DEPARTMENT OF AGRICULTURAL ECONOMICS AND ECONOMICS

Measuring Technical Efficiency and Capacity with Data Envelopment Analysis: A foundational approach using the R programming language

BY

JOHN B. WALDEN
JOE ATWOOD

STAFF PAPER 2019-2

COLLEGE OF AGRICULTURE
COLLEGE OF LETTERS AND SCIENCE

MONTANA
STATE UNIVERSITY

Measuring Technical Efficiency and Capacity with Data Envelopment Analysis: A foundational approach using the R programming language

John B. Walden*

NOAA/ NMFS/NEFSC

166 Water St. MB19

Woods Hole, MA 02543

508-495-4726

John.Walden@Noaa.Gov

Joe Atwood

Department of Agricultural Economics and Economics

Montana State University

P.O. Box 172920

Bozeman, MT 59717

jatwood@Montana.edu

*Senior authorship shared

Introduction

Data Envelopment Analysis (DEA), originally introduced by Charnes, et al. (1978) has become a popular analytical method for assessing efficiency, capacity, productivity and conducting benchmarking studies. Along with Stochastic Frontier Analysis (SFA), DEA has been used in an abundant and ever growing number of empirical studies over the last 40 years, in a diverse number of fields (Daraio, et al., 2019). DEA models can be constructed and run using a variety of software packages, such as GAMS, Excel, SAS, LINDO, along with some programs which were specifically built for DEA models such as DEAP and FRONTIER. Since DEA models are in reality linear programming (L.P.) models, any software which has an L.P. solver can be used to solve DEA problems.

Over the past several years, the software product R has become popular, and a built-in DEA and SFA package 'Benchmarking' has been developed that can easily be used to solve DEA models. This workbook is designed to demonstrate how to build R programs to solve DEA models for those who want to "build their own" rather than use the Benchmarking program. There are several advantages to this approach. First and foremost, it gives analysts the ability to customize their DEA models in whatever way they desire, and the ability to retrieve and display whatever results they wish by directly using the LP solvers in R. Secondly, if new L.P. solvers are incorporated into R, the programs could still be used with a few minor changes, potentially improving solve times for complex problems. Thirdly, the approach outlined in this workbook removes a layer for the modeler. Instead of using a package such as Benchmarking, or Fear (for bootstrapping models), the user directly solves the problems in an LP framework.

The programs that we show are simplified versions of DEA models solved using the R package lpSolve_API. We include the standard input and output oriented efficiency problem, along with the Johansen capacity model and two directional distance function models. The first directional distance function model expands outputs while contracting inputs, while the second is one used for an environmental problem where efficiency is measured in terms of expanding "desirable" outputs, while contracting "undesirable" outputs while leaving inputs unchanged. All the models shown can be found in the book "Production Frontiers" (Fare, et al., 1994), or the text "New Directions: Efficiency and Productivity" (Färe and Grosskopf, 2006). We also include a bootstrap routine based on a sub-sample bootstrap model, which runs quite fast and gives consistent bootstrap estimates. For each example, the mathematical formulation of the DEA model in equation form is presented, followed by the matrix representation, and then the R code used to generate the model. Some examples include the actual matrices that are used by the R program for small problems using a subset of the original data. Results for the first five models, along with a sample 10 observation data set are given in Appendix 2.

1.0 Input Oriented Technical Efficiency

Input oriented technical efficiency measures the ability of a firm to contract their inputs given their desire to produce a certain output. Fare, et al. (1994) proposed the following DEA model to measure input oriented technical efficiency with constant returns to scale (CRS; page64):

$$\begin{gathered} TE = min_{\lambda,z}\lambda \\ s.t. \\ y_{jm} \leq \sum_{j=1}^{J} z_j y_{jm}, m = 1,2, \dots M \\ \sum_{j=1}^{J} z_j \, x_{jn} \leq \lambda x_{jn}, n = 1,2,.., N, \\ z_j \geq 0, j = 1,2, \dots J. \end{gathered} \quad (1.1)$$

Here, y_{jm} is the quantity of output *m* produced by firm *j*, and x_{jn} is the quantity of input *n* used by firm *j* to produce output *m*. The z_j term is an individual weight given to each firm, while λ is the objective function value. Both z and λ are decision variables chosen by the model, which is estimated once for each firm in the data. The model is solved in a loop once for each firm, and returns a value that shows how much the firm needs to contract its inputs in order to be considered efficient given its output level. This LP model can also be written as:

$$\begin{gathered} TE = Min_{\lambda,z}\lambda \\ s.t. \\ y^j \leq zY \\ zX \leq \lambda x_j \\ z \in \Re_+^J \end{gathered} \quad (1.2)$$

In order to put model 1.2 into a form that can be solved in R, we transpose the matrices, bring the endogenous variable to the left hand side (LHS), and re-arrange some other terms giving:

$$\begin{gathered} TE = Min_{\lambda,z}\lambda \\ s.t. \\ Y'z \geq y_j \\ X'z - \lambda x_j \leq 0 \\ z \in \Re_j \end{gathered} \quad (1.3)$$

In matrix form, the LP model is shown below:

$$\begin{gathered} Min\ (\underline{0}' \quad 1)\begin{pmatrix} z \\ \lambda \end{pmatrix} \\ s.t. \\ \begin{pmatrix} Y' & \underline{0} \\ X' & -\underline{x}_j \end{pmatrix}\begin{pmatrix} z \\ \lambda \end{pmatrix}\begin{pmatrix} \geq \\ \leq \end{pmatrix}\begin{pmatrix} y_j \\ \underline{0} \end{pmatrix} \\ z \geq 0 \end{gathered} \quad (1.4)$$

This is equivalent to the linear programming problem $Min_x F(x) = cx, subject\ to\ Ax \leq b$.

R Code

The R program to calculate this model is shown in Appendix 1 (Program 1). The first step taken in the program is to clear all values that might be in memory from previous versions, and to read in the data (lines 1-26). We first set the working directory (line 13) and then read in an external comma separated value (csv) file that is stored in that directory (line 14). Line 15 lists out the names of the columns from the .csv file that will be used in lines 19 and 21. In order to test our code, and to provide intermediate results for this example that can be looked at easily, we limit our model to only include the first 10 observations (line 17). Next, the matrix containing the inputs (X) is created on line 19, and the output (Y) matrix is created on line 21. Note for this example, we are only using a single output. Three variables M, N, and J are then created which hold the number of columns in the Y and X matrices (M&N respectively, lines 23-24), and the number of observations (J, line 25). The Y and X matrices are then combined to form a new

matrix, YX (line 26). This is the end of the data step, and this same code can be now used across all DEA programs.

The next step is to create the A matrix for this particular problem. The A matrix is initially created on line 30, and its values are set to zero. Note that the dimension of the A matrix is M (#outputs) + N(#inputs) rows, and J+1(#observations+1) columns, and populated with zeros. Values are then placed in the A matrix from the YX matrix using the transpose operator (line 33). The transpose operator (tr) takes the 10 by 5 YX matrix (Figure 1) and fits it into the 5 by 11 A matrix (Figure 2). The first row of the YX matrix is now the first column on the A matrix.

Next, the LP model is set up. First, the problem type is declared to be a minimization (line 38). The objective function coefficients are set up in a vector that has dimension J, and is filled with zeros, and then a single value of 1 (line 39). The (in)equality conditions for the inputs and outputs are set in line 41, and then two values (nr and nc) are initialized based on the number of rows and columns in the A matrix, respectively (lines 43-44). Control values for the lpSolveAPI package are set in lines 47-48, and the values for the rows in the LP problem are set in lines 50-52. Lines 53-54 set up variables that will hold the results from each run of the DEA model.

A DEA model is solved once for each observation in the data, and is easily accomplished using a looping construct (lines 56-68). The first step in the loop sets the values in the last column of the A matrix, in the rows corresponding to the input values, equal to the negative values of the inputs from the A matrix, column j (line 57). This corresponds to the matrix representation of the problem shown in equation 1.4. The final A matrix when solving the DEA model for observation #1 is shown in Figure 3. Note, the negative values in column 11

correspond to the values in column one. Line 58 sets the right hand side values in a vector named rhs, and then passes that value to the solver in line 59. Line 60 sets the column values for the solver and line 61 passes the objective function values to the solver. Line 64 solves the model and passes the value for the solver status to the vector status, with a returned value of zero indicating that the model solved optimally. This is important in order to check whether the model solved at each iteration, or whether there were infeasibilities. The objective function value is returned to the vector objvals (line 65), and these values are printed out on line 71. For larger problems, another type of data structure may be needed to either hold, or print out the objective function values to another application, such as a spreadsheet. Finally, line 67 prints out to the screen which DMU is being solved so that for large problems it will be clear where the DEA program is in terms of all the observations, and line 68 closes the for loop.

Data for 10 observations are shown in Appendix 2 (Figure 8), along with the results from the input oriented CRS model (Figure 9). These are provided so that the interested reader can check results from their DEA code written in R against these results. All results are rounded to three decimal places.

1.1 Input oriented TE with VRS.

A variable returns to scale model requires a slight modification to the prior program, with the addition of a convexity constraint as shown below:

$$\begin{gathered} TE = Min_{\lambda,z}\lambda \\ s.t. \\ Y'z \geq y_j \\ X'z - \lambda x_j \leq 0 \\ J1'z = 1 \\ z \geq 0 \end{gathered} \tag{1.5}$$

In equation 1.5, $_{J1'}$ is a Jx1 vector of ones. In matrix form, this model would be specified as:

$$\begin{gathered} Min\ (\underline{0}' \quad 1)\begin{pmatrix}\underline{z}\\ \lambda\end{pmatrix} \\ s.t. \\ \begin{pmatrix} Y' & \underline{0} \\ X' & -\underline{x}_j \\ \underline{1}' & \underline{0}\end{pmatrix}\begin{pmatrix}\underline{z}\\ \lambda\end{pmatrix}\begin{pmatrix}\geq\\ \leq\\ =\end{pmatrix}\begin{pmatrix}\underline{y}_j\\ 0\\ 1\end{pmatrix} \\ z \geq 0 \end{gathered} \tag{1.6}$$

As can be seen from model 1.6, the VRS problem requires an additional row in the A matrix. The modification of the prior CRS model in R is shown in program 2, beginning on line 75. Here, we only present the changes from the prior R code, beginning with the reformulation of the A matrix, as the data input steps remain the same. Lines 75-80 create the A matrix, populates the cells in the matrix with the transformed YX, and adds a row of ones in order to incorporate VRS (line 80). When the LP model is constructed in the next section, and additional equality constraint is inserted into the character "rest" (line 84). The rhs value also needs to be modified for each pass through the "for" loop with a value of one inserted into the rhs matrix (line 101). Solving the model and collecting the objective function values is accomplished in lines 106-107. This completes the modifications needed for the VRS model. The A matrix for this problem is shown in figure 4.

Results for the 10 observation data set are shown in Figure 9 (Appendix 2). With this model, most observations are considered efficient, with a value of 1.000 returned for seven observations. This was expected given that VRS was imposed and the limited number of observations.

2.0 Output oriented technical efficiency with variable returns to scale (VRS)

An output oriented DEA model expands output for each DMU while holding the level of inputs fixed, and is estimated through the following L.P. problem, with vrs imposed:

$$\begin{gathered} TE = Max_{\theta,z}\theta \\ s.t. \\ Y'z - \theta y \geq 0 \\ X'z \leq x_j \\ J1'z = 1 \\ z \geq 0 \end{gathered} \tag{2.1}$$

This problem differs from the input oriented model in that output is being expanded through the term θ, and it is a maximization problem, rather than a minimization problem. In matrix form, this model would be:

$$\begin{gathered} Max\ (\underline{0}' \quad 1)\begin{pmatrix} z \\ \theta \end{pmatrix} \\ s.t. \\ \begin{pmatrix} Y' & -\underline{y}_j \\ X' & 0 \\ \underline{1}' & 0 \end{pmatrix}\begin{pmatrix} z \\ \theta \end{pmatrix}\begin{pmatrix} \geq \\ \leq \\ = \end{pmatrix}\begin{pmatrix} 0 \\ \underline{x}_j \\ 1 \end{pmatrix} \\ z \geq 0 \end{gathered} \tag{2.2}$$

The R code for this output oriented program is shown in program 3 (Appendix 1). Reading in the data and setting up the initial A matrix is done no differently than in programs 1 and 2. Since this is a maximization problem, the objective function type is set to "max" in line 121. The last column of the A matrix is set differently than in the input oriented program because this is a maximization problem and the optimization is based on the M outputs. This is seen in line 141, where the last column of the A matrix is set equal to the negative values from rows one to M of observation j of the A matrix. The A matrix for the solving the first DEA problem for the first observation is shown in Figure 5. The right hand side is set equal to the zero for the outputs,

and to the value of the inputs for the inputs, and to one for the VRS constraint (line 142). The remaining part of the program is the same as programs one and two.

Results for the output oriented model with VRS imposed are shown in Figure nine. Note that in the output oriented example, all the results are greater or equal to one.

2.1 The Johansen Capacity Model

The Johansen capacity model has been used extensively over the past two decades to estimate a physical measure of capacity and capacity utilization. It was operationalized by (Fare, et al., 1989), and is a variation of the output oriented TE measure shown in the previous section. The Johansen model is:

$$\begin{gathered} Cap = Max_{z,x_v,\theta}\theta \\ s.t. \\ Y'z - \theta y \geq 0 \\ X_F'z \leq x_{jF} \\ X_V'z - x_v = 0 \\ J1'z = 1 \\ z \geq 0 \end{gathered} \tag{2.3}$$

In the above capacity model, the inputs are partitioned into two constraints, one for the fixed factors (F) and the second for the variable factors (V). The fixed input constraint is the same as found in the output oriented TE model shown in the previous section. For the variable input constraint, x_v is the optimal level of variable inputs and is returned by the model. This constraint ensures that the variable inputs do not constrain output, rather the fixed factors do so, which is consistent with the Johansen definition of capacity. The matrix form of the above set of equations is:

$$
\begin{gathered}
Max\ (\underline{0}' \quad \underline{0}' \quad 1)\begin{pmatrix} \underline{z} \\ \underline{x}_v \\ \theta \end{pmatrix} \\
s.t. \\
\begin{pmatrix} Y' & \underline{0} & -\underline{y}_j \\ X_F' & 0 & 0 \\ X_v' & -1' & 0 \\ \underline{1}' & \underline{0} & 0 \end{pmatrix}\begin{pmatrix} \underline{z} \\ \underline{x}_v \\ \theta \end{pmatrix}\begin{pmatrix} \geq \\ \leq \\ = \\ = \end{pmatrix}\begin{pmatrix} 0 \\ \underline{x}_0^F \\ 0 \\ 1 \end{pmatrix} \\
z \geq 0
\end{gathered}
\tag{2.4}
$$

As model 2.4 shows, the A matrix is now one row and one column greater (4 x 3) than the previous output oriented TE model (3 x 2). There is also an additional row in the right hand side to accommodate the variable input, and one in the vector describing the constraints. The R code to estimate the Johansen capacity model is shown in Program 4 (Appendix 1). Unlike the prior output oriented TE model, this program requires some additional data steps, so we present the program in its entirety.

Data are read into the R program in the same manner as previous programs using "read_csv" (line 174), followed by creation of the X matrix. A vector "varindex" is then created (line 179) that contains four elements. The first two elements have the value of zero that corresponds to the fixed factors, while the final two elements have the value one, corresponding to the variable inputs. Two more vectors are created, "var3" and "var4" (line 180-181) that hold values of zero for all elements except for a single value of one, corresponding to the position of one of the variable inputs in the X matrix. Skipping to line 188, the value NV is created, that is the sum of all elements in the vector "varindex", which here is equal to two. Next, a value "NCX" is established, that is the difference between the total number of inputs (N) and the number of variable inputs NV (i.e. N-NV, line 189). The data step ends on line 190 where the YX matrix is created.

The initial A matrix that is created contains two more columns than the same A matrix created for the output oriented TE program because there are two variable inputs for this problem (line 194). This is done on line 194 through the use of the variable NV created on line 188, and allows more than two extra columns to be added if required. The optimal levels of the variable inputs are calculated endogenously through solving the DEA model, which means they need to have negative values in the A matrix and zero values in the right hand side constraint. The values from the YX matrix are then inserted into the A matrix using the transpose operator (line 195), and the value "1" is inserted into the last row of the A matrix for the VRS constraint (line 196).

The objective function coefficients are set in line 200, by setting the value "0" in columns 1 through J+NV, and then inserting 1 into the last row. This conforms to the objective function shown in equation 2.4. The signs for the constraints ("rest") are initially set in line 201, and then redefined in line 202 for the inputs using an "ifelse" construct. This line sets the constraint such that if the value in "varindex" for the column in question is equal to "1" the constraint sign is set to an equality (=), else it is set as an inequality (<=). Lines 215-216 initialize two variables that will hold the values of variable inputs returned by the model for each observation.

Line 218-235 set values for each iteration of the loop that solves the DEA problem. First, the A matrix is reconfigured. Lines 219 and 220 set the value for columns J+1 and J+2 to be equal to negative one for either var3 or var4. Only one value in the column will be equal to negative one. Then, the negative value of the outputs is placed in column J+3 for observation j (line 221). The A matrix for observation one in our data is shown in Figure 6. The right hand side (rhs) values are set in line 222, with zero being inserted for the outputs, and the variable inputs being assigned the value zero using the "varindex" vector. Since "varindex" is equal to zero for the fixed inputs, they are set equal to their observed values in the rhs vector. The column

values for the LP model are then set in lines 224-226, and the objective function is set in line 227. Line 230 stores the objective function values, while the values for the variable inputs returned by the model are stored in var1 and var2 in lines 232 and 233. Line 237 combines all the values returned by the DEA model into one data frame (results), which is then printed out on line 238.

Results for the value of theta returned by the Johansen capacity model with VRS are shown in Figure nine (Appendix 2). Since the capacity model is less constrained than the output oriented TE model (VRS), the Johansen value will naturally be greater, or equal to the output oriented TE value. Additionally, figure 10 reports the values that are returned for the optimal level of the variable inputs needed to reach capacity output. These figures are important because they tell the firms how their use of variable inputs will need to be changed in order to reach capacity output. Note that in some instances (observation #2) less variable inputs are needed.

3.0 The directional distance function (DDF) model

The directional distance function (DDF) is a more generalized version of the previous distance function models used to calculate input and output oriented TE shown in prior sections (Färe and Grosskopf, 2006). It offers flexibility for a wide range of problems because the directional vector can easily be set to different values, allowing contraction or expansion of specific outputs and inputs. It has proved particularly useful for modelling environmental problems where pollution can be treated as an "undesirable" output and efficiency can then be measured in terms of increasing the "desirable" outputs, while simultaneously contracting the "undesirable" outputs. (Färe, et al., 2005, Färe, et al., 2011, Pasurka, 2006). Below, we present a DDF model which expands outputs, while simultaneously contracting inputs, along with the associated R code. We then modify the program by splitting the outputs into either "desirable" or

"undesirable" and setting up the model so that it expands "desirable" outputs, while contracting "undesirable" outputs.

The directional distance function model with VRS, which expands outputs while simultaneously contracting inputs is constructed as:

$$\begin{gathered} Max_{z,\beta}\,\beta \\ s.t. \\ Y'z \geq y^j + \beta g_y \\ X'z \leq x^j - \beta g_x \\ J1'z = 1 \end{gathered} \tag{3.1}$$

After re-arranging terms, the following formulation is equivalent:

$$\begin{gathered} Max_{z,\beta}\,\beta \\ s.t. \\ Y'z - \beta g_y \geq y^j \\ X'z + \beta g_x \leq x^j \\ J1'z = 1 \end{gathered} \tag{3.2}$$

In matrix form, model 3.2 is constructed as:

$$\begin{gathered} Max\ (\underline{0}' \quad 1)\begin{pmatrix} z \\ \beta \end{pmatrix} \\ s.t. \\ \begin{pmatrix} Y' & -\underline{g}_y \\ X' & \underline{g}_x \\ \underline{1}' & 0 \end{pmatrix}\begin{pmatrix} \underline{z} \\ \beta \end{pmatrix}\begin{pmatrix} \geq \\ \leq \\ = \end{pmatrix}\begin{pmatrix} \underline{y}_j \\ \underline{x}_j \\ 1 \end{pmatrix} \\ z \geq 0 \end{gathered} \tag{3.3}$$

In all three models, the variables (g_y, g_x) are directional vectors, and a value needs to be chosen for each. For this example, we will set the values for the directional vectors equal to the observed values for each observation. This results in a multiplicative model where outputs are expanded by (1+β) and inputs contracted by (1-β). Other values for the directional vector, such

as (1,-1) could also have been chosen. A value of zero returned by the model indicates that the observation is efficient and no expansion of outputs or contraction of inputs can take place. A value greater than zero, for example 0.1, means that the firm is 90% efficient (1-β) and can expand outputs and contract inputs by 10%. Note that β is bounded above at one, since a value greater than one would contract inputs below zero.

The R program needed to estimate model 3.3 is in Appendix 1 (program 5), and uses the same steps as shown in program 3 to read in the data and set up the initial A matrix. The difference between this program and previous programs begins on line 246, where the first loop of the DEA model is solved. Line 246 defines a matrix, "dy", that holds the values of the outputs, and line 247 defines a matrix, "dx" to hold the values of the inputs. Because beta (β) is being used to both expand outputs, and contract inputs, it needs to act on both the outputs and inputs in the A matrix. These matrices are then inserted into the last column of the A matrix in line 248, with the values for dy being set to their negative, thus insuring the proper sign. The right hand side (rhs) value is then set to the values in the YX matrix, and the value one for the VRS constraint. This differs from the usual output (input) oriented TE model where the rhs values for the outputs (Y) (inputs, X) are set equal to zero. The program is then solved in line 255, and the objective values are stored in line 256. As stated previously, in order to calculate the measure of TE, the objective function values need to be subtracted from one.

Results for our 10 observation data set are shown in figure nine (Appendix 2). The majority of values returned by the model are zero, meaning that outputs can't be expanded while simultaneously contracting inputs. For those observations with values greater than one, the value yields the percentage that outputs should be expanded and inputs contracted to be placed on the efficient frontier. For example, observation two returned a value of .046. This means that in

order to be considered efficient, our output should be 1.046 times the observed output, and inputs should be 0.954 times the observed inputs (1-.046).

3.1 The DDF model with desirable and undesirable outputs.

One application for DDF models which has emerged in recent years have been models to estimate efficiency and other metrics for polluting industries (Färe, et al., 1989, Färe, et al., 2005, Färe, et al., 2006, Pasurka, 2006). In this type of DDF model, inputs are generally held constant and the model is constructed so as to measure efficiency in terms of expanding desirable outputs, and contracting undesirable outputs. The polluting output is considered an "undesirable" output. For example, an electric generating plant produces electricity, but also emissions such as sulfur dioxide, which is considered polluting, or "undesirable."

In order to estimate TE with both desirable and undesirable outputs, the outputs need to be separated into a vector of desirable (y) and undesirable outputs (u). Similar to the first ddf model presented above, the model expands desirable outputs, while contracting undesirable outputs, given the inputs. The ddf model for estimating TE is:

$$\begin{gathered} Max_{z,\beta}\beta \\ s.t. \\ Y'z - \beta g_y \geq y^j \\ U'z + \beta g_u = u_j \\ X'z \leq x^j \\ J1'z = 1 \end{gathered} \tag{3.4}$$

In matrix form, the model is:

$$
\begin{gathered}
Max\ (\underline{0}' \quad 1)\begin{pmatrix} z \\ \beta \end{pmatrix} \\
s.t. \\
\begin{pmatrix} Y' & -g_y \\ U' & \underline{g_u} \\ X' & 0 \\ \underline{1}' & 0 \end{pmatrix}\begin{pmatrix} \underline{z} \\ \beta \end{pmatrix}\begin{pmatrix} \geq \\ = \\ \leq \\ = \end{pmatrix}\begin{pmatrix} \underline{y}_j \\ \underline{u}_j \\ x_j \\ \underline{1} \end{pmatrix} \\
z \geq 0
\end{gathered}
\tag{3.5}
$$

The R code to implement the ddf model for a 10 observation problem is found in program six (Appendix 1), and we present the program in its entirety. A different data set is used for this model because the data needs to contain both desirable and undesirable outputs. In this example, there are two desirable outputs (Y1,Y2), one undesirable output (U1), and four inputs (FX1, FX2, VX1,VX2). The data are read in on line 279 from a .csv file, and the X matrix is created with the data on inputs (line 283). Next, two Y matrices are created, one which holds the desirable outputs (YD), and the other which holds the undesirable (YU) outputs (lines 285-286). These are then joined together to create the single output matrix Y (line 287). Lines 289-90 create two variables, MD and MU that contain the number of desirable outputs (MD) and undesirable outputs (MU) in the YD and YU matrices (two and one, respectively). Counters for the number of columns in the full Y and X matrices, along with the number of rows (observations) in X are created in lines 291-293. Finally, Y and X are joined together to form one YX matrix (line 294).

Construction of the initial A matrix is consistent with the way it was constructed in all other programs (lines 298-300). Line 303 constructs the constraint vector, with the sign for the desirable outputs being set to ">=", the undesirable output constraint is set to "=" to reflect that disposability is costly, the input constraint is set to "<=", while the vrs constraint is set to "=".

The remainder of the commands setting up the LP problem are found in lines 304-312. These commands did not differ from prior models presented above.

The DDF model is solved in lines 330-335. The last column of the A matrix for each observation is set in lines 321-323. First, two matrices are initialized, one for the desirable outputs (dyd) and one for the undesirable outputs (dyu) (321-322). Column J+1 of the A matrix is then set to the negative vector of the desirable outputs and the positive vector of the undesirable outputs (line 323). This corresponds to the construction of the A matrix found in model 3.5 above. Figure 7 shows the A matrix used for solving the DDF program for the first observation.

The rhs values for this model are equal to the observed values for each observation (line 325). Lines 326-328 set values for the LP solver to use, and the model is solved in line 330. The objective function value is passed to the variable objvals1 in order to be printed later. The loop is closed in line 335 and the objective function values are printed in line 337. Note that the TE value will be equal to 1-objvals1.

4.0 Bootstrapping DEA in R

DEA models assume that all deviations from the production frontier are due to inefficiency, rather than noise or random deviations. This has led to a great deal of research to explore the statistical properties of the DEA estimator (Fried, et al., 2008, Simar, 1996, Simar and Wilson, 2000). Exploring all of the literature surrounding this topic is well beyond the scope of this paper, but one technique that has gained popularity for exploring the statistical properties of estimators resulting from DEA models is the bootstrap (Simar and Wilson, 2000). Below we present the R code for bootstrapping the output oriented TE model using a subsample bootstrap

procedure (Geyer, 2006). This is a straightforward approach to implement, and allows one to construct bias corrected mean values, along with confidence intervals for the DEA estimates.

Program seven in Appendix 1 begins after solving the initial output oriented DEA model, where the results are stored in the dataframe objvals2 (line 342). The initial LP model is then deleted on the next line, and the bootstrap portion of the program begins on line 347. The first step in this portion of the program is to set a seed for the random draws using the "set.seed" command (line347). Next the value "n" is set equal to the number of observations in the data, a value "m" is initialized that will determine the size of the subsample bootstrap, and the variable B is set to 250 which is the number of bootstrap replicates (lines 348-350). Next, three new matrices are defined (lines 354-357). The first, "statusB" is designed to hold status codes from each bootstrap iteration and observation that tells whether the model solved. The second, "boot" will hold the objective function values for each bootstrap iteration and observation. The final matrix, "AB" is meant to hold the subsample of observations that are used for each bootstrap iteration. Note that the number of columns in the AB matrix is equal to the number of observations in the bootstrap sample (m) plus one (line 357).

The next set of commands (lines 358-364) set up the LP problem to be solved in LP_API, and the actual bootstrap routine begins on line 366. Line 367 defines a variable "pickval" that is a vector of random integers of size "m" from the number of observations in the data (J). This will change for each iteration through the B loop. The AB matrix is populated on line 368 with the values from the A matrix that correspond to observations found in "pickval". Recall that the A matrix was defined in the first part of the program to hold all the data from the transposed YX matrix. Lines 370-372 sets the rows for the LP problem using the set.row command.

Lines 374-384 run the LP model for each observation in the data. The first column in the AB matrix is set equal to the value of "s" which changes for each pass through the loop. This ensures that each observation in the data set will be evaluated against the observations contained in "pickval." Note that "pickval" changes each time through the "B" loop as a new sample is drawn from the observations. Because this is an output oriented model, the outputs for column m+1 in the AB matrix are set equal to the negative of the output values found in column one (line 376). The right-hand side for the LP model is set in line 377, and the rest of the LP_B objects are set in lines 378-381, including the objective function. The LP model is solved (line 382), and the objective function values are collected in the matrix "boot" (line 383). Line 384 ends the loop for "s" which is all the observations in the data, while the "B" loop is closed at line 388. Line 387 sets up a counter so the user knows what bootstrap iteration they are on.

In order to calculate the bias corrected scores, the returns to scale (RTS) needs to be set (line 389). The value for the convergence rate "beta" used in the bias correction is set depending on whether the model is variable returns to scale (VRS), or constant returns to scale (CRS; lines 391-392). Line 393 defines a matrix "S" to hold all the bias-corrected scores, while lines 396-400 print out various statistics for the bias corrected scores.

References

Charnes, A., W.W. Cooper, and E. Rhodes. 1978. "Measuring the efficiency of decision making units." *European Journal of Operational Research* 2:429-444.

Daraio, C., K. Kerstens, T. Nepomuceno, and R.C. Sickles. 2019. "Empirical surveys of frontier applications: a meta-review." *International Transactions in Operational Research*.

Färe, R., and S. Grosskopf. 2006. *New directions: efficiency and productivity*: Springer Science & Business Media.

Fare, R., S. Grosskopf, and E.C. Kokkelenberg. 1989. "Measuring plant capacity, utilization and technical change: a nonparametric approach." *International Economic Review*:655-666.

Fare, R., S. Grosskopf, and C.K. Lovell. 1994. *Production frontiers*: Cambridge University Press.

Färe, R., S. Grosskopf, C.K. Lovell, and C. Pasurka. 1989. "Multilateral productivity comparisons when some outputs are undesirable: a nonparametric approach." *The review of Economics and Statistics*:90-98.

Färe, R., S. Grosskopf, D.-W. Noh, and W. Weber. 2005. "Characteristics of a polluting technology: theory and practice." *Journal of econometrics* 126:469-492.

Färe, R., J.E. Kirkley, and J.B. Walden. 2006. "Adjusting technical efficiency to reflect discarding: The case of the US Georges Bank multi-species otter trawl fishery." *Fisheries Research* 78:257-265.

---. 2011. "Measuring Fishing Capacity When Some Outputs Are Undesirable." *Eastern Economic Journal* 37:553-570.

Fried, H.O., C.K. Lovell, and S.S. Schmidt. 2008. *The measurement of productive efficiency and productivity growth*: Oxford University Press.

Geyer, C.J. 2006. "5601 notes: The subsampling bootstrap." *Unpublished manuscript*.

Pasurka, C.A. 2006. "Decomposing electric power plant emissions within a joint production framework." *Energy Economics* 28:26-43.

Simar, L. 1996. "Aspects of statistical analysis in DEA-type frontier models." *Journal of Productivity Analysis* 7:177-185.

Simar, L., and P.W. Wilson. 2000. "A general methodology for bootstrapping in non-parametric frontier models." *Journal of Applied Statistics* 27:779-802.

Simar, L., and P.W. Wilson. 2000. "Statistical inference in nonparametric frontier models: The state of the art." *Journal of Productivity Analysis* 13:49-78.

Appendix 1. DEA programs written in R

Program 1. Input oriented TE with CRS

```
#First Clear any previous data stored in memory, and require lpSolveAPI and readr
rm(list=ls())
PKG <- c("lpSolveAPI", "readr")
for (p in PKG) {
  if(!require(p,character.only = TRUE)) {
    install.packages(p)
    require(p,character.only = TRUE)}
}
#########################################################################
#Beginning of Data Step
#########################################################################
setwd("~/John/joe/manual")  #Set working directory to where data is stored.
df0=read_csv("data1.csv")
names(df0)
#########################################################################
df0=df0[1:10,]
# create input X matrix
X=df0[,c("FX1","FX2","VX1","VX2")]
# create output Y matrix
Y=df0[,c("Y1")]
#########################################################################
M=ncol(Y)
N=ncol(X)
J=nrow(X)
YX=cbind(Y,X)
#########################################################################
#End of DATA Step
#########################################################################
A=matrix(0,M+N,J+1)
#M+N is the number of inputs and outputs, J+1 sets the number of
#columns equal to the the number of observations +1
A[1:(M+N),1:J]=t(YX)
#t is transpose operator, which turns rows into columns
#########################################################################
#Next, set up LP Model
#########################################################################
objtype='min'
```

```
obj=c(rep(0,J),1)
#rest defines the constraints
rest=c(rep('>=',M),rep('<=',N))
#nr is the number of rows, nc is the number of columns
nr=nrow(A)
nc=ncol(A)
#LP_API is the name chosen for the LP problem
LP_API=make.lp(nrow=nr,ncol=nc)
lp.control(LP_API,sense='min')
set.constr.type(LP_API,rest)

for(i in 1:nr){
  set.row(LP_API,i,A[i,])   #setting up rows by reading in A matrix rows
}
objvals=0
status=0
#Begin loop for DEA Program
for(j in 1:J){
  A[(M+1):(M+N),J+1]= -A[(M+1):(M+N),j] #column J+1 in A is being set to -obs j input data
  rhs=c(as.matrix(Y[j,]),rep(0,N)) #rhs is being set to obs j output data, and then zero for inputs
  set.rhs(LP_API,rhs)
  set.column(LP_API,nc,A[,nc])
  set.objfn(LP_API,obj)
#status[j] stores the solver value in case there is any question whether the program solved for
#a specific loop. Objvals[j] stores the objective function value
  status[j]=solve(LP_API)
  objvals[j]=get.objective(LP_API)

  if(j%%100==0|j==J)  print(paste('on dmu',j,'of',J)) #counter to show where we are in loop
}
# end loop   for(j in 1;J)

print(objvals)

```

Program 2. Input Oriented TE Model with VRS

```
#Define A Matrix. M+N+1 rows allows for VRS.
A=matrix(0,M+N+1,J+1)
#Next, Transform YX matrix and copy to A.
A[1:(M+N),1:J]=t(YX)
#Now, set the last row in the A matrix equal to 1, for all J values
#This is for VRS.
A[M+N+1,1:J]=1.0
#####################################################################
#set zero for all J columns, plus 1 for last columns
obj=c(rep(0,J),1)
rest=c(rep('>=',M),rep('<=',N),'=')
#####################################################################
nr=nrow(A)
nc=ncol(A)
LP_API=make.lp(nrow=nr,ncol=nc)
lp.control(LP_API,sense='min')
set.objfn(LP_API,obj)
set.constr.type(LP_API,rest)
for(i in 1:nr){
  set.row(LP_API,i,A[i,])   #setting up rows by reading in A matrix rows
}
#####################################################################
objvals2=0
status2=0
for(j in 1:J){
  A[(M+1):(M+N),J+1]=-A[(M+1):(M+N),j]
   #column J+1 in A is being set to the negative of obs j input data
  rhs=c(as.matrix(Y[j,]),rep(0,N),1)
```

```
    #rhs is being set to obs j output data, and zero for input data, and 1 for VRS
    set.rhs(LP_API,rhs)
    set.column(LP_API,nc,A[,nc]) #loading revised input data into LPApi matrix
    set.objfn(LP_API,obj)
    status2[j]=solve(LP_API)
    objvals2[j]=get.objective(LP_API)
    if(j%%100==0|j==J)  print(paste('on dmu',j,'of',J))
  }
print(objvals2)
##############################################################################
```

Program 3. Output Oriented TE Model with VRS

```
#Define A Matrix. M+N+1 rows allows for VRS.
A=matrix(0,M+N+1,J+1)
#Next, Transform YX matrix and copy to A.
A[1:(M+N),1:J]=t(YX)
#Now, set the last row in the A matrix equal to 1, for all J values
#This is for VRS.
A[M+N+1,1:J]=1.0
######################################################################
objtype='max'
#set zero for all J colums, plus 1 for last column
obj=c(rep(0,J),1)
rest=c(rep('>=',M),rep('<=',N),'=')
######################################################################
nr=nrow(A)
nc=ncol(A)
LP_API=make.lp(nrow=nr,ncol=nc)
lp.control(LP_API,sense=objtype)

set.objfn(LP_API,obj)
set.constr.type(LP_API,rest)

for(i in 1:nr){
   set.row(LP_API,i,A[i,])   #setting up rows by reading in A matrix rows
}

objvals2=0
status2=0
```

```
for(j in 1:J){
  A[1:M,J+1]=-A[1:M,j]
  rhs=c(rep(0,M),as.matrix(X[j,]),1)
  #rhs is being set to obs j output data, and J input data from X0, and 1 for VRS
  set.rhs(LP_API,rhs)
  set.column(LP_API,nc,A[,nc]) #loading revised input data into LPApi matrix
  set.objfn(LP_API,obj)

  status2[j]=solve(LP_API)
  objvals2[j]=get.objective(LP_API)

  if(j%%100==0|j==J)  print(paste('on dmu',j,'of',J))

} # end loop   for(j in 1:J)

print(objvals2)
```

Program 4. The Johansen Capacity Model

```
#########################################################################
#R Program to calculate Johansen capacity model
#Model is taken from "Production Frontiers" (1994) by Fare, Grosskopf and Lovell
#########################################################################
#First Clear any previous data stored in memory, and require lpSolveAPI and readr
rm(list=ls())
PKG <- c("lpSolveAPI", "readr")
for (p in PKG) {
  if(!require(p,character.only = TRUE)) {
    install.packages(p)
    require(p,character.only = TRUE)}
}
#########################################################################
#Beginning of Data Step
#########################################################################
setwd("~/John/joe/manual")  #Set working directory to where data is stored.
df0=read_csv("data1.csv")
#########################################################################
df0=df0[1:10,]
# create input X matrix
X=df0[,c("FX1","FX2","VX1","VX2")]
varindex=c(0,0,1,1)
var3=c(0,0,1,0)
var4=c(0,0,0,1)
# create output Y matrix
Y=df0[,c("Y1")]
#########################################################################
(M=ncol(Y))
(N=ncol(X))
```

```
(J=nrow(X))
(NV=sum(varindex))
(NCX=N-NV)
YX=cbind(Y,X)
#########################################################################
#End of DATA Step
#########################################################################
A=matrix(0,M+N+1,J+1+NV)
A[1:(M+N),1:J]=t(YX)
A[M+N+1,1:J]=1.0
#########################################################################
nr=nrow(A)
nc=ncol(A)
obj=c(rep(0,J+NV),1)
rest=c(rep('>=',M),rep('<=',N),'=')
rest[(M+1):(M+N)]=ifelse(varindex==1,'=','<=')

LP_API=make.lp(nrow=nr,ncol=nc)
lp.control(LP_API,sense='max')
set.objfn(LP_API,obj)

for(i in 1:nr){
  set.row(LP_API,i,A[i,])
}
set.constr.type(LP_API,rest)

status2=0
objvals2=0
var1=0
var2=0
```

```
217  ###################################################################
218  for(j in 1:J){
219    A[(M+1):(M+N),J+1]=as.matrix(-var3)
220    A[(M+1):(M+N),J+2]=as.matrix(-var4)
221    A[1:M,J+3]=-A[1:M,j]
222    rhs=c(rep(0,M),(1-varindex)*as.matrix(X[j,]),1)
223    set.rhs(LP_API,rhs)
224    set.column(LP_API,J+1,A[,J+1])
225    set.column(LP_API,J+2,A[,J+2])
226    set.column(LP_API,J+3,A[,J+3])
227    set.objfn(LP_API,obj)
228
229    status2[j]=solve(LP_API)
230    objvals2[j]=get.objective(LP_API)
231    z=get.variables(LP_API)
232    var1[j]=z[J+1]
233    var2[j]=z[J+2]
234    if(j%%100==0|j==J)  print(paste('on dmu',j,'of',J))
235  } # end loop   for(j in 1;J)
236
237  results<-cbind(status2,objvals2,var1,var2)
238  print(results)
239
```

Program 5. The Directional Distance Function (DDF) Model

```
# run DEA loop
objvals1=0
status1=0

for(j in 1:J){
  (dy=as.matrix(Y[j,]))
  (dx=as.matrix(X[j,]))
  A[1:(M+N),J+1]=c(-dy,dx)

  rhs=c(as.matrix(YX[j,]),1)
  set.rhs(LP_API,rhs)
  set.column(LP_API,nc,A[,nc])
  set.objfn(LP_API,obj)

  status1[j]=solve(LP_API)
  objvals1[j]=get.objective(LP_API)

  if(j%%100==0|j==J)  print(paste('on dmu',j,'of',J))
} # end loop   for(j in 1;J)
print(objvals1)
```

Program 6. DDF model with desirable and undesirable outputs

```
#########################################################################
#R Program to calculate directional distance function model with desirable and undesirable outputs
#Model is taken from "New Directions: Efficiency and Productivity" (2004) by Fare and Grosskopf
#########################################################################
#First Clear any previous data stored in memory, and require lpSolveAPI and readr
rm(list=ls())
PKG <- c("lpSolveAPI", "readr")
for (p in PKG) {
  if(!require(p,character.only = TRUE)) {
    install.packages(p)
    require(p,character.only = TRUE)}
}
#########################################################################
#Beginning of Data Step
#########################################################################
setwd("~/John/joe/manual")  #Set working directory to where data is stored.
df0=read_csv("ddf_example.csv")
#########################################################################
df0=df0[1:10,]
# create input X matrix
X=df0[,c("FX1","FX2","VX1","VX2")]
# create output Y matrix
YD=df0[,c("Y1","Y2")]       #Desireable Outputs
YU=df0[,c("U1")]           #Undesirable Outputs
Y=cbind(YD,YU)              #column bind YD and YU
#########################################################################
(MD=ncol(YD))
(MU=ncol(YU))
```

```
(M=ncol(Y))
(N=ncol(X))
(J=nrow(X))
YX=cbind(Y,X)
#########################################################################
#End of DATA Step
#########################################################################
A=matrix(0,M+N+1,J+1)
A[1:(M+N),1:J]=t(YX)
(A[(M+N+1),1:J]=rep(1,J))
##End of A Matrix build################################################
obj=c(rep(0,J),1)
rest=c(rep('>=',MD),rep('=',MU), rep('<=',N),'=')
nr=nrow(A)
nc=ncol(A)
LP_API=make.lp(nrow=nr,ncol=nc)
lp.control(LP_API,sense='max')
set.objfn(LP_API,obj)

for(i in 1:nr){
  set.row(LP_API,i,A[i,])
}

set.constr.type(LP_API,rest)
#########################################################################
# run loop
objvals1=0
status1=0
```

```
319
320   for(j in 1:J){
321     dyd=as.matrix(YD[j,])
322     dyu=as.matrix(YU[j,])
323     A[1:M,J+1]=c(-dyd,dyu)
324
325     rhs=c(as.matrix(Y[j,]),as.matrix(X[j,]),1)
326     set.rhs(LP_API,rhs)
327     set.column(LP_API,nc,A[,nc])
328     set.objfn(LP_API,obj)
329
330     status1[j]=solve(LP_API)
331     objvals1[j]=get.objective(LP_API)
332
333     if(j%%100==0|j==J)  print(paste('on dmu',j,'of',J))
334
335   } # end loop   for(j in 1;J)
336
337   print(objvals1)
338
```

339 Program 7. Bootstrap Output Oriented TE Model

```
340 #First, run initial output oriented DEA program and save results, then
341 #delete the initial lp model – LP_API
342 objvals2[j]=get.objective(LP_API)
343 delete.lp(LP_API)
344 ##########################################################################
345 # bootstrap DEA Model using lpSolveAPI
346 ##########################################################################
347 set.seed(1001)
348 (n=J)
349 (m=round(sqrt(n),0))
350 B=250
351 ############################
352 # B=number of bootstraps
353 ############################
354 statusB=matrix(NA,B,n)
355 boot=matrix(NA,B,n)
356 ############################
357 AB=matrix(0,M+N+1,m+1)
358 objB=c(rep(0,m),1)
359 ############################
360 ncB=ncol(AB)
361 LP_B=make.lp(nrow=nr,ncol=ncB)
362 lp.control(LP_B,sense=objtype)
363 set.objfn(LP_B,objB)
364 set.constr.type(LP_B,rest)
365
366 for(b in 1:B){
```

```
367       pickval=sample(1:J,m)
368       AB[,1:m]=A[,pickval]
369
370       for(i in 1:nrB){
371       set.row(LP_B,i,AB[i,])
372       }
373
374     for(s in 1:J){
375         (AB[,1]=A[,s])
376         AB[1:M,m+1]=-AB[1:M,1]
377         rhsB=c(rep(0,M),(AB[(M+1):(N+1)]),1)
378         set.rhs(LP_B,rhsB)
379         set.column(LP_B,1,AB[,1])
380         set.column(LP_B,ncB,AB[,ncB])
381         set.objfn(LP_B,objB)
382         statusB[b,s]=solve(LP_B)
383         boot[b,s]=get.objective(LP_B)
384       }# end loop for (s in 1:J)
385   #The next line prints to the screen which iteration of the bootstrap
386   #is being calculated
387     if(b%%1==0|b==B)  print(paste('on bootstrap rep',b,'of',B))
388   } # end loop on b#################
389   RTS='VRS'
390   # construct the "S" matrix
391   if(RTS=='CRS'|RTS=='crs') (beta=2/(M+N))
392   if(RTS=='VRS'|RTS=='vrs') (beta=2/(M+N+1))
393   S=matrix(0,B,n)
394     for(s in 1:J) S[,s]=objvals2[s]-((m/n)^beta)*(boot[,s]-objvals2[s])
```

```
mean(objvals2) #Calculates mean TE score for initial DEA Model
mean(S)        #Calculates mean TE score across all bootstrap runs
quantile(S,0.025) #Calculates lower 2.5% tail of bootstrap scores
quantile(S, 0.5)  #Calculates median of bootstrap scores
quantile(S, 0.975) #calculates upper 2.5% tail of bootstrap scores
```

Y1	FX1	FX2	VX1	VX2
18180	97	859	8	9
17675	70	369	7	16
14595	64	505	7	22
18180	93	756	7	10
18180	71	303	6	9
18167	85	1182	8	9
30906	70	606	8	23
17850	64	359	7	12
17170	95	516	8	15
17776	76	687	8	9

Figure 1. YX matrix for DEA program

V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11
18180	17675	14595	18180	18180	18167	30906	17850	17170	17776	0
97	70	64	93	71	85	70	64	95	76	0
859	369	505	756	303	1182	606	359	516	687	0
8	7	7	7	6	8	8	7	8	8	0
9	16	22	10	9	9	23	12	15	9	0

Figure 2. Initial A matrix for DEA program after inserting values from YX matrix.

V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11
18180	17675	14595	18180	18180	18167	30906	17850	17170	17776	0
97	70	64	93	71	85	70	64	95	76	-97
859	369	505	756	303	1182	606	359	516	687	-859
8	7	7	7	6	8	8	7	8	8	-8
9	16	22	10	9	9	23	12	15	9	-9

Figure 3. Final A matrix for solving input oriented TE model with CRS for DMU 1.

V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11
18180	17675	14595	18180	18180	18167	30906	17850	17170	17776	0
97	70	64	93	71	85	70	64	95	76	-97
859	369	505	756	303	1182	606	359	516	687	-859
8	7	7	7	6	8	8	7	8	8	-8
9	16	22	10	9	9	23	12	15	9	-9
1	1	1	1	1	1	1	1	1	1	0

Figure 4. Final A matrix for solving input oriented TE model with variable returns to scale for DMU 1.

V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11
18180	17675	14595	18180	18180	18167	30906	17850	17170	17776	-18180
97	70	64	93	71	85	70	64	95	76	0
859	369	505	756	303	1182	606	359	516	687	0
8	7	7	7	6	8	8	7	8	8	0
9	16	22	10	9	9	23	12	15	9	0
1	1	1	1	1	1	1	1	1	1	0

Figure 5. Final A matrix for solving output oriented TE problem with variable returns to scale for DMU 1.

V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V13
18180	17675	14595	18180	18180	18167	30906	17850	17170	17776	0	0	-18180
97	70	64	93	71	85	70	64	95	76	0	0	0
859	369	505	756	303	1182	606	359	516	687	0	0	0
8	7	7	7	6	8	8	7	8	8	-1	0	0
9	16	22	10	9	9	23	12	15	9	0	-1	0
1	1	1	1	1	1	1	1	1	1	0	0	0

Figure 6. Initial A matrix for Johansen plant capacity model observation 1.

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11
1	2913	7721	6080	3315	19212	22516	14188	3350	4193	753	-2913
2	3044	7972	10667	4005	508	10881	12655	8678	11153	4443	-3044
3	2458	3093	1810	18	814	5464	501	1604	9764	867	2458
4	88	117	92	74	107	95	87	79	77	82	0
5	682	1518	1100	451	1034	935	468	572	572	528	0
6	4	6	5	4	7	6	5	6	5	6	0
7	6	11	8	6	9	10	13	11	11	6	0
8	1	1	1	1	1	1	1	1	1	1	0

Figure 7. A matrix for ddf model with one undesirable output, observation one.

Appendix 2. Data used and Results from sample DEA programs

DMU	Output	Fixed Input 1	Fixed Input 2	Variable Input 1	Variable Input 2
1	18180	97	859	8	9
2	17675	70	369	7	16
3	14595	64	505	7	22
4	18180	93	756	7	10
5	18180	71	303	6	9
6	18167	85	1182	8	9
7	30906	70	606	8	23
8	17850	64	359	7	12
9	17170	95	516	8	15
10	17776	76	687	5	9

Figure 8. Outputs and inputs for 10 observations used in DEA examples

DMU	Input Oriented TE (CRS)	Input Oriented TE (VRS)	Output Oriented TE (VRS)	Johansen Capacity Model (VRS)	DDF Model (VRS)
1	1.000	1.000	1.000	1.700	0.000
2	0.846	0.950	1.168	1.168	0.046
3	0.557	1.000	1.223	1.223	0.000
4	0.900	0.900	1.050	1.700	0.033
5	1.000	1.000	1.000	1.000	0.000
6	0.999	1.000	1.001	1.701	0.000
7	1.000	1.000	1.000	1.000	0.000
8	0.894	1.000	1.000	1.000	0.000
9	0.659	0.750	1.376	1.580	0.202
10	0.978	1.000	1.023	1.739	0.000

Figure 9. Results from five DEA models using data from 10 observations.

Observation	Optimal Variable Input 1	Optimal Variable Input 2
1	8.000	23.000
2	6.518	12.097
3	7.000	12.000
4	8.000	23.000
5	6.000	9.000
6	8.000	23.000
7	8.000	23.000
8	7.000	12.000
9	7.406	18.842
10	8.000	23.000

Figure 10. Optimal levels of variable inputs from Johansen capacity model.