

A Quick Start for The R Interface to LINGO API

September 23, 2013

1 Introduction

The package *rLingo* is an R interface to LINGO API functions, which gives you the ability to run a LINGO command script to access all the major features of LINGO.

LINGO is an algebraic modeling language linked to an array of optimization engines, or solvers. It is a tool for utilizing the power of linear, integer and 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

To install the package, it requires the installation of LINGO 14.0 as well. See file INSTALL for details of the installation and platform specifications.

3 Usage

The R interface function names use the convention of 'r' + name of LINGO API function, e.g. *rLScrateEnvLng* in the R interface corresponds to *LScrateEnvLng* in LINGO API.

4 General commands

To load the package, use the command:

```
> library(rLingo)
```

To generate a LINGO API environment object, use the command:

```
> rEnv <- rLScreeEnvLng()
```

5 An application to the Acceptance Sampling Design

We are sampling items from a large lot. 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". 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 need to determine N and C, where N is the minimal sample size, and C is the critical level of defects such that, if defects observed in the sample are less-than-or-equal-to C, we accept the lot. Note that we make use of LINGO's cumulative Poisson distribution function as an approximation of the Binomial distribution.

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); ! "Good" lot fraction defective;  
LTFD = @POINTER( 2); ! "Bad" lot fraction defective;  
PRDRISK = @POINTER( 3); ! Tolerance for rejecting good lot;  
CONRISK = @POINTER( 4); ! Tolerance for accepting bad lot;  
MINSMP = @POINTER( 5); ! Lower and upper bounds on sample size;  
MAXSMP = @POINTER( 6);
```

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.

```

#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)
+   }
+
+   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)
+   }
+
+   #check solution
+   cat("\nThe Optimal sample size is",N,".\nAccept the lot if",C,"or less defectives in sample")
+
+   #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.)
```

Local optimum found!

The Optimal sample size is 197 .

Accept the lot if 9 or less defectives in sample.

\$ErrorCode

[1] 0