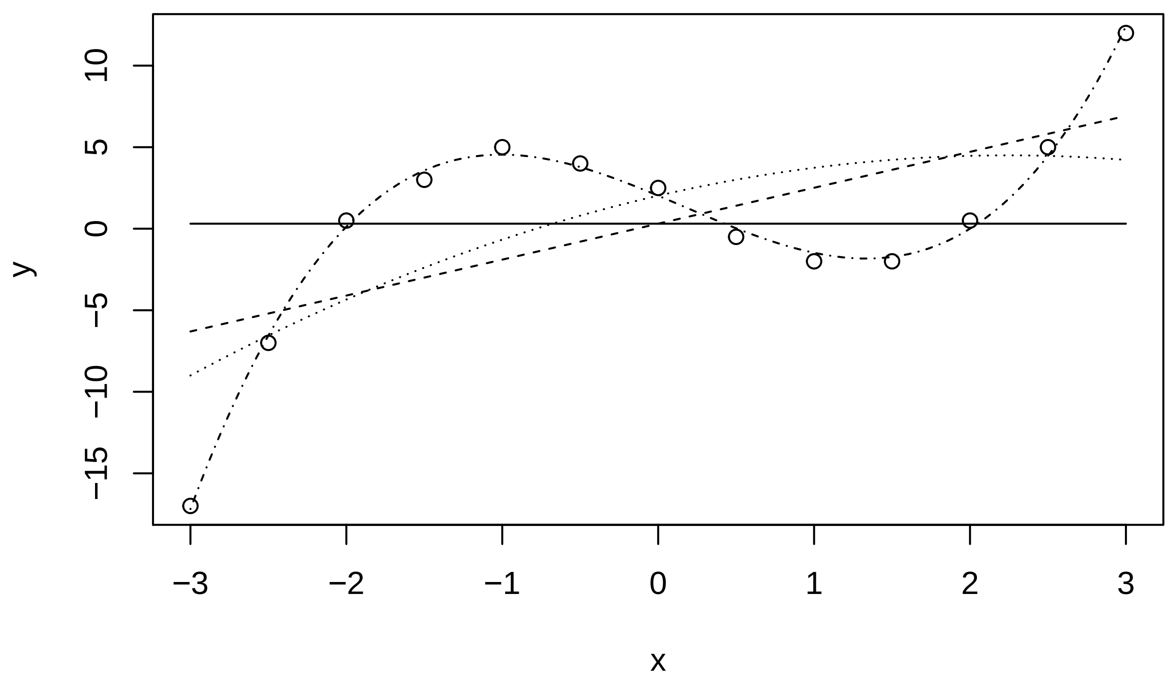
```r
T <- matrix(0,6,6)
x <- c(-3.0, -2.5, -2.0, -1.5, -1.0, -0.5, 0.0, 0.5, 1.0, 1.5, 2.0, 2.5, 3.0)
X <- outer(x, 0:4, FUN = "^")

#creating the tableau T
y <- c(-17.0, -7.0, 0.5, 3.0, 5.0, 4.0, 2.5, -0.5, -2.0, -2.0, 0.5, 5.0, 12.0)
T[1:5,1:5] <- crossprod(X,X)
T[1:5,6] <- crossprod(X,y)
```

```r
T[6,1:5] <- crossprod(y,X)
T[6,6] <- crossprod(y,y)
T
```

```
##          [,1]    [,2]     [,3]      [,4]       [,5]      [,6]
## [1,]  13.000   0.000   45.500    0.0000   284.3750    4.0000
## [2,]   0.000  45.500    0.000  284.3750     0.0000  100.2500
## [3,]  45.500   0.000  284.375    0.0000  2099.0938  -47.3750
## [4,]   0.000 284.375    0.000 2099.0938     0.0000  946.0625
## [5,] 284.375   0.000 2099.094    0.0000 16739.0234 -458.8438
## [6,]   4.000 100.250  -47.375  946.0625  -458.8438  572.0000
```

```r
plot(x, y)
a <- seq(-3, 3, length.out = 100)
p <- ncol(T) - 1
for (k in 1:4) {
T <- SWEEP(T, k)
lines(a, horner(T[1:k, p + 1])(a), lty = k)
print(c(k, T[p + 1, p + 1]))
}
```




```
## [1]   1.0000 570.7692
## [1]   2.0000 349.8887
## [1]   3.0000 319.7837
## [1] 4.000000 2.517982
```

For each k, the regression line with degree k for $y = \sum_{i=0}^{n} \beta_i x^i$ is getting plotted. For k=1, it is a

straight line. For k=2, it is a linear regression line. For k=3, it is a parabola and for k=4, it is a curve. The T[p+1,p+1] element is the RSS. Therefore, for each k, the RSS is being printed.

3.(a)Inverse Schur Complement

```
P <- matrix(c(4,5,3,1),2,2)
Q <- matrix(c(3,5,4,7,2,1),2,3)
R <- matrix(c(1,8,6,4,3,2),3,2)
S <- matrix(c(1,3,4,2,5,6,1,9,5),3,3)

K <- rbind(cbind(P, Q), cbind(R,S))
solve(K)
```

```
##               [,1]       [,2]          [,3]       [,4]        [,5]
## [1,]  1.110223e-15  1.0000000 -5.551115e-16  1.0000000 -2.00000000
## [2,]  2.340426e-01 -0.1063830  1.489362e-01 -0.0212766 -0.06382979
## [3,]  2.170213e+00 -2.5319149 -1.255319e+00 -2.1063830  3.68085106
## [4,] -1.595745e+00  1.3617021  8.936170e-01  0.8723404 -1.38297872
## [5,]  8.510638e-02 -0.7659574 -1.276596e-01 -0.5531915  1.34042553
```

```
#inverse Schur complement
S1 <- K[3:5,3:5] - K[3:5,1:2]%*%solve(K[1:2,1:2])%*%K[1:2,3:5]
solve(S1)
```

```
##            [,1]       [,2]      [,3]
## [1,] -1.2553191 -2.1063830  3.680851
## [2,]  0.8936170  0.8723404 -1.382979
## [3,] -0.1276596 -0.5531915  1.340426
```

```
for (k in 1:2) {
  K <- SWEEP(K,k)
}
K
```

```
##             [,1]       [,2]       [,3]       [,4]        [,5]
## [1,] -0.09090909  0.2727273  1.0909091  1.5454545  0.09090909
## [2,]  0.45454545 -0.3636364 -0.4545455 -0.7272727  0.54545455
## [3,] -1.72727273  1.1818182  1.7272727  3.3636364 -1.27272727
## [4,] -0.63636364 -1.0909091 -4.3636364 -5.1818182  6.63636364
## [5,] -0.36363636 -0.9090909 -1.6363636 -1.8181818  3.36363636
```

```
for (k in 3:5) {
 K <- SWEEP(K,k)
}
K #The yy block of K is equal to the inverse Schur complement
```

```
##               [,1]       [,2]          [,3]       [,4]        [,5]
## [1,]  4.163336e-16  1.0000000 -3.608225e-16  1.0000000 -2.00000000
## [2,]  2.340426e-01 -0.1063830  1.489362e-01 -0.0212766 -0.06382979
## [3,]  2.170213e+00 -2.5319149 -1.255319e+00 -2.1063830  3.68085106
## [4,] -1.595745e+00  1.3617021  8.936170e-01  0.8723404 -1.38297872
## [5,]  8.510638e-02 -0.7659574 -1.276596e-01 -0.5531915  1.34042553
```