Numerically Stable Frank Copula Functions via Multiprecision: R Package Rmpfr

Martin Mächler
ETH Zurich
May 2011 – July 2012 (LaTeX'ed January 7, 2019)

Abstract

The package **copula** (formerly `nacopula`) has provided functionality for Archimedean copulas, one of them the "Frank copula". Recently, explicit formulas for the density of those copulas have allowed for maximum likelihood estimation in high (e.g., $d = 150$)) dimensions, (Hofert, Mächler, and McNeil (2012)).

However for non-small dimensions, the evaluation of these densities is numerically challenging, and in some cases difficult. Here, we use high precision arithmetic from MPFR, via R package **Rmpfr** in order to get accurate values for the diagonal density of the Frank copula.

Subsequently, judiciously analysing and circumventing the numerical infelicities of the "traditional" evaluation, we gain accurate values even with traditional (double precision) arithmetic.

Keywords: Archimedean copulas, Frank Copula, Multiple Precision, R, MPFR, Rmpfr.

1. The diagonal density of Frank's copula

The "diagonal density" of a copula $C(\cdot)$ is the density of $\max(U_1, U_2, \ldots, U_d)$, where $\boldsymbol{U} = (U_1, U_2, \ldots, U_d)^\mathsf{T} \sim C$. The (cumulative) distribution function, by definition,

$$F^D(u) := P\left[\max(U_1, U_2, \ldots, U_d) \le u\right] = P\left[U_1 \le u, U_2, \le u, \ldots, U_d \le u\right] = C(u, u, \ldots, u), \tag{1}$$

evaluates the copula only on the diagonal and is therefore called *"the diagonal of C"*. Its density $f^D(u) := \frac{d}{du} F^D(u)$, is therefore called the diagonal density of C. For Archimedean copulas, i.e., where

$$C(\boldsymbol{u}) = C(\boldsymbol{u}; \psi) = \psi(\psi^{-1}(u_1) + \cdots + \psi^{-1}(u_d)), \ \boldsymbol{u} \in [0,1]^d, \tag{2}$$

the diagonal density is

$$\begin{aligned} f^D(u) &= \frac{d}{du} F^D(u) = \frac{d}{du} \psi\left(\sum\nolimits_{j=1}^{d} \psi^{-1}(u)\right) = \frac{d}{du} \psi(d \cdot \psi^{-1}(u)) = \\ &= \psi'\big(d \cdot \psi^{-1}(u)\big) \cdot d \cdot \frac{d}{du} \psi^{-1}(u) \\ &= d \cdot \psi'\big(d \cdot \psi^{-1}(u)\big) \cdot \left[\psi^{-1}\right]'(u). \end{aligned} \tag{3}$$

For this reason, the **copula** package's `dDiag()` function for computing the diagonal density $f^D(u)$ makes use of the following

```
> copula:::dDiagA
```

```
function (u, d, cop, log = FALSE)
{
    stopifnot(is.finite(th <- cop@theta), d >= 2)
    if (any(copAMH@name == c("AMH", "Frank", "Gumbel", "Joe")) &&
        any(i0 <- u == 0)) {
        if (log)
            u[i0] <- -Inf
        u[!i0] <- dDiagA(u[!i0], d = d, cop = cop, log = log)
        return(u)
    }
    if (log) {
        log(d) + cop@absdPsi(d * cop@iPsi(u, th), th, log = TRUE) +
            cop@absdiPsi(u, th, log = TRUE)
    }
    else {
        d * cop@absdPsi(d * cop@iPsi(u, th), th) * cop@absdiPsi(u,
            th)
    }
}
<bytecode: 0x55a179b575a0>
```

where the three functions

$$\texttt{absdPsi}(\texttt{t}, \texttt{thet}) = \left|\psi'_\theta(t)\right|, \tag{4}$$

$$\texttt{iPsi}(\texttt{u}, \texttt{thet}) = \psi_\theta^{-1}(u), \text{ and} \tag{5}$$

$$\texttt{absdiPsi}(\texttt{u}, \texttt{thet}) = \left|[\psi_\theta^{-1}]'(u)\right| \tag{6}$$

are all provided by the slots of the corresponding Archimedean copula family.

For the following explorations, we need a definition of `dDiagA` which is more flexible as it does not work with the copula family object but gets the three functions as arguments,

```
> dDiagA <- function(u, th, d, iPsi, absdPsi, absdiPsi, log = FALSE) {
      stopifnot(is.finite(th), d > 0, is.function(iPsi),
                is.function(absdPsi), is.function(absdiPsi))
      if(log) {
          log(d) + absdPsi(d*iPsi(u,th), th, log = TRUE) +
              absdiPsi(u, th, log = TRUE)
      } else {
          d* absdPsi(d*iPsi(u,th), th) * absdiPsi(u,th)
      }
  }
```

Now, for the Frank copula (Hofert and Maechler (2011)),

$$\psi_\theta(t) = -\frac{1}{\theta}\log\left(1-(1-e^{-\theta})e^{-t}\right), \quad \theta>0, \quad \text{hence,} \tag{7}$$

$$\psi_\theta^{-1}(u) = -\log\left(\frac{e^{-u\theta}-1}{e^{-\theta}-1}\right), \quad \text{and} \tag{8}$$

$$(-1)^k\psi_\theta^{(k)}(t) = \left|\psi_\theta^{(k)}(t)\right| = \frac{1}{\theta}\mathrm{Li}_{k-1}((1-e^{-\theta})e^{-t}), \quad \text{and} \tag{9}$$

$$\left|[\psi_\theta^{-1}]'(u)\right| = \theta/(e^{u\cdot\theta}-1), \tag{10}$$

where $\mathrm{Li}_s(z)$ is the *polylogarithm of order* s at z, defined as (analytic continuation of) $\sum_{k=1}^{\infty}\frac{z^k}{k^s}$. When $s=-n$, $n\in\mathbb{N}$, one has

$$\mathrm{Li}_{-n}(z) = \left(z\cdot\frac{\partial}{\partial z}\right)^n\frac{z}{1-z}, \tag{11}$$

and we note that here, only the first derivative is needed, $-\psi_\theta{}'(t) = \left|\psi_\theta{}'(t)\right|$, and hence only

$$\texttt{polylog(z, s = 0)} = \mathrm{Li}_0(z) = z/(1-z). \tag{12}$$

First note that numerically, $e^{-a}-1$ suffers from cancellation when $0<a\ll 1$, and the R (and C) function `expm1(-a)` is advisably used instead of `exp(-a) - 1`. For this reason, in **copula**, I had replaced the original `iPsi.0(u, th)` for $\psi_\theta^{-1}(u)$ by `iPsi.1()`, making use of `expm1()`. These and the derivative (10) originally were

```
> iPsi.0 <- function(u,theta) -log( (exp(-theta*u)-1) / (exp(-theta)-1) )
> iPsi.1 <- function(u,theta) -log(expm1(-u*theta) / expm1(-theta))
> absdiPsi.1 <- function(u, theta, log = FALSE)
    if(log) log(theta)-log(expm1(u*theta)) else theta/expm1(u*theta)
```

and the general k-th derivative (9), simplified, for $k=1$ (`degree = 1`),

```
> require("copula")# for polylog()
> absdPsi.1 <- function(t, theta, log=FALSE) {
    p <- -expm1(-theta)
    Li. <- polylog(log(p) - t, s = 0, log=log,
                   method="negI-s-Eulerian", is.log.z=TRUE)
    if(log) Li. - log(theta) else Li. / theta
  }
```

where we however now assume that the `polylog()` function for `s=0` would basically use the direct formula (12), such that we use

```
> absdPsi.2 <- function(t, theta, log=FALSE) {
    w <- log(-expm1(-theta)) - t
    Li. <- if(log) w - log(-expm1(w)) else -exp(w)/expm1(w)
    if(log) Li. - log(theta) else Li. / theta
  }
```

2. Computing the "diagonal MLE"

The most important use of the diagonal density is to compute the "diagonal maximum likelihood estimator", `dmle`, $\hat\theta^D$ which for a sample of observations $\boldsymbol{u}_1, \boldsymbol{u}_2, \ldots, \boldsymbol{u}_n$, ($\boldsymbol{u}_i\in[0,1]^d$) is

defined as minimizer of the negative log-likelihood,

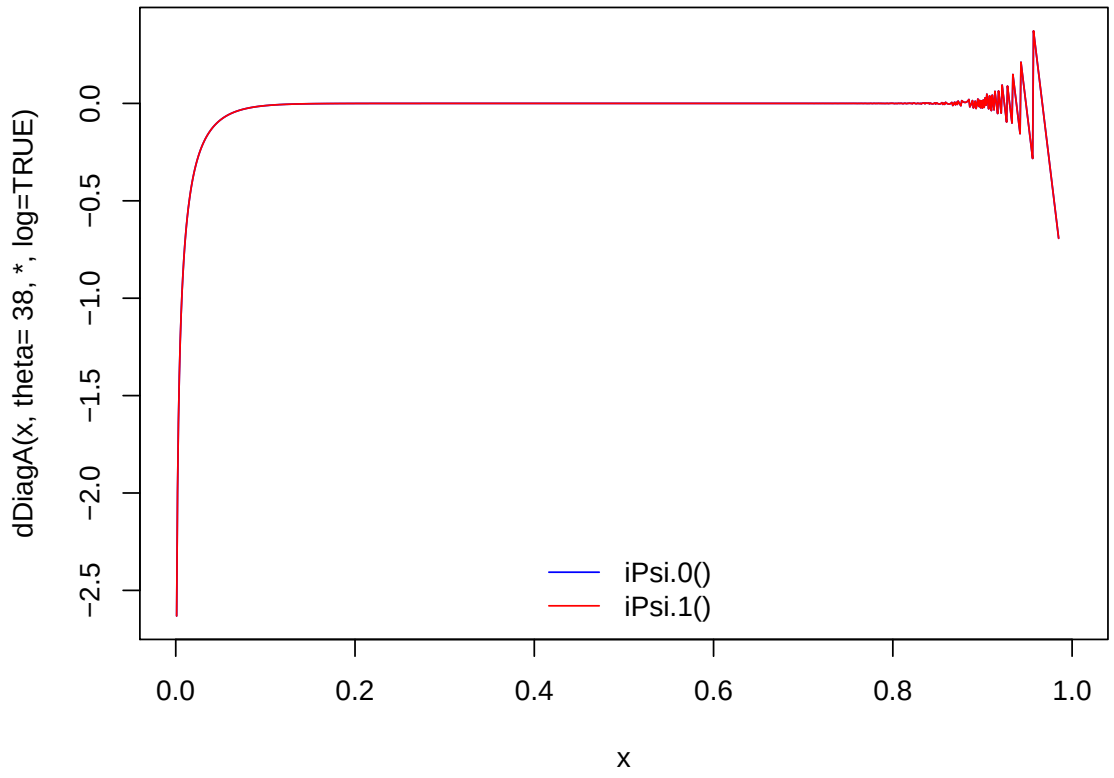
$$\hat{\theta}^D = \arg\min_\theta -l(\theta; \boldsymbol{u}_1, \ldots, \boldsymbol{u}_n), \quad \text{where} \tag{13}$$

$$l(\theta; \boldsymbol{u}_1, \ldots, \boldsymbol{u}_n) = \sum_{i=1}^{n} \log f^D(\tilde{u}_i) \quad \text{and} \tag{14}$$

$$\tilde{u}_i = \max_{j=1,\ldots,d} u_{i,j}. \tag{15}$$

In our exploration of the `dmle` estimator, we found cases with numerical problems, at first already in evaluating the logarithm of the diagonal density $\log f^D = \log f^D_\theta(u)$`= dDiag(u, theta, *, log=TRUE)` for non-small θ and "large" u, i.e., $u \approx 1$:

```
> curve(dDiagA(x, th = 38, d = 2, iPsi = iPsi.0,
                absdPsi=absdPsi.1, absdiPsi=absdiPsi.1, log = TRUE),
        ylab = "dDiagA(x, theta= 38, *, log=TRUE)",
        0, 1, col = 4, n = 1000)
> ## and using the slightly better   iPsi.1  does not help here:
> curve(dDiagA(x, th = 38, d = 2, iPsi = iPsi.1,
                absdPsi=absdPsi.2, absdiPsi=absdiPsi.1, log = TRUE),
        add = TRUE, col = 2, n=1000)
> legend("bottom", c("iPsi.0()","iPsi.1()"),col=c(4,2), lty=1, bty="n")
```



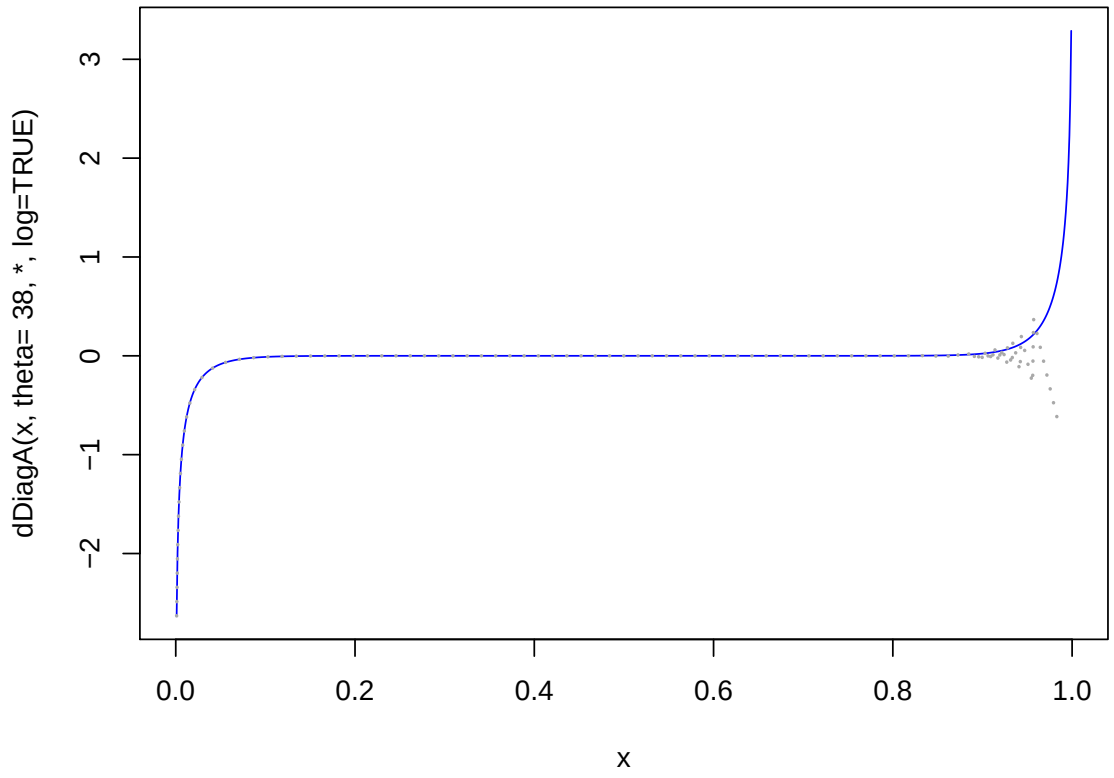
However, it's not hard to see that indeed our initial computation of Frank's ψ_θ^{-1}, i.e, (8), $\psi_\theta^{-1}(u) = -\log\left(\frac{1-e^{-u\theta}}{1-e^{-\theta}}\right)$, suffers from "division cancellation" for "large" θ ($\theta = 38$ in ex.) when computed directly with `iPsi.0()`, see above, and that the improvement of using `expm1(-t)`

instead of `exp(-t) - 1`, as used in `iPsi.1()`, see above, helps only for $t \approx 0$. However, we can rewrite ψ_θ^{-1} as

$$\psi_\theta^{-1}(u) = -\log\left(1 - \frac{e^{-u\theta} - e^{-\theta}}{1 - e^{-\theta}}\right), \tag{16}$$

which when using `log1p(e)` for $\log(1 + e)$ is much better numerically:

```
> iPsi.2 <- function(u,theta) -log1p((exp(-u*theta)-exp(-theta)) / expm1(-theta))
> curve(dDiagA(x, th = 38, d = 2, iPsi = iPsi.2,
               absdPsi=absdPsi.2, absdiPsi=absdiPsi.1, log = TRUE),
        ylab = "dDiagA(x, theta= 38, *, log=TRUE)",
        0, 1, col = 4, n = 1000)
> ## previously
> curve(dDiagA(x, th = 38, d = 2, iPsi = iPsi.1,
               absdPsi=absdPsi.2, absdiPsi=absdiPsi.1, log = TRUE),
        add = TRUE, col = "darkgray", lwd=2, lty=3, n=1000)
```



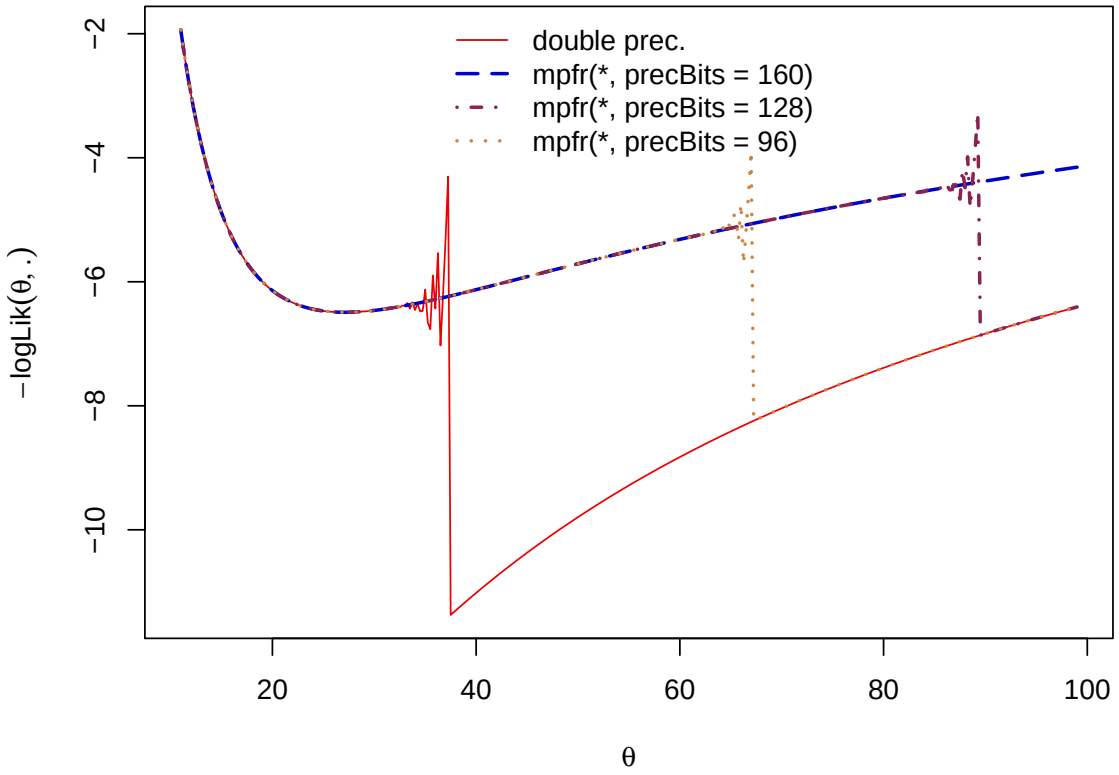
Unfortunately, this is not enough to get numerically stable evaluations of the negative log-likelihood $l()$:

```
> d <- 5
> (theta <- copFrank@iTau(tau = 0.75))

[1] 14.13852

> cop <- onacopulaL("Frank", list(theta, 1:d))
> set.seed(1); for(l in 1:4) U <- rnacopula(n = 100, cop)
> U. <- sort(apply(U, 1, max)) # build the max
```

```
> mlogL <- function(theta)
      -sum(dDiagA(U., theta, d=d, iPsi = iPsi.2,
                  absdPsi=absdPsi.2, absdiPsi=absdiPsi.1,
                  log = TRUE))
```

Now, plot this negative log likelihood function in an interval θ, close to what is proposed **copula**'s `initOpt()` function, defining a utility function that we'll reuse later:

```
> p.mlogL <- function(th, mlogL, col= "red2", lwd = 1, lty = 1,
                      add= FALSE) {
    stopifnot(is.numeric(th), is.function(mlogL))
    nll <- vapply(th, mlogL, 0.)
    if(add) lines(nll ~ th, col=col, lwd=lwd, lty=lty)
    else plot(nll ~ th, xlab=expression(theta),
              ylab = expression(- logLik(theta, .)),
              type = "l", col=col, lwd=lwd, lty=lty)
    invisible(nll) # return invisibly
  }
> thet <- seq(11, 99, by = 1/4)
> p.mlogL(thet, mlogL)
> require("Rmpfr")## compute the same with *high* accuracy ...
> ## using three different precisions:
> MPrecBits <- c(160, 128, 96)
> mkNm <- function(bits) sprintf("%03d.bits", bits)
> ## As it takes a while, cache the result:
> fnam <- sprintf("mlogL_mpfr_%s.rds", Sys.info()[["machine"]])
> if(file.exists(fn <- file.path(copDDir, fnam))) {
      nllMP <- readRDS(fn)
  } else {
      print(system.time(
      nllMP <- lapply(MPrecBits, function(pBit) {
          nlM <- thM <- mpfr(thet, precBits = pBit)
          ## (vapply() does not work for "Rmpfr":)
          for(i in seq_along(thet)) nlM[i] <- mlogL(thM[i])
          nlM
      })
    )) ## ~ 18 sec [R 3.5.1 nb-mm4 core I7vPro]; was 91 sec [2011-06-14 nb-mm icore 5]
    names(nllMP) <- mkNm(MPrecBits)
    copSrcDDir <- if(Sys.getenv("USER") == "maechler")
        '~/R/Pkgs/copula/inst/doc' else ""
    if(file.exists(copSrcDDir))# <<- only for certain users; not on CRAN etc
        saveRDS(nllMP, file = file.path(copSrcDDir, fnam), version = 2)
  }
> colB <- c("blue3","violetred4","tan3")
> ltyB <- c(5:3)
> lwdB <- c(2,2,2)
> for(i in seq_along(nllMP)) {
      lines(thet, as.numeric(nllMP[[i]]),
            col=colB[i], lty = ltyB[i], lwd = lwdB[i])
  }
> leg <- c("double prec.", sprintf("mpfr(*, precBits = %d)", MPrecBits))
> legend("top", leg,
         col= c("red3",colB), lty=c(1, ltyB), lwd=c(1,lwdB), bty="n")
```



So, clearly, high-precision computations can solve the numerical problems, if the precision is high enough. E.g., for $\theta = 100$, it needs more than 128 bits precision.

Let us look at the phenomenon in more details now. The flesh in the `mlogL()` computation is (up to the constant $\log(d)$, $d = 5$), only the sum of the two terms

```
absdPsi(d*iPsi(u,th), th, log = TRUE) +
 absdiPsi(u,          th, log = TRUE)
```

currently, with the three functions

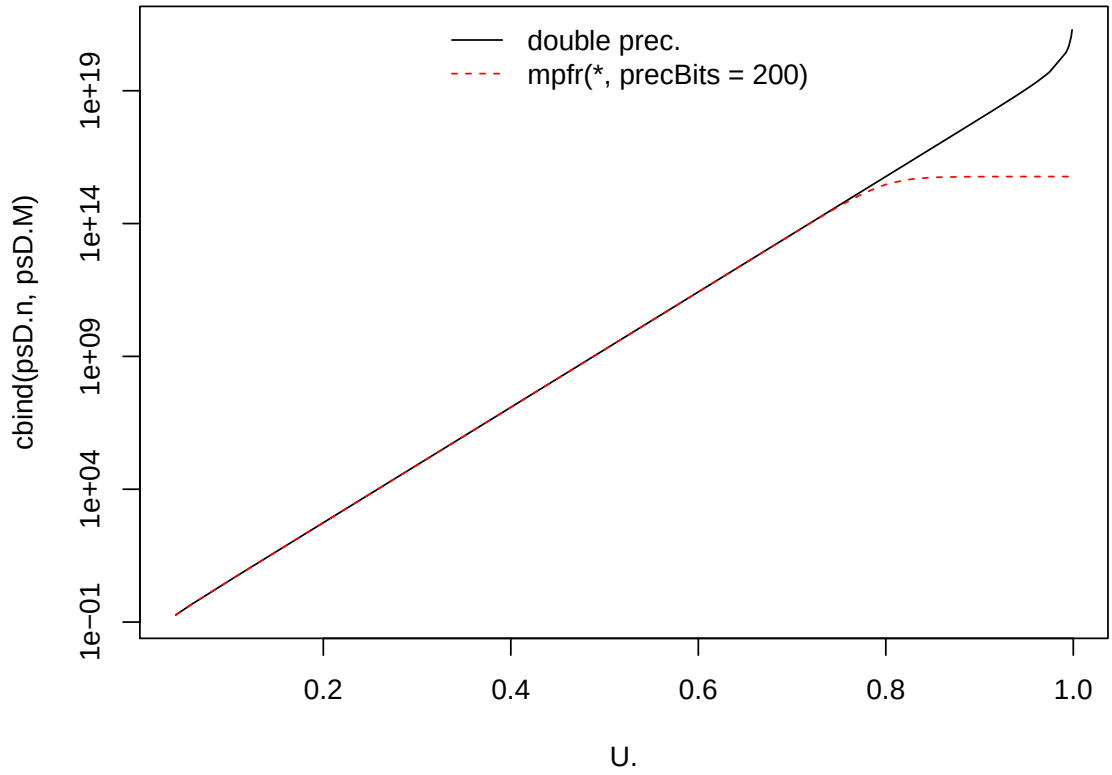
```
iPsi = iPsi.2; absdPsi = absdPsi.2; absdiPsi = absdiPsi.1
```

where we have already tried to ensure that the `iPsi()` function is ok, but now can confirm it, e.g., for $\theta = 50$. Further note, that using high-precision arithmetic, we can also "partially afford" to use the simplistic `iPsi.0()` function instead of the more stable `iPsi.2()` one:

```
> stopifnot(all.equal(iPsi.2(U.,  50 ),
                      iPsi.2(U., mpfr(50, 96))),
            all.equal(iPsi.0(U., mpfr(50, 200)),
              pI.U <- iPsi.2(U., mpfr(50, 200)), tol=1e-50) )
```

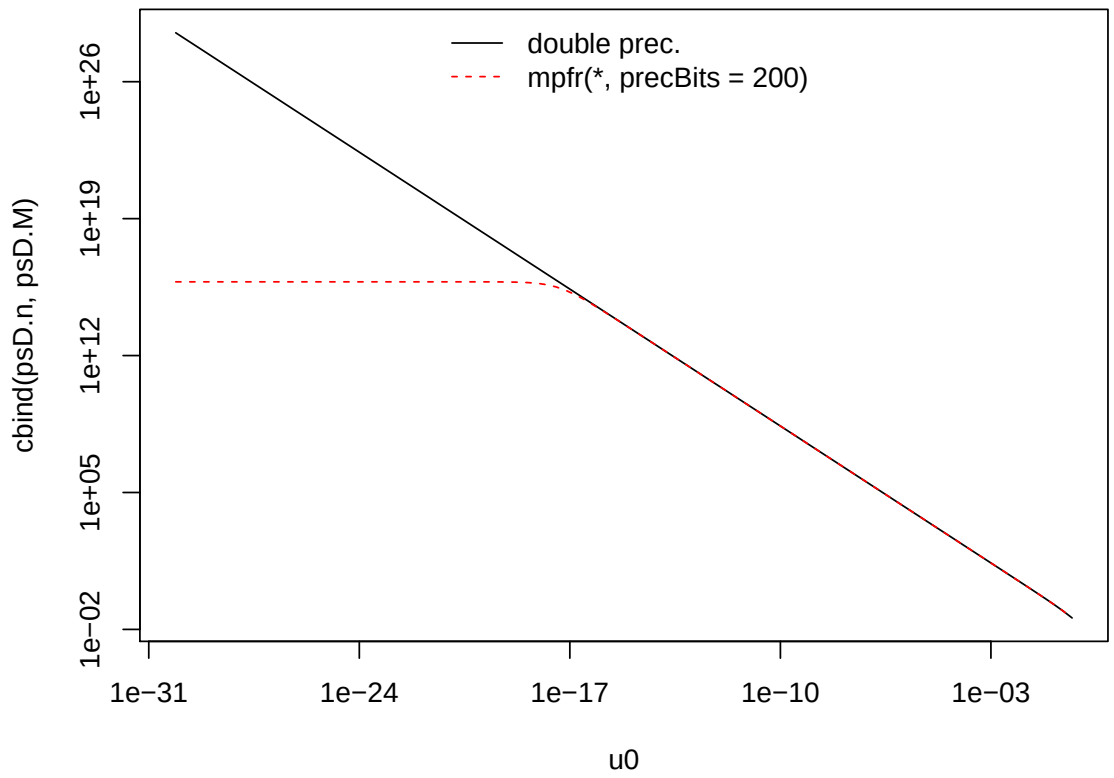
However, we can observe dramatic differences in `absdPsi.2()` ($= |\psi'(.)|$):

```
> psD.n <-            absdPsi.2(as.numeric(pI.U), 40)
> psD.M <- as.numeric(absdPsi.2(pI.U,        mpfr(40, 200)))
> matplot(U., cbind(psD.n, psD.M), type="l", log="y")
> legend("top", c("double prec.", "mpfr(*, precBits = 200)"),
         col= 1:2, lty=1:2, bty="n")
```



where we see a very large difference (note the log scale!) for $u \approx 1$, i.e., for very small `pI.U= iPsi.2(U., theta)`$= \psi_\theta^{-1}(u)$.

```
> u0 <- 2^-(100:1)
> psD.n <-            absdPsi.2(u0, 40)
> psD.M <- as.numeric(absdPsi.2(u0, mpfr(40, 200)))
> matplot(u0, cbind(psD.n, psD.M), type="l", log="xy")
> legend("top", c("double prec.", "mpfr(*, precBits = 200)"),
         col= 1:2, lty=1:2, bty="n")
```



Further investigation shows that the culprit is really the use of `log(-expm1(-theta))` inside `absdPsi.2()` which underflows for large theta, and hence should be replaced by the generally accurate `log1mexp()`, see Mächler (2012).

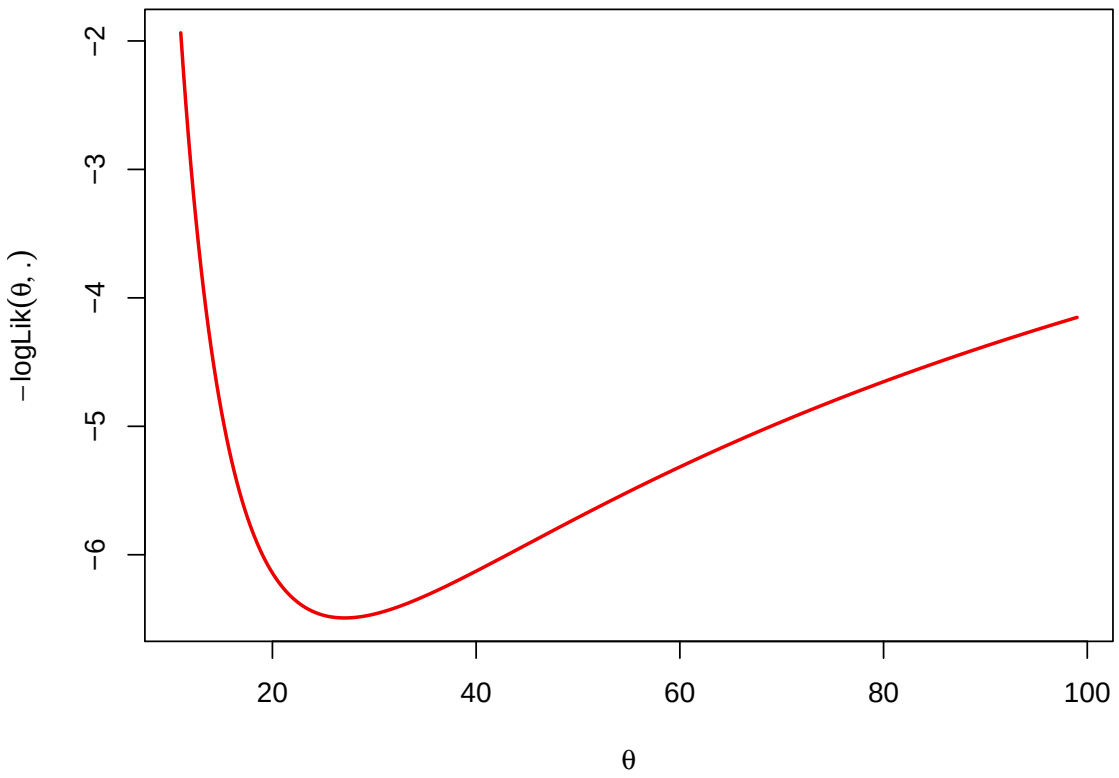
```
> log1mexp <- function(a) # accurately compute log(1-exp(-a))
  {
      stopifnot(a >= 0)
      r <- a
      tst <- a <= log(2)
      r[ tst] <- log(-expm1(-a[ tst]))
      r[!tst] <- log1p(-exp(-a[!tst]))
      r
  }
```

so that we rather compute $|\psi'(.)|$ via

```
> absdPsi.3 <- function(t, theta, log=FALSE) {
    w <- log1mexp(theta) - t
    Li. <- if(log) w - log1mexp(-w) else -exp(w)/expm1(w)
    if(log) Li. - log(theta) else Li. / theta
  }
```

Does this already solve the "diagonal likelihood" problem? We investigate via the graphic

```
> p.mlogL(th = seq(11, 99, by = 1/4),
          mlogL = (mlogL2 <- function(theta)
                   -sum(dDiagA(U., theta, d=d, iPsi = iPsi.2,
                               absdPsi = absdPsi.3, absdiPsi=absdiPsi.1,
                               log = TRUE))), lwd = 2)
```



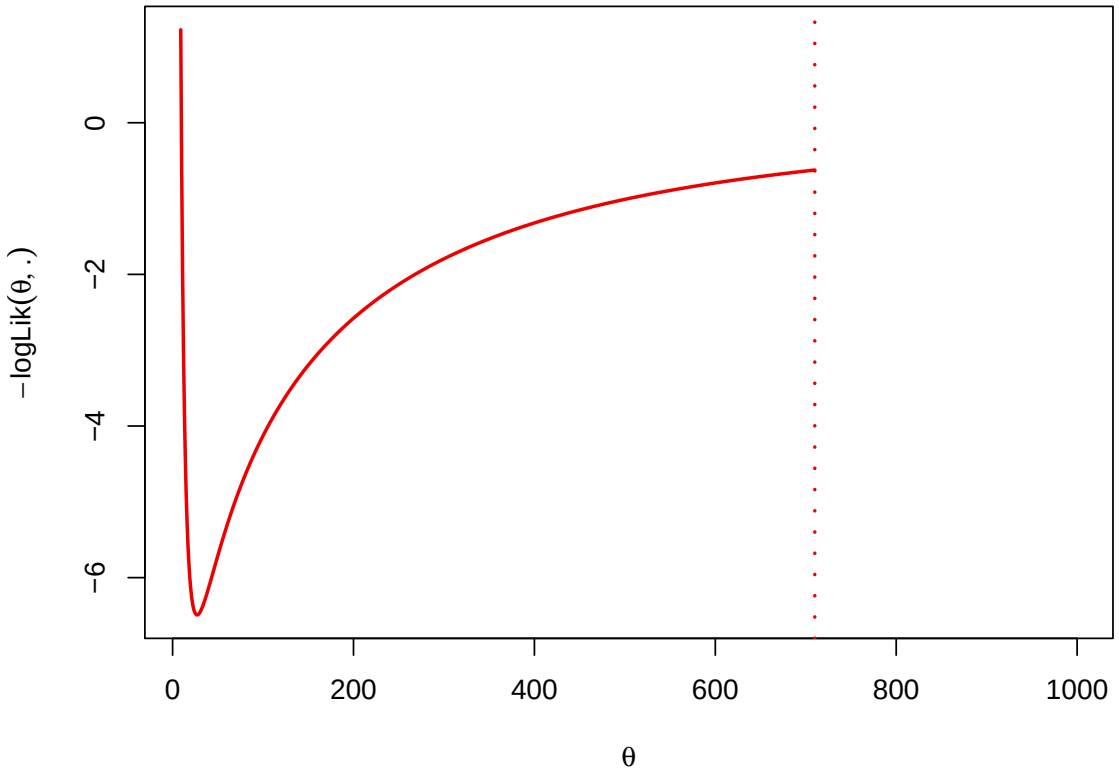
Yes, indeed, using a numerically stable version for `absdPsi()` did solve the numerical problem of computing, the "diagonal likelihood", and hence the `dmle()` ("diagonal MLE").

Well, if we really insist, there are more problems, but probably not really practical:

```
> thet <- 9:1000
> nll <- p.mlogL(thet, mlogL = mlogL2, lwd=2)
> (th0 <- thet[i0 <- max(which(is.finite(nll)))])
```

```
[1] 710
```

```
> abline(v = th0, col="red2", lty="15", lwd=2)
```



where we see that for $\theta > 710$, `mlogL()` is not finite, e.g.,

```
> dDiagA(0.999, 715, d = d, iPsi = iPsi.2, absdPsi = absdPsi.3,
                            absdiPsi = absdiPsi.1, log = TRUE)

[1] -Inf
```

which after closer inspection is from the `absdiPsi(u, th, log = TRUE)` part of `dDiagA()` and that, see (6), uses `log(theta)-log(expm1(u*theta))`, where clearly already `expm1(u*theta)) = expm1(0.999 * 715)` numerically overflows to `Inf`:

```
> absdiPsi.1(0.999, th = 715, log=TRUE)

[1] -Inf
```

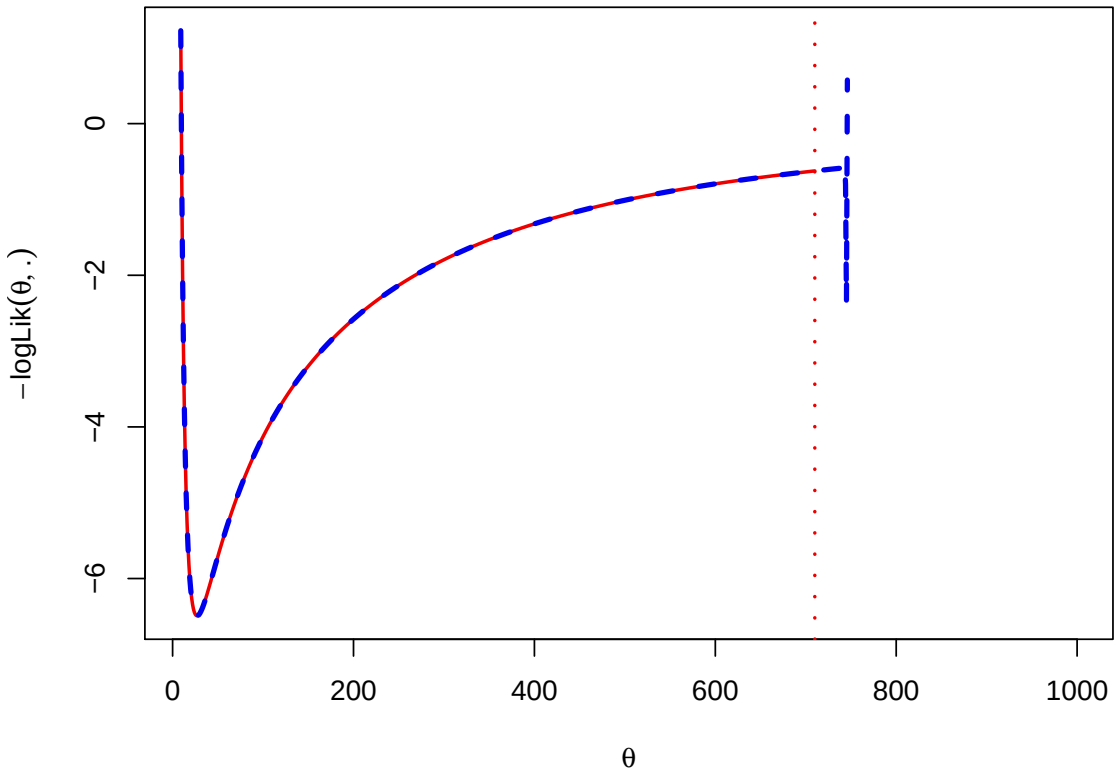
However, as $\log(\text{expm1}(y)) = \log(e^y - 1) = \log(e^y(1 - e^{-y})) = y + \log(1 - e^{-y})$, the "numerical stable" solution is to replace `log(expm1(y))` by `y + log1mexp(y)`, such that we will use

```
> absdiPsi.2 <- function(u, theta, log = FALSE)
    if(log) log(theta)- {y <- u*theta; y + log1mexp(y)} else theta/expm1(u*theta)
```

Unfortunately, this improves the situation for large θ only slightly:

```
> plot(nll ~ thet, xlab=expression(theta),
       ylab = expression(- logLik(theta, .)),
       type = "l", col="red2", lwd=2)
> abline(v = th0, col="red2", lty="15", lwd=2)
> nll3 <- p.mlogL(thet, mlogL = function(theta)
                   -sum(dDiagA(U., theta, d=d, iPsi= iPsi.2, absdPsi= absdPsi.3,
                               absdiPsi = absdiPsi.2, log = TRUE)),
                 col = "blue2", lwd=3, lty=2, add = TRUE)
```

```
> nll3[thet == 800]
[1] -Inf
```



where we can see, that this time, the numerical overflow to $-\infty$ happens in `absdPsi(d*iPsi(u,th), th, log = TRUE)`, as `iPsi(u,th) = iPsi(0.999, th=800)`$=0$ underflows to zero — by necessity: the correct value is smaller than the smallest representable double precision number:

```
> pI <- iPsi.2(u=0.999, th= mpfr(800, 200))
> cat(sapply(list(pI, .Machine$double.xmin),
             format, digits = 7), "\n")
```

```
4.495130e-348 2.225074e-308
```

The only solution here is to allow passing the *logarithm* $\log(t)$ instead of t to `absdPsi()`, i.e., we start computing `log(iPsi(.))` directly (evading the underflow to 0 there).

. . . . . to be continued.

3. Session Information

```
> toLatex(sessionInfo())
```

- R version 3.5.2 Patched (2019-01-02 r75945), `x86_64-pc-linux-gnu`
- Locale: `LC_CTYPE=en_US.UTF-8`, `LC_NUMERIC=C`, `LC_TIME=en_US.UTF-8`, `LC_COLLATE=C`, `LC_MONETARY=en_US.UTF-8`, `LC_MESSAGES=C`, `LC_PAPER=en_US.UTF-8`,

LC_NAME=C, LC_ADDRESS=C, LC_TELEPHONE=C, LC_MEASUREMENT=en_US.UTF-8, LC_IDENTIFICATION=C

- Running under: Debian GNU/Linux buster/sid
- Matrix products: default
- BLAS: /srv/R/R-patched/build.19-01-03/lib/libRblas.so
- LAPACK: /srv/R/R-patched/build.19-01-03/lib/libRlapack.so
- Base packages: base, datasets, grDevices, graphics, grid, methods, parallel, splines, stats, stats4, tools, utils
- Other packages: Rmpfr 0.7-1, VGAM 1.0-6, abind 1.4-5, bbmle 1.0.20, copula 0.999-20, gmp 0.5-13.2, gridExtra 2.3, gsl 1.9-10.3, lattice 0.20-38, mev 1.11, qrng 0.0-5, randtoolbox 1.17.1, rngWELL 0.10-5, rugarch 1.4-0, sfsmisc 1.1-3
- Loaded via a namespace (and not attached): ADGofTest 0.3, DistributionUtils 0.6-0, GeneralizedHyperbolic 0.8-4, KernSmooth 2.23-15, MASS 7.3-51.1, Matrix 1.2-15, R6 2.3.0, Rcpp 1.0.0, Rsolnp 1.16, Runuran 0.24, SkewHyperbolic 0.4-0, assertthat 0.2.0, bayesplot 1.6.0, bindr 0.1.1, bindrcpp 0.2.2, boot 1.3-20, colorspace 1.3-2, compiler 3.5.2, crayon 1.3.4, digest 0.6.18, dplyr 0.7.8, evaluate 0.12, evd 2.3-3, expm 0.999-3, ggplot2 3.1.0, ggridges 0.5.1, glue 1.3.0, gmm 1.6-2, gtable 0.2.0, htmltools 0.3.6, ismev 1.42, knitr 1.21, ks 1.11.3, lazyeval 0.2.1, magrittr 1.5, mclust 5.4.2, mgcv 1.8-26, munsell 0.5.0, mvtnorm 1.0-8, nleqslv 3.3.2, nlme 3.1-137, nloptr 1.2.1, numDeriv 2016.8-1, partitions 1.9-19, pcaPP 1.9-73, pillar 1.3.1, pkgconfig 2.0.2, plyr 1.8.4, polynom 1.3-9, pspline 1.0-18, purrr 0.2.5, revdbayes 1.3.2, rlang 0.3.0.1, rmarkdown 1.11, rootSolve 1.7, sandwich 2.5-0, scales 1.0.0, spd 2.0-1, stabledist 0.7-1, stringi 1.2.4, stringr 1.3.1, tibble 2.0.0, tidyselect 0.2.5, truncnorm 1.0-8, xfun 0.4, xts 0.11-2, yaml 2.2.0, zoo 1.8-4

4. Conclusion

References

Hofert M, Mächler M, McNeil AJ (2012). "Likelihood Inference for Archimedean Copulas in High Dimensions Under Known Margins." *Journal of Multivariate Analysis*, **110**, 133–150. doi:10.1016/j.jmva.2012.02.019.

Hofert M, Maechler M (2011). "Nested Archimedean Copulas Meet R: The nacopula Package." *Journal of Statistical Software*, **39**(9), 1–20. ISSN 1548-7660. URL http://www.jstatsoft.org/v39/i09.

Mächler M (2012). "Accurately Computing $\log(1 - \exp(-|a|))$." URL https://cran.r-project.org/web/packages/Rmpfr/vignettes/log1mexp-note.pdf.

Affiliation:

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