Flexible Rational Heteronomial Model Selection

by
Namir Clement Shammas
THE HERETIC EMPIRICIST

Contents

1/Introduction

Polynomials are very popular constructs for math, calculus, and curve fitting. A polynomial has one or more terms with the independent variable raised to an integer power. Parallel to polynomials are loosely named multivariable constructs that involve different independent variables. I will rename these constructs ***heteronomials***. An example of a very simple linear heteronomial, often used in simple multiple linear regression is:

$$Y = a + b_1X_1 + b_2X_2 \qquad (1)$$

Where X_1 and X_2 are the independent variables and Y is the dependent variable. I call equation (1) a linear heteronomial because each variable is linear. A simple example of a non-linear heteronomial is:

$$Y = a + b_1/X_1 + b_2X_2^2 \qquad (2)$$

In equation (2) variable X_1 is raised to the power –1 and variable X_2 is raised to the power 2. More advanced heteronomials involve more independent variables and different powers like –3, –2, –2.23, 3, 0.5, 1.5, 1.76, 2.84, and so on. The power zero is specially translated to the ln(x) function while the other powers are taken at face values. In this study, I limit the powers of the heteronomials to negative, zero (to use function ln(x)), and positive numbers (both integers and reals). The general form of a heteronomial is:

$$Y^{py} = a + b_1 X_1^{px1} + b_2X_2^{px2} + \ldots + b_nX_n^{pxn} \qquad (3)$$

Notice that each variable in equation (3) **can have** a power other than 1.

Padé polynomials take the ratio of two polynomials. The general form of a Padé polynomial is:

$$Y_{n,m} = (a+b_1*X+b_2*X^2+\ldots+b_m*X^m) / (1+c_1*X+c_2*X^2+\ldots+c_n*X^n) \qquad (4)$$

Notice that the intercept of the denominator polynomial is always 1. The Padé polynomials have orders of m and n which may or may not be equal. Moreover, n may be greater than m, or vice versa.

Now let me introduce you to ***rational heteronomials***. They are to heteronomials what Padé polynomials are to regular polynomials. The general form of rational heteronomials (with cross-product terms) is:

$$Y = (a + \sum_{i,j,k=1}^{n+1} b_k\, X_i^{p(i)}\ \ X_j^{p(j)})/(1 + \sum_{i,j,k=1}^{m+1} c_k\, X_i^{p(i)}\ \ X_j^{p(j)}) \qquad (5)$$

Each variable in equation (5) has its own power. The minimal condition for rational heteronomials is that **at least one single-variable term or cross-product must appear in both numerator and denominator**. The other terms (single-variables or cross-products) can appear in either the numerator or denominator. To obtain a single-variable term we multiply a variable by a unity variable (one containing a column of ones).

The linearized multiple regression model based on equation (5) is as follows.

$$Y = a + \sum_{i,j,k=1}^{n+1} b_k\, X_i^{p(i)}\ \ X_j^{p(j)} - Y * \sum_{i,j,k=1}^{m+1} c_k\, X_i^{p(i)}\ \ X_j^{p(j)} \qquad (6)$$

Notice that the terms in the denominator originally in equation (5) are now multiplied by Y and these terms are also **subtracted** from the terms in the numerators of equation (5).

☛ I highly recommend that you dedicate a fast computer to perform the calculations in this study. As the number of heteronomials terms, number of candidate variables, number of data points, and ranges of power increase, the computing time can be formidable—we are talking about days of CPU work!

2/Case of Two Terms

Now that I have introduced you to flexible rational heteronomials, you may ask about how this study will work with them. Notice that the title of the study includes the word *flexible*, and for a very good reason. The first case in this study focuses on selecting the best rational heteronomial model, based on:

- The calculations involve N *candidate* independent variables (where N > 1). I use the term *candidate* because it is possible that some of these variables may not end up in the final best model. The candidate variables appear in uppercase X. The variables that DO end up in the best model are the *selected* variables. The indices selected candidate variables appear in lowercase ix. So, please keep these two categories of variables in mind. In addition, the calculations use a *ghost* variable at index N+1. That variable is filled with 1s and serves to create single-variable terms when a candidate variable is multiplied or divided by it.

- The study selects between two to the four independent variables (i, j, k, and m from the set of N variables) that best fit the model:

$Y = (a + b*X_i^{pi} *X_j^{p\,j})/(1 + c* X_k^{pk} *X_m^{pm})$ (7a)
$Y = (a + b*X_i^{pi})/(1 + c* X_k^{pk} *X_m^{pm})$ (7b)
$Y = (a + b*X_i^{pi} *X_j^{p\,j})/(1 + c* X_k^{pk})$ (7c)
$Y = (a + b*X_i^{pi})/(1 + c* X_k^{pk})$ (7d)

- Where a, b, c, pi, pj, pk, and pm are to be determined in the calculations and yield the highest adjusted R-square statistic. When the index of any variable is N+1, the program uses a ghost/virtual unity vector, making that term a single-variable term. Since equations (7a-d) have one term in the numerator and one term in the denominator, there are no other flexible terms that may appear in either the numerator or the denominator.

The optimization for equations (7a-d) involves virtual nested loops that select the combinations of the various candidate variables and the various powers for any of the equations (7a-d). Since equations (7a-d) have two terms (the minimum requirement) they are the simplest case for flexible rational heteronomials. The questions remain:

- Which variables will be selected if you supply more than four variables?
- Where do the selected variables appear (numerator or denominator)?
- Whether each of the two terms will be a single-variable term or a cross-product term.

3/Case of Three Terms

The second case in this study focuses on selecting the best rational heteronomial model, based on:

- The calculations involve N *candidate* independent variables (where N > 2).
- The study selects three to six independent variables for the best fit model. Here is a partial list to give you an idea of the various combinations of terms:

$Y = (a + b_1*X_{i1}^{pi1} + b_2*X_{i2}^{pi2})/(1 + c_1*X_{i3}^{pi3})$ (8a)
$Y = (a + b_1*X_{i1}^{pi1})/(1+ c_1*X_{i2}^{pi2} + c_2*X_{i3}^{pi3})$ (8b)
$Y = (a + b_1*X_{i1}^{pi1}*X_{j1}^{pj1} + b_2*X_{i2}^{p2})/(1 + c_1*X_{i3}^{p3})$ (8c)
$Y = (a + b_1*X_{i1}^{pi1})/(1+ c_1*X_{i2}^{pi2}*X_{j2}^{pj2} + c_2*X_{i3}^{pi3})$ (8d)
$Y = (a + b_1*X_{i1}^{pi1}*X_{j1}^{pj1} + b_2*X_{i2}^{p2})/(1 + c_1*X_{i3}^{pi3}*X_{j3}^{pj3})$ (8e)

$$Y = (a + b_1*X_{i1}^{pi1}*X_{j1}^{pj1} + b_2*X_{i2}^{p2}*X_{j2}^{pj2})/(1 + c_1*X_{i3}^{pi3}*X_{j3}^{pj3}) \qquad (8f)$$

- Where the regression coefficients, powers, and *orientation flags* are to be determined in the calculations and yield the highest adjusted R-square statistic to choose the best model.

The optimization for equations (8a-f) involves a **single** set of (virtual) nested loops that select the combinations of the various variables, the various powers, and the various orientation flags. The code contains several if statements that guide the calculations to work with different models based on the values of the orientation flags. In addition, the calculations use the ghost/virtual unity column vector X_{N+1} to obtain single-variable terms by using $X_i^{pi} * X_{N+1}$. The questions remain:

- Which variables will be selected if you supply more than six variables?
- Where do the selected variables appear (numerator or denominator)?
- Whether each of the three terms will be a single-variable term or a cross-product term.
- Will the second term appear in the enumerator or the denominator?

The above questions also apply to the case of four terms, which I will present in the next section.

4/Case of Four Terms

The third case in this study focuses on selecting the best rational heteronomial model, based on:

- The calculations involve N *candidate* independent variables.
- The study selects four to eight independent variables. The terms are combinations of single-variable terms and/or cross-product terms.

- The regression coefficients, powers, and orientation flags are to be determined in the calculations and yield the highest adjusted R-square statistic to choose the best in the above models.

As with models with fewer terms, there is always at least one term in the numerator and in the denominator. The other two terms can appear either in the numerator or in the denominator. Again, the orientation flags help select which model to process.

The optimization for equations of four-term rational heteronomials involves a **single** set of (virtual) nested loops that select the combinations of the various variables, the various powers, and the various orientation flags. The code contains several if statements that guide the calculations to work with different models based on the values of the orientation flags.

5/The Excel Files for the Case of Two Terms

The model optimization process uses Excel files to supply input and output (the latter is also echoed to a diary text file). The Excel file has the sheet **Data** to store the values for the dependent variable and all the candidate independent variables. The sheet **Transf** contains the data that guides the calculations and transformations

The Sheet **Data**

The sheet **Data** contains the input data and the following columns (see Figure 7.1):

1. The first row has the headers that **you must enter**. The text in the headers does not affect the calculations. There are there to clarify the information in rows below.
2. Cell A1 has the header Y and the data for the dependent variable starting in cells A2 and below.
3. Columns B1, C1, and beyond have the header X1, X2, and so on. Rows 2 and down have the values for the various **candidate** independent variables.

☛ In this study the dependent variable is calculated **without injecting any errors**. The reason for this is to test if the various methods can find the best correct model. Injecting errors in the dependent variable would easily steer the calculations away from the best correct model.

The Sheet **Transf**

The sheet **Transf** contains the indices for selecting variables and their transformations and the following columns and rows (see Figure 7.2):

1. The first row contains groups of headers **that you must enter**. The first group of headers is ix1, ix2, ix3 and ix4 (notice that I am using lower case ix to indicate the indices for the candidate variables). The second group of headers is px1, px2, px3, and px4.
2. The second row has the tag **Lb** in column A to indicate that the cells to its right side contain lower bound values. These values are the lower limits of

the indices for the candidate independent variables, and for the power values.

3. The third row has the tag **Ub** in column A to indicate that the cells to its right side contain upper bound values. These values are the higher limits for the indices for the candidate independent variables, and for the power values.

Please note that the values for rows 2 and 3 define the ranges for candidate variables and the ranges of powers used.

Note that you should enter all the headers in the Excel sheets. The names that you use are your choice and do not affect the calculations. The headers help you easily identify the input data and results. I use the headers X_n to indicate the candidate independent variables. I use the headers ix_n to indicate the indices to the selected candidate independent variables.

Y	X1	X2	X3	X4	X5	X6
8.473	1	7	51	20	39	46
14.37	2	45	87	21	44	4
130.6	3	90	7	20	68	8
6.525	4	12	90	22	12	99
1.835	5	20	91	20	60	78
17.56	6	72	45	21	36	79
75.66	7	81	3	21	95	32
3.38	8	29	60	22	44	8
14.52	9	78	19	21	74	11
1.636	10	19	38	22	77	61
1.481	11	53	86	21	95	20
0.408	12	4	27	20	90	3
5.481	13	88	72	20	42	100
1.24	14	48	84	20	83	17

Figure 5.1. Sample sheet ***Data*** *for 6 candidate independent variables in file data2.xlsx.*

Figure 5.2 shows a sample sheet **Transf**. Based on the data in that sheet, the set of models examined has six candidates. So, the values under columns ix1 through ix4 range from 1 to 6 to index candidate variables X_1 through X_6, respectively. The powers range from –2 to 2 and appear as such under columns px1 through px4.

	ix1	ix2	ix3	ix4	px1	px2	px3	px4
Lb	1	1	1	1	-2	-2	-2	-2
Ub	6	6	6	6	2	2	2	2

Figure 5.2. Sample sheet ***Transf*** *for 4 independent variables with two cross-product terms in file data2.xlsx.*

The Sheet **Results**

The sheet **Results** is one of the output sheets. Figure (7.3) shows a sample sheet for two cross-product terms. The sheet has the following rows:

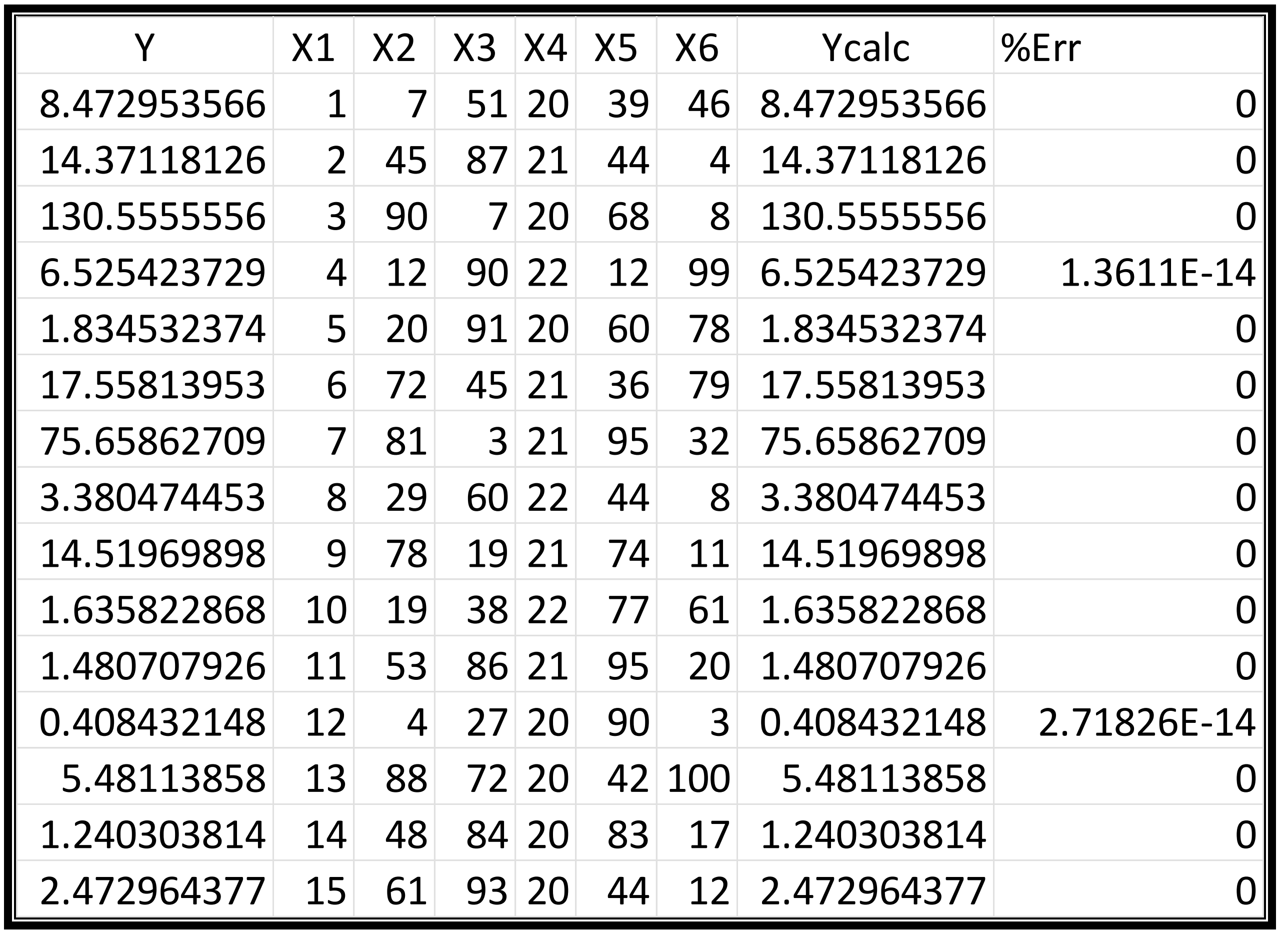
1. Row has the headers that **you need to enter**!
2. Row 2 has the following groups of values:
    a. Cell A2 has the adjusted R-square values.
    b. Cells B2 through E2 contain the indices for the *selected* variables. Thus, for example, in Figure (5.3) the index ix1 (i.e., the first) points to the *candidate* variable X_2 in sheet **Data**. Likewise, the index ix2 (i.e., the second) points to the *candidate* variable X_1 in sheet **Data**. The program will display in the console (and echo it in the diary file), the best model showing the actual candidate variables that were selected to be in the best model.
    c. Cells F2 and I2 contain the values for the best powers used with the selected variables.
    d. Cell B3 contains the intercept of the model.
    e. Cells C3 and D3 contain the slopes of the two terms.

AdjR-sqr	ix1	ix2	ix3	ix4	px1	px2	px3	px4
1	2	1	5	3	1	-1	1	1
Coeffs	2	25	0.01					

*Figure 5.3. Sample sheet **Results** for 4 selected independent variables in file data2.xlsx.*

The Sheet **Project**

The sheet Project copies the input data from the sheet **Data** and inserts two new columns. The first one is for the calculated values of the dependent variable using the best model. The second new column is the percentage error in calculating the values of the dependent variable. Figure (5.4) shows a partial sample view of sheet **Project**.



Y	X1	X2	X3	X4	X5	X6	Ycalc	%Err
8.472953566	1	7	51	20	39	46	8.472953566	0
14.37118126	2	45	87	21	44	4	14.37118126	0
130.5555556	3	90	7	20	68	8	130.5555556	0
6.525423729	4	12	90	22	12	99	6.525423729	1.3611E-14
1.834532374	5	20	91	20	60	78	1.834532374	0
17.55813953	6	72	45	21	36	79	17.55813953	0
75.65862709	7	81	3	21	95	32	75.65862709	0
3.380474453	8	29	60	22	44	8	3.380474453	0
14.51969898	9	78	19	21	74	11	14.51969898	0
1.635822868	10	19	38	22	77	61	1.635822868	0
1.480707926	11	53	86	21	95	20	1.480707926	0
0.408432148	12	4	27	20	90	3	0.408432148	2.71826E-14
5.48113858	13	88	72	20	42	100	5.48113858	0
1.240303814	14	48	84	20	83	17	1.240303814	0
2.472964377	15	61	93	20	44	12	2.472964377	0

*Figure 5.4. Sample sheet **Project** for 4 independent variables with two cross-product terms in file data2.xlsx.*

6/Code for the Case of Two Terms

Here is the MATLAB code for file clemTerm2.m that yields the results shown in the above figures.

```
clc
```

```
close all
clear all

global mdl
global xdata
global ydata
global maxVarIdx

warning("off")
dt = datetime('now','TimeZone','local','Format','y-MMM-d HH_mm_ss');
dtstr = string(dt);
filename = strcat(pwd,'\data2.xlsx');
delete("data2_*.txt");
outFile = strcat("data2_",dtstr,".txt");
diary(outFile)

fprintf("Program  clemTerm2\n\n");
fprintf("%s\n", dtstr);
fprintf("Please wait ...\n\n");
fprintf("Excel file used is %s\n", filename);
fprintf("Diary file used is %s\n", outFile);
xyData = readmatrix(filename,'Sheet','Data');

ydata = xyData(:,1);
xdata = xyData(:,2:end);
[maxrows,maxcols] = size(xdata);
xyTransf = readmatrix(filename,'Sheet','Transf');
xyTransf = xyTransf(:,2:end);

% Important assignment - must match the number of X columns in sheet
% Transf of the Excel file
numX = 4;
Lb = xyTransf(1,:);
Ub = xyTransf(2,:);
% augment indices for variables to include the index of the dummy unity
% variable
Ub(1,1:numX) = Ub(1,1:numX)+1;
maxVarIdx = Ub(1,1:numX);

nVars = length(Lb);
iFrom = Lb;
iTo = Ub;
iStep = 1 + zeros(1,nVars);
idx = Lb;
bStop = false;
iter = 0;
fprintf("------------- Calculating Max Iters ----------------\n");
while ~bStop
  iter = iter +1;
  % Start implementing the quasi-nested loops
  idx(1) = idx(1) + iStep(1);
  if idx(1) > iTo(1)
    idx(1) = iFrom(1);
    for i = 2:nVars
      idx(i) = idx(i) + iStep(i);
      if idx(i) > iTo(i) && i < nVars
        idx(i) = iFrom(i);
      elseif idx(i) > iTo(i) && i == nVars
        bStop = true;
      else
        break
      end
    end
  end
end
str = num2str(iter);
str = fliplr(regexprep(fliplr(str), '(\d+\.)?(\d{3})(?=\S+)', '$1$2,'));
fprintf("Max iters = %s\n", str);
MaxIters = iter;
FivePercent = fix(iter/20);

iFrom = Lb;
iTo = Ub;
iStep = 1 + zeros(1,nVars);
idx = Lb;
iter = 0;

bestX = zeros(1,length(Lb));
bestFx = 0;
% --------------------------------- start main loop
bStop = false;
iter = 0;
fprintf("------------- Maximizing Fx ----------------\n");
while ~bStop
  iter = iter +1;
  if mod(iter, FivePercent) == 0
    percent = 100*iter/MaxIters;
    fprintf("Progress %5.2f%%\n", percent);
  end

  r2adj = myFit(idx);
  if r2adj > bestFx
    bestFx = r2adj;
    bestX = idx;
    bestMdl = mdl;
    fprintf("Iter %d, Fx = %e, X=[", iter, bestFx)
    fprintf("%d, ", bestX);
    fprintf("]\n");
    if r2adj == 1, break; end
  end
  % Start implementing the quasi-nested loops
  idx(1) = idx(1) + iStep(1);
  if idx(1) > iTo(1)
    idx(1) = iFrom(1);
    for i = 2:nVars
      idx(i) = idx(i) + iStep(i);
      if idx(i) > iTo(i) && i < nVars
        idx(i) = iFrom(i);
      elseif idx(i) > iTo(i) && i == nVars
        bStop = true;
      else
        break
      end
    end
  end
end
fprintf("Regression model\n")
format long
AdjR2 = bestMdl.Rsquared.Adjusted;
fprintf("Adjusted R-square = %14.12f\n", AdjR2);
mdl = bestMdl
bestX = [AdjR2 bestX];

res = zeros(2,length(bestX));
res(1,:) = bestX;
res(2,2) = mdl.Coefficients{1,1}';
res(2,3) = mdl.Coefficients{2,1}';
res(2,4) = mdl.Coefficients{3,1}';
writematrix(res(1,:), filename, 'Sheet', 'Results', 'Range', 'A2:Z2');
writematrix(res(2,2:4), filename, 'Sheet', 'Results', 'Range', 'B3:Z3');

% now do the % projections
ycalc = project(res);
percErr = 100.*(ycalc-ydata)./ydata;
ProjMat = [ydata xdata ycalc percErr];
writematrix(ProjMat, filename, 'Sheet', 'Project', 'Range', strcat("A2:Z",
num2str(1+length(ydata))));

sNum = sprintf("%e + (%e) * ", res(2,2), res(2,3));
idx = res(1,2:numX+1);
px = res(1,numX+2:2*numX+1);
% ops = res(1,2*numX+2:2*numX+3);

if px(1)==0
  sX1 = sprintf("log(X%d)", idx(1));
elseif px(1)==4
  sX1 = sprintf("sqrt(X%d)", idx(1));
elseif px(1)==-4
  sX1 = sprintf("1/log(X%d)", idx(1));
elseif px(1)==1
  sX1 = sprintf("X%d", idx(1));
elseif px(1)>1
  sX1 = sprintf("X%d^%d", idx(1), px(1));
elseif px(1)==-1
  sX1 = sprintf("1/X%d", idx(1));
elseif px(1)<-1
  sX1 = sprintf("1/X%d^%d", idx(1), abs(px(1)));
else
  sX1 = sprintf("X%d", idx(1));
end

if px(2)==0
  sX2 = sprintf("log(X%d)", idx(2));
elseif px(2)==4
  sX2 = sprintf("sqrt(X%d)", idx(2));
elseif px(2)==-4
  sX2 = sprintf("log(X%d)", idx(2));
elseif px(2)==1
  sX2 = sprintf("X%d", idx(2));
elseif px(2)>1
  sX2 = sprintf("X%d^%d", idx(2), px(2));
```

```
elseif px(2)==-1
  sX2 = sprintf("1/X%d", idx(2));
elseif px(2)<-1
  sX2 = sprintf("1/X%d^%d", idx(2), abs(px(2)));
else
  sX2 = sprintf("X%d", idx(2));
end


if px(3)==0
  sX3 = sprintf("log(X%d)", idx(3));
elseif px(3)==4
  sX3 = sprintf("sqrt(X%d)", idx(3));
elseif px(3)==-4
  sX3 = sprintf("log(X%d)", idx(3));
elseif px(3)==1
  sX3 = sprintf("X%d", idx(3));
elseif px(3)>1
  sX3 = sprintf("X%d^%d", idx(3), px(3));
elseif px(3)==-1
  sX3 = sprintf("1/X%d", idx(3));
elseif px(3)<-1
  sX3 = sprintf("1/X%d^%d", idx(3), abs(px(3)));
else
  sX3 = sprintf("X%d", idx(3));
end

if px(4)==0
  sX4 = sprintf("log(X%d)", idx(4));
elseif px(4)==4
  sX4 = sprintf("sqrt(X%d)", idx(4));
elseif px(4)==-4
  sX4 = sprintf("log(X%d)", idx(4));
elseif px(4)==1
  sX4 = sprintf("X%d", idx(4));
elseif px(4)>1
  sX4 = sprintf("X%d^%d", idx(4), px(4));
elseif px(4)==-1
  sX4 = sprintf("1/X%d", idx(4));
elseif px(4)<-1
  sX4 = sprintf("1/X%d^%d", idx(4), abs(px(4)));
else
  sX4 = sprintf("X%d", idx(4));
end

if idx(1)<maxVarIdx(1) && idx(2)<maxVarIdx(2)
  sNum = sprintf("%s %s * %s)", sNum, sX1, sX2);
elseif idx(1)<maxVarIdx(1)
  sNum = sprintf("%s %s)", sNum, sX1);
elseif idx(2)<maxVarIdx(2)
  sNum = sprintf("%s %s)", sNum, sX2);
end

if idx(3)<maxVarIdx(3) && idx(4)<maxVarIdx(4)
  sDenom = sprintf("/(1 +(%e) * %s * %s)", res(2,4), sX3, sX4);
elseif idx(3)<maxVarIdx(3)
  sDenom = sprintf("/(1 +(%e) * %s)", res(2,4), sX3);
```

```
elseif idx(4)<maxVarIdx(4)
  sDenom = sprintf("/(1 +(%e) * %s)", res(2,4), sX4);
end

fprintf("%s\n%s)", sNum, sDenom)

format short
fprintf("Done!\n");
diary off
beep on
for i=1:3
  beep;
  pause(1);
end

function x = myfx(x,p)
  if p==0
    x = log(x);
  elseif p==4
    x = sqrt(x);
  elseif p==-4
    x = 1./log(x);
  elseif p > 1
    x = x.^p;
  elseif p < 0
    x = 1./x.^abs(p);
  end
end

function r = myFit(coeff)
  global mdl
  global xdata
  global ydata
  global maxVarIdx

  numX = length(maxVarIdx);
  [maxrows, maxcols] = size(xdata);
  y = ydata;
  idx = coeff(1:numX);
  px= coeff(numX+1:2*numX);

  x = zeros(maxrows,numX);
  x2 = zeros(maxrows,fix(numX/2));

  % check for redundant vars in a term
  if idx(1)==idx(2) || idx(3)==idx(4)
    r = 0;
    return;
  end

  for i=1:numX
    if idx(i)<maxVarIdx(i)
      x(:,i) = myfx(xdata(:,idx(i)),px(i));
    else
      x(:,i) = 1+zeros(maxrows,1);
    end
  end
  i1=-1;
  i2=0;
  for i=1:fix(numX/2)
    i1=i1+2;
    i2=i2+2;
    if idx(i1)<maxVarIdx(i1) && idx(i2)<maxVarIdx(i2)
      x2(:,i) = x(:,i1).*x(:,i2);
    elseif idx(i2)<maxVarIdx(i2)
      x2(:,i) = x(:,i2);
    elseif idx(i1)<maxVarIdx(i1)
      x2(:,i) = x(:,i1);
    end
  end
  X = [x2(:,1) -ydata.*x2(:,2)];
  mdl = fitlm(X,y);
  r = mdl.Rsquared.Adjusted;
end

function ycalc = project(res)
  global xdata
  global maxVarIdx

  numX = length(maxVarIdx);
  [maxrows, maxcols] = size(xdata);

  idx = res(1,2:numX+1);
  px = res(1,numX+2:2*numX+1);
  x = zeros(maxrows,numX);

  for i=1:numX
    if idx(i)<maxVarIdx(i)
      x(:,i) = myfx(xdata(:,idx(i)),px(i));
    else
      x(:,i) = 1 + zeros(maxrows,1);
    end
  end

  ycalc = res(2,2) + res(2,3) * x(:,1).*x(:,2);
  ycalc = ycalc ./(1 + res(2,4) * x(:,3).*x(:,4));
end
```

The above code performs the following general tasks:

- Initializes the names of the Excel file and the diary file.
- Opens the diary file to echo console output.
- Reads the input data from sheet **Data**. The program then copies the values for the dependent variable into column vector ydata. It also copies the values for the independent variables into matrix xdata.
- Initializes the row vectors Lb and Ub to contain the ranges for the indices of the candidate independent variables and their powers.

- Initializes the arrays iFrom, iTo, iStep, and idx that are used to control the virtual nested loops. The text for this task appears in red color.
- Uses the first virtual nested loops (coded in red), the program calculates the maximum number of iterations that will be needed to optimize the selected rational heteronomial. The program displays the result.
- Re-initializes the arrays iFrom, iTo, iStep, and idx (coded in red) and implement the second virtual loop to MAXIMIZE the values of the adjusted R-square statistic and determine the best rational heteronomial. The loop performs the following tasks.
    - Periodically display the current best results.
    - Invokes the function myFit() to calculate the adjusted R-square statistic for the current model. This function returns 0 when the numerator (and denominator) has multiple variables with the same selection index.
    - Test if the calculated adjusted R-square statistic is greater than the value in variable bestFx. If it is, then store and display the updated best adjusted R-square, the indices of the two variables, and the powers used with these two variables.
- Displays the results and write results in the sheets **Result**.
- Calculates the array of projected values for the dependent variable (by calling function project()) and their percentage errors. Write these results to sheet **Project**.
- Builds and displays a string that shows the best fitted model.

The MATLAB code has the function myFx() that transforms the values of a variable using the specified power p. The function has a set of if statements that map the values of p to different transformations. For example, when p is zero, the function returns the natural logarithm values. Also, when p is –4 or 4, the function returns the reciprocal of the logarithm or the square root values, respectively.

Here is the output generated by program clemTerm2:

```
Program  clemTerm2

2025-Jan-21 10_46_44
Please wait ...

Excel file used is I:\DropBox\Dropbox\MATLAB\Clement6\data2.xlsx
Diary file used is data2_2025-Jan-21 10_46_44.txt
------------- Calculating Max Iters ----------------
Max iters = 1,500,625
```

```
------------- Maximizing Fx ----------------
Iter 51, Fx = 9.958425e-02, X=[2, 1, 2, 1, -2, -2, -2, -2, ]
Iter 52, Fx = 7.656690e-01, X=[3, 1, 2, 1, -2, -2, -2, -2, ]
Iter 100, Fx = 8.880234e-01, X=[2, 1, 3, 1, -2, -2, -2, -2, ]
Iter 101, Fx = 8.916360e-01, X=[3, 1, 3, 1, -2, -2, -2, -2, ]
Iter 103, Fx = 9.036804e-01, X=[5, 1, 3, 1, -2, -2, -2, -2, ]
Iter 117, Fx = 9.363167e-01, X=[5, 3, 3, 1, -2, -2, -2, -2, ]
Iter 1325, Fx = 9.961542e-01, X=[2, 1, 7, 4, -2, -2, -2, -2, ]
Iter 1326, Fx = 9.964599e-01, X=[3, 1, 7, 4, -2, -2, -2, -2, ]
Iter 1341, Fx = 9.965571e-01, X=[4, 3, 7, 4, -2, -2, -2, -2, ]
Iter 1342, Fx = 9.975194e-01, X=[5, 3, 7, 4, -2, -2, -2, -2, ]
Iter 15748, Fx = 9.977347e-01, X=[5, 3, 7, 4, -1, -1, -2, -2, ]
Iter 62232, Fx = 9.989961e-01, X=[2, 1, 4, 7, -2, -2, -1, -2, ]
Iter 62233, Fx = 9.990467e-01, X=[3, 1, 4, 7, -2, -2, -1, -2, ]
Iter 62248, Fx = 9.990975e-01, X=[4, 3, 4, 7, -2, -2, -1, -2, ]
Iter 62249, Fx = 9.993560e-01, X=[5, 3, 4, 7, -2, -2, -1, -2, ]
Progress  5.00%
Iter 76655, Fx = 9.994101e-01, X=[5, 3, 4, 7, -1, -1, -1, -2, ]
Iter 122257, Fx = 9.998848e-01, X=[2, 1, 4, 7, -2, -2, 0, -2, ]
Iter 122258, Fx = 9.998871e-01, X=[3, 1, 4, 7, -2, -2, 0, -2, ]
Iter 122260, Fx = 9.998889e-01, X=[5, 1, 4, 7, -2, -2, 0, -2, ]
Iter 122273, Fx = 9.998958e-01, X=[4, 3, 4, 7, -2, -2, 0, -2, ]
Iter 122274, Fx = 9.999262e-01, X=[5, 3, 4, 7, -2, -2, 0, -2, ]
Iter 136680, Fx = 9.999321e-01, X=[5, 3, 4, 7, -1, -1, 0, -2, ]
Progress 10.00%
Progress 15.00%
Progress 20.00%
Progress 25.00%
Progress 30.00%
Progress 35.00%
Progress 40.00%
Progress 45.00%
Progress 50.00%
Progress 55.00%
Progress 60.00%
Progress 65.00%
Progress 70.00%
Iter 1100542, Fx = 1.000000e+00, X=[2, 1, 5, 3, 1, -1, 1, 1, ]
Regression model
Adjusted R-square = 1.000000000000

mdl =


Linear regression model:
    y ~ 1 + x1 + x2

Estimated Coefficients:
                   Estimate    SE    tStat    pValue
                   ________    __    _____    ______

    (Intercept)         2      0      Inf       0
    x1                 25      0      Inf       0
    x2               0.01      0      Inf       0


Number of observations: 100, Error degrees of freedom: 97
```

```
R-squared: 1,  Adjusted R-Squared: 1
F-statistic vs. constant model: 1.65e+32, p-value = 0
2.000000e+00 + (2.500000e+01) *  X2 * 1/X1)
/(1 +(1.000000e-02) * X5 * X3)
Done!
```

The output shows that the program can identify the correct model used to calculate the values of the dependent variable.

7/ The Excel Files for the Case of Three Terms

In this section, I focus on the Excel sheets **Transf** and **Results** for three (or more) independent variables and how they all differ from the same sheets for the case of two terms.

Sheet **Transf**

The sheet **Transf** contains the indices for selecting variables and their transformations and the following columns and rows (see Figure 9.1):

1. The first row contains groups of headers **that you must enter**. The first group of headers is ix1 through ix6. The second group of headers is px1, through px6. The third group is the orientation flags Orient1, Orient2, and Orient 2.
2. The second row has the tag **Lb** in column A to indicate that the cells to its right side contain lower bound values. These values are the lower limits of the indices for the candidate independent variables, the power values, and the orientation flag values. Notice that the orientation flag for the first term is AWLAYS 1. The orientation flag for the second term is 1. The orientation flag for third term is ALWAYS 2.
3. The third row has the tag **Ub** in column A to indicate that the cells to its right side contain upper bound values. These values are the higher limits of the indices for the candidate independent variables, the power values, and the orientation flag values. Notice that the orientation flag for the first term is AWLAYS 1. The orientation flag for the second term is 2. The orientation flag for third term is ALWAYS 2.

☛ Please note that the lower and upper limits for the orientation flag for the first term are ALWAYS 1. Likewise, lower and upper limits for the orientation flag for the last term are ALWAYS 2. The lower and upper limits for the orientation flag of any other terms are 1 and 2, respectively.

Figure 7.1 shows a sample sheet **Transf**. Based on the data in that sheet, the set of models examined have four candidate variables with X_1 through X_6. To reduce computation time, the values under columns ix1 through ix6 ranges are narrow. The ranges for the ranges px1 through px6 are all (–1, 1). The ranges for the orientation flags are as described above.

	ix1	ix2	ix3	ix4	ix5	ix6	px1	px2	px3	px4	px5	px6	Orient1	Orient2	Orient3
Lb	1	1	2	4	4	5	-1	-1	-1	-1	-1	-1	1	1	2
Ub	2	2	3	5	5	6	1	1	1	1	1	1	1	2	2

*Figure 7.1. Sample sheet **Transf** for 6 independent variables with three cross-product terms in file data3.xlsx.*

The Sheet **Results**

The sheet **Results** is one of the output sheets. Figure (7.2) shows a sample sheet for two cross-product terms. The sheet has the following rows:

1. Row has the headers that **you need to enter**!
2. Row 2 has the following groups of values:
    a. Cell A2 has the adjusted R-square values.
    b. Cells B2 to G2 contain the indices for the selected variables. So, for example, the index of first variable (under the header ix1) points to the candidate variable X_2. Likewise, the index of the second variable (under the header ix2) points to the candidate variable X_1. The console output shows the best model with the chosen candidate variables.
    c. Cells H2 to M2 contain the values for the best powers used with the selected variables.
    d. Cells N2 to P2 contain the orientation flags.

AdjR-sqr	ix1	ix2	ix3	ix4	ix5	ix6	px1	px2	px3	px4	px5	px6	Orient1	Orient2	Orient3
1	2	1	3	5	4	6	1	-1	1	1	1	1	1	1	2
Coeffs	2	5	5	2											

*Figure 7.2. Sample sheet **Results** for 6 selected independent variables in file data3.xlsx.*

8/ The Code for the Case of Three Terms

Here is the code for file clemTerm3.m. It is very similar to the code of clemTerm2.m. The additional statements examine the values of the orientation flags to determine the flow of program execution—whether the variable Xj appears in the numerator or denominator of the rational heteronomial.

```
% This program skips the transformation for log(x)

clc
close all
clear all

global mdl
global xdata
global ydata
global maxVarIdx

warning("off")
dt = datetime('now','TimeZone','local','Format','y-MMM-d HH_mm_ss');
dtstr = string(dt);
filename = strcat(pwd,'\data3.xlsx');
delete("data3_*.txt");
outFile = strcat("data3_",dtstr,".txt");
diary(outFile)

fprintf("Program  clemTerm3\n\n");
fprintf("%s\n", dtstr);
fprintf("Please wait ...\n\n");
fprintf("Excel file used is %s\n", filename);
fprintf("Diary file used is %s\n", outFile);
xyData = readmatrix(filename,'Sheet','Data');

ydata = xyData(:,1);
xdata = xyData(:,2:end);
[maxrows,maxcols] = size(xdata);
xyTransf = readmatrix(filename,'Sheet','Transf');
xyTransf = xyTransf(:,2:end);

% Important assignment - must match the number of X columns in sheet
% Transf of the Excel file
numX = 6;
Lb = xyTransf(1,:);
Ub = xyTransf(2,:);
% augment indices for variables to include the index of the dummy unity
% variable
Ub(1,1:numX) = Ub(1,1:numX)+1;
maxVarIdx = Ub(1,1:numX);
nVars = length(Lb);
iFrom = Lb;
iTo = Ub;
iStep = 1 + zeros(1,nVars);
idx = Lb;
bStop = false;
```

```
iter = 0;
fprintf("------------- Calculating Max Iters ----------------\n");
while ~bStop
  iter = iter +1;
  % Start implementing the quasi-nested loops
  idx(1) = idx(1) + iStep(1);
  % skip fx index 0 which is log(x) ... can be deleted
  if idx(1)==0, idx(1)=1; end
  if idx(1) > iTo(1)
    idx(1) = iFrom(1);
    for i = 2:nVars
      idx(i) = idx(i) + iStep(i);
      % skip fx index 0 which is log(x) ... can be deleted
      if idx(i)==0, idx(i)=1; end
      if idx(i) > iTo(i) && i < nVars
        idx(i) = iFrom(i);
      elseif idx(i) > iTo(i) && i == nVars
        bStop = true;
      else
        break
      end
    end
  end
end
str = num2str(iter);
str = fliplr(regexprep(fliplr(str), '(\d+\.)?(\d{3})(?=\S+)', '$1$2,'));
fprintf("Max iters = %s\n", str);
MaxIters = iter;
FivePercent = fix(iter/20);

iFrom = Lb;
iTo = Ub;
iStep = 1 + zeros(1,nVars);
idx = Lb;
iter = 0;

bestX = zeros(1,length(Lb));
bestFx = 0;
% --------------------------------- start main loop
bStop = false;
iter = 0;
fprintf("------------- Maximizing Fx ----------------\n");
while ~bStop
  iter = iter +1;
  if mod(iter, FivePercent) == 0
    percent = 100*iter/MaxIters;
    fprintf("Progress %5.2f%%\n", percent);
  end

  r2adj = myFit(idx);
  if r2adj > bestFx
    bestFx = r2adj;
    bestX = idx;
    bestMdl = mdl;
    fprintf("Iter %d, Fx = %e, X=[", iter, bestFx)
    fprintf("%d, ", bestX);
    fprintf("]\n");
    if r2adj == 1, break; end
  end
  % Start implementing the quasi-nested loops
  idx(1) = idx(1) + iStep(1);
  % skip fx index 0 which is log(x) ... can be deleted
  if idx(1)==0, idx(1)=1; end
  if idx(1) > iTo(1)
    idx(1) = iFrom(1);
    for i = 2:nVars
      idx(i) = idx(i) + iStep(i);
      % skip fx index 0 which is log(x) ... can be deleted
      if idx(i)==0, idx(i)=1; end
      if idx(i) > iTo(i) && i < nVars
        idx(i) = iFrom(i);
      elseif idx(i) > iTo(i) && i == nVars
        bStop = true;
      else
        break
      end
    end
  end
end
fprintf("Regression model\n")
format long
AdjR2 = bestMdl.Rsquared.Adjusted;
fprintf("Adjusted R-square = %14.12f\n", AdjR2);
mdl = bestMdl
bestX = [AdjR2 bestX];

res = zeros(2,length(bestX));
res(1,:) = bestX;
res(2,2:5) = mdl.Coefficients{1:4,1}';

writematrix(res(1,:), filename, 'Sheet', 'Results', 'Range', 'A2:Z2');
writematrix(res(2,2:5), filename, 'Sheet', 'Results', 'Range', 'B3:Z3');

% now do the % projections
ycalc = project(res);
percErr = 100.*(ycalc-ydata)./ydata;
ProjMat = [ydata xdata ycalc percErr];
writematrix(ProjMat, filename, 'Sheet', 'Project', 'Range', strcat("A2:Z",
num2str(1+length(ydata))));

sNum = sprintf("Y = %e + (%e)*", res(2,2), res(2,3));
idx = res(1,2:numX+1);
px = res(1,numX+2:2*numX+1);
ops = res(1,2*numX+2:2*numX+4);

if px(1)==0
  sX1 = sprintf("log(X%d)", idx(1));
elseif px(1)==4
  sX1 = sprintf("sqrt(X%d)", idx(1));
elseif px(1)==-4
  sX1 = sprintf("1/log(X%d)", idx(1));
elseif px(1)==1
  sX1 = sprintf("X%d", idx(1));
elseif px(1)>1
```

```
  sX1 = sprintf("X%d^%d", idx(1), px(1));
elseif px(1)==-1
  sX1 = sprintf("1/X%d", idx(1));
elseif px(1)<-1
  sX1 = sprintf("1/X%d^%d", idx(1), abs(px(1)));
else
  sX1 = sprintf("X%d", idx(1));
end

if px(2)==0
  sX2 = sprintf("log(X%d)", idx(2));
elseif px(2)==4
  sX2 = sprintf("sqrt(X%d)", idx(2));
elseif px(2)==-4
  sX2 = sprintf("1/log(X%d)", idx(2));
elseif px(2)==1
  sX2 = sprintf("X%d", idx(2));
elseif px(2)>1
  sX2 = sprintf("X%d^%d", idx(2), px(2));
elseif px(2)==-1
  sX2 = sprintf("1/X%d", idx(2));
elseif px(2)<-1
  sX2 = sprintf("1/X%d^%d", idx(2), abs(px(2)));
else
  sX2 = sprintf("X%d", idx(2));
end

s = sprintf("%s * %s", sX1, sX2);
sNum = strcat(sNum, s);

if px(5)==0
  sX5 = sprintf("log(X%d)", idx(5));
elseif px(5)==4
  sX5 = sprintf("sqrt(X%d)", idx(5));
elseif px(5)==-4
  sX5 = sprintf("1/log(X%d)", idx(5));
elseif px(5)==1
  sX5 = sprintf("X%d", idx(5));
elseif px(5)>1
  sX5 = sprintf("X%d^%d", idx(5), px(5));
elseif px(5)==-1
  sX5 = sprintf("1/X%d", idx(5));
elseif px(5)<-1
  sX5 = sprintf("1/X%d^%d", idx(5), abs(px(5)));
else
  sX5 = sprintf("X%d", idx(5));
end

if px(6)==0
  sX6 = sprintf("log(X%d)", idx(6));
elseif px(6)==4
  sX6 = sprintf("sqrt(X%d)", idx(6));
elseif px(6)==-4
  sX6 = sprintf("1/log(X%d)", idx(6));
elseif px(6)==1
  sX6 = sprintf("X%d", idx(6));
elseif px(6)>1
```

```
  sX6 = sprintf("X%d^%d", idx(6), px(6));
elseif px(6)==-1
  sX6 = sprintf("1/X%d", idx(6));
elseif px(6)<-1
  sX6 = sprintf("1/X%d^%d", idx(6), abs(px(6)));
else
  sX6 = sprintf("X%d", idx(6));
end

sDenom = sprintf("1 + (%e) * %s * %s", res(2,5), sX5, sX6);

if px(3)==0
  sX3 = sprintf("log(X%d)", idx(3));
elseif px(3)==4
  sX3 = sprintf("sqrt(X%d)", idx(3));
elseif px(3)==-4
  sX3 = sprintf("1/log(X%d)", idx(3));
elseif px(3)==1
  sX3 = sprintf("X%d", idx(3));
elseif px(3)>1
  sX3 = sprintf("X%d^%d", idx(3), px(3));
elseif px(3)==-1
  sX3 = sprintf("1/X%d", idx(3));
elseif px(3)<-1
  sX3 = sprintf("1/X%d^%d", idx(3), abs(px(3)));
else
  sX3 = sprintf("X%d", idx(3));
end

if px(4)==0
  sX4 = sprintf("log(X%d)", idx(4));
elseif px(4)==4
  sX4 = sprintf("sqrt(X%d)", idx(4));
elseif px(4)==1
  sX4 = sprintf("X%d", idx(4));
elseif px(4)==-4
  sX4 = sprintf("1/log(X%d)", idx(4));
elseif px(4)>1
  sX4 = sprintf("X%d^%d", idx(4), px(4));
elseif px(4)==-1
  sX4 = sprintf("1/X%d", idx(4));
elseif px(4)<-1
  sX4 = sprintf("1/X%d^%d", idx(4), abs(px(4)));
else
  sX4 = sprintf("X%d", idx(4));
end

if ops(2)==1
  sNum = sprintf("%s + (%e) * %s * %s", sNum, res(2,4), sX3, sX4);
else
  sDenom = sprintf("%s + (%e) * %s * %s", sDenom, res(2,4), sX3, sX4);
end
smodel = sprintf("%s)/(%s)", sNum, sDenom);
fprintf("%s\n", smodel)

format short
fprintf("Done!\n");
```

```
diary off
beep on
for i=1:3
  beep;
  pause(1);
end

function x = myfx(x,p)
  if p==0
    x = log(x);
  elseif p==4
    x = sqrt(x);
  elseif p==-4
    x = 1./log(x);
  elseif p > 1
    x = x.^p;
  elseif p < 0
    x = 1./x.^abs(p);
  end
end

function r = myFit(coeff)
  global mdl
  global xdata
  global ydata
  global maxVarIdx

  numX = length(maxVarIdx);
  [maxrows, ~] = size(xdata);
  y = ydata;
  idx = coeff(1:numX);
  px= coeff(numX+1:2*numX);
  orient = coeff(2*numX+1:2*numX+3);
  x = zeros(maxrows,numX);
  x2 = zeros(maxrows,fix(numX/2));

  % check for redundant vars in a term
  if idx(1)==idx(2) || idx(3)==idx(4) || idx(5)==idx(6)
    r = 0;
    return;
  end

  % check X1,X2 with X3,X4
  if idx(1)==idx(3) && px(1)==px(3) && ...
     idx(2)==idx(4) && px(2)==px(4) && orient(1) == orient(2)
    r = 0;
    return;
  end

  % check X1,X2 with X3,X4
  if idx(1)==idx(4) && px(1)==px(4) && ...
     idx(2)==idx(3) && px(2)==px(3) && orient(1) == orient(2)
    r = 0;
    return;
  end

  % check X1,X2 with X5,X6
  if idx(1)==idx(5) && px(1)==px(5) && ...
     idx(2)==idx(6) && px(2)==px(6) && orient(1) == orient(3)
    r = 0;
    return;
  end

  % check X1,X2 with X5,X6
  if idx(1)==idx(6) && px(1)==px(6) && ...
     idx(2)==idx(5) && px(2)==px(5) && orient(1) == orient(3)
    r = 0;
    return;
  end

  % check X3,X4 with X5,X6
  if idx(3)==idx(5) && px(3)==px(5) && ...
     idx(4)==idx(6) && px(4)==px(6) && orient(2) == orient(3)
    r = 0;
    return;
  end

  % check X3,X4 with X5,X6
  if idx(4)==idx(6) && px(4)==px(6) && ...
     idx(3)==idx(5) && px(3)==px(5) && orient(2) == orient(3)
    r = 0;
    return;
  end

  for i=1:numX
    if idx(i)<maxVarIdx(i)
      x(:,i) = myfx(xdata(:,idx(i)),px(i));
    else
      x(:,i) = 1+zeros(maxrows,1);
    end
  end

  i1=-1;
  i2=0;
  for i=1:fix(numX/2)
    i1=i1+2;
    i2=i2+2;
    if idx(i1)<maxVarIdx(i1) && idx(i2)<maxVarIdx(i2)
      x2(:,i) = x(:,i1).*x(:,i2);
    elseif idx(i2)<maxVarIdx(i2)
      x2(:,i) = x(:,i2);
    elseif idx(i1)<maxVarIdx(i1)
      x2(:,i) = x(:,i1);
    end
  end
  if orient(2)==2, x2(:,2) = -ydata.*x2(:,2); end
  X = [x2(:,1) x2(:,2) -ydata.*x2(:,3)];
  mdl = fitlm(X,y);
  r = mdl.Rsquared.Adjusted;
end

function ycalc = project(res)
  global xdata
  global maxVarIdx
  numX = length(maxVarIdx);
  [maxrows, ~] = size(xdata);

  idx = res(1,2:numX+1);
  px = res(1,numX+2:2*numX+1);
  orient = res(2*numX+2:2*numX+4);
  x = zeros(maxrows,numX);

  for i=1:numX
    if idx(i)<maxVarIdx(i)
      x(:,i) = myfx(xdata(:,idx(i)),px(i));
    else
      x(:,i) = 1 + zeros(maxrows,1);
    end
  end

  xNum = res(2,2) + res(2,3) * x(:,1).*x(:,2);
  xDenom = 1 + res(2,5) * x(:,5).*x(:,6);
  if orient(2)==1
    xNum = xNum + res(2,4) * x(:,3).*x(:,4);
  else
    xDenom = xDenom + res(2,4) * x(:,3).*x(:,4);
  end
  ycalc = xNum ./ xDenom;
end
```

The local variables orient handle the orientation flags and determine if a variable is part of the numerator or denominator. To speed up calculations, I decided to skip over the power zero. The code for the virtual nested loop detects if an element of array idx is zero. If it is, the code assigns 1 to that array element.

The output of the above program is:

```
Program  clemTerm3

2025-Feb-18 17_48_23
Please wait ...

Excel file used is C:\Users\nsham\Dropbox\MATLAB\Clement6\data3.xlsx
Diary file used is data3_2025-Feb-18 17_48_23.txt
-------------- Calculating Max Iters ----------------
Max iters = 93,312
-------------- Maximizing Fx ----------------
Iter 2, Fx = 3.099874e-01, X=[2, 1, 2, 4, 4, 5, -1, -1, -1, -1, -1, -1, 1, 1, 2, ]
Iter 3, Fx = 3.549897e-01, X=[3, 1, 2, 4, 4, 5, -1, -1, -1, -1, -1, -1, 1, 1, 2, ]
Iter 29, Fx = 3.644370e-01, X=[2, 1, 2, 5, 4, 5, -1, -1, -1, -1, -1, -1, 1, 1, 2, ]
Iter 30, Fx = 4.062640e-01, X=[3, 1, 2, 5, 4, 5, -1, -1, -1, -1, -1, -1, 1, 1, 2, ]
Iter 48, Fx = 4.111966e-01, X=[3, 1, 4, 5, 4, 5, -1, -1, -1, -1, -1, -1, 1, 1, 2, ]
Iter 192, Fx = 4.134784e-01, X=[3, 1, 2, 5, 6, 5, -1, -1, -1, -1, -1, -1, 1, 1, 2, ]
Iter 210, Fx = 4.180776e-01, X=[3, 1, 4, 5, 6, 5, -1, -1, -1, -1, -1, -1, 1, 1, 2, ]
Iter 245, Fx = 4.276383e-01, X=[2, 1, 2, 4, 4, 6, -1, -1, -1, -1, -1, -1, 1, 1, 2, ]
Iter 246, Fx = 4.351780e-01, X=[3, 1, 2, 4, 4, 6, -1, -1, -1, -1, -1, -1, 1, 1, 2, ]
Iter 254, Fx = 4.409462e-01, X=[2, 1, 3, 4, 4, 6, -1, -1, -1, -1, -1, -1, 1, 1, 2, ]
Iter 255, Fx = 4.488345e-01, X=[3, 1, 3, 4, 4, 6, -1, -1, -1, -1, -1, -1, 1, 1, 2, ]
Iter 276, Fx = 4.516872e-01, X=[3, 2, 2, 5, 4, 6, -1, -1, -1, -1, -1, -1, 1, 1, 2, ]
Iter 488, Fx = 9.985644e-01, X=[2, 1, 2, 4, 4, 7, -1, -1, -1, -1, -1, -1, 1, 1, 2, ]
Iter 489, Fx = 9.985651e-01, X=[3, 1, 2, 4, 4, 7, -1, -1, -1, -1, -1, -1, 1, 1, 2, ]
Iter 492, Fx = 9.986992e-01, X=[3, 2, 2, 4, 4, 7, -1, -1, -1, -1, -1, -1, 1, 1, 2, ]
Iter 4139, Fx = 9.990010e-01, X=[2, 3, 2, 4, 4, 7, 1, -1, 1, -1, -1, -1, 1, 1, 2, ]
Progress  5.00%
Progress 10.00%
Progress 15.00%
Progress 20.00%
Progress 25.00%
Progress 30.00%
Progress 35.00%
Progress 39.99%
Progress 44.99%
Iter 44750, Fx = 1.000000e+00, X=[2, 1, 3, 5, 4, 6, 1, -1, 1, 1, 1, 1, 1, 1, 2, ]
Regression model
Adjusted R-square = 1.000000000000

mdl = 


Linear regression model:
    y ~ 1 + x1 + x2 + x3

Estimated Coefficients:
                       Estimate         SE    tStat    pValue
                   ________________    __    _____    ______

    (Intercept)    2.00000000000013    0      Inf       0
    x1             5.00000000000002    0      Inf       0
    x2             4.99999999999984    0      Inf       0
    x3             1.99999999999993    0      Inf       0


Number of observations: 100, Error degrees of freedom: 96
R-squared: 1,  Adjusted R-Squared: 1
F-statistic vs. constant model: 5.84e+26, p-value = 0
Y = 2.000000e+00 + (5.000000e+00)*X2 * 1/X1 + (5.000000e+00) * X3 * X5)/(1 +
(2.000000e+00) * X4 * X6)
Done!
```

The output shows that the program can identify the correct rational heteronomial. The sheet **Project** in file data3.xslx also shows very low percentage error values.

9/ The Excel Files for the Case of Four Terms

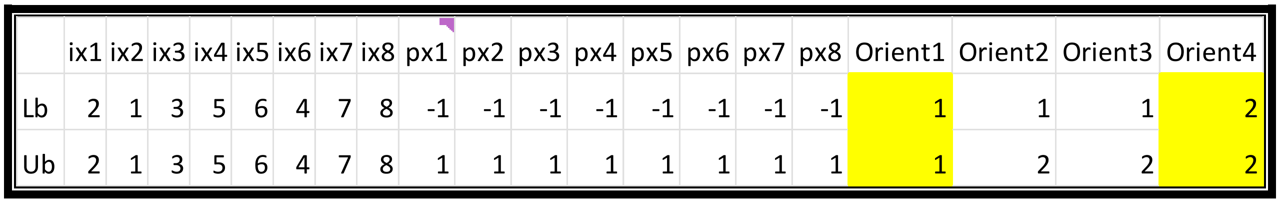
In this section, I focus on the Excel sheets **Transf** and **Results** for three (or more) independent variables and how they all differ from the same sheets for the case of two independent variables.

Sheet **Transf**

The sheet **Transf** contains the indices for selecting variables and their transformations and the following columns and rows (see Figure 9.1):

1. The first row contains groups of headers **that you must enter**. The first group of headers is ix1 through ix8. The second group of headers is px1, through px8. The third group is the orientation flags Orient1 through Orient4.
2. The second row has the tag **Lb** in column A to indicate that the cells to its right side contain lower bound values. These values are the lower limits of the indices for the candidate independent variables, the power values, and the orientation flag values. Notice that the orientation flag for the first term is AWLAYS 1. The orientation flags for the second and third terms are 1. The orientation flag for fourth term is ALWAYS 2.
3. The third row has the tag **Ub** in column A to indicate that the cells to its right side contain upper bound values. These values are the higher limits of the indices for the candidate independent variables, the power values, and the orientation flag values. Notice that the orientation flag for the first term is AWLAYS 1. The orientation flags for the second and third terms are 2. The orientation flag for fourth term is ALWAYS 2.

Figure 9.1 shows a sample sheet **Transf**. Based on the data in that sheet, the set of models examined have eight candidate variables with X_1 through x_8. To reduce computation time, the values under columns ix1 through ix8 ranges are narrow. The ranges for the ranges px1 through px8 are all (-1, 1). The ranges for the orientation flags are as described above.



	ix1	ix2	ix3	ix4	ix5	ix6	ix7	ix8	px1	px2	px3	px4	px5	px6	px7	px8	Orient1	Orient2	Orient3	Orient4
Lb	2	1	3	5	6	4	7	8	-1	-1	-1	-1	-1	-1	-1	-1	1	1	1	2
Ub	2	1	3	5	6	4	7	8	1	1	1	1	1	1	1	1	1	2	2	2

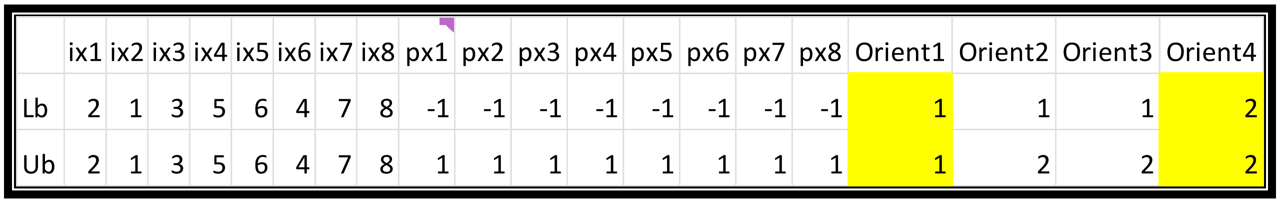
*Figure 9.1. Sample sheet **Transf** for 8 independent variables with three cross-product terms in file data4.xlsx.*

The Sheet **Results**

The sheet **Results** is one of the output sheets. Figure (7.2) shows a sample sheet for two cross-product terms. The sheet has the following rows:

1. Row has the headers that **you need to enter**!
2. Row 2 has the following groups of values:
   a. Cell A2 has the adjusted R-square values.
   b. Cells B2 to I2 contain the indices for the selected variables.

c. Cells J2 to Q2 contain the values for the best powers used with the selected variables.
d. Cells R2 to U2 contain orientation flags.



	ix1	ix2	ix3	ix4	ix5	ix6	ix7	ix8	px1	px2	px3	px4	px5	px6	px7	px8	Orient1	Orient2	Orient3	Orient4
Lb	2	1	3	5	6	4	7	8	-1	-1	-1	-1	-1	-1	-1	-1	1	1	1	2
Ub	2	1	3	5	6	4	7	8	1	1	1	1	1	1	1	1	1	2	2	2

*Figure 9.2. Sample sheet **Results** for 8 selected independent variables in file data8.xlsx.*

10/ The Code for the Case of Four Terms

Here is the code for file clemTerm4.m. It is very similar to the code of clemTerm3.m. The additional statements examine more indices, powers, and orientation flags,

```
% This program skips the transformation for log(x)

clc
close all
clear all

global mdl
global xdata
global ydata
global maxVarIdx

warning("off")
dt = datetime('now','TimeZone','local','Format','y-MMM-d HH_mm_ss');
dtstr = string(dt);
filename = strcat(pwd,'\data4.xlsx');
delete("data4_*.txt");
outFile = strcat("data4_",dtstr,".txt");
diary(outFile)

fprintf("Program  clemTerm4\n\n");
fprintf("%s\n", dtstr);
fprintf("Please wait ...\n\n");
fprintf("Excel file used is %s\n", filename);
fprintf("Diary file used is %s\n", outFile);
xyData = readmatrix(filename,'Sheet','Data');

ydata = xyData(:,1);
xdata = xyData(:,2:end);
[maxrows,maxcols] = size(xdata);
xyTransf = readmatrix(filename,'Sheet','Transf');
xyTransf = xyTransf(:,2:end);

% Important assignment - must match the number of X columns in sheet
% Transf of the Excel file
numX = 8;
Lb = xyTransf(1,:);
Ub = xyTransf(2,:);
% augment indices for variables to include the index of the dummy unity
% variable
Ub(1,1:numX) = Ub(1,1:numX)+1;
maxVarIdx = Ub(1,1:numX);
nVars = length(Lb);
iFrom = Lb;
iTo = Ub;
iStep = 1 + zeros(1,nVars);
idx = Lb;
bStop = false;
iter = 0;
fprintf("------------- Calculating Max Iters ----------------\n");
while ~bStop
  iter = iter +1;
  % Start implementing the quasi-nested loops
  idx(1) = idx(1) + iStep(1);
  % skip fx index 0 which is log(x) ... can be deleted
  if idx(1)==0, idx(1)=1; end
  if idx(1) > iTo(1)
    idx(1) = iFrom(1);
    for i = 2:nVars
      idx(i) = idx(i) + iStep(i);
      % skip fx index 0 which is log(x) ... can be deleted
      if idx(i)==0, idx(i)=1; end
      if idx(i) > iTo(i) && i < nVars
        idx(i) = iFrom(i);
      elseif idx(i) > iTo(i) && i == nVars
        bStop = true;
      else
        break
      end
    end
  end
end
str = num2str(iter);
str = fliplr(regexprep(fliplr(str), '(\d+\.)?(\d{3})(?=\S+)', '$1$2,'));
fprintf("Max iters = %s\n", str);
MaxIters = iter;
FivePercent = fix(iter/20);

iFrom = Lb;
iTo = Ub;
iStep = 1 + zeros(1,nVars);
idx = Lb;

bestX = zeros(1,length(Lb));
bestFx = 0;

% --------------------------------- start main loop
bStop = false;
iter = 0;
fprintf("------------- Maximizing Fx ----------------\n");
while ~bStop
```

```
  iter = iter +1;
  if mod(iter, FivePercent) == 0
    percent = 100*iter/MaxIters;
    fprintf("Progress %5.2f%%\n", percent);
  end

  r2adj = myFit(idx);
  if r2adj > bestFx
    bestFx = r2adj;
    bestX = idx;
    bestMdl = mdl;
    fprintf("Iter %d, Fx = %e, X=[", iter, bestFx)
    fprintf("%d, ", bestX);
    fprintf("]\n");
    if r2adj == 1, break; end
  end
  % Start implementing the quasi-nested loops
  idx(1) = idx(1) + iStep(1);
  % skip fx index 0 which is log(x) ... can be deleted
  if idx(1)==0, idx(1)=1; end
  if idx(1) > iTo(1)
    idx(1) = iFrom(1);
    for i = 2:nVars
      idx(i) = idx(i) + iStep(i);
      % skip fx index 0 which is log(x) ... can be deleted
      if idx(i)==0, idx(i)=1; end
      if idx(i) > iTo(i) && i < nVars
        idx(i) = iFrom(i);
      elseif idx(i) > iTo(i) && i == nVars
        bStop = true;
      else
        break
      end
    end
  end
end
fprintf("Regression model\n")
format long
AdjR2 = bestMdl.Rsquared.Adjusted;
fprintf("Adjusted R-square = %14.12f\n", AdjR2);
mdl = bestMdl
bestX = [AdjR2 bestX];

res = zeros(2,length(bestX));
res(1,:) = bestX;
res(2,2:6) = mdl.Coefficients{1:5,1}';
writematrix(res(1,:), filename, 'Sheet', 'Results', 'Range', 'A2:Z2');
writematrix(res(2,2:6), filename, 'Sheet', 'Results', 'Range', 'B3:Z3');

% now do the % projections
ycalc = project(res);
percErr = 100.*(ycalc-ydata)./ydata;
ProjMat = [ydata xdata ycalc percErr];
writematrix(ProjMat, filename, 'Sheet', 'Project', 'Range', strcat("A2:Z",
num2str(1+length(ydata))));

sNum = sprintf("Y = %e + (%e)*", res(2,2), res(2,3));
```

```
idx = res(1,2:numX+1);
px = res(1,numX+2:2*numX+1);
ops = res(1,2*numX+2:2*numX+4);

if px(1)==0
  sX1 = sprintf("log(X%d)", idx(1));
elseif px(1)==4
  sX1 = sprintf("sqrt(X%d)", idx(1));
elseif px(1)==-4
  sX1 = sprintf("1/log(X%d)", idx(1));
elseif px(1)==1
  sX1 = sprintf("X%d", idx(1));
elseif px(1)>1
  sX1 = sprintf("X%d^%d", idx(1), px(1));
elseif px(1)==-1
  sX1 = sprintf("1/X%d", idx(1));
elseif px(1)<-1
  sX1 = sprintf("1/X%d^%d", idx(1), abs(px(1)));
else
  sX1 = sprintf("X%d", idx(1));
end

if px(2)==0
  sX2 = sprintf("log(X%d)", idx(2));
elseif px(2)==4
  sX2 = sprintf("sqrt(X%d)", idx(2));
elseif px(2)==-4
  sX2 = sprintf("1/log(X%d)", idx(2));
elseif px(2)==1
  sX2 = sprintf("X%d", idx(2));
elseif px(2)>1
  sX2 = sprintf("X%d^%d", idx(2), px(2));
elseif px(2)==-1
  sX2 = sprintf("1/X%d", idx(2));
elseif px(2)<-1
  sX2 = sprintf("1/X%d^%d", idx(2), abs(px(2)));
else
  sX2 = sprintf("X%d", idx(2));
end

s = sprintf("%s * %s", sX1, sX2);
sNum = strcat(sNum, s);

if px(7)==0
  sX7 = sprintf("log(X%d)", idx(7));
elseif px(7)==4
  sX7 = sprintf("sqrt(X%d)", idx(7));
elseif px(7)==1
  sX7 = sprintf("X%d", idx(7));
elseif px(7)>1
  sX7 = sprintf("X%d^%d", idx(7), px(7));
elseif px(7)==-1
  sX7 = sprintf("1/X%d", idx(7));
elseif px(7)<-1
  sX7 = sprintf("1/X%d^%d", idx(7), abs(px(7)));
else
  sX7 = sprintf("X%d", idx(7));
```

```
end

if px(8)==0
  sX8 = sprintf("log(X%d)", idx(8));
elseif px(8)==4
  sX8 = sprintf("sqrt(X%d)", idx(8));
elseif px(8)==1
  sX8 = sprintf("X%d", idx(8));
elseif px(8)>1
  sX8 = sprintf("X%d^%d", idx(8), px(8));
elseif px(8)==-1
  sX8 = sprintf("1/X%d", idx(8));
elseif px(8)<-1
  sX8 = sprintf("1/X%d^%d", idx(8), abs(px(8)));
else
  sX8 = sprintf("X%d", idx(8));
end

sDenom = sprintf("1 + (%e) * %s * %s", res(2,6), sX7, sX8);

if px(3)==0
  sX3 = sprintf("log(X%d)", idx(3));
elseif px(3)==4
  sX3 = sprintf("sqrt(X%d)", idx(3));
elseif px(3)==-4
  sX3 = sprintf("1/log(X%d)", idx(3));
elseif px(3)==1
  sX3 = sprintf("X%d", idx(3));
elseif px(3)>1
  sX3 = sprintf("X%d^%d", idx(3), px(3));
elseif px(3)==-1
  sX3 = sprintf("1/X%d", idx(3));
elseif px(3)<-1
  sX3 = sprintf("1/X%d^%d", idx(3), abs(px(3)));
else
  sX3 = sprintf("X%d", idx(3));
end

if px(4)==0
  sX4 = sprintf("log(X%d)", idx(4));
elseif px(4)==4
  sX4 = sprintf("sqrt(X%d)", idx(4));
elseif px(4)==1
  sX4 = sprintf("X%d", idx(4));
elseif px(4)==-4
  sX4 = sprintf("1/log(X%d)", idx(4));
elseif px(4)>1
  sX4 = sprintf("X%d^%d", idx(4), px(4));
elseif px(4)==-1
  sX4 = sprintf("1/X%d", idx(4));
elseif px(4)<-1
  sX4 = sprintf("1/X%d^%d", idx(4), abs(px(4)));
else
  sX4 = sprintf("X%d", idx(4));
end

if ops(2)==1
```

```
  sNum = sprintf("%s + (%e) * %s * %s", sNum, res(2,4), sX3, sX4);
else
  sDenom = sprintf("%s + (%e) * %s * %s", sDenom, res(2,4), sX3, sX4);
end

if px(5)==0
  sX5 = sprintf("log(X%d)", idx(5));
elseif px(5)==4
  sX5 = sprintf("sqrt(X%d)", idx(5));
elseif px(5)==-4
  sX5 = sprintf("1/log(X%d)", idx(5));
elseif px(5)==1
  sX5 = sprintf("X%d", idx(5));
elseif px(5)>1
  sX5 = sprintf("X%d^%d", idx(5), px(5));
elseif px(5)==-1
  sX5 = sprintf("1/X%d", idx(5));
elseif px(5)<-1
  sX5 = sprintf("1/X%d^%d", idx(5), abs(px(5)));
else
  sX5 = sprintf("X%d", idx(5));
end

if px(6)==0
  sX6 = sprintf("log(X%d)", idx(6));
elseif px(6)==4
  sX6 = sprintf("sqrt(X%d)", idx(6));
elseif px(6)==-4
  sX6 = sprintf("1/log(X%d)", idx(6));
elseif px(6)==1
  sX6 = sprintf("X%d", idx(6));
elseif px(6)>1
  sX6 = sprintf("X%d^%d", idx(6), px(6));
elseif px(6)==-1
  sX6 = sprintf("1/X%d", idx(6));
elseif px(6)<-1
  sX6 = sprintf("1/X%d^%d", idx(6), abs(px(6)));
else
  sX6 = sprintf("X%d", idx(6));
end

if ops(3)==1
  sNum = sprintf("%s + (%e) * %s * %s", sNum, res(2,5), sX5, sX6);
else
  sDenom = sprintf("%s + (%e) * %s * %s", sDenom, res(2,5), sX5, sX6);
end

smodel = sprintf("%s)\n/(%s)", sNum, sDenom);
fprintf("%s\n", smodel)

format short
fprintf("Done!\n");
diary off
beep on
for i=1:3
  beep;
  pause(1);
```

```
end

function x = myfx(x,p)
  if p==0
    x = log(x);
  elseif p==4
    x = sqrt(x);
  elseif p==-4
    x = 1./log(x);
  elseif p > 1
    x = x.^p;
  elseif p < 0
    x = 1./x.^abs(p);
  end
end

function r = myFit(coeff)
  global mdl
  global xdata
  global ydata
  global maxVarIdx

  numX = length(maxVarIdx);
  [maxrows, ~] = size(xdata);
  y = ydata;
  idx = coeff(1:numX);
  px= coeff(numX+1:2*numX);
  orient = coeff(2*numX+1:2*numX+4);
  x = zeros(maxrows,numX);
  x2 = zeros(maxrows,fix(numX/2));

  % check for redundant vars in a term
  if idx(1)==idx(2) || idx(3)==idx(4) || idx(5)==idx(6) || idx(7)==idx(8)
    r = 0;
    return;
  end

  % check X1,X2 with X3,X4
  if idx(1)==idx(3) && px(1)==px(3) && ...
     idx(2)==idx(4) && px(2)==px(4) && orient(1) == orient(2)
    r = 0;
    return;
  end

  % check X1,X2 with X3,X4
  if idx(1)==idx(4) && px(1)==px(4) && ...
     idx(2)==idx(3) && px(2)==px(3) && orient(1) == orient(2)
    r = 0;
    return;
  end

  % check X1,X2 with X5,X6
  if idx(1)==idx(5) && px(1)==px(5) && ...
     idx(2)==idx(6) && px(2)==px(6) && orient(1) == orient(3)
    r = 0;
    r = 0;
    return;
  end

  % check X1,X2 with X5,X6
  if idx(1)==idx(6) && px(1)==px(6) && ...
     idx(2)==idx(5) && px(2)==px(5) && orient(1) == orient(3)
    r = 0;
    r = 0;
    return;
  end

  % check X1,X2 with X7,X8
  if idx(1)==idx(7) && px(1)==px(7) && ...
     idx(2)==idx(8) && px(2)==px(8) && orient(1) == orient(4)
    r = 0;
    return;
  end

  % check X1,X2 with X7,X8
  if idx(1)==idx(8) && px(1)==px(8) && ...
     idx(2)==idx(7) && px(2)==px(7) && orient(1) == orient(4)
    r = 0;
    r = 0;
    return;
  end

  % check X3,X4 with X5,X6
  if idx(3)==idx(5) && px(3)==px(5) && ...
     idx(4)==idx(6) && px(4)==px(6) && orient(2) == orient(3)
    r = 0;
    r = 0;
    return;
  end

  % check X3,X4 with X5,X6
  if idx(4)==idx(6) && px(4)==px(6) && ...
     idx(3)==idx(5) && px(3)==px(5) && orient(2) == orient(3)
    r = 0;
    return;
  end

  % check X3,X4 with X7,X8
  if idx(3)==idx(7) && px(3)==px(7) && ...
     idx(4)==idx(8) && px(4)==px(8) && orient(2) == orient(4)
    r = 0;
    return;
  end

  % check X3,X4 with X7,X8
  if idx(4)==idx(8) && px(4)==px(8) && ...
     idx(3)==idx(7) && px(3)==px(7) && orient(2) == orient(4)
    r = 0;
    return;
  end

  % check X5,X6 with X7,X8
  if idx(5)==idx(7) && px(5)==px(7) && ...
     idx(6)==idx(8) && px(6)==px(8) && orient(3) == orient(4)
    r = 0;
    return;
  end

  % check X5,X6 with X7,X8
  if idx(6)==idx(7) && px(6)==px(7) && ...
     idx(5)==idx(8) && px(5)==px(8) && orient(3) == orient(4)
    r = 0;
    return;
  end


  for i=1:numX
    if idx(i)<maxVarIdx(i)
      x(:,i) = myfx(xdata(:,idx(i)),px(i));
    else
      x(:,i) = 1+zeros(maxrows,1);
    end
  end

  i1=-1;
  i2=0;
  for i=1:fix(numX/2)
    i1=i1+2;
    i2=i2+2;
    if idx(i1)<maxVarIdx(i1) && idx(i2)<maxVarIdx(i2)
      x2(:,i) = x(:,i1).*x(:,i2);
    elseif idx(i2)<maxVarIdx(i2)
      x2(:,i) = x(:,i2);
    elseif idx(i1)<maxVarIdx(i1)
      x2(:,i) = x(:,i1);
    end
  end
  if orient(2)==2, x2(:,2) = -ydata.*x2(:,2); end
  if orient(3)==2, x2(:,3) = -ydata.*x2(:,3); end
  X = [x2(:,1) x2(:,2) x2(:,3) -ydata.*x2(:,4)];
  mdl = fitlm(X,y);
  r = mdl.Rsquared.Adjusted;
end

function ycalc = project(res)
  global xdata
  global maxVarIdx

  numX = length(maxVarIdx);
  [maxrows, ~] = size(xdata);

  idx = res(1,2:numX+1);
  px = res(1,numX+2:2*numX+1);
  x = zeros(maxrows,numX);
  orient = res(1,2*numX+2:2*numX+5);

  for i=1:numX
    if idx(i)<maxVarIdx(i)
      x(:,i) = myfx(xdata(:,idx(i)),px(i));
    else
      x(:,i) = 1 + zeros(maxrows,1);
    end
  end

  xNum = res(2,2) + res(2,3) * x(:,1).*x(:,2);
  xDenom = 1 + res(2,6) * x(:,7).*x(:,8);
  if orient(2)==1
    xNum = xNum + res(2,4) * x(:,3).*x(:,4);
  else
    xDenom = xDenom + res(2,4) * x(:,3).*x(:,4);
  end
  if orient(3)==1
    xNum = xNum + res(2,5) * x(:,5).*x(:,6);
  else
    xDenom = xDenom + res(2,5) * x(:,5).*x(:,6);
  end
  ycalc = xNum ./ xDenom;
end
```

To speed up calculations, I decided to skip over the power zero. The code for the virtual nested loop detects if an element of array idx is zero. If it is, the code assigns 1 to that array element.

Here is the output from the above program:

```
Program  clemTerm4

2025-Feb-18 17_50_13
Please wait ...

Excel file used is C:\Users\nsham\Dropbox\MATLAB\Clement6\data4.xlsx
Diary file used is data4_2025-Feb-18 17_50_13.txt
------------- Calculating Max Iters ----------------
Max iters = 262,144
------------- Maximizing Fx ----------------
Iter 1, Fx = 7.851827e-01, X=[2, 1, 3, 5, 6, 4, 7, 8, -1, -1, -1, -1, -1, -1,
-1, -1, 1, 1, 1, 2, ]
Iter 4, Fx = 7.873576e-01, X=[3, 2, 3, 5, 6, 4, 7, 8, -1, -1, -1, -1, -1, -1,
-1, -1, 1, 1, 1, 2, ]
Iter 5, Fx = 7.913150e-01, X=[2, 1, 4, 5, 6, 4, 7, 8, -1, -1, -1, -1, -1, -1,
-1, -1, 1, 1, 1, 2, ]
Iter 8, Fx = 7.933910e-01, X=[3, 2, 4, 5, 6, 4, 7, 8, -1, -1, -1, -1, -1, -1,
-1, -1, 1, 1, 1, 2, ]
Iter 40, Fx = 7.933933e-01, X=[3, 2, 4, 5, 6, 5, 7, 8, -1, -1, -1, -1, -1, -
1, -1, -1, 1, 1, 1, 2, ]
Iter 56, Fx = 7.952527e-01, X=[3, 2, 4, 5, 7, 5, 7, 8, -1, -1, -1, -1, -1, -
1, -1, -1, 1, 1, 1, 2, ]
Iter 133, Fx = 7.991406e-01, X=[2, 1, 4, 5, 6, 4, 7, 9, -1, -1, -1, -1, -1, -
1, -1, -1, 1, 1, 1, 2, ]
...
Iter 3618, Fx = 9.262600e-01, X=[3, 1, 3, 5, 6, 5, 7, 8, -1, 1, 1, 1, -1, -1,
-1, -1, 1, 1, 1, 2, ]
Iter 7169, Fx = 9.496565e-01, X=[2, 1, 3, 5, 6, 4, 7, 8, -1, -1, 1, 1, 1, -1,
-1, -1, 1, 1, 1, 2, ]
Iter 7170, Fx = 9.498254e-01, X=[3, 1, 3, 5, 6, 4, 7, 8, -1, -1, 1, 1, 1, -1,
-1, -1, 1, 1, 1, 2, ]
```

```
Iter 7172, Fx = 9.501704e-01, X=[3, 2, 3, 5, 6, 4, 7, 8, -1, -1, 1, 1, 1, -1,
-1, -1, 1, 1, 1, 2, ]
Progress  5.00%
Progress 10.00%
Progress 15.00%
Progress 20.00%
Progress 25.00%
Progress 30.00%
Progress 35.00%
Progress 40.00%
Progress 45.00%
Progress 50.00%
Iter 131089, Fx = 9.978346e-01, X=[2, 1, 3, 5, 7, 4, 7, 8, -1, -1, -1, -1, -
1, -1, -1, -1, 1, 1, 2, 2, ]
Iter 131092, Fx = 9.978353e-01, X=[3, 2, 3, 5, 7, 4, 7, 8, -1, -1, -1, -1, -
1, -1, -1, -1, 1, 1, 2, 2, ]
...
Iter 142353, Fx = 9.982739e-01, X=[2, 1, 3, 5, 7, 4, 7, 8, -1, -1, 1, 1, -1, 1, -1, -
1, 1, 1, 2, 2, ]
Iter 142356, Fx = 9.982751e-01, X=[3, 2, 3, 5, 7, 4, 7, 8, -1, -1, 1, 1, -1, 1, -1, -
1, 1, 1, 2, 2, ]
Iter 142865, Fx = 9.983760e-01, X=[2, 1, 3, 5, 7, 4, 7, 8, -1, 1, 1, 1, -1, 1, -1, -1,
1, 1, 2, 2, ]
Progress 55.00%
Progress 60.00%
Progress 65.00%
Progress 70.00%
Iter 195841, Fx = 1.000000e+00, X=[2, 1, 3, 5, 6, 4, 7, 8, 1, -1, 1, 1, 1, 1,
1, 1, 1, 1, 2, 2, ]
Regression model
Adjusted R-square = 1.000000000000

mdl =


Linear regression model:
    y ~ 1 + x1 + x2 + x3 + x4

Estimated Coefficients:
                        Estimate          SE    tStat    pValue
                   __________________    __    _____    ______

    (Intercept)      1.99999999999999    0      Inf       0
    x1              0.299999999999999    0      Inf       0
    x2             0.0999999999999994    0      Inf       0
    x3             0.0999999999999994    0      Inf       0
    x4              0.149999999999999    0      Inf       0


Number of observations: 100, Error degrees of freedom: 95
R-squared: 1,  Adjusted R-Squared: 1
F-statistic vs. constant model: 2.45e+28, p-value = 0
Y = 2.000000e+00 + (3.000000e-01)*X2 * 1/X1 + (1.000000e-01) * X3 * X5)
/(1 + (1.500000e-01) * X7 * X8 + (1.000000e-01) * X6 * X4)
Done!
```

The output shows that the program can identify the correct rational heteronomial. The sheet **Project** in file data4.xslx also shows very low percentage error values.

The ZIP file for this study also includes MATLAB files, Excel files, and text diary files for cases of four terms.

11/Conclusion

Using brute force search that employed virtual nested loops has advantages and disadvantages. The main advantage is that you are guaranteed to get the best results. The main disadvantage lies in the fact that using more candidate variables and wider ranges for selecting variables and powers may result in calculations that run for days!!! I have tried using mathematical optimization methods as an alternative approach. Unfortunately, the results were disappointing. The results seem to give good *alternative* models, but these models yield poor cross-validation data to check the values of the dependent variable. When doing cross-validation using equation (7), the results are poor. When doing cross-validation using equation (8) (which requires using the observed Y values in the right-hand side of that equation), the results are very good. However, using equation (8) for cross-validation defeats the purpose of calculating something you already know! I even implemented iterative methods to *estimate* values of Y for the right-hand side of equation (8), but the results were still poor.