Data Envelopment Analysis using glpkAPI in R

Konrad Miziolek
Portland State University

Jordan Beary
Portland State University

Shreyas Vasanth
Portland State University

Surekha Chanamolu
Portland State University

Rudraxi Mitra
Portland State University

ETM

ENGINEERING & TECHNOLOGY MANAGEMENT

Title: Data Envelopment Analysis using glpkAPI in R

Course Title: ETM – Operations Research

Course Number: 540

Instructor: Dr. Tim Anderson

Term: Fall

Year: 2017

Author(s): Konrad Miziolek, Jordan Beary, Shreyas Vasanth, Surekha Chanamolu, Rudraxi Mitra

ETM OFFICE USE ONLY

Report No.:	
Type:	Student Project
Note:	

Abstract

The work done here is primarily a wrapper function written to separate some of the more difficult-to-use glpkAPI functionality from the end-user. The user, when prompted, selects the appropriate configuration of the .mod file to the task (for example, output-oriented CRS), and the data file, as a .dat. The function then loads the required glpkAPI library, and carries forward the model. It allocates the problem and workspace, reads the model file and data file the user selects, builds the problem, and solves it. The function returns primal values, and, if dual = TRUE is selected, also returns dual weights.

Using R package glpkAPI for Data Envelopment Analysis

Konrad Miziolek, Jordan Beary, Shreyas Vasanth, Surekha Chanamolu, Rudraxi Mitra

Thursday, December 7 2017

GitHub

For all project materials please visit our GitHub[1]

[1] Download Materials GitHub

Purpose:

The purpose for this project was to use the R package 'glpkAPI' for DEA models which have been programmed in gmpl. This library is optimal for DEA models using .mod files; using glpkAPI for manually building the model and data from scratch is an arduous task better suited for other packages such as Benchmarking. Likewise, editing the model is easier done by editing the .mod file in an editor such as Gusek, or even R, than it is through glpkAPI.

The wrapper function we have created allows for gmpl DEA models to use the glpkAPI interface to be solved by GLPK.

However, our wrapper function enables one to input data into an R dataframe for inputs and outputs and calculate the efficiencies and dual weights by modifying the associated .mod and .dat files.

The .mod files are separate from the data, specified in a .dat format. This decoupling allows for a general-use .mod file for running the same DEA problem with different datasets by specifying the data and permutations of the model (IE input-oriented VRS, input-oriented CRS, etc). The user will specify which permutations are to be used.

The work done here is primarily a wrapper function written to separate some of the more difficult-to-use glpkAPI functionality from the end-user. The user, when prompted, selects the appropriate configuration of the .mod file to the task (for example, output-oriented CRS), and the data file, as a .dat. The function then loads the required glpkAPI library, and carries forward the model. It allocates the problem and workspace, reads the model file and data file the user selects, builds the problem, and solves it. The function returns primal values, and, if dual = TRUE is selected, also returns dual weights.

The primal values are formatted as a list of thetas and as a lambda matrix, and duals as a list of duals and a dual matrix. These objects are returned to the user as dataframes.

DEA - Brief Review

Data Envelopment Analysis (DEA) is a non-parametric analytical methodology used for efficiency analysis. The primary elements of DEA are a set of decision making units (DMUs) along with their measured inputs and outputs. In addition to the efficiency value of each Decision-Making Units (DMU), DEA also provides benchmarking information, which can be used to improve the efficiency of the DMU. The DMUs may be different branches of the small large bank or different hospitals or a project.

DEA produces a single comprehensive measure of performance for each of DMUs. The best ratio among all the DMUs is the benchmarking target, which is the most efficient one and it would identify other DMUs that would be rated by comparing the ratio to the best one. The two kinds of information, the efficiency level and the benchmarking information, are inseparable. The efficiency is measured based on the distance between the observed DMU and the reference DMU, which serves as a benchmarking target.

Writing .mod and .dat files in GMPL

GNU Mathematical Programming Language or MathProg is the native language of GLPK. GMPL is a flexible language and gives the programmer the ability to write low and high-level math programs. This is simple example to get familiar with GMPL syntax.

```
var x1;
var x2;
maximize obj: 0.6 * x1 + 0.5 * x2;
s.t. c1: x1 + 2 * x2 <= 1;
s.t. c2: 3 * x1 + x2 <= 2;
solve;
display x1, x2;
end;
```

For more insight into programming with GMPL please refer to Andrew Makhorin's GMPL Introductory Manual[2] as we will only be discussing the specifics of our DEA mod and dat files.

[2] See Makhorin GMPL manual PDF

.mod file

The mod file is written in a way that the user does not need to edit to fit their specific DEA problem. However, if adjustments need to be made it is important you, as a user, understand how this file is structured. The model is comprised of five objects: sets, parameters, variables, constraints, and objectives.

There are two different kinds of statements in GMPL - declaration and functional. The set statement is written first and sets are always the declaration type. We name the objects of a DEA problem: DMUs, inputs, and outputs. These are symbolic names for the actual field names. For example 'baseball players' are DMUs, 'at bats' are inputs, and 'number of hits' are outputs.

Parameter statements are also always declaration type. In the mod file the parameters are generalized to 'input_data' and 'output_data' inluding the units referenced for each - {dmus,inputs} and {dmus,outputs}. Parameters always include an expression to satisfy. In our model the input and output data must be greater than zero.

The program follows the parameter statements, which holds the variables, objective function, and set of constraints. Thetas and lambdas are the result variables involved in a DEA problem and they must be declared before the objective function. We have created two versions of the mod file depending on whether the user's specific problem is input-oriented or output-oriented. The constraints subject the objective function to benchmarking between the DMUs and finish with a returns-to-scale (RTS) statement. Our wrapper function injects the correct RTS depending on the users specification in R.

.dat file

Compared to the mod file the dat files are simple to structure. Similar to declaring the sets in the mod file, set statements are the first declaration in a dat file. Currently a number is assigned to each DMU - the number of DMUs in the user data is stated in the set for dmus. For now it is a vector of numbers, but for future work we will be able to use the DMU's natural name.

set dmus := 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15;

For inputs and outputs the user must declare the field names including in each set. Continuing with the baseball example - 'atbats' and 'numberofhits' are the input and output sets. We found errors when there was whitespace in the set names.

set inputs := AvailableTonKm OperatingCost NonflightAssets ;

set outputs := RevenuePassengerKm NonpassengerRevenue ;

The same parameters declared in the mod file are referenced again before each input and output data list.

Warnings

This function is dependent on glpkAPI continuing to return primal values as a list of thetas and lambda matrix. If that is changed downstream, this function will need to be edited.

Mod files require a DMU number in each row with the data, while Benchmarking, TFDEA (and possibly other packages) treat those as inputs, causing results to not match. Input data for those libraries with a column for DMU number will work, but outputs with a column for DMU number will not work.

(

)

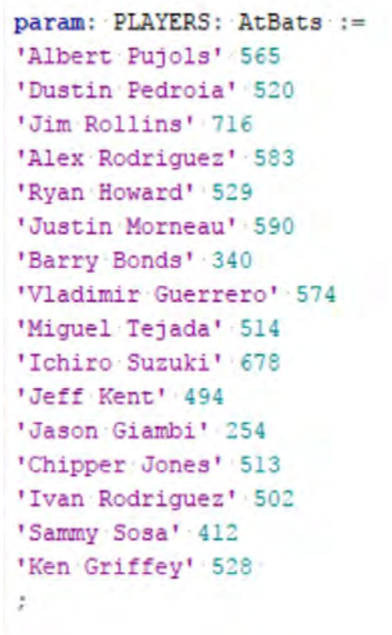
Above is an example in the dat file of DMU's (player names) that are in their own column before input/output values, which will cause errors in the final output values.

The architecture of this function will need to be changed once this is built into a package. At the moment, there is only one mod file that is overwritten between uses; for robustness, a local copy of the model should be made to avoid the possibility of corruption of the file. Without this package, it is not possible to write out the file locally. With a package, one could save a new copy of the file in the folder.

Likewise, if there is a need for future users to type in the data with R, the simplest method we could come up with is typing in the data as a dataframe and writing it out locally. Then, using the readLines function and adding the necessary lines for a .dat file, it can be again written out locally, then passed in to our glpkAPIDEA function.

Therefore, we suggest not using this function for that purpose. Instead, it is simpler to create the data in a text editor or spreadsheet software (Excel, Google Sheets) and copying the data over into a .dat file.

data ex.JPG data ex.bb



```
param: PLAYERS: AtBats :=
'Albert Pujols' 565
'Dustin Pedroia' 520
'Jim Rollins' 716
'Alex Rodriguez' 583
'Ryan Howard' 529
'Justin Morneau' 590
'Barry Bonds' 340
'Vladimir Guerrero' 574
'Miguel Tejada' 514
'Ichiro Suzuki' 678
'Jeff Kent' 494
'Jason Giambi' 254
'Chipper Jones' 513
'Ivan Rodriguez' 502
'Sammy Sosa' 412
'Ken Griffey' 528
;
```

Figure 1: wrong DMU format

glpkAPI Wrapper Function (glpkAPIDEA).

glpkAPI arguments:

1. **Mod_path**, the file path to the mod file
2. **Data_path**, the file path to the data file
3. **Returns To Scale (RTS)**: One of

- "crs"
- "vrs"
- "irs" (not tested yet)
- "drs" (not tested yet)

Arguments are case-insensitive. More RTS possibilities can be added later without much difficulty.

4. **Orientation**: Specify whether this is an input or output-oriented model. One of

- "in"
- "out" (not tested yet)

Arguments are case-insensitive.

5. **Dual**: Whether to extract the dual weighs (shadow prices).

   Default is FALSE.

Required libraries:

glpkAPI (loaded by our function, but not installed)

Required inputs:

1. .mod file: one of the input-oriented or output-oriented .mod files.
2. .dat file (included, or made by the user). This file needs to include both inputs and outputs for the DEA analysis. Text files can be saved as .dat, and aside from a bare skeleton set of required text in the file, they are easy to run.

These files are specified by the user but can also be done with a file.choose() call.

glpkAPIDEA outputs:

Currently, the function supports efficiency scores, the lambda matrix, and dual weights.

Extracting input and output weights from the glplAPI model object may be possible; we didn't look at this.

```
glpkAPIDEA <- function(Mod_path, Data_path, RTS, Orientation, Dual = FALSE)
{
  # Steps:
  #
  #   1. Data Handling
  #
  #     a. If X and Y are null (data is in a .dat file somewhere)
  #        User chooses the .dat file
  #     b. If X and Y are not null (user wants to pass in their dataframe)
  #        Have the user choose the default .dat file
  #        Read in the .dat file and replace a string with the data
  #        Write the data to the default file
  #
  #       Note: 1. a. is currently done outside of this wrapper function
  #             to ensure the data is read in properly.
  #       Note: 1. b. is not yet supported.
  #
  #   2. Model Building
  #
  #     a. Read in the .mod file (input vs output mod files)
  #     b. Check which type of parameter RTS is desired
  #        Replace the string in .mod file that affects which set of constraints are used
  #
  #   3. Model Running
  #
  #     * Create the problem and workspace (see glpkAPI vignette)
  #     * Load the model
  #     * Load the data
  #     * Solve the problem

  #   4. Results
  #
  #     * Check if Dual = TRUE
  #       * If yes, get dual weights
  #
  #     * Get the Primal values
  #     * Decompose into Thetas and lambda matrix

  # Step 1
  # a.
  # b.

  # Step 2

  mod_filepath <- Mod_path #User passes in the mod and .dat filepaths.
  data_filepath <- Data_path

  rts_string <<- toupper(RTS) #provides some robustness against user inputs
  orientation_string <- toupper(Orientation)

  ### Returns to Scale strings to replace the "### RTS" character string in our mod file

  if (rts_string == "CRS")
  {
    rts_replace_string <<- "s.t. PI1{td in dmus}: sum{d in dmus} lambda[d,td] >= 0;"
  }

  if (rts_string == "VRS")
  {
    rts_replace_string <<- "s.t. PI1{td in dmus}: sum{d in dmus} lambda[d,td] = 1;"
  }

  if (rts_string == "DRS")
  {
    rts_replace_string <<- "s.t. PI1{td in dmus}: sum{d in dmus} lambda[d,td] >= 1;"
  }

  if (rts_string == "IRS")
  {
    rts_replace_string <<- "s.t. PI1{td in dmus}: sum{d in dmus} lambda[d,td] <= 1;"
  }


  # Step 2

  ### Read in the file

  mod_file <- readLines(mod_filepath)
  mod_file <- gsub(pattern = "### RTS Constraint", replace = rts_replace_string, x = mod_file)
  writeLines(mod_file, con= mod_filepath)

  # Step 3.

  ### Creating the glpkAPI model, as in the glpkAPI vignette.


  mip <- initProbGLPK()
  setProbNameGLPK(mip, "DEA Example")
  dea <- mplAllocWkspGLPK()
  ### Since the model and data are in separate files, it is necessary
  ### to read in both.
  result <- mplReadModelGLPK(dea, mod_filepath, skip=0)
  mplReadDataGLPK(dea, data_filepath)

  result <- mplGenerateGLPK(dea)
  result <- mplBuildProbGLPK(dea, mip)

  solveSimplexGLPK(mip)
  mplPostsolveGLPK(dea, mip, GLP_MIP)
  solution_list <- getColsPrimGLPK(mip)

  mod_file <- readLines(mod_filepath)

  mod_file[25] <- "### RTS Constraint" # Based on our file structure,
  ### line 25 is where our RTS constraint is.

  writeLines(mod_file, mod_filepath)
```

```
  # Step 4.

  # solution list returned by glpkAPI as list where first x elements are DMUs and the
  # next x^2 elements are  the lambda matrix as a list.
  # To get the number of DMUs, one needs to solve a simple quadratic equation
  # of the form x^2 + x = c for the positive root of x. x^2 + x = length of solution, so
  # for a = 1, b = 1, -c = length of solution. x = (-b + sqrt (b^2 - (4 *1 * -c) ) / 2a

  c <- length(solution_list)
  dmus <- (-1 + sqrt(1 + 4*c))/2

  if (Dual == "TRUE")
  {
    duals_solution_list <- getColsDualGLPK(mip)

    duals <- duals_solution_list[1:dmus]
    duals_mat_list <- round(duals_solution_list[dmus+1:length(duals_solution_list)], 6)

    duals_list <<- data.frame(c(1:dmus), duals_solution_list)
    duals_matrix <<- data.frame(matrix(duals_solution_list, nrow = dmus, ncol = dmus))
  }

  #first n elements are efficiency scores
  theta_list <<- solution_list[1:dmus]

  #rest next n^2 elements belong to n x n lambda matrix
  lambda_list <<- round(solution_list[dmus+1:length(solution_list)], 6)

  thetas <<- data.frame(c(1:dmus), theta_list)
  colnames(thetas) <<- c("DMU", "Efficiency")

  lambdas <<- data.frame(matrix(lambda_list, nrow = dmus, ncol = dmus))
  colnames(lambdas) <<- 1:dmus
  rownames(lambdas) <<- 1:dmus
}
```

Reproducible examples

Benchmarking package produces the same results as our use of the glpkAPI model.

Code and associated files will probably be moved to GitHub for the final report.

Airline efficiency scores (PDX ETM Data Repository)

```
mod_filepath <- source_mod_file_from_github
data_filepath <-  source_airline_dat_file_from_github

glpkAPIDEA(mod_filepath, data_filepath, RTS = "crs", Orientation = "in", Dual = FALSE)

airline_inputs <- source_airline_inputs_from_github
airline_outputs <- source_airline_outputs_from_github

benchmarking_dea <- dea(airline_inputs, airline_outputs, RTS = "crs", ORIENTATION = "in")
tfdea_dea <- DEA(airline_inputs, airline_outputs, rts = "crs", orientation = "input")
```

```
## Warning, data has DMU's with outputs that are zero, this may cause numerical problems
```

```
results <- cbind(thetas, benchmarking_dea$eff, tfdea_dea$eff)
colnames(results) = c("DMU", "Our Tool", "Benchmarking", "TFDEA")
results
```

```
##    DMU  Our Tool Benchmarking     TFDEA
## 1    1 0.8542026    0.8542026 0.8542026
## 2    2 0.8444781    0.8444781 0.8444781
## 3    3 0.9475184    0.9475184 0.9475184
## 4    4 0.9417881    0.9417881 0.9417881
## 5    5 1.0000000    1.0000000 1.0000000
## 6    6 0.9765782    0.9765782 0.9765782
## 7    7 1.0000000    1.0000000 1.0000000
## 8    8 0.9030706    0.9030706 0.9030706
## 9    9 0.7793639    0.7793639 0.7793639
## 10  10 0.7855413    0.7855413 0.7855413
## 11  11 0.9080609    0.9080609 0.9080609
## 12  12 0.9348293    0.9348293 0.9348293
## 13  13 0.9215399    0.9215399 0.9215399
## 14  14 1.0000000    1.0000000 1.0000000
## 15  15 1.0000000    1.0000000 1.0000000
```

Gusek DEA Example

```
data_filepath <- source_dea_dat_file_from_github

glpkAPIDEA(mod_filepath, data_filepath, RTS = "vrs", Orientation = "in", Dual = FALSE)

dea_inputs <- source_dea_inputs_from_github
dea_outputs <- source_dea_outputs_from_github

benchmarking_dea <- dea(dea_inputs, dea_outputs, RTS = "vrs",  ORIENTATION="in" )
```

```
tfdea_dea <- DEA(dea_inputs, dea_outputs, rts = "vrs", orientation = "input")

results <- cbind(thetas, benchmarking_dea$eff, tfdea_dea$eff)
colnames(results) = c("DMU", "Our Tool", "Benchmarking", "TFDEA")
results
```

```
##       DMU  Our Tool Benchmarking     TFDEA
## DMU1    1 1.0000000    1.0000000 1.0000000
## DMU2    2 0.9860181    0.9860181 0.9860181
## DMU3    3 0.8523087    0.8523087 0.8523087
## DMU4    4 1.0000000    1.0000000 1.0000000
## DMU5    5 1.0000000    1.0000000 1.0000000
## DMU6    6 1.0000000    1.0000000 1.0000000
## DMU7    7 1.0000000    1.0000000 1.0000000
## DMU8    8 1.0000000    1.0000000 1.0000000
## DMU9    9 1.0000000    1.0000000 1.0000000
## DMU10  10 0.7965913    0.7965913 0.7965913
## DMU11  11 0.8959897    0.8959897 0.8959897
## DMU12  12 1.0000000    1.0000000 1.0000000
## DMU13  13 1.0000000    1.0000000 1.0000000
## DMU14  14 1.0000000    1.0000000 1.0000000
## DMU15  15 0.6152070    0.6152070 0.6152070
## DMU16  16 0.9678167    0.9678167 0.9678167
## DMU17  17 1.0000000    1.0000000 1.0000000
## DMU18  18 1.0000000    1.0000000 1.0000000
## DMU19  19 0.6957355    0.6957355 0.6957355
## DMU20  20 1.0000000    1.0000000 1.0000000
## DMU21  21 1.0000000    1.0000000 1.0000000
## DMU22  22 0.8057353    0.8057353 0.8057353
## DMU23  23 0.4363224    0.4363224 0.4363224
## DMU24  24 0.8185575    0.8185575 0.8185575
## DMU25  25 0.8370609    0.8370609 0.8370609
## DMU26  26 1.0000000    1.0000000 1.0000000
## DMU27  27 0.6857021    0.6857021 0.6857021
## DMU28  28 0.7994826    0.7994826 0.7994826
## DMU29  29 1.0000000    1.0000000 1.0000000
## DMU30  30 1.0000000    1.0000000 1.0000000
## DMU31  31 0.8188357    0.8188357 0.8188357
## DMU32  32 0.7338936    0.7338936 0.7338936
## DMU33  33 1.0000000    1.0000000 1.0000000
## DMU34  34 0.7501900    0.7501900 0.7501900
## DMU35  35 1.0000000    1.0000000 1.0000000
## DMU36  36 0.7754417    0.7754417 0.7754417
## DMU37  37 1.0000000    1.0000000 1.0000000
## DMU38  38 0.8398729    0.8398729 0.8398729
## DMU39  39 0.7928489    0.7928489 0.7928489
## DMU40  40 0.8307721    0.8307721 0.8307721
## DMU41  41 1.0000000    1.0000000 1.0000000
## DMU42  42 0.8330071    0.8330071 0.8330071
## DMU43  43 0.9686677    0.9683243 0.9683243
## DMU44  44 1.0000000    1.0000000 1.0000000
## DMU45  45 0.8211967    0.8211967 0.8211967
## DMU46  46 0.8538798    0.8538798 0.8538798
## DMU47  47 0.4605260    0.4605260 0.4605260
## DMU48  48 0.6651606    0.6651606 0.6651606
## DMU49  49 1.0000000    1.0000000 1.0000000
## DMU50  50 0.6897815    0.6897815 0.6897815
## DMU51  51 0.6163303    0.6163303 0.6163303
## DMU52  52 0.6557960    0.6557960 0.6557960
## DMU53  53 1.0000000    1.0000000 1.0000000
## DMU54  54 0.8641071    0.8641071 0.8641071
## DMU55  55 1.0000000    1.0000000 1.0000000
## DMU56  56 0.6903551    0.6887150 0.6887150
## DMU57  57 0.6333229    0.6333229 0.6333229
## DMU58  58 1.0000000    1.0000000 1.0000000
## DMU59  59 0.8827931    0.8827931 0.8827931
## DMU60  60 0.9004117    0.9004117 0.9004117
## DMU61  61 0.7986199    0.7986199 0.7986199
## DMU62  62 0.7983280    0.7983280 0.7983280
## DMU63  63 0.9769562    0.9769562 0.9769562
## DMU64  64 0.6980224    0.6944621 0.6944621
## DMU65  65 1.0000000    1.0000000 1.0000000
## DMU66  66 0.5498293    0.5498293 0.5498293
## DMU67  67 1.0000000    1.0000000 1.0000000
## DMU68  68 0.5451632    0.5451632 0.5451632
## DMU69  69 1.0000000    1.0000000 1.0000000
```

Baseball DEA Example (from slides by Robert Vanderbei at Princeton, `http://orfe.princeton.edu/~rvdb/307/lectures/lec8_show.pdf`)

```
data_filepath <- source_baseball_dat_file_from_github

glpkAPIDEA(mod_filepath, data_filepath, RTS = "crs", Orientation = "in", Dual = FALSE)

baseball_inputs <- source_baseball_inputs_from_github

baseball_outputs <- source_baseball_outputs_from_github
```

```
benchmarking_dea <- dea(baseball_inputs, baseball_outputs, RTS = "crs", ORIENTATION="in" , DUAL = TRUE)
tfdea_dea <- DEA(baseball_inputs, baseball_outputs, rts = "crs", orientation = "input")
```

```
## Warning, data has DMU's with outputs that are zero, this may cause numerical problems
```

```
results <- cbind(thetas, benchmarking_dea$eff, tfdea_dea$eff)
colnames(results) = c("DMU", "Our Tool", "Benchmarking", "TFDEA")
results
```

```
##       DMU  Our Tool Benchmarking     TFDEA
## DMU1    1 0.9849710    0.9849710 0.9849710
## DMU2    2 0.9500546    0.9500546 0.9500546
## DMU3    3 1.0000000    1.0000000 1.0000000
## DMU4    4 1.0000000    1.0000000 1.0000000
## DMU5    5 1.0000000    1.0000000 1.0000000
## DMU6    6 0.8397454    0.8397454 0.8397454
## DMU7    7 1.0000000    1.0000000 1.0000000
## DMU8    8 1.0000000    1.0000000 1.0000000
## DMU9    9 0.8704708    0.8704708 0.8704708
## DMU10  10 1.0000000    1.0000000 1.0000000
## DMU11  11 0.8941253    0.8941253 0.8941253
## DMU12  12 0.7466169    0.7466169 0.7466169
## DMU13  13 1.0000000    1.0000000 1.0000000
## DMU14  14 0.8278968    0.8278968 0.8278968
## DMU15  15 0.9179704    0.9179704 0.9179704
## DMU16  16 0.8507021    0.8507021 0.8507021
```

Extensions or future Work

Allow the user to create the data as a datframe in R, then write it out to a file for glpkAPI. This could be easier when this exists in a package (so the user doesn't need to specify where the files will be written out to). In our opinion, the easiest method of bringing in data is by copying data from a text editor into the appropriate place in a .dat file.

Adding a parameter to the gmpl file so that the logical expression of which RTS constraint to choose is done in the mod file and not in the wrapper function.

Currently, the file's RTS is restored to the original after the model file is done running. Because I couldn't get the string-substitution to work on a long string, currently we're replacing the RTS that the model runs with the "### RTS Constraint" used for the next .mod call based on it being in line 25. This is extremely fragile and needs to be fixed.

Creating a copy of the mod file for each DEA run, so that we're not relying on using and re-using one file which may become corrupted downstream.