# MTH 4230 R Notes 6

## 1 Polynomial Regression Models

### 1.1 Introduction

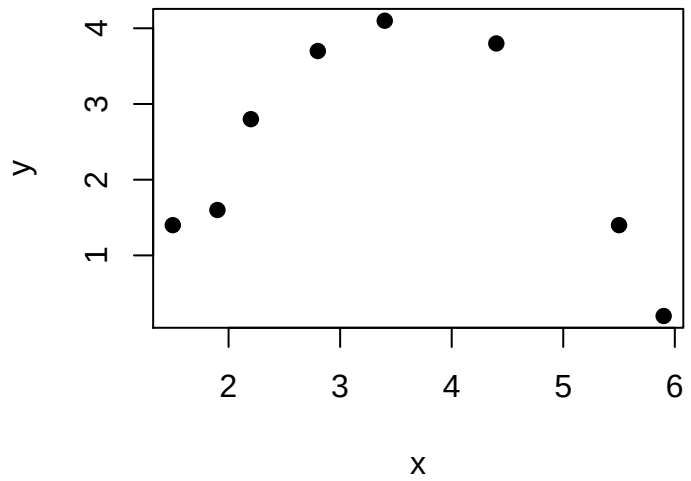
- There are a few ways to perform a ***polynomial regression*** analysis in R.

  The first is to insert variables representing the polynomial terms ($x^2$, $x^3$, etc.) into the *data frame* **before** fitting the model.

  The second is to specify the polynomial terms in the model *formula* **during** the call to `lm()` using the **"as is"** function `I()`.

- Below, we create a *data frame* and plot the data. The function `with()` allows us to refer to variables in the *data frame* (`x` and `y`) *without* having to use the dollar sign operator `$`:

```
my.data <- data.frame(y = c(1.4, 1.6, 2.8, 3.7, 4.1, 3.8, 1.4, 0.2),
                      x = c(1.5, 1.9, 2.2, 2.8, 3.4, 4.4, 5.5, 5.9))
with(my.data, plot(x, y, pch = 19))
```



## 1.2 Polynomial Regression by Creating Variables Representing the Polynomial Terms

- To fit a ***quadratic regression model***,

$$Y = \beta_0 + \beta_1 X + \beta_2 X^2 + \epsilon$$

we create the **quadratic** term and add it to the *data frame*:

```
my.data$x2 <- my.data$x^2
```

Next we double check that it was created correctly and inserted in the *data frame*:

```
head(my.data)
```

```
##     y   x    x2
## 1 1.4 1.5  2.25
## 2 1.6 1.9  3.61
## 3 2.8 2.2  4.84
## 4 3.7 2.8  7.84
## 5 4.1 3.4 11.56
## 6 3.8 4.4 19.36
```

Now it's a simple matter to fit the ***quadratic regression model*** using `lm()`:

```
my.reg <- lm(y ~ x + x2, data = my.data)
```

Here's the summary of the fit:

```
summary(my.reg)
```

```
##
## Call:
## lm(formula = y ~ x + x2, data = my.data)
##
## Residuals:
##         1         2         3         4         5
##  0.368766 -0.507764  0.035696  0.010532  0.002591
##         6         7         8
##  0.172075 -0.052068 -0.029828
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -5.05309    0.71797  -7.038 0.000894
## x            5.13377    0.44408  11.561 8.50e-05
## x2          -0.71837    0.05885 -12.206 6.53e-05
##
## Residual standard error: 0.2927 on 5 degrees of freedom
## Multiple R-squared:  0.9698,Adjusted R-squared:  0.9577
## F-statistic: 80.22 on 2 and 5 DF,  p-value: 0.0001588
```

From the output above, the equation of the ***fitted quadratic regression model*** is

$$\hat{Y} = -5.053 + 5.134X - 0.718X^2 \,.$$

The coefficient $\beta_2$ for the quadratic term ($X^2$) is statistically significant, so we can't drop that term from the model.

## 1.3 Plotting the Fitted Polynomial Curve with the Data

- Once we obtain the fitted polynomial, as above, we can add it to a scatterplot of the data using `curve()`.

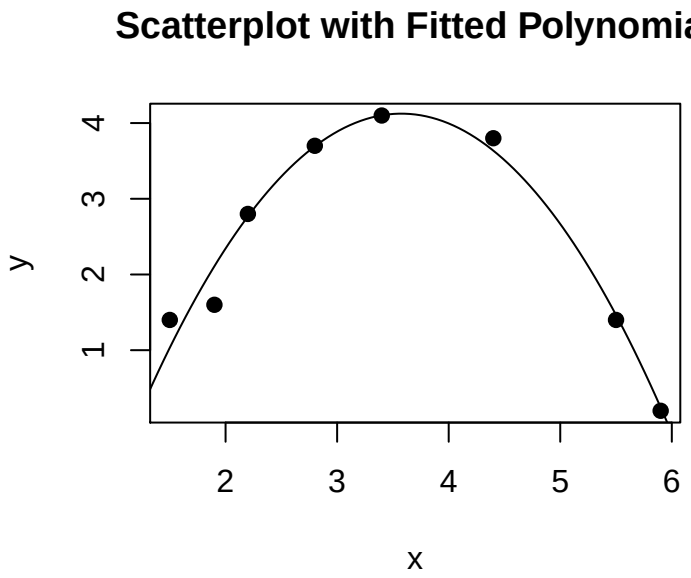
```
curve()    # Draws a curve corresponding to an expression involving
           # the variable name x over the interval [from, to].
```

We need to specify `add = TRUE` in the call to `curve()` so that curve is added to the current plot instead of overwriting it.

Below, after reproducing the scatterplot, we add the fitted polynomial (from above):

```
with(my.data,
     plot(x, y, pch = 19, main = "Scatterplot with Fitted Polynomial"))

curve(-5.053 + 5.134*x - 0.718*x^2, from = 1.3, to = 6.2, add = TRUE)
```



## 1.4 Polynomial Regression Using the "As Is" Function `I()`

- The following **WILL NOT** fit a quadratic polynomial:

```
my.reg <- lm(y ~ x + x^2, data = my.data)        # This DOES NOT work
```

The reason is that the '^' operator has a special meaning when used in a *formula* in R. (It's used to define *interactions*. Type `?formula` for more information).

Instead, we must use the **"as is"** function, `I()`.

```
I()    # "As is" function.  In R formulas (such as those passed to
       # lm()), it interprets operators such as "+", "-", "*" and "^"
       # as arithmetical operators (instead of as formula operators).
```

- Here's an example (using `my.data` from above):

```
my.reg <- lm(y ~ x + I(x^2), data = my.data)
```

The above command will fit the ***quadratic regression model***, as desired.

- The **"as is"** function can be used to include higher order polynomial terms in the model too, for example `I(x^3)`.

## 1.5 Polynomial Regression Using Centered Predictors

- It's sometimes preferable to ***center*** the predictor variable before fitting a polynomial regression model to reduce problems associated with ***multicollinearity***.
- It's a simple matter to ***center*** the predictor and then include it as a new variable in the *data frame*, for example:

```
my.data$cntrd.x <- my.data$x - mean(my.data$x)
```

Now we check that the ***centered*** variable was created and inserted in the *data frame*:

```
head(my.data)

##     y   x    x2 cntrd.x
## 1 1.4 1.5  2.25   -1.95
## 2 1.6 1.9  3.61   -1.55
## 3 2.8 2.2  4.84   -1.25
## 4 3.7 2.8  7.84   -0.65
## 5 4.1 3.4 11.56   -0.05
## 6 3.8 4.4 19.36    0.95
```

Once the ***centered*** version of the predictor has been created and added to the *data frame*, we can fit a quadratic model to the centered $X$'s. Below we do it using the **"as is"** function `I()`:

```
my.reg <- lm(y ~ cntrd.x + I(cntrd.x^2), data = my.data)
```

# 2 Regression Models with Interactions

- There are two ways to include an ***interaction*** term in a regression model in R. The first (and easiest) is to include the ***interaction*** in the model *formula* passed to `lm()`. The second is to create new variables representing the interaction terms by taking the **products** ($X_1X_2$, etc.) of predictors, and then insert them into the *data frame* for use in fitting the model. We'll only look at the first of these two ways.

## 2.1 Regression Models with Two-Predictor Interactions

- To specify an ***interaction*** term in a model *formula* in R, we use the colon symbol `:` or, if we want to include **every** possible interaction term, the asterisk symbol `*`.
- For example suppose we have data with two predictors $X_1$ and $X_2$:

```
y = c(1.3, 1.6, 1.3, 1.5, 7.1, 8.8, 9.5, 13.2)
x1 = c(1.5, 1.9, 1.7, 2.3, 1.4, 2.0, 2.6, 3.0)
x2 = c(4.4, 4.9, 4.7, 5.3, 6.7, 7.9, 8.1, 8.4)
```

and we create a *data frame* containing the data:

```
my.data <- data.frame(y = y, x1 = x1, x2 = x2)
```

- To fit a model that includes the ***interaction*** between $x_1$ and $x_2$, we type:

```
my.reg <- lm(y ~ x1+x2+x1:x2, data = my.data)
```

The term `x1:x2` in the model *formula* above corresponds to the **interaction** term.

A summary regression analysis is below.

```
summary(my.reg)

##
## Call:
## lm(formula = y ~ x1 + x2 + x1:x2, data = my.data)
##
## Residuals:
##       1       2       3       4       5       6
## -0.3280  0.2468 -0.2343  0.1671  0.7775 -0.4091
```

```
##       7       8
## -1.0667  0.8468
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   8.0428     7.1103   1.131   0.3212
## x1          -10.1823     3.6992  -2.753   0.0512
## x2           -0.1231     1.0304  -0.119   0.9107
## x1:x2         1.4242     0.5008   2.844   0.0467
##
## Residual standard error: 0.8482 on 4 degrees of freedom
## Multiple R-squared:  0.9815,Adjusted R-squared:  0.9676
## F-statistic: 70.58 on 3 and 4 DF,  p-value: 0.0006406
```

We can see from the `x1:x2` line of the output that there's a statistically significant **interaction** between $X_1$ and $X_2$, so the interaction should be retained in the model.

### 2.2 Regression Models with Higher Order Interactions

- When there are more than two predictors, higher order **interactions** are again specified using the colon operator, e.g. the three-way interaction `x1:x2:x3`.

  We can fit the model containing **all possible interactions** using the asterisk symbol `*` in the *formula* passed to `lm()`.

- Suppose, for example, that we have a *data frame* `my.data2` containing a **response variable** $Y$ and **three predictors** $X_1$, $X_2$, and $X_3$.

  Typing:

```
my.reg <- lm(y ~ x1*x2*x3, data = my.data2)
```

  is equivalent to typing:

```
my.reg <- lm(y ~ x1 + x2 + x3 + x1:x2 + x1:x3 + x2:x3 + x1:x2:x3, data = my.data2)
```

  The asterisk symbol `*` in the *formula* indicates that we want to include the **predictors and all possible interactions** between them.

- For more information about specifying models, type `?formula`.

## 3 Plotting a Fitted Regression Surface

- When there are **two predictors**, we can plot a fitted regression surface using a few different methods. We'll look at the following graphics functions:

```
persp()             # Draws perspective plots of a surface over the
                    # x1,x2 plane
contour()           # Creates a contour plot or adds contours to an ex-
                    # isting plot
filled.contour()    # Creates a filled contour plot
image()             # Creates a grid of colored rectangles (heat map)
                    # with colors corresponding to the values in a
                    # matrix z
```

- Another useful function for generating grid point is the following.

```
expand.grid()       # Returns a data frame containing all combinations
                    # of values in the supplied vectors
```

## 3.1 Plotting a Fitted Regression Surface Using `persp()`

- The **"perspective"** function `persp()` will create a three dimensional graph of a surface. It takes *vector* arguments `x` and `y`, containing $x$-axis and $y$-axis (horizontal-plane) grid line coordinates ($X_1$ and $X_2$ values in our case) above whose intersections the surface heights are passed as a *matrix* argument `z`.
- For example, to use `persp()` to plot a fitted regression surface, first we create the $x$ and $y$ axis coordinates of grid lines on the horizontal plane:

```
x1.grid <- seq(from = 1.3, to = 6.2, by = 0.1)
x2.grid <- seq(from = 4.1, to = 16.1, by = 0.5)
```

Next, we'll need to create a *matrix* containing the heights of the regression surface above the intersections of the grid lines.

First, though, we obtain the intersections of the grid lines using `expand.grid()`:

```
x1.x2.grid <- expand.grid(x1 = x1.grid, x2 = x2.grid)
```

```
head(x1.x2.grid)
```

```
##    x1  x2
## 1 1.3 4.1
## 2 1.4 4.1
## 3 1.5 4.1
## 4 1.6 4.1
## 5 1.7 4.1
## 6 1.8 4.1
```

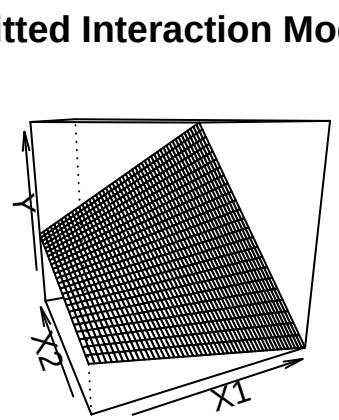
In `expand.grid()`, the names, `x1` and `x2` (which we pick), have to be the same as the names in the *data frame* used to fit the regression model because we're going to use `predict()` to obtain the regression surface.

Now we're ready to obtain the heights of the regression surface over the grid points returned by `expand.grid()`. We use `predict()`, which returns a *vector* that we then convert to a *matrix* for use with `persp()`:

```
my.fitted.surface.vec <- predict(my.reg, newdata = x1.x2.grid)
my.fitted.surface.mat <- matrix(my.fitted.surface.vec,
                                nrow = length(x1.grid),
                                ncol = length(x2.grid))
```

Now we're ready to create the surface plot using `persp()`. The optional arguments `theta` and `phi` are used to rotate the image:

```
persp(x = x1.grid, y = x2.grid,
      z = my.fitted.surface.mat,
      theta = -25,
      xlab = "X1", ylab = "X2", zlab = "Y",
      main = "Fitted Interaction Model")
```



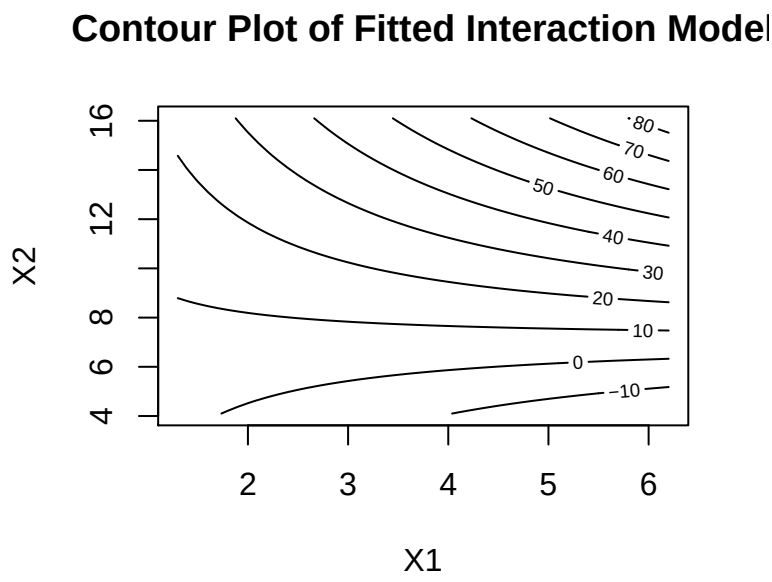

## 3.2 Plotting a Fitted Regression Surface Using `contour()`

- The functions `contour()` and `filled.contour()` will create contour plots of a surface.

Like `persp()`, they take *vector* arguments `x` and `y` containing $x$-axis and $y$-axis (horizontal-plane) grid line coordinates ($X_1$ and $X_2$ values in our case) and a *matrix* argument `z` giving the surface heights above the grid line intersections.

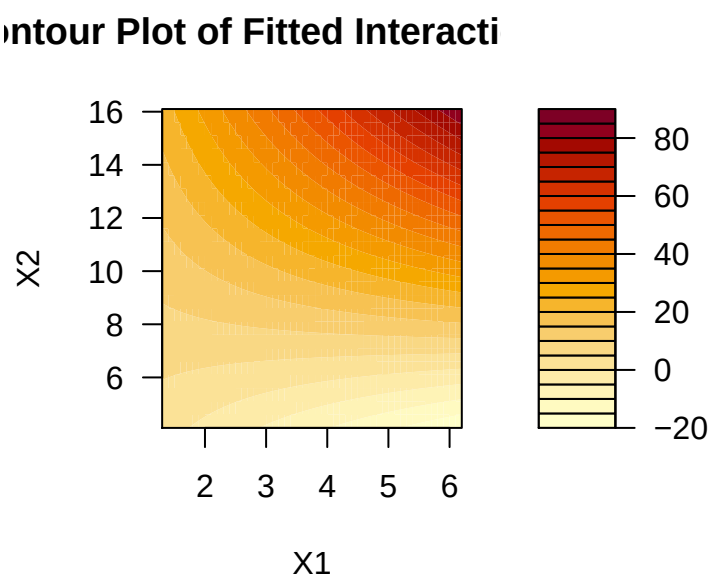
- For example, using `x1.grid`, `x2.grid`, and `my.fitted.surface.mat` (from above), the following will make the contour plot:

```
contour(x = x1.grid, y = x2.grid,
        z = my.fitted.surface.mat,
        xlab = "X1", ylab = "X2",
        main = "Contour Plot of Fitted Interaction Model")
```




If we prefer to have the areas between contours filled with solid colors, we use `filled.contour()`:

```
filled.contour(x = x1.grid, y = x2.grid,
               z = my.fitted.surface.mat,
               xlab = "X1", ylab = "X2",
               main = "Contour Plot of Fitted Interaction Model")
```




### 3.3 Plotting a Fitted Regression Surface Using `image()`

- The function `image()` will create a grid of colored rectangles (heat map) whose colors correspond to values of a surface above the horizontal plane.

  As with `persp()`, `contour()`, and `filled.contour()`, it takes *vector* arguments `x` and `y` containing $x$-axis and $y$-axis (horizontal-plane) grid line coordinates ($X_1$ and $X_2$ values in our case) and a *matrix* argument `z` giving the surface heights above the grid line intersections.

- For example, using `x1.grid`, `x2.grid`, and `my.fitted.surface.mat` (from above), the following makes the desired plot:

```
image(x = x1.grid, y = x2.grid,
      z = my.fitted.surface.mat,
      xlab = "X1", ylab = "X2",
      main = "Image Plot of Fitted Interaction Model")
```

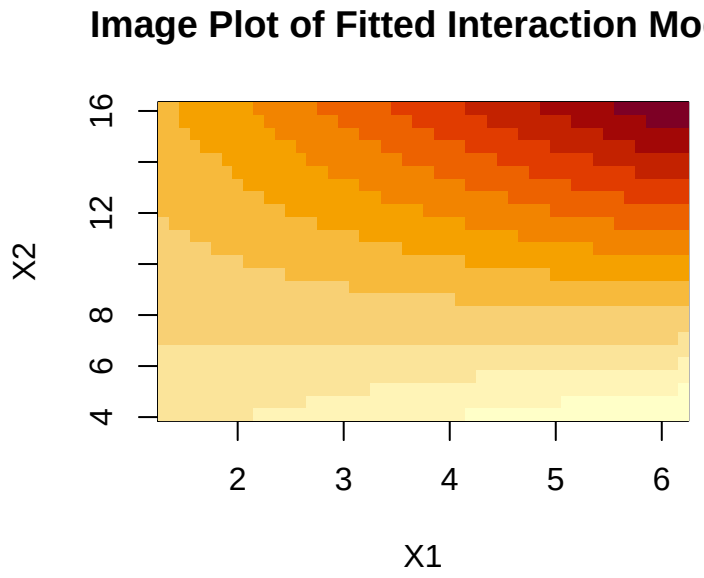


Image Plot of Fitted Interaction Model
X2
16
12
8
6
4
2
3
4
5
6
X1