Theory and practice of computer technologies used for creating DEA software

Eugene P. MORGUNOV
Siberian State Aerospace University, Krasnoyarsk, Russia
email: emorgunov@mail.ru

Abstract

It is obvious that effective employing DEA method is impossible without effective computer software. Various computer technologies may be used for creating DEA software. In the paper, the main features, advantages, and disadvantages of these technologies are examined. Among such technologies there are—using MATLAB, C language, database management systems, so called CGI-programming (it is used for installing DEA software on the Internet servers), etc.

One of the keynotes the paper has is demonstrating some practical programming techniques using different programming languages. The paper is accompanied by a bundle of specially compiled examples. These examples are simple enough to be understood without hard working on them.

Materials contained in the paper may be useful for various kinds of researchers, namely—

– for those who have sound background in computer programming field and might want to develop DEA software of their own;

– for scholars who don't plan to develop DEA software by themselves but want to get a good insight into how such software is being developed and functioning and what problems there may be on the way of developing DEA software;

– for everyone who has at least basic skills in computer programming and want to better understand DEA method by means of programming some simple DEA models.

Keywords: DEA; software development.

1. Introduction

Some preliminary notes about what this paper is not meant to serve for. The paper is not a textbook on computer programming or MATLAB programming. It is assumed that a user has some skills in programming with MATLAB. The paper is not a description of some real DEA software too.

When one decides to create software for solving DEA problems it is necessary to get acquainted with DEA software that is already accessible to a researcher—DEAP (Coelli, 1996), DEA-Solver (Cooper), OnFront (Färe), etc. Another important thing is to have in mind some questions. Decisions that may be made for these questions are very important. First of all we list these questions—

– choice of programming language;
– choice of operating system (OS);
– choice of database management system (DBMS);
– use of Internet technologies;
– use of special libraries of various mathematical sub-routines (for example, sub-routines for manipulating vectors and matrices).

It is necessary to take into account the goal one has attempting to create the DEA software. Possible goals may be—

– to develop professional DEA software and to distribute it among DEA researchers and practitioners;

– to study DEA method by means of programming some simple DEA models.

2. Overview of components for creating DEA software

2.1. Programming language

Various programming languages are used for creating DEA software. For example, T. Coelli's DEAP (Coelli, 1996) is written in FORTRAN, DEA-Solver (Cooper) is written in Visual Basic and must be run as an ordinary Excel file. Such programming language like FORTRAN has long time and rich traditions in the field of programming for scientific purposes. Nevertheless, we think that C/C++ language is worth to be looked at closely. This is because there is a very good instrument for visual developing software—Borland C++ Builder. Using this tool, user interface is becoming much easier to create.

2.2. Operating system

The state of things in computer world is characterized by dominance of Windows operating system. However, UNIX operating system has some important advantages, which make it possible to consider using this OS for running DEA software, We have strong evidence that Linux OS is becoming more and more popular last years.

2.3. Database management system

The choice of DBMS is crucial for professional DEA software that is intended for processing large amounts of data. The main advantage of DBMS is that a user has not to deal with lots of small files pertaining to every particular DEA study (data files, instruction files, DMUs files, etc.). Instead he can work within the database. A user can avoid repeatedly entering the same data many times.

The correct (proper) choice of DBMS has great influence on reliability of data storage and speed of processing. Last years there become more and more popular non-commercial DBMSs such as PostgreSQL and MySQL. These DBMSs are very reliable and have full set of basic features that every professional DBMS must have (including full support of SQL language). In contrast to commercial DBMSs (Oracle, Microsoft SQL Server, etc.) non-commercial DBMSs may be used free of charge accordingly to their license agreements.

2.4. Internet technologies

Using these technologies would allow to put DEA software on a server and to give access to the software through the Internet. Such an approach gives all advantages Internet can give—

– easiness of centralized updating the software with its new versions;

– possibility of solving large-scale problems for those users who don't have access to a powerful computer;

– possibility of renting the software without buying it (a user might get access to the software through Internet with password and need not to install the software on his/her personal computer).

2.5. Special libraries of various mathematical sub-routines

There are a number of such libraries that can be obtained on Internet. The main advantage of the libraries is that a programmer can incorporate programming code for typical tasks into his/her software. This approach can save some amount of time and make DEA software more reliable. One of such libraries is GNU Scientific Library (GSL). The GNU Scientific Library (GSL) is a numerical library for C and C++ programmers. It is free software under the GNU General Public License (see http://www.gnu.org). The library provides a wide range of mathematical routines such as random number generators, special functions and least-squares fitting. There are over 1000 functions in total.

Disadvantages of using scientific libraries may be a little slower speed of calculations and some dependence on the programming code that you are not an author of.

Now we proceed to the main part of the paper, namely—to making some guidelines for creating DEA software. At first, we concentrate on possible needs of a person who might want to study DEA more thoroughly by means of programming simple DEA problems.

3. Guidelines for a DEA user who would decide to program simple DEA models

First of all we need to choose the programming language. In order for your first steps in DEA programming be easier we would recommend to use MATLAB.

We suggest the following way of working. We begin by creating a very simple MATLAB program for solving a particular DEA problem that is taken from textbook by T. Coelli et al. (see pages 143–144). So running DEAP program can prove results that our program gives.

We will illustrate CRS input-oriented DEA model. There are five observations with one output and two inputs.

Table 1. Data for simple DEA problem

DMU	Y	X1	X2
1	1	2	5
2	2	2	4
3	3	6	6
4	1	3	2
5	2	6	2

3.1. First DEA MATLAB program

The text of the program is in DEA_example_1.m file. As DEA is based on linear programming (LP) method, the key action of our first simple program is to solve LP problem that is prepared employing data for DMUs under examination. This task is equivalent to creating matrix of constraints, vector of objective function, and vector of RHSs, and then calling MATLAB function—linprog(). Manipulations with matrices are very easy in MATLAB. The type of DEA problem we will solve is so called ‘envelopment’ form (see (Coelli, 1998)).

```
% ------------------------------------------------------------------------
% Program: simple CRS input-oriented DEA problem for one DMU
% Author:  Eugene P. Morgunov
% Date:    May 21, 2005
% ------------------------------------------------------------------------

% This example of solving DEA problem is taken from the file deap.pdf (p. 15)
```

```
% conditions -- CRS input-oriented problem, 5 DMUs, 2 inputs, 1 output
fprintf( 'Problem -- CRS input-oriented, 5 DMUs, 2 inputs, 1 output\n\n' );

% we have 6 variables -- 1 variable is theta
%                        5 variables are lambda's
% we name all them as x's (for simplicity) --
% so x( 1 ) == theta
%    x( 2 ) == lambda1
%    x( 3 ) == lambda2, etc.

% we must prepare parameters for linprog() function

% objective function
f = [ 1; 0; 0; 0; 0; 0 ];

% matrix of constraints is formed for DMU 3 (as in example in deap.pdf file)

% we should keep in mind that linprog() function demands that signs of
% constraints be '<=' (and not '>=') so we can multiply constraints,
% that are in the form of inequalities, by -1. That's why values for outputs
% gain minus sign;
% the first column in this matrix represents values for DMU that is being
% optimized (DMU 3 in this case)
A = [  0 -1 -2 -3 -1 -2;
      -6  2  2  6  3  6;
      -6  5  4  6  2  2
    ];

% column of RHSs
% minus sign by value of y appears because of multiplying constraint
% by -1
b = [ -3; 0; 0 ];

% lower bounds on decision variables are zeros
lb = zeros( 6, 1 );

% call to linprog function for solving linear problem
% we don't have constraints in the form of equalities so pass empty matrices
x = linprog( f, A, b, [], [], lb);

% display results
fprintf( 'Results of computations\n\n' );
disp( x );

% the value of efficiency measure that DMU 3 has
fprintf( 'Efficiency measure (Value of theta) = %f\n\n', x( 1, 1 ) );

fprintf( 'Reference DMUs and their weights\n\n' );

% reference DMUs and their weights
for i = 2 : 6   % lambda's have numbers from 2 to 6
  if ( x( i ) > 0.000001 )  % we use 0.000001 instead of zero
    % (i - 1) -- because the first lambda has number 2, etc.
    fprintf( 'DMU %d   Weight = %5.4f\n', i - 1, x( i ) );
  end
end
```

This program can compute efficiency score for only one DMU at a time. So in order to obtain efficiency scores for all DMUs in a set, we need to make some changes in the program. First we replace rows 2 and 3 in matrix A with values of inputs of the next DMU, then we replace row 1 in vector of RHSs with value of output of the next DMU. Our data set has 5 DMUs, so we have to make changes five times. Obviously, it is not very convenient.

Now let’s make our program a little more smart.

3.2. Second DEA MATLAB program

We extend our program by adding option to choose between CRS and VRS assumptions and by organizing the loop for processing all DMUs one by one. So the extended version of the simple DEA program is below (see DEA_example_2.m file).

```
% ------------------------------------------------------------------------
% Program: simple CRS/VRS input-oriented DEA problem
% Author:  Eugene P. Morgunov
% Date:    May 22, 2005
% ------------------------------------------------------------------------

% This example of solving CRS input-oriented DEA problem is taken
% from the file deap.pdf (p. 15) and extended for VRS case

% we have 6 variables -- 1 variable is theta
%                        5 variables are lambda's
% we name all them as x's (for simplicity) --
% so x( 1 ) == theta
%    x( 2 ) == lambda1
%    x( 3 ) == lambda2, etc.

clear;

% scale assumption
%scale = 'VRS';
scale = 'CRS';

fprintf( 'Scale assumption is %s\n\n', scale );

% objective function
% only coefficient by theta is 1 and coefficients by lambda's are zeros
f = [ 1; 0; 0; 0; 0; 0 ];

% this fragment can be deleted
% matrix of constraints is formed for DMU 3
% A = [  0 -1 -2 -3 -1 -2;
%       -6  2  2  6  3  6;
%       -6  5  4  6  2  2
%     ];

X_orig = [ 2 2 6 3 6;
           5 4 6 2 2
         ];

Y_orig = [ 1 2 3 1 2 ];

% we make all elements of this vector to be negative because of type
% of constraint for Y's (outputs) -- this constraint is '>=', but we need
% '<=' for using linprog(). So we muptiply both parts of the constraint by -1
% and invert '>=' for '<='
Y_orig_negative = Y_orig * (-1);

% lower bounds on decision variables
lb = zeros( 6, 1 );

% if scale assumption is VRS then we need one more constraint --
% for lambda's (their sum equals to 1)
if scale == 'VRS'
  % zero at the first position -- because coeff. by theta in the constraint
  % for lambda's is zero
  Aeq = [ 0 1 1 1 1 1 ];
  beq = [ 1 ];            % sum of lambda's equals to 1
```

```
end

% matrix of results
results = [];

for i = 1 : 5      % we have 5 DMUs
  fprintf( 'DMU number %d\n', i );
  fprintf( '---------------------\n\n' );

  % matrix of constraints is formed for every DMU

  % we begin with matrix of outputs (y's)
  Y = [ 0                      Y_orig_negative ];

  % the first column in X matrix is x's, multiplied by -1,
  % of current (being evaluated) DMU
  X = [ X_orig( :, i ) * (-1)  X_orig ];

  % now we combine constraints for y's and x's into matrix A
  A = [ Y; X ];

  % column of RHSs
  % constraint for Y's will have the value of y for evaluated DMU,
  % multiplied by -1, as RHS
  % RHSs for x's are zeros
  b = [ Y_orig( i ) * (-1); zeros( 2, 1 ) ];

  % call to linprog() function for solving linear problem
  if scale == 'CRS'
    % we don't have constraints in the form of equalities so pass empty ma-
trices
    x = linprog( f, A, b, [], [], lb );       % CRS
  elseif scale == 'VRS'
    x = linprog( f, A, b, Aeq, beq, lb );     % VRS
  end

  disp( x );

  % the value of efficiency measure that current (being evaluated) DMU has
  fprintf( 'Value of theta = %f\n\n', x( 1, 1 ) );

  % accumulate results
  results = [ results   x( 1, 1 ) ];

  % reference DMU's and their weights
  for j = 2 : 6   % lambda's have numbers from 2 to 6
    if ( x( j ) > 0.000001 )  % we use 0.000001 instead of zero
      % (j - 1) -- because the first lambda has number 2, etc.
      % fprintf( 'DMU %d   Weight = %20.18f\n', j - 1, x( j ) );
      fprintf( 'DMU %d   Weight = %5.4f\n', j - 1, x( j ) );
    end
  end
  fprintf( '---------------------\n\n' );
end

% print all efficiencies together
for i = 1 : length( results )
  fprintf( 'DMU %d has efficiency %f\n', i, results( i ) );
end
```

This version has one great drawback. Data for a DEA problem is incorporated into the source code. If dataset changes new data must be entered into the source code anew. So the next

step of improving the program is to add support of reading data from a text file. Support for variable number of DMUs and inputs/outputs will be added also.

3.3. Third DEA MATLAB program

In accordance with the principle of modular program design, we place programming code for reading data file into separate source file. Many DEA researchers know about DEAP program by T. Coelli, so we assume the same data file format, which DEAP assumes—columns for outputs come first, then go columns for inputs. Now dataset can be read from a data file by simply calling get_data() function in this manner—

```
[ X, Y ] = get_data( data_file, DMUs_num, inputs_num, outputs_num );
```

```
% Function for getting data from a text file
% Parameters
%         data_file - the name of a text file containing data for all DMUs;
%         DMUs_num - number of DMUs;
%         inputs_num - number of inputs;
%         outputs_num - number of outputs.
% Return value - vector of two elements:
%         X - matrix of inputs;
%         Y - matrix of outputs
```

This function is placed into separate source file— get_data.m.

In order to make this simple program more stable and convenient for a user, we include support for a separate instruction file. So a user can put all necessary instructions into this file and not touch MATLAB source code at all. All output will go into output file.

In order to conserve space we don't incorporate all source code of this version of DEA program into the text of the paper. See file DEA_example_3.m, which is in a bundle with the paper.

3.4. Some tips for further developing this simple program

The next step may be adding support for varying orientation of the model, because only input-oriented case is implemented now. Another important step may be adding functionality for calculation slacks, radial movements, and projected values for every DMU.

3.5. Brief discussion

As it can be seen from looking at MATLAB source codes, manipulating with data is rather easy and a user can concentrate on essence of DEA method. When a DEA program is implementing with C language a user must devote much time to realizing various auxiliary functions such as allocating computer's memory for matrices, etc. For example, how may look a fragment of C code for creating a simplex matrix—

```
  /* simplex matrix will be organized as array of pointers to arrays
     of doubles */
  double **matrix;

  /* array of pointers to doubles */
  matrix = ( double ** )malloc( *rows * sizeof( double * ) );

  /* now create subarrays of doubles and combine all of them in one
     two-dimensional array */
  for ( i = 0; i < *rows; i++ )
  {
    matrix[ i ] = ( double * )malloc( *cols * sizeof( double ) );
```

```
}
```

Of course, programming with MATLAB can give a user good insight into how DEA works. However, if speed of processing is critical then there is only one solution—to develop DEA program using languages like C, C++, etc. And now we proceed to the next part of the paper.

4. Guidelines for a computer programmer who would decide to develop DEA software

Professional DEA software may be developed using various programming tools described above. If one would use Cartesian product of all sets of different kinds of tools he/she would get a lot of combined technologies some of which are viable and can be used to develop real DEA software. In this paper, for conserving space we will not discuss all possible combinations of different programming tools. Instead, we go straightly to discussing two of them, namely—the first, desktop DEA software based on OS Windows and Borland C++ Builder products, and the second, Internet DEA software based on professional relational DBMS (such as PostgreSQL), C and Perl (or PHP) languages, and CGI-technology.

4.1. Desktop DEA software

This kind of DEA software is a research tool for an individual researcher or practitioner. So the software is to be installed on a personal computer.

The software may be subdivided onto three main subsystems—*kernel*, *database*, and *user interface*. The main part is, of course, kernel.

4.1.1. Kernel

*Kerne*l is designed for realizing DEA models and all other auxiliary mathematical processing, e.g., correlations, clusterizations, etc. It seems to be wise, if kernel would be implemented in C language. This language has many extensions incorporated in particular compilers by various firms (Borland, Microsoft, etc.). It is known from theory of software engineering that portability is a very important feature of modern software. Portability means that in order to port (to move, simply saying) some software to different operating systems, it is sufficient only to recompile software in environment of target operating system. Such a feature is very convenient because nowadays we can see that OS Linux becomes more and more popular. So it is very likely that some users would like to run DEA software on Linux systems and not on Windows systems. If this is the case, kernel must be implemented in C language, and also using ANSI subset of C language only. So every C compiler will be able to compile such source codes without need to edit them before.

4.1.2. Database

The next part of DEA software is *database*. Until recent times all accessible DEA software didn't use real databases. Instead, all data and all results were stored in a lot of files. Sometimes these files are just ordinary text files as in DEAP program (Coelli, 1996), sometimes they have special structure as in OnFront program (Färe), and sometimes they are Excel files as in DEA-Solver software (Cooper). But general idea is that a user has to know names of all these files, a user has to manipulate with these files 'by hands' using operating system's standard tools for working with files and directories. It is not very convenient when it comes to hundreds of files. Natural way to solve this problem is to use a database. A database can help a user to avoid repeatedly entering the same data again and again. A database can eliminate many errors that arise from an operator's mistakes and can ensure consistency of data by means of various checks of data being entered. So quality of data and convenience in working with them may be increased significantly.

With use of databases there may be implemented so-called *repository* of data. The main idea of repository is to have one center of information for all studies a researcher would make. The same objects (DMUs, firms, etc.) and variables may take part in different studies so it seems to be very convenient for users of the software to enter detailed info about any object or any variable into the database *only once*. While implementing a particular study a user has only to include an object's ID (identifier) or a variable's ID into the set of objects or variables for this study. So a user may have possibility to implement some investigations including several different studies that include the same (or similar to some degree) sets of objects and variables. Such an approach may be rather informally called 'cross-studying' or 'inter-studying'. For example, a researcher might want to know how a particular object performed in two or three (or more) different periods of time (given that these periods were studied separately). Manipulating with unique identifiers of objects and variables instead of their descriptions will reduce risk of mistakes and will save a user of the software from excessive typing.

When a researcher needs to study multilevel hierarchies of objects (e.g., big transnational corporations) a database can store inter-level relations between objects in consistent form. So a user will be saved from having lots of data files for every group of objects at every level in hierarchy.

As a physical format of relational database tables, format of Paradox tables may be chosen. This format has rather good support of basic database features, such as primary and foreign keys, referential integrity, indices, default values for database fields, etc. Short guidelines for tables' logical structure might be as follows—

Table «Studies' descriptions»
Attributes:
- **Study's identifier**
- Study's name
- Study's description
- Date of implementing a study
- Name of a chief researcher

Table «Descriptions of all DMUs»
Attributes:
- **Object's identifier**
- Object's short name
- Object's full name

Table «List of all variables»
Attributes:
- **Variable's identifier**
- Variable's short name
- Variable's full name

Table «Objects included in studies»
Attributes:
- **Study's identifier**
- **Object's identifier**

Table «Variables included in studies»
Attributes:
- **Study's identifier**
- **Variable's identifier**
- Type of a variable (input or output)

Table «Repository of data for all objects»
Attributes:

Object's identifier
Variable's identifier
Value of a variable
Number of a period this value is from
Date for this period (if any)
Description of this value

Table «Parameters of studies»
Attributes:

Study's identifier
Output file name
Total count of DMUs
Count of time periods
Count of input variables
Count of output variables
Scale assumption (CRS, VRS, NIRS, NDRS)
Orientation (input or output)

Table «Data for studies»
Attributes:

Study's identifier
Object's identifier
Variable's identifier
Raw value of a variable (as it was taken from the repository)
Value of a variable (values of variables may be pre-processed in some way before using them in a particular DEA study)
Number of time period
Date this value was obtained for
Description of this value

Note. Key attributes that serve for creating relations (links) between tables are shown in boldface.

This schematic structure of database tables is only a sketch and not a real structure. So not all tables and not all attributes of tables are present. Nevertheless, even this simplified structure can give some insight into how might a database look from inside.

4.1.3. User interface

And the last subsystem of DEA software is *user interface*. This part of DEA software may be implemented in Borland C++ Builder. This development tool belongs to the class of so-called RAD—rapid application development tools. All basic interface elements can be created with help of a mouse and almost without writing source code by hand. Of course, when it comes to creating non-standard elements of user interface, a mouse is of little help. But nevertheless, a programmer can create a first version of user interface in a relatively short time. Another pleasant thing is that modern version of Borland C++ Builder – C++ BuilderX supports development in UNIX environment. So needs of potential UNIX users can be taken into account.

4.2. Internet DEA software

This sort of DEA software is meant to be installed on Internet web-servers. Those may benefit from such a technology who don't want—for some reason — to install desktop DEA software on their personal computers or don't have computers powerful enough to process large amounts of data.

The main subsystems of this software are the same as for desktop one—*kernel*, *database*, and *user interface*. Kernel, written in C language, may be adopted from desktop variation of the software. Operating system may be not only Windows but also UNIX system (e.g. Linux or FreeBSD). Database logical structure may be the same too, but use of relational database management system (DBMS) is strongly recommended. Commercial DBMSs such as Oracle or DB2 are very expensive and need rather powerful computer for installing. Instead, non-commercial DBMSs can be used—PostgreSQL or MySQL. They are very reliable and efficient and have good conformance with SQL standards.

Installing DEA software on Internet server is impossible without special software—so-called *http-server* (www-server). This software provides connection between user's Internet browser and DEA software installed on Internet server. This sort of computer architecture is known under the name 'client–server'. So DEA software plays the role of server and user's Internet browser plays the role of client. In this case, a user needn't install any additional software on his/her computer because Internet browser is enough. As a good non-commercial www-server, Apache may be recommended. It has variations for Windows and UNIX platforms.

User interface may be created using Perl or PHP languages. These languages are so-called *scripting languages*. They have special means in order to communicate with DBMS and allow a computer programmer to create friendly user interface that is controlled from within an Internet browser. This communications are organized through so-called CGI-interface.

As a simple schematic illustrative example of Internet DEA software the following one may be suggested—well known DEAP program (Coelli, 1996) plays the role of kernel and database, and user interface may be written in Perl language (see files accompanying the paper).

5. Conclusions

This brief discussion and short examples cannot serve as a full-length project for development of DEA software. The aim of the paper was only to outline basic principles and show the most popular computer technologies that may be combined in order to create DEA software. The long-term objective might be to bring DEA software to that very high level of maturity, which is already achieved in DEA theory. Of course, this objective can be achieved only with joint efforts of DEA community.

References

Coelli, T.J. (1996), A Guide to DEAP Version 2.1: A Data Envelopment Analysis (Computer) Program, CEPA Working Paper 96/8, Department of Econometrics,University of New England.

Coelli, T. (1998), An Introduction to Efficiency and Productivity Analysis [Text] / T. Coelli, D. S. Prasada Rao, G. E. Battese. – Boston : Kluwer Academic Publishers, 1998. – 275 pp.

Cooper, W. W. (2000), Data Envelopment Analysis [Text] : A Comprehensive Text with Models, Applications, References, and DEA-Solver Software / W. W. Cooper, L. M. Seiford, K. Tone. – Boston : Kluwer Academic Publishers, 2000. – 318 pp.

Färe, R. OnFront: The Professional Tool for Efficiency and Productivity Measurement./ R. Färe, S. Grosskopf. – http://www.emq.com.