A Quick Start for The R Interface to LINGO API

March 31, 2018

1 Introduction

The package *rLingo* is an R interface to LINGO optimization system. LINGO is an algebraic modeling language linked to an array of optimization engines, or solvers. It provides the power of linear, integer, quadratic, conic, and general nonlinear optimization to formulate large problems concisely, solve them, and analyze the solution. Optimization helps you find the answer that yields the best result; attains the highest profit, output, or happiness; or the one that achieves the lowest cost, waste, or discomfort. Often these problems involve making the most efficient use of your resources including money, time, machinery, staff, inventory, and more. For more information on LINGO, please refer to www.lindo.com.

2 Installation

We describe the following steps: a) Install R, b) Install LINGO, c) Connect R and LINGO, d) Test the connection on included examples, and e) Outline how to build your own optimization examples.

2.1 Installation Details

2.1.1 Windows

a) Download R, if you have not already, from https://cran.r-project.org. Installation is usually straightforward. The biggest decision is 32 bit vs. 64 bit. If you have a 64 bit computer, choose 64 bit.

b) Download LINGO, if you have not already, from www.lindo.com. Installation is usually straightforward. The biggest decision is 32 bit vs. 64 bit. If you have a 64 bit

computer, choose 64 bit.

c) Connecting R and LINGO. You need to run the command script install-rlingo.bat. Under the default installation of LINGO, this file is in directory: c:\LINGO64_18\Programming Samples\R. There are a variety of ways of running this script. One way is from a command line as follows:

Press the "Windows" key (4 dark squares) on the lower left of the keyboard.
Type the word "command" in the search bar that is displayed.
Click right button on "command prompt" and choose "Run as administrator".
In the command prompt window that is displayed, type in:

cd c:\LINGO64_18\Programming Samples\R
install-rlingo

If all goes well you should see a message ending with:

*** testing if installed package can be loaded*
** DONE (rLingo)*
Finished

d) You can test if the connection works by running several rLingo optimization model scripts that are included in the LINGO distribution. In a default installation these scripts are in c:\LINGO64_18\Programming Samples\R\samples. One way of testing them is from the command prompt. Continuing from above, type:

cd samples (or also *cd c:\LINGO64_18\Programming Samples\R\samples*)

Start R, either by clicking on the R icon on the desktop, or by typing in the command prompt window:

R

Tell R to connect to rLingo by typing:

library(rLingo)

Example 1: There is a sales territories design example (see later discussion) that minimizes the maximum distance from centroid to other regions. This is an example of a mixed integer programming model. You can run it by typing at the R prompt:

source('SaleTerrDsgn.R')

You should see the following output:

Global optimum found!
District Design Analysis
Input data summary:
3= Number districts.
22.33= Average work/district. *25.00 = Max allowed.*
187.00= Average sales value/district. *181.00 = Min allowed.*
Solution summary:

District	Centroid	Workload	Sales_potential	Distance_from_centroid
1	AMARILLO	23.0	183.00	1063.36
2	BRACKETTVIL	21.0	183.00	1556.40
3	KLONDIKE	23.0	195.00	1325.16

Assignment detail:

Centroid	*Customer Assigned*
AMARILLO	*AMARILLO*
AMARILLO	*EL_PASO*
AMARILLO	*LUBBOCK*
AMARILLO	*TEXHOMA*
AMARILLO	*TEXLINE*
BRACKETTVIL	*BROWNSVILLE*
BRACKETTVIL	*BRACKETTVIL*
BRACKETTVIL	*CORPUS_CHR*
BRACKETTVIL	*MARFA*
BRACKETTVIL	*PECOS*
KLONDIKE	*BEAUMONT*
KLONDIKE	*CLARKSVILLE*
KLONDIKE	*GALVESTON*
KLONDIKE	*KLONDIKE*
KLONDIKE	*TEXARKANA*
KLONDIKE	*WACO*

Example 2: There is an acceptance sampling example (see later discussion) that determines the minimum sample size needed to accept or reject a lot of discrete items. This is an example of a nonlinear integer optimization model, which requires the Nonlinear license. You can run it by typing at the R prompt:

source('samsizr.R')

You should see the following output:

Given:

0.03 = AQL = Fraction defective allowed in acceptable/good lot.

0.08 = LTFD = Fraction defective in unacceptable/bad lot.

0.09 = Producer risk = Prob(rejecting a good lot).

0.05 = Consumer risk = Prob(accepting a bad lot).

Local optimum found!

The Optimal (Minimum) sample size is 197.

Accept the lot if 9 or less defectives in sample.

Example 3: There is a forecasting "curve fit" model for predicting the adoption/sales rate over time of a new product, the so-called Bass model (see later discussion). It fits a forecast curve to three or more initial observations, choosing the best values for three parameters so as to minimize the sum of squared errors between the forecast and the actual sales. You can run it by typing:

source('BassModelCT.R')

You should see the following output:

Local optimum found!

Min sum of squared errors is 0.3979429

Estimated market size, M is 215.2067

Innovator coefficient, p is 0.3048617
Copycat, word-of-mouth coefficient, q is 0.2045085.

2.1.2 Linux or other Unix-like systems

NOTE: Make sure to login as the 'root' user to perform these steps.

1. Install Lingo 18.0 for the host platform (e.g. Linux 32 or 64-bit).
2. Install R (and R-devel if R headers are missing on your system).
3. Make sure $LINGO_18_HOME points to the installation path of Lingo. If not,

   $export LINGO_18_HOME="/home/lingo18"

   To automatically set this variable, please add the above command to the ~/.bashrc file. Note that your Lingo folder may lie in a folder other than /home/lingo, in which case, you should modify the above statement accordingly.
4. Navigate to the "lingo18/Programming Samples/R" directory where rLingo_18.0.tar.gz is located.
5. Run the following command to install the rLingo package.

   $./install-rlingo.sh
6. Test whether the rLingo package has been installed successfully with the following R-command:

   library(rLingo)

3 Usage

Preparing a LINGO model for use with R involves two steps: 1) Writing the formulae of the LINGO model, and 2) Writing a small R script that describes how data are supplied to the LINGO model and how the solution results are retrieved from LINGO. With regard to (1) there are hundreds of example LINGO models in the MODELS library at www.lindo.com. There is a good chance that some model in that library is similar to what you are trying to do.

If you are using RLingo with R Shiny, Shiny may modify the system environment variables. This can prevent Lingo from finding its license file. To prevent this, in your R code add the following setenv() call to set the LINGO_18_HOME environment variable to point to the main Lingo folder. Note that this call should be made right before the call to rLScreateEnvLng():

 # Set the Lingo home env variable to allow
 # Lingo to find the license file
Sys.setenv(LINGO_18_HOME = "/home/ubuntu/lingo18")

Here, the main Lingo folder is "/home/ubuntu/lingo18". Your main Lingo folder will most likely be different and you should adjust accordingly.

4 Some Complete Examples

4.1 An application to the Acceptance Sampling Design

If the number of defectives in the lot is 3% or less, the lot is considered "good". If the defects exceed 8%, the lot is considered "bad". Testing every item is expensive, e.g., a tested item is destroyed in the process, so we want to take a small sample and test it. We want a producer risk (probability of rejecting a good lot) below 9% and a consumer risk (probability of accepting a bad lot) below 5%. We want to find the smallest sample size, N and a defectives threshold C, such that if we take a sample of size N, and accept the lot if the number of defectives in the sample is C or less, the error probabilities above will be satisfied. We make use of LINGO's cumulative Poisson distribution function as an approximation of the Binomial distribution. Here is the LINGO model:

Model: samsizr

```
MODEL:
! Acceptance Sampling Design;
! From a large lot, take a sample of size N, accept if C or less are defective;
! Poisson approximation to number defective is used;

DATA:
   AQL = @POINTER( 1); ! Argument 1 is "Good" lot fraction defective;
   LTFD = @POINTER( 2); ! Argument 2 is "Bad" lot fraction defective;
   PRDRISK = @POINTER( 3); ! Tolerance for rejecting good lot;
   CONRISK = @POINTER( 4); ! Tolerance for accepting bad lot;
   MINSMP = @POINTER( 5); ! Lower bound on sample size;
   MAXSMP = @POINTER( 6); ! Upper bound on sample size;
ENDDATA

[OBJ] MIN = N;

! Tolerance for rejecting a good lot;
1 - @PPOISCDF( N * AQL, C) <= PRDRISK;

! Tolerance for accepting a bad lot;
@PPOISCDF( N * LTFD, C) <= CONRISK;

! Give solver some help in getting into range;
@BND( MINSMP, N, MAXSMP);
@BND( 1, C, MAXSMP);

! Make variables general integer;
@GIN( N); @GIN( C);
```

```
DATA:
   @POINTER( 7) = N;
   @POINTER( 8) = C;
   @POINTER( 9) = @STATUS();
ENDDATA

END
```

Then using the R interface to LINGO API, we can solve the above integer, nonlinear optimization model with the following R code.

```
> #load the package
> library(rLingo)
> #define a function to run the samsizr example
> samsizr <- function(AQL,LTFD,PRDRISK,CONRISK,MINSMP,MAXSMP)
+ {
+     pResult <- list(ErrorCode = LSERR_NO_ERROR_LNG)
+
+     #create Lingo enviroment object
+     pLINGO <- rLScreateEnvLng();
+     if(is.null(pLINGO))
+     {
+         cat("\ncannot create LINGO environment!\n")
+         return(pResult)
+     }
+
+     #open LINGO's log file
+     pResult <- rLSopenLogFileLng(pLINGO,"samsizr.log")
+     if(pResult$ErrorCode != LSERR_NO_ERROR_LNG)
+     {
+         return(pResult)
+     }
+
+     #pass memory transfer pointers to LINGO
+     #define pnPointersNow
+     pnPointersNow = integer(1)
+
+     #@POINTER(1)
+     AQL_1 = c(AQL)
+     pResult <- rLSsetDouPointerLng(pLINGO, AQL_1, pnPointersNow)
+     if(pResult$ErrorCode != LSERR_NO_ERROR_LNG)
+     {
```

```
+         return(pResult)
+     }
+
+     #@POINTER(2)
+     LTFD_1 = c(LTFD)
+     pResult <- rLSsetDouPointerLng(pLINGO, LTFD_1, pnPointersNow)
+     if(pResult$ErrorCode != LSERR_NO_ERROR_LNG)
+     {
+         return(pResult)
+     }
+
+     #@POINTER(3)
+     PRDRISK_1 = c(PRDRISK)
+     pResult <- rLSsetDouPointerLng(pLINGO, PRDRISK_1, pnPointersNow)
+     if(pResult$ErrorCode != LSERR_NO_ERROR_LNG)
+     {
+         return(pResult)
+     }
+
+     #@POINTER(4)
+     CONRISK_1 = c(CONRISK)
+     pResult <- rLSsetDouPointerLng(pLINGO, CONRISK_1, pnPointersNow)
+     if(pResult$ErrorCode != LSERR_NO_ERROR_LNG)
+     {
+         return(pResult)
+     }
+
+     #@POINTER(5)
+     MINSMP_1 = c(MINSMP)
+     pResult <- rLSsetDouPointerLng(pLINGO, MINSMP_1, pnPointersNow)
+     if(pResult$ErrorCode != LSERR_NO_ERROR_LNG)
+     {
+         return(pResult)
+     }
+
+     #@POINTER(6)
+     MAXSMP_1 = c(MAXSMP)
+     pResult <- rLSsetDouPointerLng(pLINGO, MAXSMP_1, pnPointersNow)
+     if(pResult$ErrorCode != LSERR_NO_ERROR_LNG)
+     {
+         return(pResult)
+     }
+
+     #@POINTER(7)
```

```
+     N = numeric(1)
+     pResult <- rLSsetDouPointerLng(pLINGO, N, pnPointersNow)
+     if(pResult$ErrorCode != LSERR_NO_ERROR_LNG)
+     {
+         return(pResult)
+     }
+
+     #@POINTER(8)
+     C = numeric(1)
+     pResult <- rLSsetDouPointerLng(pLINGO, C, pnPointersNow)
+     if(pResult$ErrorCode != LSERR_NO_ERROR_LNG)
+     {
+         return(pResult)
+     }
+
+     #@POINTER(9)
+     dStatus = c(-1.0)
+     pResult <- rLSsetDouPointerLng(pLINGO, dStatus, pnPointersNow)
+     if(pResult$ErrorCode != LSERR_NO_ERROR_LNG)
+     {
+         return(pResult)
+     }
+
+     #Here is the script we want LINGO to run
+     cScript = "SET ECHOIN 1 \n TAKE samsizr.lng \n GO \n QUIT \n"
+
+     #Run the script
+     pResult <- rLSexecuteScriptLng(pLINGO, cScript)
+     if(pResult$ErrorCode != LSERR_NO_ERROR_LNG)
+     {
+         return(pResult)
+     }
+
+     #Close the log file
+     pResult <- rLScloseLogFileLng(pLINGO)
+     if(pResult$ErrorCode != LSERR_NO_ERROR_LNG)
+     {
+         return(pResult)
+     }
+
+     #check solution
+     cat("\nGiven:\n")
+     cat(' ', AQL_1, " = AQL = Fraction defective allowed in acceptable/good lot.\n")
+     cat(' ', LTFD_1, " = LTFD = Fraction defective in unacceptable/bad lot.\n")
```

```
+     cat(' ', PRDRISK_1, " = Producer risk = Prob(rejecting a good lot).\n")
+     cat(' ', CONRISK_1, " = Consumer risk = Prob(accepting a bad lot).\n")
+
+     if(dStatus == LS_STATUS_GLOBAL_LNG)
+     {
+         cat("\nGlobal optimum found!")
+     }
+     else if(dStatus == LS_STATUS_LOCAL_LNG)
+     {
+         cat("\nLocal optimum found!")
+     }
+     else
+     {
+         cat("\nSolution is non-optimal\n")
+         return(pResult)
+     }
+
+
+     if(pResult$ErrorCode != LSERR_NO_ERROR_LNG)
+     {
+         return(pResult)
+     }
+
+     #@POINTER(2)
+     LTFD_1
+     cat("\nThe Optimal(Minimum) sample size is",N,".\nAccept the lot if",C,
+     "or less defectives in sample.\n\n")
+
+     #delete Lingo enviroment object
+     pResult <- rLSdeleteEnvLng(pLINGO)
+     if(pResult$ErrorCode != LSERR_NO_ERROR_LNG)
+     {
+         return(pResult)
+     }
+
+     return(pResult)
+ }
> #run the function
> samsizr(0.03,0.08,0.09,0.05,125.,400.)

Given:
  0.03  = AQL = Fraction defective allowed in acceptable/good lot.
  0.08  = LTFD = Fraction defective in unacceptable/bad lot.
  0.09  = Producer risk = Prob(rejecting a good lot).
```

```
  0.05  = Consumer risk = Prob(accepting a bad lot).

Local optimum found!
The Optimal(Minimum) sample size is 197 .
Accept the lot if 9 or less defectives in sample.

$ErrorCode
[1] 0
```

4.2 An application to the Production Forecasting Model

Bass (1969) introduced a model for predicting first purchases of a new product. There are three parameters specifying the model:
M = Estimate of the market size, that is, the total number of customers that will eventually buy the product. For simplicity, we assume that the product is a durable good, so that a customer will purchase it at most once.
p = Probability that a candidate customer in a given period will purchase the product just by chance, without any influence from existing customers. These folks are called the innovators.
q = Rate at which a candidate customer is induced to purchase the product because of interaction with existing customers, the greater the number of existing customers, the greater the inducement. These folks are the followers.
If p = 0, then the Bass model approaches a Logistics S curve for total cumulative sales.
If q = 0, then the Bass model approaches an exponential cdf for total cumulative sales.

Model: BassModelCT

```
MODEL:
SETS:
   PERIOD: SALES, ERROR, FORECAST;
ENDSETS

[OBJECTIVE] MIN = SSE;
SSE = @SUM( PERIOD( T)| T #LE# TRAINON: ERROR( T) * ERROR( T));
@FOR( PERIOD( T):
   @FREE( ERROR( T));
   @FREE( FORECAST(T));
);
PQ = P + Q;
@FOR( PERIOD( t):
   FORECAST(t)=( M*PQ^2/ P)*(@EXP(-PQ*t)/( (Q/ P)*@EXP(-PQ*t)+1)^2);
   ERROR( T) = FORECAST( T) - SALES( T);
);
DATA:
```

```
    PERIOD = @POINTER( 1); ! Argument 1 is number periods of initial sales data;
    SALES = @POINTER( 2); ! Argument 2 is the vector of sales data;
    TRAINON = @POINTER( 3); ! Argument 3 is periods to use to "train" the model;
    ULP = @POINTER( 4);
    ULQ = @POINTER( 5);
    ULM = @POINTER( 6);
    LLP = @POINTER( 7);
    LLQ = @POINTER( 8);
    LLM = @POINTER( 9);
    @POINTER( 10) = P;
    @POINTER( 11) = Q;
    @POINTER( 12) = M;
    @POINTER( 13) = ERROR;
    @POINTER( 14) = FORECAST;
    @POINTER( 15) = @STATUS();
    @POINTER( 16) = SSE;
 ENDDATA

 END
```

Then using the R interface to LINGO API, we can solve the above nonlinear optimization model with the following R code.

```
> #load the package
> library(rLingo)
> #define a function to run the BassModelCT example
> BassModelCT <- function(SALES, TRAINON, ULP, ULQ, ULM, LLP, LLQ, LLM)
+ {
+     pResult <- list(ErrorCode = LSERR_NO_ERROR_LNG)
+
+     #Create Lingo enviroment object
+     pLINGO <- rLScreateEnvLng();
+     if(is.null(pLINGO))
+     {
+         cat("\ncannot create LINGO environment!\n")
+         return(pResult)
+     }
+
+     #Open LINGO's log file
+     pResult <- rLSopenLogFileLng(pLINGO,"BassModelCT.log")
+     if(pResult$ErrorCode != LSERR_NO_ERROR_LNG)
+     {
```

```
+         return(pResult)
+     }
+
+     #Pass memory transfer pointers to LINGO
+     #Define pnPointersNow
+     pnPointersNow = integer(1)
+
+     #@POINTER(1)
+     #Note that PERIOD is a set, so its pointer must be passed to LINGO first.
+     PERIOD = c("")
+     for(i in 1:length(SALES))
+     {
+         PERIOD = paste(PERIOD, i, "\n");
+     }
+     pResult <- rLSsetCharPointerLng(pLINGO, PERIOD, pnPointersNow)
+     if(pResult$ErrorCode != LSERR_NO_ERROR_LNG)
+     {
+         return(pResult)
+     }
+
+     #@POINTER(2)
+     pResult <- rLSsetDouPointerLng(pLINGO, SALES, pnPointersNow)
+     if(pResult$ErrorCode != LSERR_NO_ERROR_LNG)
+     {
+         return(pResult)
+     }
+
+     #@POINTER(3)
+     pResult <- rLSsetDouPointerLng(pLINGO, TRAINON, pnPointersNow)
+     if(pResult$ErrorCode != LSERR_NO_ERROR_LNG)
+     {
+         return(pResult)
+     }
+
+     #@POINTER(4)
+     pResult <- rLSsetDouPointerLng(pLINGO, ULP, pnPointersNow)
+     if(pResult$ErrorCode != LSERR_NO_ERROR_LNG)
+     {
+         return(pResult)
+     }
+
+     #@POINTER(5)
+     pResult <- rLSsetDouPointerLng(pLINGO, ULQ, pnPointersNow)
+     if(pResult$ErrorCode != LSERR_NO_ERROR_LNG)
```

```
+     {
+         return(pResult)
+     }
+
+     #@POINTER(6)
+     pResult <- rLSsetDouPointerLng(pLINGO, ULM, pnPointersNow)
+     if(pResult$ErrorCode != LSERR_NO_ERROR_LNG)
+     {
+         return(pResult)
+     }
+
+     #@POINTER(7)
+     pResult <- rLSsetDouPointerLng(pLINGO, LLP, pnPointersNow)
+     if(pResult$ErrorCode != LSERR_NO_ERROR_LNG)
+     {
+         return(pResult)
+     }
+
+     #@POINTER(8)
+     pResult <- rLSsetDouPointerLng(pLINGO, LLQ, pnPointersNow)
+     if(pResult$ErrorCode != LSERR_NO_ERROR_LNG)
+     {
+         return(pResult)
+     }
+
+     #@POINTER(9)
+     pResult <- rLSsetDouPointerLng(pLINGO, LLM, pnPointersNow)
+     if(pResult$ErrorCode != LSERR_NO_ERROR_LNG)
+     {
+         return(pResult)
+     }
+
+     #@POINTER(10)
+     P = c(1.0)
+     pResult <- rLSsetDouPointerLng(pLINGO, P, pnPointersNow)
+     if(pResult$ErrorCode != LSERR_NO_ERROR_LNG)
+     {
+         return(pResult)
+     }
+
+     #@POINTER(11)
+     Q = c(1.0)
+     pResult <- rLSsetDouPointerLng(pLINGO, Q, pnPointersNow)
+     if(pResult$ErrorCode != LSERR_NO_ERROR_LNG)
```

```
+     {
+         return(pResult)
+     }
+
+     #@POINTER(12)
+     M = c(1.0)
+     pResult <- rLSsetDouPointerLng(pLINGO, M, pnPointersNow)
+     if(pResult$ErrorCode != LSERR_NO_ERROR_LNG)
+     {
+         return(pResult)
+     }
+
+     #@POINTER(13)
+     ERROR = numeric(length(SALES))
+     pResult <- rLSsetDouPointerLng(pLINGO, ERROR, pnPointersNow)
+     if(pResult$ErrorCode != LSERR_NO_ERROR_LNG)
+     {
+         return(pResult)
+     }
+
+     #@POINTER(14)
+     FORECAST = numeric(length(SALES))
+     pResult <- rLSsetDouPointerLng(pLINGO, FORECAST, pnPointersNow)
+     if(pResult$ErrorCode != LSERR_NO_ERROR_LNG)
+     {
+         return(pResult)
+     }
+
+     #@POINTER(15)
+     dStatus = c(-1.0)
+     pResult <- rLSsetDouPointerLng(pLINGO, dStatus, pnPointersNow)
+     if(pResult$ErrorCode != LSERR_NO_ERROR_LNG)
+     {
+         return(pResult)
+     }
+
+     #@POINTER(16)
+     dObjective = c(-1.0)
+     pResult <- rLSsetDouPointerLng(pLINGO, dObjective, pnPointersNow)
+     if(pResult$ErrorCode != LSERR_NO_ERROR_LNG)
+     {
+         return(pResult)
+     }
+
```

```
+     #Here is the script we want LINGO to run
+     cScript = "SET ECHOIN 1 \n TAKE BassModelCT.lng \n GO \n QUIT \n"
+
+     #Run the script
+     pResult <- rLSexecuteScriptLng(pLINGO, cScript)
+     if(pResult$ErrorCode != LSERR_NO_ERROR_LNG)
+     {
+         return(pResult)
+     }
+
+     #Close the log file
+     pResult <- rLScloseLogFileLng(pLINGO)
+     if(pResult$ErrorCode != LSERR_NO_ERROR_LNG)
+     {
+         return(pResult)
+     }
+
+     #Check solver status
+     if(dStatus == LS_STATUS_GLOBAL_LNG)
+     {
+         cat("\nGlobal optimum found!")
+     }
+     else if(dStatus == LS_STATUS_LOCAL_LNG)
+     {
+         cat("\nLocal optimum found!")
+     }
+     else
+     {
+         cat("\nSolution is non-optimal, status = ", dStatus, "\n")
+         return(pResult)
+     }
+
+     #Check solution
+     cat("\nMin sum of squared errors is", dObjective,"\n")
+     #
+     cat("Estimated market size, M is ", M,"\n")
+     #
+     cat("Innovator coefficient, p is ", P,"\n")
+     #
+     cat("Copycat, word-of-mouth coefficient, q is ", Q,"\n")
+
+     #Delete Lingo enviroment object
+     pResult <- rLSdeleteEnvLng(pLINGO)
+     if(pResult$ErrorCode != LSERR_NO_ERROR_LNG)
```

```
+     {
+         return(pResult)
+     }
+
+     return(pResult)
+ }
> #All inputs are initialized here
> #Weekly sales for the movie "Gravity";
> SALES = c(55.8, 43.2, 30.0, 20.1, 12.8, 8.5, 6.1, 3.2, 2.6);
> TRAINON = c(6)
> ULP = c(1.0)
> ULQ = c(1.0)
> ULM = c(9999)
> LLP = c(0.0)
> LLQ = c(0.0)
> LLM = c(0.0)
> #Run the function
> BassModelCT(SALES, TRAINON, ULP, ULQ, ULM, LLP, LLQ, LLM)

Local optimum found!
Min sum of squared errors is 0.3979429
Estimated market size, M is  215.2067
Innovator coefficient, p is  0.3048617
Copycat, word-of-mouth coefficient, q is  0.2045085
$ErrorCode
[1] 0
```

4.3 An application to the Markowitz Portfolio Model

The following is a standard Markowitz Portfolio model with a constraint on number of assets in the portfolio. The LINGO model is:

Model: PORTCardCor

```
MODEL:
SETS:
   STOCK: RET, SD, AMT, Z;
   COVMAT( STOCK, STOCK) | &1 #LE# &2: CORR;
ENDSETS

! Minimize portfolio variance, with a 2 applied when I < J;
[RISK] MIN = PVAR;

PVAR = @SUM( STOCK( I): AMT( I) * AMT( I) * SD(I)^2)
     + 2 * @SUM( COVMAT( I, J) | I #LT# J:
```

```
                AMT( I) * AMT( J) * SD(I)*SD(j)*CORR( I, J));

        ! Use exactly 100% of the starting budget;
        [BUDGET] @SUM( STOCK: AMT) = 1;

        ! Required return at end of period;
        [RETURN] @SUM( STOCK: AMT * RET) > TARG;

        ! Cardinality constraints;
        @FOR( STOCK( I):
            @BIN( Z( I)); ! Z(i) is binary, i.e., 0 or 1;
                AMT( I) <= Z( I); ! If AMT(I) > 0, then Z(i) = 1;
        );
        ! Limit on number of stocks at positive level;
        @SUM( STOCK( I): Z( I)) <= CARD;

        DATA:
        STOCK = @POINTER( 1);
        CARD = @POINTER( 2);
        TARG = @POINTER( 3);
        RET = @POINTER( 4);
        SD = @POINTER( 5);
        CORR = @POINTER( 6);
        ENDDATA

        END
```

Then using the R interface to LINGO API, we can solve the above mixed integer conic optimization model with the following R code.

```
> #load the package
> library(rLingo)
> #define a function to run the PORTCardCor example
> PORTCardCor <- function(STOCK, CARD, TARG, RET, SD, CORR)
+ {
+     pResult <- list(ErrorCode = LSERR_NO_ERROR_LNG)
+
+     #Create Lingo enviroment object
+     pLINGO <- rLScreateEnvLng();
+     if(is.null(pLINGO))
+     {
+         cat("\ncannot create LINGO environment!\n")
```

```
+         return(pResult)
+     }
+
+     #Open LINGO's log file
+     pResult <- rLSopenLogFileLng(pLINGO,"PORTCardCor.log")
+     if(pResult$ErrorCode != LSERR_NO_ERROR_LNG)
+     {
+         return(pResult)
+     }
+
+     #Pass memory transfer pointers to LINGO
+     #Define pnPointersNow
+     pnPointersNow = integer(1)
+
+     #@POINTER(1)
+     #Note that STOCK is a set, so its pointer must be passed to LINGO first.
+     pResult <- rLSsetCharPointerLng(pLINGO, STOCK, pnPointersNow)
+     if(pResult$ErrorCode != LSERR_NO_ERROR_LNG)
+     {
+         return(pResult)
+     }
+
+     #@POINTER(2)
+     pResult <- rLSsetDouPointerLng(pLINGO, CARD, pnPointersNow)
+     if(pResult$ErrorCode != LSERR_NO_ERROR_LNG)
+     {
+         return(pResult)
+     }
+
+     #@POINTER(3)
+     pResult <- rLSsetDouPointerLng(pLINGO, TARG, pnPointersNow)
+     if(pResult$ErrorCode != LSERR_NO_ERROR_LNG)
+     {
+         return(pResult)
+     }
+
+     #@POINTER(4)
+     pResult <- rLSsetDouPointerLng(pLINGO, RET, pnPointersNow)
+     if(pResult$ErrorCode != LSERR_NO_ERROR_LNG)
+     {
+         return(pResult)
+     }
+
+     #@POINTER(5)
```

```
+     pResult <- rLSsetDouPointerLng(pLINGO, SD, pnPointersNow)
+     if(pResult$ErrorCode != LSERR_NO_ERROR_LNG)
+     {
+         return(pResult)
+     }
+
+     #@POINTER(6)
+     pResult <- rLSsetDouPointerLng(pLINGO, CORR, pnPointersNow)
+     if(pResult$ErrorCode != LSERR_NO_ERROR_LNG)
+     {
+         return(pResult)
+     }
+
+     #@POINTER(7)
+     AMT = numeric(length(RET))
+     pResult <- rLSsetDouPointerLng(pLINGO, AMT, pnPointersNow)
+     if(pResult$ErrorCode != LSERR_NO_ERROR_LNG)
+     {
+         return(pResult)
+     }
+
+     #@POINTER(8)
+     Z = numeric(length(RET))
+     pResult <- rLSsetDouPointerLng(pLINGO, Z, pnPointersNow)
+     if(pResult$ErrorCode != LSERR_NO_ERROR_LNG)
+     {
+         return(pResult)
+     }
+
+     #@POINTER(9)
+     PVAR = numeric(1)
+     pResult <- rLSsetDouPointerLng(pLINGO, PVAR, pnPointersNow)
+     if(pResult$ErrorCode != LSERR_NO_ERROR_LNG)
+     {
+         return(pResult)
+     }
+
+     #@POINTER(10)
+     dStatus = c(-1.0)
+     pResult <- rLSsetDouPointerLng(pLINGO, dStatus, pnPointersNow)
+     if(pResult$ErrorCode != LSERR_NO_ERROR_LNG)
+     {
+         return(pResult)
+     }
```

```
+
+     #Here is the script we want LINGO to run
+     cScript = "SET ECHOIN 1 \n TAKE PORTCardCor.lng \n GO \n QUIT \n"
+
+     #Run the script
+     pResult <- rLSexecuteScriptLng(pLINGO, cScript)
+     if(pResult$ErrorCode != LSERR_NO_ERROR_LNG)
+     {
+         return(pResult)
+     }
+
+     #Close the log file
+     pResult <- rLScloseLogFileLng(pLINGO)
+     if(pResult$ErrorCode != LSERR_NO_ERROR_LNG)
+     {
+         return(pResult)
+     }
+
+     #Check solver status
+     if(dStatus == LS_STATUS_GLOBAL_LNG)
+     {
+         cat("\nGlobal optimum found!")
+     }
+     else if(dStatus == LS_STATUS_LOCAL_LNG)
+     {
+         cat("\nLocal optimum found!")
+     }
+     else
+     {
+         cat("\nSolution is non-optimal, status = ", dStatus, "\n")
+         return(pResult)
+     }
+
+     #Check solution
+     cat("\nMin portfolio variance is", PVAR,"\n")
+     for(i in 1:length(Z))
+     {
+          if(Z[i] > 0.9)
+          {
+               cat(AMT[i]*100, "percent of budget for ")
+               cat(strsplit(STOCK,split=" ")[[1]][2*(i-1)+1], " \n")
+          }
+     }
+
```

```
+     #Delete Lingo enviroment object
+     pResult <- rLSdeleteEnvLng(pLINGO)
+     if(pResult$ErrorCode != LSERR_NO_ERROR_LNG)
+     {
+         return(pResult)
+     }
+
+     return(pResult)
+ }
> #All inputs are initialized here
> #The stocks we are considering
> STOCK = c("Alcoa \n Ford \n Deere \n Dupont \n PandG
+  Chevron \n 3M \n USsteel \n Mcrsft \n")
> #Limit on stocks in protfolio
> CARD = c(4)
> #Target return, 1+r, per year
> TARG = c(1.07)
> #Expected yearly growth factor (1+r) for each stock
> RET = c(0.9853,1.2487,1.1279,1.0995,1.0531,1.0868,1.0992,1.0722,1.1089)
> #Estimated yearly standard deviation in growth factor
> SD = c(0.4134,0.5698,0.2883,0.2935,0.1550,0.2266,0.1946,0.6275,0.2194)
> #Correlation matrix of the growth factors/returns, upper diagonal only
> CORR = c(1.0000,0.5388,0.6670,0.6312,0.2909,0.5687,0.6028,0.6506,0.4155,
+                 1.0000,0.6204,0.5321,0.1150,0.3330,0.3775,0.4368,0.2620,
+                        1.0000,0.6837,0.3213,0.4626,0.5921,0.6108,0.3577,
+                               1.0000,0.3715,0.4817,0.6796,0.5356,0.4646,
+                                      1.0000,0.3688,0.5061,0.2056,0.4199,
+                                             1.0000,0.5976,0.4257,0.4010,
+                                                    1.0000,0.4421,0.4723,
+                                                           1.0000,0.3526,
+                                                                  1.0000)
> #Run the function
> PORTCardCor(STOCK, CARD, TARG, RET, SD, CORR)

Global optimum found!
Min portfolio variance is 0.02024319
61.47317 percent of budget for PandG
11.92972 percent of budget for Chevron
13.28488 percent of budget for 3M
13.31223 percent of budget for Mcrsft
$ErrorCode
[1] 0
```

4.4 An application to Sales Territories Design

We want to group customers into territories or regions so that: 1) The sales potential for each region is similar that of others, 2) The workload of each region is similar to that of others, 3) Each region is fairly compact so the sales rep assigned to a region does not incur a lot of travel time. The simplest estimate of sales potential of a customer is past sales. The estimated workload of a customer is the estimated hours per week that the sales rep should spend with the customer. One measure of compactness of a region is the sum of the distances of all customers from the most central customer of the region. A simple measure of distance is the straightline distance based on the latitude and longitude of each customer. Qualitatively, the model is to minimize the maximum distance from centroid over regions, subject to: workload in each region <= target workload and sales potential in each region >= target sales potential. The LINGO model is:

Model: SaleTerrDsgn

```
MODEL:
SETS:
   CUSTOMER: WORK, SALESV,
         WORKA, DISTA, SALESA,
         LATI, LNGT;
   CXC( CUSTOMER, CUSTOMER): DIST, Z;
ENDSETS

DATA:
   CUSTOMER = @POINTER( 1);
   NDS = @POINTER( 2);! Number of regions to construct;
   SALEST = @POINTER( 3);! Sales minimum target for each region;
   WORKT = @POINTER( 4);! Work maximum target for each region;
   SUMDWGT = @POINTER( 5);! "Epsilon" weight assigned to minimizing
                          total distance from centroids in objective, in
                          addition to main objective of minimizing
                          maximum distance in any region;
   WORK = @POINTER( 6);! Workload;
   SALESV = @POINTER( 7);! Sales value potential;
   LATI = @POINTER( 8);! Latitude;
   LNGT = @POINTER( 9);! Longitude;

   ! Variables:
   Z(i,j) = 1 is customer i is assigned to centroid j, where
         the centroid of a region is the most central
         customer of the region, thus
   Z(j,j) = 1 if customer j is the centroid of its region;

   ! Each customer i must be assigned to a region
```

```
   centered at some customer j;

   @POINTER( 10) = WORKA;
   @POINTER( 11) = DISTA;
   @POINTER( 12) = SALESA;
   @POINTER( 13) = DIST;
   @POINTER( 14) = Z;
   @POINTER( 15) = @STATUS();
ENDDATA

SUBMODEL SALESASGN:
   @FOR( CUSTOMER(i):
      @SUM( CXC(i,j): Z(i,j)) = 1;
   );

   @FOR( CXC(i,j):
      @BIN( Z(i,j)); ! The Z(i,j) are binary, 0 or 1;
      ! If i assigned to j, then j is a centroid;
      Z(i,j) <= Z(j,j);
   );

   @FOR( CUSTOMER(j):
      ! Calculate Work assigned to region centered at j;
      WORKA(j) = @SUM( CXC(i,j): WORK(i)*Z(i,j));
      ! Calculate Sales Value assigned to j;
      SALESA(j) = @SUM( CXC(i,j): SALESV(i)*Z(i,j));
      ! Calculate distance assigned to j;
      DISTA(j) = @SUM( CXC(i,j): DIST(i,j)*Z(i,j));
   );

   ! Minimize weighted combination of maximum distance in any
   region + total distance over all regions;
   OBJV = DISTMX + SUMDWGT*@SUM( CUSTOMER(j): DISTA(j));
   MIN = OBJV;

   ! Limit on number of districts;
   @SUM( CUSTOMER(i): Z(i,i)) <= NDS;

   @FOR( CUSTOMER(j):
      ! Sales minimum target for each region;
      SALESA(j) >= SALEST*Z(j,j);
      ! Work maximum target for each region;
      WORKA(j) <= WORKT*Z(j,j);
      ! Maximum total distance incurred in any region;
```

```
        DISTMX >= DISTA(j);
    );
ENDSUBMODEL

CALC:
    ! Prepare distance matrix;
    ! This portion calculates a distance matrix DIST(i,j);
    D2R=@PI()/180; ! Degrees to radians conversion factor;

    ! Compute Great Circle Distances. Radius of earth = 6371 km.
    Notice this simplifies if LATI(i) = LATI(j) or LNGT(i) = LNGT(j);
    @FOR( CXC(i,j):
        @IFC( i #EQ# j: DIST(i,j) = 0;! Get rid of trivial roundoff;
        @ELSE
        DIST( i,j) = 6371*@acos(@sin(D2R*LATI(i))*@sin(D2R*LATI(j))+
                    @cos(D2R*LATI(i))*@cos(D2R*LATI(j))
                    *@cos(@ABS(D2R*(LNGT(i)-LNGT(j)))));
        );
    );

    @SOLVE( SALESASGN);
ENDCALC

END
```

Then using the R interface to LINGO API, we can solve the above mixed integer model with the following R code.

```
> #load the package
> library(rLingo)
> SaleTerrDsgn <- function(NDS, SALEST, WORKT, SUMDWGT,
+ CUSTOMER, WORK, SALESV, LATI, LNGT)
+ {
+     pResult <- list(ErrorCode = LSERR_NO_ERROR_LNG)
+
+     #Create Lingo enviroment object
+     pLINGO <- rLScreateEnvLng();
+     if(is.null(pLINGO))
+     {
+         cat("\ncannot create LINGO environment!\n")
+         return(pResult)
+     }
+
```

```
+     #Open LINGO's log file
+     pResult <- rLSopenLogFileLng(pLINGO,"SaleTerrDsgn.log")
+     if(pResult$ErrorCode != LSERR_NO_ERROR_LNG)
+     {
+         return(pResult)
+     }
+
+     #Pass memory transfer pointers to LINGO
+     #Define pnPointersNow
+     pnPointersNow = integer(1)
+
+     #@POINTER(1)
+     #Note that CUSTOMER is a set, so its pointer must be passed to LINGO first.
+     pResult <- rLSsetCharPointerLng(pLINGO, CUSTOMER, pnPointersNow)
+     if(pResult$ErrorCode != LSERR_NO_ERROR_LNG)
+     {
+         return(pResult)
+     }
+
+     #@POINTER(2)
+     pResult <- rLSsetDouPointerLng(pLINGO, NDS, pnPointersNow)
+     if(pResult$ErrorCode != LSERR_NO_ERROR_LNG)
+     {
+         return(pResult)
+     }
+
+     #@POINTER(3)
+     pResult <- rLSsetDouPointerLng(pLINGO, SALEST, pnPointersNow)
+     if(pResult$ErrorCode != LSERR_NO_ERROR_LNG)
+     {
+         return(pResult)
+     }
+
+     #@POINTER(4)
+     pResult <- rLSsetDouPointerLng(pLINGO, WORKT, pnPointersNow)
+     if(pResult$ErrorCode != LSERR_NO_ERROR_LNG)
+     {
+         return(pResult)
+     }
+
+     #@POINTER(5)
+     pResult <- rLSsetDouPointerLng(pLINGO, SUMDWGT, pnPointersNow)
+     if(pResult$ErrorCode != LSERR_NO_ERROR_LNG)
+     {
```

```
+         return(pResult)
+     }
+
+     #@POINTER(6)
+     pResult <- rLSsetDouPointerLng(pLINGO, WORK, pnPointersNow)
+     if(pResult$ErrorCode != LSERR_NO_ERROR_LNG)
+     {
+         return(pResult)
+     }
+
+     #@POINTER(7)
+     pResult <- rLSsetDouPointerLng(pLINGO, SALESV, pnPointersNow)
+     if(pResult$ErrorCode != LSERR_NO_ERROR_LNG)
+     {
+         return(pResult)
+     }
+
+     #@POINTER(8)
+     pResult <- rLSsetDouPointerLng(pLINGO, LATI, pnPointersNow)
+     if(pResult$ErrorCode != LSERR_NO_ERROR_LNG)
+     {
+         return(pResult)
+     }
+
+     #@POINTER(9)
+     pResult <- rLSsetDouPointerLng(pLINGO, LNGT, pnPointersNow)
+     if(pResult$ErrorCode != LSERR_NO_ERROR_LNG)
+     {
+         return(pResult)
+     }
+
+     #@POINTER(10)
+     WORKA = numeric(length(WORK))
+     pResult <- rLSsetDouPointerLng(pLINGO, WORKA, pnPointersNow)
+     if(pResult$ErrorCode != LSERR_NO_ERROR_LNG)
+     {
+         return(pResult)
+     }
+
+     #@POINTER(11)
+     DISTA = numeric(length(WORK))
+     pResult <- rLSsetDouPointerLng(pLINGO, DISTA, pnPointersNow)
+     if(pResult$ErrorCode != LSERR_NO_ERROR_LNG)
+     {
```

```
+         return(pResult)
+     }
+
+     #@POINTER(12)
+     SALESA = numeric(length(WORK))
+     pResult <- rLSsetDouPointerLng(pLINGO, SALESA, pnPointersNow)
+     if(pResult$ErrorCode != LSERR_NO_ERROR_LNG)
+     {
+         return(pResult)
+     }
+
+     #@POINTER(13)
+     DIST = matrix(numeric(length(WORK)*length(WORK)),
+     nrow=length(WORK),ncol=length(WORK))
+     pResult <- rLSsetDouPointerLng(pLINGO, DIST, pnPointersNow)
+     if(pResult$ErrorCode != LSERR_NO_ERROR_LNG)
+     {
+         return(pResult)
+     }
+
+     #@POINTER(14)
+     Z = matrix(numeric(length(WORK)*length(WORK)),
+     nrow=length(WORK),ncol=length(WORK))
+     pResult <- rLSsetDouPointerLng(pLINGO, Z, pnPointersNow)
+     if(pResult$ErrorCode != LSERR_NO_ERROR_LNG)
+     {
+         return(pResult)
+     }
+
+     #@POINTER(15)
+     dStatus = c(-1.0)
+     pResult <- rLSsetDouPointerLng(pLINGO, dStatus, pnPointersNow)
+     if(pResult$ErrorCode != LSERR_NO_ERROR_LNG)
+     {
+         return(pResult)
+     }
+
+     #Here is the script we want LINGO to run
+     cScript = "SET ECHOIN 1 \n TAKE SaleTerrDsgn.lng \n GO \n QUIT \n"
+
+     #Run the script
+     pResult <- rLSexecuteScriptLng(pLINGO, cScript)
+     if(pResult$ErrorCode != LSERR_NO_ERROR_LNG)
+     {
```

```
+         return(pResult)
+     }
+
+     #Close the log file
+     pResult <- rLScloseLogFileLng(pLINGO)
+     if(pResult$ErrorCode != LSERR_NO_ERROR_LNG)
+     {
+         return(pResult)
+     }
+
+     #Check solver status
+     if(dStatus == LS_STATUS_GLOBAL_LNG)
+     {
+         cat("\nGlobal optimum found! \n")
+     }
+     else if(dStatus == LS_STATUS_LOCAL_LNG)
+     {
+         cat("\nLocal optimum found! \n")
+     }
+     else
+     {
+         cat("\nSolution is non-optimal, status = ", dStatus, "\n")
+         return(pResult)
+     }
+
+     cat("\nDistrict Design Analysis \n")
+     cat(" Input data summary: \n")
+
+     cat(sprintf("%10.0f= Number districts.\n",NDS ))
+
+     SUMWORK = sum(WORK)
+     cat(sprintf("%10.2f= Average work/district.       ", SUMWORK/NDS))
+     cat(sprintf("%10.2f = Max allowed.\n", WORKT))
+
+     SUMSALESV = sum(SALESV)
+     cat(sprintf("%10.2f= Average sales value/district.", SUMSALESV/NDS))
+     cat(sprintf("%10.2f = Min allowed.\n", SALEST))
+
+     cat(" Solution summary:\n")
+
+     S1 = " District    Centroid    Workload   "
+     S2 = "Sales_potential  Distance_from_centroid\n"
+     S = paste(S1, S2, sep="")
+     cat(S)
```

```
+     id = 0
+     for(j in 1:length(WORK))
+     {
+         if(Z[j,j] > 0.5)
+         {
+             id = id + 1
+             cat(sprintf("%6.0f   %12s%12.1f%14.2f%17.2f\n",id,
+             strsplit(CUSTOMER,split=" ")[[1]][2*(j-1)+1],
+             WORKA[j],SALESA[j],DISTA[j]))
+         }
+     }
+
+     cat(" Assignment detail:\n")
+     cat("       Centroid      Customer Assigned\n")
+     for(j in 1:length(WORK))
+     {
+         if(Z[j,j] > 0.5)
+         {
+             for(i in 1:length(WORK))
+             {
+                 if(Z[j,i] > 0)
+                 {
+                     cat(sprintf("%15s%15s\n",
+                     strsplit(CUSTOMER,split=" ")[[1]][2*(j-1)+1],
+                     strsplit(CUSTOMER,split=" ")[[1]][2*(i-1)+1]))
+                 }
+             }
+         }
+     }
+
+     #Delete Lingo enviroment object
+     pResult <- rLSdeleteEnvLng(pLINGO)
+     if(pResult$ErrorCode != LSERR_NO_ERROR_LNG)
+     {
+         return(pResult)
+     }
+
+     return(pResult)
+ }
> #All inputs are initialized here
>
> #Number of regions to construct
> NDS = c(3)
> #Sales minimum target for each region
```

```
> SALEST = c(181)
> #Work maximum target for each region
> WORKT = c(25)
> #"Epsilon" weight assigned to minimizing total
> #distance from centroids in objective, in addition
> #to main objective of minimizing maximum distance
> #in any region.
> SUMDWGT = c(0.75)
> #Customers
> C1 = c("AMARILLO \n BEAUMONT \n BROWNSVILLE \n BRACKETTVIL \n CLARKSVILLE \n ")
> C2 = c("CORPUS_CHR \n EL_PASO \n GALVESTON \n KLONDIKE \n LUBBOCK \n MARFA \n ")
> C3 = c("PECOS \n TEXARKANA \n TEXHOMA \n TEXLINE \n WACO \n")
> CUSTOMER = paste(C1, C2, C3, sep="")
> #Workload
> WORK = c(7,3,6,1,3,6,9,5,4,5,3,5,5,1,1,3)
> #Sales value potential
> SALESV = c(46,31,42,13,17,39,64,39,54,44,46,43,33,17,12,21)
> #Latitude
> LATI = c(35.2,30.08,25.92,29.31,33.61,27.8,31.84,
+ 29.3,33.32,33.57,30.31,31.4,33.44,36.5,36.37,31.47)
> #Longitude
> LNGT = c(-101.81,-94.14,-97.48,-100.41,-95.05,-97.39,
+ -106.43,-94.79,-95.75,-101.87,-104.02,-103.5,-94.07,-101.78,-103.01,-97.24)
> #Run the function
> SaleTerrDsgn(NDS, SALEST, WORKT, SUMDWGT, CUSTOMER, WORK, SALESV, LATI, LNGT)

Global optimum found!

District Design Analysis
 Input data summary:
         3= Number districts.
     22.33= Average work/district.            25.00 = Max allowed.
    187.00= Average sales value/district.    181.00 = Min allowed.
 Solution summary:
 District    Centroid    Workload   Sales_potential  Distance_from_centroid
     1       AMARILLO        23.0        183.00          1063.36
     2    BRACKETTVIL        21.0        183.00          1556.40
     3       KLONDIKE        23.0        195.00          1325.16
 Assignment detail:
       Centroid      Customer Assigned
       AMARILLO       AMARILLO
       AMARILLO        EL_PASO
       AMARILLO        LUBBOCK
       AMARILLO        TEXHOMA
```

```
       AMARILLO        TEXLINE
    BRACKETTVIL    BROWNSVILLE
    BRACKETTVIL    BRACKETTVIL
    BRACKETTVIL     CORPUS_CHR
    BRACKETTVIL          MARFA
    BRACKETTVIL          PECOS
       KLONDIKE       BEAUMONT
       KLONDIKE    CLARKSVILLE
       KLONDIKE      GALVESTON
       KLONDIKE       KLONDIKE
       KLONDIKE      TEXARKANA
       KLONDIKE           WACO
$ErrorCode
[1] 0
```

4.5 An application to stochastic programming on college financing model

We want to decide how to allocate our initial wealth of 55,000 in each of three periods between bonds and stocks so that at the end of three periods we are highly likely to have at least 80,000. Each period, there are two possible outcomes for the return on bonds and stocks. The LINGO model is:

Model: SP_College

```
MODEL:
SETS:
   TIME;
   ASSETS: RETURN_1, RETURN_2, RETURN_3;
   TXA( TIME, ASSETS): RETURN, INVEST;
   Scenarios;
   RV( Scenarios, ASSETS, TIME ): SCENARIORETURNS;
   GV( Scenarios ): SOVER, SUNDER, PRBSCENE;
ENDSETS

DATA:
   TIME = @POINTER( 1 ); ! Time stages;
   ASSETS = @POINTER ( 2 ); ! Names of investments available;
   INITIAL = @POINTER( 3 ); ! Initial capital;
   GOAL = @POINTER( 4 ); ! Goal after three stages;
   PENALTY = @POINTER( 5 ); ! Penalty/unit short of goal;
   Scenarios = 1..8;
ENDDATA

NP = @SIZE( TIME); ! Number of periods;
```

```
! Minize expected shortage penalty, minus overage;
MIN = PENALTY * UNDER - OVER;

! Initial conditions on what we can invest;
! in each new period, the holdings of each commodity
= random return * previous holdings;
   @SUM( ASSETS( A): RETURN( T, A) * INVEST( T - 1, A)) =
   @SUM( ASSETS( A): INVEST( T, A))
);

! Summarize at the end,
Our final wealth;
FINAL = @SUM( ASSETS( A): INVEST( NP, A));
! Were we over or under our goal....;
OVER - UNDER = FINAL - GOAL;

! SP Related Declarations;

! Step 2: Staging information;
! Specify the stage of each investment decision,
with first stage always being 0;
   @SPSTGVAR( T-1, INVEST( T, A));
);

! Construct the outcome table;
SETS:
   OUTCOMES;
   SXA( OUTCOMES, ASSETS): O_RETURN;
ENDSETS
DATA:
   OUTCOMES = @POINTER( 6 );
   O_RETURN = @POINTER( 7 );
ENDDATA
! The Returns are random variables. Specify
the stage of each;
   @SPSTGRNDV( 1, RETURN_1( A));
   RETURN( @INDEX( T1), A) = RETURN_1( A);
   @SPSTGRNDV( 2, RETURN_2( A));
   RETURN( @INDEX( T2), A) = RETURN_2( A);
   @SPSTGRNDV( 3, RETURN_3( A));
   RETURN( @INDEX( T3), A) = RETURN_3( A);
);
```

```
CALC:
   @SET( 'TERSEO', 1);
   @SOLVE();
   I = 1;
   @WHILE( I #LE# @SPNUMSCENE():
      @SPLOADSCENE( I);
      PRBSCENE( I ) = @SPPRBSCENE( I);
      @FOR( TIME( T) | T #GT# @INDEX( T0): @FOR( ASSETS( A):
            SCENARIORETURNS( I, A, T - 1 ) = RETURN( T, A ) ) );
      SOVER( I ) = OVER;
      SUNDER( I ) = UNDER;
      I = I + 1;
   );

ENDCALC
DATA:
   @POINTER( 8 ) = SCENARIORETURNS;
   @POINTER( 9 ) = SOVER;
   @POINTER( 10 ) = SUNDER;
   @POINTER( 11 ) = PRBSCENE;
ENDDATA

END
```

Then using the R interface to LINGO API, we can solve the above stochastic programming model with the following R code.

```
> #load the package
> library(rLingo)
> SP_College <- function(TIME, ASSETS, INITIAL, GOAL, PENALTY, OUTCOMES, O_RETURN)
+ {
+     pResult <- list(ErrorCode = LSERR_NO_ERROR_LNG)
+
+     #Create Lingo enviroment object
+     pLINGO <- rLScreateEnvLng();
+     if(is.null(pLINGO))
+     {
+         cat("\ncannot create LINGO environment!\n")
+         return(pResult)
+     }
+
+     #Open LINGO's log file
```

```
+     pResult <- rLSopenLogFileLng(pLINGO,"SP_College.log")
+     if(pResult$ErrorCode != LSERR_NO_ERROR_LNG)
+     {
+         return(pResult)
+     }
+
+     #Pass memory transfer pointers to LINGO
+     #Define pnPointersNow
+     pnPointersNow = integer(1)
+
+     #@POINTER(1)
+     pResult <- rLSsetCharPointerLng(pLINGO, TIME, pnPointersNow)
+     if(pResult$ErrorCode != LSERR_NO_ERROR_LNG)
+     {
+         return(pResult)
+     }
+
+     #@POINTER(2)
+     pResult <- rLSsetCharPointerLng(pLINGO, ASSETS, pnPointersNow)
+     if(pResult$ErrorCode != LSERR_NO_ERROR_LNG)
+     {
+         return(pResult)
+     }
+
+     #@POINTER(3)
+     pResult <- rLSsetDouPointerLng(pLINGO, INITIAL, pnPointersNow)
+     if(pResult$ErrorCode != LSERR_NO_ERROR_LNG)
+     {
+         return(pResult)
+     }
+
+     #@POINTER(4)
+     pResult <- rLSsetDouPointerLng(pLINGO, GOAL, pnPointersNow)
+     if(pResult$ErrorCode != LSERR_NO_ERROR_LNG)
+     {
+         return(pResult)
+     }
+
+     #@POINTER(5)
+     pResult <- rLSsetDouPointerLng(pLINGO, PENALTY, pnPointersNow)
+     if(pResult$ErrorCode != LSERR_NO_ERROR_LNG)
+     {
+         return(pResult)
+     }
```

```
+
+     #@POINTER(6)
+     pResult <- rLSsetCharPointerLng(pLINGO, OUTCOMES, pnPointersNow)
+     if(pResult$ErrorCode != LSERR_NO_ERROR_LNG)
+     {
+         return(pResult)
+     }
+
+     #@POINTER(7)
+     pResult <- rLSsetDouPointerLng(pLINGO, O_RETURN, pnPointersNow)
+     if(pResult$ErrorCode != LSERR_NO_ERROR_LNG)
+     {
+         return(pResult)
+     }
+
+     lengthTIME = length(strsplit(TIME,' ')[[1]]) / 2
+     lengthASSETS = length(strsplit(ASSETS,' ')[[1]]) / 2
+     numScene = 2 ^ ( lengthTIME - 1 )
+
+     #@POINTER(8)
+     RETURN = array( 0, dim = c( lengthTIME, lengthASSETS, numScene ) )
+     pResult <- rLSsetDouPointerLng(pLINGO, RETURN, pnPointersNow)
+     if(pResult$ErrorCode != LSERR_NO_ERROR_LNG)
+     {
+         return(pResult)
+     }
+
+     #@POINTER(9)
+     OVER = array( 0, dim = c( numScene ) )
+     pResult <- rLSsetDouPointerLng(pLINGO, OVER, pnPointersNow)
+     if(pResult$ErrorCode != LSERR_NO_ERROR_LNG)
+     {
+         return(pResult)
+     }
+
+     #@POINTER(10)
+     UNDER = array( 0, dim = c( numScene ) )
+     pResult <- rLSsetDouPointerLng(pLINGO, UNDER, pnPointersNow)
+     if(pResult$ErrorCode != LSERR_NO_ERROR_LNG)
+     {
+         return(pResult)
+     }
+
+     #@POINTER(11)
```

```
+     PRBSCENE = array( 0, dim = c( numScene ) )
+     pResult <- rLSsetDouPointerLng(pLINGO, PRBSCENE, pnPointersNow)
+     if(pResult$ErrorCode != LSERR_NO_ERROR_LNG)
+     {
+         return(pResult)
+     }
+
+     #Here is the script we want LINGO to run
+     cScript = "SET ECHOIN 1 \n TAKE SP_College.lng \n GO \n QUIT \n"
+
+     #Run the script
+     pResult <- rLSexecuteScriptLng(pLINGO, cScript)
+     if(pResult$ErrorCode != LSERR_NO_ERROR_LNG)
+     {
+         return(pResult)
+     }
+
+     #Close the log file
+     pResult <- rLScloseLogFileLng(pLINGO)
+     if(pResult$ErrorCode != LSERR_NO_ERROR_LNG)
+     {
+         return(pResult)
+     }
+
+     cat( '\n                   Surplus     ' )
+     for( j in 2:lengthTIME )
+     {
+         cat( '              ', strsplit(TIME,split=" ")[[1]][2*(j-1)+1] );
+     }
+     cat("\n")
+
+     cat( '   Scenario        Return      Prob' )
+     for( j in 2:lengthTIME )
+     {
+         cat('     Bond   Stock')
+     }
+     cat("\n")
+
+     X_SURPLUS = 0
+     for( i in 1:numScene )
+     {
+         cat( sprintf( "%10d", i ) )
+         cat( sprintf( "%15.3f", OVER[i] - UNDER[i] ) )
+         cat( sprintf( "%10.3f", PRBSCENE[i] ) )
```

```
+         for( t in 2:lengthTIME )
+         {
+              cat(' ')
+              for( a in 1:lengthASSETS )
+              {
+                  cat( sprintf( "   %4.1f%%", 100*( RETURN[t,a,i] - 1 ) ) )
+              }
+         }
+         cat("\n")
+         X_SURPLUS = X_SURPLUS + PRBSCENE[i] * ( OVER[i] - UNDER[i] )
+     }
+
+     cat( sprintf( "\n   Expected Surplus: %15.3f\n", X_SURPLUS ) )
+
+     #Delete Lingo enviroment object
+     pResult <- rLSdeleteEnvLng(pLINGO)
+     if(pResult$ErrorCode != LSERR_NO_ERROR_LNG)
+     {
+         return(pResult)
+     }
+
+     return(pResult)
+ }
> #All inputs are initialized here
> #Time stages
> TIME = c("T0 \n T1 \n T2 \n T3 \n")
> #Names of investments available
> ASSETS = c("BONDS \n STOCKS \n")
> #Initial capital
> INITIAL = c(55)
> #Goal after three stages
> GOAL = c(80)
> #Penalty/unit short of goal
> PENALTY = c(4)
> OUTCOMES = c("GOOD \n BAD \n")
> O_RETURN = c(1.14, 1.25, 1.12, 1.06)
> #Run the function
> SP_College(TIME, ASSETS, INITIAL, GOAL, PENALTY, OUTCOMES, O_RETURN)

In Error Callback: Error 11:
  Invalid input.  A syntax error has occurred.

      1]
```

```
$ErrorCode
[1] 11
```

5 Troubleshooting

For some error messages, users may refer to the following table for fixing.

Problem	How to Fix
Unable to open file	1. File type consistency: LINGO supports two file types, .lng and .lg4. Make sure that if you edit a LINGO file, you save it in a format consistent with what is used in your R code. E.g., if your R code says: *cScript = "SET ECHOIN 1 \n TAKE PORTCardCor.lng"* then make sure if you edit PORTCardCor, to save it in .lng, not .lg4 format. 2. Check whether the specified file is in the same folder of the R file.
No Lingo log file	Allow Lingo write permission to the folder where the Lingo log file will be sent.