APPENDIX D
PROGRAMS

There are three MATHEMATCA Programs that are used in this analysis. One of these programs were developed by Girod and Triantis (1996) to perform DEA and FDH efficiency calculations. This program was converted from MATHEMATICA Version 2.2 to Version 3.0. The only code change required was in the syntax of file read statements. The FDH program only allows a fixed number of inputs so this code must be modified as the number of inputs used in the analysis is modified. The second MATHEMATICA program was developed as part of this research to perform Benchmark Correspondence performance calculations. The third program applies the proposed variation on the Benchmark Correspondence method in the next three sections. The listing for each of these programs is provided. Some of the data analysis was done using SAS with the code procedures given in Section D.4.

D.1 FDH Program

Mathematica FDH Application 1.0

MODULE 1 Introduction and Requirements

This Mathematica Notebook has been developed to analyze data using the Data Envelopment Analysis Model. This Notebook requires the user to have available a file containing the data set of input and output parameters of the system. The data must saved in ASCII format and contain the complete data set for the observations (Decision Making Units) to be modelled. The file must have the following characteristics:
Columns- outputs and inputs, all output columns preceeding input columns;i.e. y1 then y2 then x1 x2 x3 columns and so forth.
Rows- Observations(DMUs).
It requires to have these characteristics:

1 4 6
2 5 7
2 4 7

In the following section you will be asked to enter the appropiate information necessary to run the model.. You must make sure that the parameters you enter are correct, because the model will give wrong results if the parameters are

MODULE 2 Specification and Reading of Input Data File

Specifying Input Parameters

```
$PrePrint = Short[#1, 2]&;

Input["How many outputs?"];
```

```
out = %
```

```
General::spell1 :
 Possible spelling error: new symbol name "out" is similar to existing symbol "Out".
```

```
Input["How many inputs?"];
```

```
in = %
```

```
Input["How many observations?"];
```

```
obs = %
```

```
Input["Input a value for Sigma?"];
```

```
sig = %
```

Reading the Data File

```
InputString["Please type in the complete  file name along with the
```

```
filename = %
```

```
OpenRead[filename]
```

■ Creating Variables and giving them test values

```
datamatrix = Do[Do[y[i, k] = Read[filename, Number], {i, out}]; Do[x
```

```
Null
Null
```

■ MODULE 4 FDH Model

▪ Input Reducing Model (FDH-IRM)

```
Do[Do[Do[temp[l, k] = If[x[i, l] ≥ x[i, k], If[y[1, l] ≤ y[1, k], Max[x[1, k]/x[1, l], x[2, k]/x[2, l], x[3
Do[Fdh[l] = Min[temp[l, 1], temp[l, 2], temp[l, 3], temp[l, 4], temp[l, 5], temp[l, 6], temp[
   temp[l, 14], temp[l, 15], temp[l, 16], temp[l, 17], temp[l, 18], temp[l, 19], temp[l, 20]
   temp[l, 47], temp[l, 48], temp[l, 49], temp[l, 50]], {l, obs}]
Do[Print[l]; Print[Fdh[l]], {l, obs}]
Print[Array[Fdh, {obs}]]
Do[Do[If[temp[l, k] < 1, Print[l]; Print[k]], {k, obs}], {l, obs}]
```

D.2 Benchmark Correspondence Program

Benchmark Correspondence

Read Data and Initialize Variables

Specifying Input Parameters

```
$PrePrint = Short[#1, 2]&;

Input["How many desirable outputs?"];

dout = %

Input["How many inputs?"];

in = %

Input["How many observations?"];

maxobs = %
```

Reading the Data File

```
InputString["Please type in the complete file name along with the

filename = %
```

OpenRead[filename]

```
results = OpenWrite["data/full_1out.txt"]
```

```
OutputStream[data/benchout.txt, 12]
```

Create Variables and give them test values

```
datamatrix = Do[Do[y[i, t] = Read[filename, Number], {i, dout}];
  Do[x[j, t] = Read[filename, Number], {j, in}], {t, maxobs}]
Write[results, "Data Values Outputs"]
Do[Do[Write[results, y[i, t]], {i, dout}, {t, maxobs}]]
Write[results, "Data Values for Inputs"]
Do[Do[Write[results, x[j, t]], {j, in}, {t, maxobs}]]
```

Partition the Data Set

The sets Dg (dominating), Dd (dominated), and Di (dominance indifferent) are defined. Undesirable outputs are ignored.

Initialize the array, Di, of Benchmark Observations. The first observtion, t=1, is a Benchmark Observation.

The values of the array Di are all set to a value of zero. The first observation is a benchmark observation.

```
Do[Di[t] = 0, {t, maxobs}]
Di[1] = 1
```

Partition Observations

Each observation, t, is compared to all observations in the set that is a benchmark observation - Di[t]=1. The first observation, t=1, is a benchmark observation. The second observation, at t=2, is a benchmark if it neither dominates or is dominated by the previous observation. For the remaining observations, t > 2, it is retained as a benchmark if it neither dominates or is dominated by any of the previous observations that were retained as benchmark observations at the time they were made. The one dimensional array Di[observations] is used to indicate whether or not an observation at time, t, is dominated by the benchmark observations, dominates the benchmark observations, or is itself a benchmark observations. If Di[t] = 1 the observation is a benchmark. If Di[t] = 3 the observation dominates the benchmarks up until the time t. If Di[t] = 2 the observation is dominated by the benchmarks up until the time t.

```
benchcount = 1
Do[Dd = 0; Dg = 0;
  Do[If[Di[t1] == 1, weakindom = 0;
   weakoutdom = 0; docount = 0; gicount = 0; gocount = 0; dicount = 0
    Do[If[x[i, t1] <= x[i, t], gicount = gicount + 1],
     {i, in}];
    Do[If[y[j, t1] >= y[j, t], gocount = gocount + 1],
     {j, dout}];
    DO[IF[x[i, t1] == x[i, t], weakindom = weakindom + 1],
     {i, in}];
    DO[IF[y[j, t1] == y[j, t], weakoutdom = weakoutdom + 1],
     {j, dout}];
    If[gicount == in &&
     (weakindom != in || weakoutdom != dout) && gocount == dout, Dg = Dg + 1];
     weakindom = 0; weakoutdom = 0;
    Do[If[x[i, t1] >= x[i, t], dicount = dicount + 1],
     {i, in}];
   Do[If[y[j, t1] <= y[j, t], docount = docount + 1],
    {j, dout}];
   DO[IF[x[i, t1] == x[i, t], weakindom = weakindom + 1],
    {i, in}];
    DO[IF[y[j, t1] == y[j, t], weakoutdom = weakoutdom + 1],
     {j, dout}];
    If[dicount == in &&
     (weakindom != in || weakoutdom != dout) && docount == dout, Dd = Dd + 1]],
      {t1, t}];
   If[Dd != benchcount && Dg != benchcount,   Di[t] = 1; benchcount = benchcount + 1];
   If[Dg == benchcount, Di[t] = 2];
   If[Dd == benchcount, Di[t] = 3];
   Write[
  results, "OUTSIDE LOOP", "Dd =", Dd, "Dg = ", Dg, "benchcount =", benchcount],
   {t, 2, maxobs}]
```

```
Write[results, "Dominance Results for each observation - 1 is Di, 2 is Dg, 3 is Dd"]
Do[Write[results, Di[t]], {t, maxobs}]
```

Evaluate the Efficiency of Progress and Regress Observations.

For the vector Di[t1], if equal to 1 then the obervation at time t is a benchmark observation, if equal to 2 then the observation at time t is a regress observation, if equal to 3 then the observation at time t is a progress observation. A regress (or a progress) observation is then compared to all previous dominance indifferent observations.

Regress Observations

The metric of regress takes the maximum ratio of inputs (or outputs) for the regress observation compared to a benchmark observation. The minimum of this ratio for the regress observation as compared to all previous benchmark observations is then taken as the metric.

```
Do[If[Di[t] == 2, inlist = {∞}; outlist = {∞};
     Do[
        If[Di[t1] == 1, inratio = {0}; outratio = {0};
         Do[AppendTo[inratio, x[i, t1] / x[i, t]], {i, in}];
         Do[AppendTo[outratio, y[j, t] / y[j, t1]], {j, dout}];
         maxin = Max[inratio]; AppendTo[inlist, maxin];
         maxout = Max[outratio]; AppendTo[outlist, maxout];
        Write[results, "t1= ", t1, "maxin= ", maxin, "maxout= ", maxout]],
     {t1, t}];
     inmetric = Min[inlist]; outmetric = Min[outlist];
Write[results, "Regress", "out=  ", outmetric, "in= ", inmetric]],
{t, maxobs}]
```

Progress Observations

The metric of progress takes the minimum ratio of inputs (or outputs) for the progress observation compared to a benchmark observation. The maximum of this ratio for the progress observation as compared to all previous benchmark observations is then taken as the metric.

```
Do[If[Di[t] == 3, inlist = {-∞}; outlist = {-∞};
     Do[
        If[Di[t1] == 1, inratio = {∞}; outratio = {∞};
         Do[AppendTo[inratio, x[i, t1] / x[i, t]], {i, in}];
         Do[AppendTo[outratio, y[j, t] / y[j, t1]], {j, dout}];
         minin = Min[inratio]; AppendTo[inlist, minin];
         minout = Min[outratio]; AppendTo[outlist, minout];
        Write[results, "t1= ", t1, "minin= ", minin, "minout= ", minout]],
      {t1, t}];
      inmetric = Max[inlist]; outmetric = Max[outlist];
     Write[results, "Progress", "out = ", outmetric, "in = ", inmetric]],
{t, maxobs}]


Close[results]
```

D.3 Proposed Method

Dominance Based Performance

MODULE 2 Specification and Reading of Input Data File

Specifying Input Parameters

```
$PrePrint = Short[#1, 2]&;

Input["How many desirable outputs?"];

dout = %

Input["How many undesirableoutputs?"];

badout = %

Input["How many inputs?"];

in = %
```

```
Input["How many observations?"];
```

```
maxobs = %
```

```
Input["At what observation in the time series do calculations begin?"]
```

```
minobs = %
```

Reading the Data File

```
InputString["Enter the complete file name along with the directory
```

```
filename = %
```

```
OpenRead[filename]
```

```
results = OpenWrite["data/proposed8ut.txt"]
```

```
OutputStream[data/proposed8out.txt, 23]
```

■ Creating Variables and giving them test values

```
numout = dout + badout;
datamatrix = Do[Do[y[i, t] = Read[filename, Number], {i, numout}];
  Do[x[j, t] = Read[filename, Number], {j, in}], {t, maxobs}]
Do[Do[Write[results, "Outputs ", y[i, t]], {i, numout}], {t, maxobs
Do[Do[Write[results, "Inputs ", x[j, t]], {j, in}], {t, maxobs}]
```

Partition the Data Set

The sets Dg (dominating), Dd (dominated), and Di (dominance indifferent) are defined, but these sets do not correspond to Benchmark Correspondence Sets. In Benchmark Correspondence, each observation at time t, is compared to the previous benchmark observations (in Di) and then either placed in the Dominance Indifferent set, Di, or not. The approach here is to at each time, t, partition all observations, <= t, into sets based on the observation at time, t, as the reference.

Define the arrays of Dg, Dd, and Di - relative to the observation at time t

The values of the arrays Dg, Dd, and Di of diminsion t by t are all set to a value of zero.

```
Do[Do[Di[t1, t] = 0; Dg[t1, t] = 0; Dd[t1, t] = 0, {t1, t}], {t, maxobs}]
```

Partition Dominating Observations - Dg - Tulkens Definition

A two diminsional Array, Dg, is used to track the results of dominance comparisons. This array is of dimension [t, maxobs], where maxobs is the total number of observations and t is the reference production plan being evaluated. Where a dominant observation is found it is designated with a value of "1" in the array Dg.

```
Do[Do[incount = 0; outcount = 0; weakin = 0; weakout = 0;
  Do[
    If[x[i, t1] <= x[i, t], incount = incount + 1],
    {i, in}];
  Do[
    If[y[j, t1] >= y[j, t], outcount = outcount + 1],
    {j, dout}];
  Do[If[x[i, t1] == x[i, t], weakin = weakin + 1],
    {i, in}];
  Do[If[y[j, t1] == y[j, t], weakout = weakout + 1],
    {j, dout}];
  If[
   incount == in && outcount == dout && (weakin != in || weakout != dout), Dg[t1, t] = 1],
 {t1, t - 1}],
{t, minobs, maxobs}]


Do[Print["Dg - Partition for time t = ", t]; Do[Print[Dg[t1, t]], {t1, t - 1}],
 {t, minobs, maxobs}]
```

Partition Dominated Observations - Dd - Tulkens Definition

A two diminsional Array, Dd, is used to track the results of dominance comparisons. This array is of dimension [t,maxobs], where maxobs is the total number of observations and t is the reference production plan being evaluated. Where a dominated observation is found it is designated with a value of "1" in the array Dd.

```
Do[Do[incount = 0; outcount = 0; weakin = 0; weakout = 0;
  Do[
    If[x[i, t1] >= x[i, t], incount = incount + 1],
    {i, in}];
  Do[
    If[y[j, t1] <= y[j, t], outcount = outcount + 1],
    {j, dout}];
  Do[If[x[i, t1] == x[i, t], weakin = weakin + 1],
    {i, in}];
  Do[If[y[j, t1] == y[j, t], weakout = weakout + 1],
    {j, dout}];
  If[
   incount == in && outcount == dout && (weakin != in || weakout != dout), Dd[t1, t] = 1],
 {t1, t - 1}],
{t, minobs, maxobs}]


Do[Print["Dd - Partition for time t = ", t]; Do[Print[Dd[t1, t]], {t1, t - 1}],
 {t, minobs, maxobs}]
```

Partition Dominance Indifferent Observations - Di - Tulkens Definition

Those observations that do not belong to Dd or Dg belong to the set Di.

```
Do[
 Do[
  If[Dd[t1, t] == 0, If[Dg[t1, t] == 0,
     Di[t1, t] = 1]],
 {t1, t - 1}],
{t, minobs, maxobs}]
  Do[Print["Di - Partition for time t = ", t];
 Do[Print[Di[t1, t]], {t1, t - 1}], {t, minobs, maxobs}]
```

Partition Observations with Undesirable Outputs into Sets, TE, TI, and TU

Define the arrays of TE, TI, and TU

The values of the arrays TE, TI, and TU of diminsion t by t are all set to a value of zero.

```
Do[Do[TE[t1, t] = 0; TI[t1, t] = 0; TU[t1, t] = 0, {t1, t}], {t, maxobs}]
```

Define the Technically Efficiency Set, TE

Define the set that is Technically Efficiency - TE.

```
Do[Do[incount = 0; outcount = 0; badcount = 0; weakin = 0; weakout = 0; weakbad = 0;
  Do[If[x[i, t1] <= x[i, t], incount = incount + 1],
     {i, in}];
  Do[If[y[j, t1] >= y[j, t], outcount = outcount + 1],
     {j, dout}];
  If[badout > 0,
     Do[If[y[j, t1] <= y[j, t], badcount = badcount + 1],
     {j, dout + 1, badout}];
     Do[If[y[j, t1] == y[j, t], weakbad = weakbad + 1],
     {j, dout + 1, badout}]];
 Do[If[x[i, t1] == x[i, t], weakin = weakin + 1],
     {i, in}];
 Do[If[y[j, t1] == y[j, t], weakout = weakout + 1],
     {j, dout}];
  If[incount == in && outcount == dout && badcount == badout &&
     (weakin != in || weakout != dout || weakbad != badout), TE[t1, t] = 1],
{t1, t - 1}],
{t, minobs, maxobs}]
Do[Print["TE - Partition for time t = ", t]; Do[Print[TE[t1, t]], {t1, t - 1}],
 {t, minobs, maxobs}]
```

Define the Technically InEfficient Set, TI

Define the set that is Technically InEfficiency - TI.

```
Do[Do[incount = 0; outcount = 0; badcount = 0; weakin = 0; weakout = 0; weakbad = 0;
   Do[If[x[i, t1] >= x[i, t], incount = incount + 1],
     {i, in}];
  Do[If[y[j, t1] <= y[j, t], outcount = outcount + 1],
     {j, dout}];
  If[badout > 0,
     Do[If[y[j, t1] >= y[j, t], badcount = badcount + 1],
     {j, dout + 1, badout}];
     Do[If[y[j, t1] == y[j, t], weakbad = weakbad + 1],
     {j, dout + 1, badout}]];
 Do[If[x[i, t1] == x[i, t], weakin = weakin + 1],
     {i, in}];
 Do[If[y[j, t1] == y[j, t], weakout = weakout + 1],
     {j, dout}];
  If[incount == in && outcount == dout && badcount == badout &&
     (weakin != in || weakout != dout || weakbad != badout), TI[t1, t] = 1],
{t1, t - 1}],
{t, minobs, maxobs}]
Do[Print["TI - Partition for time t = ", t]; Do[Print[TI[t1, t]], {t1, t - 1}],
 {t, minobs, maxobs}]
```

Partition Dominance Indifferent Observations - TU

Those observations that do not belong to TE or TI belong to the set TU. The observation t that is the reference is always assigned placed in the set TU.

```
Do[
 Do[
  If[TE[t1, t] == 0, If[TI[t1, t] == 0,
     TU[t1, t] = 1]],
 {t1, t - 1}],
{t, minobs, maxobs}]
  Do[Print["TU - Partition for time t = ", t];
 Do[Print[TU[t1, t]], {t1, t - 1}], {t, minobs, maxobs}]
```

Define Membership in the Sets, SE and SI

This part of the program is only executed if there are 18 inputs. The program also assumes that there is 1 product output, and 3 undesirable outputs.

Define the Substitution Efficient Set, SE.

Only the observations that are part of the set, TU, are evaluated. The program is written for the classification scheme of the actual data as described in Chapter 4. The outputs must be in the following order: 1 - Boards (product), 2 - Scrap, 3 - Rework, 4 - Water. The inputs follow the outputs and are in the following order: 1 - Salary Hours, 2 - Labor Hours, 3 through 17 for Inputs 1 through 15, 18 - Power.

The classification of the outputs is: Product (Boards), Recycled Output (Scrap, ReWork), and Permitted Discharge (Water). The classification of inputs is: Non-Material (Salary Hours, Labor Hours), Recycled Input (Input 1, Input 2, Inpt 3, Input 4, and Input 5), Renewable Inputs (Inputs 10 and 12), Non-Renewable Inputs (Input 6, Power), and Not Classified (Input 7, Input 8, Input 9, Input 11, Input 15, 13, 14).

The index for the y array of outputs are the following: product output (1), recycled outputs (2, 3), and permitted discharge (4).

The index for the x array of inputs are the following: non-material inputs (1, 2), recycled inputs (3, 4, 5, 6, 7), renewable inputs (12, 14), and non-renewable inputs (8, 18). All other inputs are part of the not-classified category and are not part of the algorithm for determining membership in the sets SE and SI.

The algorithm is to:

Only perform the test for production plans in the set, TU.

Determine for each subset of inputs if desirable substitution is occurring (less desirable inputs decreasing with more desirable increasing). If so the production plan is a member of set SE.

Determine for each subset of outputs if desirable substitution is occurring (less desirable outputs decreasing with more desirable increasing). If so the production plan is a member of set SE.

Determine for each subset of inputs if undesirable substitution is occurring (more desirable inputs decreasing with less desirable increasing). If so the production plan is a member of set SI.

Determine for each subset of outputs if undesirable substitution is occurring (more desirable outputs decreasing with less desirable increasing). If so the production plan is a member of set SI.

```
Do[Do[SE[t1, t] = 0; SI[t1, t] = 0, {t1, t}], {t, maxobs}]
```

■ Substitution Efficient in Inputs - SE

```
If[in == 18, Do[Do[maybe = 0; upper = 0; lower = 0;
  If[ TU[t1, t] == 1 && x[1, t1] >= x[1, t] && x[2, t1] >= x[2, t] &&
    x[3, t1] <= x[3, t] && x[4, t1] <= x[4, t] && x[5, t1] <= x[5, t] &&
     x[6, t1] <= x[6, t] && x[7, t1] <= x[7, t] && x[8, t1] <= x[8, t] &&
     x[12, t1] <= x[12, t] && x[14, t1] <= x[14, t] && x[18, t1] <= x[18, t],
    Print["maybe is 1", t1, t]; maybe = 1];
  If[ x[1, t1] == x[1, t] && x[2, t1] == x[2, t], upper = 1];
  If[ x[3, t1] == x[3, t] && x[4, t1] == x[4, t] && x[5, t1] == x[5, t] &&
     x[6, t1] == x[6, t] && x[7, t1] == x[7, t] && x[8, t1] == x[8, t] &&
     x[12, t1] == x[12, t] && x[14, t1] == x[14, t] && x[18, t1] == x[18, t], lower = 1];
   If[maybe == 1 && upper != 1 && lower != 1,
    Print["found one", "t1 =", t1, "t =", t]; SE[t1, t] = 1],
  {t1, t - 1}],
 {t, minobs, maxobs}]]
```

```
If[in == 18, Do[Do[maybe = 0; upper = 0; lower = 0;
  If[ TU[t1, t] == 1 && x[1, t1] >= x[1, t] && x[2, t1] >= x[2, t] &&
    x[3, t1] >= x[3, t] && x[4, t1] >= x[4, t] && x[5, t1] >= x[5, t] &&
     x[6, t1] >= x[6, t] && x[7, t1] >= x[7, t] && x[8, t1] <= x[8, t] &&
     x[12, t1] <= x[12, t] && x[14, t1] <= x[14, t] && x[18, t1] <= x[18, t],
    Print["maybe is 1", t1, t]; maybe = 1];
  If[ x[1, t1] == x[1, t] &&
     x[2, t1] == x[2, t] && x[3, t1] == x[3, t] && x[4, t1] == x[4, t] &&
     x[5, t1] == x[5, t] && x[6, t1] == x[6, t] && x[7, t1] == x[7, t], upper = 1];
  If[ x[8, t1] == x[8, t] && x[12, t1] == x[12, t] &&
     x[14, t1] == x[14, t] && x[18, t1] == x[18, t], lower = 1];
   If[maybe == 1 && upper != 1 && lower != 1,
    Print["found one", "t1 =", t1, "t =", t]; SE[t1, t] = 1],
  {t1, t - 1}],
 {t, minobs, maxobs}]]
```

```
If[in == 18, Do[Do[maybe = 0; upper = 0; lower = 0;
  If[ TU[t1, t] == 1 && x[1, t1] >= x[1, t] && x[2, t1] >= x[2, t] &&
    x[3, t1] >= x[3, t] && x[4, t1] >= x[4, t] && x[5, t1] >= x[5, t] &&
     x[6, t1] >= x[6, t] && x[7, t1] >= x[7, t] && x[8, t1] <= x[8, t] &&
     x[12, t1] >= x[12, t] && x[14, t1] >= x[14, t] && x[18, t1] <= x[18, t],
    Print["maybe is 1", t1, t]; maybe = 1];
  If[ x[1, t1] == x[1, t] && x[2, t1] == x[2, t] && x[3, t1] == x[3, t] &&
     x[4, t1] == x[4, t] && x[5, t1] == x[5, t] && x[6, t1] == x[6, t] &&
     x[7, t1] == x[7, t] && x[12, t1] == x[12, t] && x[14, t1] == x[14, t], upper = 1];
  If[x[8, t1] == x[8, t] && x[18, t1] == x[18, t], lower = 1];
   If[maybe == 1 && upper != 1 && lower != 1,
    Print["found one", "t1 =", t1, "t =", t]; SE[t1, t] = 1],
   {t1, t - 1}],
 {t, minobs, maxobs}]]


Print["Membership in the set SE for Inputs"]
Do[Print["Partition for time t = ", t]; Do[Print[SE[t1, t]], {t1, t - 1}],
 {t, minobs, maxobs}]
```

■ Substitution Efficient in Outputs - SE

```
If[dout == 1 && badout == 3, Do[Do[maybe = 0; upper = 0; lower = 0;
  If[ TU[t1, t] == 1 && y[1, t1] >= y[1, t] && y[2, t1] <= y[2, t] &&
    y[3, t1] <= y[3, t] && y[4, t1] <= y[4, t], Print["maybe is 1", t1, t]; maybe = 1];
  If[ y[1, t1] == y[1, t], upper = 1];
  If[y[2, t1] == y[2, t] && y[3, t1] == y[3, t] && y[4, t1] == y[4, t], lower = 1];
   If[maybe == 1 && upper != 1 && lower != 1,
    Print["found one", "t1 =", t1, "t =", t]; SE[t1, t] = 1],
   {t1, t - 1}],
 {t, minobs, maxobs}]]


If[dout == 1 && badout == 3, Do[Do[maybe = 0; upper = 0; lower = 0;
  If[ TU[t1, t] == 1 && y[1, t1] >= y[1, t] && y[2, t1] >= y[2, t] &&
    y[3, t1] >= y[3, t] && y[4, t1] <= y[4, t], Print["maybe is 1", t1, t]; maybe = 1];
  If[ y[1, t1] == y[1, t] && y[2, t1] == y[2, t] && y[3, t1] == y[3, t] , upper = 1];
  If[ y[4, t1] == y[4, t], lower = 1];
   If[maybe == 1 && upper != 1 && lower != 1,
    Print["found one", "t1 =", t1, "t =", t]; SE[t1, t] = 1],
   {t1, t - 1}],
 {t, minobs, maxobs}]]



Print["Membership in the set SE for Inputs and Outputs"]
Do[Print["Partition for time t = ", t]; Do[Print[SE[t1, t]], {t1, t - 1}],
 {t, minobs, maxobs}]
```

■ Substitution Inefficient in Inputs - SI

```
If[in == 18, Do[Do[maybe = 0; upper = 0; lower = 0;
  If[ TU[t1, t] == 1 && x[1, t1] <= x[1, t] && x[2, t1] <= x[2, t] &&
    x[3, t1] >= x[3, t] && x[4, t1] >= x[4, t] && x[5, t1] >= x[5, t] &&
     x[6, t1] >= x[6, t] && x[7, t1] >= x[7, t] && x[8, t1] >= x[8, t] &&
     x[12, t1] >= x[12, t] && x[14, t1] >= x[14, t] && x[18, t1] >= x[18, t],
    Print["maybe is 1", t1, t]; maybe = 1];
  If[ x[1, t1] == x[1, t] && x[2, t1] == x[2, t], upper = 1];
  If[ x[3, t1] == x[3, t] && x[4, t1] == x[4, t] && x[5, t1] == x[5, t] &&
     x[6, t1] == x[6, t] && x[7, t1] == x[7, t] && x[8, t1] == x[8, t] &&
     x[12, t1] == x[12, t] && x[14, t1] == x[14, t] && x[18, t1] == x[18, t], lower = 1];
   If[maybe == 1 && upper != 1 && lower != 1,
    Print["found one", "t1 =", t1, "t =", t]; SI[t1, t] = 1],
   {t1, t - 1}],
 {t, minobs, maxobs}]]
```

```
If[in == 18, Do[Do[maybe = 0; upper = 0; lower = 0;
  If[ TU[t1, t] == 1 && x[1, t1] <= x[1, t] && x[2, t1] <= x[2, t] &&
    x[3, t1] <= x[3, t] && x[4, t1] <= x[4, t] && x[5, t1] <= x[5, t] &&
     x[6, t1] <= x[6, t] && x[7, t1] <= x[7, t] && x[8, t1] >= x[8, t] &&
     x[12, t1] >= x[12, t] && x[14, t1] >= x[14, t] && x[18, t1] >= x[18, t],
    Print["maybe is 1", t1, t]; maybe = 1];
  If[ x[1, t1] == x[1, t] &&
     x[2, t1] == x[2, t] && x[3, t1] == x[3, t] && x[4, t1] == x[4, t] &&
     x[5, t1] == x[5, t] && x[6, t1] == x[6, t] && x[7, t1] == x[7, t], upper = 1];
  If[ x[8, t1] == x[8, t] && x[12, t1] == x[12, t] &&
     x[14, t1] == x[14, t] && x[18, t1] == x[18, t], lower = 1];
   If[maybe == 1 && upper != 1 && lower != 1,
    Print["found one", "t1 =", t1, "t =", t]; SI[t1, t] = 1],
   {t1, t - 1}],
 {t, minobs, maxobs}]]
```

```
If[in == 18, Do[Do[maybe = 0; upper = 0; lower = 0;
  If[ TU[t1, t] == 1 && x[1, t1] <= x[1, t] && x[2, t1] <= x[2, t] &&
    x[3, t1] <= x[3, t] && x[4, t1] <= x[4, t] && x[5, t1] <= x[5, t] &&
     x[6, t1] <= x[6, t] && x[7, t1] <= x[7, t] && x[8, t1] >= x[8, t] &&
     x[12, t1] <= x[12, t] && x[14, t1] <= x[14, t] && x[18, t1] >= x[18, t],
    Print["maybe is 1", t1, t]; maybe = 1];
  If[ x[1, t1] == x[1, t] && x[2, t1] == x[2, t] && x[3, t1] == x[3, t] &&
     x[4, t1] == x[4, t] && x[5, t1] == x[5, t] && x[6, t1] == x[6, t] &&
     x[7, t1] == x[7, t]  && x[12, t1] == x[12, t] && x[14, t1] == x[14, t], upper = 1];
  If[x[8, t1] == x[8, t] && x[18, t1] == x[18, t], lower = 1];
   If[maybe == 1 && upper != 1 && lower != 1,
    Print["found one", "t1 =", t1, "t =", t]; SI[t1, t] = 1],
   {t1, t - 1}],
 {t, minobs, maxobs}]]


Print["Membership in the set SI for Inputs"]
Do[Print["Partition for time t = ", t]; Do[Print[SI[t1, t]], {t1, t - 1}],
 {t, minobs, maxobs}]
```

■ Substitution Inefficient in Outputs - SI

```
If[dout == 1 && badout == 3, Do[Do[maybe = 0; upper = 0; lower = 0;
  If[ TU[t1, t] == 1 && y[1, t1] <= y[1, t] && y[2, t1] >= y[2, t] &&
    y[3, t1] >= y[3, t] && y[4, t1] >= y[4, t], Print["maybe is 1", t1, t]; maybe = 1];
  If[ y[1, t1] == y[1, t], upper = 1];
  If[y[2, t1] == y[2, t] && y[3, t1] == y[3, t] && y[4, t1] == y[4, t], lower = 1];
   If[maybe == 1 && upper != 1 && lower != 1,
    Print["found one", "t1 =", t1, "t =", t]; SI[t1, t] = 1],
   {t1, t - 1}],
 {t, minobs, maxobs}]]



If[dout == 1 && badout == 3, Do[Do[maybe = 0; upper = 0; lower = 0;
  If[ TU[t1, t] == 1 && y[1, t1] <= y[1, t] && y[2, t1] <= y[2, t] &&
    y[3, t1] <= y[3, t] && y[4, t1] >= y[4, t], Print["maybe is 1", t1, t]; maybe = 1];
  If[ y[1, t1] == y[1, t] && y[2, t1] == y[2, t] && y[3, t1] == y[3, t] , upper = 1];
  If[ y[4, t1] == y[4, t], lower = 1];
   If[maybe == 1 && upper != 1 && lower != 1,
    Print["found one", "t1 =", t1, "t =", t]; SI[t1, t] = 1],
   {t1, t - 1}],
 {t, minobs, maxobs}]]


Print["Membership in the set SI for Inputs and Outputs"]
Do[Print["Partition for time t = ", t]; Do[Print[SI[t1, t]], {t1, t - 1}],
 {t, minobs, maxobs}]
```

Set Membership Counts

The membership in each set (Dg, Dd, Di, TE, TI, TU, SE, SI, and TU without SE and SI) is counted for each time t.

```
Do[numDi[t] = 0; numDg[t] = 0; numDd[t] = 0; numTE[t] = 0;
 numTI[t] = 0; numTU[t] = 0; numSE[t] = 0; numSI[t] = 0, {t, maxobs}]
```

```
Do[
 Do[
  If[Dg[t1, t] == 1, numDg[t] = numDg[t] + 1];
  If[Dd[t1, t] == 1, numDd[t] = numDd[t] + 1];
  If[Di[t1, t] == 1, numDi[t] = numDi[t] + 1];
  If[TE[t1, t] == 1, numTE[t] = numTE[t] + 1];
  If[TI[t1, t] == 1, numTI[t] = numTI[t] + 1];
  If[TU[t1, t] == 1, numTU[t] = numTU[t] + 1];
  If[SE[t1, t] == 1, numSE[t] = numSE[t] + 1];
  If[SI[t1, t] == 1, numSI[t] = numSI[t] + 1],
  {t1, t - 1}],
 {t, minobs, maxobs}]
```

```
 Do[Print["Count for Dg at time t ", t];   Print[numDg[t]], {t, minobs, maxobs}]
Do[ Print["Count for Dd at time t ", t];
 Print[numDd[t]], {t, minobs, maxobs}]
Do[Print["Count for Di at time t ", t];
 Print[numDi[t]], {t, minobs, maxobs}]
  Do[ Print["Count for TE at time t ", t];
 Print[numTE[t]], {t, minobs, maxobs}]
Do[ Print["Count for TI at time t ", t];
 Print[numTI[t]], {t, minobs, maxobs}]
Do[ Print["Count for TU at time t ", t];
 Print[numTU[t]], {t, minobs, maxobs}]
 Do[Print["Count for SE at time t ", t];
 Print[numSE[t]], {t, minobs, maxobs}]
 Do[ Print["Count for SI at time t ", t];
 Print[numSI[t]], {t, minobs, maxobs}]
```

Distance Measures

Each distance measure is calculated for the distance between all observations t and all observations < t. Previously defined Arrays indicate the set membership of each observation.

Define the Arrays that store the distance measures - dsum[m, t], deuclid[m,t], daxial [m,t]

```
Do[Do[badeuclid[t1, t] = 0; xeuclid[t1, t] = 0; ueuclid[t1, t] = 0, {t1, t - 1}],
 {t, maxobs}]
```

Euclidean Distance - Inputs

```
Do[
 Do[distance = 0.0;
  Do[distance = distance + (Abs[x[i, t] - x[i, t1]])^2, {i, in}];
  xeuclid[t1, t] = Sqrt[distance], {t1, t - 1}],
 {t, maxobs}]
Do[Print["xEuclid", "     t= ", t]; Do[Print[xeuclid[t1, t]], {t1, t - 1}], {t, maxobs}]
```

Euclidean Distance - Outputs

```
Do[
 Do[distance = 0.0;
  Do[distance = distance + (Abs[y[i, t] - y[i, t1]])^2, {i, dout}];
  ueuclid[t1, t] = Sqrt[distance], {t1, t - 1}],
 {t, maxobs}]
Do[Print["uEuclid", "     t= ", t]; Do[Print[ueuclid[t1, t]], {t1, t - 1}], {t, maxobs}]
```

Euclidean Distance - Undesirable Outputs

```
Do[
 Do[distance = 0.0;
  Do[distance = distance + (Abs[y[i, t] - y[i, t1]])^2, {i, dout + 1, dout + badout}];
  badeuclid[t1, t] = Sqrt[distance], {t1, t - 1}],
 {t, maxobs}]
Do[
 Print["badEuclid", "    t= ", t]; Do[Print[badeuclid[t1, t]], {t1, t - 1}], {t, maxobs}]
```

D.4 SAS Program

The SAS program was simply the “proc corr” procedure and the “proc reg” procedure. The options on the “ proc reg” procedure were “/ COLLINOINT P R DW PARTIAL.”