Best Heteronomial Model Selection

by
Namir Clement Shammas

Introduction

Polynomials are popular constructs in math, calculus and regression analysis. Other popular constructs in regression analysis belong to multiple linear(ized) models. A simple example of a multiple regression model with two independent variables is:

$$Y = a + b_1*X_1 + b_2*X_2 \qquad (1)$$

I will call the above multivariable model a linear ***heteronomial***. It is linear because all the variables are in linear form. A simple example of a nonlinear heteronomial is:

$$Y = a + b_1/X_1 + b_2*X_2^2 \qquad (2)$$

The above model has a linear value for the dependent variable, raises variable X_1 to the power of -1, and raises variable X_2 to power 2. A multivariable equation becomes a nonlinear heteronomial when at least one variable is nonlinear.

The general form of heteronomials is:

$$Y^{yp} = a + b_1*X_1^{px1} + b_2*X_2^{px2} \quad + \ldots + X_n^{pxn} \qquad (3)$$

The above heteronomial shows that each variable in the model can have its own power. The powers can be negative, zero (special code for using the function ln(x)), and positive. The non-zero powers can be integers or floating-point numbers. Using floating point negative and zero powers requires the data to be positive.

I can generalize equation (3) further by using the following form:

$$f(Y, yp) = a + b_1*f(X_1, px1) + b_2*f(X_2, px2) + \cdots + b_n*f(X_n, pxn) \qquad (4)$$

Where f(x,p) = x^p when $p \neq 0$
= ln(x) when p = 0 (5)

This study will focus on heteronomials with the above kind of power transformations. Using other functions like trigonometric and hyperbolic functions requires a more elaborate power-coding scheme to map special powers to special functions. The power management scheme uses distinct initial powers, power increments, and final powers for each variable.

Please note that the total number of models tested is the product of the total number of transformations for each variable. The total number of models can easily increase to very high values with the increase in the number of transformations for each variable.

This study looks at the selection of the best heteronomials models using Python's xlwings package and Excel data files. The Python code will replace Excel VBA, since the xlwings package works well with opened Excel files. This feature allows you to see changes in the Excel sheets instantly.

The Case of One Independent Variable

Let's start with the simple case of the regression model shown next:

$$Y^{yp} = a + b_1 * X_1^{px1} \qquad (6)$$

The Excel workbook bestmlr1.xlsx is the one I use to handle modeling with equation (6). The workbook has the following sheets:

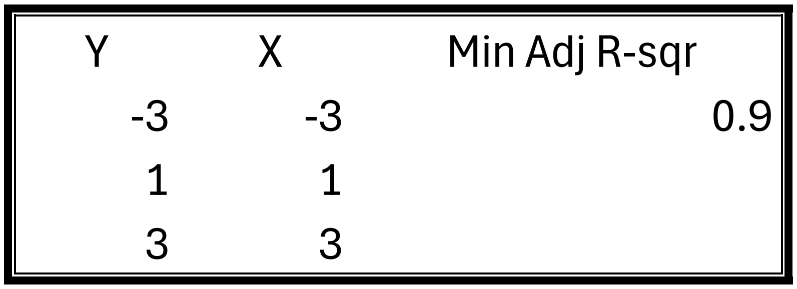
- The **Data** sheet has the following columns (see Figure 1):
  - Column A contains the values for the dependent variable Y. The column has the header Y in the first row.
  - Column B contains the values for the dependent variable X. The column has the header X in the first row.
  - Column C contains the values for the transformed values of variable Y. The column has the header Yt in the first row.
  - Column D contains the values for the transformed values of variable X. The column has the header Xt in the first row.
- The **Transf** sheet contains the information for the transformation ranges. It has the following columns (see Figure 2):

  - Column A has the header Y in row 1. Rows 2, 3, and 4 contain the values for the initial power, power increment, and final power used with variable Y.
  - Column B has the header X in row 1. Rows 2, 3, and 4 contain the values for the initial power, power increment, and final power used with variable X. In the case of more than one independent variables, columns C and beyond store the data for initial powers, power increments, and final powers.
  - Column C has the header **Min Adj R-sqr** in row 1. Row 2 contains the minimum value for the adjusted R-square statistic used to qualify a model to be listed in the sheet **List**.
- Sheet **Results** displays the results for the best model (see Figure 3). They include:
  - The adjusted R-square value in cell B1.
  - The F statistic in cell B2.
  - The p-Value for the F statistic in cell B3.
  - The AIC statistic in cell B4.
  - Cells C2 to F2 display the powers of Y and X for the best heteronomial.
  - Cells C3 to F3 display the intercept and slope for the best heteronomial.
  - Cells C4 to F4 display the intercept and slope for the best heteronomial.
  - Cells C5 to F5 display the p-Values for the intercept and slope for the best heteronomial.
- Sheet **List** displays the list of models with qualifying adjusted R-square values (as appearing in cell C2 of the **Transf** sheet). Figure 4 shows a partial view of the results AFTER I sorted the columns using Excel’s sorting feature. The sheet has the following columns:
  - Column A displays the values for the adjusted R-square statistic.
  - Column B displays the values for powers of variable Y.
  - Column C displays the values for powers of variable X.
  - Column D displays the values for the intercepts.
  - Column E displays the values for the slopes.
- Sheets **Yt** and **Xt** contain columns of the transformed values of variables Y and X, respectively. I regard these sheets as *scratch* sheets that store intermediate data. The program calculates the various transformed values

once, stores them in these sheets, and then copy them to the **Data** sheet as needed. This scheme reduces computational time.

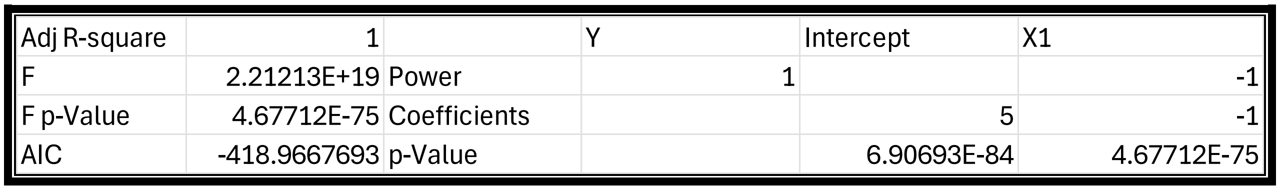
Y	X	Yt	Xt
4	1	64	1
4.5	2	91.125	8
4.666666667	3	101.6296297	27
4.75	4	107.171875	64
4.8	5	110.592	125
4.833333333	6	112.912037	216
4.857142857	7	114.5889213	343
4.875	8	115.8574219	512
4.888888889	9	116.8504801	729
4.9	10	117.649	1000

Figure 1. The Data sheet.



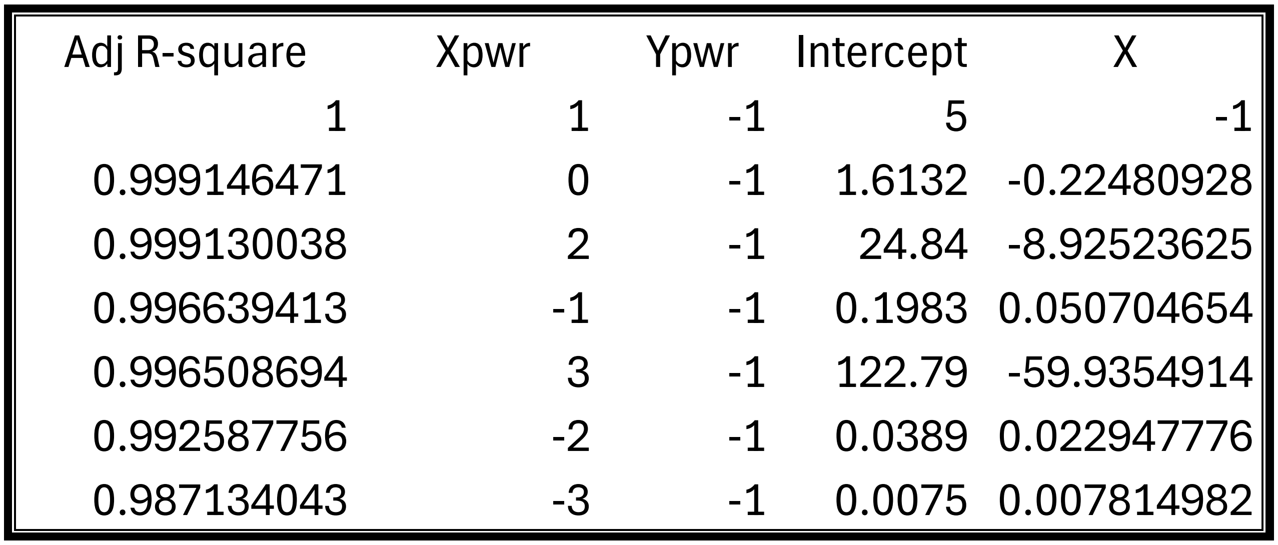
Y	X	Min Adj R-sqr
-3	-3	0.9
1	1	
3	3	

Figure 2. The Transf sheet.



Adj R-square	1		Y	Intercept	X1
F	2.21213E+19	Power	1		-1
F p-Value	4.67712E-75	Coefficients		5	-1
AIC	-418.9667693	p-Value		6.90693E-84	4.67712E-75

Figure 3. The Results sheet.



Adj R-square	Xpwr	Ypwr	Intercept	X
1	1	-1	5	-1
0.999146471	0	-1	1.6132	-0.22480928
0.999130038	2	-1	24.84	-8.92523625
0.996639413	-1	-1	0.1983	0.050704654
0.996508694	3	-1	122.79	-59.9354914
0.992587756	-2	-1	0.0389	0.022947776
0.987134043	-3	-1	0.0075	0.007814982

Figure 4. The List sheet (partial view).

Here is the Python code for the calculations that determine the heteronomial model. Makes sure that you have already installed the Python packages statsmodels, pandas, xlwings, numpy, and pyttsx3.

```
import statsmodels.api as sm
import pandas as pd
import xlwings as xw
import numpy as np
import pyttsx3

def get_column_letter(num):
    letters = ''
    while num:
        mod = (num - 1) % 26
        letters += chr(mod + 65)
        num = (num - 1) // 26
    return ''.join(reversed(letters))


def fx(x,pwr):
  if pwr > 0:
    return x**pwr
  elif pwr < 0:
    return 1/x**abs(pwr)
  else:
    return np.log(x)

def calcColumns(sheet):
  col = 0
  while 1:
    col += 1
    if sheet.range(1,col).value is None:
      break
  return col-1

e = pyttsx3.init()
```

```
rate = e.getProperty('rate')
e.setProperty('rate', rate-50)

# Load your data (replace with your actual data)
wb = xw.Book('bestmlr1.xlsx')
sheetData = wb.sheets['Data']
sheetTransf = wb.sheets['Transf']
sheetRes = wb.sheets['Results']
sheetList = wb.sheets['List']
sheetYt = wb.sheets['Yt']
sheetXt = wb.sheets['Xt']
# clear sheets
sheetList.range('A2:Z10000').value = ""
sheetYt.range('A1:Z10000').value = ""
sheetXt.range('A1:Z10000').value = ""
row = 1
while 1:
  row += 1
  if sheetData.range(row,1).value is None:
    break
maxrows = row - 1

yPwrs = sheetTransf.range("A2:A4").value
xPwrs = sheetTransf.range("B2:B4").value
minAdjR2 = sheetTransf.range("C2").value

yCols = 0
for yPwr in np.arange(yPwrs[0],yPwrs[2]+yPwrs[1],yPwrs[1]):
  yCols += 1
  for row in range(2,maxrows+1):
    sheetYt.range(row-1,yCols).value = fx(sheetData.range(row,1).value,yPwr)

xCols = 0
for xPwr in np.arange(xPwrs[0],xPwrs[2]+xPwrs[1],xPwrs[1]):
  xCols += 1
  for row in range(2,maxrows+1):
    sheetXt.range(row-1,xCols).value = fx(sheetData.range(row,2).value,xPwr)


bestAdjR2 = 0
gRow = 1

e.say("Initialization Done")
e.runAndWait()

yCols = 0
for yPwr in np.arange(yPwrs[0],yPwrs[2]+yPwrs[1],yPwrs[1]):
  yCols += 1
  s = get_column_letter(yCols)
  s2 = s + "1:" + s + str(maxrows-1)
  Ysr = sheetYt.range(s2)
  Ysr.copy()
  Ydr = sheetData.range("C2")
  Ydr.paste()

  xCols = 0
  for xPwr in np.arange(xPwrs[0],xPwrs[2]+xPwrs[1],xPwrs[1]):
    xCols += 1
    s = get_column_letter(xCols)
    s2 = s + "1:" + s + str(maxrows-1)
    Xsr = sheetXt.range(s2)
    Xsr.copy()
    Xdr = sheetData.range("D2")
    Xdr.paste()

    data = sheetData.range((2,3),(maxrows,4)).value

    # Convert data to a pandas DataFrame
    df = pd.DataFrame(data)
    df.columns = ['Yt','Xt']
    # Define dependent and independent variables
    y = df['Yt']
    X = df['Xt']
    # Add a constant to the independent variables (for the intercept term)
    X = sm.add_constant(X)

    # Fit the model
    model = sm.OLS(y, X).fit()

    if model.rsquared_adj > bestAdjR2:
      bestAdjR2 = model.rsquared_adj
      bestYpwr = yPwr
      bestXpwr = xPwr
      bestModel = model

    if model.rsquared_adj >= minAdjR2:
      gRow += 1
      sheetList.range((gRow,1)).value = model.rsquared_adj
      sheetList.range((gRow,2)).value = yPwr
      sheetList.range((gRow,3)).value = xPwr
      sheetList.range((gRow,4)).value= model.params.iloc[0]
      sheetList.range((gRow,5)).value= model.params.iloc[1]

model = bestModel

# Print the summary of the best regression results
print(model.summary())
print("Best Y power", bestYpwr)
print("Best X power", bestXpwr)

sheetRes.range("B1").value = model.rsquared_adj
sheetRes.range("B2").value = model.fvalue
sheetRes.range("B3").value = model.f_pvalue
sheetRes.range("B4").value = model.aic
sheetRes.range("D2").value = bestYpwr
sheetRes.range("F2").value = bestXpwr
sheetRes.range("E3").value = model.params.iloc[0]
sheetRes.range("F3").value = model.params.iloc[1]
sheetRes.range("E4").value = model.pvalues.iloc[0]
sheetRes.range("F4").value = model.pvalues.iloc[1]

e.say("Calculations Done")
e.runAndWait()
```

The program performs the following general tasks:

- Initialize the text-to-speech engine.
- Connect Python with the Excel workbook using the xw.Book() function.
- Initialize the variables that access the different sheets in the Excel workbook.
- Clear sheets **List**, **Yt**, and **Xt**.
- Determine the maximum rows in sheet **Data**.
- Obtain and store the ranges of transformations for the variables X and Y. Also obtain the minimal adjusted R-square value.
- Populate the various columns of sheets **Xt** and **Yt** with the values of variables X and Y with different powers.
- Start the main process using two nested loops to access the transformed values of Y and X and copy them in sheet **Data**.
- Create the data frame object df that maps the cells in sheet **Data** for the transformed values of variables X and Y.
- Select the variables for the multiple linearized regression. The column labeled Yt supply the transformed values of variable Y. The columns labeled Xt supplies the transformed values of variable X. In the case of using two independent variables, the labels for X are Xt1, and Xt2. In the case of using three independent variables, the labels for X are Xt1, Xt2, and Xt3.
- Add a constant term to the regression model.
- Call sm.OLS(y, X).fit to obtain a multiple regression model that is stored in object model.
- Determine if the new model object has a better adjusted R-square value than the one store in variable bestAdjR2. If this condition is true, the program stores the adjusted R-square value, the powers for X and Y, and the model object.
- Determine if the new model object has an adjusted R-square value that is greater or equal to the one store in variable minAdjR2. If this condition is true, the program lists the current model results in the next available row in sheet **List**. You should manually sort the results based on the values of the adjusted R-square values.
- Display on the console the summary of the best regression model object along with the best powers.
- Populate sheet **Results** with the results of the best regression model.
- Announce the end of the computations.

Open the Excel file bestmlr1.xlsx before you run the Python program. If needed, enter or edit the data in sheets **Data** and **Transf**. As the Python program runs you can see changes to the cells in sheets like **Data** and **List**. When the Python program ends, it displays results in the console and in sheets **List** and **Results**.

The Python code contains the function fx that transforms the value of a variable using a power value. The current implementation of function fx handles negative, zero, and positive powers. Notice that the power zero causes function fx to return the natural logarithm value of x.

Should you desire to use other functions for transformations, then you need to code function fx to detect the supplied power values and use them to evaluate a special function. For example, if you are using the range of -3 to 3 in increments of 1 for regular powers, you can use the powers of 4 and 5 to evaluate, for example, the sine and cosine functions. The function fx should use separate if statements to detect the value of 4 and 5 and return sin(x) and cos(x), respectively. The code for function fx would look like:

```
def fx(x,pwr):
  if pwr == 4:
    return np.sin(x)
  if pwr == 5:
    return np.cos(x)
  if pwr > 0:
    return x**pwr
  elif pwr < 0:
    return 1/x**abs(pwr)
  else:
    return np.log(x)
```

As show above, function fx should detect the special coded powers first.

Models for Heteronomials of Order Two and Up

The study includes versions of bestmlr1 that work with heteronomials of orders 2 to 5. The code for these functions is very similar to that of bestmlr1. Of course, since these programs have two or more independent variables, the code has expanded parts to handle the additional variables. For example, the Python code uses three or more nested for loops to prepare the data for regression calculations. Likewise, the Excel workbooks have additional columns in the sheets mentioned earlier to accommodate the additional variables. The workbooks have additional *scratch* sheets to store transformed data.

You will find the code for the various versions of the bestmlr programs and Excel files in the ZIP file for this project. The ZIP file contains the PDF version of the document, and the files listed in the next table.

Number of Independent Variables	***Files***
1	bestmlr1.xlsx bestmlr1.py
2	bestmlr2.xlsx bestmlr2.py
3	bestmlr3.xlsx bestmlr3.py
4	bestmlr4.xlsx bestmlr4.py
5	bestmlr5.xlsx bestmlr5.py