NOAA Technical Memorandum NMFS-NE-160

This report series represents a secondary level of scientifiic publishing. All issues employ thorough internal scientific review; some issues employ external scientific review. By design, reviews are transparent collegial reviews, not anonymous peer reviews. All issues may be cited in formal scientific communications.

Measuring Technical Efficiency and Capacity in Fisheries by Data Envelopment Analysis Using the General Algebraic Modeling System (GAMS): A Workbook

John B. Walden[1] and James E. Kirkley[2]

[1]National Marine Fisheries Serv., Woods Hole Lab., Woods Hole, MA 02543
[2]Coll. of William & Mary, Virginia Inst. of Marine Science, Gloucester Point, VA 23062

U. S. DEPARTMENT OF COMMERCE
Norman Y. Mineta, Secretary
National Oceanic and Atmospheric Administration
D. James Baker, Administrator
National Marine Fisheries Service
Penelope D. Dalton, Assistant Administrator for Fisheries
Northeast Region
Northeast Fisheries Science Center
Woods Hole, Massachusetts

October 2000

Contents

Figures

Tables

Acronyms

CU	=	capacity utilization
CRS	=	constant returns to scale
CSV	=	comma-separated value
DEA	=	data envelopment analysis
DMU	=	decision-making unit
GAMS	=	General Algebraic Modeling System
NRS	=	nonincreasing returns to scale
VRS	=	variable returns to scale

INTRODUCTION

This workbook presents several programs for modeling production efficiency and fishing capacity which were prepared for the National Marine Fisheries Service Workshop on Capacity Estimation in Marine Fisheries ("National Capacity Workshop") held in Silver Spring, Maryland, during September 29 - October 1, 1999. The programs are written in the General Algebraic Modeling System (GAMS) language, a mathematical programming language used in a variety of linear, nonlinear, and mixed-integer programming models, general equilibrium models, and network models.[1]

Each program in this workbook uses data envelopment analysis (DEA) to calculate measures of productive efficiency and fishing capacity. DEA was developed over 20 yr ago as a tool in management science to examine efficiency in the public sector (Charnes *et al.* 1994). The programs are based on a DEA model written in GAMS by Olesen and Petersen (1996), who argue that GAMS is preferable to the specialized DEA software packages currently available because of the flexibility that GAMS offers. The user can easily modify GAMS code to account for changes to the "standard" DEA model, and can exert greater control over output formats, features not typically available in the specialized packages. These features are important in fisheries where production analysis often differs from that available with the standard models.

The programs in this workbook are simplified versions of those programs presented at the National Capacity Workshop, and are designed to show the basic GAMS syntax for developing DEA models. Because there was little time at the workshop to discuss all of the programs, this workbook was prepared. Each workbook program is preceded by a description of the mathematical model on which it is based, and then followed by an explanation of important program sections.

BRIEF OVERVIEW OF DEA

Charnes, Cooper, and Rhodes first introduced DEA in 1978. DEA extended the Farrell (1957) technical measure of efficiency from a single-input, single-output process to a multiple-input, multiple-output process. Since then, DEA has been used to assess efficiency in many different areas, ranging from the public sector to natural resource sectors such as the fishing industry.

DEA uses linear programming methods to extract information about the production process of each decision-making unit (DMU, *e.g.*, firm or fishing vessel). This information extraction is accomplished by calculating a maximum performance measure for each firm, and comparing this measure to similarly calculated measures for all other firms. Each firm's performance measure traces out a best-practice frontier, and all DMUs lie either on or below the frontier (Charnes *et al.* 1994). A best-practice frontier maps out the maximum level of output (minimum level of input) that could be produced (used) for any given level of input (output). Figure 1 shows a graphical representation of an output-oriented DEA model with a single input for 10 firms. The best-practice frontier is traced through the points representing the maximum level of output for a given input; any points below the frontier are deemed inefficient. For example, the DMU at point (8,12) produces 12 units of output with 8 units of input, while the DMU at point (8,9) produces 9 units of output with 8 units of input. The second DMU is deemed to be inefficient compared to the first because only 9 units of output (versus 12) are produced for the same level of input. Inefficiency for any DMU is determined by comparison to either another DMU or to a convex combination of other DMUs on the frontier which utilize the same level of input and produce the same or higher level of output. The analysis is accomplished by requiring solutions that can increase some outputs (decrease some inputs) without worsening the other inputs or outputs (Charnes *et al.* 1994).

The one-input, one-output case can be expanded to cases involving multiple inputs and multiple outputs. Charnes *et al.* (1978) proposed a method in which the multiple-input, multiple-output model was reduced to a ratio with a single "virtual" input and single "virtual" output by estimating a set of weights depicting each DMU in the most favorable position relative to other DMUs. In equation form, the model is as follows:

$$\max_{u,v} h_0(u,v) = \frac{\sum_r u_r y_{ro}}{\sum_i v_i x_{io}}$$

s.t.:

$$\frac{\sum_r u_r y_{rj}}{\sum_i v_i x_{ij}} \le 1, \text{ for } j = 0,1,\ldots,n,$$

$$\frac{u_r}{\sum_i v_i x_{io}} \ge \varepsilon, \text{for } r = 1,\ldots,s, \text{and}$$

$$\frac{v_i}{\sum_i v_i x_{io}} \ge \varepsilon, \text{for } i = 1,\ldots,m,$$

where:

y_{rj} = quantity of output r produced by firm j,
x_{ij} = quantity of input i produced by firm j,
u_r = weight for output r,
v_i = weight for input i, and
ϵ = small positive quantity.

The estimated ratio provides a measure of technical efficiency for each DMU. However, there are an infinite number of solutions because if (u^*,v^*) is optimal, then $(\beta u^*,\beta v^*)$ is also optimal for $\beta > 0$ (Charnes *et al.* 1994). This problem is corrected by converting the ratio form into an equivalent linear programming problem as follows:

$$\max_{u,v} w_0 = \sum_r u_r y_{ro}$$

s.t.:

$$\sum_i v_i x_{i0} = 1,$$

$$\sum_r u_r y_{rj} - \sum_i v_i x_{ij} \le 0,$$

$$u_r \ge \varepsilon, \text{and}$$

$$v_i \ge \varepsilon.$$

Färe *et al.* (1994) developed a variation of the preceding linear programming approach to model efficiency, productivity, and capacity. The models they use measure the efficiency of individual producers by constructing a "best-practice frontier" through a piecewise linear envelopment of the data generated by all producers in the group. Estimates generated by those models are therefore "relative" measures based on the best producers within the group.

The following sections describe several linear programming models to estimate input and output technical efficiency and capacity output based on the approach used by Färe *et al.* (1994). Each model is accompanied by an example and description of a GAMS program to solve the model.

INPUT-ORIENTED TECHNICAL EFFICIENCY

An input-oriented technical efficiency model examines the vector of inputs used in the production of any output bundle, and measures whether a firm is using the minimum inputs necessary to produce a given bundle of outputs. Efficiency is measured by the maximum reduction in inputs which will still allow a given output bundle to be produced. Figure 2 depicts the results of an input-oriented model using a single-input, single-output example. Firms to the right of the frontier are deemed to be inefficient because they could produce the same level of output for less input. For example, the point (6,5) means 6 units of input are used to produce 5 units of output. Another firm is using 3 units of input to produce 5 units of output. The first firm is technically inefficient compared to the second firm because much more input is used to produce the same level of output.

Färe *et al.* (1994) proposed the following input-oriented DEA model to measure technical efficiency:

$$\min_{\lambda,z} \lambda$$

s.t.:

$$u_{jm} \le \sum_{j=1}^{J} z_j u_{jm}, m = 1,2,\ldots,M, \qquad \text{Eq 1}$$

$$\sum_{j=1}^{J} z_j x_{jn} \le \lambda x_{jn}, n = 1,2,\ldots,N, \qquad \text{Eq 2}$$

$$z_j \ge 0, j = 1,2,\ldots J, \qquad \text{Eq 3}$$

where:

λ = efficiency measure to be calculated for each DMU j,
u_{jm} = quantity of output m produced by DMU j,
x_{jn} = quantity of input n used by DMU j, and
z_j = intensity variable for DMU j.

Since the variable λ is calculated for each DMU, the preceding formulation is estimated once for each DMU in the data set. Equations 1 and 2 define a set of constraints for each output and input. If there are two outputs, Equation 1 will define a set of constraints, one for each output. A value of λ=1.0 means that a firm is considered efficient, while a value λ<1.0 means a firm is inefficient. Thus, a value of λ=0.70 means that a firm could reduce its inputs by 30%, and produce the same level of output. An example of the GAMS formulation for this model is shown in Figure 3. In this example, there are two outputs, called "spec1" and "spec2", and four inputs, called "fix1", "fix2", "var1", and "var2".[2]

Lines 1-6 define six sets to be used in the model. Sets are the basic building blocks of GAMS, and correspond to the *m*, *n*, and *j* indexes in the DEA technical efficiency equations. Line 1 defines a set called "inout" which has as members "spec1", "spec2", "fix1", "fix2", "var1", and "var2". All inputs and outputs are defined as members of one set initially. Lines 2 and 3 define subsets of "inout" that contain, respectively, the outputs (*m*) or inputs (*n*), initially included as members of "inout". Subsets can only contain elements which exist in a previously defined set. Lines 4-6 define sets which correspond to index *j*. Line 4 defines a set called "obs" which contains 500 members. Line 5 defines a subset of "obs", called "subobs", which in this case holds the first 10 members of set "obs". Declaring "subobs" to be a subset of "obs" allows different sets of data to be used as long as they contain between 1 and 500 observations. In GAMS syntax, the "*" operator in a set declaration signifies all elements between 1 and 10. This is simpler than typing in 1,2,3,...,10, especially when there are a large number of observations. Line 6 declares another subset of "obs", called "actobs", which is initially empty. This is referred to in GAMS as a dynamic set, because its membership can change.

Line 7 is an example of an alias statement, which allows the set "subobs" to be referred to using either the name "subobs", or "subobs1". As shown in line 22, this is needed for the "loop" statement. Line 8 reads the data that will be used in the model through the use of the "table" statement. In this example, a table "act", with indexes "obs" and "inout", is declared, and the values are initialized with the "table" statement. The rows of table "act" correspond to observations (*j*), while the columns correspond to either inputs or outputs. The column headings need to align with the data contained in the column, so each data element is right justified. For large data sets, it is usually easier to read in an external file containing the data. An example of this will be shown in a subsequent program.

Line 9 uses a semicolon to end the "table" statement. It is generally good practice to end every GAMS statement with a semicolon, as is shown on lines 6, 7, and 9.

Lines 10-13 define the variables to be used in the model. Variables are sometimes referred to as activities, columns, decision variables, or endogenous variables. Variable values are generally unknown until after the model is solved. The two variables in this example are called "lambda" and "weight". "Lambda" is the decision variable to be optimized, while "weight" is the intensity variable that corresponds to z_j in Equations 1, 2, and 3. Line 13 declares "weight" to be a positive variable. The default variable type in GAMS is free, meaning variables can take on positive or negative values.[3] The decision variable "lambda" must be of type free.

Lines 14-18 define the equations used in the model. Equations must first be declared in GAMS (lines 15 and 16) and then defined (lines 17 and 18). "Constr1" is indexed over the sets "output" and "obs", while "constr2" is indexed over the sets input and "obs". When the equations are defined in lines 17 and 18, the subsets "actobs" and "subobs" are substituted for the set "obs". The reason will be clear after examining programming lines 22-28. Being able to index an equation is a very powerful tool. "Constr1" actually defines 20 different equations because there are two outputs and 10 observations, and this is succinctly represented by one equation definition. On lines 17 and 18 immediately after the equations are found two dots ".."; these are required by GAMS in the equation definition.

Lines 19 and 20 define a parameter "score1", used to hold results from the model. Line 21 defines a model called "tedea", which consists of two equations, "constr1" and "constr2". An alternative way to define this model would be as follows: "model tedea /all/;". This indicates that GAMS should use all (or in this case both) of the equations in the model. By specifying which equations to include in the model, one has the ability to use several different model formulations in the same program.

Lines 22-28 solve the model. In a DEA model, the linear programming problem is solved once for each DMU, which is accomplished in GAMS through the use of the "loop" statement (line 22). Because the subset "subobs" is referenced in the equation definitions, the loop has to be indexed over subset "subobs1" (which has been declared previously to be an alias for "subobs"). In line 23, the set "actobs" is made an empty set, which is needed for each pass through the loop. In line 24, one observation is added to the set "actobs", which is the current observation of subset "subobs1". Referring back to the equation definitions on lines 17 and 18, this has the effect of indexing "constr1" only over the set "output", and indexing "constr2" only over the set input, because "actobs" only contains one observation (j_0). Comparing these equations with the mathematical model reveals that the right- and left-hand sides of Equation 1 have been switched and that the inequality has been reversed. However, the fundamental relationship of the equation remains the same. Line 25 tells GAMS to use the OSL solver to solve the LP model. The default solver that comes with GAMS is BDMLP, but this software was found not to solve more complex models at each iteration. Both the OSL and the MINOS5 solvers are able to solve most DEA problems.[4] Line 26 solves the model through the SOLVE statement, by telling GAMS to minimize "lambda". Line 27 stores the value of "lambda" obtained from each solve statement in the parameter "score1". The ".l" extension tells GAMS to use the solution value of "lambda". The "loop" statement is then closed on line 28 with a ");". Line 29 uses the display statement in GAMS to print the values held in parameter "score1" to the listing file.

Results from this model are shown in Table 1 (along with the results from the output-oriented model which will be presented in the next section). Only observation #4 is deemed to be efficient (λ=1.00). All other observations could reduce their inputs and still produce the same level of output, if they used their inputs as efficiently as observation #4.

OUTPUT-ORIENTED TECHNICAL EFFICIENCY

Output technical efficiency is a measure of the potential output of a DMU given that inputs are held constant. Färe *et al.* (1994) modeled the output technical efficiency measure for any DMU using linear programming:

$$\max_{\theta,z} \theta$$

s.t.:

$$\theta u_{jm} \le \sum_{j=1}^{J} z_j u_{jm}, m = 1,2,\ldots, M,$$

$$\sum_{j=1}^{J} z_j x_{jn} \le x_{jn}, n = 1,2,\ldots, N, \text{and}$$

$$z_j \ge 0, j = 1,2,\ldots., J.$$

where:

θ = output technical efficiency measure,
u_{jm} = quantity of output m produced by DMU j,
x_{jn} = quantity of input n produced by DMU j, and
z_j = intensity variable for DMU j.

An example of the GAMS formulation for this model is shown in Figure 4. This example uses the same inputs and outputs as the previous example. A value of θ=1.0 signifies that the DMU is efficient; a value of θ>1.0 indicates that the DMU is inefficient. For example, a score of 1.25 means that it should be possible to increase all

outputs from a DMU by 25% with the same level of inputs.

In Figure 4, line 1 is an example of a dollar control operator.[5] This operator allows comments to be placed on lines using a "/*" to begin the comment and a "*/" to end the comment. Line 2 shows an example of a comment in the GAMS program. Lines 3-8 are the same as lines 1-6 in the input-oriented program, and define the sets used in the program.

Lines 10-13 demonstrate a way to read in data from an external file. In this example, a comma-separated value (CSV), Microsoft Excel file is read into the program. Line 10 names the table "act", and defines it to consist of dimension "obs" and "inout". Line 11 shows the use of the dollar control operator "$ondelim", which allows GAMS to read a CSV-formatted file from Microsoft Excel. Line 12 uses the "$include" command to read into GAMS the contents of an external file (in the format shown in Figure 5). The first column of the input file must have a heading of "dummy" for the file to be properly read into GAMS.

Lines 14-17 define the variables used in the model, while lines 18-22 declare the equations and then define them. Line 25 defines an external file where results will be written, and names this file "primal". Lines 26-32 are the same as in the input-oriented program, with the exception that the variable "theta" is to be maximized rather than "lambda" minimized.

Lines 33-37 write messages to an external file indicating whether the model was solved at each iteration. This is important because the model may fail to solve some observations but may successfully solve others. By examining these messages, the user can easily see if the model solved at each iteration. Line 33 is telling GAMS that the file referenced by the name "primal" is the current file, and items referenced through further use of the "put" statement are to be written to that file. Lines 34-37 use an "if-else" construct to determine what messages will be written to file "primal". Line 34 tests the return codes from GAMS to determine if the model solved correctly. The suffix ".modelstat" appended to "tedea" tests for a return code of "1", which indicates that the solution is optimal. The suffix ".solvestat" informs the user that the solver terminated normally and there were no problems solving the model when the return code is "1". If the model solved correctly and the solver completed normally, then line 35 instructs that the observation number, the term "optimal", and the term "normal completion" be written out to a file. If these conditions do not exist, then lines 36 and 37 indicate that the codes which are returned should be written to the file.[6]

Lines 39-44 write the technical efficiency results to a CSV file which can be opened in Excel. Line 40 tells GAMS that the file will be of type ".csv", by using the suffix ".pc", and setting this equal to 5.[7] Lines 42 and 43 use a loop structure to print out the observation number (or DMU) and the efficiency estimate that was stored in parameter "score1". Line 44 closes the open file "res" with the "putclose" command.

Results from the output-oriented model are summarized in Table 1. The inefficient firms are identical to those found using the input-oriented model. In fact, results for the output-oriented model are equal to $1/\lambda$.[8] Theta values from the output-oriented model indicate how much each DMU should be able to increase output production given that the inputs are held constant. For example, in Table 1, firm (observation) #1 had a value of θ=1.10. This value indicates that this firm should have been able to increase production of both "spec1" and "spec2" by 10% (*e.g.*, 13,295x1.1; 27,065x1.1), if inputs were used as efficiently as firm (observation) #4.

AN OUTPUT-ORIENTED MODEL WITH SLACK VARIABLES

In the output-oriented technical efficiency model, DMUs are deemed to be efficient if θ=1, and inefficient if θ>1. For the inefficient DMUs, outputs are expanded proportionally by multiplying theta times output. However, by incorporating slack values, nonradial levels can be obtained which are greater then radially expanded output levels. In a linear programming model, slack values are derived by converting inequality constraints to equality constraints and adding slack variables. A full discussion of slack values is given in Intriligator (1971), but the general approach can be seen in the following simple example from Intriligator (1971):

$$\max_{x} F(x)$$

s.t.:

$$g(x) \le b, \text{and}$$

$$x \ge 0$$

Inequality constraints are turned into equality constraints by adding a vector of *m* slack variables:

$$s \equiv b - g(x) = (s_1, s_2, s_3, \ldots, s_m)'$$

The problem is now written as :

$$\max_{x,s} F(x)$$

s.t.:

$$g(x) + s = b,$$

$$x \ge 0, \text{ and}$$

$$s \ge 0.$$

The non-negativity condition on the slack variables ensures that the inequality constraints are met. Revising the output-oriented model in the prior section yields the following model formulation:

$$\underset{\theta,z,s}{\text{Max}}\ \theta$$

s.t.:

$$\theta u_{jm} = \sum_{j=1}^{J} z_j u_{jm} - S_m, m = 1,2,\ldots, M, \quad \text{Eq 4}$$

$$\sum_{j=1}^{J} z_j x_{jn} + S_n = x_{jn}, n = 1,2,\ldots, N, \quad \text{Eq 5}$$

$$z_j \geq 0, j = 1,2,\ldots, J, \quad \text{Eq 6}$$

$$S_m \geq 0, m = 1,2,\ldots, M, \text{ and} \quad \text{Eq 7}$$

$$S_n \geq 0, n = 1,2,\ldots, N. \quad \text{Eq 8}$$

Equation 4 defines a set of constraints for each output which equates theta times the observed output level for DMU *j* to the sum over all DMUs of the intensity variables (z_j) times each DMU's output level, minus the slack output level. The non-negativity constraint (Equation 7) on the slack variable means that the variable will have a value of zero or greater. When the left-hand side of Equation 4 equals the summation on the right-hand side, the slack variable is zero. When the left-hand side is less than the summation on the right-hand side, the slack variable is positive, so that the equality constraint holds. Adding the slack variable to each side of Equation 4 yields the following:

$$\theta u_{jm} + S_m = \sum_{j=1}^{J} z_j u_{jm} \quad \text{Eq 9}$$

The z_j variables map out the linear segments of the frontier (Färe *et al.* 1994), and determine frontier output. By adding the value of the slack variable s_m to the term θu_{jm}, the output for product *m* on the left-hand side of Equation 9 is exactly equal to the frontier output on the right-hand side.

GAMS[9] allows the user to display the values of the slack variables directly, but slack variables can also be included by modifying the constraints and incorporating them explicitly. An example is Figure 6, which uses the original output-oriented technical efficiency program from the previous section.

In Figure 6, lines 1-13 set up the problem in the same manner as the original output technical efficiency measure. Lines 14-18 define the variables to be used and include two new variables that were not in the previous example. The variables "slack1", to handle output slack, and "slack2", to handle input slack, are declared on lines 17 and 18.[10] Line 19 specifies that both slack variables are positive. Lines 23-24 define the equations that now incorporate the slack variables. Both equations have equality constraints rather than the inequality constraints that were in the original output efficiency model. On line 36, the output slack values returned by the model are stored in parameter "score2". Results from the program are then written to a file on lines 45-60.

Table 2 shows results from the output-oriented DEA model. Values of theta (the decision variable) are the same as obtained from the output-oriented technical efficiency model. Slack values for output "spec1" are provided for each observation except #4, which is on the best-practice frontier (θ=1 and slack values are zero). For all observations, the slack for "spec2" is zero.

MEASURING CAPACITY WITH AN OUTPUT-ORIENTED DEA MODEL

Färe *et al.* (1994) developed a DEA model of capacity based on the definition of capacity given by Johansen (1968): Capacity is the "maximum amount that can be produced per unit of time with existing plant and equipment, provided that the availability of variable factors of production is not restricted." To model capacity, the input vector is separated into a subvector of fixed inputs, and a subvector of variable inputs. The subvector of fixed inputs is bounded by observed values, while the bounds on the variable inputs are allowed to vary. This essentially constrains production by the fixed factors, consistent with the Johansen definition.

The mathematical model proposed by Färe *et al.* (1994) is as follows:

$$\underset{\theta,z,\lambda}{\text{Max}}\ \theta$$

s.t.

$$\theta u_{jm} \leq \sum_{j=1}^{J} z_j u_{jm}, m = 1,2,\ldots, M,$$

$$\sum_{j=1}^{J} z_j x_{jn} \leq x_{jn}, n \in F_x,$$

$$\sum_{j=1}^{J} z_j x_{jn} = \lambda_{jn} x_{jn}, n \in V_x,$$

$$z_j \geq 0, j = 1,2,\ldots, J,$$

$$\lambda_{jn} \geq 0, n \in V_x.$$

where:

θ = capacity measure,
u_{jm} = quantity of output *m* produced by firm *j*,
x_{jn} = quantity of input *n* used by firm *j*,
$n \in F_x$ = inputs belonging to the set of fixed factors,
$n \in V_x$ = inputs belonging to the set of variable factors,
λ_{jn} = input utilization rate of variable input *n* by firm *j*, and
z_j = intensity variable for firm *j*.

Programming of this model formulation in GAMS is shown in Figure 7. One advantage GAMS has over other available DEA programs is that it allows direct estimation of λ, the variable input utilization rate. The value of lambda is the ratio of the optimal use of each input to its actual usage. For example, a value of 1.25 means that the vari-

able input should be increased 25% for a firm to be on the best-practice frontier.

This GAMS model differs from the output-oriented model in that the set of inputs is divided into two subsets containing either fixed or variable inputs (lines 4 and 5). Lines 6-15 are the same as in the output-oriented DEA program, with the exception of lines 10 and 15. Line 10 instructs GAMS not to print the following lines to the listing file. This is particularly useful in reducing the size of the listing file when the data set is quite large. Line 15 instructs GAMS to resume printing to the listing file.

Lines 16-19 declare the variables used in the program. The variable "lambda" (line 19) is declared over the set of variable inputs. Lines 21-24 declare the model equations with the input constraints addressed. Lines 31-44 are similar to the GAMS program used to model output-oriented efficiency.

Lines 47-61 direct the output results to a CSV file that can be read in Microsoft Excel. Lines 45 and 46 estimate capacity in GAMS internally rather than externally. Capacity for each DMU is calculated by multiplying theta by each output, and then summing over all outputs. Lines 52-61 use a series of "loop" statements to write the results to a CSV file. Lines 51-54 put in header information, while lines 55-61 write the data values and model results to a file.

Table 3 presents the results from this model, and lists the level of variable inputs used by each firm and the optimal variable utilization rate, lambda. The optimal variable utilization rate indicates how each firm would have to change the use of each variable input to operate at capacity. For example, for observation #2, variable input 1 would need to be decreased to 5.04 (8x0.63), and variable input 2 would need to be increased to 185.4 (127x1.46).

MODELING RETURNS TO SCALE

The previous programs implicitly assume constant returns to scale. From an economic viewpoint, allowing variable returns to scale results in a less restrictive model than that imposed by constant returns to scale. From an operations research viewpoint, just the opposite is true because an additional constraint, convexity, is imposed on the model. The practical implication of "imposing" variable returns to scale is that it is easier for some observations to be deemed efficient and placed on the frontier because imposition of the convexity constraint means that the supporting hyperplane does not have to pass through the origin (Charnes *et al.* 1994).

To impose variable returns to scale, the following equation is added to the model:

$$\sum_{j=1}^{J} z_j = 1$$

Alternatively, to impose nonincreasing returns to scale requires the following equation to be used:

$$\sum_{j=1}^{J} z_j \leq 1$$

Imposing variable or nonincreasing returns to scale requires a single equation in GAMS. To show the programming steps in GAMS, only the necessary adjustments to the previous capacity output program are shown in Figure 8.

The DEA constraint imposing variable returns to scale is declared on line 5, and then defined on line 9. Line 10 declares a model "tedea", which includes all four equations. Line 9 could be modified to impose nonincreasing returns to scale by changing the "=E=" (equality) to an "=L=" (inequality).

Table 4 shows results from an output-oriented DEA model under three different assumptions: 1) nonincreasing returns to scale (NRS), 2) variable returns to scale (VRS), and 3) constant returns to scale (CRS). Both the NRS and CRS results are identical. With the VRS model, it is easier for DMUs to be placed on the best-practice frontier. This is evident in Table 4 in that more observations have a score of 1.00 under the VRS model then under the NRS or CRS models. Further explanation into why this occurs can be found in Charnes *et al.* (1994) and Färe *et al.* (1994).

MODELING CAPACITY UTILIZATION

Capacity utilization (CU) generally refers to the proportion of potential capacity that is used, and is typically measured as the ratio of actual output to capacity output (Kirkley and Squires 1999). This ratio generally cannot exceed 1.0. Färe *et al.* (1989) proposed that CU be measured as the ratio of output technical efficiency to capacity output. This ratio corrects for downward bias that could arise because actual observed output may be inefficiently produced. Shown in Figure 9 is a GAMS program which estimates technical efficiency, capacity output, and capacity utilization.

The entire program will not be reviewed, but key lines will be highlighted. Lines 28-31 define the four equations to be used. These equations have been previously defined in the programs for output technical efficiency and capacity output. A parameter "score1(obs,*)" is defined on line 33, which will hold results from two different models. The "*" allows use of explicit labels in the parameter instead of a specific set. This is seen on line 45, where the label "TE" is used to hold the value of theta estimated by GAMS.

Lines 38 and 39 define two separate models. The first (line 38) defines "tedea", which models output technical efficiency, and the second (line 39) defines "cap", which models capacity output. These models are solved in two different "loop" statements, with the results stored in parameter "score1". Capacity utilization is calculated in line 69, and then also stored in "score1". Capacity for each vessel is calculated in lines 70 and 71. The estimates of technical efficiency, capacity output, total capacity, and capacity utilization are written to a file in lines 72-91.

Table 5 presents the results from this program using output-oriented DEA models with constant returns to scale. For all observations (except observation #4), observed capacity utilization was less than the unbiased capacity utilization, as expected.

SUMMARY AND CONCLUSIONS

Various DEA models were presented which estimate input-oriented technical efficiency, output-oriented technical efficiency (with and without explicit slack variables), capacity, and capacity utilization. Each model was accompanied by a GAMS program. A key objective was to demonstrate the flexibility of GAMS in modeling DEA problems. This is particularly important in fisheries where production problems generally differ from the "standard" model.

ENDNOTES

1. Information on GAMS can be obtained from the GAMS website at *www.gams.com*.
2. "Spec1" and "spec2" are species landed by each vessel; "fix1" and "fix2" are fixed inputs; and "var1" and "var2" are variable inputs. The order in which variables are read into GAMS in the data table does not matter, and the GAMS language is not case sensitive.
3. Variables can also be of type negative, binary, or integer.
4. A list of available solvers can be found at the GAMS website, *www.gams.com*.
5. For a complete list of dollar control operators, see the GAMS users guide.
6. A complete list of model status codes and solve status codes can be found in the GAMS users guide.
7. A complete list of output file types can be found in the GAMS users guide.
8. This result only holds when both models assume constant returns to scale. Models with variable returns to scale are discussed in a subsequent section.
9. There is a global option in GAMS which allows one to obtain the slack values from the equation listing. Inserting the line "option solslack=1;" forces the equation to display slack variables rather then level values. A parameter could then be defined to hold the results from the equation output as follows: "Parameter score3(subobs1,output)=constr1.l(output,subobs1);". However, in this approach, the level values for the equations cannot be simultaneously obtained. For more information on this, refer to the GAMS user manual.
10. Because the DEA program is being executed once per DMU in the data set, the slack variables do not need to be indexed over the set "obs". An alternative formulation for the output slack variable would be "slack1(output, actobs)".

ACKNOWLEDGMENTS

We thank Rita Curtis, Phil Logan, Barbara Rountree, and Fred Serchuk for helpful comments and suggestions.

REFERENCES CITED

Charnes, A.; Cooper, W.; Lewin, A.; Seiford, L. 1994. Data envelopment analysis, theory, methodology and applications. Norwell, MA: Kluwer Academic Publishers.

Charnes, A.; Cooper, W.; Rhodes, E. 1978. Measuring the efficiency of decision making units. *Eur. J. Oper. Res.* 2(6):429-444.

Färe, R.; Grosskopf, S.; Kokkenlenbergl, E. 1989. Measuring plant capacity utilization and technical change: a nonparametric approach. *Int. Econ. Rev.* 30(1989):655-666.

Färe, R.; Grosskopf, S.; Lovell, C. 1994. Production frontiers. New York, NY: Cambridge University Press.

Farrell, M.J. 1957. The measurement of productive efficiency. *J. R. Stat. Soc. Ser. A* 120(3):253-290.

Intrilligator, M.D. 1971. Mathematical optimization and economic theory. Englewood Cliffs, NJ: Prentice-Hall.

Johansen, L. 1968. Production functions and the concept of capacity. *In:* Recent research on the function of production. Namur, France: Namur University Center for Study and Research.

Kirkley, J.; Squires, D. 1999. Capacity and capacity utilization in fishing industries. [Unpubl. manuscr.]. Gloucester Point, VA: Virginia Institute of Marine Science.

Olesen, O.B.; Petersen, N.C. 1996. A presentation of GAMS for DEA. *Comput. Oper. Res.* 23(4):323-339.

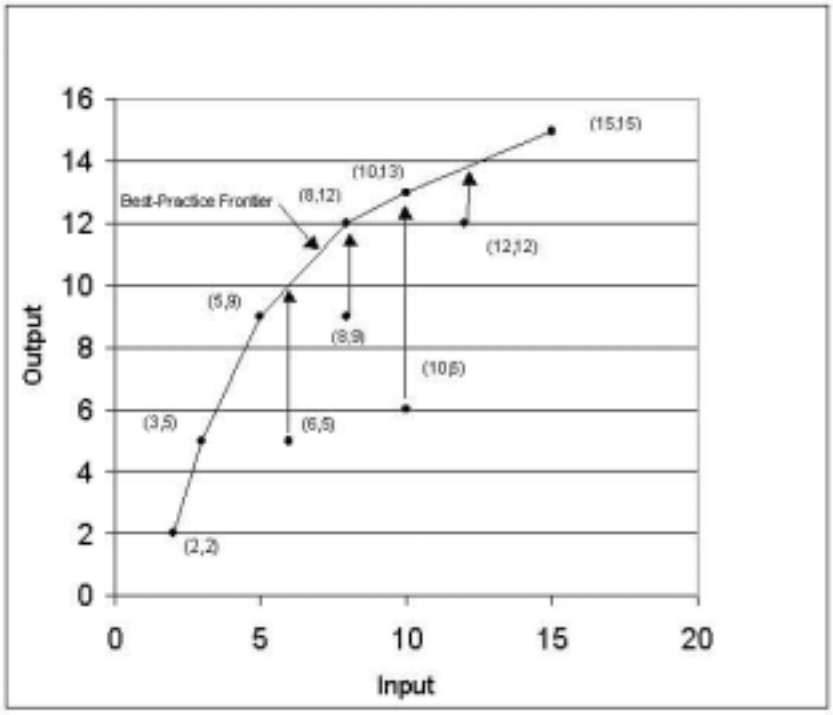

Figure 1. Output-oriented DEA model.

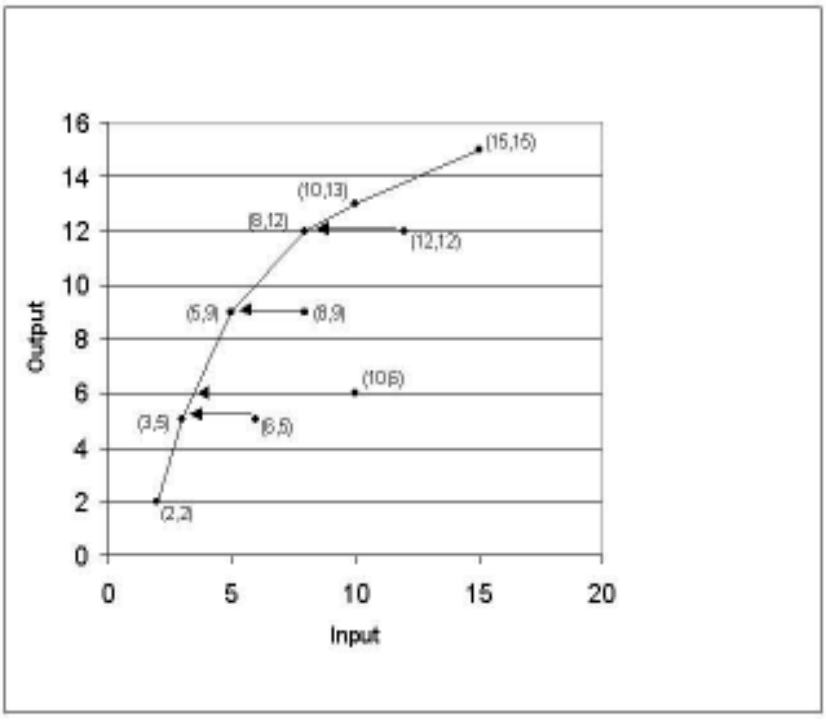

Figure 2. Input-oriented DEA model.

```
1.  Sets inout /spec1, spec2, fix1, fix2, var1, var2/
2.  Output(inout) /spec1, spec2/
3.  Input(inout) /fix1, fix2, var1, var2/
4.  Obs /1*500/
5.  Subobs(obs) /1*10/
6.  Actobs(obs);
7.  Alias (subobs, subobs1);
8.  Table act(obs,inout) input output table
```

	Spec1	Spec2	Fix1	Fix2	Var1	Var2
1	13295	27065	55	60	4	94
2	13255	10090	63	70	8	127
3	614	3427	59	59	6	35
4	106461	58705	63	69	5	185
5	3540	9130	53	60	5	46
6	602	6900	62	74	5	37
7	12920	18128	69	78	6	133
8	8312	5145	65	63	8	162
9	3276	4430	70	62	3	24
10	4143	8486	63	61	5	81

```
9.  ;
10. Variables
11. Lambda efficiency score
12. Weight(obs) intensity variable;
13. Positive variable weight;
14. Equations
15. Constr1(output,obs) DEA constraint for each output
16. Constr2(input,obs) DEA constraint for each input;
17. Constr1(output,actobs).. sum(subobs,
    weight(subobs)*act(subobs, output)) =G= act(actobs,output);
18. Constr2(input,actobs).. sum(subobs,
    weight(subobs)*act(subobs,input)) =L= lambda*act(actobs,input);
19. Parameter
20. Score1(obs) efficiency scores;
21. Model tedea /constr1, constr2/;
22. Loop (subobs1,
23. Actobs(obs)=no;
24. Actobs(subobs1)=yes;
25. Option LP=OSL;
26. Solve tedea minimizing lambda using LP;
27. Score1(subobs1)=lambda.l;
28. );
29. Display score1;
```

Figure 3. Example of a GAMS program to solve an input-oriented DEA model which measures technical efficiency.

```
1.  $oninline
2.  /*next define inputs and outputs*/
3.  Set inout /spec1, spec2, fix1, fix2, var1, var2/
4.  Output(inout) /spec1, spec2/
5.  Input(inout) /fix1, fix2, var1, var2/
6.  Obs /1*500/
7.  Subobs(obs) /1*10/
8.  Actobs(obs);
9.  Alias (subobs, subobs1)
10. Table act(obs,inout) input output table
11. $ondelim
12. $include "data.csv"
13. $offdelim
14. Variables
15. Theta efficiency score
16. Weight(obs) weights;
17. Positive variable weight;
18. Equations
19. Constr1(output,obs) DEA constraint for each output
20. Constr2(input,obs) DEA constraint for each input;
21. Constr1(output, actobs).. sum(subobs,weight(subobs)*act(subobs,
    output)) =G= theta*act(actobs,output);
22. Constr2(input, actobs).. sum(subobs,
    weight(subobs)*act(subobs,input)) =L= act(actobs, input);
23. Parameter
24. Score1(obs) efficiency scores;
25. File primal /teout_res.txt/;
26. Model tedea /constr1, constr2/;
27. Loop (subobs1,
28. Actobs(obs)=no;
29. Actobs(subobs1)=yes;
30. Option LP=OSL;
31. Solve tedea maximizing theta using LP;
32. Score1(subobs1)=theta.l;
33. Put primal;
34. If ((tedea.modelstat eq 1 and tedea.solvestat eq 1),
35. Put @1, subobs1.tl, @10, "optimal", @20, "normal completion"/
36. Else put @1, subobs1.tl, @10, tedea.modelstat:>2:0,
37. @20, tedea.solvestat:>2:0/ );
38. );
39. File res /teoutput.csv/;
40. Res.pc=5;
41. Put res;
42. Loop (subobs1,
43. Put subobs1.tl, score1(subobs1)/ );
44. Putclose;
```

Figure 4. Example of a GAMS program to solve an output-oriented DEA model which measures technical efficiency.

Dummy	Spec1	Spec2	Fix1	Fix2	Var1	Var2
1	13295	27065	55	60	4	94
2	13255	10090	63	70	8	127
3	614	3427	59	59	6	35
4	106461	58705	63	69	5	185
5	3540	9130	53	60	5	46
6	602	6900	62	74	5	37
7	12920	18128	69	78	6	133
8	8312	5145	65	63	8	162
9	3276	4430	70	62	3	24
10	4143	8486	63	61	5	81

Figure 5. Format of data in a Microsoft Excel file which can be read into GAMS.

```
1.  Set inout /spec1, spec2, fix1, fix2, var1, var2/
2.  Output(inout) /spec1, spec2/
3.  Input(inout) /fix1, fix2, var1, var2/
4.  Obs /1*500/
5.  Subobs(obs) /1*10/
6.  Actobs(obs);
7.  Alias (subobs, subobs1)
8.  $offlisting
9.  Table act(obs,inout) input output table
10. $ondelim
11. $Include "data.csv"
12. $offdelim
13. $onlisting
14. Variables
15. Theta efficiency score
16. Weight(obs) weights
17. Slack1(output) output slack
18. Slack2(input) input slack;
19. Positive variable weight, slack1, slack2;
20. Equations
21. Constr1(output,obs) DEA constraint for each output
22. Constr2(input,obs) DEA constraint for each input;
23. Constr1(output,actobs).. sum(subobs,
    weight(subobs)*act(subobs, output))-slack1(output) =E=
    theta*act(actobs, output);
24. Constr2(input, actobs)..
    sum(subobs,weight(subobs)*act(subobs,input)) +slack2(input)
    =E= act(actobs, input);
25. Parameter
26. Score1(obs) efficiency scores
27. Score2(obs,output) output slack values;
28. File primal /teout_res.txt/;
29. Model tedea /constr1, constr2/;
30. Loop (subobs1,
31. Actobs(obs)=no;
32. Actobs(subobs1)=yes;
33. Option LP=OSL;
34. Solve tedea maximizing theta using LP;
35. Score1(subobs1)=theta.l;
36. Score2(subobs1,output)=slack1.l(output);
37. Put primal;
38. If ((tedea.modelstat eq 1 and tedea.solvestat eq 1),
39. Put @1, subobs1.tl, @10, "optimal", @20, "normal completion"/
40. Else
41. Put @1, subobs1.tl, @10, tedea.modelstat:>2:0,
42. @20, tedea.solvestat:>2:0/
43. );
44. );
45. File res /teslack.csv/;
46. Res.pc=5;
47. Res.pw=160;
48. Put res;
49. Put 'obs', 'theta',
50. Loop (output,
51. Put output.tl);
52. Put 'sp1slack', 'sp2slack',
53. Loop (subobs1,
54. Put /
55. Put subobs1.tl, score1(subobs1),
56. Loop (output,
57. Put act(subobs1,output));
58. Loop (output,
59. Put score2(subobs1,output));
60. );
61. Putclose;
```

Figure 6. Example of a GAMS program to solve an output-oriented DEA model which measures technical efficiency, and which allows slack variables to be included by modifying the constraints and incorporating them explicitly.

```
1.  $oninline
2.  Set inout /spec1, spec2, fix1, fix2, var1, var2/
3.  Output(inout) /spec1, spec2/
4.  Fixed(inout) /fix1, fix2/
5.  Var(inout) /var1, var2/
6.  Obs /1*500/
7.  Subobs(obs) /1*10/
8.  Actobs(obs);
9.  Alias (subobs, subobs1)
10. $offlisting
11. Table act(obs,inout) input output table
12. $ondelim
13. $include "data.csv"
14. $offdelim
15. $onlisting
16. Variables
17. Theta efficiency score
18. Weight(obs) weights
19. Lambda(obs, var);
20. Positive variable weight, lambda;
21. Equations
22. Constr1(output, obs) DEA constraint for each output
23. Constr2(fixed, obs) DEA constraint for fixed inputs
24. Constr3(var, obs) DEA constraint for variable inputs;
25. Constr1(output, actobs)..
    sum(subobs,weight(subobs)*act(subobs, output)) =G=
    theta*act(actobs, output);
26. Constr2(fixed, actobs).. sum(subobs,
    weight(subobs)*act(subobs,fixed)) =L= act(actobs, fixed);
27. Constr3(var, actobs).. sum(subobs,
    weight(subobs)*act(subobs,var)) =E=
    lambda(actobs,var)*act(actobs,var);
28. Parameter
29. Score1(obs) theta estimates
30. Score2(obs,var) hold variable input levels;
31. File capdea /grcapres.txt/;
32. Model tedea /constr1, constr2, constr3/
33. Loop (subobs1,
34. Actobs(obs)=no;
35. Actobs(subobs1)=yes;
36. Option LP=OSL;
37. Solve tedea maximizing theta using LP;
38. Score1(subobs1)=theta.l;
39. Score2(subobs1,var)=lambda.l(subobs1,var);
40. Put capdea;
41. If ((tedea.modelstat eq 1 and tedea.solvestat eq 1),
42. Put @1, subobs1.tl, @10, "optimal", @20, "normal completion"/
43. Else put @1, subobs1.tl, @10, tedea.modelstat:>2:0, @20, tedea.
    solvestat:>2:0/)
44. );
45. Parameter capest(subobs1) capacity estimate;
46. Capest(subobs1)=sum(output,
    score1(subobs1)*act(subobs1,output));
47. File res /cap_inp.csv/;
48. Res.pc=5;
49. Res.pw=160;
50. Put res;
51. Put "obs", "theta", "capacity",
52. Loop (output, put output.tl);
53. Loop (var, put var.tl);
54. Put "var1o", "var2o"
55. Loop (subobs1,
56. Put /
57. Put subobs1.tl, score1(subobs1), capest(subobs1),
58. Loop (output, put act(subobs1,output));
59. Loop (var, put act(subobs1, var));
60. Loop (var, put score2(subobs1,var));
61. );
62. Putclose;
```

Figure 7. Example of a GAMS program to solve an output-oriented DEA model which measures capacity.

```
1.  Equations
2.  Constr1(output,obs) DEA constraint for each output
3.  Constr2(fixed,obs) DEA constraint for fixed inputs
4.  Constr3(var,obs) DEA constraint for variable inputs
5.  Constr4 DEA constraint for variable returns to scale;
6.  Constr1(output,actobs).. sum(subobs,weight(subobs)*act
    (subobs,output)) =G= theta*act(actobs,output);
7.  Constr2(fixed,actobs)..
    sum(subobs,weight(subobs)*act(subobs,fixed)) =L= act(actobs,fixed);
8.  Constr3(var,actobs).. sum(subobs,weight(subobs)*act(subobs,var))
    =E= lambda(actobs,var) *act(actobs,var);
9.  Constr4.. sum(subobs,weight(subobs)) =E= 1;
10. Model tedea /constr1,constr2,const
```

Figure 8. Example of programming steps which can adjust the GAMS program exemplified in Figure 7 in order to impose variable or nonincreasing returns to scale.

```
$oninline;
Set inout /spec1, spec2, fix1, fix2, var1, var2/
Output(inout) /spec1, spec2/
Input(inout) /fix1, fix2, var1, var2/
Fixed(inout) /fix1, fix2/
Var(inout) /var1, var2/
Obs /1*500/
Subobs(obs) /1*10/
Actobs(obs);
/*next, define an alias for the set subobs*/
Alias (subobs, subobs1)
$offlisting
Table act(obs,inout)  input output table
$ondelim
$include "data.csv"
$offdelim
$onlisting
Variables
Theta  efficiency score
Weight(obs)  weights
Lambda(obs, var);
Positive variable weight, lambda;
Equations
Constr1(output,obs)  DEA constraint for each output
Constr2(input,obs)  DEA constraint for all inputs
Constr3(fixed,obs)  DEA constraint for fixed inputs
Constr4(var,obs)  DEA constraint for variable inputs;
Constr1(output, actobs).. sum(subobs,weight(subobs)*act
(subobs,output)) =G= theta*act(actobs, output);
Constr2(input, actobs).. sum(subobs,
weight(subobs)*act(subobs,input)) =L= act(actobs, input);
Constr3(fixed, actobs).. sum(subobs,
weight(subobs)*act(subobs,fixed)) =L= act(actobs, fixed);
Constr4(var, actobs).. sum(subobs,
weight(subobs)*act(subobs,var)) =E=
lambda(actobs,var)*act(actobs,var);
Parameter
Score1(obs,*)  theta estimates
Score2(obs,var)  hold variable input levels
;
File effic /greff.txt/
Capmod /grcap.txt/;
Model tedea /constr1, constr2/
Model cap /constr1, constr3, constr4/
Loop (subobs1,
Actobs(obs)=no;
Actobs(subobs1)=yes;
Option LP=OSL;
Solve tedea  maximizing theta using LP;
Score1(subobs1,"TE")=theta.l;
Put effic;
If ((tedea.modelstat eq 1 and tedea.solvestat eq 1),
Put @1, subobs1.tl, @10, "optimal", @20, "normal completion"/
Else
Put @1, subobs1.tl, @10, tedea.modelstat:>2:0, @20,
Tedea.solvestat:>2:0/
)
);
Loop (subobs1,
Actobs(obs)=no;
Actobs(subobs1)=yes;
Option LP=OSL;
Solve cap maximizing theta using LP;
Score1(subobs1,"cap")=theta.l;
Score2(subobs1,var)=lambda.l(subobs1,var);
Put capmod;
If ((tedea.modelstat eq 1 and tedea.solvestat eq 1),
Put @1, subobs1.tl, @10, "optimal", @20, "normal completion"/
Else
Put @1, subobs1.tl, @10, tedea.modelstat:>2:0, @20,
Tedea.solvestat:>2:0/
)
);
Score1(subobs1,"CU")=score1(subobs1,"TE")/score1(subobs1,"cap");
Parameter capest(subobs1) capacity estimate;
Capest(subobs1)=sum(output, score1(subobs1,"cap")*act(subobs1,
output));
File res /caputil1.csv/;
Res.pc=5;
Res.pw=160;
Put res;
Put 'obs';
Loop (output,
Put output.tl);
Loop (input,
Put input.tl);
Put 'TE', 'capout', 'capacity', 'CU',
Loop (subobs1,
Put /
Put subobs1.tl,
Loop (output,
Put act(subobs1,output));
Loop (input,
Put act(subobs1,input));
Put score1(subobs1,"TE"), score1(subobs1,"cap"),
Capest(subobs1), score1(subobs1,"CU");
);
Putclose res;
```

Figure 9. Example of a GAMS program to solve an output-oriented DEA model which estimates technical efficiency, capacity output, and capacity utilization.

Table 1. An example of results from both an input- and output-oriented DEA model

Observations (Obs)	Outputs		Inputs				Technical Efficiency Measures	
			Fixed		Variable		Input-Oriented Model (Lambda)	Output-Oriented Model (Theta)
	Spec1	Spec2	Fix1	Fix2	Var1	Var2		
1	13295	27065	55	60	4	94	0.91	1.10
2	13255	10090	63	70	8	127	0.25	3.99
3	614	3427	59	59	6	35	0.31	3.24
4	106461	58705	63	69	5	185	1.00	1.00
5	3540	9130	53	60	5	46	0.62	1.60
6	602	6900	62	74	5	37	0.59	1.70
7	12920	18128	69	78	6	133	0.43	2.33
8	8312	5145	65	63	8	162	0.10	9.99
9	3276	4430	70	62	3	24	0.58	1.72
10	4143	8486	63	61	5	81	0.33	3.03

Table 2. An example of results from an output-oriented DEA model, including slack variables

Observations (Obs)	Technical Efficiency Measure (Theta)	Variables			
		Spec1	Spec2	Spec1 Slack	Spec2 Slack
1	1.10	13295	27065	39441	0
2	3.99	13255	10090	20143	0
3	3.24	614	3427	18151	0
4	1.00	106461	58705	0	0
5	1.60	3540	9130	20812	0
6	1.70	602	6900	20268	0
7	2.33	12920	18128	46458	0
8	9.99	8312	5145	10176	0
9	1.72	3276	4430	8179	0
10	3.03	4143	8486	34064	0

Table 3. An example of results from the capacity DEA model

Observations (Obs)	Capacity Measure (Theta)	Capacity	Outputs		Variable Inputs			
			Spec1	Spec2	Var1	Var2	Var1 Optimal	Var2 Optimal
1	1.89	76,280	13,295	27,065	4	94	1.09	1.71
2	5.82	135,868	13,255	10,090	8	127	0.63	1.46
3	14.65	59,201	614	3,427	6	35	0.71	4.52
4	1.00	165,166	106,461	58,705	5	185	1.00	1.00
5	5.41	68,545	3,540	9,130	5	46	0.84	3.38
6	8.37	62,792	602	6,900	5	37	0.98	4.92
7	3.55	110,220	12,920	18,128	6	133	0.91	1.52
8	10.42	140,222	8,312	5,145	8	162	0.57	1.04
9	11.91	91,778	3,276	4,430	3	24	1.50	6.93
10	6.12	77,289	4,143	8,486	5	81	0.88	2.02

Table 4. An example of results from output-oriented DEA models with different assumptions regarding returns to scale

Observations (Obs)	Nonincreasing Returns to Scale	Variable Returns to Scale	Constant Returns to Scale
1	1.10	1.00	1.10
2	3.99	3.86	3.99
3	3.24	1.00	3.24
4	1.00	1.00	1.00
5	1.60	1.00	1.60
6	1.70	1.10	1.70
7	2.33	2.27	2.33
8	9.99	5.34	9.99
9	1.72	1.00	1.72
10	3.03	2.71	3.03

Table 5. An example of capacity and capacity utilization based on an output-oriented DEA model

Observations (Obs)	Outputs		Technical Efficiency Measure	Capacity Output Measure	Capacity	Observed Capacity Utilization	Unbiased Capacity Utilization
	Spec1	Spec2					
1	13295	27065	1.10	1.89	76,280	0.53	0.58
2	13255	10090	3.99	5.82	135,868	0.17	0.69
3	614	3427	3.24	14.65	59,201	0.07	0.22
4	106461	58705	1.00	1.00	165,166	1.00	1.00
5	3540	9130	1.60	5.41	68,545	0.18	0.30
6	602	6900	1.70	8.37	62,792	0.12	0.20
7	12920	18128	2.33	3.55	110,220	0.28	0.66
8	8312	5145	9.99	10.42	140,222	0.10	0.96
9	3276	4430	1.72	11.91	91,778	0.08	0.14
10	4143	8486	3.03	6.12	77,289	0.16	0.50