

Partition Model Selection

Working with Heteronomials

by

Namir Clement Shammass

THE HERETIC EMPIRICIST

Contents

Introduction	1
Partition Regression	2
The Basics	3
The Case of One Independent Variable	3
Test Program for Function ebrm1()	10
Working with Errors in the Dependent Variable	14
Testing Model with No Correlation between the Variables	17
Working with Relatively Small Data Sets	20
Models for Heteronomials of Order Two and Up	26
The Zip File	26

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 simple linear heteronomial, often used in simple multiple linear regression is:

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

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

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

In equation (2) the variable X_1 is raised to the power -1 and the variable X_2 is raised to the power 2. More advanced heteronomials involve more independent variables and different powers like -3, -2, 3, 0.5, 1.5, and so on. The power zero is translated to the $\ln(x)$ function. The other powers are taken at face value. 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:

$$Y^{py} = a + b_1 X_1^{px1} + b_2 X_2^{px2} + \dots + b_n X_n^{pxn} \quad (3)$$

Notice that each variable in equation (3) **can have** a non-unity power!

Partition Regression

Partitioning the data in a set of training data and one or more sets of testing data allows us to test the models derived in the training phase with data the models have not yet encountered. This approach allows us to examine the robustness of the models.

I will test using the partition approach to perform curve fitting with heteronomials, like the ones in equation (1). The general approach uses the following steps:

1. Divide the data into bins (or pages). The first bin should be bigger than the other bins. I will choose the first bin to be twice as big as the remaining data bins. The first bin is the *training bin*. The other bins are the *testing bins*.
2. Determine the set of heteronomial models using the data in the training bin.
3. Display the results for the best model that employs the training data.
4. Use the set of models generated in step 2 to calculate the corresponding sets of predicted values of the dependent variable in the testing bins. By comparing the original values and the set of predicted ones in each test bin, we calculate the adjusted coefficients of determination (adjusted R-square).
5. Determine for each bin, (that yields the highest values of the adjusted R-square) of the set of the models obtained in step 2.
6. Display the results (best powers and adjusted R-square) for each test bin.
7. Display detailed results for the best fit in the test bins.
8. Display the results for the model using ALL the data.

There are basically the following results to compare:

1. The best model (using the training bin data) that fits the test bins.

2. The best model that uses the training bin.
3. The best model that uses all the data.

Option 1 has the most weight because it applies the best model from the training data to test data it has not seen! Option 3 is the next most important weight, because it uses all the data.

The Basics

The regression models are built using various powers for each variable. For example, in the case of one independent variable, we have:

$$Y^{py} = a + b X^{px} \quad (4)$$

In the case of two and three independent variables, we have:

$$Y^{py} = a + b_1 X_1^{px1} + b_2 X_2^{px2} \quad (5)$$

$$Y^{py} = a + b_1 X_1^{px1} + b_2 X_2^{px2} + b_3 X_3^{px3} \quad (6)$$

And so on. Since we don't know the powers to best fit predicted values of the dependent variable, we need to search each power within a range. The values for all the powers used fall in user-defined ranges. The search uses lower and upper range limits and search increments specified by the user. These values can be negative, zero, and positive. In the case of zero, the code uses function $\ln(x)$. The models tested comprise of the combination of various powers for each variable. For example, using equation (1), we can select the powers 0, 1, and 2 for each variable to study 9 regression models. Applying the same power ranges to equation (2), yields 27 models. Applying the same power ranges to equation (3), yields 81 models. Each regression variable can have its own unique range of powers. Note that as the number of enumerated powers increases, the total number of models jumps very quickly.

The Case of One Independent Variable

Let's start with the simple case of the regression model in equation (4). Here is the MATLAB code for the calculations that determine the models in the main bin and then test them with the other bins.

```
function [bestR2,bestMIdx,bestPwrs,bestCoeffs,mdlBest,mdlAll] ...
    = ebrml(xdata,ydata,xlow,xstep,xhi)
    warning("off")
```

```

fprintf("Please wait ....\n");
[S,xdata,ydata]= Init(xdata,ydata,xlow,xstep,xhi);
i1 = S.Blow(1);
i2 = S.Bhi(1);
x = xdata(i1:i2,:);
y = ydata(i1:i2);
bestR2a = 0;
for i=1:S.M
    pArr = S.pwrMat(i,1:S.V+1);
    py = pArr(1);
    px1 = pArr(2);
    eval(S.sMdlX);
    eval(S.sMdlY);
    mdl = fitlm(X,yt);
    r2a = mdl.Rsquared.Adjusted;
    if isnan(r2a) || isinf(r2a) || r2a < 0 || r2a > 1
        r2a = 0;
    end
    if bestR2a < r2a
        bestR2a = r2a;
        bestMdl = mdl;
        bestpArr = pArr;
    end
    S.stats(i,1:S.V+2) = ...
        [mdl.Coefficients{1,1}, ...
        mdl.Coefficients{2,1}, ...
        r2a];
end
fprintf("for Bin 1\nBest powers: ")
fprintf("%f, ", bestpArr);
fprintf("\nBest Model:\n")
bestMdl
anova(bestMdl,"summary")

fprintf("Finished calculations with Bin 1\n\n")
cr = size(S.V+1,1);
for iBin=2:S.B
    x = xdata(S.Blow(iBin):S.Bhi(iBin),:);
    y = ydata(S.Blow(iBin):S.Bhi(iBin));
    for i=1:S.M
        pArr = S.pwrMat(i,1:S.V+1);
        py = pArr(1);
        px1 = pArr(2);
        for k=1:S.V+1
            cr(k) = S.stats(i,k);
        end
        eval(S.sEvalX);
        eval(S.sEvalY);
        % yhat=yhat';
        e = y-yhat;
        SSres = sum(dot(e,e));
        ymean = mean(y);
        e = y-ymean;
        SStot = sum(dot(e,e));
        r2 = max(0,1 - SSres/SStot);
        if r2 > 0
            r2adj = 1 - (1-r2)*(S.N-1)/(S.N - S.V - 1);

```

```

        else
            r2adj = 0;
        end
        S.Bin(iBin,i) = r2adj;
        S.Bin(1,i) = S.Bin(1,i) + S.Bin(iBin,i);
    end
    [bestR2,bestMIdx] = max(S.Bin(iBin,:));
    bestPwrs = S.pwrMat(bestMIdx,:);
    fprintf("Bin # %d\n", iBin)
    fprintf("Best R2adj = %f\n", bestR2)
    fprintf("Best model # %d\n", bestMIdx)
    fprintf("Powers: ")
    fprintf(" %f, ", bestPwrs)
    fprintf("\n")

end
fprintf("\n\n")
S.Bin(1,:) = S.Bin(1,:)/(S.B-1);
[bestR2,bestMIdx] = max(S.Bin(1,:));
bestPwrs = S.pwrMat(bestMIdx,:);
bestCoeffs = S.stats(bestMIdx,1:S.V+1);
x = xdata(S.Blow(1):S.Bhi(1),:);
y = ydata(S.Blow(1):S.Bhi(1));
py = bestPwrs(1);
px1 = bestPwrs(2);
eval(S.sMdlX);
eval(S.sMdlY);
mdlBest = fitlm(X,yt);
fprintf("Using data in ALL bins ----- \n")
x = xdata;
y = ydata;
eval(S.sMdlX);
eval(S.sMdlY);
mdlAll = fitlm(X,yt);
end

function y = fxt(x,pwr)
    if pwr > 0
        y = x.^pwr;
    elseif pwr < 0
        y = 1./x.^abs(pwr);
    else
        y = log(x);
    end
end

function y = fxinv(x,pwr)
    if pwr > 0
        y = x.^(1/pwr);
    elseif pwr < 0
        y = x.^abs(pwr);
    else
        y = exp(x);
    end
end

```

```

function [S,x,y]= Init(x,y,xlow,xstep,xhi)
    [rows,cols] = size(x);
    S.N = rows;
    S.V = cols;
    S.B = 4;
    m = S.B+1;
    delta = fix(S.N/m);
    S.Blow(1) = 1;
    S.Bhi(1) = 2*delta;
    S.Blow(2) = 2*delta+1;
    S.Bhi(2) = 3*delta;
    S.Blow(3) = 3*delta+1;
    S.Bhi(3) = 4*delta;
    S.Blow(4) = 4*delta+1;
    S.Bhi(4) = S.N;
    % shuffle array
    idx = randperm(S.N);
    x = x(idx,:);
    y = y(idx);

    S.smdlX = "X = [fxt(x(:,1),px1)];";
    S.smdlY = "yt = fxt(y,py);";
    S.sEvalX = "yhat=cr(1)+cr(2)*fxt(x(:,1),px1);";
    S.sEvalY = "yhat = fxinv(yhat,1/py);";
    % count the total number of models
    count=0;
    for py=xlow(S.V+1):xstep(S.V+1):xhi(S.V+1)
        for px1=xlow(1):xstep(1):xhi(1)
            count=count+1;
        end
    end
    S.M = count;

    S.pwrMat = zeros(S.M,S.V+1);
    S.stats= zeros(S.M,S.V+2);
    i=0;
    for py=xlow(S.V+1):xstep(S.V+1):xhi(S.V+1)
        for px1=xlow(1):xstep(1):xhi(1)
            i=i+1;
            S.pwrMat(i,1:S.V+1)=[py,px1];
            % next struct maps rc(1) to rc(4) and RsqrAdj
            S.stats(i,1:S.V+2) = [0,0,0];
        end
    end

    % Note use S.Bin(1,) to store the average values
    % of S.Bin(2,) and up.
    S.Bin = zeros(S.B,S.M);

end

```

The function `embr1()` performs the calculations for the model in equation (4). The function has the following input parameters:

- The parameter `xdata` is a single column vector that contains all the data for the independent variable. In the case of multiple independent variables,

xdata is a matrix where each column stores values for a specific independent variable.

- The parameter ydata is a single column vector that contains all the data for the dependent variable.
- The parameter xlow is a single row vector that contains all the initial powers. The first value belongs to the dependent variable. The second value belongs to the independent variable. In the case of multiple independent variables, the second value belongs to the first independent variable, the third value belongs to the second independent variable, and so on.
- The parameter xstep is a single row vector that contains all the power increment values. The first value belongs to the dependent variable. The second value belongs to the independent variable. In the case of multiple independent variables, the second value belongs to the first independent variable, the third value belongs to the second independent variable, and so on.
- The parameter xhi is a single row vector that contains all the final powers. The first value belongs to the dependent variable. The second value belongs to the independent variable. In the case of multiple independent variables, the second value belongs to the first independent variable, the third value belongs to the second independent variable, and so on.

The function has the following output parameters:

- The parameter bestR2 contains the adjusted R-square for the best model.
- The parameter bestMIIdx contains the index of the best model.
- The parameter bestPwrs is an array that contains the powers of the best model. The first value belongs to the dependent variable. The second value belongs to the independent variable. In the case of multiple independent variables, the second value belongs to the first independent variable, the third value belongs to the second independent variable, and so on.
- The parameter bestCoeffs is the array of regression coefficients for the best model. The first value is the intercept. The second value is the slope. In the case of multiple independent variables, the second value is the slope of the first independent variable, the third value is the slope of the second independent variable, and so on.
- The parameter mdlBest contains the object for the best model.
- The parameter mdlAll contains the object for the model that processes all the regression data.

The function `embr1()` calls the local function `Init()` to initialize the data by transforming the regression variables and initializing the index ranges that define the main bin and the test bins. The parameters for function `Init()` are:

- The parameter `x` is a single column vector that contains all the data for the independent variable. In the case of multiple independent variables, `x` is a matrix where each column has data for a specific independent variable.
- The parameter `y` is a single column vector that contains all of the data for the dependent variable.
- The parameter `xlow` is a single row vector that contains all the initial powers. The first value belongs to the dependent variable. The second value belongs to the independent variable. In the case of multiple independent variables, the second value belongs to the first independent variable, the third value belongs to the second independent variable, and so on.
- The parameter `xstep` is a single row vector that contains all the power increment values. The first value belongs to the dependent variable. The second value belongs to the independent variable. In the case of multiple independent variables, the second value belongs to the first independent variable, the third value belongs to the second independent variable, and so on.
- The parameter `xhi` is a single row vector that contains all the final powers. The first value belongs to the dependent variable. The second value belongs to the independent variable. In the case of multiple independent variables, the second value belongs to the first independent variable, the third value belongs to the second independent variable, and so on.

The function `Init()` has the following output parameters:

- The parameter `x` is a single column vector that contains all the data for the independent variable. In the case of multiple independent variables, `x` is a matrix where each column has data for a specific independent variable.
- The parameter `y` is a single column vector that contains all the data for the dependent variable.
- The structure `S` that contains the following information:
 - The fields `N`, `V`, and `B` store the number of data points, number of independent variables, and number of bins, respectively.
 - Arrays `Blow` and `Bhi` store the ranges for the indices that define the various bins of data.

- The strings sMdlX and sMdlY store the transformations of x and y data.
- The strings sEvalX and sEvalY are used to calculate projected y values and their inverse transformation, respectively.
- The matrix pwrMat stores the powers used for each regression model.
- The matrix stats stores the regression coefficient for each regression model.
- The matrix Bin stores the array of adjusted R-square values.

There are also the helper functions fxt() and fxinv(). The first transforms data based on the supplied power value. The second function performs the inverse transformation, based on the power value.

After calling the function Init(), the function embr1() performs the following tasks:

- Initialize the local arrays x and y to store the data for the main bin, from parameters xdata and ydata, respectively.
- Set the best adjusted R-square value to 0.
- Use a loop to iterate over each regression model:
 - Obtain the powers used for transforming the data from matrix S.pwrMat and store the powers in variables py and px1. In the case of multiple independent variables, the powers are stored in variables px1, px2, and so on.
 - Use the MATLAB function eval() to evaluate strings S.sMdlX and S.sMdlY. This step transforms the x and y data using the powers stored in variables py and px1. In the case of multiple independent variables, the powers are stored in variables px1, px2, and so on. This task ends up storing the independent variable(s) in matrix X and the dependent variable in array yt.
 - Invoke the MATLAB function fitlm() to calculate the regression model and store it in variable mdl. The arguments for calling function fitlm() are variables X and yt.
 - Store the adjusted R-square value of the model in variable r2a.
 - Validate the value in variable r2a and set it to 0 if it is Nan, or Inf, or outside the interval (0, 1).
 - Test if r2a is greater than bestR2a. If it is, then update bestR2a, the best model (stored in variable bestMdl), and the best array of powers (stored in variable bestpArr).

- Store the regression coefficient and the adjusted R-square of the current model in matrix S.stats. The first column stores the intercept values. The second column stores the slope values. The third column stores the adjusted R-square values. In the case of multiple independent variables, the slopes for variables appear in the second column to the column before last. The last column stores the adjusted R-square values.
- Display the regression results and ANOVA table for the main bin of data.
- Use a loop to iterate over the test bins:
 - Store the xdata and ydata for the current bin in variables x and y, respectively.
 - Use a loop for each regression model:
 - Obtain the powers for the current model from matrix S.pwrMat and store them in variables py and px1.
 - Copy the regression coefficients of the current model from matrix S.stats into array cr.
 - Use the MATLAB function eval() with strings S.sEvalX and S.sEvalY to calculate the projected values of y.
 - Calculate (and validate) the value of the adjusted R-square based on the projected values of y and those in the current bin.
 - Store adjusted R-square value in matrix S.Bin.
 - Accumulate the values of of the adjusted R-square in row S.Bin(1,i).
 - Obtain the best model in the current bin and store that information in variables bestR2 and bestMIdx. Also store the powers for the best model in variable bestPwrs.
 - Calculate the regression model for the data in the main bin.

Test Program for Function ebrm1()

This section tests a simple model of $Y = 5 + 2 * X$. The following listings contains the code for the test program:

```
clc
clear all

n=1250;
xdata=zeros(n,1);
ydata=zeros(n,1);
for i=1:n
```

```

z = 1;
for j=1:1
    z = z + 10;
    xdata(i,j) = 1+rand*z;
end
ydata(i) = 5 + 2 * xdata(i,1);
end

[bestR2,bestMIdx,bestPwrs,bestCoeffs,mdlBest,mdlAll] = ...
    ebrml(xdata,ydata,zeros(1,2),0.5+zeros(1,2),...
    4+zeros(1,2))
fprintf("Regression ANOVA table for best model\n")
anova(mdlBest,"summary")
fprintf("Regression ANOVA table for ALL data\n")
anova(mdlAll,"summary")
% save variables for future use
save("test11.mat", "bestR2","bestPwrs", ...
    "bestCoeffs","mdlBest","mdlAll")

```

The test program performs the following tasks:

- Sets the number of data points to 1250. This value ends up assigning 500 points to the main bin, and 250 points to each test bin. These numbers are based on the fact that function ebrml() uses three test bins.
- Create the arrays xdata and ydata and assign random values to xdata. Calculate the values for ydata using $ydata(i) = 5 + 2 * xdata(i,1)$.
- Call function ebrml() and supply the following arguments:
 - The array xdata and ydata supply the regression data.
 - The array zeros(1, 2) supplies zeroes as the lower limits for the powers of both Y and X.
 - The array 0.5+zeros(1, 2) supplies 0.5 as the increments for the powers of both Y and X.
 - The array 4+zeros(1, 2) supplies 4 as the upper limits for the powers of both Y and X.
- Display the output parameters bestR2, bestMIdx, bestPwrs, bestCoeffs, mdlBest, and mdlAll.
- Display the regression ANOVA table for the best model.
- Display the regression ANOVA table for the model mdlAll.
- Save the output parameters to a .mat file for future use.

I will present the output of the test program piecewise. Ther first set of output lines are:

```

Please wait ....
for Bin 1
Best powers: 1.000000, 1.000000,
Best Model:

```

```
bestMdl =

Linear regression model:
  y ~ 1 + x1

Estimated Coefficients:

```

	Estimate	SE	tStat	pValue
(Intercept)	5	2.4455e-08	2.0446e+08	0
x1	2	3.4681e-09	5.7669e+08	0

```

Number of observations: 500, Error degrees of freedom: 498
Root Mean Squared Error: 2.42e-07
R-squared: 1, Adjusted R-Squared: 1
F-statistic vs. constant model: Inf, p-value = 0

ans =

3x5 table

```

	SumSq	DF	MeanSq	F	pValue
Total	19436	499	38.949		
Model	19436	1	19436	Inf	0
Residual	0	498	0		

```

Finished calculations with Bin 1

```

The above output indicates that the best powers for Y and X are 1 and 1. This is the linear relationship coded in the test program. The output shows the correct values of 5 for the intercept, and 2 for the slope.

The second part of the output is:

```

Bin # 2
Best R2adj = 1.000000
Best model # 21
Powers: 1.000000, 1.000000,
Bin # 3
Best R2adj = 1.000000
Best model # 21
Powers: 1.000000, 1.000000,
Bin # 4
Best R2adj = 1.000000
Best model # 21
Powers: 1.000000, 1.000000,

```

The output for bins 2, 3, and 4 shows that the best powers belong to the linear relationship between variables X and Y. These calculations are based on the best model obtained in the main bin— $Y=5+2*X$.

The third output part shows the results for the best model:

```
Using data in ALL bins -----
bestR2 =
      1

bestMIdx =
      21

bestPwrs =
      1      1

bestCoeffs =
      5.0000      2.0000

mdlBest =

Linear regression model:
  y ~ 1 + x1

Estimated Coefficients:
              Estimate          SE          tStat          pValue
-----
(Intercept)      5      3.4555e-08      1.4469e+08      0
x1                2      4.8036e-09      4.1635e+08      0

Number of observations: 500, Error degrees of freedom: 498
Root Mean Squared Error: 3.42e-07
R-squared: 1, Adjusted R-Squared: 1
F-statistic vs. constant model: 1.55e+33, p-value = 0
```

And finally, the output for the regression using all of the data is:

```
mdlAll =

Linear regression model:
  y ~ 1 + x1
```

Estimated Coefficients:

	Estimate	SE	tStat	pValue
(Intercept)	5	0	Inf	0
x1	2	0	Inf	0

Number of observations: 1250, Error degrees of freedom: 1248

R-squared: 1, Adjusted R-Squared: 1

F-statistic vs. constant model: 2.53e+32, p-value = 0

Since the values of Y were error-free the various output segments agree that $Y = 5 + 2 \cdot X$ is the best model. If we inject small errors in the values of Y, the best model would most likely still be the linear model. The values for the intercept and slope would be close to 5 and 2, respectively.

Working with Errors in the Dependent Variable

The previous test code calculated values of Y using $Y = 5 + 2 \cdot X$. The regression calculations were able to identify the model and calculate the exact values for the regression coefficients. In this section, I present a test program that injects +/- 5% error in the Y values.

The code for test program test1Err.m is:

```
clc
clear all
diary test1Err.txt
n=1250;
xdata=zeros(n,1);
ydata=zeros(n,1);
for i=1:n
    z = 1;
    for j=1:1
        z = z + 10;
        xdata(i,j) = 1+rand*z;
    end
    ydata(i) = 5 + 2 * xdata(i,1);
    ydata(i) = ydata(i)*(1+0.1*(rand-0.5));
end

[bestR2,bestMIdx,bestPwrs,bestCoeffs,mdlBest,mdlAll] = ...
    ebrml(xdata,ydata,zeros(1,2),0.5+zeros(1,2),...
    4+zeros(1,2))
save("test11.mat", "bestR2","bestPwrs", ...
    "bestCoeffs","mdlBest","mdlAll")
diary off
```

The output (which varies between different runs) is:

```

Please wait ....
for Bin 1
Best powers: 1.000000, 1.000000,
Best Model:

bestMdl =

Linear regression model:
  y ~ 1 + x1

Estimated Coefficients:
              Estimate          SE          tStat          pValue
              

---

          

---

          

---

          

---


(Intercept)    5.0449      0.058436      86.332    9.2973e-302
x1              1.9973      0.0080331     248.64              0

Number of observations: 500, Error degrees of freedom: 498
Root Mean Squared Error: 0.56
R-squared: 0.992, Adjusted R-Squared: 0.992
F-statistic vs. constant model: 6.18e+04, p-value = 0

ans =

3x5 table
              SumSq      DF      MeanSq      F      pValue
              

---

      

---

      

---

      

---

      

---


Total         19546      499        39.17
Model         19390        1       19390     61821          0
Residual      156.19      498        0.31364

Finished calculations with Bin 1

Bin # 2
Best R2adj = 0.993258
Best model # 21
Powers: 1.000000, 1.000000,
Bin # 3
Best R2adj = 0.991390
Best model # 21
Powers: 1.000000, 1.000000,
Bin # 4
Best R2adj = 0.993555
Best model # 21
Powers: 1.000000, 1.000000,

Using data in ALL bins -----

bestR2 =

0.9927

```

```
bestMIdx =
    21

bestPwrs =
    1    1

bestCoeffs =
    5.0449    1.9973

mdlBest =

Linear regression model:
y ~ 1 + x1

Estimated Coefficients:
              Estimate          SE          tStat          pValue
-----
(Intercept)    5.0449    0.058436    86.332    9.2973e-302
x1             1.9973    0.0080331   248.64         0

Number of observations: 500, Error degrees of freedom: 498
Root Mean Squared Error: 0.56
R-squared: 0.992, Adjusted R-Squared: 0.992
F-statistic vs. constant model: 6.18e+04, p-value = 0

mdlAll =

Linear regression model:
y ~ 1 + x1

Estimated Coefficients:
              Estimate          SE          tStat          pValue
-----
(Intercept)    5.0201    0.034975   143.54         0
x1             1.9973    0.0049098   406.81         0

Number of observations: 1250, Error degrees of freedom: 1248
Root Mean Squared Error: 0.546
R-squared: 0.993, Adjusted R-Squared: 0.993
F-statistic vs. constant model: 1.65e+05, p-value = 0
```

The main bin and the test bins favor the linear model. The regression coefficients are close to those of the error-free model.

If you run the above code several times you may see that the main bin favors a nonlinear model, and the test bins favor the linear model as obtained by the main bin. The Partition bins favor the linear model.

When the test bins agree about the best model, that model is vindicated as being the best. By contrast, when the test bins fail to be on the same proverbial page, there is no true best model. This is often confirmed by the low values of the adjusted R-square statistic. In this case, you may want to expand or shift the ranges of powers for some or all the regression variables.

Testing Model with No Correlation between the Variables

This section looks at the extreme case, where the values of the variables X and Y are random and uncorrelated—there is no model that describes a relationship between X and Y.

The MATLAB code for testing this case appears below. Note the assignment of random values to the elements of array ydata.

```
clc
clear all
diary test1Err2.txt
n=1250;
xdata=zeros(n,1);
ydata=zeros(n,1);
for i=1:n
    z = 1;
    for j=1:1
        z = z + 10;
        xdata(i,j) = 1+rand*z;
    end
    ydata(i) = 50*rand;
end

[bestR2,bestMIdx,bestPwrs,bestCoeffs,mdlBest,mdlAll] = ...
    ebrml(xdata,ydata,zeros(1,2),0.5+zeros(1,2),...
    4+zeros(1,2))
save("test13.mat", "bestR2","bestPwrs", ...
    "bestCoeffs","mdlBest","mdlAll")
diary off
```

A sample output from the above test program is:

```
Please wait ....
for Bin 1
```

```

Best powers: 4.000000, 0.500000,
Best Model:

bestMdl =

Linear regression model:
  y ~ 1 + x1

Estimated Coefficients:
              Estimate              SE              tStat              pValue
-----
(Intercept)  9.1406e+05  2.8679e+05  3.1873  0.0015266
x1           1.3098e+05  1.1141e+05  1.1756  0.24031

Number of observations: 500, Error degrees of freedom: 498
Root Mean Squared Error: 1.68e+06
R-squared: 0.00277, Adjusted R-Squared: 0.000765
F-statistic vs. constant model: 1.38, p-value = 0.24

ans =

3x5 table
              SumSq              DF              MeanSq              F              pValue
-----
Total       1.4102e+15  499  2.8261e+12
Model       3.9029e+12   1  3.9029e+12  1.3821  0.24031
Residual    1.4063e+15  498  2.8239e+12

Finished calculations with Bin 1

Bin # 2
Best R2adj = 0.000000
Best model # 1
Powers: 0.000000, 0.000000,
Bin # 3
Best R2adj = 0.000000
Best model # 1
Powers: 0.000000, 0.000000,
Bin # 4
Best R2adj = 0.000000
Best model # 1
Powers: 0.000000, 0.000000,

Using data in ALL bins -----

bestR2 =

0

bestMIdx =

```

```

1

bestPwrs =

    0    0

bestCoeffs =

    2.9141    0.0107

mdlBest =

Linear regression model:
    y ~ 1 + x1

Estimated Coefficients:
              Estimate      SE      tStat      pValue
-----
(Intercept)    2.9141    0.13051    22.328  4.8939e-77
x1             0.010727   0.070927    0.15124   0.87985

Number of observations: 500, Error degrees of freedom: 498
Root Mean Squared Error: 0.984
R-squared: 4.59e-05, Adjusted R-Squared: -0.00196
F-statistic vs. constant model: 0.0229, p-value = 0.88

mdlAll =

Linear regression model:
    y ~ 1 + x1

Estimated Coefficients:
              Estimate      SE      tStat      pValue
-----
(Intercept)    2.987    0.07771    38.438  6.3822e-214
x1            -0.021711  0.042768   -0.50765   0.61179

Number of observations: 1250, Error degrees of freedom: 1248
Root Mean Squared Error: 0.938
R-squared: 0.000206, Adjusted R-Squared: -0.000595
F-statistic vs. constant model: 0.258, p-value = 0.612

```

The main bin favors a nonlinear model, while the test bins favor the power (log-log) models. The adjusted R-square values are very low (and some are even

negative) to indicate that there isn't any correlation between the regression variables.

Working with Relatively Small Data Sets

The test code in the previous section worked with 1250 data points. What if you have, say, 100 data points or even less? Using multiple bins with non-overlapping data may not work well since each bin will have relatively a small number of data points. In this section we look at another approach. Here, each bin has 80% of the shuffled data. So, when we move from one bin to another, the models obtained from the first bin will encounter many data points they have already processed in the first bin.

The scheme that I described above can also be used with large number of data points. This option is all yours!

Here is the code for the MATLAB function `ebrm1Small()`:

```
function [bestR2,bestMIdx,bestPwrs,bestCoeffs,mdlBest,mdlAll] ...
    = ebrm1Small(xdata,ydata,xlow,xstep,xhi)
% handles relatively small data sets
warning("off")
fprintf("Please wait ....\n");
[S,xdata,ydata]= Init(xdata,ydata,xlow,xstep,xhi);
% shuffle array
idx = randperm(S.N);
x = xdata(idx,:);
y = ydata(idx);
dSize = fix(0.8*S.N);
x = x(1:dSize,:);
y = y(1:dSize);
bestR2a = 0;
for i=1:S.M
    pArr = S.pwrMat(i,1:S.V+1);
    py = pArr(1);
    px1 = pArr(2);
    eval(S.sMdlX);
    eval(S.sMdlY);
    mdl = fitlm(X,yt);
    r2a = mdl.Rsquared.Adjusted;
    if isnan(r2a) || isinf(r2a) || r2a < 0 || r2a > 1
        r2a = 0;
    end
    if bestR2a < r2a
        bestR2a = r2a;
        bestMdl = mdl;
        bestpArr = pArr;
    end
end
```

```

        S.stats(i,1:S.V+2) = ...
            [mdl.Coefficients{1,1}, ...
            mdl.Coefficients{2,1}, ...
            r2a];
    end
    fprintf("for Bin 1\nBest powers: ")
    fprintf("%f, ", bestpArr);
    fprintf("\nBest Model:\n")
    bestMdl
    anova(bestMdl,"summary")

    fprintf("Finished calculations with Bin 1\n\n")
    cr = size(S.V+1,1);
    for iBin=2:S.B
        % shuffle array
        idx = randperm(S.N);
        x = xdata(idx,:);
        y = ydata(idx);
        x = x(1:dSize,:);
        y = y(1:dSize);
        for i=1:S.M
            pArr = S.pwrMat(i,1:S.V+1);
            py = pArr(1);
            px1 = pArr(2);
            for k=1:S.V+1
                cr(k) = S.stats(i,k);
            end
            eval(S.sEvalX);
            eval(S.sEvalY);
            % yhat=yhat';
            e = y-yhat;
            SSres = sum(dot(e,e));
            ymean = mean(y);
            e = y-ymean;
            SStot = sum(dot(e,e));
            r2 = max(0,1 - SSres/SStot);
            if r2 > 0
                r2adj = 1 - (1-r2)*(S.N-1)/(S.N - S.V - 1);
            else
                r2adj = 0;
            end
            S.Bin(iBin,i) = r2adj;
            S.Bin(1,i) = S.Bin(1,i) + S.Bin(iBin,i);
        end
        [bestR2,bestMIdx] = max(S.Bin(iBin,:));
        bestPwrs = S.pwrMat(bestMIdx,:);
        fprintf("Bin # %d\n", iBin)
        fprintf("Best R2adj = %f\n", bestR2)
        fprintf("Best model # %d\n", bestMIdx)
        fprintf("Powers: ")
        fprintf(" %f, ", bestPwrs)
        fprintf("\n")
    end
    fprintf("\n\n")
    S.Bin(1,:) = S.Bin(1,:)/(S.B-1);
    [bestR2,bestMIdx] = max(S.Bin(1,:));

```

```

bestPwrs = S.pwrMat(bestMIdx,:);
bestCoeffs = S.stats(bestMIdx,1:S.V+1);
x = xdata(1:dSize,:);
y = ydata(1:dSize);
py = bestPwrs(1);
px1 = bestPwrs(2);
eval(S.sMdlX);
eval(S.sMdlY);
mdlBest = fitlm(X,yt);
fprintf("Using data in ALL bins -----\n")
x = xdata;
y = ydata;
eval(S.sMdlX);
eval(S.sMdlY);
mdlAll = fitlm(X,yt);
end

function y = fxt(x,pwr)
    if pwr > 0
        y = x.^pwr;
    elseif pwr < 0
        y = 1./x.^abs(pwr);
    else
        y = log(x);
    end
end

function y = fxinv(x,pwr)
    if pwr > 0
        y = x.^(1/pwr);
    elseif pwr < 0
        y = x.^abs(pwr);
    else
        y = exp(x);
    end
end

function [S,x,y]= Init(x,y,xlow,xstep,xhi)
    [rows,cols] = size(x);
    S.N = rows;
    S.V = cols;
    S.B = 4;

    S.sMdlX = "X = [fxt(x(:,1),px1)];";
    S.sMdlY = "yt = fxt(y,py);";
    S.sEvalX = "yhat=cr(1)+cr(2)*fxt(x(:,1),px1);";
    S.sEvalY = "yhat = fxinv(yhat,1/py);";
    % count the total number of models
    count=0;
    for py=xlow(S.V+1):xstep(S.V+1):xhi(S.V+1)
        for px1=xlow(1):xstep(1):xhi(1)
            count=count+1;
        end
    end
    S.M = count;
end

```

```

S.pwrMat = zeros(S.M,S.V+1);
S.stats= zeros(S.M,S.V+2);
i=0;
for py=xlow(S.V+1):xstep(S.V+1):xhi(S.V+1)
    for px1=xlow(1):xstep(1):xhi(1)
        i=i+1;
        S.pwrMat(i,1:S.V+1)=[py,px1];
        % next struct maps rc(1) to rc(4) and RsqrAdj
        S.stats(i,1:S.V+2) = [0,0,0];
    end
end
end

% Note use S.Bin(1,) to store the average values
% of S.Bin(2,) and up.
S.Bin = zeros(S.B,S.M);

end

```

The above function has the same input and output parameters as function `ebrm1()`. The difference between the two functions is the internal code. The above function has the following changes:

- The arrays `S.Blow` and `S.Bhi` have been removed as they are no longer needed, since the function uses one bin.
- Each bin has 80% random elements from the re-shuffled main data arrays `xdata` and `ydata`.

The accompanying test program, `test1Small.m`, is:

```

clc
clear all
diary test1Small.txt
n=100;
xdata=zeros(n,1);
ydata=zeros(n,1);
for i=1:n
    z = 1;
    for j=1:1
        z = z + 10;
        xdata(i,j) = 1+rand*z;
    end
    ydata(i) = 5 + 2 * xdata(i,1);
end

[bestR2,bestMIdx,bestPwrs,bestCoeffs,mdlBest,mdlAll] = ...
    ebrm1Small(xdata,ydata,zeros(1,2),0.5+zeros(1,2),...
    4+zeros(1,2))
save("test1Small.mat", "bestR2","bestPwrs", ...
    "bestCoeffs","mdlBest","mdlAll")
diary off

```

Notice that the above test program works with 100 data points. It calls function `ebrm1Small()`. The output is (which agrees with the previous output):

```

Please wait ....
for Bin 1
Best powers: 1.000000, 1.000000,
Best Model:

bestMdl =

Linear regression model:
  y ~ 1 + x1

Estimated Coefficients:
              Estimate          SE          tStat          pValue
                                                                                    
(Intercept)      5          5.9575e-08      8.3928e+07          0
x1                2          7.6567e-09      2.6121e+08          0

Number of observations: 80, Error degrees of freedom: 78
Root Mean Squared Error: 2.16e-07
R-squared: 1, Adjusted R-Squared: 1
F-statistic vs. constant model: 4.63e+33, p-value = 0

ans =

3x5 table
              SumSq          DF          MeanSq          F          pValue
                                                                                                        
Total          3182.3          79          40.282
Model          3182.3          1          3182.3      4.6273e+33          0
Residual      5.3643e-29          78      6.8772e-31

Finished calculations with Bin 1

Bin # 2
Best R2adj = 1.000000
Best model # 21
Powers: 1.000000, 1.000000,
Bin # 3
Best R2adj = 1.000000
Best model # 21
Powers: 1.000000, 1.000000,
Bin # 4
Best R2adj = 1.000000
Best model # 21
Powers: 1.000000, 1.000000,

Using data in ALL bins -----

bestR2 =

1

```



```
bestMIdx =
    21

bestPwrs =
    1    1

bestCoeffs =
    5.0000    2.0000

mdlBest =

Linear regression model:
y ~ 1 + x1

Estimated Coefficients:

```

	Estimate	SE	tStat	pValue
(Intercept)	5	0	Inf	0
x1	2	0	Inf	0

```

Number of observations: 80, Error degrees of freedom: 78
R-squared: 1, Adjusted R-Squared: 1
F-statistic vs. constant model: 3.81e+31, p-value = 0

mdlAll =

Linear regression model:
y ~ 1 + x1

Estimated Coefficients:

```

	Estimate	SE	tStat	pValue
(Intercept)	5	0	Inf	0
x1	2	0	Inf	0

```

Number of observations: 100, Error degrees of freedom: 98
R-squared: 1, Adjusted R-Squared: 1

```

Models for Heteronomials of Order Two and Up

The study includes versions of ebrm1 that work with heteronomials of orders 2 to 5. The code for these functions is very similar to that of ebrm1. Of course, since these functions have two or more independent variables, the code has expanded parts and bigger arrays/matrices to handle the additional variables.

Readers may ask if they can use the same independent variable in two terms of a regression model. The answer is yes, if each occurrence of that independent variable has power ranges that DO NOT OVERLAP! The matrix xdata would need two columns with the same data. Of course, there is no guarantee that such an approach would yield meaningful results, but one is free to experiment.

Note that the test programs write a copy of the console output to diary files (with .txt extensions). These files contain HTML tags `` and ``. You can delete these tags to get a better view of the text files.

The Zip File

You will find the code for the various versions of the ebrm functions, their test programs, and sample output 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.

<i>Number of Independent Variables</i>	<i>Files</i>
1	ebrm1.m test1.m test1.txt test1Err.m test1Err.txt test1Err2.m test1Err2.txt test11.mat test12.mat test13.mat ebrm1Small.m test1Small.m test1Small.txt test11Small.mat

<i>Number of Independent Variables</i>	<i>Files</i>
2	ebrm2.m test2.m test2.txt test21.mat ebrm2Small.m test2Small.m test2Small.txt test21Small.mat
3	ebrm3.m test3.m test3.txt test31.mat ebrm3Small.m test3Small.m test3Small.txt test31Small.mat
4	ebrm4.m test4.m test4.txt test41.mat ebrm4Small.m test4Small.m test4Small.txt test41Small.mat
5	ebrm5.m test5.m test5.txt test51.mat ebrm5Small.m test5Small.m test5Small.txt test51Small.mat