LogrankA
An R Function to Calculate the Logrank Test for Aggregated Survival Data

Jonas Richter-Dumke
<jrd.r.project@gmail.com>

July 15, 2013

Abstract. The `LogrankA` function is an implementation of the logrank test for aggregated survival data in the statistical programming language R. It is shown that `LogrankA` can have performance advantages over alternative logrank implementations if used with aggregated data.

1. Introduction

The logrank test is a major tool for survival analysis, working hand in hand with both the Kaplan-Meier estimator and the Cox proportional hazards model. It is used to compare the survival of two or more groups across time, testing for significant differences of the group specific survival functions. The potency of a new medicament in the treatment of cancer, the lifetime of different engine designs, the effect of social class on the age of transition to parenthood – a logrank test can be applied in all of these cases to help finding an answer. Being such a prominently featured tool in survival analysis, the logrank test is implemented in the R language – a widespread programming language for statistical computing. The implementation of the logrank test already included in the default installation of R is the `survdiff` function of the package `survival` (cp. Therneau and Lumley 2012, p. 78). The `survival` package is included in every R installation but has to be loaded with `library()` before it can be used. An alternative is the `logrank` function of the external package `Hmisc` (cp. Harrell 2012, 260ff). The latter function is limited to two groups and both of these implementations lack the feature of performing a logrank test on *aggregated* data – a feature which becomes handy when working on large datasets with a relatively small number of realized unique variable-value combinations.

The objective of this paper is to present `LogrankA`, a new implementation of the logrank test in R, capable of performing a logrank test with aggregated data across unlimited groups. The source code of the implementation will be explained step-by-step

using survival data from AIDS patients. Finally it will be shown how the new `LogrankA` function performs against its alternatives.

2. Theoretical Background

2.1. Logrank Test [1] [2]

The logrank test is a test for difference of the survival in two or more groups. The basic idea behind the logrank test is that if there is no difference in the survival of the different groups, than the distribution of deaths across all groups equals the distribution of the risk-population across all groups. This logic is then translated to each time interval in the survival data. To perform a logrank test, group specific data for each time interval on the number of events, the number of censorings and the risk-population are needed. Given that data, group specific expected events e_{kt} and group specific observed events d_{kt} can be calculated for any given time interval. The observed events are the sum of group specific events/deaths for each time interval. The expected events are the share of the risk-population of group k, r_{kt}, in the total risk population r_t multiplied by the sum of group independent events/deaths d_t per time interval.

$$e_{kt} = d_t \cdot \frac{r_{kt}}{r_t} \tag{1}$$

Let O_k be the sum of all observed events over time and E_k the sum of all expected events over time for each group k.

$$O_k = \sum_t d_{kt} \tag{2}$$

$$E_k = \sum_t e_{kt} \tag{3}$$

The logrank test statistic LR is then calculated as the sum of group specific quotients.

$$\mathrm{LR} = \sum_k \frac{(O_k - E_k)^2}{E_k} \tag{4}$$

The LR value follows the χ^2 distribution with $k-1$ degrees of freedom. The p-value of the χ^2 distribution at LR is the significance of a difference in the survival between the groups.

Other calculations of the logrank test are possible and return slightly different results. The logrank variant chosen here is comparably easy to compute for more than two groups and was chosen for this reason.

[1] cp. Peto et al. 1977, p. 9

[2] For an excellent (german) explanation of the logrank test see Ziegler, Lange, and Bender 2007.

2.2. Aggregated Data

Aggregated data is data where a single row in the dataset does not correspond to a single observation but to multiple observations. A special column indicates the number of observations sharing the value combination given in the data row. For each realized combination of values across all variables a row is generated in the aggregated dataset.

An artificial example would be data on age and sex for the US population. When using individual data the dataset would have one row for each observation. Assuming a complete population sample that would be 309 million rows of data with two columns (age and sex) for the year 2010. The number of rows can be vastly reduced using aggregated data. At most the aggregated dataset will include 222 rows of data (one row for each combination of sex with the assumed 111 age groups) and three column (age, sex and count). For each realized combination of age and sex the count of occurrences is given. No information is lost but the size of the dataset has been vastly reduced.

3. Example Data

The package `MASS` (cp. Ripley et al. 2012) contains various datasets covering a broad thematic range and is included in every R installation. The `LogrankA` function will be tested and explained by the use of the `Aids2` dataset. This dataset contains survival data of AIDS patients in Australia. It contains 2843 cases and variables for state, sex, time of diagnosis, time of end of observation, survival status at end of observation, transmission category and age at diagnosis (cp. ibid., 7f). The head of this dataset is shown in Table 1.

state	sex	diag	death	status	T.categ	age
NSW	M	10905	11081	D	hs	35
NSW	M	11029	11096	D	hs	53
NSW	M	9551	9983	D	hs	42
NSW	M	9577	9654	D	haem	44
NSW	M	10015	10290	D	hs	39
NSW	M	9971	10344	D	hs	36

Table 1: Top rows of dataframe `Aids2`

To use this data in a survival analysis the survival time must be computed. To reduce the number of distinct values and therefore allowing an efficient aggregation of the data the survival time is converted into weeks. The status variable is recoded to 0 (censoring) or 1 (event/death). A new variable `agegr` for age-groups is computed. Additionally some random missing values are introduced to the data to check the behaviour of the logrank functions in case of incomplete data. Unneeded columns are dropped. Finally, a binary variable `agebin` is added to the dataframe. It separates individuals into the two age-groups "below age 40" and "above age 40". This way the `LogrankA` function can be checked against the `logrank` function of the `Hmisc` package, which at the time of the writing of this paper only supports the testing of two groups.

```r
#install.packages("MASS")
library(MASS)

# import australian aids data
aids2.ind <- Aids2

# write some random NAs in the data to check NA behaviour of LogrankA
set.seed(1987)
rand.i <- sample(1:length(aids2.ind$status), size = 400)
aids2.ind$status[rand.i[1:200]] <- NA
aids2.ind$death[rand.i[201:400]] <- NA
# recode  status to 0/1
aids2.ind$status <- as.numeric(aids2.ind$status) - 1
# generate survival time in weeks
stime.days <- aids2.ind$death - aids2.ind$diag
aids2.ind$stime <- round(stime.days / 7, 0)
# generate age groups
aids2.ind$agegr <- cut(aids2.ind$age, c(0, 20, 40, 60, 100),
                       right = FALSE)
# generate binary variable separating below and above age 40
# useful for 2-group logrank test
aids2.ind$agebin[as.numeric(aids2.ind$agegr) <= 2] <- 0
aids2.ind$agebin[as.numeric(aids2.ind$agegr) > 2] <- 1
aids2.ind$agebin <- factor(aids2.ind$agebin, levels = c(0, 1),
                           labels = c("below age 40", "above age 40"))
# drop unneeded columns
aids2.ind <- aids2.ind[ , c(5, 8, 9, 10)]
```

Table 2 shows the first rows of the data used to compare the `LogrankA` function against its alternatives. This is individual-level data. For the aggregated data the same base-dataset is used. This way it can be shown that the `LogrankA` function computes the same values with aggregated data as the other logrank implementations do with the corresponding individual data.

```r
# transform to aggregated data (NAs treated as group)
stime.f <- factor(aids2.ind$stime, exclude = NULL)   # factorization needed in
status.f <- factor(aids2.ind$status, exclude = NULL) # order for aggregate()
                                                     # to output NAs
aids2.aggr <- aggregate(aids2.ind$stime,
                        by = list(status.f, stime.f, aids2.ind$agegr,
                                  aids2.ind$agebin),
                        FUN = length)
aids2.aggr[ , 1] <- as.numeric(as.character(aids2.aggr[ , 1])) # back to numeric
aids2.aggr[ , 2] <- as.numeric(as.character(aids2.aggr[ , 2]))
colnames(aids2.aggr) <- c("status", "stime", "agegr", # restore lost colnames
                          "agebin", "n")
```

status	stime	agegr	agebin
1.00	25.00	[20,40)	below age 40
1.00	10.00	[40,60)	above age 40
1.00	62.00	[40,60)	above age 40
1.00	11.00	[40,60)	above age 40
1.00	39.00	[20,40)	below age 40
1.00	53.00	[20,40)	below age 40
1.00	56.00	[20,40)	below age 40
1.00	147.00	[20,40)	below age 40
1.00	70.00	[20,40)	below age 40
1.00	62.00	[20,40)	below age 40
1.00	2.00	[40,60)	above age 40
1.00	44.00	[20,40)	below age 40

Table 2: Top rows of dataframe `aids2.ind`

status	stime	agegr	agebin	n
1.00	0.00	[0,20)	below age 40	5
1.00	2.00	[0,20)	below age 40	4
1.00	8.00	[0,20)	below age 40	1
1.00	13.00	[0,20)	below age 40	1
0.00	28.00	[0,20)	below age 40	1
0.00	31.00	[0,20)	below age 40	1
1.00	33.00	[0,20)	below age 40	1
1.00	37.00	[0,20)	below age 40	2
1.00	46.00	[0,20)	below age 40	1
.	48.00	[0,20)	below age 40	1
1.00	52.00	[0,20)	below age 40	1
1.00	55.00	[0,20)	below age 40	1

Table 3: Top rows of dataframe `aids2.aggr`

The column `n` as seen in the aggregation of the `Aids2` dataset in Table 3 is the weight. It shows the number of observations with the same value combinations. The aggregation of the data reduced the number of rows from 2843 to 849.

4. LogrankA Function

In this chapter the source code of the `LogrankA` function will be discussed. For the complete source code see appendix A.

4.1. Input

The input section ensures that all the necessary data is given (in the right format) before starting the logrank calculation. The header of `LogrankA` shows the input needed for the logrank calculation:

```
LogrankA <- function(surv, group, weight) {
```

An object of type `survival` is expected as first input argument. This object is generated with the function `Surv` of the package `survival` and holds information about the survival time and censoring status of each observation. Data on the group affiliation is stored in the variable `group`. The argument `weight` is optional. It specifies the number of occurrences for each value combination in an aggregated dataset and is not to be confused with time-weights common in different flavours of the logrank test.

```
  # are all necessary arguments specified?
  if (missing(surv)) {
    stop("No survival argument specified")
  }
```

```
if (missing(group)) {
  stop("No group argument specified")
}
# is the survival argument a survival object?
if (is.Surv(surv) == FALSE) {
  stop("Survival argument not a survival object")
}
```

The input is checked for errors prior to the logrank calculation so that helpful error messages instead of false values are returned to the user. At first it is checked if any mandatory arguments (`survival` and `group`) are missing from the input. The `survival` argument must be an object of type `survival`, otherwise an error is returned.

```
# extract time and status from survival object
time <- surv[ , 1]
status <- surv[ , 2]
```

For the logrank calculation the survival object will not be used directly. Instead vectors of time and status/censoring values are extracted from it.

```
# if weight is not specified, default to a dummy weight vector of 1s
if (missing(weight)) {
  weight <- rep(1, length(time))
  # otherwise, check if weight is numeric...
} else if (!is.numeric(weight)) {
  stop("Non-numeric weight value found")
  # ...and non-negative...
} else if (any(na.omit(weight) < 0)) {
  stop("Negative weight value found")
  # ...and integer (check for existence fractional part)
} else if (any(na.omit(weight) %% 1 > 0)) {
  stop("Non-integer weight value found")
}
```

If the `weight` argument is not specified it is assumed that the input data is not aggregated. Therefore a dummy vector of ones is defined as the weight variable. This way the same code can be used for aggregated and individual data. If specified, the weight vector must only contain non-negative integers.

```
# is the censoring of type "right"?
if (attr(surv, "type") != "right") {
  stop("Censorings are not of type right")
}
# are all arguments of equal length?
if (any(c(length(time), length(status),
          length(group), length(weight)) != length(time))) {
  stop("Arguments differ in length")
}
```

```
# is more than one group specified?
if (length(unique(group)) == 1) {
  stop("Only a single group specified")
}
```

The logrank test requires right-censored data whereas the `Surv` function is capable of handling other types of censoring. The survival object contains an attribute with information about the censoring which must be of type "right". The arguments `time`, `status`, `group` and `weight` must have the same number of rows. A logrank test on a single group is prohibited.

```
# drop observations containing missing values
survdata <- data.frame(time, status, group, weight)
if (any(is.na(survdata))) {
  na.dropped <- na.omit(survdata);
  time <- na.dropped$time;
  status <- na.dropped$status;
  group <- na.dropped$group;
  weight <- na.dropped$weight;
  n.dropped <- sum(survdata$weight) - sum(na.dropped$weight)
} else {
  n.dropped <- 0
}
```

If all the error tests have been passed the input variables are checked for missing values. If a missing value is detected in any of the input variables the whole observation (the corresponding row across all input variables) is dropped from the data. The internal variable `n.dropped` tracks the number of dropped observations.

4.2. Group Independent Statistics

The group independent statistics are the variables needed for the computation of the logrank test statistic which ignore the group affiliation. These are the risk population at the start of the process time r_{t_0}, the risk population at the beginning of each time interval r_i and the number of events in each time interval d_i. Other variables are calculated as intermediate steps. The heavy use of the `table()` function is grounded in the interval nature of the logrank test. It is important to know what happened in a given time interval. The table function makes it possible to aggregate over the timepoints so it is known what happened between the consecutive timepoints t_1 and t_2.

```
# risk population at start of process time
r.t0 <- sum(weight)
```

The weights of the data correspond to the number of observations for any given value combination. The risk population at the start of the process time is therefore calculated as the sum of weights. r_{t_0} can also be understood as the total number of observations.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	0	0	1	0	1	0	2	1	2	0	0	1	1	1	0
2	0	0	1	0	0	0	1	1	2	0	1	1	0	1	1
3	1	1	1	1	1	0	0	0	0	1	0	0	0	0	1
4	1	0	1	0	0	1	0	0	0	0	1	0	0	1	0
5	2	1	0	1	1	0	0	0	0	2	0	0	3	0	0
6	0	0	1	0	0	0	2	0	0	1	0	0	1	2	1

Table 4: Subset of `table(weight, time)`

```
# number of observations (events and censored cases) for each time interval i
w <- sort(unique(weight))
  # each count of weight gets multiplicated with corresponding
  # value for weight. result is the number of occurrences
obs.i <- colSums(table(weight, time) * w)
```

In order to know the risk population for every time interval r_i the total number of observations for every time interval `obs.i` are calculated. The expression `table(weight, time)` produces a table of weight against time with time on the x-axis (see table 4 for an extract of the full table). To get the number of observations (events plus censored cases) for each time interval each value for `weight` has to be multiplied with the corresponding count for `weight`. The resulting values have to be summed up across each column to get the needed variable `obs.i`.

```
# risk population at the beginning of each time interval i
RevCumsumRev <- function (x) {rev(cumsum(rev(x)))}
r.i <- RevCumsumRev(obs.i)
```

The risk population at the beginning of each time interval r_i is calculated backwards over the number of observations by time interval: If a censoring or an event happened in interval i, the object which has terminated was part of the dataset up to and including interval i (assuming no late entry occurs).

```
# number of group-independent events in each time interval i
  # censored cases excluded from the total number of observations by
  # multiplying the status (0 for censored cases) with the weight
sw <- status * weight
d <- sort(unique(sw))
d.i <- colSums(table(sw, time) * d)
```

The calculation of the number of events per time interval is similar to the calculation of the number of observations for each time interval. The difference is in the exclusion of censored cases. These cases are excluded by multiplying the status (which is 0 for censored cases and 1 for events) with the weight.

4.3. Group Specific Statistics

The group specific statistics include the risk population of each group at the beginning of each time interval r_{ki}, the expected events of each group in each time interval e_{ki}, the total observed events of each group O_k and the total expected events of each group E_k.

```
# number of observations for each time interval i by group k
obs.ki <- colSums(table(weight, time, group) * w)
# risk population of group k at the beginning of each time interval i
r.ki <- apply(obs.ki, 2, RevCumsumRev)
```

The calculation of r_{ki} is similar to the calculation of the group independent risk population by time r_i explained above. Instead of a two dimensional table for time and weight however a table with three dimensions is computed. The third dimension is the group affiliation.

```
# expected events of group k in each time interval i
e.ki <- d.i * r.ki / r.i
```

The number of expected events of each group is the result of a very straightforward calculation already explained in chapter 2.1.

```
# total observed events group k
O.k <- colSums(table(sw, group) * d)
```

To calculate the total observed events for each group O_k a table of the number of events for each row in the original data against the group variable is computed. Each row of the table is then multiplied by the corresponding values of `d`. The sum over the columns of this (fairly artificial) table results in the total observed events for each group.

```
# total expected events group k
E.k <- colSums(e.ki)
```

The last calculation before it all comes together in the logrank test statistic is very easy. The total expected events for each group E_k are the sum of the time specific expected events for each group e_{ki}.

4.4. Logrank Test Statistic

The calculation of the LR value is much easier than the calculation of the statistics leading to it.

```
# actual logrank test statistic
LR.k <- (O.k - E.k)^2 / E.k
LR <- sum(LR.k)
```

A quotient between the squared difference of observed and expected events and the expected events is calculated. The sum of this quotient across all groups is the logrank test statistic.

```
  # degrees of freedom for chi^2 test of LR
  df <- length(O.k) - 1
  # chi^2 test of LR
  p.chi2 <- pchisq(LR, df = df, lower.tail = FALSE)
```

The last step in order to calculate a significance of the survival differences between the groups is to test the LR value in the χ^2 distribution. The cumulative probability distribution function for χ^2 is used with the option `lower.tail` set to `FALSE`. This option sets the calculated p-value to $P(X > x)$. The number of degrees of freedom for the distribution is given by the number of groups minus one.

4.5. Output

All results are aggregated in the output section of the `LogrankA` function.

```
# output
  r.kt0 <- t(t(r.ki[1, ]))
  logrank.parameter <- cbind(r.kt0, t(t(O.k)), t(t(E.k)), LR.k)
  colnames(logrank.parameter) <- c("N", "Obs. events", "Exp. events",
                                   "(O-E)^2/E")
  cat("Valid observations: ", r.t0, "\n",
      "Dropped observations: ", n.dropped, "\n\n",
      "Logrank test statistic = ", LR, " on ", df,
      " degrees of freedom,\np = ", p.chi2, "\n\n", sep = "")
  list(p.chi2 = p.chi2, df = df, LR = LR,
       lr.parameter = logrank.parameter)
}
```

The p-value, the degrees of freedom, the logrank test statistic, and a list with the group specific number of observations, observed events and expected events are printed and can be used as input for other functions. In addition a short text summary is printed to the console. The `LogrankA` function ends here.

5. Comparison of Logrank Implementations

The results and the performance of the `LogrankA` function are tested against the `survdiff` function of the package `survival` and the `logrank` function of the package `Hmisc`.

5.1. Calculation Results [3]

The computed LR values for the different logrank functions can be found in table 5. The tests show that the `LogrankA` function computes exactly the same logrank test statistic for the aggregated and individual data. It can also be shown that the output of `LogrankA` is slightly different than the output of `survdiff` and `logrank` regarding

[3] For the source code used to benchmark the functions please see appendix B.

Test	LR	elapsed
`LogrankA`, individual, 4 groups	21.46	2.46
`LogrankA`, aggregated, 4 groups	21.46	0.92
`survdiff`, individual, 4 groups	21.86	1.08
`LogrankA`, individual, 2 groups	9.80	2.41
`LogrankA`, aggregated, 2 groups	9.80	0.91
`survdiff`, individual, 2 groups	9.94	1.08
`logrank`, individual, 2 groups	9.94	0.17

Elapsed time in seconds (100 repetitions each).

Table 5: Calculation- and speed results for different logrank tests

the LR value. This difference is based on a different computation of the logrank test statistic. `survdiff` and `logrank` use a formula for LR with variances in the quotients. The `LogrankA` function uses the logrank "flavor" as described by Richard and Julian Peto et al.[4] in 1977 (cp. Peto et al. 1977, p. 9). The differences are small and simply mirror the fact that there is more than one way to compute a logrank statistic. The number of observed and expected events for each group is identical between `LogrankA` and `survdiff`. The reported number of dropped and valid cases is also the same for all functions.

It can be concluded that the `LogrankA` function calculates the correct results on the testdata `Aids2`.

5.2. Calculation Speed [5]

In table 5 the calculation speeds for 100 repetitions of each logrank test function are shown. Regarding individual data the `LogrankA` function performs more than two times slower than its alternatives. `survdiff` performs about five times slower than the simple yet highly efficient `logrank` function.

The `LogrankA` function performs faster on aggregated data than it does on individual data. In the test scenario the processing of the aggregated data is more than two times faster than the logrank calculation on the corresponding individual data. The performance gain is due to the difference in the number of rows between the aggregated and the individual data. Compared with the individual `aids2.ind` dataset the aggregated dataset `aids2.aggr` reduced the number of rows by 1994.

On the aggregated dataset the `LogrankA` function performs slightly faster than the `survdiff` function on the corresponding non-aggregated data.

[4] The designation "logrank" can be attributed to them.

[5] System used for benchmark: `R version 2.15.2 (2012-10-26), x86_64-pc-linux-gnu, Intel(C) Core(TM) i7 CPU X 940 @ 2.13GHz` × `8, 7.7 GiB Memory.`

The `LogrankA` function has the potential to outperform the alternative logrank implementations on highly compressed aggregated data. For individual data the `LogrankA` function is slower than its alternatives.

6. Outlook

Future versions of `LogrankA` might include features seen in `survdiff` as for example the option to take a formula object as input or the option to include time-weights. The interface can also be improved. The user should be able to access the complete table of risk-population, expected and observed events for each group over time. A method for `xtable` – a function that produces latex table code out of R objects – would also be nice.

As for now the `LogrankA` function allows a logrank test with an unlimited number of groups on aggregated data. `LogrankA` is able to handle missing data. The merits of aggregated data are mirrored by `LogrankA` – the higher the compression of the data after the aggregation the faster `LogrankA` performs on the data.

References

Harrell, Frank E. (Mar. 2012). *Package 'Hmisc'*. URL: http://cran.r-project.org/web/packages/Hmisc/Hmisc.pdf.

Peto, R. et al. (1977). "Design and analysis of randomized clinical trials requiring prolonged observation of each patient. II. analysis and examples". In: *British journal of cancer* 35.1, pp. 1–39.

Ripley, Brian et al. (Aug. 2012). *Package 'MASS'*. URL: http://cran.r-project.org/web/packages/MASS/MASS.pdf.

Therneau, Terry and Thomas Lumley (Apr. 2012). *Package 'survival'*. URL: http://cran.r-project.org/web/packages/survival/survival.pdf.

Ziegler, A., S. Lange, and R. Bender (2007). "Überlebenszeitanalyse: Der Log-Rang-Test". In: *Deutsche Medizinische Wochenschrift* 132, pp. 39–41.

A. Source code of LogrankA

```r
LogrankA <- function(surv, group, weight) {
  # are all necessary arguments specified?
  if (missing(surv)) {
    stop("No survival argument specified")
  }
  if (missing(group)) {
    stop("No group argument specified")
  }
  # is the survival argument a survival object?
  if (is.Surv(surv) == FALSE) {
    stop("Survival argument not a survival object")
  }

  # extract time and status from survival object
  time <- surv[ , 1]
  status <- surv[ , 2]

  # if weight is not specified, default to a dummy weight vector of 1s
  if (missing(weight)) {
    weight <- rep(1, length(time))
    # otherwise, check if weight is numeric...
  } else if (!is.numeric(weight)) {
    stop("Non-numeric weight value found")
    # ...and non-negative...
  } else if (any(na.omit(weight) < 0)) {
    stop("Negative weight value found")
    # ...and integer (check for existence fractional part)
  } else if (any(na.omit(weight) %% 1 > 0)) {
    stop("Non-integer weight value found")
  }
  # is the censoring of type "right"?
  if (attr(surv, "type") != "right") {
    stop("Censorings are not of type right")
  }
  # are all arguments of equal length?
  if (any(c(length(time), length(status),
            length(group), length(weight)) != length(time))) {
    stop("Arguments differ in length")
  }
  # is more than one group specified?
  if (length(unique(group)) == 1) {
    stop("Only a single group specified")
  }

  # drop observations containing missing values
  survdata <- data.frame(time, status, group, weight)
```

```r
if (any(is.na(survdata))) {
  na.dropped <- na.omit(survdata);
  time <- na.dropped$time;
  status <- na.dropped$status;
  group <- na.dropped$group;
  weight <- na.dropped$weight;
  n.dropped <- sum(survdata$weight) - sum(na.dropped$weight)
} else {
  n.dropped <- 0
}

# risk population at start of process time
r.t0 <- sum(weight)
# number of observations (events and censored cases) for each time interval i
w <- sort(unique(weight))
  # each count of weight gets multiplicated with corresponding
  # value for weight. result is the number of occurrences
obs.i <- colSums(table(weight, time) * w)
# risk population at the beginning of each time interval i
RevCumsumRev <- function (x) {rev(cumsum(rev(x)))}
r.i <- RevCumsumRev(obs.i)
# number of group-independent events in each time interval i
  # censored cases excluded from the total number of observations by
  # multiplying the status (0 for censored cases) with the weight
sw <- status * weight
d <- sort(unique(sw))
d.i <- colSums(table(sw, time) * d)

# number of observations for each time interval i by group k
obs.ki <- colSums(table(weight, time, group) * w)
# risk population of group k at the beginning of each time interval i
r.ki <- apply(obs.ki, 2, RevCumsumRev)
# expected events of group k in each time interval i
e.ki <- d.i * r.ki / r.i
# total observed events group k
O.k <- colSums(table(sw, group) * d)
# total expected events group k
E.k <- colSums(e.ki)

# actual logrank test statistic
LR.k <- (O.k - E.k)^2 / E.k
LR <- sum(LR.k)
# degrees of freedom for chi^2 test of LR
df <- length(O.k) - 1
# chi^2 test of LR
p.chi2 <- pchisq(LR, df = df, lower.tail = FALSE)

# output
r.kt0 <- t(t(r.ki[1, ]))
```

```
  logrank.parameter <- cbind(r.kt0, t(t(O.k)), t(t(E.k)), LR.k)
  colnames(logrank.parameter) <- c("N", "Obs. events", "Exp. events",
                                   "(O-E)^2/E")
  cat("Valid observations: ", r.t0, "\n",
      "Dropped observations: ", n.dropped, "\n\n",
      "Logrank test statistic = ", LR, " on ", df,
      " degrees of freedom,\np = ", p.chi2, "\n\n", sep = "")
  list(p.chi2 = p.chi2, df = df, LR = LR,
       lr.parameter = logrank.parameter)
}
```

B. Source code of benchmark

```
#install.packages(c("survival", "rbenchmark", "Hmisc"))
library(survival)
library(rbenchmark)
library(Hmisc)

# survival objects for aggregated and individual data
surv.ind <- Surv(aids2.ind$stime, aids2.ind$status)
surv.aggr <- Surv(aids2.aggr$stime, aids2.aggr$status)

# LR-chisq LogrankA individual vs. LogrankA aggregated vs. survdiff individual,
# 4 groups
loga.ind.4gr <- LogrankA(surv = surv.ind, group = aids2.ind$agegr)
loga.aggr.4gr <- LogrankA(surv = surv.aggr, group = aids2.aggr$agegr,
                          weight = aids2.aggr$n)
sdiff.4gr <- survdiff(surv.ind ~ aids2.ind$agegr, rho = 0)

# LR-chisq LogrankA individual vs. LogrankA aggregated vs. survdiff individual
# vs. logrank individual, 2 groups
loga.ind.2gr <- LogrankA(surv = surv.ind, group = aids2.ind$agebin)
loga.aggr.2gr <- LogrankA(surv = surv.aggr, group = aids2.aggr$agebin,
                          weight = aids2.aggr$n)
sdiff.2gr <- survdiff(surv.ind ~ aids2.ind$agebin, rho = 0)
logrank.2gr <- pchisq(logrank(S = surv.ind, group = aids2.ind$agebin), df = 1,
                      lower.tail = FALSE)

# speed LogrankA individual vs. LogrankA aggregated vs. survdiff individual,
# 4 groups
speed.results <- benchmark(
  LogrankA(surv = surv.ind, group = aids2.ind$agegr),
  LogrankA(surv = surv.aggr, group = aids2.aggr$agegr, weight = aids2.aggr$n),
  survdiff(surv.ind ~ aids2.ind$agegr, rho = 0),
  LogrankA(surv = surv.ind, group = aids2.ind$agebin),
  LogrankA(surv = surv.aggr, group = aids2.aggr$agebin, weight = aids2.aggr$n),
  survdiff(surv.ind ~ aids2.ind$agebin, rho = 0),
  logrank(S = surv.ind, group = aids2.ind$agebin),
  columns = c("elapsed"), order = NULL
)

results <- data.frame(Test = c("\\texttt{LogrankA}, individual, 4 groups",
                               "\\texttt{LogrankA}, aggregated, 4 groups",
                               "\\texttt{survdiff}, individual, 4 groups",
                               "\\texttt{LogrankA}, individual, 2 groups",
                               "\\texttt{LogrankA}, aggregated, 2 groups",
                               "\\texttt{survdiff}, individual, 2 groups",
                               "\\texttt{logrank}, individual, 2 groups"),
                      LR = c(loga.ind.4gr$LR,
```

```
                              loga.aggr.4gr$LR,
                              sdiff.4gr$chisq,
                              loga.ind.2gr$LR,
                              loga.aggr.2gr$LR,
                              sdiff.2gr$chisq,
                              qchisq(logrank.2gr[1], df = 1,
                                     lower.tail = FALSE)),
                       Time = speed.results
                       )

# output results
results
```