Comparing the performance of computation methods
an example with package Countr and the fertility data

Tarak Kharrat\(^1\) and Georgi N. Boshnakov\(^2\)

\(^1\)Salford Business School, University of Salford, UK.
\(^2\)School of Mathematics, University of Manchester, UK.

Abstract

This short document compares the performance of the different algorithms implemented in Countr to fit renewal-count models. The computation described here is based on the fertility data shipped with the package and the weibull-count model which allows using series based methods on top of the other convolution methods. More details about the different computation methods can be found in Baker and Kharrat (2017).

This vignette is part of package Countr (see Kharrat et al., 2019).

1 Prerequisites

We will do the analysis of the data with package Countr, so we load it:

```
library(Countr)
```

Package rebenchmark (Kusnierczyk, 2012) will also be used here to facilitate performance computation

```
library(rbenchmark)
```

2 Comparing performance of different methods

The data used here is the fertility data shipped with the package and described in length in Winkelmann (1995).

The execution time depends obviously on the machine used but the relative order should remain the same regardless of the characteristic of your machine. The `benchmark()` routine from the rbenchmark package Kusnierczyk (2012) will be used to compare the different computation methods discussed in Baker and Kharrat (2017). We selected the parameters for every method such as we achieve an error or at least $10^{-8}$. The code below reproduces the results reported in Baker and Kharrat (2017, Table 2). As a benchmark, we use an adaptation of the original code gently provided by Blake McShane (McShane et al., 2008). The McShane’s code is implemented in a separate file and is not shown here. We repeat the iterations 1000 times.

```
source("mcShaneCode.R")
data(fertility)
```

```
## config: choose parameters (selection process not shown here)
children <- fertility$children
```
shape <- 1.116
scale <- rep(2.635, length(children))
rep <- 1000
nstepsConv <- c(132, 24, 132, 24, 132, 36)
ntermsSeries <- c(20, 17)
conv_series_acc <- 1e-7

## performance model
perf <- benchmark(direct0 =
  dWeibullCount_loglik(children, shape, scale, "conv_direct",
                       1, TRUE, nstepsConv[1],
                       conv_extrap = FALSE),
  direct1 =
  dWeibullCount_loglik(children, shape, scale, "conv_direct",
                       1, TRUE, nstepsConv[2],
                       conv_extrap = TRUE),
  naive0 =
  dWeibullCount_loglik(children, shape, scale, "conv_naive",
                       1, TRUE, nstepsConv[3],
                       conv_extrap = FALSE),
  naive1 = dWeibullCount_loglik(children, shape, scale, "conv_naive",
                                1, TRUE, nstepsConv[4],
                                conv_extrap = TRUE),
  dePril0 = dWeibullCount_loglik(children, shape, scale, "conv_dePril",
                                1, TRUE, nstepsConv[5],
                                conv_extrap = FALSE),
  dePril1 = dWeibullCount_loglik(children, shape, scale, "conv_dePril",
                                1, TRUE, nstepsConv[6],
                                conv_extrap = TRUE),
  series_mat =
  dWeibullCount_loglik(children, shape, scale, "series_mat", 1, TRUE,
                       series_terms = ntermsSeries[1]),
  series_acc =
  dWeibullCount_loglik(children, shape, scale, "series_acc", 1, TRUE,
                       series_terms = ntermsSeries[2],
                       series_acc_eps = conv_series_acc),
  mcShane = dWeibullCount_McShane(scale, shape, children, jmax = 150),
  replications = rep, order = "relative",
  columns = c("test", "replications", "relative", "elapsed")
)
print(perf)
As can been seen, all the methods outperform the McShane’s original code with the series methods almost 20 times faster and the extrapolated convolution methods roughly 6 times faster. As noted in the package documentation, the series methods are less robust to large values of count and may fail for some application. We therefore encourage the users to use the extrapolated convolution method (the default in Countr) as much as possible.

We conclude this document by saving the work space to avoid re-running the computation in future exportation of the document:

```r
save.image()
```

References


