expint: Exponential integral and incomplete gamma function

Vincent Goulet
Université Laval

1 Introduction

The exponential integral

$$E_1(x) = \int_x^\infty \frac{e^{-t}}{t}\,dt, \quad x \in \mathbb{R}$$

and the incomplete gamma function

$$\Gamma(a, x) = \int_x^\infty t^{a-1}e^{-t}\,dt, \quad x > 0, \quad a \in \mathbb{R}$$

are two closely related functions that arise in various fields of mathematics.

expint is a small package provides facilities to compute the exponential integral and the incomplete gamma function. Furthermore, and perhaps most conveniently for R package developers, the package also gives easy access to the underlying C workhorses through an API. The C routines are derived from the GNU Scientific Library (GSL; Galassi et al., 2009).

The package **expint** started its life in version 2.0-0 of **actuar** (Dutang et al., 2008), where I extended the range of admissible values in the computation of limited expected value functions. This required an incomplete gamma function that accepts negative values of argument a, as explained at the beginning of Appendix A of Klugman et al. (2012).

2 Exponential integral

Abramowitz and Stegun (1972, Section 5.1) first define the exponential integral as

$$E_1(x) = \int_x^\infty \frac{e^{-t}}{t}\,dt. \tag{1}$$

An alternative definition (to be understood in terms of the Cauchy principal value due to the singularity of the integrand at zero) is

$$\mathrm{Ei}(x) = -\int_{-x}^{\infty} \frac{e^{-t}}{t}\,dt = \int_{-\infty}^{x} \frac{e^{t}}{t}\,dt, \quad x > 0.$$

The above two definitions are related as follows:

$$E_1(-x) = -\mathrm{Ei}(x), \quad x > 0. \tag{2}$$

The exponential integral can also generalized to

$$E_n(x) = \int_1^{\infty} \frac{e^{-xt}}{t^n}\,dt, \quad n = 0, 1, 2, \ldots, \quad x > 0,$$

where n is then the *order* of the integral. The latter expression is closely related to the incomplete gamma function (section 3) as follows:

$$E_n(x) = x^{n-1}\Gamma(1-n, x). \tag{3}$$

One should note that the first argument of function Γ is negative for $n > 1$.

The following recurrence relation holds between exponential integrals of successive orders:

$$E_{n+1}(x) = \frac{1}{n}[e^{-x} - xE_n(x)]. \tag{4}$$

Finally, $E_n(x)$ has the following asymptotic expansion:

$$E_n(x) \asymp \frac{e^{-x}}{x}\left(1 - \frac{n}{x} + \frac{n(n+1)}{x^2} - \frac{n(n+1)(n+2)}{x^3} + \ldots\right). \tag{5}$$

3 Incomplete gamma function

From a probability theory perspective, the incomplete gamma function is usually defined as

$$P(a, x) = \frac{1}{\Gamma(a)}\int_0^{x} t^{a-1}e^{-t}\,dt, \quad x > 0, \quad a > 0.$$

Function `pgamma` already implements this function in R (just note the differing order of the arguments).

Now, the definition of the incomplete gamma function of interest for this package is rather the following (Abramowitz and Stegun, 1972, Section 6.5):

$$\Gamma(a, x) = \int_x^{\infty} t^{a-1}e^{-t}\,dt, \quad x > 0, \quad a \in \mathbb{R}. \tag{6}$$

Note that a can be negative with this definition. Of course, for $a > 0$ one has

$$\Gamma(a, x) = \Gamma(a)[1 - P(a, x)]. \tag{7}$$

Integration by parts of the integral in (6) yields the recursive relation

$$\Gamma(a, x) = -\frac{x^a e^{-x}}{a} + \frac{1}{a}\Gamma(a + 1, x). \tag{8}$$

When $a < 0$, this relation can be used repeatedly k times until $a + k$ is a positive number. The right hand side can then be evaluated with (7). If $a = 0, -1, -2, \ldots$, this calculation requires the value of

$$G(0, x) = \int_x^\infty \frac{e^{-t}}{t}\, dt = E_1(x),$$

the exponential integral defined in (1).

4 R interfaces

expint provides one main and four auxiliary R functions to compute the exponential integral, and one function to compute the incomplete gamma function. Their signatures are the following:

```
expint(x, order = 1L, scale = FALSE)
expint_E1(x, scale = FALSE)
expint_E2(x, scale = FALSE)
expint_En(x, order, scale = FALSE)
expint_Ei(x, scale = FALSE)
gammainc(a, x)
```

Let us first go over function `gammainc` since there is less to discuss. The function takes in argument two vectors or real numbers (non-negative for argument `x`) and returns the value of $\Gamma(a, x)$. The function is vectorized in arguments `a` and `x`, so it works similar to, say, `pgamma`.

We now turn to the `expint` family of functions. The function `expint` is a unified interface to compute exponential integrals $E_n(x)$ of any (non-negative) order, with default the most common case $E_1(x)$. The function is vectorized in arguments `x` and `order`. In other words, one can compute the exponential integral of a different order for each value of x.

```
> expint(c(1.275, 10, 12.3), order = 1:3)
[1] 1.408099e-01 3.830240e-06 3.009983e-07
```

The argument `order` should be a vector of integers. Non-integer values are silently coerced to integers using truncation towards zero.

When the argument `scale` is `TRUE`, the result is scaled by e^x.

The functions `expint_E1`, `expint_E2` and `expint_En` are simpler, slightly faster ways to directly compute exponential integrals $E_1(x)$, $E_2(x)$ and $E_n(x)$, the latter for a *single* order n (the first value of `order` if `order` is a vector).

```
> expint_E1(1.275)
[1] 0.1408099
> expint_E2(10)
[1] 3.83024e-06
> expint_En(12.3, order = 3L)
[1] 3.009983e-07
```

Finally, the function `expint_Ei` is provided as a convenience to compute $\mathrm{Ei}(x)$ using (2).

```
> expint_Ei(5)
[1] 40.18528
> -expint_E1(-5)     # same
[1] 40.18528
```

5 Accessing the C routines

The actual workhorses behind the R functions of section 4 are C routines with the following prototypes:

```
double expint_E1(double x, int scale);
double expint_E2(double x, int scale);
double expint_En(double x, int order, int scale);
double gamma_inc(double a, double x);
```

expint makes these routines available to other packages through declarations in the header file 'include/expintAPI.h' in the package installation directory. If you want to use a routine — say `expint_E1` — in your package **pkg**, proceed as follows:

1. Add the package **expint** to the `Imports` and `LinkingTo` directives of the 'DESCRIPTION' file of **pkg**;

2. Add an entry '`import(expint)`' in the 'NAMESPACE' file of **pkg**;

3. Define the routine with a call to `R_GetCCallable` in the initialization routine `R_init_pkg` of **pkg** (R Core Team, 2025, Section 5.4). For the current example, the file 'src/init.c' of **pkg** would contain the following code:

```
void R_init_pkg(DllInfo *dll)
{
    R_registerRoutines(/* native routine registration */);

    pkg_expint_E1 = (double(*)(double,int,int))
                    R_GetCCallable("expint", "expint_E1");
}
```

4. Define a native routine interface, say `pkg_expint_E1` to avoid any name clash, in 'src/init.c' that will call `expint_E1`:

```
double(*pkg_expint_E1)(double,int);
```

5. Declare the routine in a header file of **pkg** with the keyword `extern` to expose the interface to all routines of the package. In our example, 'src/pkg.h' would contain:

```
extern double(*pkg_expint_E1)(double,int);
```

6. Include the package header file 'pkg.h' in any C file making use of the routine `pkg_expint_E1`.

To help developers get started, **expint** ships with a complete test package implementing the above; see the 'example_API' sub-directory in the installation directory. This test package uses the `.External` R to C interface and, as a bonus, shows how to vectorize an R function on the C side (the code for this being mostly derived from base R).

There are various ways to define a package API. The approach described above was derived from the package **zoo** (Zeileis and Grothendieck, 2005). The package **xts** (Ryan and Ulrich, 2024) — and probably a few others on CRAN — draws from **Matrix** (Bates and Maechler, 2025) to propose a somewhat simpler approach where the API exposes routines that can be used directly in a package. However, the provided header file can be included only once in a package,

otherwise one gets 'duplicate symbols' errors at link time. This constraint does not show in the example provided with **xts** or in packages **RcppXts** (Eddelbuettel, 2022) and **TTR** (Ulrich, 2023) that link to it (the only two at the time of writing). A way around the issue is to define a native routine calling the routines exposed in the API. In this scenario, tests I conducted proved the approach I retained to be up to 10% faster most of the time.

6 Implementation details

As already stated, the C routines mentioned in section 5 are derived from code in the GNU Scientific Library (Galassi et al., 2009).

For exponential integrals, the main routine `expint_E1` computes $E_1(x)$ using Chebyshev expansions (Gil et al., 2007, chapter 3). Routine `expint_E2` computes $E_2(x)$ using `expint_E1` with relation (4) for $x < 100$, and using the asymptotic expression (5) otherwise. Routine `expint_En` simply relies on `gamma_inc` to compute $E_n(x)$ for $n > 2$ through relation (3).

For the sake of providing routines that better fit within the R ecosystem and coding style, I made the following changes to the original GSL code:

1. routines compute a single value and return their result by value;
2. accordingly, calculation of the approximation error is dropped in all routines;
3. `gamma_inc` computes $\Gamma(a, x)$ for $a > 0$ with (7) using the routines `gammafn` and `pgamma` of the R API, rather than using the GSL routines, as the example below illustrates;

```
> options(digits = 20)
> gammainc(1.2, 3)

[1] 0.06542142809100923162

> gamma(1.2) * pgamma(3, 1.2, 1, lower = FALSE)

[1] 0.06542142809100923162
```

4. `gamma_inc` computes $\Gamma(a, x)$ for $-0.5 < a < 0$ using the recursion (8) instead of a series expansion as in the GSL routines, thereby relying on the accuracy of `pgamma` near $a = 0.5$ (fixes issue #2; see Appendix A for additional details).

7 Alternative packages

The Comprehensive R Archive Network[1] (CRAN) contains a number of packages with features overlapping **expint**. I review the similarities and differences here.

The closest package in functionality is **gsl** (Hankin, 2006). This package is an R wrapper for the special functions and quasi random number generators of the GNU Scientific Library. As such, it provides access to basically the same C code as used in **expint**. Apart from the changes to the GSL code mentioned in section 6, the main difference between the two packages is that installation from source of **gsl** requires that the GSL be installed on one's system, whereas **expint** is a regular, free standing R package.

VGAM (Yee, 2015) is a large, high quality package that provides functions to compute the exponential integral $\mathrm{Ei}(x)$ for real values, as well as $e^{-x}\,\mathrm{Ei}(x)$ and $E_1(x)$ and their derivatives (up to the third derivative). Functions `expint`, `expexpint` and `expint.E1` are wrappers to the Netlib[2] FORTRAN subroutines in file `ei.f`. **VGAM** does not provide an API to its C routines.

The package **pracma** (Borchers, 2016) provides a large number of functions from numerical analysis, linear algebra, numerical optimization, differential equations and special functions. Its versions of `expint`, `expint_E1`, `expint_Ei` and `gammainc` are entirely written in R with perhaps less focus on numerical accuracy than the GSL and Netlib implementations. The functions are not vectorized, and the incomplete gamma function is supported for $a \geq -1$ only.

The package **frmqa** had a function `gamma_inc_err` that computed the incomplete gamma function using the incomplete Laplace integral, but it was only valid for $a = j + 0.5$, $j = 0, 1, 2, \ldots$. (The package was removed from CRAN in 2022.)

Package **zipfR** (Evert and Baroni, 2007) introduces a set of functions to compute various quantities related to the gamma and incomplete gamma functions, but these are essentially wrappers around the base R functions `gamma` and `pgamma` with no new functionalities.

8 Examples

Let us tabulate the values of $E_n(x)$ for $x = 1.275, 10, 12.3$ and $n = 1, 2, \ldots, 10$ as found in examples 4–6 of Abramowitz and Stegun (1972, section 5.3).

[1] https://cran.r-project.org
[2] https://www.netlib.org

```
> x <- c(1.275, 10, 12.3)
> n <- 1:10
> structure(t(outer(x, n, expint)),
+           dimnames = list(paste("n =", n),
+                           paste("x =", x)))
         x = 1.275       x = 10     x = 12.3
n = 1  0.14080993 4.156969e-06 3.439534e-07
n = 2  0.09989831 3.830240e-06 3.211177e-07
n = 3  0.07603031 3.548763e-06 3.009983e-07
n = 4  0.06083077 3.304101e-06 2.831550e-07
n = 5  0.05046793 3.089729e-06 2.672346e-07
n = 6  0.04301687 2.900528e-06 2.529517e-07
n = 7  0.03743074 2.732441e-06 2.400730e-07
n = 8  0.03310097 2.582217e-06 2.284066e-07
n = 9  0.02965340 2.447221e-06 2.177930e-07
n = 10 0.02684699 2.325303e-06 2.080990e-07
```

We may also tabulate the values of $\Gamma(a, x)$ for $a = -1.5, -1, -0.5, 1$ and $x = 1, 2, \ldots, 10$.

```
> a <- c(-1.5, -1, -0.5, 1)
> x <- 1:10
> structure(t(outer(a, x, gammainc)),
+           dimnames = list(paste("x =", x),
+                           paste("a =", a)))
            a = -1.5       a = -1     a = -0.5        a = 1
x = 1  1.264878e-01 1.484955e-01 1.781477e-01 3.678794e-01
x = 2  1.183299e-02 1.876713e-02 3.009876e-02 1.353353e-01
x = 3  1.870260e-03 3.547308e-03 6.776136e-03 4.978707e-02
x = 4  3.706365e-04 7.995573e-04 1.733500e-03 1.831564e-02
x = 5  8.350921e-05 1.992938e-04 4.773965e-04 6.737947e-03
x = 6  2.045031e-05 5.304291e-05 1.379823e-04 2.478752e-03
x = 7  5.310564e-06 1.478712e-05 4.127115e-05 9.118820e-04
x = 8  1.440569e-06 4.267206e-06 1.266464e-05 3.354626e-04
x = 9  4.042025e-07 1.264846e-06 3.964430e-06 1.234098e-04
x = 10 1.165117e-07 3.830240e-07 1.260904e-06 4.539993e-05
```

9 Acknowledgments

I built on the source code of R and many of the packages cited in this manual to create **expint**, so the R Core Team and the package developers deserve credit. I also extend my thanks to Dirk Eddelbuettel who was of great help in putting together the package API, through both his posts in online forums and private communication. Joshua Ulrich provided a fix to the API infrastructure to avoid duplicated symbols that was implemented in version 0.1-6 of the package.

A Additional details on the computation of the incomplete gamma function

Issue #2 raised by Geoffrey Poole highlights that the function `gammainc` in versions of **expint** prior to 0.2-0 returned inconsistent results for small negative values of a and a small value of x. Figure 1 illustrates the problem by comparing the behaviour of $\Gamma(a, 10^{-5})$ around $a = -0.5$ between `gammainc` of **expint** (again, prior to 0.2-0) and `incgam` of **pracma**.

The function `gamma_inc` for the package **gsl** shows the same defect[3], indicating that the problem must lie in the GSL code. Indeed, the GSL routine for the incomplete gamma function treats specially the case $-0.5 < a < 0$, and reverts to the recursion (8) only for $a \leq -0.5$.

For $-0.5 < a < 0$, the GSL computes the incomplete gamma function using the relation

$$\Gamma(a, x) = \Gamma(a)Q(a, x),$$

with $Q(a, x)$ defined as follows:

$$\begin{aligned} Q(a, x) &= 1 - P(a, x) \\ &= 1 - \frac{\gamma(a, x)}{\Gamma(a)}, \end{aligned}$$

with

$$\gamma(a, x) = P(a, x)\Gamma(a) = \int_0^x t^{a-1} e^{-t}\, dt.$$

The GSL routine `gamma_inc_Q_series` carries the computation of $Q(a, x)$ using a series expansion. The code is not obvious and requires a fair share of reverse engineering. Hence, I document my findings here.

[3] The solid line in Figure 1 is actually traced using `gamma_inc`.

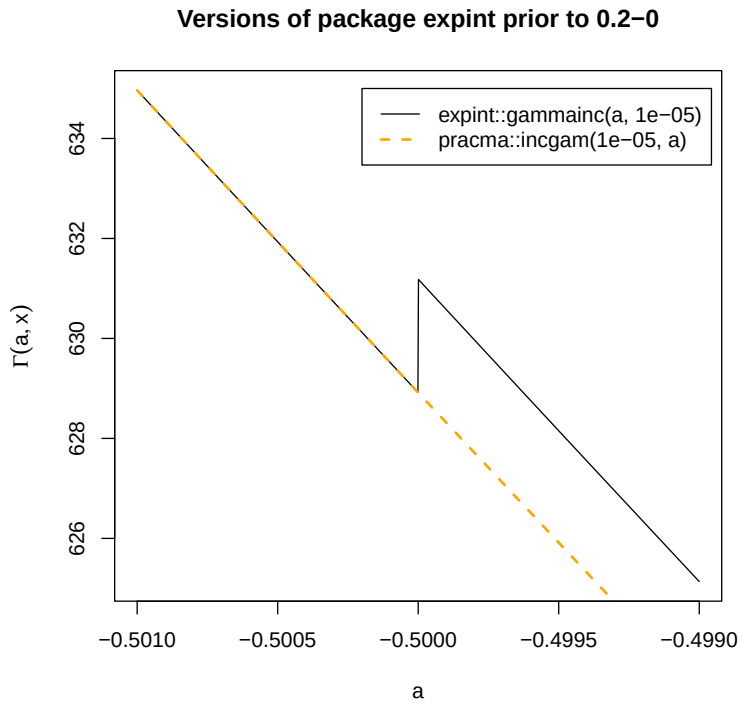

Figure 1: Incomplete gamma function for small negative values of a and a small value of x

First, a series expansion for $\gamma(a,x)$ is (Abramowitz and Stegun, 1972, section 6.5.33):

$$\gamma(a,x) = \sum_{n=0}^{\infty} \frac{(-1)^n}{a+n} \frac{x^{a+n}}{n!}. \tag{9}$$

We can rewrite this expansion as

$$\begin{aligned} \gamma(a,x) &= \frac{x^a}{a} - \sum_{n=1}^{\infty} \frac{(-1)^{n-1}}{a+n} \frac{x^{a+n}}{n!} \\ &= \frac{x^a}{a} - \frac{x^{a+1}}{a+1} \sum_{n=1}^{\infty} (-1)^{n-1} \left(\frac{a+1}{a+n}\right) \frac{x^{n-1}}{n!}. \end{aligned} \tag{10}$$

Second, using the fact that $\Gamma(a+1) = a\Gamma(a)$, we can rewrite $Q(a,x)$ as

follows:

$$\begin{aligned}
Q(a,x) &= 1 - \frac{x^a}{\Gamma(a+1)} + \frac{x^a}{\Gamma(a+1)} - \frac{\gamma(a,x)}{\Gamma(a)} \\
&= 1 - \frac{x^a}{\Gamma(a+1)} + \frac{1}{\Gamma(a)}\left(\frac{x^a}{a} - \gamma(a,x)\right) \\
&= 1 - \frac{x^a}{\Gamma(a+1)} + \frac{x^a}{\Gamma(a+1)}\frac{\Gamma(a+1)}{\Gamma(a)}\frac{x}{x^{a+1}}\left(\frac{x^a}{a} - \gamma(a,x)\right) \\
&= 1 - \frac{x^a}{\Gamma(a+1)} + \frac{x^a}{\Gamma(a+1)}\left(\frac{a}{a+1}\right)x\left(\frac{a+1}{x^{a+1}}\right)\left(\frac{x^a}{a} - \gamma(a,x)\right).
\end{aligned}$$

If we define $\phi(a) = 1 - x^a/\Gamma(a+1)$ and we simplify the last term above using (10), we finally obtain:

$$Q(a,x) = \phi(a) + (1-\phi(a))\left(\frac{a}{a+1}\right)x\sum_{n=1}^{\infty}(-1)^{n-1}\left(\frac{a+1}{a+n}\right)\frac{x^{n-1}}{n!}. \tag{11}$$

This is the expression used to compute $Q(a,x)$.

The only remaining element is the computation of $\phi(a)$. For this, the routine `gamma_inc_Q_series` uses the product of the Taylor series expansion of x^a around $a = 0$,

$$x^a = \sum_{n=0}^{\infty}\frac{(a\ln x)^n}{n!},$$

and the following Taylor series expansion of the reciprocal gamma function (Weisstein, 2026):

$$\frac{1}{\Gamma(a+1)} = 1 + \gamma a + \left(\frac{\gamma^2}{2} - \frac{\pi^2}{12}\right)a^2 + \left(\frac{\gamma^3}{6} - \frac{\gamma\pi^2}{12} + \frac{\zeta(3)}{3}\right)a^3 + \cdots,$$

where γ is the Euler constant, and ζ is the Riemann zeta function (Abramowitz and Stegun, 1972, chapters 6 and 23). Putting all these pieces together, we obtain:

$$\begin{aligned}
\phi(a) &= 1 - \frac{x^a}{\Gamma(a+1)} \\
&= -(\ln x + \gamma)a + \left\{\frac{\pi}{12} - (\ln x + \gamma)^2\right\}a^2 \\
&\quad + \left\{(\ln x + \gamma)\left(\frac{\pi}{12} - \frac{(\ln x + \gamma)^2}{6}\right) - \frac{\zeta(3)}{3}\right\}a^3 + \cdots.
\end{aligned}$$

One will recognize above the coefficients `c1`, `c2` and `c3` of `term1` in the routine `gamma_inc_Q_series`. It is not clear where the numerical values in the other seven coefficients come from. Here endeth reverse engineering.

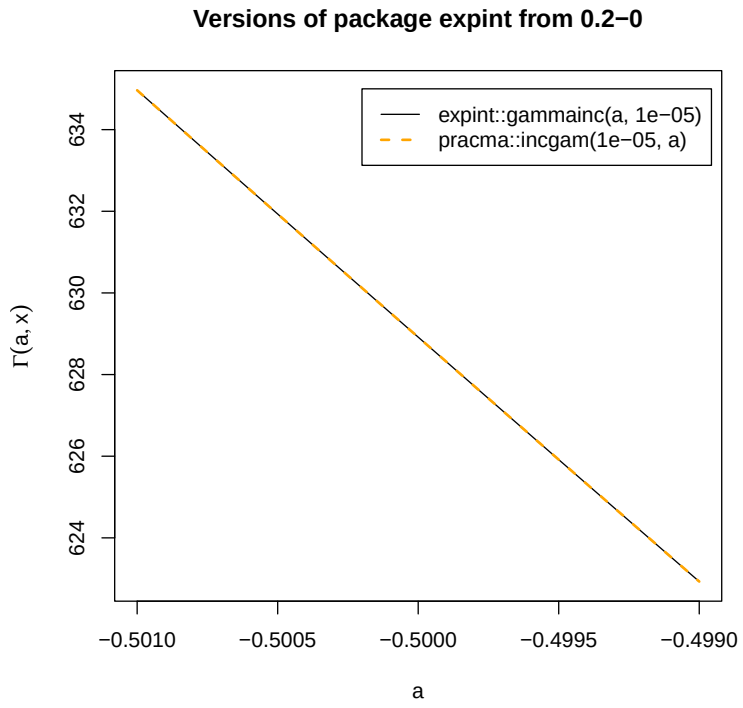

Figure 2: Fixed incomplete gamma function

The fix introduced in **expint** 0.2-0 follows the same strategy as **pracma**: just use the recursion (8) also for $-0.5 < a < 0$, and rely on the accuracy of `pgamma` for small values of a to yield the correct result. Figure 2 shows that the results from **expint** and **pracma** now match.

One last implementation detail: the original GSL code uses a loop to compute (8) as many times as necessary for $a < -0.5$. Simply extending usage of the loop to $a < 0$ does not work due to rounding errors in computations involving values of a in $(-0.5, 0)$. Therefore, **expint** keeps as a special case the single application of the recursive relation for a in the latter range.

References

M. Abramowitz and I. A. Stegun. *Handbook of Mathematical Functions*. Dover, 1972. URL https://personal.math.ubc.ca/~cbm/aands/.

D. Bates and M. Maechler. *Matrix: Sparse and Dense Matrix Classes and Methods*, 2025. URL https://cran.r-project.org/package=Matrix. R package version 1.7-4.

H. W. Borchers. ***pracma**: Practical Numerical Math Functions*, 2016. URL https://cran.r-project.org/package=pracma. R package version 2.4.6.

C. Dutang, V. Goulet, and M. Pigeon. **actuar**: An R package for actuarial science. *Journal of Statistical Software*, 25(7), 2008. URL https://www.jstatsoft.org/v25/i07.

D. Eddelbuettel. *RcppXts: Interface the xts API via Rcpp*, 2022. URL https://cran.r-project.org/package=RcppXts. R package version 0.0.6.

S. Evert and M. Baroni. **zipfR**: Word frequency distributions in R. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics, Posters and Demonstrations Sessions*, pages 29–32, Prague, Czech Republic, 2007. URL https://cran.r-project.org/package=zipfR. R package version 0.6-70.

M. Galassi, J. Davies, J. Theiler, B. Gough, G. Jungman, Alken P., M. Booth, F. Rossi, and R. Ulerich. *GNU Scientific Library Reference Manual*, third edition, 2009. URL https://www.gnu.org/software/gsl/.

A. Gil, J. Segura, and N. M. Temme. *Numerical Methods for Special Functions*. Society for Industrial and Applied Mathematics, 2007. ISBN 978-0-89871634-4. URL https://dx.doi.org/10.1137/1.9780898717822.

R. K. S. Hankin. Special functions in R: introducing the **gsl** package. *R News*, 6, October 2006.

S. A. Klugman, H. H. Panjer, and G. Willmot. *Loss Models: From Data to Decisions*. Wiley, New York, 4 edition, 2012. ISBN 978-1-11831532-3.

R Core Team. *Writing R Extensions*, 2025. URL https://cran.r-project.org/doc/manuals/R-exts.html. Manual for R version 4.5.2.

J. A. Ryan and J. M. Ulrich. *xts: eXtensible Time Series*, 2024. URL https://cran.r-project.org/package=xts. R package version 0.14.1.

J. Ulrich. *TTR: Technical Trading Rules*, 2023. URL https://cran.r-project.org/package=TTR. R package version 0.24.4.

Eric W. Weisstein. Gamma function. From MathWorld – A Wolfram Resource, 2026. URL https://mathworld.wolfram.com/GammaFunction.html.

T. W. Yee. *Vector Generalized Linear and Additive Models: With an Implementation in R*. Springer, 2015. ISBN 978-1-49392818-7. URL https://cran.r-project.org/package=VGAM.

A. Zeileis and G. Grothendieck. zoo: S3 infrastructure for regular and irregular time series. *Journal of Statistical Software*, 14(6):1–27, 2005. doi: 10.18637/jss.v014.i06.