An R Package for Group Sequential Boundaries Using Alpha Spending Functions

T. Charles Casper

December 7, 2021

1 Summary

Reboussin *et al.*[1] created a `FORTRAN` program, `ld98`, which performs computations related to group sequential analysis in clinical trials using spending functions. The original method was proposed by Lan and DeMets [2] and the method has also been described in [3]. These kinds of methods have been developed because of the ethical responsibility to monitor accumulating data during a clinical trial. The trial stops early if an interim analysis reveals a large enough difference between treatment groups. The Fortran program is interactive and has four options which compute: (1) bounds given analysis times and a spending function, (2) drift parameters corresponding to given power and bounds, (3) probabilities given times, bounds, and drift parameters, and (4) a confidence interval given times, bounds, and final test statistic value. The program consists of one main program, an input routine, 17 subroutines, and 5 functions related to probability and probability density calculations.

We created an `R` package [5], `ldbounds`, which performs essentially the same tasks, plus has some additional functionality. The functions in the package are not interactive in the same way as the Fortran program (prompting for inputs, etc.). This

package has some advantages: it uses R's capability to work with the "whole object" as opposed to always using loops and also uses R's built in probability distribution calculations, it has better output graphics, and it allows for more freedom in specifying spending functions. The current version has many updates and improvements over the previous version, as discussed in Section 4.

2 Introduction

In most clinical trials, data are accumulated over a period of time that could be years. In these types of experiments there is a lot at stake. The amount of money and other resources spent is very large. Also, humans are the subjects of the experiments, leading to a very sensitive ethical situation. For these reasons, a clinical trial should not go on longer than is necessary to show a significant result with the accumulating data. In fact, most clinical trials experience interim review of data by an independent Data and Safety Monitoring Board (DSMB). This board may decide to recommend early termination of the trial. However, significance relies on the expected frequency of certain events under the null hypothesis. If a trial is monitored at several analysis times, the probability of observing a "significant" result increases. It has been shown that, with 10 analysis times, the overall type I error rate of 5% increases to about 20%.[4]

This is the motivation behind group sequential methods. The goal is to allow for several looks of the data throughout the course of a trial and still control the overall type I error rate. Prior to the alpha spending function method, other techniques had several limitations. Among these was the need to specify the number and exact times of all interim analyses before a trial began. The alpha spending function method, proposed by Lan and DeMets [2, 3], is much more flexible and interim analysis times can be specified or adjusted at any time during the trial. The basic concept is to define a function which determines how much of the total type I error can be "spent" at each analysis, based only on the previous analysis times and the amount of change in the

specified function since the last time. It should be mentioned that these methods are not always strictly followed by a DSMB, but instead they are used more as guidelines. Lan and DeMets also created a `FORTRAN` program to make computations related to alpha spending functions and group sequential boundaries. In more recent years, this program was revised and improved in terms of efficiency [1]. Our goal was to create an `R` package to accomplish the same computations and give more flexibility (see Section 1).

3 Methods

Formally, the group sequential method is the following: In a clinical trial, define a set of critical values $\{Z_C(k), k = 1, \ldots, K\}$, one critical value at each analysis, such that the trial is to continue if the observed statistic $Z(k)$ satisfies $|Z(k)| < Z_C(k)$ (two-sided test) and the overall type I error rate is α.

To see how the alpha spending function method works in the context of a one-sided test, let $B(t)$ be a standard Brownian motion process on $[0, 1]$. Next, let τ be the stopping time defined by $\tau = \inf\{t : B(t) > b\}$, for some fixed b. Now, let

$$\alpha^*(t) = P(\tau \leq t) = 2 - 2\Phi\left(b/\sqrt{t}\right) I\{0 < t \leq 1\}.$$

We notice that, if $b = z_{\alpha/2}$, the $1 - \alpha/2$ quantile of the standard normal distribution, then $\alpha^*(1) = \alpha$. Now, assume that $B(t)$ is only observed at times $\{t_i : i = 1, \ldots, K; 0 < t_1 < \cdots < t_K = 1\}$. Define b_1 to satisfy

$$P\left(B(t_1) > b_1\right) = P\left(\tau \in (0, t_1]\right) = \alpha^*(t_1).$$

Similarly, define $b_2, \ldots, b_K$ such that for $i = 2, \ldots, K$

$$P\left(B(t_i) > b_i,\ B(t_j) < b_j,\ j = 1, \ldots, i-1\right) = P\left(\tau \in (t_{i-1}, t_i]\right) = \alpha^*(t_i) - \alpha^*(t_{i-1}).$$

The idea is that the amount that α^* has increased since the last analysis time is alotted to the boundary crossing probability at the current analysis time. It is a very

straightforward task to show that

$$b_1 = \sqrt{t_1}\Phi^{-1}\left(2\Phi\left(b/\sqrt{t_1}\right) - 1\right).$$

However, finding $b_2, \ldots, b_K$ involves computing densities iteratively [4], similar to

$$f_i(t) = \int_{-\infty}^{b_{i-1}} f_{i-1}(u)\frac{1}{\sqrt{2\pi}}\exp\left\{-\frac{1}{2}(t-u)^2\right\}\, du$$

This is done by numerical integration. Then, the result is integrated and the optimal boundary is found by searching for the limit of integration.

This situation assumes fixed b. The general method can account for $b(t)$. Of course, in practice, the function α^* (the alpha spending function) is specified. As long as this function is increasing and satisfies $\alpha^*(0) = 0$ and $\alpha^*(1) = \alpha$, the desired type I error level, then the function has the desired properties to lead to a boundary. We notice that at each step, the boundary can be calculated based on only the previous values and times and the change in α^*. Thus, the exact number and placement of analysis times need not be specified in advance, but instead can be specified at any time. Also, the method has only been described for a one-sided test. For a symmetric two-sided test, $\alpha^*/2$ can be used and then the negative of the boundary calculated is used for the lower boundary. In a similar way, the method can be applied with different α^*'s used for the upper and the lower boundary. There are many statistical techniques that are needed to apply these methods in cases when the data come in different forms [1].

4 Updates in Version 2.0.0

1. The main functions from the previous version, `bounds` and `drift`, were renamed `ldBounds` and `ldPower`.

2. Print/summary Methods

   (a) New print methods for classes "ldBounds" and "ldPower".

(b) Edited summary method for "ldPower": to give power when drift is known

3. Better two-sided bounds and defaults

   (a) Created `sides` argument for `ldBounds`

   (b) Added ability to specify the number of looks, which then automatically creates equally spaced analysis times

   (c) `Sides=2`, but with only one alpha, etc., creates symmetric bounds, splitting the provided alpha in half for each, upper and lower bounds

   (d) Print and summary methods updated to account for this

4. `ldPower` to use output from `ldBounds`

   (a) `ldPower` can now accept an "ldBounds" object as the first argument, in which case it does not require times and boundaries

   (b) Defaults to drift of 0 when nothing is specified

5. Nominal alpha level at each look provided (for one-sided or symmetric bounds)

   (a) Bounds now has additional value: nominal alpha at each look.

   (b) Summary method displays this

6. `asf` changed from asf/alpha only `asf`

   (a) `asf` now doesn't use alpha (this was only a change to `alphas` function)

7. Strage integer errors

   (a) ldBounds and alphas now have checks that take into account floats

8. Ability to specify non-alpha spending boundaries

   (a) Created new function `commonbounds` to calculate Pocock or O'Brien-Fleming (non-alpha-spending) bounds at equally spaced times.

   (b) These also become "ldBounds" objects

(c) Adjusted print and summary methods to accommodate

9. Calculate adjusted p-values

   (a) Added p-value functionality to ldPower function; stepwise and likelihood ratio ordering available.

   (b) Uses a new `adj.p` function which, ironically, calls `ldPower`.

   (c) Edited printing and summary functions to use this as well.

10. Graphics

    (a) Now uses base graphics, very simple

    (b) Option to show on either z-value or b-value scale

    (c) Ability to add to existing plot

    10. Conditional power

        (a) Added a function to calculate simple conditional power

5 The `ldbounds` Package

The `R` package `ldbounds` is somewhat based on the `FORTRAN` program to do calculations related to group sequential boundaries using spending functions. It consists of 27 functions: 8 method functions, 14 utility functions and 5 main functions (See Figure 1). The main functions are `ldBounds`, `ldPower`, `lastbound`, `commonbounds`, and `condpower`.

The package is useful in four different categories of situations: (1) to calculate boundaries (both standard boundaries and given spending functions), (2) to calculate the final boundary for given interim boundaries, (3) to compute treatment effect for a given power and boundaries, (4) to calculate probabilities (e.g., power) for given boundaries and treatment effect, (5) to compute a confidence interval or p-value at the end of the trial, adjusted for multiple looks, and (6) to calculate conditional

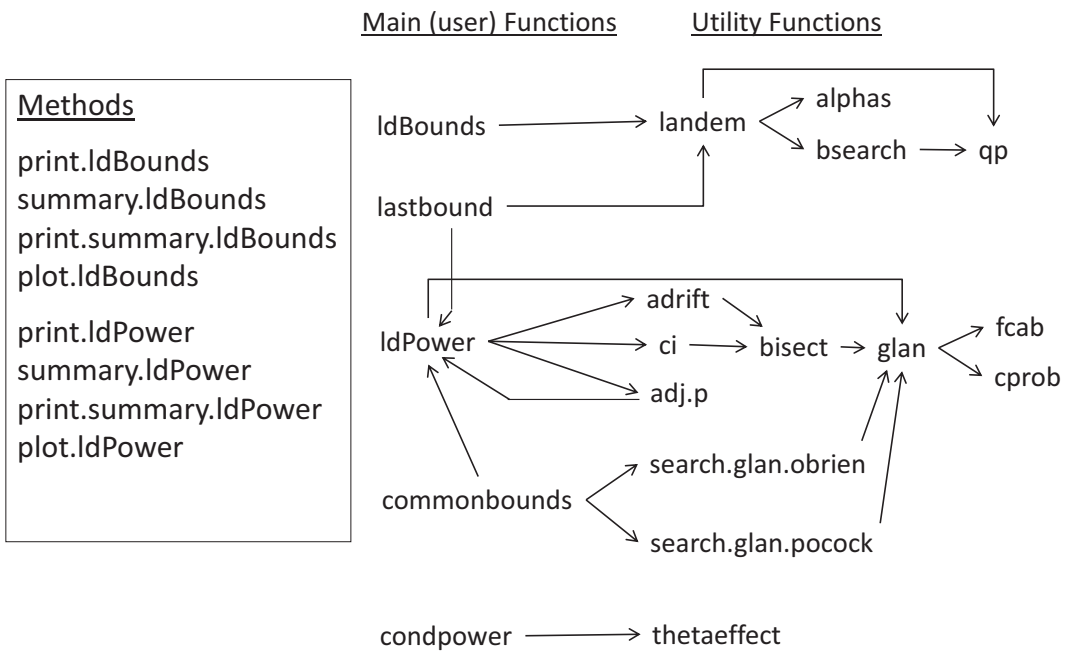

Figure 1: *Package* `ldbounds`. Schematic description of relationships between `ldbounds` functions.

power at an interim time. The functions `ldBounds` and `commonbounds` are used in the first case, `lastbound` is used in the second, `ldPower` can help us in cases 3–5, and `condpower` is used in the last scenario.

5.1 The `ldBounds` Function

The function `ldBounds` determines group sequential boundaries for interim analyses of accumulating data in clinical trials using the Lan-DeMets alpha spending function method. These can be used as guidelines for early stopping of the trial. The following statement shows how to specify a function call:

```
ldBounds(t, t2, iuse = 1, asf = NULL,
         alpha = 0.05, phi = rep(1, length(alpha)),
         sides = 2, ztrun = rep(8, length(alpha)))
```

Here `t` corresponds to the vector of analysis times, which must be increasing and in (0,1], or a number of analysis times (assumed to be equally spaced); `t2` is the second time scale, usually in terms of amount of accumulating information, by default `t2` is the same as `t`; `iuse` designates the type of alpha spending function(s)or the values to use for lower and upper bounds, respectively (in the two-sided case). The program allows the user to specify the spending function from four different predefined options, or the user can also specify a new spending function in which case `asf` must be used. `alpha` is the type I error (lower and upper in two-sided situations). The overall alpha must be greater than 0 and less than or equal to 1. `phi` is a vector of values used when `iuse`=3 or 4, `sides` designates one- or two-sides boundaries, and `ztrun` is a vector of values specifying where to truncate lower and upper boundaries, respectively, with a default of $(-8, 8)$ (or just 8 for one-sided), which is essentially no truncation.

The function `ldBounds` generates an object of class "ldBounds" which contains data about the boundaries calculated and exit probabilities, among other things. A nice way to extract information, as with most `R` object classes, is by using the `summary` and `print` methods for the "ldBounds" class. This can be done by setting `digit` to the desired number of digits.

To illustrate the use of `ldBounds` we present some examples from Reboussin, *et. al.* [1]. Using 5 equally spaced interim analyses with two-sided O'Brien-Fleming type boundaries and $\alpha = 0.05$, we have:

```
time <- seq(0.2,1,length=5)
obf.bd <- ldBounds(time)
summary(obf.bd)

##
```

```
## Lan-DeMets bounds for a given spending function
##
## n =  5
## Overall alpha:  0.05
##
## Type: Two-Sided Symmetric Bounds
## Lower alpha:  0.025
## Upper alpha:  0.025
## Spending function:  O'Brien-Fleming
##
## Boundaries:
##    Time    Lower   Upper    Exit pr.   Diff. pr.  Nominal Alpha
## 1   0.2  -4.8769  4.8769  1.0777e-06  1.0777e-06     1.0777e-06
## 2   0.4  -3.3569  3.3569  7.8830e-04  7.8723e-04     7.8808e-04
## 3   0.6  -2.6803  2.6803  7.6161e-03  6.8278e-03     7.3565e-03
## 4   0.8  -2.2898  2.2898  2.4424e-02  1.6807e-02     2.2034e-02
## 5   1.0  -2.0310  2.0310  5.0000e-02  2.5576e-02     4.2255e-02
```

The `plot` method for class "ldBounds" can be used to check the plot of the boundaries calculated. (See Figure 2).

If we want to calculate the bounds for the one-sided case, we have:

```
   obf.bd2 <- ldBounds(5,sides=1)
   summary(obf.bd)

##
## Lan-DeMets bounds for a given spending function
##
## n =  5
## Overall alpha:  0.05
```

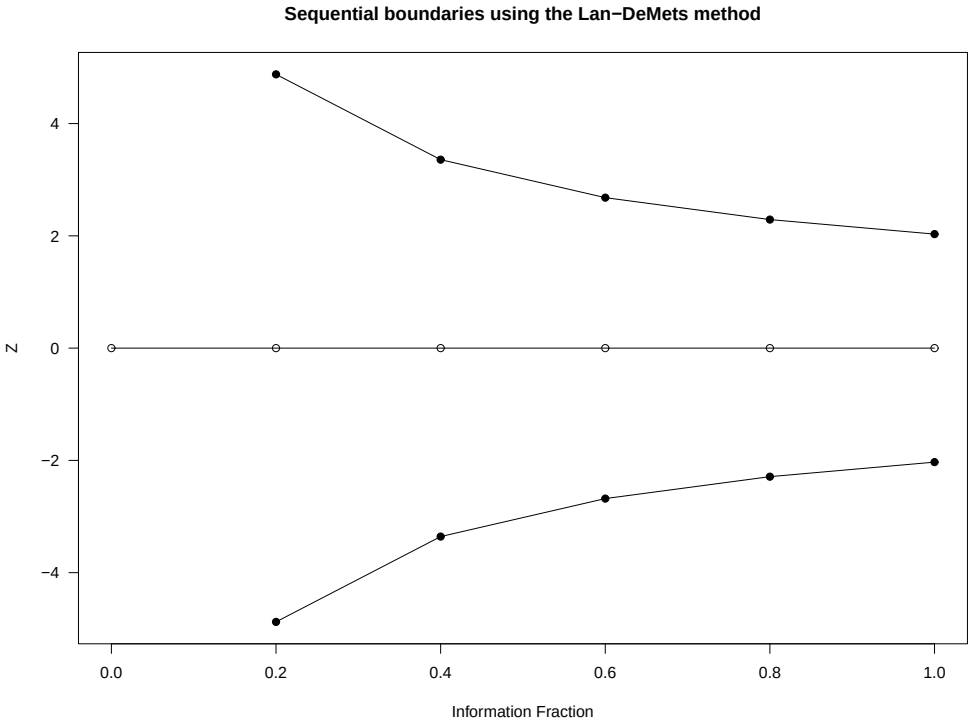

Figure 2: *Boundaries.* Two-sided plot for `plot(obf.bd)`.

```
##
## Type: Two-Sided Symmetric Bounds
## Lower alpha:  0.025
## Upper alpha:  0.025
## Spending function:  O'Brien-Fleming
##
## Boundaries:
##    Time    Lower   Upper    Exit pr.   Diff. pr.  Nominal Alpha
## 1   0.2  -4.8769  4.8769  1.0777e-06  1.0777e-06     1.0777e-06
## 2   0.4  -3.3569  3.3569  7.8830e-04  7.8723e-04     7.8808e-04
## 3   0.6  -2.6803  2.6803  7.6161e-03  6.8278e-03     7.3565e-03
## 4   0.8  -2.2898  2.2898  2.4424e-02  1.6807e-02     2.2034e-02
## 5   1.0  -2.0310  2.0310  5.0000e-02  2.5576e-02     4.2255e-02
```

Here the boundaries are shorter than in the two-sided case, which is of course a consequence of specifying the same `alpha` in both cases. We can also get the boundaries plot again (See Figure 3).

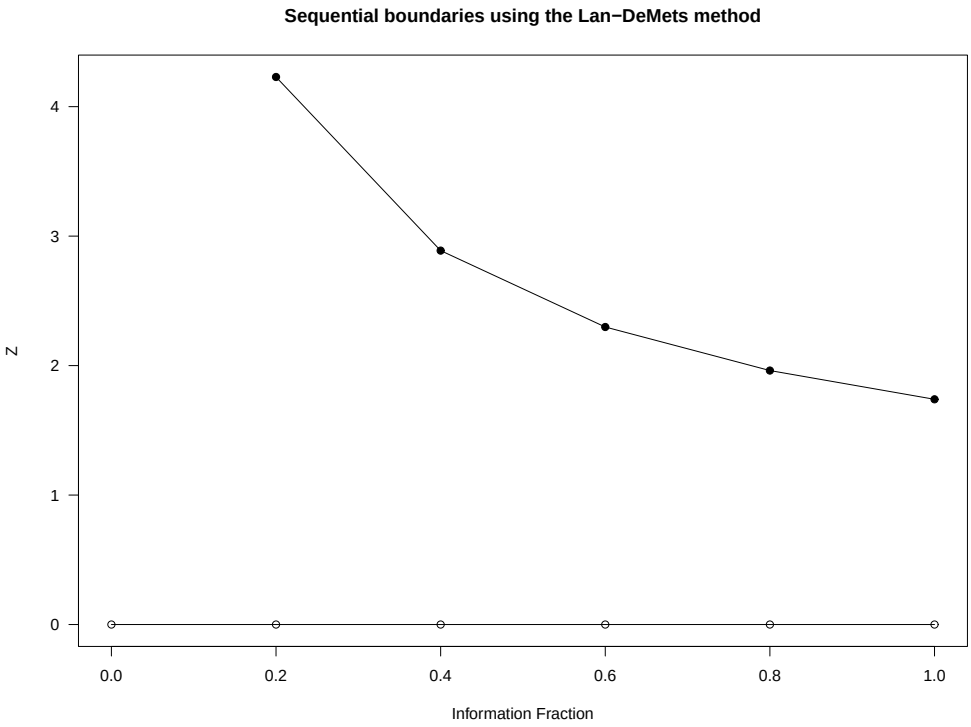

Figure 3: *Boundaries II.* One-sided plot for `plot(obf.bd)`.

5.2 The `commonbounds` Function

The `commonbounds` function is very similar to the `ldBounds` function. Both create boundaries as an object of class "ldBounds". However, `commonbounds` calculates boudaries of O'Brien-Fleming and Pocock types, without using a spending-function approach. For Haybittle-Peto type boundaries, see function `lastbound`. Specification is:

```
commonbounds(looks, t = (1:looks)/looks, t2 = t,
             iuse = "OF", alpha = 0.05, sides = 2)
```

Here `looks` is a number of analysis times (equally spaced); alternatively, `t` specifies the analysis times (but a warning is given if these are not equally spaced); `t2` is teh second time scale; `iuse` is either O'Brien-Fleming ("OF") or Pocock ("PK"); `alpha` and `sides` are as used in `ldBounds`.

5.3 The `lastbound` Function

This function also returns an object of class "ldBounds". The interim analysis times and boundaries are provided, and a final boundary is calculated and returned, along with the earlier boundaries that were provided. Specification is:

```
lastbound(t, t2, alpha = 0.05, sides = 2
          za = NULL, zb)
```

Here `t`, `t2`, `alpha`, and `sides` are the same arguments as for `ldBounds`. The arguments `za` and `zb` are vectors of lower and upper boundaries, respectively, excluding the final analysis time. The following example calculates a Haybittle-Peto boundary with 3 looks and early boundaries at 3 (z-value):

```
hpb <- lastbound(3,zb=c(3,3))
summary(hpb)
```

```
##
## Group sequential boundaries
##
## n =  3
## Overall alpha:  0.05
##
## Type: Two-Sided Symmetric Bounds
## Lower alpha:  0.025
## Upper alpha:  0.025
```

```
## Boundary type (non-alpha-spending):  User defined to obtain final boundary value.
##
## Boundaries:
##       Time    Lower   Upper   Exit pr.  Diff. pr.  Nominal Alpha
## 1  0.33333  -3.0000  3.0000  0.0026998  0.0026998      0.0026998
## 2  0.66667  -3.0000  3.0000  0.0049232  0.0022234      0.0026998
## 3  1.00000  -1.9751  1.9751  0.0499998  0.0450766      0.0482603
```

5.4 The `condpower` Function

The function `condpower` determines conditional power, given interim results and hypothesized treatment effect. Specification is:

```
condpower(z.crit, z.val, accr, outcome.type,
          par.c, par.t, N, sigma = NULL)
```

Here `z.crit` is the critical value on the z-value scale; `z.val` is the current test statistic (positive means treatment/experimental arm is estimated to have a more favorable outcome); `accr` is the amount of information accrued (number of subjects for binary and continuous outcomes, number of events for survival); `outcome.type` is binary ("bin"), continuous ("mean"), or survival ("surv"); `par.c` and `par.t` are hypothesized parameter values in control and experimental groups, respectively (unless `par.t` is not specified, in which case `par.c` is the hypothesized treatment effect; `N` is the total target sample size (or number of events); and `sigma` is the assumed standard deviation (when outcome is continuous). The following example calculates conditional power for a binary outcome with assumed rates of poor outcome of 25% and 15% in the control and experimental groups, respectively. The total planned sample size is 900, with interim analysis at 300. The final critical value is 2.0 (perhaps accounting for group sequential monitoring. The interim z-statistic is 0.067 (experimental slightly better than control).

```
   condpower(2.0,0.067,300,"bin",0.25,0.15,900)

## [1] 0.7452924
```

5.5 The `ldPower` Function

The `ldPower` function was designed to calculate different things, depending on the arguments specified. It calculates drift (effect) if the power and the bounds are given; it can calculate the confidence interval or p-value at the end of the trial if the last value of the standardized test statistic is provided. Finally, it is able to return the power and other probabilities given drift for specified boundaries. The function has the following parameters:

```
ldPower(t, za = NULL, zb = NULL, t2 = t, pow = NULL,
        drift = NULL, conf = NULL, method = NULL,
        pvaltime = NULL, zval = zb[length(zb)])
```

Here `t` is either a vector of analysis times or an "ldBounds" obect; `za` is the vector of lower boundaries and symmetric to `zb` by default; `zb` is the vector of upper boundaries; `t2` is the second time scale, `pow` is the desired power (in case of wanting this parameter and drift is not specified); `drift` is the true drift (i.e. treatment effect when t=1); `conf` is the confidence level when a confidence interval for drift is wanted; `method` is the type of p-value and `zval` is the final observed Z statistic (i.e. when trial is stopped), it is required when a confidence interval or p-value is requested.

Similarly to `ldBounds`, the function `ldPower` generates an object of class "ldPower" which contains data about the boundaries, exit probabilities, and the corresponding calculations (depending on what was requested). As in `ldBounds`, the user can extract the information that it contains by using the `summmary` and `print` methods.

We now show some applications of `ldPower`. First we want to calculate the probabilities associated with a set of bondaries given drift.

```r
time <- c(0.13,0.4,0.69,0.9,0.98,1)
upper <- c(5.3666,3.7102,2.9728,2.5365,2.2154,1.9668)
bound.pr <- ldPower(time,zb=upper,drift=3.242)
summary(bound.pr)
```

```
##
## Lan-DeMets method for group sequential boundaries
##
## n =  6
##
## Boundaries:
##    Time    Lower   Upper
## 1  0.13  -5.3666  5.3666
## 2  0.40  -3.7102  3.7102
## 3  0.69  -2.9728  2.9728
## 4  0.90  -2.5365  2.5365
## 5  0.98  -2.2154  2.2154
## 6  1.00  -1.9668  1.9668
##
## Power :  0.8996411
##
## Drift:  3.242
##
##    Time  Lower probs  Upper probs    Exit pr.  Cum exit pr.
## 1  0.13   0.0000e+00   1.3483e-05  1.3483e-05    1.3483e-05
## 2  0.40   0.0000e+00   4.8468e-02  4.8468e-02    4.8481e-02
## 3  0.69   0.0000e+00   3.4279e-01  3.4279e-01    3.9127e-01
## 4  0.90   0.0000e+00   3.1827e-01  3.1827e-01    7.0954e-01
## 5  0.98   2.2251e-08   1.3325e-01  1.3325e-01    8.4279e-01
```

```
## 6  1.00   6.4894e-08   5.6851e-02  5.6851e-02    8.9964e-01
```

Another common task is to calculate a confidence interval at the end of the trial. In order to do this, we need, in addition to the boundaries, the last value of the standardized test statistic `zval`, equal to 2.82 in this example.

```
   time <- c(0.2292,0.3333,0.4375,0.5833,0.7083,0.8333)
   upper <- c(2.53,2.61,2.57,2.47,2.43,2.38)
   bound.ci <- ldPower(time,zb=upper,conf=0.95,zval=2.82)
   summary(bound.ci)

##
## Lan-DeMets method for group sequential boundaries
##
## n =  6
##
## Boundaries:
##      Time  Lower  Upper
## 1  0.2292  -2.53   2.53
## 2  0.3333  -2.61   2.61
## 3  0.4375  -2.57   2.57
## 4  0.5833  -2.47   2.47
## 5  0.7083  -2.43   2.43
## 6  0.8333  -2.38   2.38
##
## Confidence interval at the end of the trial:
##
## Confidence level:  0.95
## Last Z value:  2.82
##  95 % confidence interval: ( 0.1716782 , 4.504662 )
```

The following example shows the computing of boundaries using `ldBounds`, and using these in `ldPower`. In this example the power is provided and we want to get the drift.

```
   time <- seq(0.2,1,length=5)
   power.fam <- ldBounds(time,iuse=3)
   bound.dr <- ldPower(power.fam,pow=0.9)
   summary(bound.dr)
```

```
##
## Lan-DeMets method for group sequential boundaries
##
## n =  5
##
## Boundaries:
##    Time    Lower   Upper
## 1   0.2  -2.5758  2.5758
## 2   0.4  -2.4919  2.4919
## 3   0.6  -2.4108  2.4108
## 4   0.8  -2.3391  2.3391
## 5   1.0  -2.2754  2.2754
##
## Power :  0.9
##
## Drift:  3.455209
##
##    Time  Lower probs  Upper probs  Exit pr.  Cum exit pr.
## 1   0.2   1.8858e-05     0.151361  0.151380       0.15138
## 2   0.4   1.2004e-06     0.251277  0.251278       0.40266
## 3   0.6   1.2723e-07     0.233648  0.233648       0.63631
```

```
## 4   0.8   1.7278e-08     0.164962  0.164962       0.80127
## 5   1.0   0.0000e+00     0.098731  0.098731       0.90000
```

6 Discussion

The `R` package `ldbounds` is able to perform the same computations as the `FORTRAN` program. It also allows for more flexibility for specifying alpha spending functions. Many improvements have been made in the current version, including better ways to specify arguments, more efficient argument defaults, and better graphics.

References

[1] Reboussin DM, DeMets DL, Kim K, Lan KKG. Computations for group sequential boundaries using the Lan-DeMets spending function method. *Controlled Clinical Trials*. 2000; 21:190-207.

[2] Lan KKG, DeMets DL. Discrete sequential boundaries for clinical trials. *Biometrika*. 1983; 70:659-63.

[3] DeMets DL, Lan G. "The alpha spending function approach to interim data analyses" in *Recent Advances in Clinical Trial Design and Analysis*, ed. PF Thall. Kluwer Academic Publishers, Boston; 1995.

[4] Armitage P, McPherson CK, Rowe BC. Repeated significance tests on accumulating data. *Journal of the Royal Statistical Society*. 1969; 132:235-44

[5] R Development Core Team (2005). 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.