Accurately Computing $\log(1 - \exp(-|a|))$ Assessed by the Rmpfr package

Martin Mächler
ETH Zurich
April, Oct. 2012 (LaTeX'ed July 14, 2024)

Abstract

In this note, we explain how $f(a) = \log(1 - e^{-a}) = \log(1 - \exp(-a))$ can be computed accurately, in a simple and optimal manner, building on the two related auxiliary functions `log1p(x)` $(= \log(1 + x))$ and `expm1(x)` $(= \exp(x) - 1 = e^x - 1)$. The cutoff, a_0, in use in R since 2004, is shown to be optimal both theoretically and empirically, using **Rmpfr** high precision arithmetic. As an aside, we also show how to compute $\log(1 + e^x)$ accurately and efficiently.

Keywords: Accuracy, Cancellation Error, R, MPFR, Rmpfr.

1. Introduction: Not log() nor exp(), but log1p() and expm1()

In applied mathematics, it has been known for a very long time that direct computation of $\log(1 + x)$ suffers from severe cancellation (in "$1 + x$") whenever $|x| \ll 1$, and for that reason, we have provided `log1p(x)` in R, since R version 1.0.0 (released, Feb. 29, 2000). Similarly, `log1p()` has been provided by C math libraries and has become part of C language standards around the same time, see, for example, IEEE and Open Group (2004).

Analogously, since R 1.5.0 (April 2002), the function `expm1(x)` computes $\exp(x) - 1 = e^x - 1$ accurately also for $|x| \ll 1$, where $e^x \approx 1$ is (partially) cancelled by "-1".

In both cases, a simple solution for small $|x|$ is to use a few terms of the Taylor series, as

$$\mathrm{log1p}(x) = \log(1 + x) = x - x^2/2 + x^3/3 - + \dots, \text{ for } |x| < 1, \tag{1}$$
$$\mathrm{expm1}(x) = \exp(x) - 1 = x + x^2/2! + x^3/3! + \dots, \text{ for } |x| < 1, \tag{2}$$

and $n!$ denotes the factorial.

We have found, however, that in some situations, the use of `log1p()` and `expm1()` may not be sufficient to prevent loss of numerical accuracy. The topic of this note is to analyze the important case of computing $\log(1 - e^x) = \log(1 - \exp(x))$ for $x < 0$, computations needed in accurate computations of the beta, gamma, exponential, Weibull, t, logistic, geometric and hypergeometric distributions, and even for the logit link function in logistic regression. For the beta and gamma distributions, see, for example, DiDonato and Morris (1992)[1], and further references mentioned in R's `?pgamma` and `?pbeta` help pages. For the logistic distribution,

[1] In the Fortran source, file "708", also available as http://www.netlib.org/toms/708, the function ALNREL() computes log1p() and REXP() computes expm1().

$F_L(x) = \frac{e^x}{1+e^x}$, the inverse, aka quantile function is $q_L(p) = \text{logit}(p) := \log\frac{p}{1-p}$. If the argument p is provided on the log scale, $\tilde{p} := \log p$, hence $\tilde{p} \le 0$, we need

$$\texttt{qlogis}(\tilde{p},\ \texttt{log.p} = \texttt{TRUE}) = q_L\left(e^{\tilde{p}}\right) = \text{logit}\left(e^{\tilde{p}}\right) = \log\frac{e^{\tilde{p}}}{1-e^{\tilde{p}}} = \tilde{p} - \log\left(1-e^{\tilde{p}}\right), \qquad (3)$$

and the last term is exactly the topic of this note.

2. log1p() and expm1() for log(1 - exp(x))

Contrary to what one would expect, for computing $\log\left(1-e^x\right) = \log(1-\exp(x))$ for $x < 0$, neither

$$\log(1-\exp(x)) = \log(-\text{expm1}(x)), \quad \text{nor} \qquad (4)$$
$$\log(1-\exp(x)) = \text{log1p}(-\exp(x)), \qquad (5)$$

are uniformly sufficient for numerical evaluation. In (5), when x approaches 0, $\exp(x)$ approaches 1 and $\text{log1p}(-\exp(x))$ loses accuracy. In (4), when x is large, $\text{expm1}(x)$ approaches -1 and similarly loses accuracy. Because of this, we will propose to use a function `log1mexp(x)` which uses either `expm1` (4) or `log1p` (5), where appropriate. Already in R 1.9.0 (R Development Core Team (2004)), we have defined the macro `R_D_LExp(x)` to provide these two cases automatically[2].

To investigate the accuracy losses empirically, we make use of the R package **Rmpfr** for arbitrarily accurate numerical computation, and use the following simple functions:

```
R> library(Rmpfr)
R> t3.l1e <- function(a)
  {
      c(def   = log(1 - exp(-a)),
        expm1 = log( -expm1(-a)),
        log1p = log1p(-exp(-a)))
  }
R> ##' The relative Error of log1mexp computations:
R> relE.l1e <- function(a, precBits = 1024) {
      stopifnot(is.numeric(a), length(a) == 1, precBits > 50)
      da <- t3.l1e(a)  ## double precision
      a. <- mpfr(a, precBits=precBits)
      ## high precision *and* using the correct case:
      mMa <- if(a <= log(2)) log(-expm1(-a.)) else log1p(-exp(-a.))
      structure(as.numeric(1 - da/mMa), names = names(da))
  }
```

where the last one, `relE.l1e()` computes the relative error of three different ways to compute $\log(1-\exp(-a))$ for positive a (instead of computing $\log(1-\exp(x))$ for negative x).

```
R> a.s <- 2^seq(-55, 10, length = 256)
R> ra.s <- t(sapply(a.s, relE.l1e))
R> cbind(a.s, ra.s) # comparison of the three approaches
```

```
              a.s          def       expm1        log1p
  [1,] 2.7756e-17         -Inf -7.9755e-17         -Inf
```

[2] look for "log(1-exp(x))" in http://svn.r-project.org/R/branches/R-1-9-patches/src/nmath/dpq.h

```
  [2,] 3.3119e-17       -Inf -4.9076e-17       -Inf
  [3,] 3.9520e-17       -Inf -7.8704e-17       -Inf
  [4,] 4.7157e-17       -Inf -4.5998e-17       -Inf
  [5,] 5.6271e-17  1.8162e-02 -7.3947e-17  1.8162e-02
  [6,] 6.7145e-17  1.3504e-02 -4.4921e-17  1.3504e-02
  [7,] 8.0121e-17  8.8009e-03 -1.2945e-17  8.8009e-03
 .......
 .......
[251,] 4.2329e+02  1.0000e+00  1.0000e+00 -3.3151e-17
[252,] 5.0509e+02  1.0000e+00  1.0000e+00  2.9261e-17
[253,] 6.0270e+02  1.0000e+00  1.0000e+00  1.7377e-17
[254,] 7.1917e+02  1.0000e+00  1.0000e+00 -4.7269e-12
[255,] 8.5816e+02  1.0000e+00  1.0000e+00  1.0000e+00
[256,] 1.0240e+03  1.0000e+00  1.0000e+00  1.0000e+00
```

This is revealing: Neither method, log1p or expm1, is uniformly good enough. Note that for large a, the relative errors evaluate to `1`. This is because all three double precision methods give 0, *and* that is the best approximation in double precision (but not in higher `mpfr` precision), hence no problem at all, and we can restrict ourselves to smaller a (smaller than about 710, here).

What about really small a's? Note here that

```
R> t3.l1e(1e-20)
```

```
    def   expm1   log1p
   -Inf -46.052    -Inf
```

```
R> as.numeric(t3.l1e(mpfr(1e-20, 256)))
```

```
[1] -46.052 -46.052 -46.052
```

both the default and the `log1p` method return `-Inf`, so, indeed, the `expm1` method is absolutely needed here.

Figure 1 visualizes the relative errors[3] of the three methods. Note that the default basically gives the maximum of the two methods' errors, whereas the final `log1mexp()` function will have (approximately) minimal error of the two.

```
R> matplot(a.s, abs(ra.s), type = "l", log = "xy",
          col=cc, lty=lt, lwd=ll, xlab = "a", ylab = "", axes=FALSE)
R> legend("top", leg, col=cc, lty=lt, lwd=ll, bty="n")
R> draw.machEps <- function(alpha.f = 1/3, col = adjustcolor("black", alpha.f)) {
      abline(h = .Machine$double.eps, col=col, lty=3)
      axis(4, at=.Machine$double.eps, label=quote(epsilon[c]), las=1, col.axis=col)
  }
R> eaxis(1); eaxis(2); draw.machEps(0.4)
```

In Figure 2 below, we zoom into the region where all methods have about the same (good) accuracy. The region is the rectangle defined by the ranges of `a.` and `ra2`:

```
R> a. <- (1:400)/256
R> ra <- t(sapply(a., relE.l1e))
R> ra2 <- ra[,-1]
```

In addition to zooming in Figure 1, we want to smooth the two curves, using a method

[3] Absolute value of relative errors, $\left|(\hat f(a) - f(a))/f(a)\right| = \left|1 - \hat f(a)/f(a)\right|$, where $f(a) = \mathrm{log1mexp}(a)$ (7) is computed accurately by a 1024 bit **Rmpfr** computation

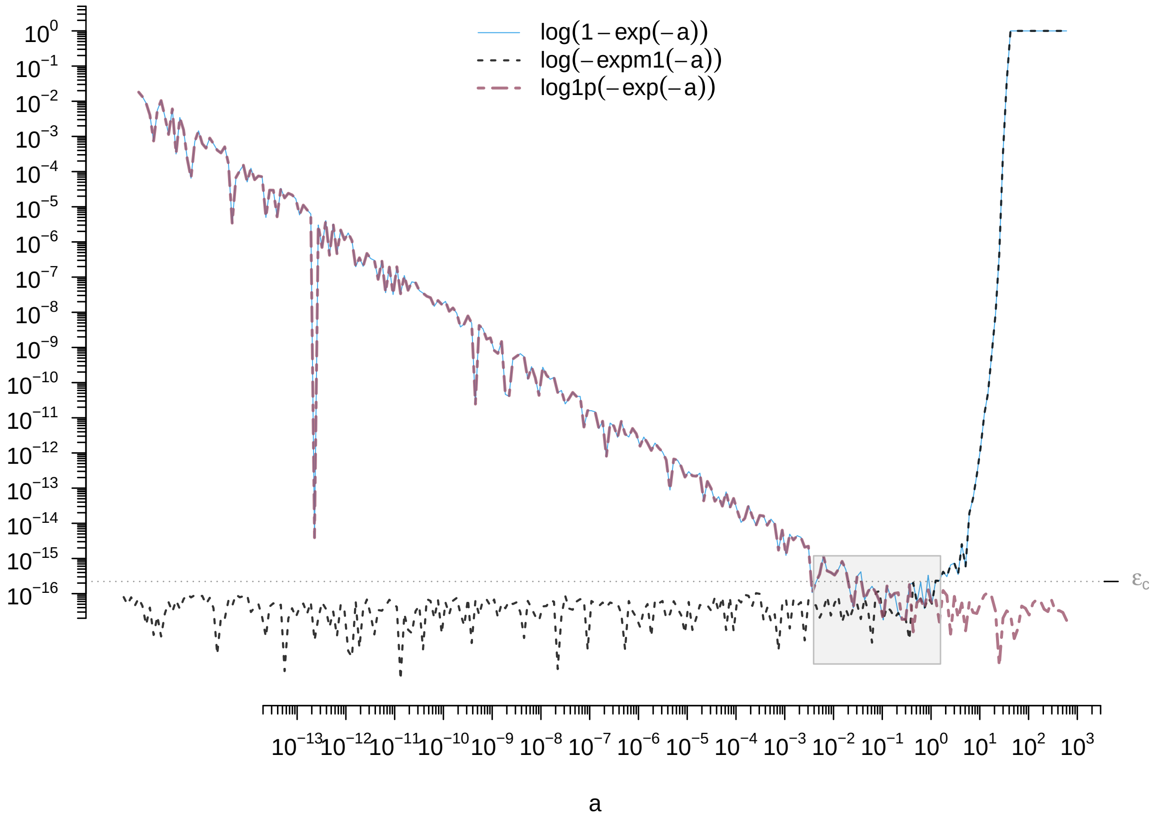


Figure 1: Relative errors* of the default, $\log(1-e^{-a})$, and the two methods "`expm1`" $\log(-\mathrm{expm1}(-a))$ and "`log1p`" $\mathrm{log1p}(-\exp(-a))$. Figure 2 will be a zoom into the gray rectangular region where all three curves are close.

assuming approximately normal errors. Notice however that neither the original, nor the log-transformed values have approximately symmetric errors, so we use `MASS::boxcox()` to determine the "correct" power transformation,

```
R> da <- cbind(a = a., as.data.frame(ra2))
R> library(MASS)
R> bc1 <- boxcox(abs(expm1) ~ a, data = da, lambda = seq(0,1, by=.01), plotit=.plot.BC)
R> bc2 <- boxcox(abs(log1p) ~ a, data = da, lambda = seq(0,1, by=.01), plotit=.plot.BC)
R> c(with(bc1, x[which.max(y)]),
     with(bc2, x[which.max(y)]))## optimal powers
```

```
[1] 0.38 0.30
```

```
R> ## ==> taking ^ (1/3) :
R> s1 <- with(da, smooth.spline(a, abs(expm1)^(1/3), df = 9))
R> s2 <- with(da, smooth.spline(a, abs(log1p)^(1/3), df = 9))
```

i.e, the optimal boxcox exponent turns out to be close to $\frac{1}{3}$, which we use for smoothing in a "zoom–in" of Figure 1. Then, the crossover point of the two curves already suggests that the cutoff, $a_0 = \log 2$ is empirically very close to optimal.

```
R> matplot(a., abs(ra2), type = "l", log = "y", # ylim = c(-1,1)*1e-12,
          col=cc[-1], lwd=ll[-1], lty=lt[-1],
```

```
            ylim = yl, xlab = "a", ylab = "", axes=FALSE)
R> legend("topright", leg[-1], col=cc[-1], lwd=ll[-1], lty=lt[-1], bty="n")
R> eaxis(1); eaxis(2); draw.machEps()
R> lines(a., predict(s1)$y ^ 3, col=cc[2], lwd=2)
R> lines(a., predict(s2)$y ^ 3, col=cc[3], lwd=2)
```

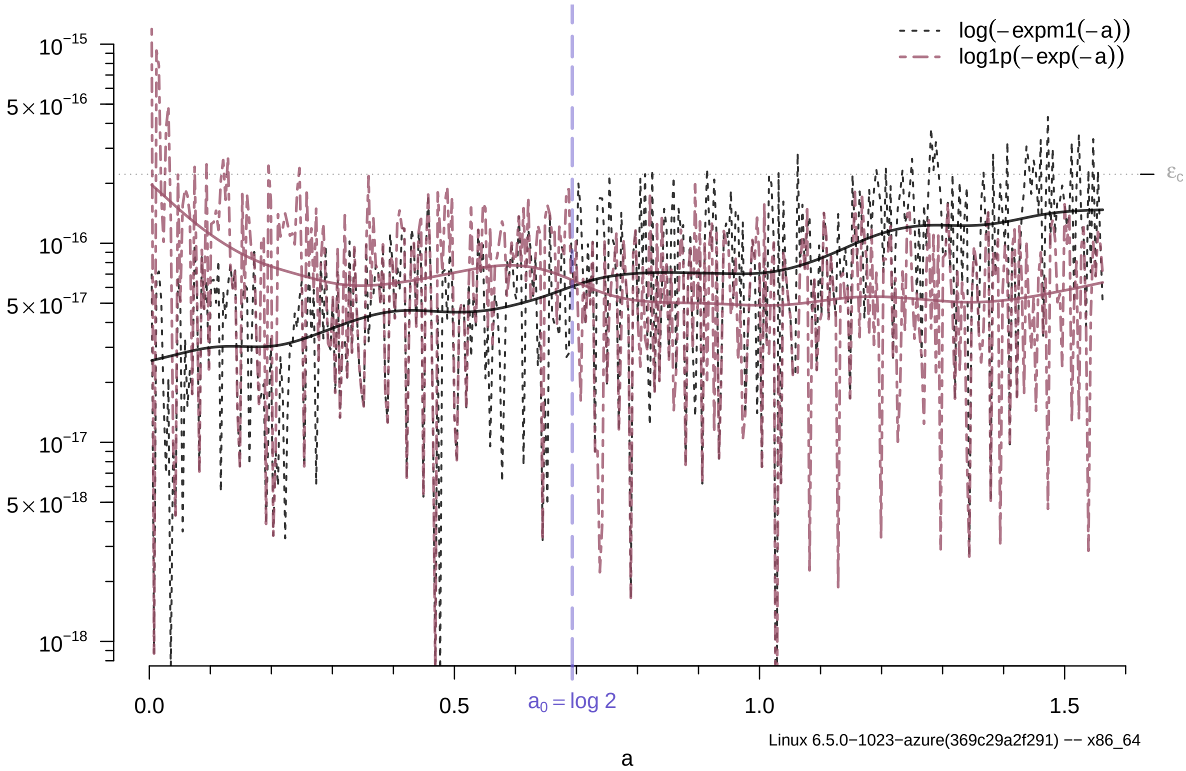


Figure 2: A "zoom in" of Figure 1 showing the region where the two basic methods, "`expm1`" and "`log1p`" switch their optimality with respect to their relative errors. Both have small relative errors in this region, typically below $\varepsilon_c :=$ `.Machine$double.eps` $= 2^{-52} \approx 2.22 \cdot 10^{-16}$. The smoothed curves indicate crossover close to $a = a_0 := \log 2$.

Why is it very plausible to take $a_0 := \log 2$ as approximately optimal cutoff? Already from Figure 2, empirically, an optimal cutoff a_0 is around 0.7. We propose to compute

$$f(a) = \log\left(1 - e^{-a}\right) = \log(1 - \exp(-a)), \quad a > 0, \tag{6}$$

by a new method or function `log1mexp(a)`. It needs a cutoff a_0 between choosing `expm1` for $0 < a \le a_0$ and `log1p` for $a > a_0$, i.e.,

$$f(a) = \mathrm{log1mexp}(a) := \begin{cases} \log(-\mathrm{expm1}(-a)) & 0 < a \le a_0 \quad (:= \log 2 \approx 0.693) \\ \mathrm{log1p}(-\exp(-a)) & \qquad a > a_0. \end{cases} \tag{7}$$

The mathematical argument for choosing a_0 is quite simple, at least informally: In which situations does $1 - e^{-a}$ loose bits (binary digits) *entirely independently* of the computational algorithm? Well, as soon as it "spends" bits just to store its closeness to 1. And that is as soon as $e^{-a} < \frac{1}{2} = 2^{-1}$, because then, at least one bit cancels. This however is equivalent to $-a < \log(2^{-1}) = -\log(2)$ or $a > \log 2 =: a_0$.

3. Computation of log(1+exp(x))

Related to $\mathrm{log1mexp}(a) = \log(1 - e^{-a})$ is the log survival function of the logistic distribution $\log(1 - F_L(x)) = \log \frac{1}{1+e^x} = -\log(1 + e^x) = -g(x)$, where

$$g(x) := \log(1 + e^x) = \mathrm{log1p}(e^x), \tag{8}$$

which has a "+"" instead of a "−", compared to log1mexp, and is easier to analyze and compute, its only problem being large x's where e^x overflows numerically.[4] As $g(x) = \log(1 + e^x) = \log(e^x(e^{-x} + 1)) = x + \log(1 + e^{-x})$, we see from (1) that

$$g(x) = x + \log(1 + e^{-x}) = x + e^{-x} + \mathcal{O}((e^{-x})^2), \tag{9}$$

for $x \to \infty$. Note further, that for $x \to -\infty$, we can simplify $g(x) = \log(1 + e^x)$ to e^x.

A simple picture quickly reveals how different approximations behave, where we have used `uniroot()` to determine the zero crossing, but will use slightly simpler cutoffs $x_0 = 37$, x_1 and x_2, in (10) below:

```
R> ## Find x0, such that  exp(x) =.= g(x) for x < x0 :
R> f0 <- function(x) { x <- exp(x) - log1p(exp(x))
                      x[x==0] <- -1 ; x }
R> u0 <- uniroot(f0, c(-100, 0), tol=1e-13)
R> str(u0, digits=10)
```

```
List of 5
 $ root      : num -36.39022698
 $ f.root    : num 2.465190329e-32
 $ iter      : int 81
 $ init.it   : int NA
 $ estim.prec: num 7.815970093e-14
```

```
R> x0 <- u0[["root"]] ## -36.39022698 --- note that ~= \log(\eps_C)
R> all.equal(x0, -52.5 * log(2), tol=1e-13)
```

```
[1] TRUE
```

```
R> ## Find x1, such that  x + exp(-x) =.= g(x) for x > x1 :
R> f1 <- function(x) { x <- (x + exp(-x)) - log1p(exp(x))
                      x[x==0] <- -1 ; x }
R> u1 <- uniroot(f1, c(1, 20), tol=1e-13)
R> str(u1, digits=10)
```

```
List of 5
 $ root      : num 16.40822612
 $ f.root    : num 3.552713679e-15
 $ iter      : int 18
 $ init.it   : int NA
 $ estim.prec: num 5.684341886e-14
```

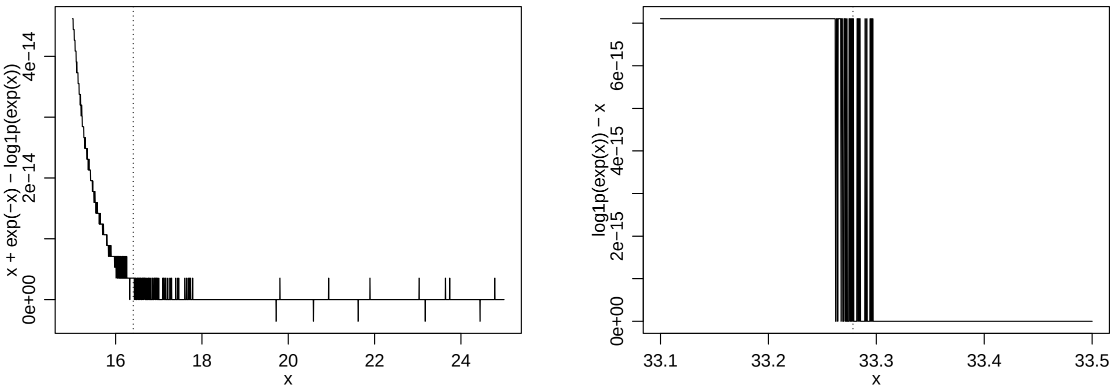
```
R> x1 <- u1[["root"]] ## 16.408226
R> ## Find x2, such that  x =.= g(x)  for x > x2 :
R> f2 <- function(x) { x <- log1p(exp(x)) - x ; x[x==0] <- -1 ; x }
R> u2 <- uniroot(f2, c(5, 50), tol=1e-13)
R> str(u2, digits=10)
```

[4] Indeed, for $x = 710$, $-g(x) = \log(1 - F_L(x)) =$ `plogis(710, lower=FALSE, log.p=TRUE)`, underflowed to `-Inf` in R versions before 2.15.1 (June 2012) from when on (10) has been used.

```
List of 5
 $ root      : num 33.2783501
 $ f.root    : num 7.105427358e-15
 $ iter      : int 9
 $ init.it   : int NA
 $ estim.prec: num 6.394884622e-14
```

```
R> x2 <- u2[["root"]] ## 33.27835
```

```
R> par(mfcol= 1:2, mar = c(4.1,4.1,0.6,1.6), mgp = c(1.6, 0.75, 0))
R> curve(x+exp(-x) - log1p(exp(x)), 15, 25,   n=2^11); abline(v=x1, lty=3)
R> curve(log1p(exp(x)) - x,       33.1, 33.5, n=2^10); abline(v=x2, lty=3)
```



Using double precision arithmetic, a fast and accurate computational method is to use

$$\hat{g}(x) = \text{log1pexp}(x) := \begin{cases} \exp(x) & x \le -37 \\ \text{log1p}(\exp(x)) & -37 < x \le x_1 := 18, \\ x + \exp(-x) & x_1 < x \le x_2 := 33.3, \\ x & x > x_2, \end{cases} \tag{10}$$

where only the cutoff $x_1 = 18$ is important and the other cutoffs just save computations.[5] Figure 3 visualizes the relative errors of the careless "default", $\log(1+e^x)$, its straightforward correction $\text{log1p}(e^x)$, the intermediate approximation $x + e^{-x}$, and the large x ($= x$), i.e., the methods in (10), depicting that the (easy to remember) cutoffs x_1 and x_2 in (10) are valid. Note that the default method is fully accurate on this x range and only problematic when e^x begins to overflow, i.e., $x > e_{\text{Max}}$, which is

```
R> (eMax <- .Machine$double.max.exp * log(2))
```

```
[1] 709.78
```

```
R> exp(eMax * c(1, 1+1e-15))
```

```
[1] 1.7977e+308         Inf
```

where we see that indeed e_{Max} =`eMax` is the maximal exponent without overflow.

[5] see plot `curve(log1p(exp(x)) - x, 33.1, 33.5, n=2^10)` above, revealing a somewhat fuzzy cutoff x_2.

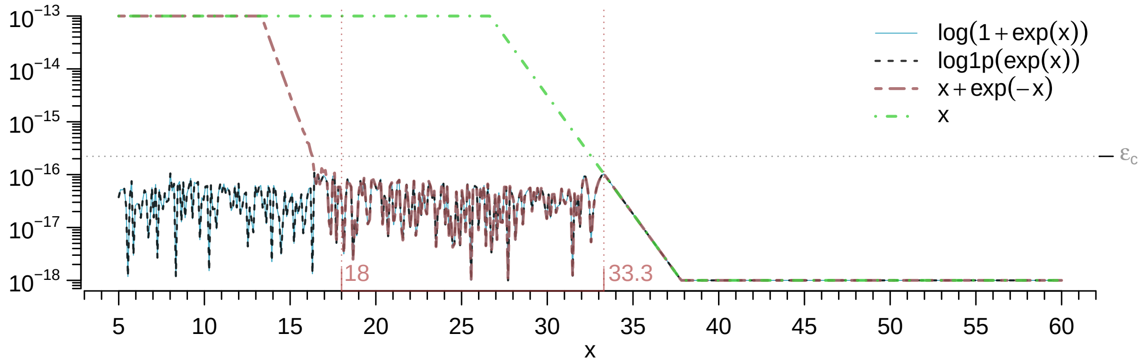


Figure 3: Relative errors (via **Rmpfr**, see footnote of Fig. 1) of four different ways to numerically compute $\log(1+e^x)$. Vertical bars at $x_1 = 18$ and $x_2 = 33.3$ visualize the (2nd and 3rd) cutpoints of (10).

4. Conclusion

We have used high precision arithmetic (R package **Rmpfr**) to empirically verify that computing $f(a) = \log\left(1 - e^{-a}\right)$ is accomplished best via equation (7). In passing, we have also shown that accurate computation of $g(x) = \log(1 + e^x)$ can be achieved via (10). Note that a version of this note is available as vignette (in `Sweave`, i.e., with complete R source) from the **Rmpfr** package vignettes.

Session Information

```
R> toLatex(sessionInfo(), locale=FALSE)
```

- R version 4.4.1 (2024-06-14), `x86_64-pc-linux-gnu`
- Running under: `Ubuntu 24.04 LTS`
- Matrix products: default
- BLAS: `/usr/lib/x86_64-linux-gnu/openblas-pthread/libblas.so.3`
- LAPACK: `/usr/lib/x86_64-linux-gnu/openblas-pthread/libopenblasp-r0.3.26.so` ; LAPACK version3.12.0
- Base packages: base, datasets, grDevices, graphics, methods, stats, utils
- Other packages: MASS 7.3-61, Rmpfr 0.9-6, gmp 0.7-4, polynom 1.4-1, sfsmisc 1.1-18
- Loaded via a namespace (and not attached): buildtools 1.0.0, compiler 4.4.1, knitr 1.48, maketools 1.3.0, sys 3.4.2, tools 4.4.1, xfun 0.45

References

DiDonato AR, Morris Jr AH (1992). "Algorithm 708: Significant digit computation of the incomplete beta function ratios." *ACM Transactions on Mathematical Software*, **18**(3), 360–373. ISSN 0098-3500. URL http://doi.acm.org/10.1145/131766.131776.

IEEE, Open Group (2004). "The Open Group Base Specifications Issue 6 — log1p." In *IEEE Std 1003.1, 2004 Edition*. URL http://pubs.opengroup.org/onlinepubs/009604599/functions/log1p.html.

R Development Core Team (2004). *R: A language and environment for statistical computing (Ver. 1.9.0)*. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-00-3, URL http://www.R-project.org.

Affiliation:

Martin Mächler
Seminar für Statistik, HG G 16
ETH Zurich
8092 Zurich, Switzerland
E-mail: maechler@stat.math.ethz.ch
URL: https://stat.ethz.ch/~maechler