Project 2

Benjamin Draves

Exercise 1

(a)

Horner's scheme iteratively adds successive polynomial degree until the polynomial is constructed. For instance, suppose that we have a coefficient vector $c = (c_1, c_2, \ldots, c_n)$. Then at any stage, k, of the process we have polynomial $p_k(x) = \sum_{i=0}^{k} c_{n-i} x^{k-i}$. Notice here that we assume that c_n corresponds with the highest order on the polynomial (namely k) and c_{n-k} corresponds with the constant term in the p_k polynomial. Then from here we wish to build p_{k+1} we can do so by the following

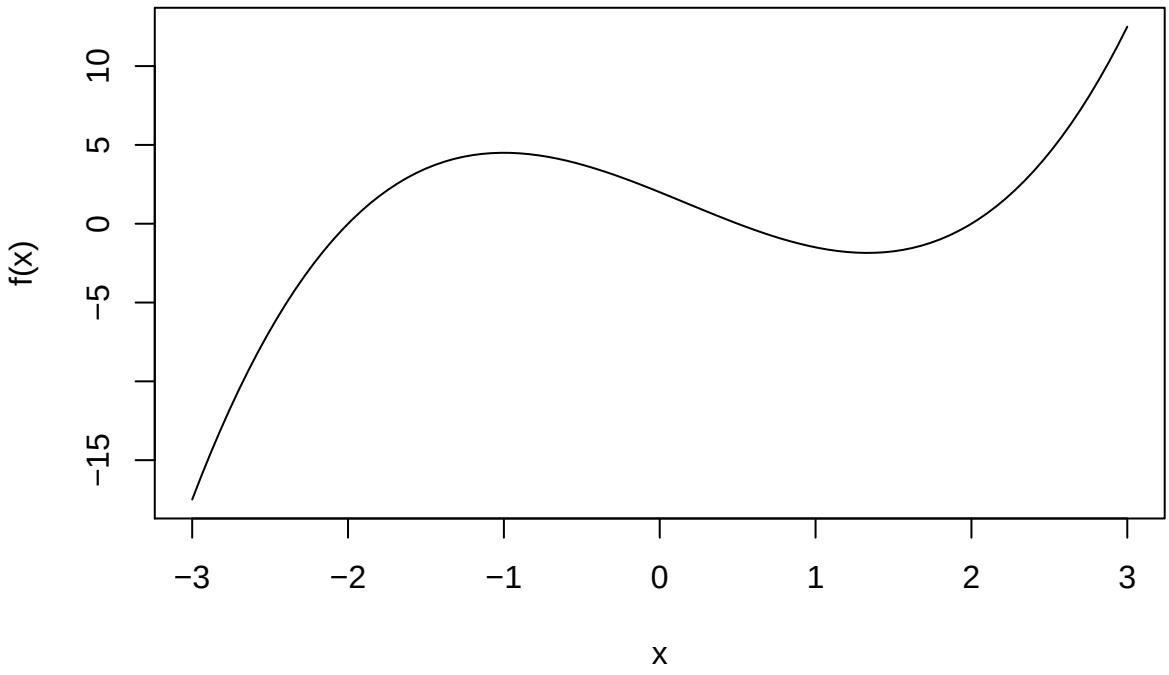
$$p_{k+1}(x) = \sum_{i=0}^{k+1} c_{n-i} x^{(k+1)-i} = c_{n-(k+1)} + x \sum_{i=0}^{k} c_{n-i} x^{k-i} = c_{n-(k+1)} + x p_k(x)$$

Relating this to the code, we see that s acts as polynomial at state each stage in the process. Then we look in descending order over of the coefficient vector $coef$ and update the polynomial by multiplying $x * s$ and then adding on the next coefficient. This is captured in the recursive formula written above.

(b)

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

# plot function
c <- c(2, -4, -1/2, 1)
f <- horner(c)
curve(f, from = -3, 3)
```



(c)

```
poly_der_coef <- function(coef) seq(1, length(coef) - 1) * coef[-1]

newtons <- function(c, root = 0, max.iter = 1000) {

    # set up polynomials
    p <- horner(c)
    p_prime <- horner(poly_der_coef(c))

    # set up stopping criterion
    tol <- 1e-05
    diff <- Inf
    iter.num <- 1

    while (iter.num < max.iter & diff > tol) {
        # two conditions: (i) iteration limit (ii) convergence
        # criterion

        # update root: if derivative is zero, then Netwons update is
        # undefined, return Nan, else use the Newtons update
        new_root <- ifelse(abs(p_prime(root)) < tol, return(NaN),
            root - p(root)/p_prime(root))
        diff <- abs(new_root - root)
        root <- new_root
        iter.num <- iter.num + 1
    }
    ifelse(iter.num == max.iter, NaN, root)
}

c <- c(2, -4, -1/2, 1)
c(newtons(c, root = -1.5), newtons(c, root = 1.5), newtons(c,
```

```
    root = -1))
```

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

Here we see that for the first two starting points that the algorithm successfully converges to two different roots of $f(x)$, namely $x = -2, 2$. When $x_0 = -1$ however, we see that $f'(-1.5) = 0$. Therefore, the Newton's included a not a number value. As we will never be able to update this number, we simply return not a number.

(d)

Here by simply rearranging terms in the recursive definition of the Legendre polynomials we have

$$L_{e_n}(x) = \frac{2(n-1)+1}{n} L_{e_{n-1}}(x) - \frac{n-1}{n} L_{e_{n-2}}(x)$$

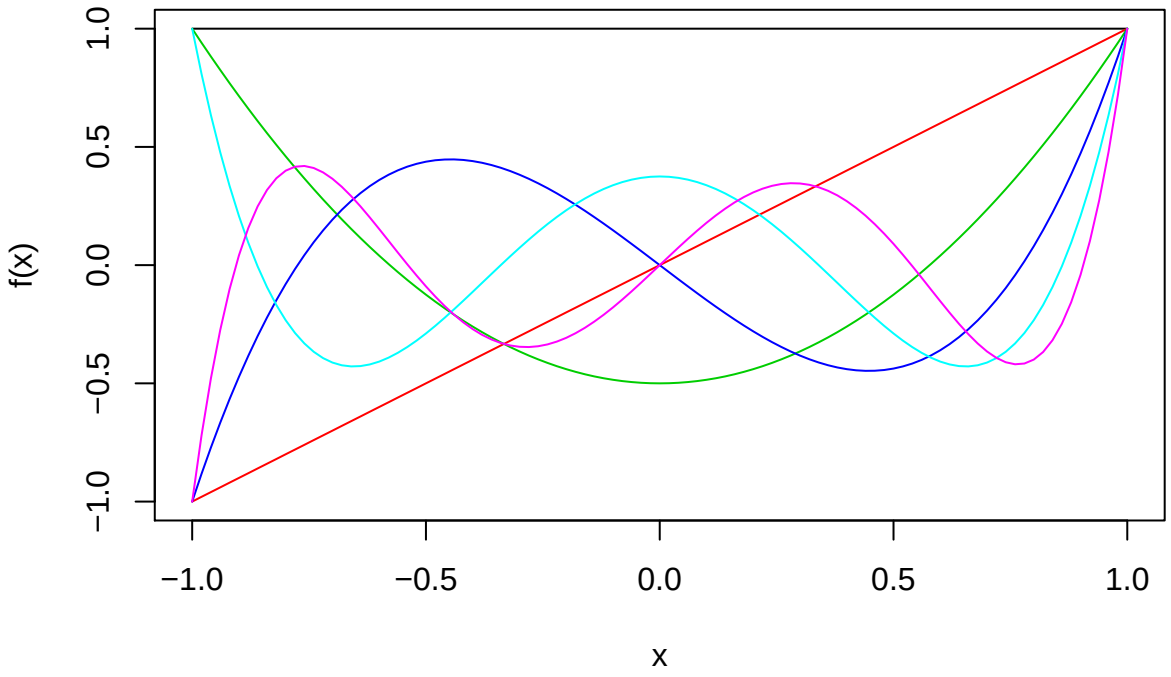
Here it is easy to see that $L_{e_0}(x) = 1$ and $L_{e_1}(x) = x$. Using these, we can define a recursive function that constructs the nth order Legendre polynomial as follows.

```
legendre_coef <- function(n) {

    if (n == 0)
        return(c(1))  # return coefficents for L_{e_0}
    if (n == 1)
        return(c(0, 1))  # return coefficents for L_{e_1}
 else {
        ln1 <- (2 * (n - 1) + 1)/n * c(0, legendre_coef(n -
            1))  # recursively get coefficents for L_{n-1}
        tmp <- (n - 1)/n * legendre_coef(n - 2)  # recursively get coefficents for L_{n-2}
        ln2 <- c(tmp, rep(0, length(ln1) - length(tmp)))  # match coefficents lengths
        return(ln1 - ln2)
    }
}
```

In addition we add a plot of the first 6 six Legrende polynomials.

```
# A plot of the first 6 polynomials
for (i in 0:5) {
    f <- horner(legendre_coef(i))
    if (i == 0)
        curve(f, from = -1, to = 1, col = i + 1, ylim = c(-1,
            1)) else curve(f, from = -1, to = 1, col = i + 1, add = TRUE)
}
```



Exercise 2

(a)

```
#import data
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)
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)

#create design matrix
d <- 4
X <- outer(x, 0:d, '^')

#build T
T <- rbind(cbind(crossprod(X), crossprod(X, y)), cbind(crossprod(y,X), crossprod(y)))
```

(b)

```
SWEEP <- function(A, k) {
    D <- A[k, k]
    ifelse(D == 0, A[k, ] <- A[k, ], A[k, ] <- A[k, ]/D)
    for (i in (1:nrow(A))[-k]) {
        B <- A[i, k]
        A[i, ] <- A[i, ] - B * A[k, ]
        A[i, k] <- -B/D
    }
    A[k, k] <- 1/D

    return(A)
}

SWEEP(SWEEP(T, 1), 1)
```

```
##         [,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
```

```
SWEEP(SWEEP(T, 2), 2)
```

```
##         [,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
```

```
SWEEP(SWEEP(T, 3), 3)
```

```
##         [,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
```

```
SWEEP(SWEEP(T, 4), 4)
```

```
##         [,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
```

```
SWEEP(SWEEP(T, 5), 5)
```

```
##         [,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
```

Notice that in each of the examples above we see that the kth column is just the same as the original column. As sweep replaces each column sweep with the inverse of its previous value, but iterating, we simple get the inverse of the inverse. That is, we simply replace the original value with itself. Indeed this is the case as the next piece of code shows.

```
tol <- 10e-10
sum(abs(SWEEP(SWEEP(T, 1),1) - T) > tol)
```

```
## [1] 0
```

```
sum(SWEEP(SWEEP(T, 2),2)- T > tol)
```

```
## [1] 0
```

```
sum(SWEEP(SWEEP(T, 3),3)- T > tol)
```

```
## [1] 0
```

```
sum(SWEEP(SWEEP(T, 4),4)- T > tol)
```

```
## [1] 0
```

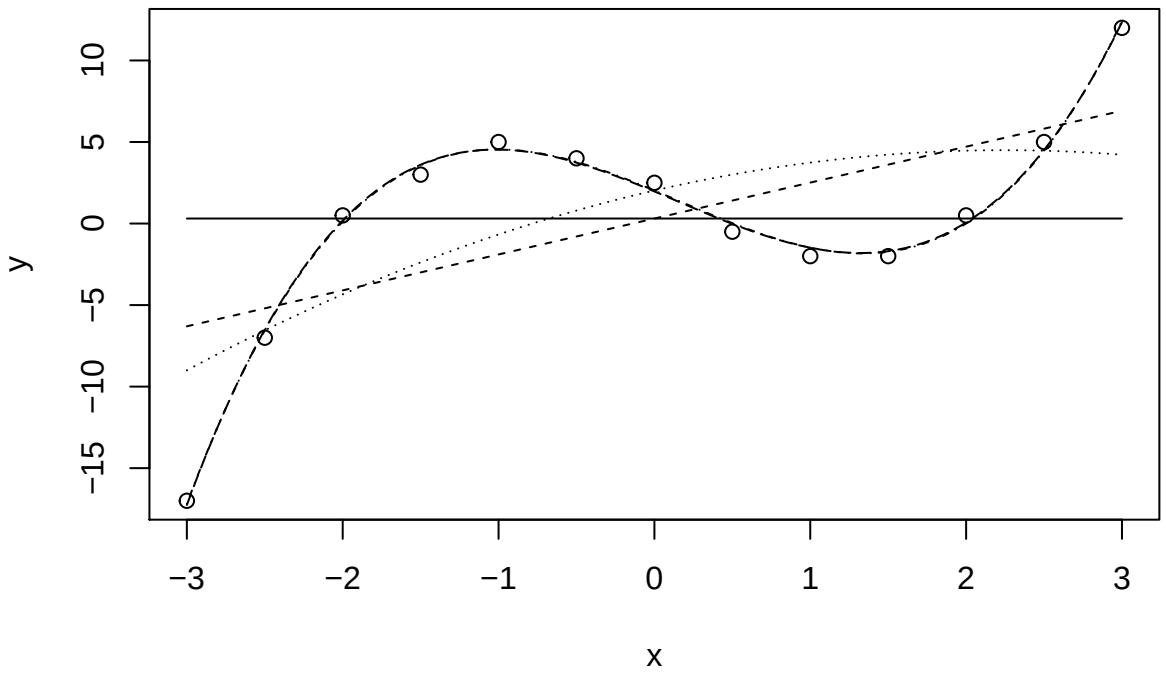
```
sum(SWEEP(SWEEP(T, 5),5)- T > tol)
```

```
## [1] 0
```

Therefore we see that the SWEEP of the SWEEP is simply the identity.

(c)

```
plot(x, y)
a <- seq(-3, 3, length.out = 100)
p <- ncol(T) - 1
for (k in 1:p) {
    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
## [1] 5.000000 2.486895
```

Notice for $k = 1$ we sweep the $k = 1$ row and plot first column of $X^T y$ block. This corresponds to the regression fit with the data provided in the first column of $X^T X$. As that column are all ones, this corresponds to the regression model $\mathbb{E}(\mathbf{Y}|\mathbf{X}) = \beta_0$ which just fits the grand mean. For $k = 2$ we sweep the $k = 2$ row and plot the first and second column of $X^T y$ block. Note if X was just these first and second columns, then this would correspond to the regression fit with the data provided in those two columns. As that columns are $[1, X]$, this corresponds to the regression model $\mathbb{E}(Y|X) = \beta_0 + \beta_1 X$ which just fits the linear fit. This iterates for the different features until we plot the fit over all columns of X which is the regression model $\mathbb{E}(Y|X) = \beta_0 + \beta_1 X + \beta_2 X^2 + \beta_3 X^3 + \beta_4 X^4$

For any k the lower right block after that row has been swept can be written as follows

$$y^T y - y^T X_{1:k} (X_{1:k}^T X_{1:k})^{-1} y$$

This corresponds to the model RSS for the model $\mathbb{E}(Y|X) = \sum_{j=0}^{k} \beta_j X^k$. Therefore we see we are iteratively printing out the model RSS statistics.

Exercise 3

(a)

Preforming the sweep operator on the xx-block.

$$\Sigma = \begin{bmatrix} \Sigma_{xx} & \Sigma_{xy} \\ \Sigma_{xy}^T & \Sigma_{yy} \end{bmatrix} \longrightarrow \begin{bmatrix} \Sigma_{xx}^{-1}\Sigma_{xx} & \Sigma_{xx}^{-1}\Sigma_{xy} \\ \Sigma_{xy}^T - \Sigma_{xy}^T\Sigma_{xx}^{-1}\Sigma_{xx} & \Sigma_{yy} - \Sigma_{xy}^T\Sigma_{xx}^{-1}\Sigma_{xy} \end{bmatrix}$$
$$\longrightarrow \begin{bmatrix} I & \Sigma_{xx}^{-1}\Sigma_{xy} \\ 0 & \Sigma_{yy} - \Sigma_{xy}^T\Sigma_{xx}^{-1}\Sigma_{xy} \end{bmatrix} \longrightarrow \begin{bmatrix} \Sigma_{xx}^{-1} & \Sigma_{xx}^{-1}\Sigma_{xy} \\ -\Sigma_{xy}^T\Sigma_{xx}^{-1} & \Sigma_{yy} - \Sigma_{xy}^T\Sigma_{xx}^{-1}\Sigma_{xy} \end{bmatrix}$$

Hence, if we define $S = \Sigma_{yy} - \Sigma_{xy}^T\Sigma_{xx}^{-1}\Sigma_{xy}$ we can sweep the yy-block as follows

$$\begin{bmatrix} \Sigma_{xx}^{-1} & \Sigma_{xx}^{-1}\Sigma_{xy} \\ -\Sigma_{xy}^T\Sigma_{xx}^{-1} & S \end{bmatrix} \longrightarrow \begin{bmatrix} \Sigma_{xx}^{-1} - \Sigma_{xx}^{-1}\Sigma_{xy}S^{-1}\Sigma_{xy}^T\Sigma_{xx}^{-1} & 0 \\ -S^{-1}\Sigma_{xy}^T\Sigma_{xx}^{-1} & I \end{bmatrix} \longrightarrow \begin{bmatrix} \Sigma_{xx}^{-1} - \Sigma_{xx}^{-1}\Sigma_{xy}S^{-1}\Sigma_{xy}^T\Sigma_{xx}^{-1} & \Sigma_{xx}^{-1}\Sigma_{xy}S^{-1} \\ -S^{-1}\Sigma_{xy}^T\Sigma_{xx}^{-1} & S^{-1} \end{bmatrix}$$

From this we see that the yy-block of Σ^{-1} is the inverse of the Schurr complement of the yy-block of Σ. Hence we see that these are almost commutatble in the sense that there is a relation between the Schurr complement and its inverse from a block matrix.

(b)

Using the calculations made in (a) we see that $(\Sigma^{-1})_{xx} = \Sigma_{xx}^{-1} - \Sigma_{xx}^{-1}\Sigma_{xy}S^{-1}\Sigma_{xy}^T\Sigma_{xx}^{-1}$. Now notice that if we would have swept the yy-block first followed by the xx-block, the first sweep would result in producing $SC_{xx}(\Sigma) = \Sigma_{xx} - \Sigma_{xy}^T\Sigma_{yy}^{-1}\Sigma_{xy}$. From here, when sweeping the xx-block would have produced $(SC_{xx}(\Sigma))^{-1}$. Hence we see that by symmetry that if we were to sweep yy then xx that $(\Sigma^{-1})_{xx} = (SC_{xx}(\Sigma))^{-1} = (\Sigma_{xx} - \Sigma_{xy}\Sigma_{yy}^{-1}\Sigma_{xy}^T)^{-1}$.

(c)

Here we see that the conditional distribution has the following form

$$\mathbf{y}|\mathbf{x} \sim N(\mu_y + \Sigma_{xy}^T\Sigma_{xx}^{-1}(\mathbf{x} - \mu_x), SC_{yy}(\Sigma))$$

Recall that we achieved $SC_{yy}(\Sigma)$ after sweeping the xx-block of the Σ matrix above. Moreover, by the definition of $SC_{yy}(\Sigma) = \Sigma_{yy} - \Sigma_{xy}^T \Sigma_{xx}^{-1} \Sigma_{xy}$ we see that the first two terms in the conditional mean are quite close to the desired quantity. Hence proceeding in a similar way as we did to set up the tableau in the regression case, consider the following form

$$\begin{aligned}
\begin{bmatrix} \Sigma_{xx} & \Sigma_{xy} & \mu_x - \mathbf{x} \\ \Sigma_{xy}^T & \Sigma_{yy} & \mu_y \end{bmatrix} &\longrightarrow \begin{bmatrix} I & \Sigma_{xx}^{-1}\Sigma_{xy} & \Sigma_{xx}^{-1}(\mu_x - \mathbf{x}) \\ 0 & \Sigma_{yy} - \Sigma_{xy}^T \Sigma_{xx}^{-1} \Sigma_{xy} & \mu_y - \Sigma_{xy}^T \Sigma_{xx}^{-1}(\mu_x - \curvearrowleft) \end{bmatrix} \\
&\longrightarrow \begin{bmatrix} \Sigma_{xx}^{-1} & \Sigma_{xx}^{-1}\Sigma_{xy} & \Sigma_{xx}^{-1}(\mu_x - \mathbf{x}) \\ -\Sigma_{xy}^T \Sigma_{xx}^{-1} & \Sigma_{yy} - \Sigma_{xy}^T \Sigma_{xx}^{-1} \Sigma_{xy} & \mu_y + \Sigma_{xy}^T \Sigma_{xx}^{-1}(\mathbf{x} - \mu_x) \end{bmatrix} \\
&= \begin{bmatrix} \Sigma_{xx}^{-1} & \Sigma_{xx}^{-1}\Sigma_{xy} & \Sigma_{xx}^{-1}(\mu_x - \mathbf{x}) \\ -\Sigma_{xy}^T \Sigma_{xx}^{-1} & \mathrm{Var}(\mathbf{y}|\mathbf{x}) & \mathbb{E}(\mathbf{y}|\mathbf{x}) \end{bmatrix}
\end{aligned}$$

Here we see that in the bottom row in the second and third entry that we recover the conditional variance and conditional mean of $\mathbf{y}|\mathbf{x}$.