CS 229 - Machine Learning
Xiangliang Zhang
Computer Science (CS)/Statistics (STAT) Program
Computer, Electrical and Mathematical Sciences & Engineering (CEMSE) Division
King Abdullah University of Science and Technology (KAUST)

HOMEWORK
I

Henrique Aparecido Laureano

Spring Semester 2018

Contents

Data

Generate a set of data samples (x, t) by yourself, e.g., $t = f(x) + \text{noise}$. x is a variable with real value in one dimension, and t is the target variable with real value in one dimension too. Function $f(x)$ can be any non-linear function, e.g., $\sin x$, $\cos x$, or more complex ones, and random noises are added.

Randomly choose 80% of the data for learning the regression function. Use the remaining 20% for evaluating the learned function.

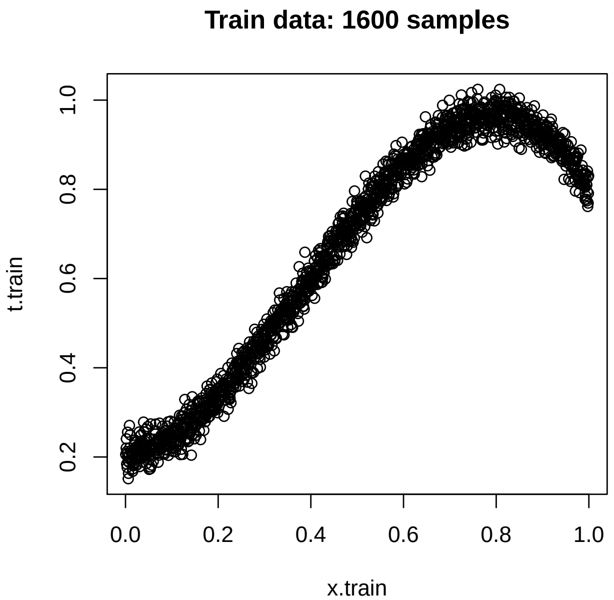
Solution:

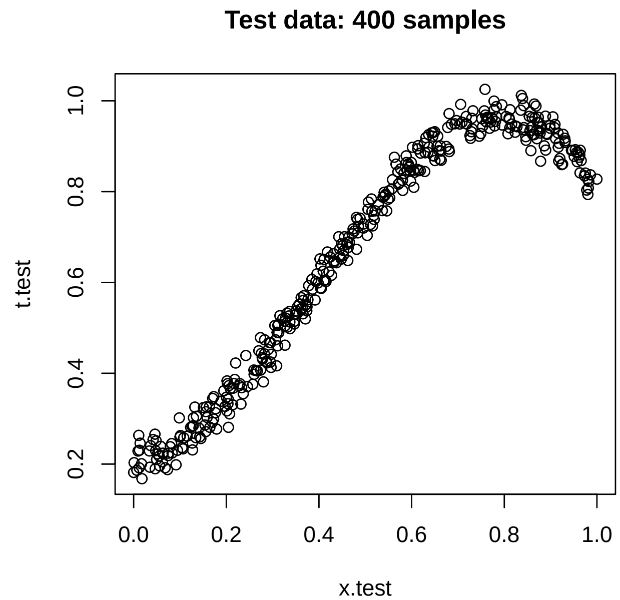
$$f(x) = 0.2 + 0.15x + 3.25x^2 - 2.8x^3, \qquad \text{(coefficient values chosen randomly)}$$

$$x: \quad \text{two thousand different values in the interval } [0, 1]$$

$$\text{noise}: \quad \text{two thousand random samples of a Normal}(0, 0.025^2)$$

```r
# <r code> =================================================================== #
                             # 2 thousand different numbers in the interval [0, 1]
x <- seq(0, 1, length.out = 2e3)

fx <- .2 + .15 * x + 3.25 * x**2 - 2.8 * x**3                # non-linear behavior

                               # adding the random noise and building the target t
t <- fx + rnorm(2e3, sd = .025)

              # sampling 1600 numbers between 0 and 2 thousand, without reposition
random.choose <- sample(2e3, 16e2)

                                            # defining the train and test datasets
t.train <- t[ random.choose]; x.train <- x[ random.choose]
t.test  <- t[-random.choose]; x.test  <- x[-random.choose]

                                                                  # plotting data
par(mfrow = c(1, 2))                                      # graphical definitions

plot(t.train ~ x.train, main = "Train data: 1600 samples")
plot(t.test  ~ x.test,  main = "Test data: 400 samples")
# </r code> ================================================================== #
```





□

Regression basis function

Choose any kind of basis function, e.g., polynomial function. Try an appropriate number of basis functions.

Solution:

Polynomial function with three and four basis: Polynomial of degree 3 (cubic) and 4.

```
# <r code> ===================================================================== #
p <- 6                                             # defining the polynomial degree
xsp <- function(p, xs) {  # building the matrix with the desired polynomial degree
  xsp <- matrix(NA, nrow = length(xs), ncol = p + 1)     # creating an empty matrix
  xsp[ , 1:2] <- cbind(1, xs)                           # first two standard columns
  for (i in 2:p) xsp[ , i + 1] <- xs**i                      # making the polynomium
  return(xsp)
}
xs3.train <- xsp(p = 3, xs = x.train)               # matrix of dimensions 1600 x 4
xs4.train <- xsp(p = 4, xs = x.train)               # matrix of dimensions 1600 x 5
xs5.train <- xsp(p = 5, xs = x.train)               # matrix of dimensions 1600 x 6
xs6.train <- xsp(p = 6, xs = x.train)               # matrix of dimensions 1600 x 7
xs3.test  <- xsp(p = 3, xs = x.test)                # matrix of dimensions  400 x 4
xs4.test  <- xsp(p = 4, xs = x.test)                # matrix of dimensions  400 x 5
xs5.test  <- xsp(p = 5, xs = x.test)                # matrix of dimensions  400 x 6
xs6.test  <- xsp(p = 6, xs = x.test)                # matrix of dimensions  400 x 7
# </r code> ==================================================================== #
```

□

Task

---

(1)

Implement the *batch gradient descent* algorithm.

<u>Solution:</u>

Algorithm:

$$
\begin{aligned}
w_i^{j+1} &:= w_i^j - \eta \frac{\partial}{\partial w_i} \left[\frac{1}{2N} \sum_{n=1}^{N} (y(x_n, w) - t_n)^2 \right] \\
&:= w_i^j - \eta \frac{1}{N} \sum_{n=1}^{N} (y(x_n, w) - t_n) x_n,
\end{aligned}
$$

with

- w being the vector of coefficients;
- η being a real constant in the interval $(0, 1]$;
- N being the sample size;
- $y(x_n, w)$ being the linear predictor, in this case $y(x_n, w) = \sum_{i=0}^{p} w_i x^i$;
- t_n being the response target variable;
- x being the covariables.

Cost function/stop criterium:

$$
\frac{1}{2N} \sum_{n=1}^{N} (y(x_n, w) - t_n)^2.
$$

```
# <r code> =================================================================== #
bgd <- function(t, x) {                                      # batch gradient descent
  t = matrix(t)          # converting the target value to matrix of dimension n x 1
  n = length(t)                                                       # sample size
  p = ncol(x)                                              # number of coefficients
```

```r
  # i keep only two vectors of coefficients in the memory. the coefficients of the
  #                                    actual step and of the immediately past step
  w = matrix(numeric(p))          # coefficients for the past step. dimension p x 1
  w.new = matrix(numeric(p))    # coefficients for the actual step. dimension p x 1
  i = 1                                      # counter for the number of iterations
  cost = 1      # cost function (convergence criterium). random value to initialize
  cf <- numeric(1)    # creating vector to keep the cost function at each iteration
  eta = .5                                                                # constant
     # establishing convergence criterium. desired difference between steps: 0.001
  while (cost > .001) {
    error = x %*% w - t                                # breaking algorithm in parts
    delta = t(x) %*% error / n                         # breaking algorithm in parts
    w.new = w - eta * delta                              # computing new coefficents
    cost = sum((x %*% w.new - t)**2) / (2 * n)             # computing cost function
    cf[i] = cost                                # keeping the iteration cost function
    w = w.new                   # at each step the new coefficient became the old one
    # print(c(i, w))            # showing number of iterations and actual coefficents
    i = i + 1                                    # updating the number of iterations
  }
      # returning final coeffients, number of iterations and cost function at each
      #                                                                     iteration
  return(list(w = t(w), i = i, cf = cf))
}
# </r code> ==================================================================== #
```

□

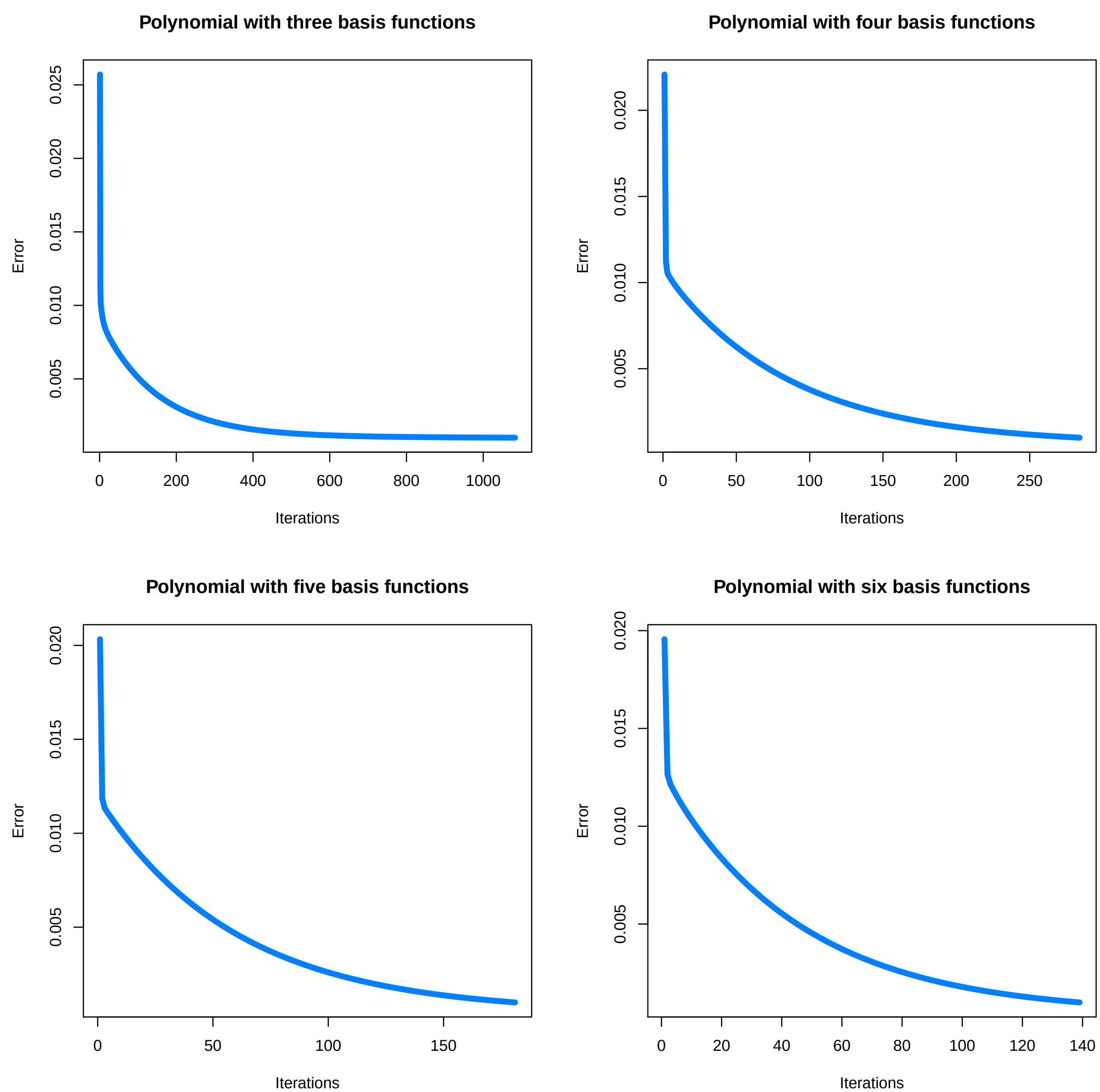
a)

Show the decreasing of error function with the increasing of iteration numbers.

Solution:

```r
# <r code> ===================================================================== #
par(mfrow = c(2, 2))                             # dividing the graphical window in four

                                           # running the bgd with a three degree polynomial
bgd.3train <- bgd(t = t.train, x = xs3.train)
plot(bgd.3train$cf, type = "l", lwd = 5, col = "#0080ff" # plotting error function
     , xlab = "Iterations", ylab = "Error"
     , main = "Polynomial with three basis functions")
                                            # running the bgd with a four degree polynomial
bgd.4train <- bgd(t = t.train, x = xs4.train)
plot(bgd.4train$cf, type = "l", lwd = 5, col = "#0080ff" # plotting error function
     , xlab = "Iterations", ylab = "Error"
     , main = "Polynomial with four basis functions")
                                            # running the bgd with a five degree polynomial
```

```
bgd.5train <- bgd(t = t.train, x = xs5.train)
plot(bgd.5train$cf, type = "l", lwd = 5, col = "#0080ff" # plotting error function
     , xlab = "Iterations", ylab = "Error"
     , main = "Polynomial with five basis functions")
                                    # running the bgd with a six degree polynomial
bgd.6train <- bgd(t = t.train, x = xs6.train)
plot(bgd.6train$cf, type = "l", lwd = 5, col = "#0080ff" # plotting error function
     , xlab = "Iterations", ylab = "Error"
     , main = "Polynomial with six basis functions")
# </r code> ==================================================================== #
```



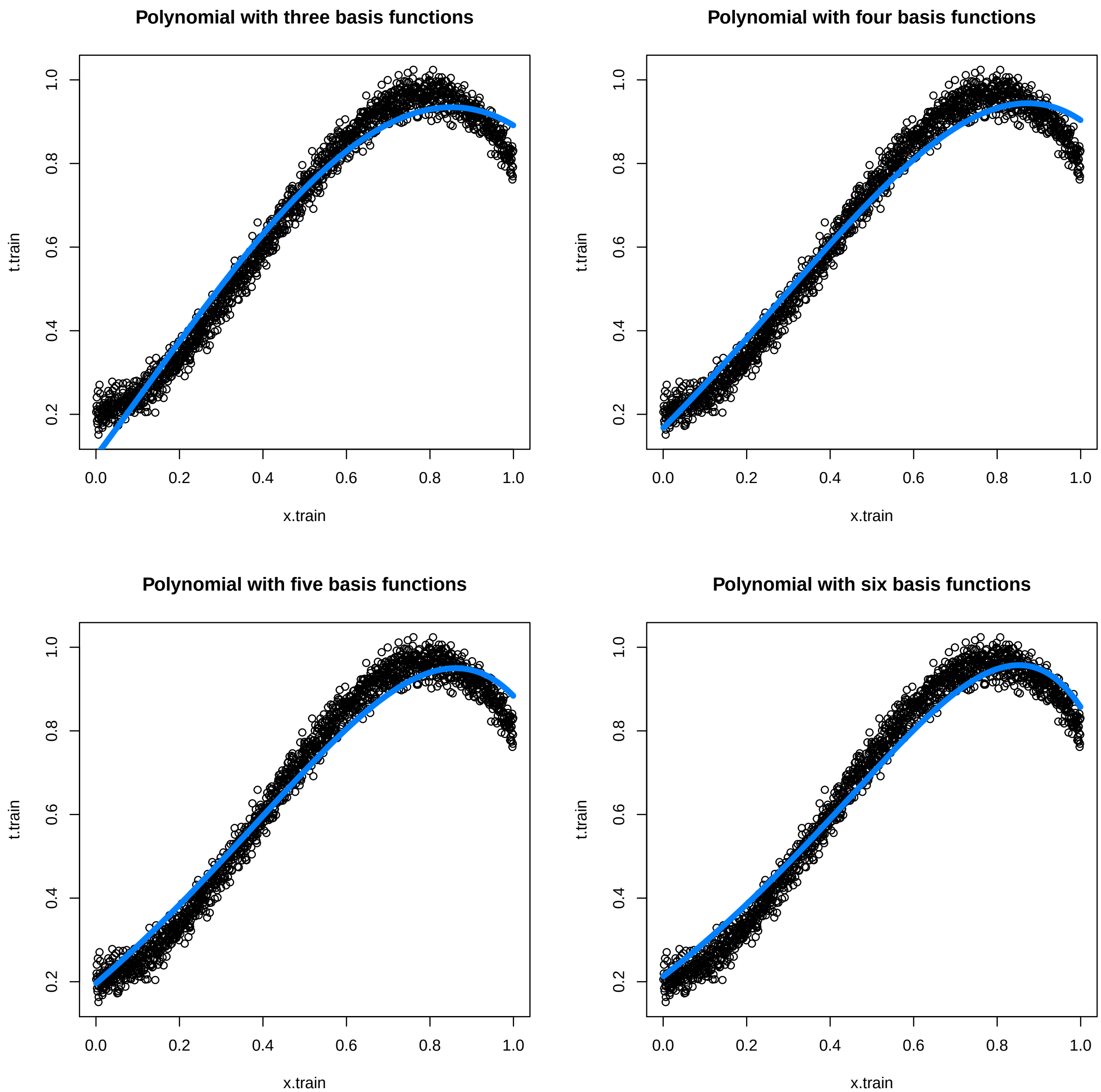
□

b)

Give the results of obtained coefficient, w.

Solution:

```
# <r code> =================================================================== #
round(bgd.3train$w, 6)  # polynomial with three basis functions: w_{0}, ..., w_{3}

         [,1]     [,2]    [,3]      [,4]
[1,] 0.099789 1.351393 0.27639 -0.836567

round(bgd.4train$w, 6)  # polynomial with four basis functions:  w_{0}, ..., w_{4}

        [,1]     [,2]     [,3]      [,4]      [,5]
[1,] 0.16735 1.005517 0.375372 -0.149181 -0.495614

round(bgd.5train$w, 6)  # polynomial with five basis functions:  w_{0}, ..., w_{5}

         [,1]    [,2]     [,3]     [,4]     [,5]      [,6]
[1,] 0.195974 0.86093 0.401727 0.025085 -0.22218 -0.378251

round(bgd.6train$w, 6)  # polynomial with six basis functions:   w_{0}, ..., w_{6}

         [,1]     [,2]     [,3]    [,4]      [,5]      [,6]      [,7]
[1,] 0.212712 0.779609 0.404252 0.09837 -0.101921 -0.228531 -0.307827

par(mfrow = c(2, 2))                       # dividing the graphical window in four
                                                             # plotting train data
plot(t.train ~ x.train, main = "Polynomial with three basis functions")
t.bgd.3train <- with(bgd.3train, w) %*% t(xs3.train)  # computing the fitted curve
                                          # inserting the fitted curve in the plot
lines(sort(x.train), t.bgd.3train[order(x.train)], col = "#0080ff", lwd = 5)
                                                             # plotting train data
plot(t.train ~ x.train, main = "Polynomial with four basis functions")
t.bgd.4train <- with(bgd.4train, w) %*% t(xs4.train)  # computing the fitted curve
                                          # inserting the fitted curve in the plot
lines(sort(x.train), t.bgd.4train[order(x.train)], col = "#0080ff", lwd = 5)
                                                             # plotting train data
plot(t.train ~ x.train, main = "Polynomial with five basis functions")
t.bgd.5train <- with(bgd.5train, w) %*% t(xs5.train)  # computing the fitted curve
                                          # inserting the fitted curve in the plot
lines(sort(x.train), t.bgd.5train[order(x.train)], col = "#0080ff", lwd = 5)
                                                             # plotting train data
plot(t.train ~ x.train, main = "Polynomial with six basis functions")
t.bgd.6train <- with(bgd.6train, w) %*% t(xs6.train)  # computing the fitted curve
                                          # inserting the fitted curve in the plot
lines(sort(x.train), t.bgd.6train[order(x.train)], col = "#0080ff", lwd = 5)
# </r code> ================================================================== #
```



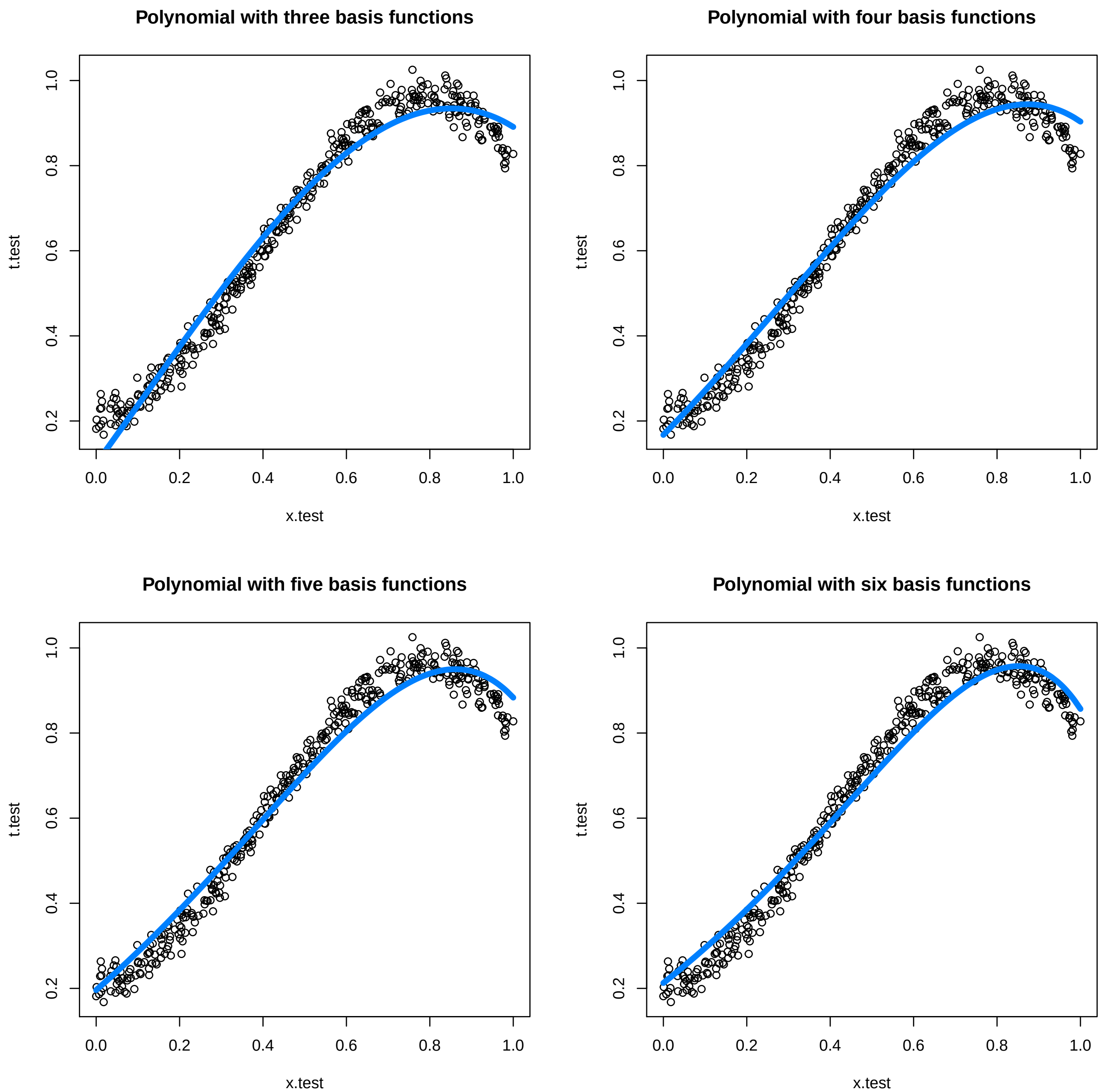
□

c)

Show the predicted $f(x)$ when applying the learned regression function on testing data x, and compare it with the corresponding actual target t on the same figure.

<u>Solution:</u>

```r
# <r code> =================================================================== #
par(mfrow = c(2, 2))                              # dividing the graphical window in four

                                                                     # plotting test data
plot(t.test ~ x.test, main = "Polynomial with three basis functions")
     # computing the curve using the coefficients estimated with the train dataset
t.bgd.3test <- with(bgd.3train, w) %*% t(xs3.test)
                                                        # inserting the curve in the plot
lines(sort(x.test), t.bgd.3test[order(x.test)], col = "#0080ff", lwd = 5)

                                                                     # plotting test data
plot(t.test ~ x.test, main = "Polynomial with four basis functions")
     # computing the curve using the coefficients estimated with the train dataset
t.bgd.4test <- with(bgd.4train, w) %*% t(xs4.test)
                                                        # inserting the curve in the plot
lines(sort(x.test), t.bgd.4test[order(x.test)], col = "#0080ff", lwd = 5)

                                                                     # plotting test data
plot(t.test ~ x.test, main = "Polynomial with five basis functions")
     # computing the curve using the coefficients estimated with the train dataset
t.bgd.5test <- with(bgd.5train, w) %*% t(xs5.test)
                                                 # inserting the fitted curve in the plot
lines(sort(x.test), t.bgd.5test[order(x.test)], col = "#0080ff", lwd = 5)

                                                                     # plotting test data
plot(t.test ~ x.test, main = "Polynomial with six basis functions")
     # computing the curve using the coefficients estimated with the train dataset
t.bgd.6test <- with(bgd.6train, w) %*% t(xs6.test)
                                                        # inserting the curve in the plot
lines(sort(x.test), t.bgd.6test[order(x.test)], col = "#0080ff", lwd = 5)
# </r code> ================================================================== #
```



$\square$

d)

What is the Root-Mean-Square Error on test set?

<u>Solution:</u>

$$\text{Root-Mean-Square (RMS) Error :} \quad E_{\text{RMS}} = \sqrt{\frac{1}{N}\sum_{n=1}^{N}(y(x_n, w^\star) - t_n)^2}$$

```r
# <r code> =============================================================== #
rmse <- function(x, w, t) {                           # root-mean-square error
  n = length(t)                                                 # sample size
  rmse = sum((x %*% t(w) - t)**2) / (2 * n)               # computing the rmse
  return(rmse)                                            # returning the rmse
}
             # rmse in the test data using a polynomial with three basis functions
             #                          (coefficients estimated with the train data)
(rmse.3test <- rmse(x = xs3.test, w = with(bgd.3train, w), t = t.test))

[1] 0.0009584382

              # rmse in the test data using a polynomial with four basis functions
              #                         (coefficients estimated with the train data)
(rmse.4test <- rmse(x = xs4.test, w = with(bgd.4train, w), t = t.test))

[1] 0.0009343287

              # rmse in the test data using a polynomial with five basis functions
              #                         (coefficients estimated with the train data)
(rmse.5test <- rmse(x = xs5.test, w = with(bgd.5train, w), t = t.test))

[1] 0.0009415637

               # rmse in the test data using a polynomial with six basis functions
               #                        (coefficients estimated with the train data)
(rmse.6test <- rmse(x = xs6.test, w = with(bgd.6train, w), t = t.test))

[1] 0.0009527641

# </r code> ============================================================== #
```

□

(2)

Implement the *stochastic gradient descent* algorithm.

<u>Solution:</u>

Algorithm:

$$
\begin{aligned}
&\text{for } n = 1 : N \\
&\quad w_i^n := w_i^{n-1} - \eta \frac{\partial}{\partial w_i} \left[\frac{1}{2} (y(x_n, w) - t_n)^2 \right] \\
&\quad\quad := w_i^{n-1} - \eta (y(x_n, w) - t_n) x_n.
\end{aligned}
$$

```r
# <r code> ================================================================ #
sgd <- function(t, x) {                            # stochastic gradient descent
  t = matrix(t)        # converting the target value to matrix of dimension n x 1
  n = length(t)                                                    # sample size
  p = ncol(x)                                           # number of coefficients
  w = matrix(numeric(p))              # storing the coefficients. dimension p x 1
  cf <- numeric(1)   # creating vector to keep the cost function at each iteration
  eta = .5                                                            # constant
  for (i in 1:n) {                          # for each observation in the data
    xn = matrix(x[i, ], ncol = p)         # selecting a line in the n x p matrix x
    error = xn %*% w - t[i]                        # breaking algorithm in parts
    delta = t(xn) %*% error                        # breaking algorithm in parts
    w = w - eta * delta                                  # computing coefficents
    cf[i] = error**2           # computing and keeping the iteration cost function
  }                # returning final coeffients and cost function at each iteration
  return(list(w = t(w), cf = cf))
}
# </r code> =============================================================== #
```

□

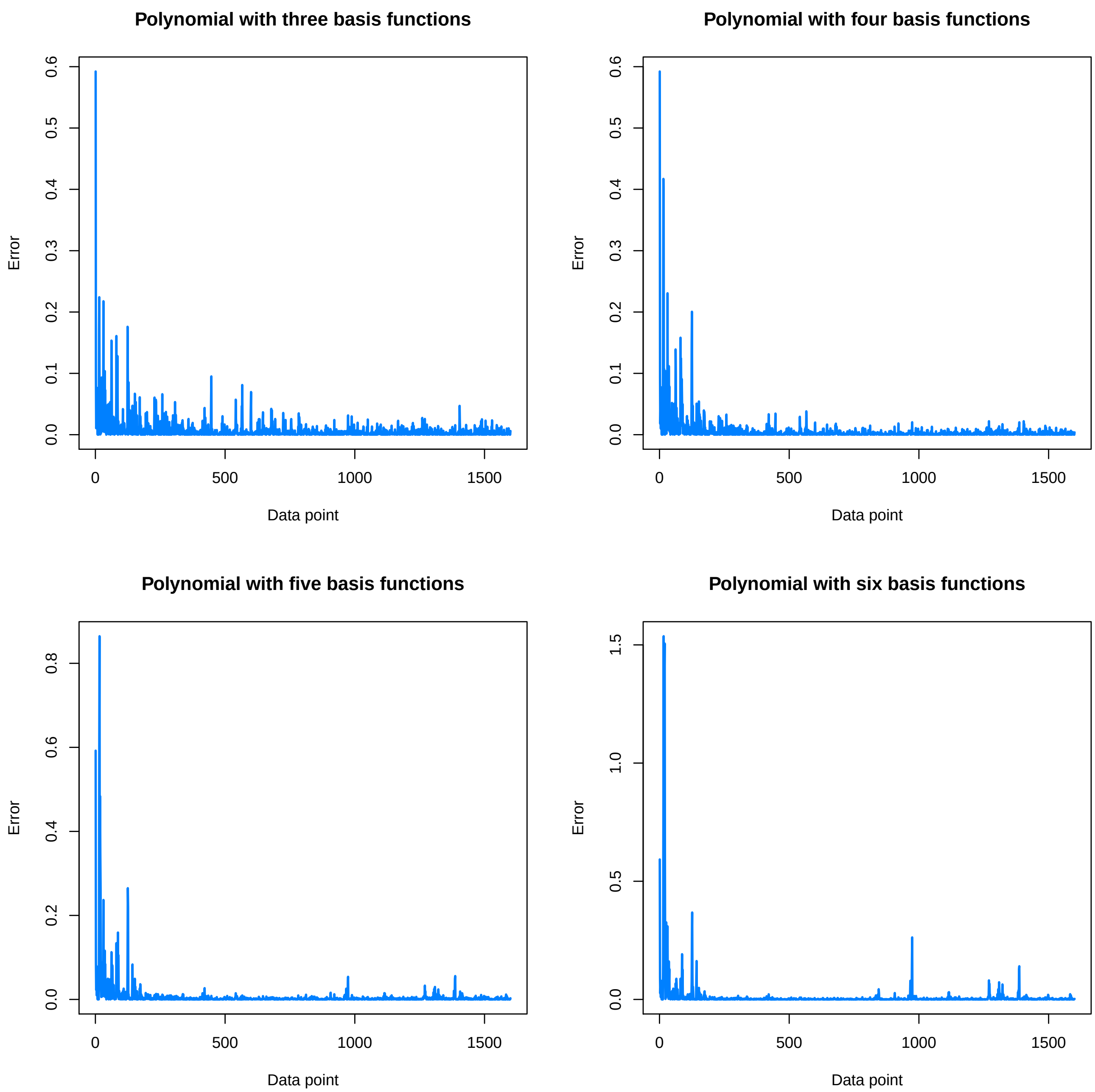
a)

Show the decreasing of error function with the increasing number of used data points.

<u>Solution:</u>

```r
# <r code> ================================================================ #
par(mfrow = c(2, 2))                      # dividing the graphical window in four
                                   # running the sgd with a three degree polynomial
sgd.3train <- sgd(t = t.train, x = xs3.train)
plot(sgd.3train$cf, type = "l", lwd = 2, col = "#0080ff" # plotting error function
     , xlab = "Data point", ylab = "Error"
     , main = "Polynomial with three basis functions")
                                    # running the sgd with a four degree polynomial
sgd.4train <- sgd(t = t.train, x = xs4.train)
plot(sgd.4train$cf, type = "l", lwd = 2, col = "#0080ff" # plotting error function
     , xlab = "Data point", ylab = "Error"
     , main = "Polynomial with four basis functions")
                                    # running the sgd with a five degree polynomial
sgd.5train <- sgd(t = t.train, x = xs5.train)
plot(sgd.5train$cf, type = "l", lwd = 2, col = "#0080ff" # plotting error function
     , xlab = "Data point", ylab = "Error"
     , main = "Polynomial with five basis functions")
                                     # running the sgd with a six degree polynomial
sgd.6train <- sgd(t = t.train, x = xs6.train)
plot(sgd.6train$cf, type = "l", lwd = 2, col = "#0080ff" # plotting error function
```

```
      , xlab = "Data point", ylab = "Error"
      , main = "Polynomial with six basis functions")
# </r code> =================================================================== #
```



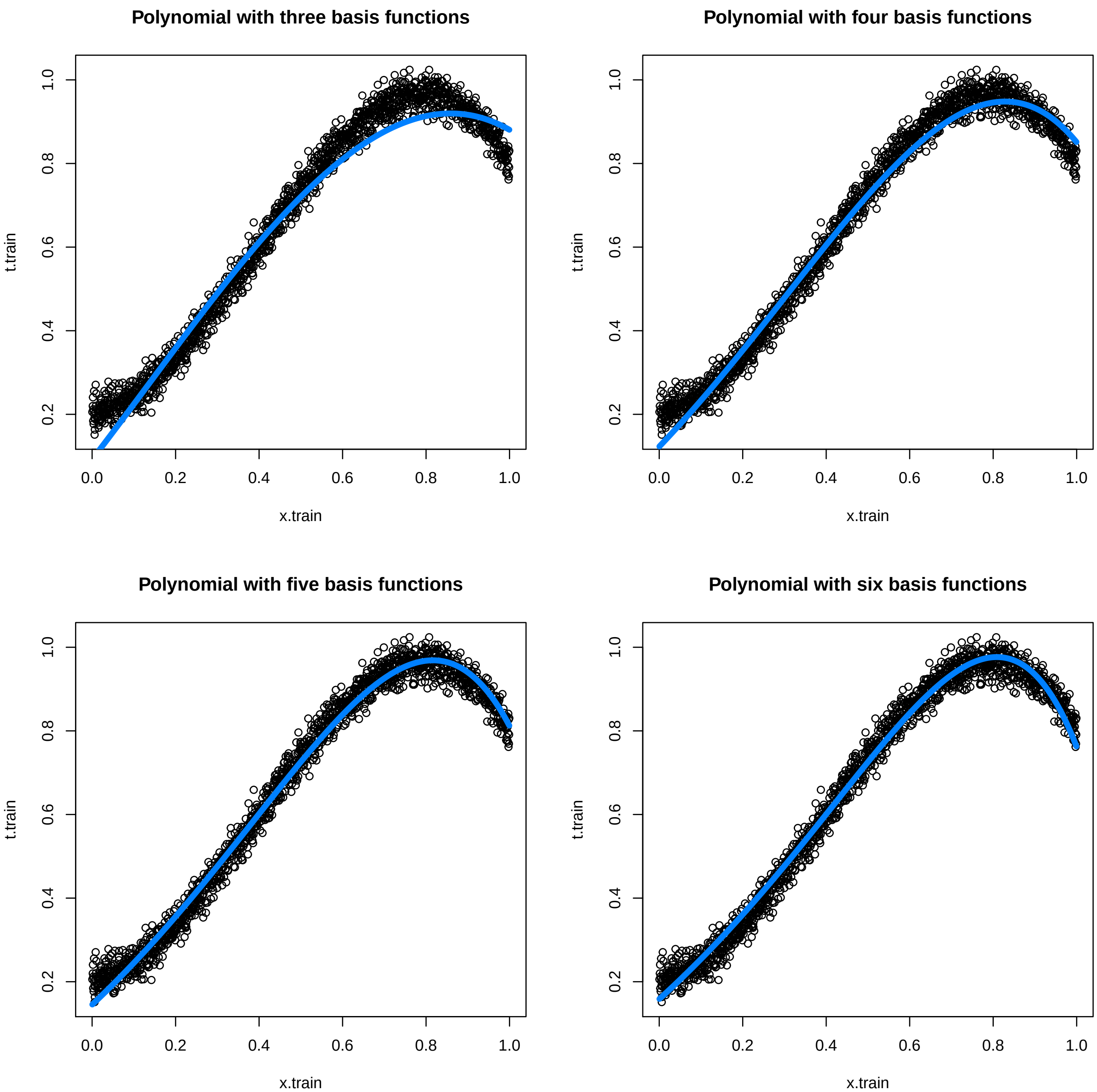
□

b)

Give the results of obtained coefficient, w.

<u>Solution:</u>

```
# <r code> ================================================================ #
round(sgd.3train$w, 6)  # polynomial with three basis functions: w_{0}, ..., w_{3}

         [,1]     [,2]    [,3]      [,4]
[1,] 0.091969 1.299038 0.34072 -0.851316

round(sgd.4train$w, 6)  # polynomial with four basis functions:  w_{0}, ..., w_{4}

         [,1]     [,2]     [,3]      [,4]      [,5]
[1,] 0.122751 1.049928 0.569441 -0.157787 -0.733624

round(sgd.5train$w, 6)  # polynomial with five basis functions:  w_{0}, ..., w_{5}

        [,1]     [,2]     [,3]     [,4]      [,5]     [,6]
[1,] 0.14513 0.941967 0.565228 0.051273 -0.324984 -0.56695

round(sgd.6train$w, 6)  # polynomial with six basis functions:   w_{0}, ..., w_{6}

         [,1]     [,2]     [,3]     [,4]      [,5]      [,6]      [,7]
[1,] 0.158402 0.906452 0.519173 0.095223 -0.177157 -0.330536 -0.410002

par(mfrow = c(2, 2))                        # dividing the graphical window in four

                                                              # plotting train data
plot(t.train ~ x.train, main = "Polynomial with three basis functions")
t.sgd.3train <- with(sgd.3train, w) %*% t(xs3.train)  # computing the fitted curve
                                          # inserting the fitted curve in the plot
lines(sort(x.train), t.sgd.3train[order(x.train)], col = "#0080ff", lwd = 5)

                                                              # plotting train data
plot(t.train ~ x.train, main = "Polynomial with four basis functions")
t.sgd.4train <- with(sgd.4train, w) %*% t(xs4.train)  # computing the fitted curve
                                          # inserting the fitted curve in the plot
lines(sort(x.train), t.sgd.4train[order(x.train)], col = "#0080ff", lwd = 5)

                                                              # plotting train data
plot(t.train ~ x.train, main = "Polynomial with five basis functions")
t.sgd.5train <- with(sgd.5train, w) %*% t(xs5.train)  # computing the fitted curve
                                          # inserting the fitted curve in the plot
lines(sort(x.train), t.sgd.5train[order(x.train)], col = "#0080ff", lwd = 5)

                                                              # plotting train data
plot(t.train ~ x.train, main = "Polynomial with six basis functions")
t.sgd.6train <- with(sgd.6train, w) %*% t(xs6.train)  # computing the fitted curve
                                          # inserting the fitted curve in the plot
lines(sort(x.train), t.sgd.6train[order(x.train)], col = "#0080ff", lwd = 5)
# </r code> =============================================================== #
```

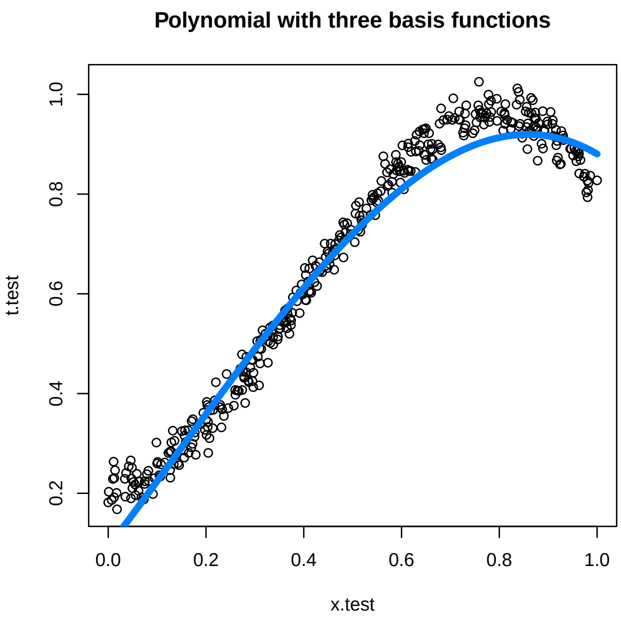


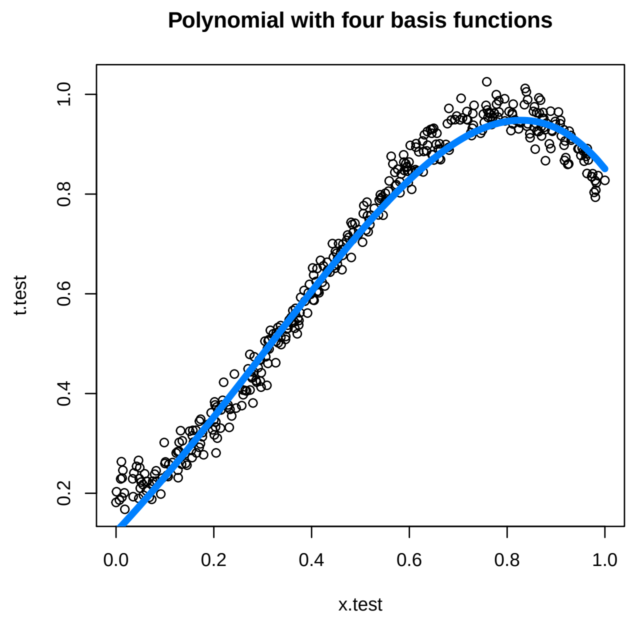
c)

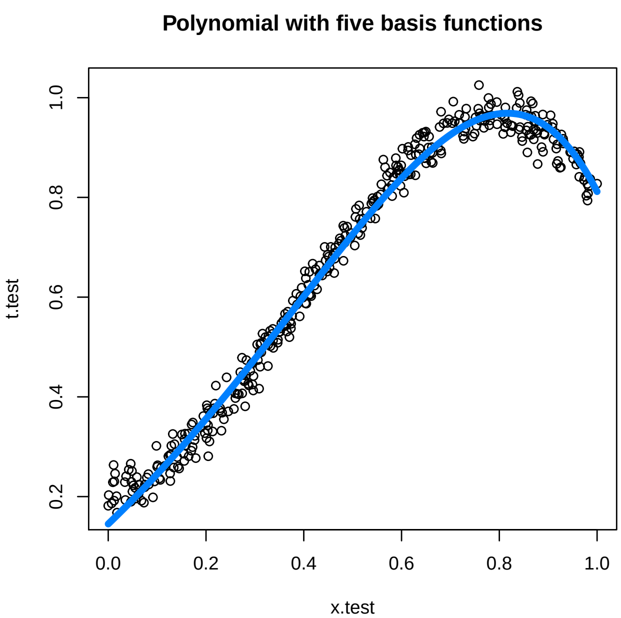
Show the predicted $f(x)$ when applying the learned regression function on testing data x, and compare it with the corresponding actual target t on the same figure.

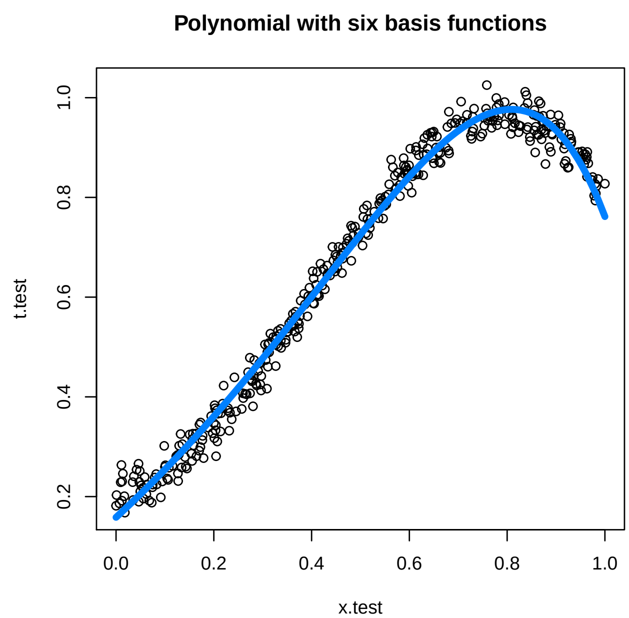
<u>Solution:</u>

```r
# <r code> ================================================================== #
par(mfrow = c(2, 2))                           # dividing the graphical window in four

                                                                # plotting test data
plot(t.test ~ x.test, main = "Polynomial with three basis functions")
     # computing the curve using the coefficients estimated with the train dataset
t.sgd.3test <- with(sgd.3train, w) %*% t(xs3.test)
                                                 # inserting the curve in the plot
lines(sort(x.test), t.sgd.3test[order(x.test)], col = "#0080ff", lwd = 5)

                                                                # plotting test data
plot(t.test ~ x.test, main = "Polynomial with four basis functions")
     # computing the curve using the coefficients estimated with the train dataset
t.sgd.4test <- with(sgd.4train, w) %*% t(xs4.test)
                                                 # inserting the curve in the plot
lines(sort(x.test), t.sgd.4test[order(x.test)], col = "#0080ff", lwd = 5)

                                                                # plotting test data
plot(t.test ~ x.test, main = "Polynomial with five basis functions")
     # computing the curve using the coefficients estimated with the train dataset
t.sgd.5test <- with(sgd.5train, w) %*% t(xs5.test)
                                          # inserting the fitted curve in the plot
lines(sort(x.test), t.sgd.5test[order(x.test)], col = "#0080ff", lwd = 5)

                                                                # plotting test data
plot(t.test ~ x.test, main = "Polynomial with six basis functions")
     # computing the curve using the coefficients estimated with the train dataset
t.sgd.6test <- with(sgd.6train, w) %*% t(xs6.test)
                                                 # inserting the curve in the plot
lines(sort(x.test), t.sgd.6test[order(x.test)], col = "#0080ff", lwd = 5)
# </r code> ================================================================= #
```









d)

What is the Root-Mean-Square Error on test set?

<u>Solution:</u>

```
# <r code> ===================================================================== #
             # rmse in the test data using a polynomial with three basis functions
             #                          (coefficients estimated with the train data)
```

```r
(rmse.3test <- rmse(x = xs3.test, w = with(sgd.3train, w), t = t.test))
```

```
[1] 0.001072366
```

```r
              # rmse in the test data using a polynomial with four basis functions
              #                        (coefficients estimated with the train data)
(rmse.4test <- rmse(x = xs4.test, w = with(sgd.4train, w), t = t.test))
```

```
[1] 0.0005841753
```

```r
              # rmse in the test data using a polynomial with five basis functions
              #                        (coefficients estimated with the train data)
(rmse.5test <- rmse(x = xs5.test, w = with(sgd.5train, w), t = t.test))
```

```
[1] 0.000424704
```

```r
               # rmse in the test data using a polynomial with six basis functions
               #                       (coefficients estimated with the train data)
(rmse.6test <- rmse(x = xs6.test, w = with(sgd.6train, w), t = t.test))
```

```
[1] 0.0004092639
```

```r
# </r code> ==================================================================== #
```

□

(3)

Implement the *maximum likelihood* algorithm.

<u>Solution:</u>

The least squares estimator by maximum likelihood is given by

$$w = (\Phi^\top \Phi)^{-1} \Phi^\top t, \quad (\text{ all vectors/matrices}).$$

```r
# <r code> ===================================================================== #
ls <- function(x, t) {              # least squares estimator by maximum likelihood
  w = solve(t(x) %*% x) %*% t(x) %*% t                    # computing the coefficients
  return(list(w = t(w)))                         # returning the estimated coefficients
}
# </r code> ==================================================================== #
```

□

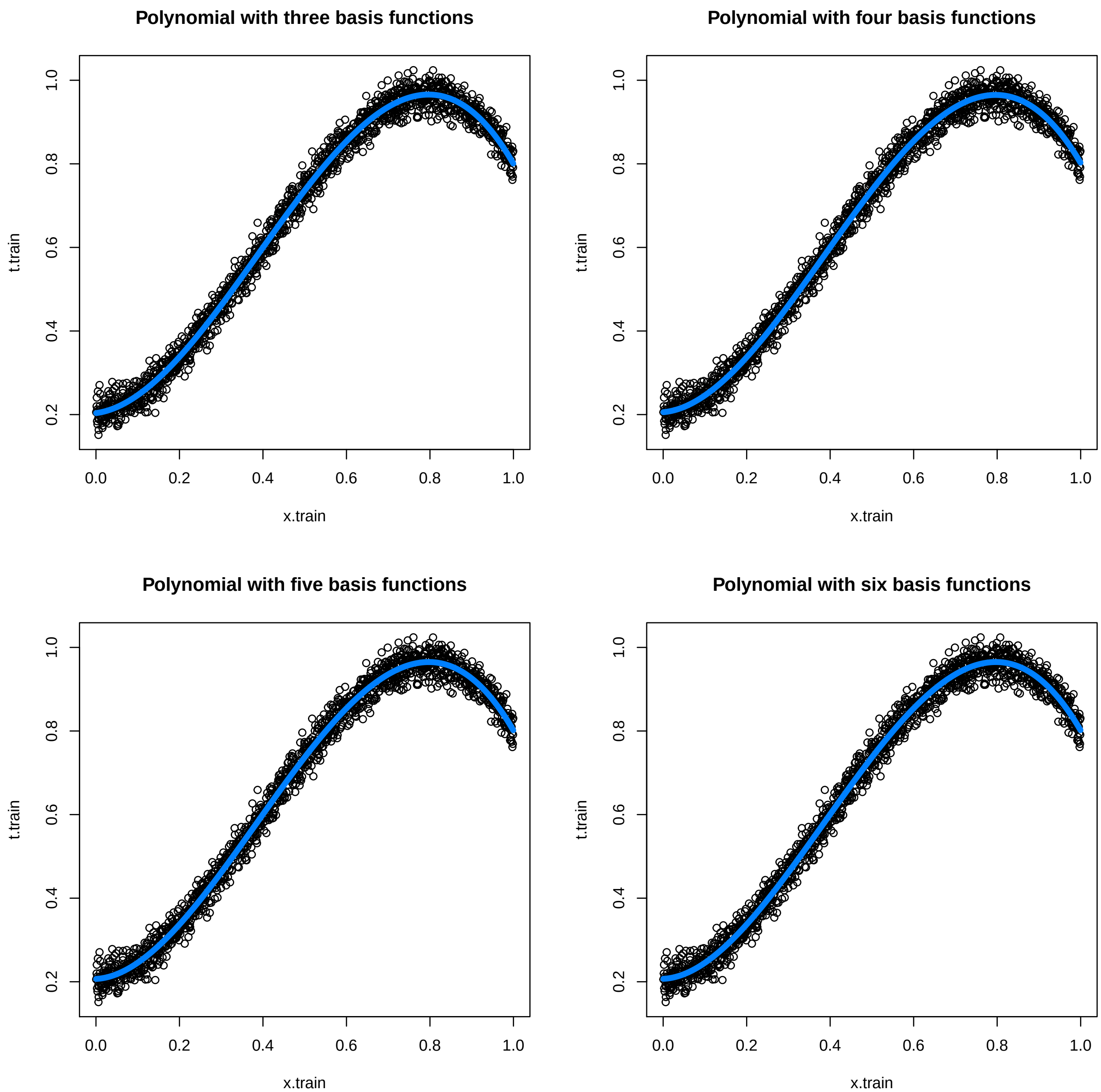
a)

Give the results of obtained coefficient, w.

Solution:

```
# <r code> =================================================================== #
                        # polynomial with three basis functions: w_{0}, ..., w_{3}
(ls.3train <- ls(x = xs3.train, t = t.train))

$w
          [,1]      [,2]     [,3]      [,4]
[1,] 0.2035217 0.1299265 3.276339 -2.809832

                         # polynomial with four basis functions: w_{0}, ..., w_{4}
(ls.4train <- ls(x = xs4.train, t = t.train))

$w
          [,1]       [,2]     [,3]      [,4]      [,5]
[1,] 0.2057053 0.08615743 3.473318 -3.115967 0.1528609

                         # polynomial with five basis functions: w_{0}, ..., w_{5}
(ls.5train <- ls(x = xs5.train, t = t.train))

$w
          [,1]       [,2]     [,3]      [,4]      [,5]      [,6]
[1,] 0.2067511 0.05471622 3.693799 -3.704164 0.8144303 -0.264468

                          # polynomial with six basis functions: w_{0}, ..., w_{6}
(ls.6train <- ls(x = xs6.train, t = t.train))

$w
         [,1]       [,2]     [,3]      [,4]     [,5]      [,6]      [,7]
[1,] 0.207054 0.04202729 3.820568 -4.210595 1.762703 -1.097852 0.2774456

par(mfrow = c(2, 2))                      # dividing the graphical window in four
                                                            # plotting train data
plot(t.train ~ x.train, main = "Polynomial with three basis functions")
t.ls.3train <- with(ls.3train, w) %*% t(xs3.train)     # computing the fitted curve
                                              # inserting the fitted curve in the plot
lines(sort(x.train), t.ls.3train[order(x.train)], col = "#0080ff", lwd = 5)

                                                            # plotting train data
plot(t.train ~ x.train, main = "Polynomial with four basis functions")
t.ls.4train <- with(ls.4train, w) %*% t(xs4.train)     # computing the fitted curve
                                              # inserting the fitted curve in the plot
lines(sort(x.train), t.ls.4train[order(x.train)], col = "#0080ff", lwd = 5)

                                                            # plotting train data
```

```r
plot(t.train ~ x.train, main = "Polynomial with five basis functions")
t.ls.5train <- with(ls.5train, w) %*% t(xs5.train)    # computing the fitted curve
                                                    # inserting the fitted curve in the plot
lines(sort(x.train), t.ls.5train[order(x.train)], col = "#0080ff", lwd = 5)

                                                                   # plotting train data
plot(t.train ~ x.train, main = "Polynomial with six basis functions")
t.ls.6train <- with(ls.6train, w) %*% t(xs6.train)    # computing the fitted curve
                                                    # inserting the fitted curve in the plot
lines(sort(x.train), t.ls.6train[order(x.train)], col = "#0080ff", lwd = 5)
# </r code> =================================================================== #
```



□

b)

Compare the value of w obtained at (1) and (2) by gradient descent algorithm, and (3) by maximum likelihood.

Solution:

Polynomial with three basis functions:

```
# <r code> =================================================================== #
matrix(cbind(bgd.3train$w, sgd.3train$w, ls.3train$w), ncol = 3
       , dimnames = list(c("w0", "w1", "w2", "w3"), c("bgd", "sgd", "ls")))
# </r code> ================================================================== #
```

```
           bgd         sgd         ls
w0  0.09978855  0.09196927  0.2035217
w1  1.35139335  1.29903803  0.1299265
w2  0.27638962  0.34071953  3.2763390
w3 -0.83656720 -0.85131641 -2.8098323
```

Batch and stochastic gradient descent present similar coefficients. Maximum likelihood present bigger estimatives, principally for w_2 and w_3, corresponding respectively to the terms of second and third degree.

Polynomial with four basis functions:

```
# <r code> =================================================================== #
matrix(cbind(bgd.4train$w, sgd.4train$w, ls.4train$w), ncol = 3
       , dimnames = list(c("w0", "w1", "w2", "w3", "w4"), c("bgd", "sgd", "ls")))
# </r code> ================================================================== #
```

```
          bgd        sgd          ls
w0  0.1673501  0.1227514  0.20570531
w1  1.0055175  1.0499278  0.08615743
w2  0.3753722  0.5694413  3.47331782
w3 -0.1491813 -0.1577872 -3.11596659
w4 -0.4956139 -0.7336245  0.15286094
```

Same behavior. Similar estimatives for batch and stochastic gradient descent, and different values for maximum likelihood.

Polynomial with five basis functions:

```
# <r code> =================================================================== #
matrix(cbind(bgd.5train$w, sgd.5train$w, ls.5train$w), ncol = 3
       , dimnames = list(c("w0", "w1", "w2", "w3", "w4", "w5")
                         , c("bgd", "sgd", "ls")))
# </r code> ================================================================== #
```

```
           bgd         sgd          ls
w0  0.19597420  0.14513017  0.20675113
w1  0.86093012  0.94196741  0.05471622
w2  0.40172691  0.56522773  3.69379917
w3  0.02508541  0.05127282 -3.70416385
w4 -0.22217979 -0.32498357  0.81443029
w5 -0.37825104 -0.56695026 -0.26446802
```

The same. The more different coefficients are observed with the maximum likelihood approach.

Polynomial with six basis functions:

```r
# <r code> =============================================================== #
matrix(cbind(bgd.6train$w, sgd.6train$w, ls.6train$w), ncol = 3
       , dimnames = list(c("w0", "w1", "w2", "w3", "w4", "w5", "w6")
                         , c("bgd", "sgd", "ls")))
# </r code> ============================================================== #
```

```
           bgd         sgd          ls
w0  0.21271228  0.15840160  0.20705400
w1  0.77960924  0.90645217  0.04202729
w2  0.40425244  0.51917307  3.82056784
w3  0.09837045  0.09522338 -4.21059501
w4 -0.10192089 -0.17715659  1.76270316
w5 -0.22853128 -0.33053631 -1.09785154
w6 -0.30782721 -0.41000158  0.27744560
```

The same. The more different coefficients are observed with the maximum likelihood approach.

In summary:

Looking to the fitted curves in the train dataset presented before, the best fits are seen with the maximum likelihood approach. Very similar results are obtained when comparing the batch with the stochastic gradient descent.

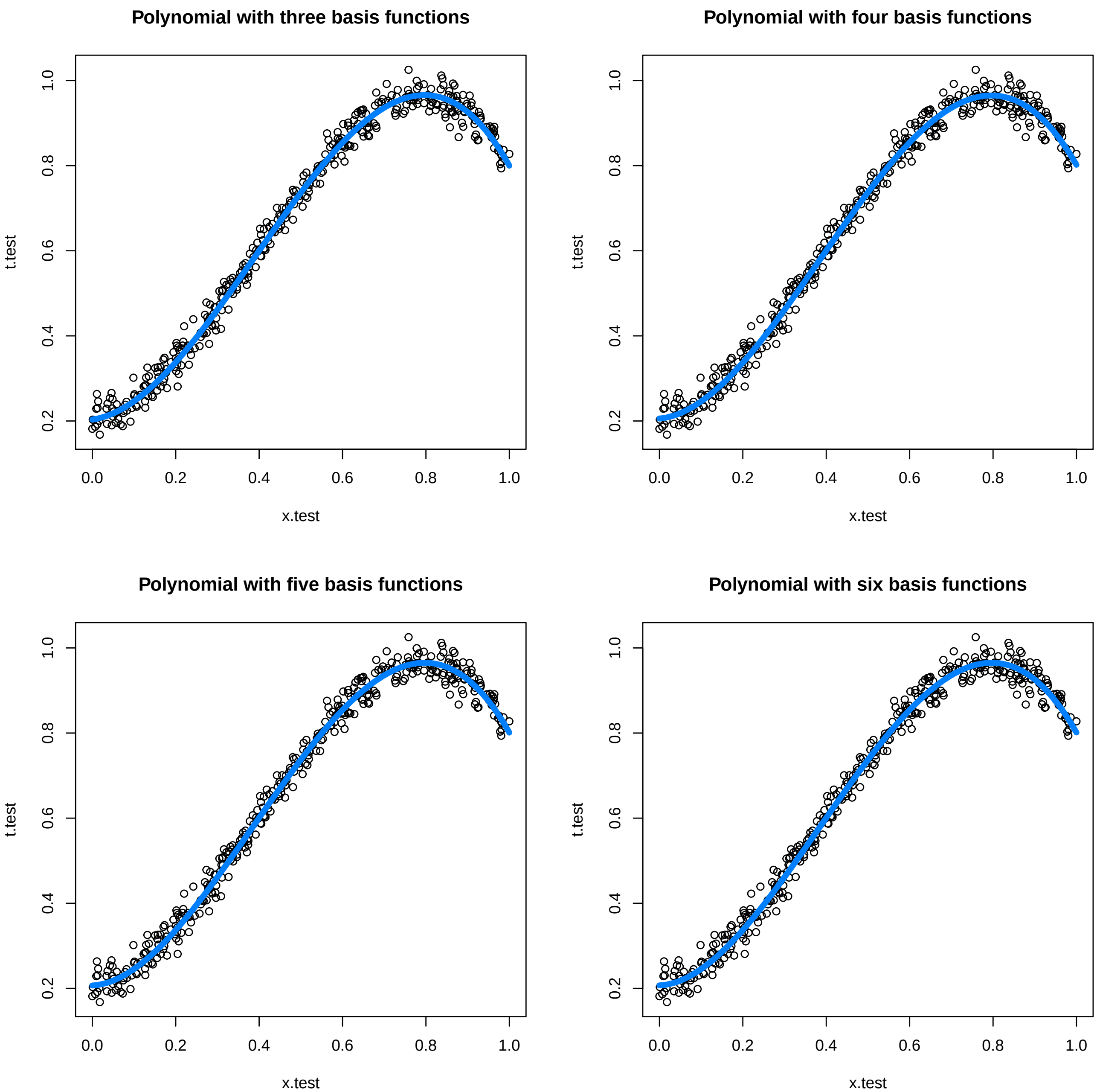
□

c)

Show the predicted $f(x)$ when applying the learned regression function on testing data x, and compare it with the corresponding actual target t on the same figure.

Solution:

```r
# <r code> ============================================================== #
par(mfrow = c(2, 2))                       # dividing the graphical window in four

                                                              # plotting test data
plot(t.test ~ x.test, main = "Polynomial with three basis functions")
     # computing the curve using the coefficients estimated with the train dataset
t.ls.3test <- with(ls.3train, w) %*% t(xs3.test)
                                                 # inserting the curve in the plot
lines(sort(x.test), t.ls.3test[order(x.test)], col = "#0080ff", lwd = 5)

                                                              # plotting test data
plot(t.test ~ x.test, main = "Polynomial with four basis functions")
     # computing the curve using the coefficients estimated with the train dataset
t.ls.4test <- with(ls.4train, w) %*% t(xs4.test)
                                                 # inserting the curve in the plot
lines(sort(x.test), t.ls.4test[order(x.test)], col = "#0080ff", lwd = 5)

                                                              # plotting test data
plot(t.test ~ x.test, main = "Polynomial with five basis functions")
     # computing the curve using the coefficients estimated with the train dataset
t.ls.5test <- with(ls.5train, w) %*% t(xs5.test)
                                          # inserting the fitted curve in the plot
lines(sort(x.test), t.ls.5test[order(x.test)], col = "#0080ff", lwd = 5)

                                                              # plotting test data
plot(t.test ~ x.test, main = "Polynomial with six basis functions")
     # computing the curve using the coefficients estimated with the train dataset
t.ls.6test <- with(ls.6train, w) %*% t(xs6.test)
                                                 # inserting the curve in the plot
lines(sort(x.test), t.ls.6test[order(x.test)], col = "#0080ff", lwd = 5)
# </r code> ============================================================= #
```



□

d)

What is the Root-Mean-Square Error on test set? Which method has the best performance? Batch gradient descent, stochastic gradient descent or maximum likelihood?

<u>Solution:</u>

```
# <r code> =================================================================== #
             # rmse in the test data using a polynomial with three basis functions
             #                        (coefficients estimated with the train data)
(rmse.3test <- rmse(x = xs3.test, w = with(ls.3train, w), t = t.test))

[1] 0.0003058933

              # rmse in the test data using a polynomial with four basis functions
              #                        (coefficients estimated with the train data)
(rmse.4test <- rmse(x = xs4.test, w = with(ls.4train, w), t = t.test))

[1] 0.000305442

              # rmse in the test data using a polynomial with five basis functions
              #                        (coefficients estimated with the train data)
(rmse.5test <- rmse(x = xs5.test, w = with(ls.5train, w), t = t.test))

[1] 0.0003056077

               # rmse in the test data using a polynomial with six basis functions
               #                       (coefficients estimated with the train data)
(rmse.6test <- rmse(x = xs6.test, w = with(ls.6train, w), t = t.test))

[1] 0.0003055881

# </r code> ================================================================== #
```

Looking only to the RMSE the method with best performance is the maximum likelihood, with average RMSE of 0.0003. Nevertheless, the RMSE obtained with the gradient descent (batch and stochastic) are also pretty similar (0.0009 and 0.0004, in average).

Looking to the fitted curves (in the train and in the test datasets), the best results are obtained with the maximum likelihood approach.

The data was generated with a cubic polynomial added of a random noise. Four different degrees of polynomials was tested and the bests results was obtained with the polynomial of degree six (the highest of the four), this in all the three approachs.

■