Journal of Statistical Software
March 2008, Volume 25, Issue 6. http://www.jstatsoft.org/

Software for Implementing the Sequential Elimination of Level Combinations Algorithm

Kjell Johnson
Pfizer Global Research and Development

Abhyuday Mandal
University of Georgia

Tan Ding
University of Georgia

Abstract

Genetic algorithms (GAs) are a popular technology to search for an optimum in a large search space. Using new concepts of forbidden array and weighted mutation, Mandal, Wu, and Johnson (2006) used elements of GAs to introduce a new global optimization technique called sequential elimination of level combinations (SELC), that efficiently finds optimums. A SAS macro, and MATLAB and R functions are developed to implement the SELC algorithm.

Keywords: genetic algorithms, high-dimensional optimization, SAS, R, MATLAB.

1. Introduction

Locating optimal values in a large search space is a primary goal for many scientific problems. In the pharmaceutical industry, for example, work has primarily focused on synthesizing new compounds that are effective against a particular disease or condition. Historically, chemists have used scientific knowledge to build new compounds that have promising properties for alleviating a condition or disease. While compounds may be designed to be theoretically effective, often their interactions with other parts of the body (e.g., liver, kidney, intestine, etc.) render them ineffective or toxic (Welling 1997). Hence, physically creating one compound at a time has become a less effective drug discovery approach.

As robotic technology has improved, chemists have been able to synthesize and explore a large number of new compounds. This technology, known as combinatorial chemistry, has been widely applied in the pharmaceutical industry and is gaining interest in other areas of chemical manufacturing (Leach and Gillet 2003; Gasteiger and Engel 2003). In short, combinatorial chemistry uses robotics to combine sets of monomers to create thousands of new compounds at a time. This technology has been used to enhance the diversity of compound libraries, to explore specific regions of chemical space (i.e., focused library design), and to optimize one or more pharmaceutical endpoints such as target efficacy or ADMET (absorption, distribution, metabolism, excretion, toxicology) properties (Rouhi 2003).

In a typical combinatorial chemistry problem, a core molecule is identified to which monomers are attached at multiple locations. Each attachment location may have tens or hundreds of potential monomers. Clearly, the full combinatorial library can become dauntingly large for a core molecule with just a few attachment points. Constrained by resources, most combinatorial libraries cannot be fully created. Instead, chemists use their scientific knowledge to identify the most promising monomer combinations and the monomer combinations to avoid. By construction, the SELC is an ideal method for searching for optimal molecules in combinatorial chemistry (Mandal *et al.* 2006).

The SELC method is also useful in computer experiments. Much research which in the past decades could only be conducted by performing physical experiments, can now be done by computer experiments instead. In a computer experiment, a response, $y(x)$, is calculated for each set of input variables, x, using numerical methods implemented by (complex) computer code (Santner, Williams, and Notz 2003). In such cases, the complex numerical method can be thought of as a "black box", and the SELC method can be used to select the optima efficiently.

We can consider such real-life scenarios as large-dimensional design-of-experiment problems where the main challenge is to identify the optimal design settings. In scientific and engineering research, statistical design and analysis of experiments is an effective and commonly used tool to understand and/or improve a system. Identifying important factors and choosing factor levels are among the first and most fundamental issues for an experimenter. But when there are a large number of important factors, designing an experiment can be difficult. Classical experimental design relies heavily on algebraic properties such as orthogonality (Hedayat, Sloane, and Stufken 1999). However, orthogonality does not allow the flexibility to accommodate all kinds of promising follow-up runs, which, in turn makes finding suitable designs for large-scale problems difficult, particularly when the factors have more than two levels.

The use of high-fidelity computer simulations of physical phenomena (Bates, Buck, Riccomagno, and Wynn 1996) has stimulated new research into ways in which experimental design can be applied to such problems. Greedy algorithms are popular choices for these types of problems (Cormen, Leiserson, Rivest, and Stein 2001). In short, a greedy algorithm identifies the direction of an optimum at each stage and searches toward it. But for a complex response surface, the identified direction of the optima may not be the direction of the global optimal. However assuming some kind of regularity, the global maximum will not be in the vicinity of the local minimum and vice versa. One technique, motivated by design of experiments, was introduced by Wu, Mao, and Ma (1990), and is known as sequential elimination of levels (SEL). The idea of SEL is opposite to that of greedy algorithms; instead of focusing on factor levels that improve the response, SEL focuses on those levels that worsen the response. Based on this idea, SEL eliminates one level of each factor in each sequence of the experiment. When all factors are independent and the response surface is smooth, SEL can accurately locate optimal design points. However, when factors interact or the response surface contains local optimums, SEL does not accurately identify optimal design points. In these more realistic situations, SELC, the modified version of SEL using ideas from GAs, has been shown to be able to be a more effective optimization technique.

GAs have most often been viewed from a biological perspective. The metaphors of natural selection, cross breeding, and mutation have been helpful in providing a structure to explain how and why GAs work. Thus, most practical applications of GAs are rooted in the context where optimization is the primary goal. In order to understand how GAs function as optimizers, Reeves and Wright (1999) considered GAs to be a form of sequential experimental design. Recently, GAs have been successfully applied in solving statistical problems, particularly for searching for near-optimal designs (Hamada, Martz, Reese, and Wilson 2001; Heredia-Langner, Carlyle, Montgomery, Borror, and Runger 2003; Heredia-Langner, Montgomery, Carlyle, and Borror 2004). Because of the desire to explore increasing large experimental spaces in the pharmaceutical and engineering industries, there is an imminent need for software that implements new methodology (Willis 2007).

In this work we develop software across three widely used platforms: SAS (SAS Institute Inc. 2003), R (R Development Core Team 2008), and MATLAB (The MathWorks, Inc. 2007). Each implementation is straightforward and simple to use, and only requires the base platform of each software. Each function is freely available along with this paper at `http://jstatsoft.org/v25/i06/`. The article is organized as follows: Firstly, we review the GAs and the SELC method in Section 2. In Section 3, we provide the implementation of the algorithm in SAS, MATLAB, and R, and illustrate the algorithm using all three implementations. Finally, we conclude this work in Section 4.

2. Methodology review

2.1. Genetic algorithms

Genetic algorithms (GAs) are a stochastic optimization tool that were inspired by Darwin's theory of evolution (Holland 1975, 1992). The basic idea of GAs is to solve optimization problems via an evolutionary process which results in better solutions based on good solutions. The basic steps of a GA are as follows:

1. Solution representation: Each observation must be represented by a chromosome that defines its characteristics.
2. Selection: Identify the best chromosomes based on a fitness criterion (e.g., the larger the response, the better the fitness for a maximization problem).
3. Reproduction: Two chromosomes are randomly chosen, weighted by their fitness values, and the following operations are performed.
   (a) Crossover: Split the pair of chromosomes at a same random location, and combine the head part of one with the tail of the other and vice versa.
   (b) Mutation: Change the level of randomly chosen factor(s) for the newly created offspring.
4. Repeat steps 2 and 3 until some convergence or stopping criterion is met.

To illustrate how a GA works, consider optimizing the function,

$$f(x) = sin^2\left(\frac{x}{18}\right) + cos\left(\frac{x}{18}\right),$$

over the range of integers $x = 0, 1, \ldots, 31$. Solving this problem first requires that the range of solutions be represented by a chromosome. Consider the following binary representation for the integers:

$$\begin{array}{ccccccc} 0 & = & 0 & 0 & 0 & 0 & 0 \\ 1 & = & 0 & 0 & 0 & 0 & 1 \\ 2 & = & 0 & 0 & 0 & 1 & 0 \\ & & & & \vdots & & \\ 31 & = & 1 & 1 & 1 & 1 & 1 \end{array}$$

For this example, suppose x values of 6, 10, 20, 26, and 29 are selected and evaluated:

$$\begin{array}{lclcl} f(6) & = & f(00110) & = & 1.052 \\ f(10) & = & f(01010) & = & 1.128 \\ f(20) & = & f(10100) & = & 1.247 \\ f(26) & = & f(11010) & = & 1.110 \\ f(29) & = & f(11101) & = & 0.958 \end{array}$$

Based on their fitness scores, x values of 10 and 20 have the highest probability of being selected, and without loss of generality, suppose that these two chromosomes are selected. If the randomly selected crossover location is between the second and third gene from the left, then the new offspring would be:

$$\begin{array}{ccccc} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \end{array}$$

Finally, each gene has a positive probability of mutation to another value. After mutation, these new offspring become

$$\begin{array}{ccccc} 0 & 1 & 1 & 0 & \mathbf{1} \\ 1 & 0 & \mathbf{1} & 1 & 0 \end{array}$$

Evaluating the new offspring yields $f(01101) = f(13) = 1.187$ and $f(10110) = f(22) = 1.225$.

2.2. SELC algorithm

As mentioned in the Introduction, the SELC was proposed as an alternative to the SEL when seeking optimums in a high dimensional space where factors interact or where local optimums exist. The unique features of the SELC that allow it to find optimums in these settings are the forbidden array and weighted mutation scheme and these concepts are summarized below.

Forbidden array

The forbidden array is defined by its strength and order and contains design points that have demonstrated poor fitness values or are a priori known to produce undesirable fitness values. The strength of the array defines the number of design points placed into the array at each iteration of the algorithm. More specifically, a forbidden array with strength s consists of the s worst runs of the experiment at each iteration of the algorithm. The order of the forbidden array defines the factor combinations to be prevented from being constructed in subsequent iterations of the algorithm. A forbidden array with order k, for example, indicates that any combination of k or more levels from any design point in the forbidden array be prevented from being constructed in subsequent iterations of the algorithm. Thus, strength and order define the size of the forbidden array: as strength increases and order decreases, the number of forbidden design points increases.

We shall use an example from combinatorial chemistry to illustrate the construction of the forbidden array. On a core molecule, suppose that three monomers (denoted by 1, 2, and 3) can be added to each of three locations (denoted by A, B, and C). Note that in this example the monomers act as genes and form the chromosomes, which are the compounds to be evaluated. Suppose that 9 of the possible 27 compounds are created and analyzed (y is the fitness criterion):

A	B	C	y
1	1	1	10.1
1	2	2	53.6
1	3	3	43.8
2	1	2	13.4
2	2	3	46.9
2	3	1	55.1
3	1	3	5.7
3	2	1	43.6
3	3	2	47.0

In this case, the objective is to find compounds that maximize y. Therefore, the forbidden array will consist of compounds that have the lowest values of y. If the forbidden array has strength = 2 and order = 2, then it is constructed using the compounds

A	B	C
1	1	1
3	1	3

and is defined as

A	B	C
1	1	*
1	*	1
*	1	1
3	1	*
3	*	3
*	1	3

where * is the wildcard which represents any admissible value.

Weighted mutation

The second feature that makes the SELC algorithm unique is its weighted mutation scheme. In a traditional GA, genes at each loci mutate at random with small positive probability.

This approach is unbiased toward any level of any factor in the experiment. However, upon collecting data we begin to gain insight about the effect of each factor or factor combinations on the fitness criterion. This information can then be used to allow the algorithm to focus on factors and levels of factors that improve the fitness criterion. In the weighted mutation scheme of the SELC algorithm, we use linear regression to determine significant main effects and pairwise interactions. (Because of the effect hierarchy principal (Wu and Hamada 2000), we do not explore beyond pairwise interactions.) If a factor, F_j, has a significant main effect and no significant pairwise interactions, then the mutation probability for each level, l, of the factor is proportional to the average fitness of that level for the data collected thus far in the experiment

$$p_{j_l} \propto \bar{y}(F_j = l), \quad \text{for } j = 1, 2, \ldots, J, \text{ and } l = 1, 2, \ldots, L.$$

If two factors, F_j and F_k, have a significant interaction, then the probability of mutation is joint on F_j and F_k. When either factor is chosen, then the mutation will be weighted jointly with probability

$$p_{j_l k_m} \propto \bar{y}(F_j = l, F_k = m), \quad \text{for } j, k = 1, 2, \ldots, J, \text{ and } l, m = 1, 2, \ldots, L.$$

If the selected factor does not have a significant main effect or interaction, then its value will be changed to any possible level with equal probability.

The SELC algorithm

The SELC algorithm combines orthogonal arrays, genetic algorithms, forbidden arrays, and weighted mutations as follows:

1. Begin with an initial design. Given no prior information about the design space, the SELC should be initiated with an orthogonal array, which helps to estimate a large number of factor effects efficiently. If there is prior knowledge about the design space or about design points that should initially be included in the forbidden array, then the algorithm should begin at step 3.
2. Create the selected design points; stop if the stopping criterion is achieved (see below).
3. Construct the forbidden array and calculate the weighted mutation probabilities.
4. Determine the new offspring using a genetic algorithm with weighted mutation probabilities.
5. Check offspring eligibility. An offspring is eligible if it is not prohibited by the forbidden array and has not been previously created. If an offspring is not eligible, then discard it and return to step 4.
6. Repeat steps 4 and 5 until the number of desired new offspring are achieved.
7. Return to step 2.

The stopping rule is subjective and depends on the progression of the algorithm and experimental constraints. As runs are added, the experimenter can monitor the progress towards optimization of the fitness criterion. Once a satisfactory level of fitness has been achieved the algorithm can be stopped. Alternatively, resources may limit the number of iterations of the algorithm.

3. Code description

To make the SELC algorithm available to a wide range of users, we have implemented it in SAS, R, and MATLAB. These are available along with this paper. Each implementation requires a data set that contains the initial design points and measured response. In addition, for all three implementations, the user can specify an a priori forbidden array.

3.1. SAS implementation

For the `SELC` macro, the factors in the initial design data set must have the variable names X1, X2, ..., Xn, and the response must be named Y. Similarly, the factors in the forbidden array data set must have the variable names X1, X2, ..., Xn.

Prior to calling the `SELC` macro, the user must first initialize a sequence of global macro variables that define the number of levels of each factor. For example, if an experiment has four factors, each with ten levels, then the user must specify the following code:

```
%let L1 = 10;
%let L2 = 10;
%let L3 = 10;
%let L4 = 10;
%global L1 L2 L3 L4;
```

The user can the call the `SELC` macro, which has the form:

```
 %SELC( DFILE =,
 FFILE =,
 FARRAY =,
 NUMOFF =,
 STRENGTH =,
 ORDER =,
 NUMFACT =,
 DIR =,
 SEED =);
```

Macro arguments are defined as:

`DFILE`	=	Dataset of the initial design.
`FFILE`	=	Dataset of the forbidden array (if available).
`FARRAY`	=	Name of the output data set that will contain the forbidden array plus the design points specified by `STRENGTH`.
`NUMOFF`	=	Number of desired offspring.
`STRENGTH`	=	Strength of the forbidden array.
`ORDER`	=	Order of the forbidden array.
`NUMFACT`	=	Total number of factors in the experiment.
`DIR`	=	Direction of desired response: `1` if maximum response is desired, `0` if minimum response is desired.
`SEED`	=	Seed value for random operations. Must be a non-negative integer. Default value is clock time.

When the macro is called, it generates a data set named `NEXT_GEN` that contains the design points to be performed in the next experiment.

3.2. R and MATLAB implementation

For both the R and MATLAB code, the initial design can be provided in a text file. The file should not have a header; instead, the column position indicates the factors $X_1, X_2, \ldots, X_n$, and the last column should contain the response, y. For R and MATLAB, the user can also specify forbidden array data set, where the column position corresponds to each factor.

The call to the R and MATLAB functions have the same form:

```
SELC(initialdesign, forbiddenarray, strength, order, level_true,
     direction, number_of_offspring, seed)
```

The arguments to these functions are defined similarly as in the SAS macro.

3.3. Example

An example from a combinatorial chemistry problem is used to illustrate how the `SELC` software works. Onto a core molecule, monomers can be added to two positions (Figure 1). Five monomers are selected for the core, 34 for location A, and 241 for location B; the objective is to find combination(s) of monomers that produce highly active compounds. In this experiment, we start with an initial design and a priori forbidden array. The initial design and forbidden arrays for this example are available electronically along with this paper.

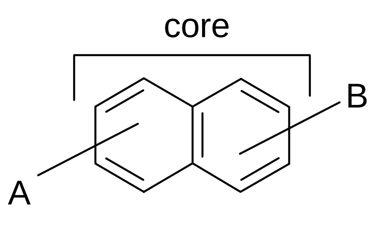

Figure 1: Hypothetical example from combinatorial chemistry.

SAS example

To generate the next set of five design points using the SELC SAS macro, we must specify the following code:

```
%LET L1=5;
%LET L2=34;
%LET L3=241;
%GLOBAL L1 L2 L3;

%SELC(dfile    = initialdesign,
    ffile    = forbiddenarray,
    farray   = currentforbiddenarray,
    numoff   = 5,
    strength = 2,
    order    = 2,
    numfact  = 3,
    dir      = 1,
    seed     = 1);
```

Upon submitting this code, SAS creates the NEXT_GEN data set that contains 5 new offspring:

```
Obs          x1           x2           x3
  1           4           13           40
  2           3           19           14
  3           3            6          223
  4           1           28           30
  5           2            9           13
```

And the updated forbidden array, CURRENTFORBIDDENARRAY, contains the following design points:

```
x1          x2          x3
 3          10           3
 1          13           9
 1          23          10
             .
             .
             .
 5          20         182
```

Often no information will be available for the forbidden array. If this is the case, simply create a forbidden array data set that contains the names of the factors, but does not contain any design points. For example:

```
data forbiddenarray;
  input x1-x3;
cards;
run;
```

R *example*

We use the same data set to illustrate the R code. First, we define all of the arguments:

```
R> initialdesign <- as.matrix(read.table("initialdesign.txt"))
R> forbiddenarray <- as.matrix(read.table("forbiddenarray.txt"))
R> strength <- 2
R> order <- 2
R> level_true <- c(5, 34, 241)
R> direction <- 1
R> number_of_offspring <- 5
R> seed <- 1
```

The call to the function is:

```
R> next_gen <- SELC(initialdesign, forbiddenarray, strength, order,
                    level_true, direction, number_of_offspring, seed)
```

The object, *next_gen*, consists of the new designs points to be performed in the next generation as well as the updated forbidden array:

```
next_gen
$newruns
          V1 V2  V3
    p1new  2  4  49
    p2new  3 24 228
    p1new  1 22   1
    p2new  3 23  93
    p1new  4  4  92
$currentforbiddenarray
   V1  V2  V3
1   1  23  10
2   1  23  23
3   1  25  10
        .
        .
        .
```

If no information exists for the forbidden array, then define an array that does not include levels that exist in the original design. For this example, the following forbidden array is sufficient:

```
R> forbiddenarray <- cbind(0, 0, 0)
```

MATLAB *example*

Finally, we illustrate the code for MATLAB. Similarly to R, we define the arguments to the function.

```
>> load -ascii initialdesign.txt
>> load -ascii forbiddenarray.txt
>> strength = 2
>> order = 2
>> level_true = [5,34,241]
>> direction = 1
>> number_of_offspring = 5
>> seed=0
```

Then, the SELC function will generate offspring as follows:

```
>> [nextgen,currentforbiddenarry] = SELC(initialdesign,forbiddenarray, ...
          strength,order,level_true, direction,number_of_offspring,seed)

nextgen =
     1     6   226
     2    18    30
     2     6     6
     1    32    28
     3     6   127
```

Similar to SAS and R, the updated forbidden array is contained in currentforbiddenarray. If no forbidden array information exists, then define:

```
>> forbiddenarray = [];
```

4. Summary and concluding remarks

Because experimental design spaces are becoming increasingly large, there is an imminent need for methods to efficiently explore these spaces and for software to implement these methods. Indeed, selecting an optimal design in an extremely large search space is not an easy or straightforward task. For these types of problems, the SELC algorithm has been shown to efficiently and effectively identify optimums. To make the implementation of this method widely available, we have introduced software for implementing the SELC algorithm across three commonly used platforms in both academics and industry: SAS, R, and MATLAB. Each implementation is straightforward and easy to use, requiring only that data sets be prepared in the same format for input to each piece of software.

It is important to note that the software implementations provided in this work have been optimized for large design spaces consisting of a relatively small number of factors each containing a large number of levels. In addition, SELC is not designed to optimize a specific objective function. Instead, it is designed to identify new, promising, feasible design points for follow-up. Lastly, because each software package use different random number generators, the next generation of suggested design points will differ among SAS, R, and MATLAB. However, each implementation will identify key level combinations that are important for predicting the response.

Acknowledgments

The authors are thankful to Dirk Bornemeier of Pfizer Global Research and Development, Michigan Laboratories, who helped focus on the Pharmaceutical application. The authors are also thankful to C. F. Jeff Wu of Georgia Institute of Technology for his valuable suggestions and comments. The second author's research is supported by grants from the University of Georgia Research Foundation.

References

Bates RA, Buck RJ, Riccomagno E, Wynn HP (1996). "Experimental Design and Observation for Large Systems." *Journal of the Royal Statistical Society Series B*, **58**(1), 77–94.

Cormen TH, Leiserson CE, Rivest RL, Stein C (2001). *Introduction to Algorithms.* MIT Press, Cambridge, MA, 2nd edition.

Gasteiger J, Engel T (2003). *Chemoinformatics: A Textbook.* Wiley-VCH, Weinheim.

Hamada M, Martz HF, Reese CS, Wilson AG (2001). "Finding Near-Optimal Bayesian Experimental Designs via Genetic Algorithms." *The American Statistician*, **55**(3), 175–181.

Hedayat AS, Sloane NJA, Stufken J (1999). *Orthogonal Arrays: Theory and Applications.* Springer-Verlag, New York.

Heredia-Langner A, Carlyle WM, Montgomery DC, Borror CM, Runger GC (2003). "Genetic Algorithms for the Construction of D-Optimal Designs." *Journal of Quality Technology*, **35**(1), 28–46.

Heredia-Langner A, Montgomery DC, Carlyle WM, Borror CM (2004). "Model-robust Optimal Designs: A Genetic Algorithm Approach." *Journal of Quality Technology*, **36**(3), 263–279.

Holland JH (1975). *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence.* University of Michigan Press, Ann Arbor.

Holland JH (1992). *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence.* Complex Adaptive Systems. MIT Press, Cambridge, MA, 1st edition.

Leach AR, Gillet VJ (2003). *An Introduction to Chemoinformatics.* Kluwer Academic Publishers, London.

Mandal A, Wu CFJ, Johnson K (2006). "SELC: Sequential Elimination of Level Combinations by Means of Modified Genetic Algorithms." *Technometrics*, **48**(2), 273–283.

R Development Core Team (2008). *R: A Language and Environment for Statistical Computing.* R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0, URL `http://www.R-project.org/`.

Reeves CL, Wright CC (1999). "Genetic Algorithms and the Design of Experiments." In LD Davis, K DeJong, MD Vose, LD Whitley (eds.), "Evolutionary Algorithms," pp. 207–226. Springer-Verlag, New York.

Rouhi AM (2003). "Custom Synthesis for Drug Discovery." *Chemical and Engineering News*, **81**(7), 75–78.

Santner TJ, Williams BJ, Notz W (2003). *Design and Analysis of Computer Experiments.* Springer-Verlag, New York.

SAS Institute Inc (2003). *SAS/STAT Software, Version 9.1.* Cary, NC. URL http://www.sas.com/.

The MathWorks, Inc (2007). *MATLAB – The Language of Technical Computing, Version 7.5.* The MathWorks, Inc., Natick, Massachusetts. URL http://www.mathworks.com/products/matlab/.

Welling PG (1997). *Pharmacokinetics: Processes, Mathematics, and Applications.* American Chemical Society, Washington.

Willis R (2007). "Finding Needles." *Drug Discovery News.* 2007-05-21, URL http://www.drugdiscoverynews.com/index.php?newsarticle=1318.

Wu CFJ, Hamada M (2000). *Experiments: Planning, Analysis, and Parameter Design Optimization.* John Wiley & Sons, New York.

Wu CFJ, Mao SS, Ma FS (1990). "SEL: A Search Method Based on Orthogonal Arrays." In S Ghosh (ed.), "Statistical Design and Analysis of Industrial Experiments," pp. 279–310. Marcel Dekker Inc., New York.

Affiliation:

Kjell Johnson
Pfizer Global Research and Development
Ann Arbor, MI 48105, United States of America
E-mail: Kjell.Johnson@pfizer.com

Abhyuday Mandal
Department of Statistics
University of Georgia
Athens, GA 30602-1952, United States of America
E-mail: amandal@stat.uga.edu
URL: http://www.stat.uga.edu/~amandal/

Journal of Statistical Software http://www.jstatsoft.org/
published by the American Statistical Association http://www.amstat.org/

Volume 25, Issue 6 *Submitted:* 2007-08-23
March 2008 *Accepted:* 2008-02-24