Monte Carlo Simulation of 5-fold Cross Validation Mean Squared Error

Anuradha Ramachandran

Consider the following data generating processes:
$DGP1 : Y = -2I(X < 0) + 2I(X < 0) + 4\epsilon$
$DGP2 : Y = 1 + 3X - 2X^2 + X^3 - 0.5X^4 + 4\epsilon$
$DGP3 : Y = sin(\frac{\pi}{2}X) + 4\epsilon$
$DGP4 : Y = 4sin(3\pi X)I(X > 0) + 4\epsilon$
where X is uniformly distribued $\in [-1, 1]$ and ϵ is a standard Gaussian random variable.

For each DGP, the following actions are reported: Calculate the in-sample MSE and the 5-fold cross-validated MSE for polynomial regression (using OLS) of orders d = 1, 2, . . ., 20. In-sample mean squared error and the cross-validated MSE is plotted as a function of d and the order of the polynomial regression selected by cross-validation is also reported. The best polynomial is fitted in the full data, and the estimated function, together with the data, as well as the true function is plotted. This procedure is performed for 100 and 10000 samples.
Note: Code for all these sections can be found in the appendix.

Results for n=100
As the degree of model complexity increases the CV MSE for 100 samples is very large but the in sample MSE value is not as high as the one predicted by CV MSE. The best polynomial fit predicted by 5 fold cross validation method for
DGP1 :2
DGP2 :3
DGP3 :1
DGP4: 6

In sample MSE and 5-fold CV MSE as a function of degree of polynomial upto 20 for all 4 DGP and the their approximation is shown below:

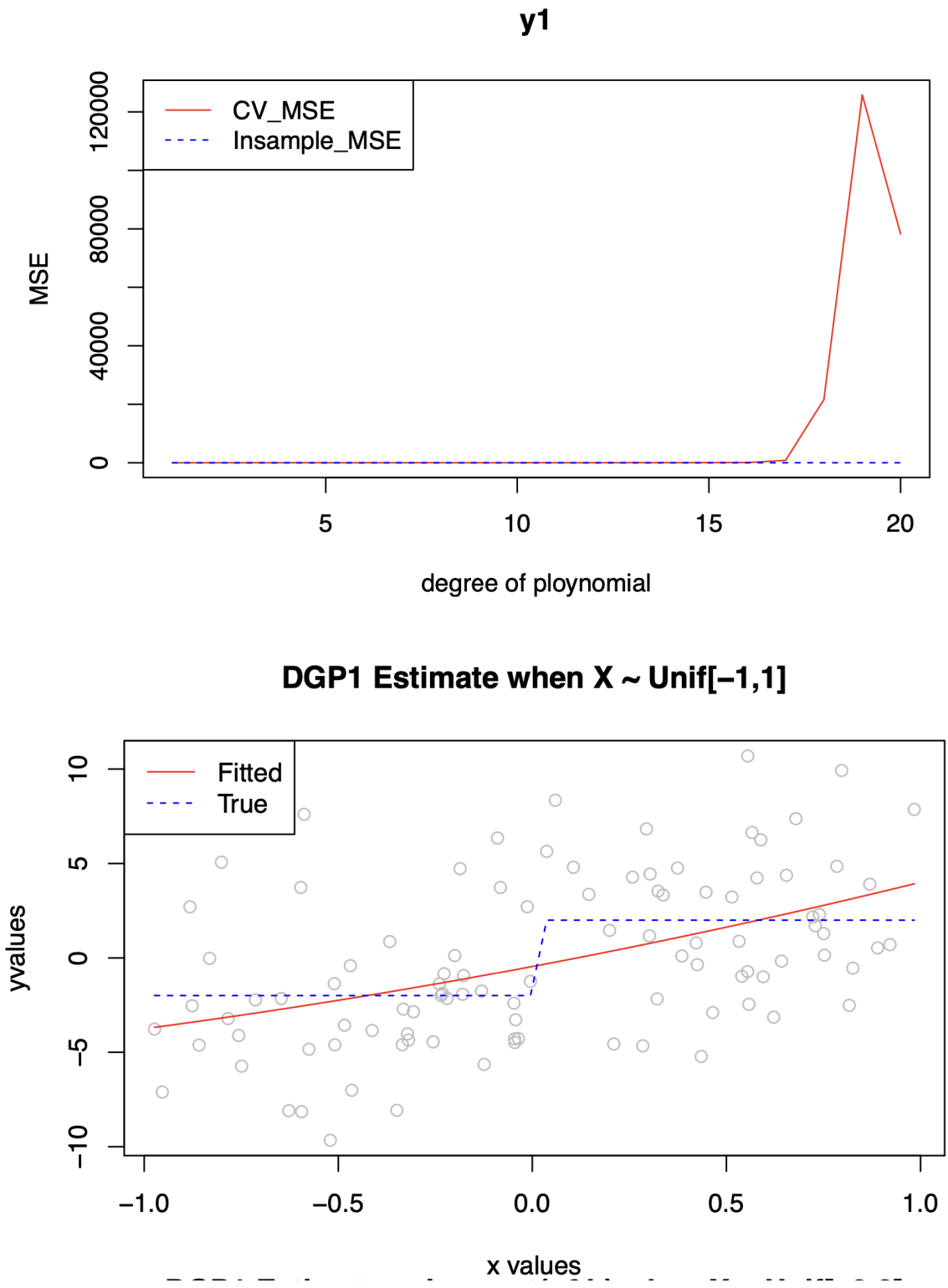


Figure 1: dgp1_est

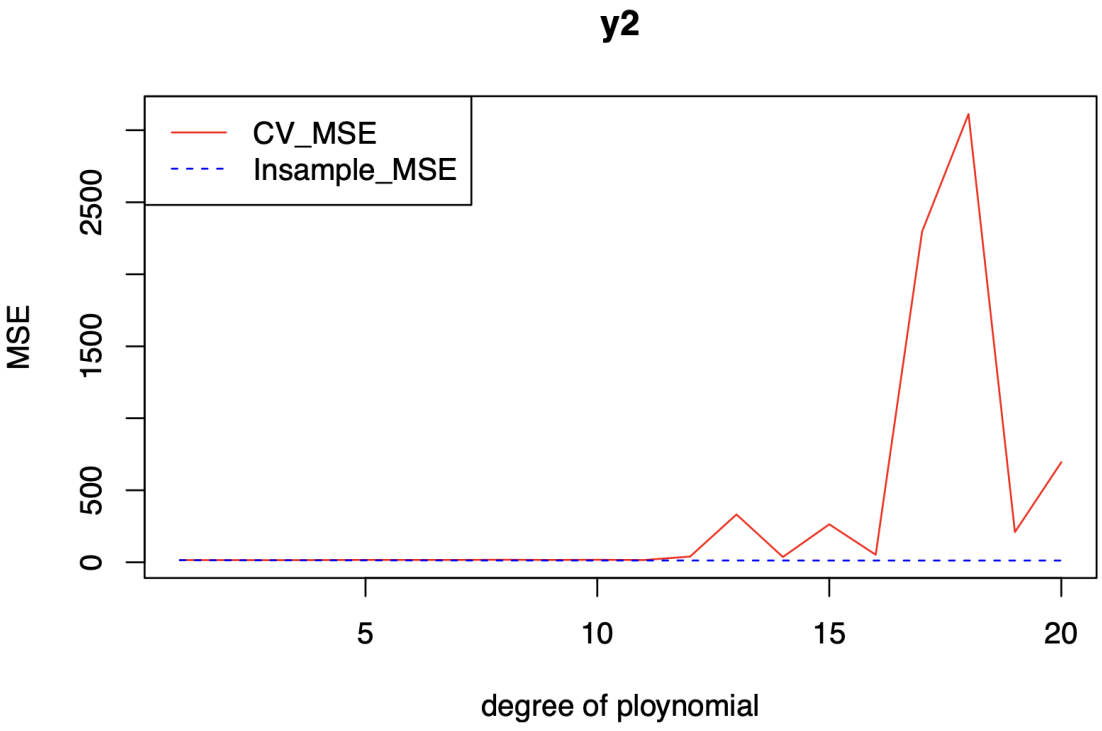


Figure 2: dpg2

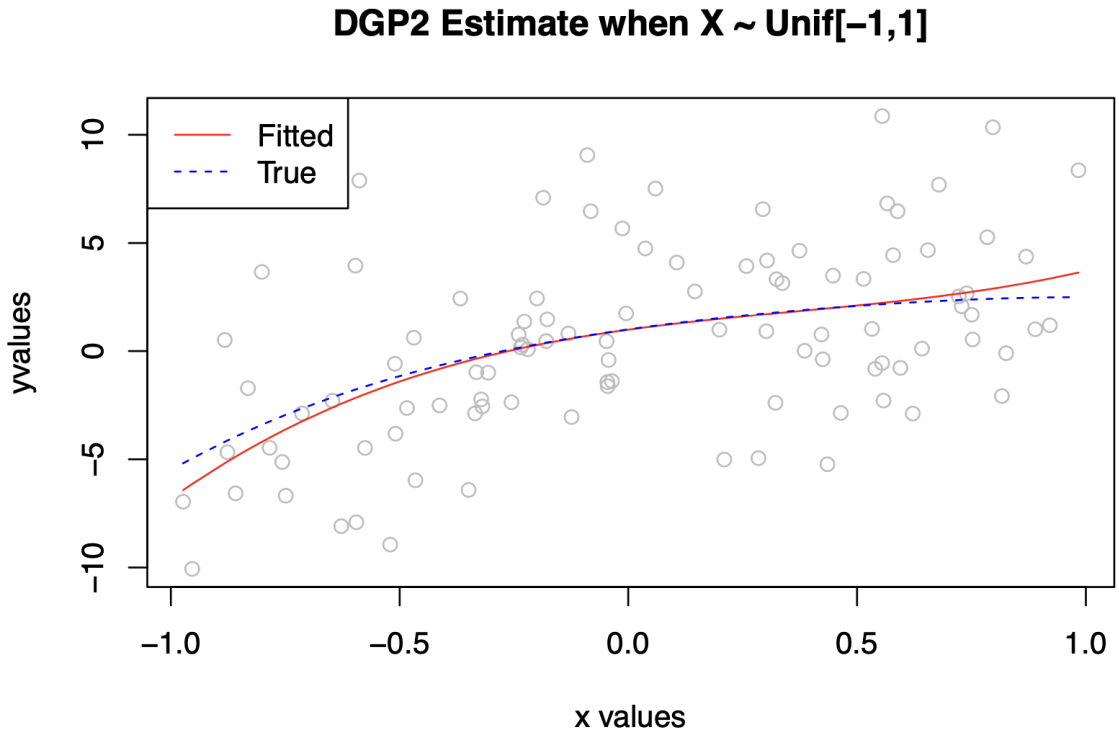


Figure 3: dgp2_est

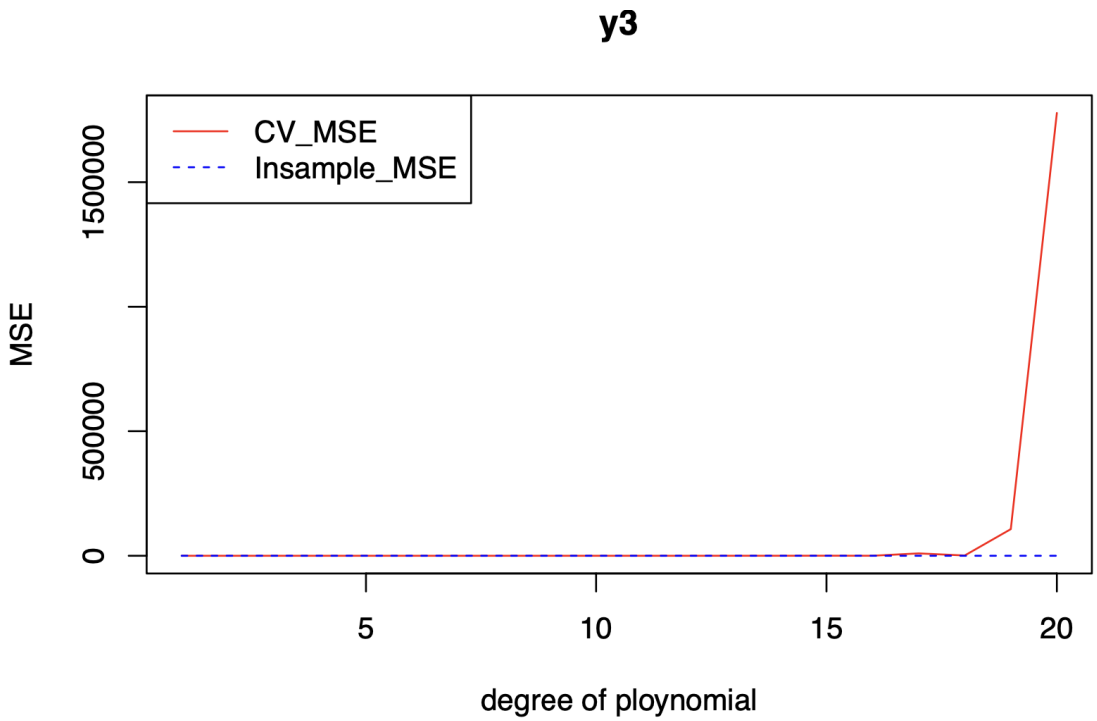


Figure 4: dpg3

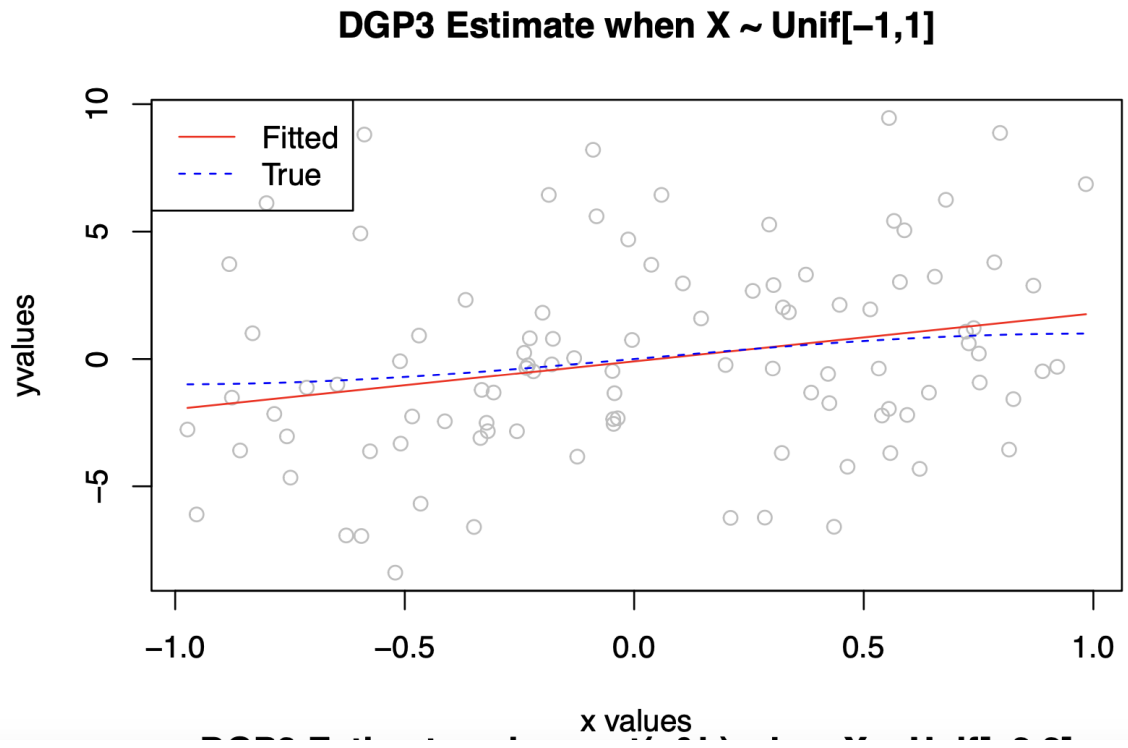
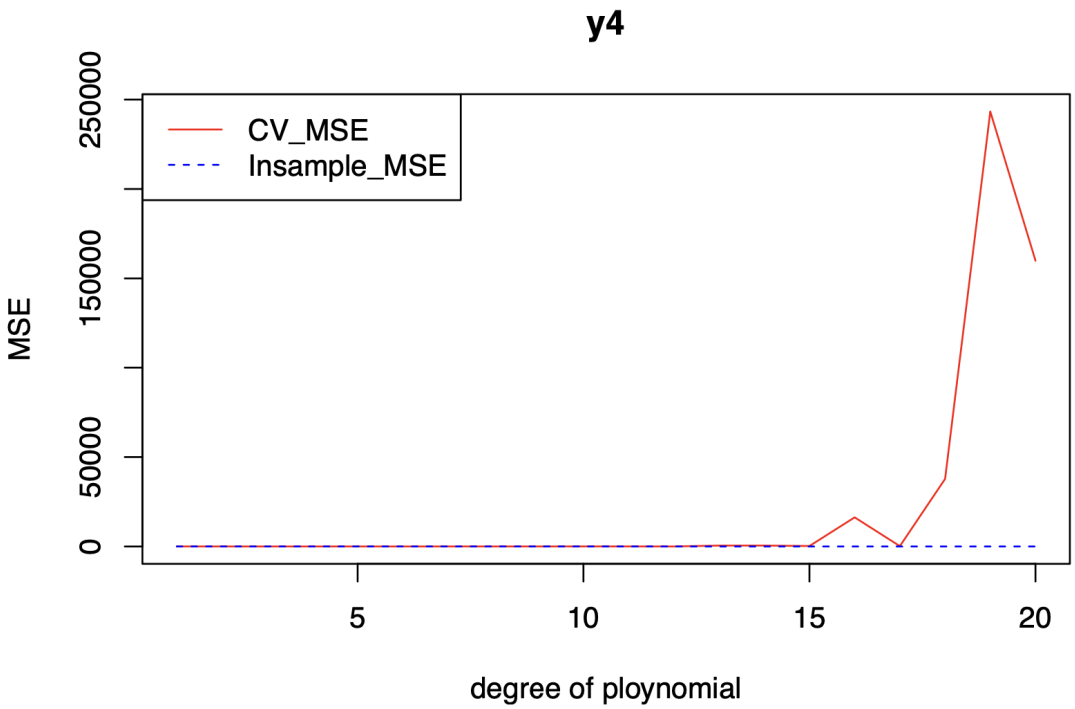
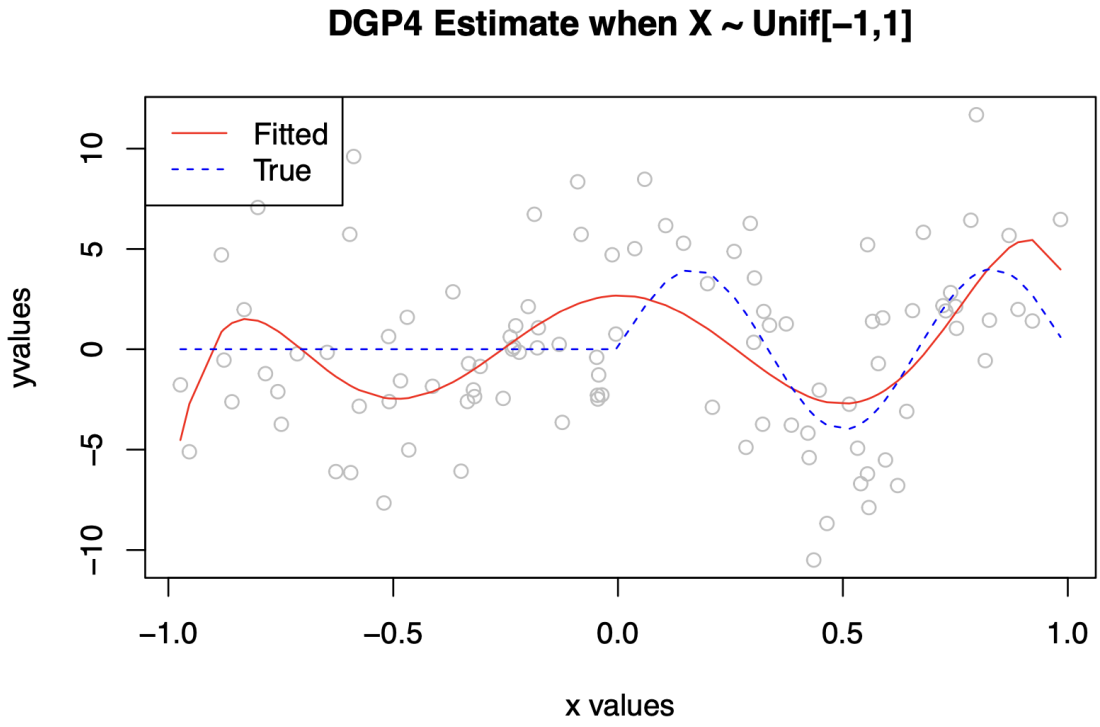


Figure 5: dpg3_est





Results for n=10000

The degree of polynomial selected for all simulated data from dgp1 to 4 are different (higher in most cases) than the one selected by cross validation approach when n=100.

For DGP1: For n= 100 the best polynomial fit is: 2

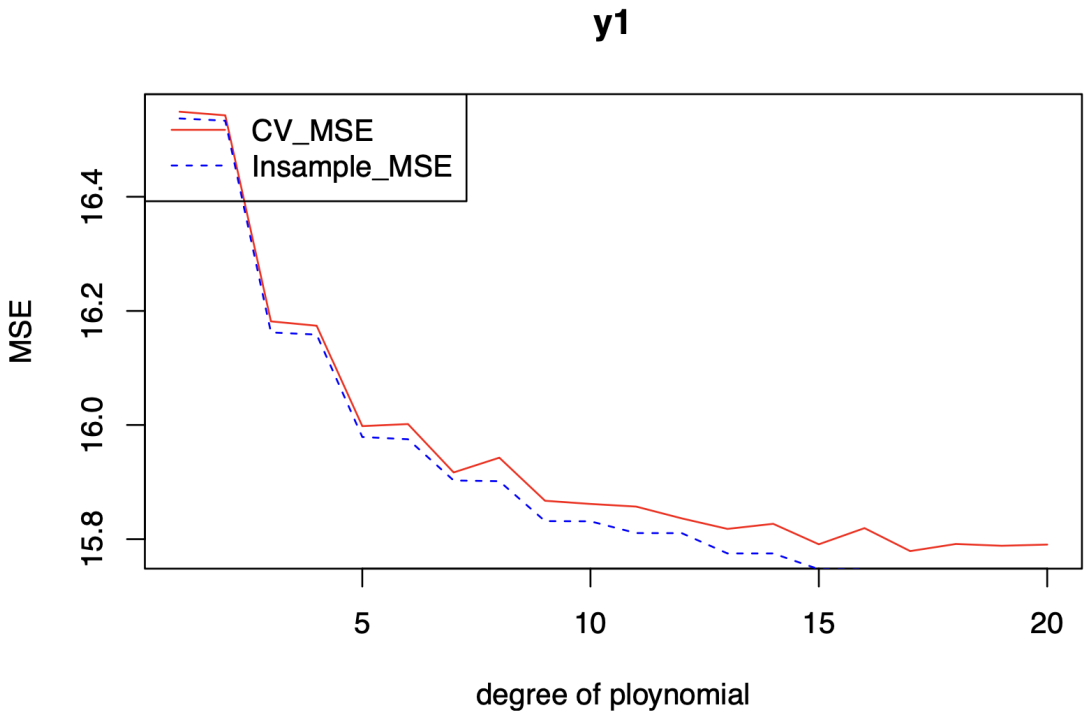
For n= 10000 the best polynomial fit is: 17

For DGP2:

For n= 100 the best polynomial fit is: 3
For n= 10000 the best polynomial fit is: 3
For DGP3:
For n= 100 the best polynomial fit is: 1
For n= 10000 the best polynomial fit is: 2
For DGP4: For n= 100 the best polynomial fit is: 6
For n= 10000 the best polynomial fit is: 20

The CV-MSE has decreased in magnitude as n=10000 and is more stable as the degree of complexity of the model increases. Since the sample size is large, the data is scattered much more compared to n=100 case and hence the variance of the estimate might decrease as the degree of polynomial increases since it tries to fit the training data perfectly. The CV-MSE and in sample sample are very similar in all the cases.

In sample MSE and 5-fold CV MSE as a function of degree of polynomial upto 20 for all 4 DGP and the their approximation is shown below:



Can these predictions be generalized well in populations where X lies outside the range $\in [-1, 1]$
As seen from the plots below, the model trained on uniform[-1,1] dataset does not estimate the data appropriately from uniform[-2,2] distribution. For DGP2 and DGP3 it seems the model seems to work till certain values of X out of [-1,1] range. However it starts to diverge as x gets far away from this range where the model was trained. From the plots it can be seen that beyond X $\in [-1, 1]$, the fitted line starts to diverge from the true line for data generated from unif $[-2, 2]$ distribution. These following plots were generated for a sample size of 10000 where X belong to Uniform distribution $\in [-2, 2]$

Appendix

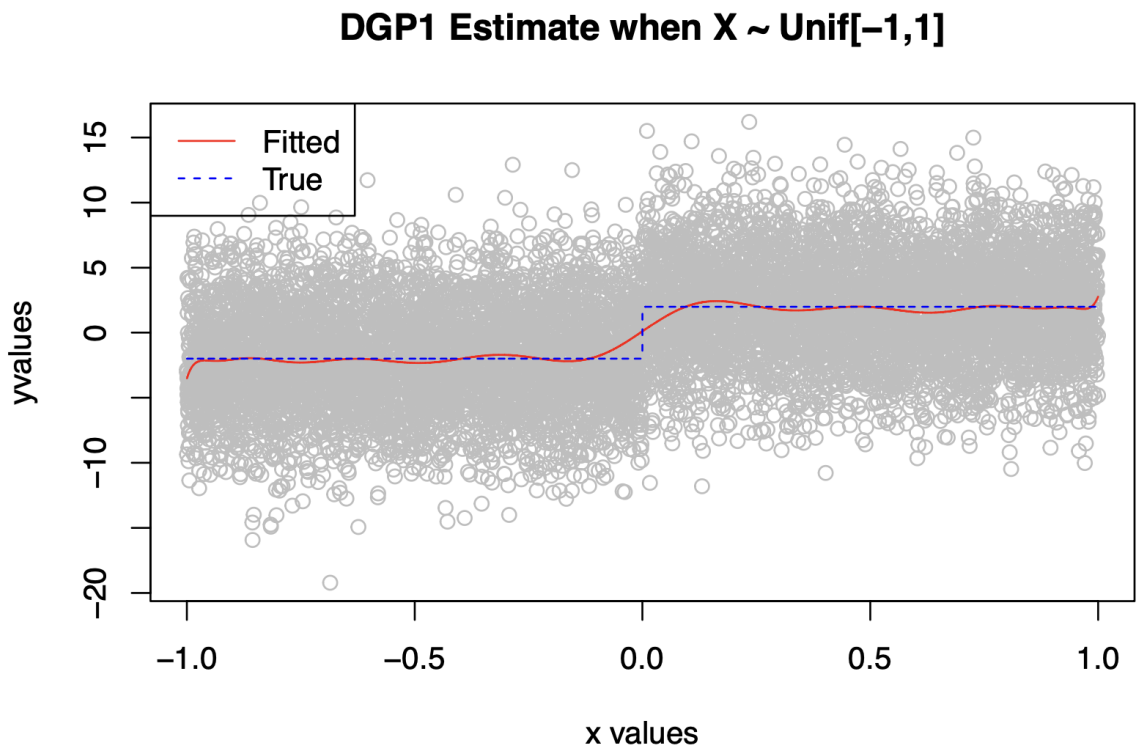


Figure 6: p1e

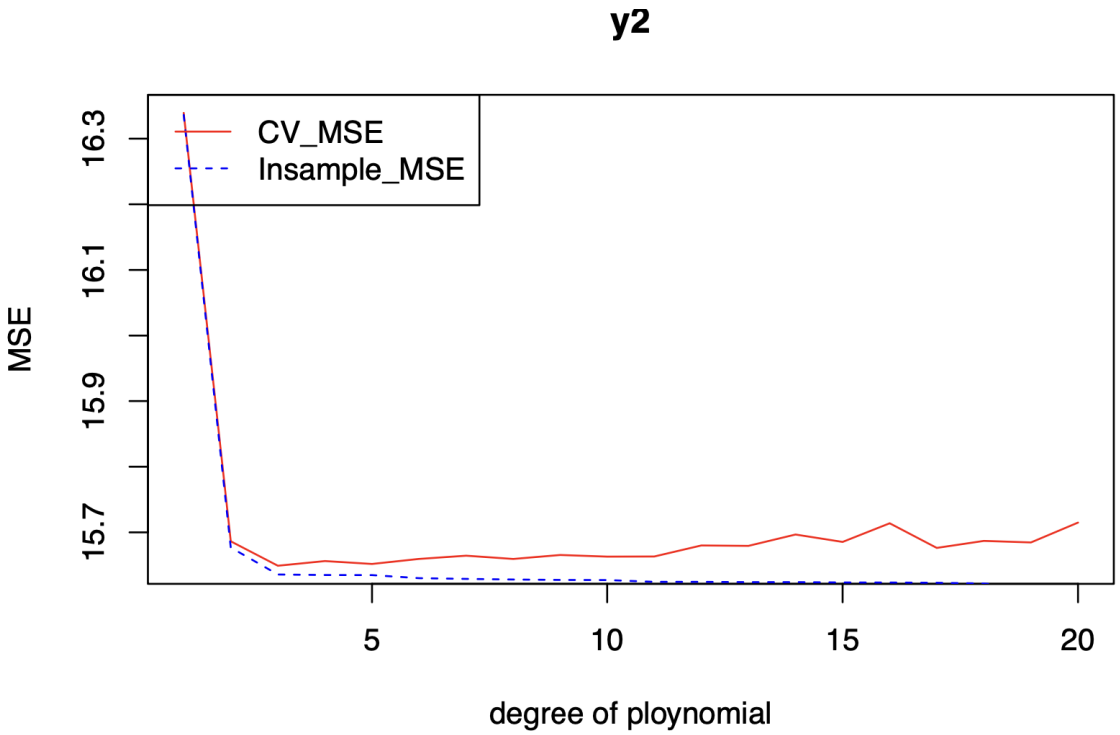


Figure 7: p2

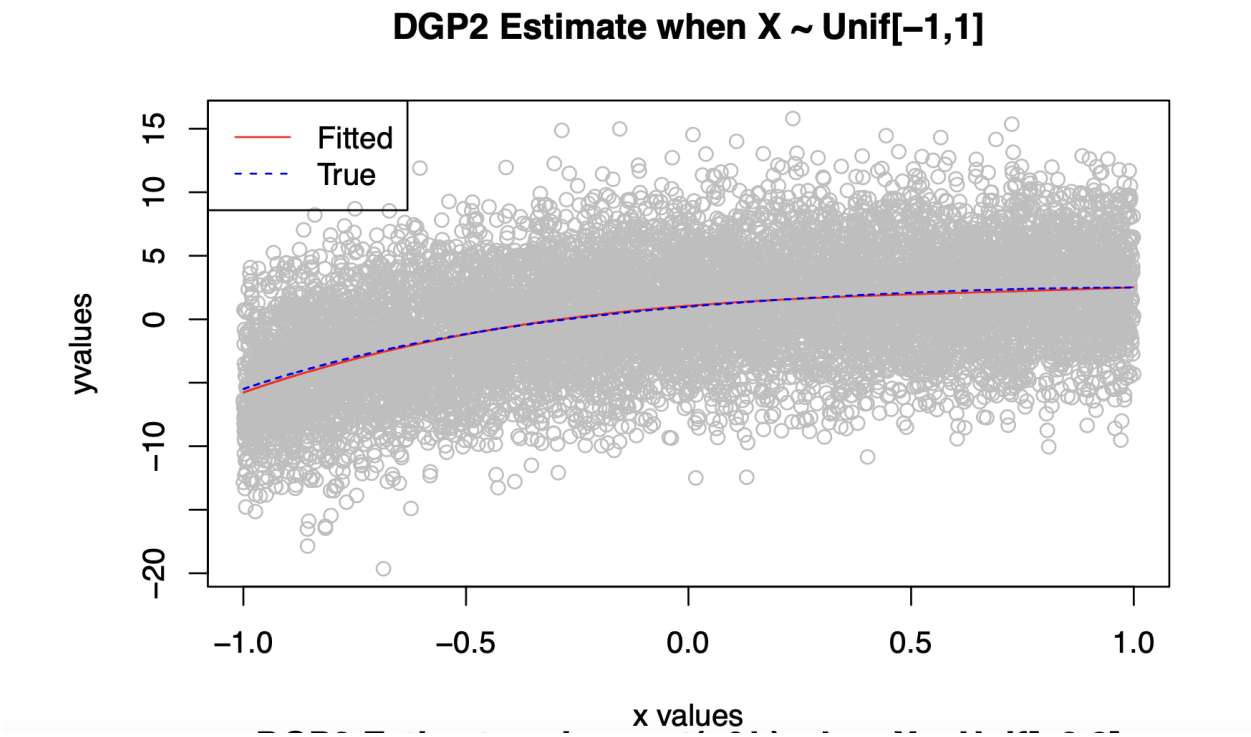


Figure 8: p2e

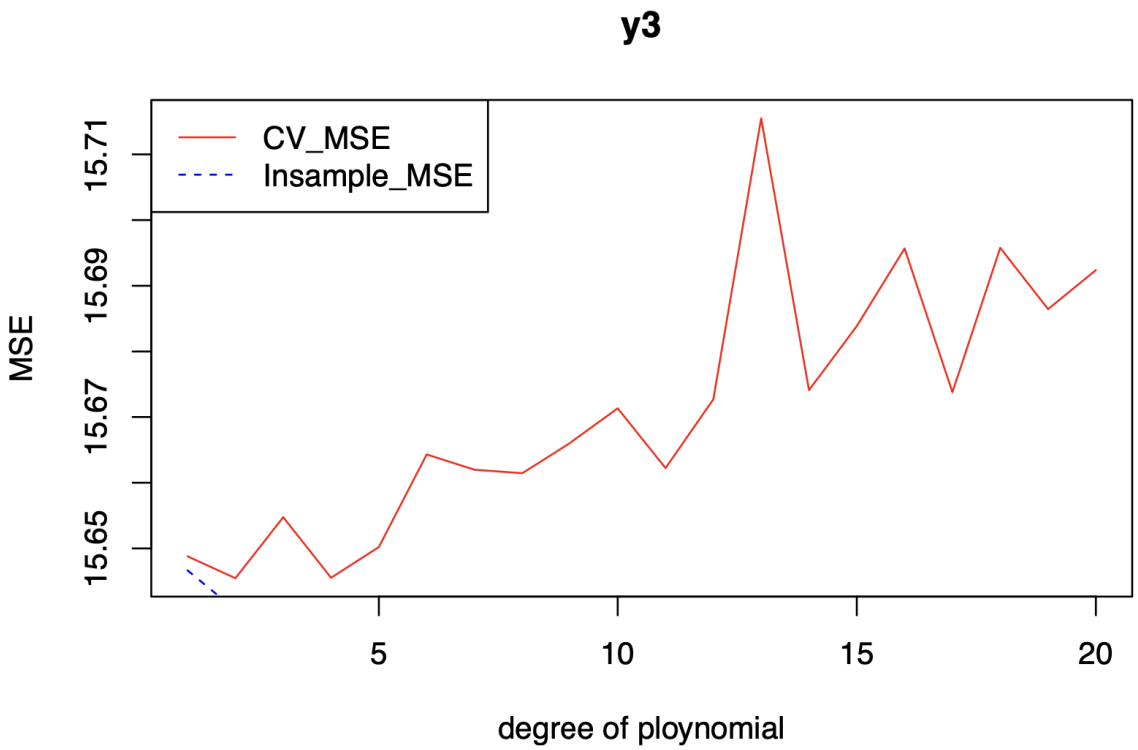


Figure 9: p3

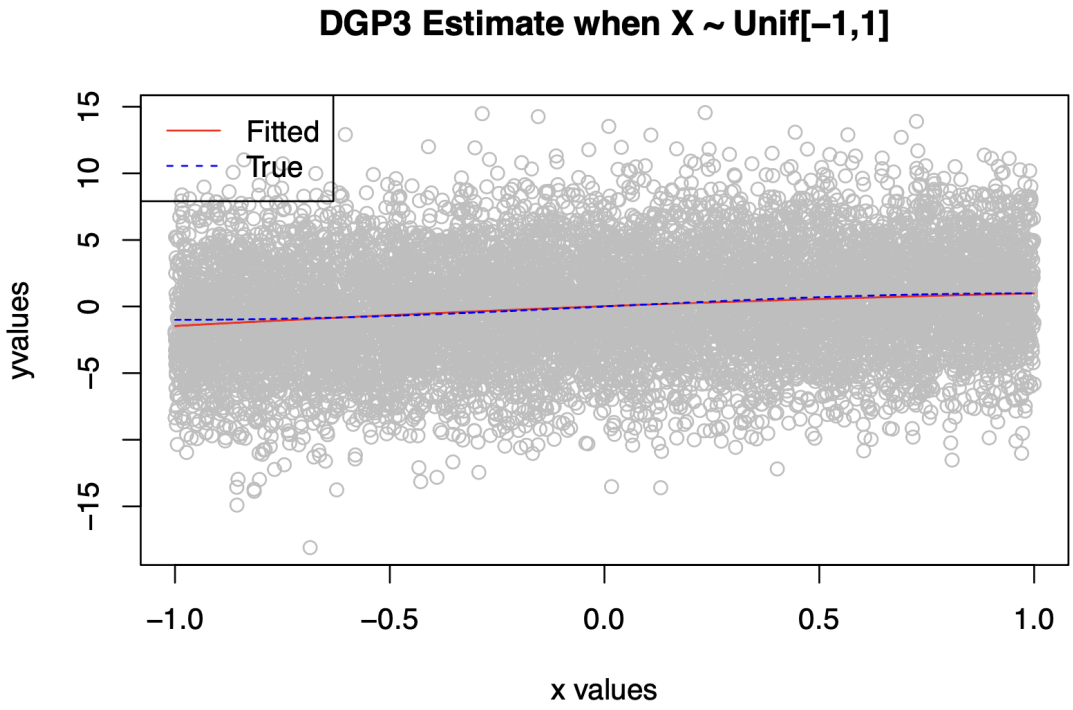


Figure 10: p3e

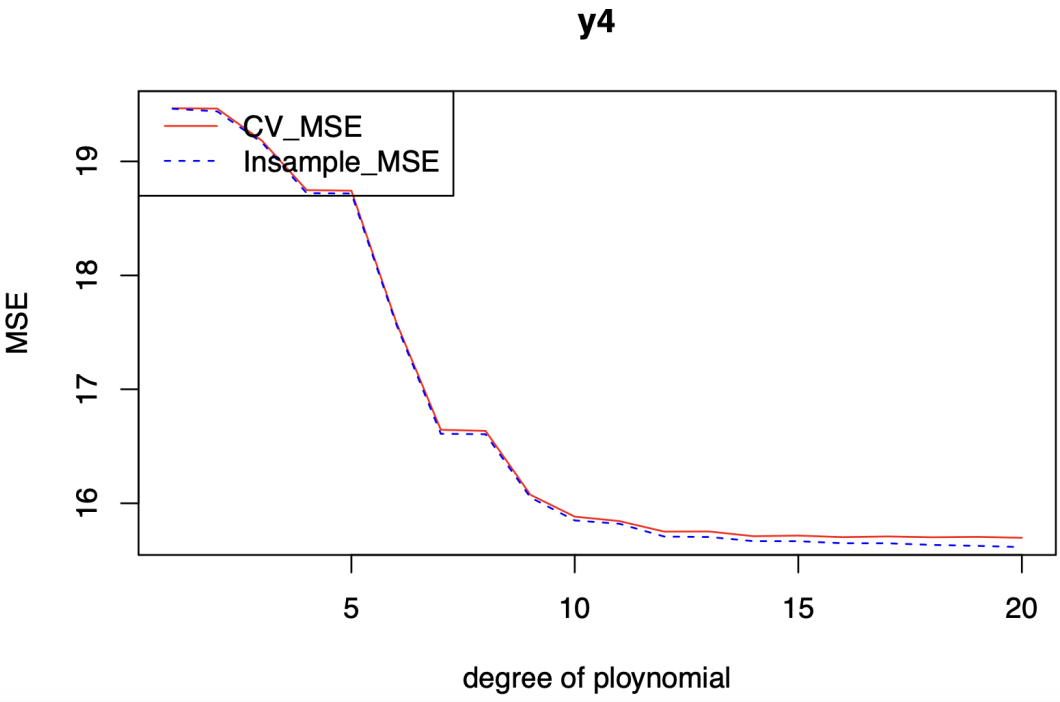


Figure 11: p4

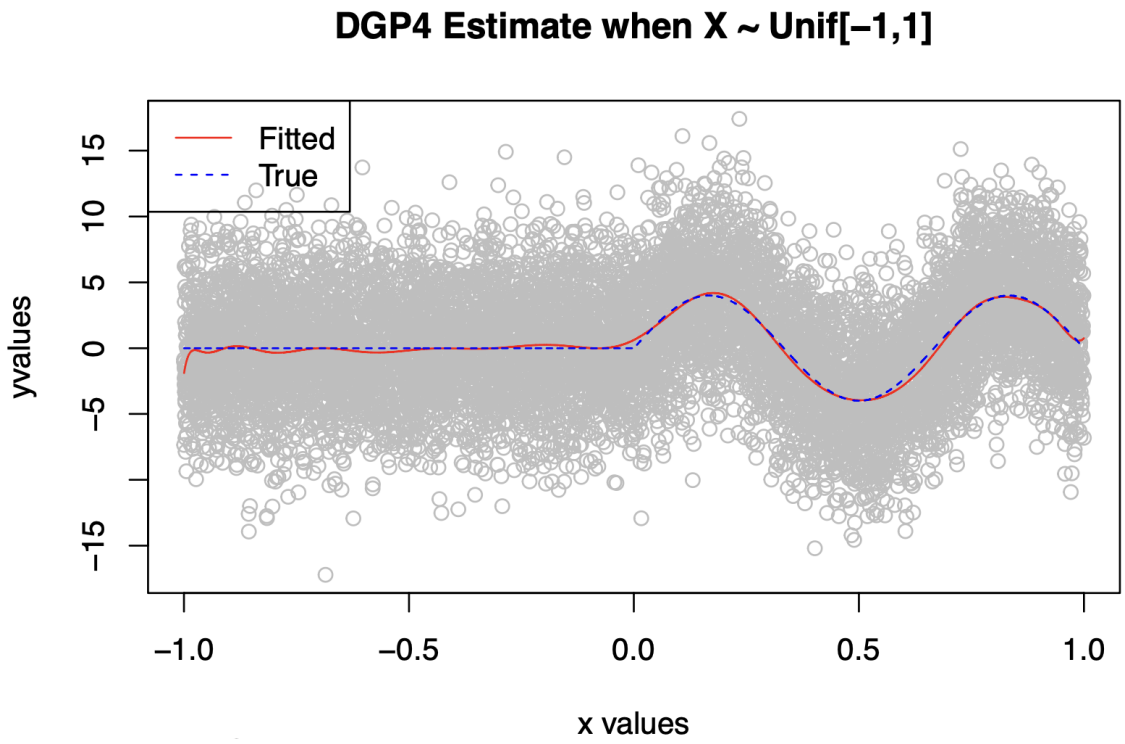


Figure 12: p4e

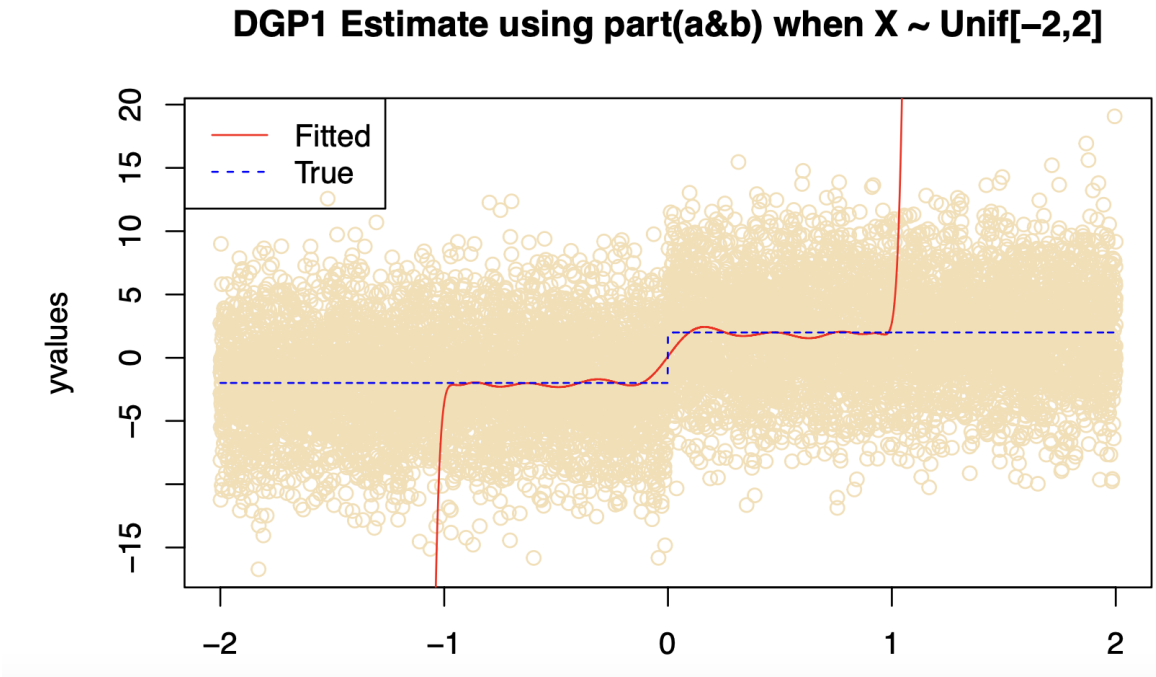


Figure 13: 3p1

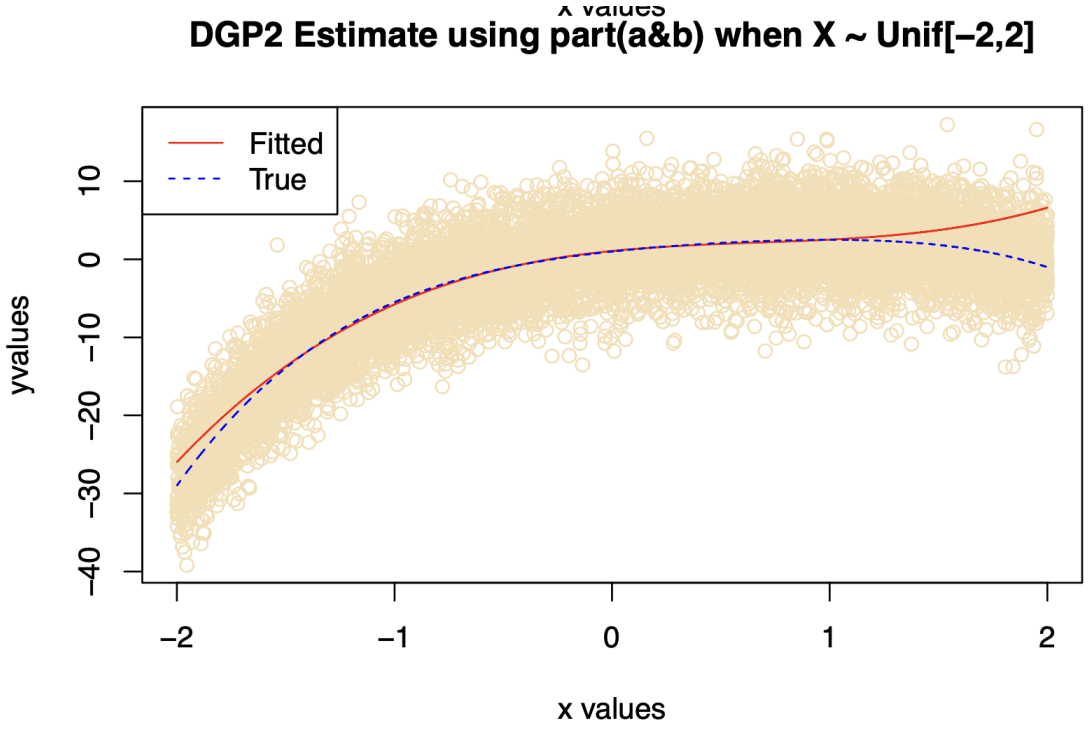


Figure 14: 3p2

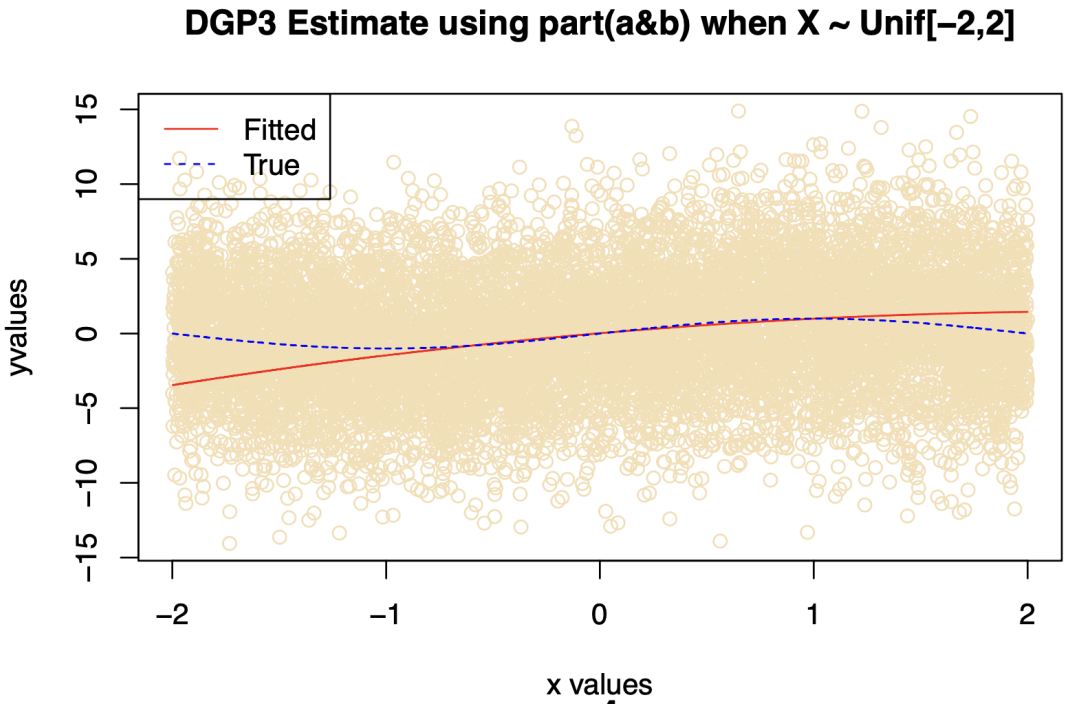


Figure 15: 3p3

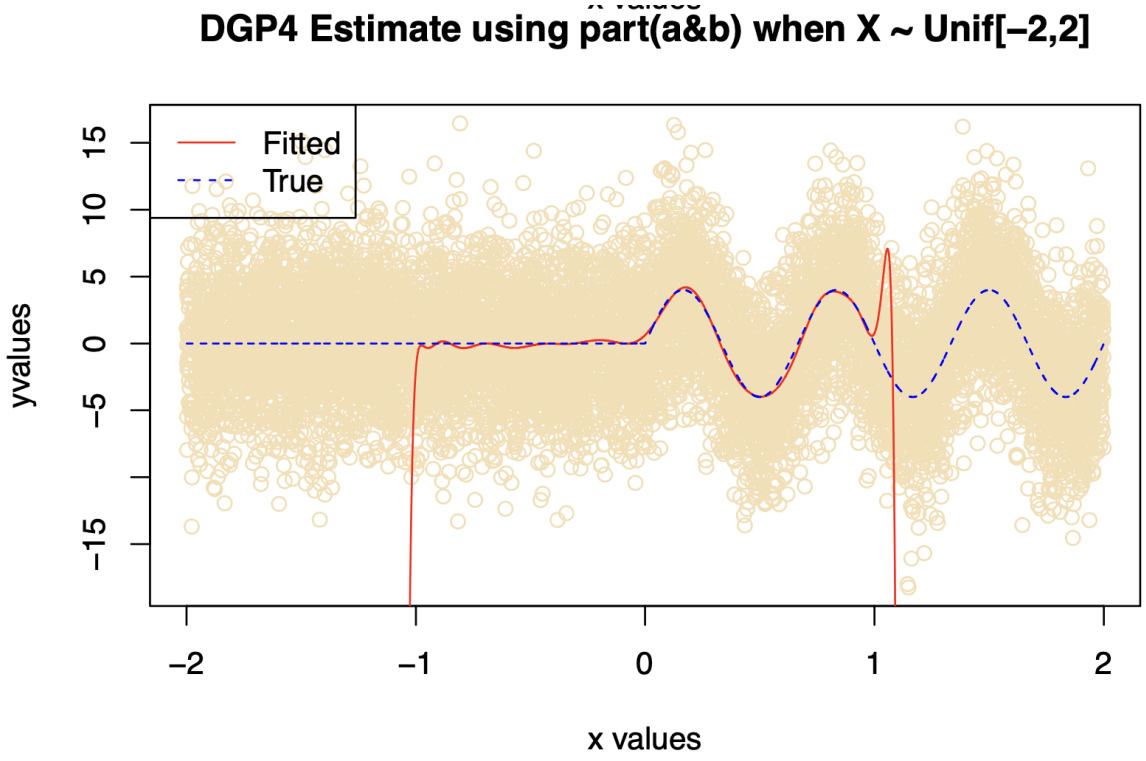


Figure 16: 3p4

```
# Function to simulate data
dgp = function(n = 100, low = -1, high = 1) {
    x = runif(n, low, high)
    epsilon = rnorm(n, mean = 0, sd = 1)
    dgp1 = -2 * (x < 0) + 2 * (x > 0) + 4 * epsilon
    dgp2 = 1 + 3 * x - 2 * x^2 + x^3 - 0.5 * x^4 + 4 * epsilon
    dgp3 = sin((pi/2) * x) + 4 * epsilon
    dgp4 = 4 * sin(3 * pi * x) * (x > 0) + 4 * epsilon
    out = data.frame(x = x, y1 = dgp1, y2 = dgp2, y3 = dgp3,
        y4 = dgp4)
    return(out)
}
```

```
mse = function(data, model = "y1", d = 1, k = 5) {
    y = data[, model]
    x = data[, "x"]
    datanew = data.frame(x = x, y = y)
    # K-cross validation MSE create folds randomly
    n = nrow(datanew)
    folds = sample(rep(1:k, length = n))
    mse = rep(NA, k)
    for (j in 1:k) {
        # train model on all folds except j
        train = folds != j
        ols = lm(y ~ poly(x, degree = d, raw = T), data = datanew[train,
            ])
```

```
        # compute MSE on fold j (not used for training)
        yhat = predict(ols, newdata = datanew[!train, ])
        mse[j] = mean((datanew[!train, "y"] - yhat)^2)

    }
    # compute average mse
    mse.cv = mean(mse)

    # In sample MSE
    ols2 = lm(y ~ poly(x, degree = d, raw = T), data = datanew)
    ypred = predict(ols2, newdata = datanew)
    mse.insample = mean((datanew[, "y"] - ypred)^2)

    # mse.list = c(mse.cv,mse.insample)

    return(c(mse.cv, mse.insample))
}
```

```
# Function to plot MSE
plotfunc = function(data) {
    mse.cv = rep(NA, 20)
    mse.sample = rep(NA, 20)
    cols = c("y1", "y2", "y3", "y4")
    d = seq(1, 20)
    for (model in cols) {
        for (i in d) {
            mse.data = mse(data, model = model, d = i)
            mse.cv[i] = mse.data[1]
            mse.sample[i] = mse.data[2]
        }

        # plot CV MSE and MSE sample vs d
        plot(d, mse.cv, type = "l", col = "red", xlab = "degree of ploynomial",
            ylab = "MSE", main = model, lty = 1)
        lines(d, mse.sample, type = "l", col = "blue", lty = 2)
        legend(x = "topleft", legend = c("CV_MSE", "Insample_MSE"),
            col = c("red", "blue"), lty = c(1, 2), ncol = 1)

        # Best ploynomial fir from CV MSE for each model
        best.polycv = which.min(mse.cv)
        cat("The best polynomial fit for ", model, "is", which.min(mse.cv))

        # Fitting the best ploynomial to the full data
        y = data[, model]
        x = data[, "x"]
        datanew = data.frame(x = x, y = y)
        ols_fit = lm(y ~ poly(x, degree = best.polycv, raw = T),
            data = datanew)
        # plot(x,y) Estimated function
        sorted.data = datanew[order(datanew$x), ]
        x = sorted.data[, "x"]
        y = sorted.data[, "y"]
        yhat = predict(ols_fit, newdata = datanew[order(datanew$x),
            ])
```

```
# For part(c)
n = nrow(sorted.data)
x2 = runif(n = n, -2, 2)
epsilon = rnorm(n, mean = 0, sd = 1)

if (model == "y1") {
    plot(x, y, col = "grey", xlab = "x values", ylab = "yvalues",
        main = "DGP1 Estimate when X ~ Unif[-1,1]")
    lines(x, yhat, type = "l", col = "red", lty = 1)
    lines(x, -2 * (x < 0) + 2 * (x > 0), type = "l",
        col = "blue", lty = 2)
    legend(x = "topleft", legend = c("Fitted", "True"),
        col = c("red", "blue"), lty = c(1, 2), ncol = 1)

    # cat('Fitting data for DGP1 when data is from
    # X ~ Unif[-2,2]')
    y2 = -2 * (x2 < 0) + 2 * (x2 > 0) + 4 * epsilon
    data2 = data.frame(x = x2, y = y2)
    sorted.data2 = data2[order(data2$x), ]
    x2.sorted = sorted.data2[, "x"]
    p2.pred = predict(ols_fit, newdata = data2[order(data2$x),
        ])
    plot(x2, y2, col = "wheat", xlab = "x values", ylab = "yvalues",
        main = "DGP1 Estimate using part(a&b) when X ~ Unif[-2,2]")
    lines(x2.sorted, p2.pred, type = "l", col = "red",
        lty = 1)
    lines(x2.sorted, -2 * (x2.sorted < 0) + 2 * (x2.sorted >
        0), type = "l", col = "blue", lty = 2)
    legend(x = "topleft", legend = c("Fitted", "True"),
        col = c("red", "blue"), lty = c(1, 2), ncol = 1)

}
if (model == "y2") {
    plot(x, y, col = "grey", xlab = "x values", ylab = "yvalues",
        main = "DGP2 Estimate when X ~ Unif[-1,1]")
    lines(x, yhat, type = "l", col = "red", lty = 1)
    lines(x, 1 + 3 * x - 2 * x^2 + x^3 - 0.5 * x^4, type = "l",
        col = "blue", lty = 2)
    legend(x = "topleft", legend = c("Fitted", "True"),
        col = c("red", "blue"), lty = c(1, 2), ncol = 1)

    # cat('Fitting data for DGP2 when data is from
    # X ~ Unif[-2,2]')
    y2 = 1 + 3 * x2 - 2 * x2^2 + x2^3 - 0.5 * x2^4 +
        4 * epsilon
    data2 = data.frame(x = x2, y = y2)
    sorted.data2 = data2[order(data2$x), ]
    x2.sorted = sorted.data2[, "x"]
    p2.pred = predict(ols_fit, newdata = data2[order(data2$x),
        ])
    plot(x2, y2, col = "wheat", xlab = "x values", ylab = "yvalues",
        main = "DGP2 Estimate using part(a&b) when X ~ Unif[-2,2]")
    lines(x2.sorted, p2.pred, type = "l", col = "red",
```

```
        lty = 1)
    lines(x2.sorted, 1 + 3 * x2.sorted - 2 * x2.sorted^2 +
        x2.sorted^3 - 0.5 * x2.sorted^4, type = "l",
        col = "blue", lty = 2)
    legend(x = "topleft", legend = c("Fitted", "True"),
        col = c("red", "blue"), lty = c(1, 2), ncol = 1)

}
if (model == "y3") {
    plot(x, y, col = "grey", xlab = "x values", ylab = "yvalues",
        main = "DGP3 Estimate when X ~ Unif[-1,1]")
    lines(x, yhat, type = "l", col = "red", lty = 1)
    lines(x, sin(pi/2 * x), type = "l", col = "blue",
        lty = 2)
    legend(x = "topleft", legend = c("Fitted", "True"),
        col = c("red", "blue"), lty = c(1, 2), ncol = 1)

    # cat('Fitting data for DGP3 when data is from
    # X ~ Unif[-2,2]')
    y2 = sin((pi/2) * x2) + 4 * epsilon
    data2 = data.frame(x = x2, y = y2)
    sorted.data2 = data2[order(data2$x), ]
    x2.sorted = sorted.data2[, "x"]
    p2.pred = predict(ols_fit, newdata = data2[order(data2$x),
        ])
    plot(x2, y2, col = "wheat", xlab = "x values", ylab = "yvalues",
        main = "DGP3 Estimate using part(a&b) when X ~ Unif[-2,2]")
    lines(x2.sorted, p2.pred, type = "l", col = "red",
        lty = 1)
    lines(x2.sorted, sin((pi/2) * x2.sorted), type = "l",
        col = "blue", lty = 2)
    legend(x = "topleft", legend = c("Fitted", "True"),
        col = c("red", "blue"), lty = c(1, 2), ncol = 1)

}
if (model == "y4") {
    plot(x, y, col = "grey", xlab = "x values", ylab = "yvalues",
        main = "DGP4 Estimate when X ~ Unif[-1,1]")
    lines(x, yhat, type = "l", col = "red", lty = 1)
    lines(x, 4 * sin(3 * pi * x) * (x > 0), type = "l",
        col = "blue", lty = 2)
    legend(x = "topleft", legend = c("Fitted", "True"),
        col = c("red", "blue"), lty = c(1, 2), ncol = 1)

    # cat('Fitting data for DGP4 when data is from
    # X ~ Unif[-2,2]')
    y2 = 4 * sin(3 * pi * x2) * (x2 > 0) + 4 * epsilon
    data2 = data.frame(x = x2, y = y2)
    sorted.data2 = data2[order(data2$x), ]
    x2.sorted = sorted.data2[, "x"]
    p2.pred = predict(ols_fit, newdata = data2[order(data2$x),
        ])
    plot(x2, y2, col = "wheat", xlab = "x values", ylab = "yvalues",
```

```
            main = "DGP4 Estimate using part(a&b) when X ~ Unif[-2,2]")
        lines(x2.sorted, p2.pred, type = "l", col = "red",
            lty = 1)
        lines(x2.sorted, 4 * sin(3 * pi * x2.sorted) * (x2.sorted >
            0), type = "l", col = "blue", lty = 2)
        legend(x = "topleft", legend = c("Fitted", "True"),
            col = c("red", "blue"), lty = c(1, 2), ncol = 1)

    }

  }
}
```