Numerically Stable Frank Copula Functions via Multiprecision: R Package Rmpfr

Martin Mächler
ETH Zurich

May 2011 (LaTeX'ed September 23, 2011)

Abstract

The package **nacopula** .... Archimedean copulas
The package **Rmpfr** ....

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 **nacopula** package's `dDiag()` function for computing the diagonal density $f^D(u)$ makes use of the following

```
> nacopula:::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@psiDabs(d * cop@psiInv(u, th), th, log = TRUE) +
            cop@psiInvD1abs(u, th, log = TRUE)
    }
    else {
        d * cop@psiDabs(d * cop@psiInv(u, th), th) * cop@psiInvD1abs(u,
            th)
    }
}
```

where the three functions

$$\texttt{psiDabs(t,thet)} = \left|\psi_\theta'(t)\right|, \tag{4}$$

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

$$\texttt{psiInvD1abs(u,thet)} = \left|[\psi_\theta^{-1}]'(u)\right| \tag{6}$$

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, psiInv, psiDabs, psiInvD1abs, log = FALSE) {
      stopifnot(is.finite(th), d > 0, is.function(psiInv),
                is.function(psiDabs), is.function(psiInvD1abs))
      if(log) {
          log(d) + psiDabs(d*psiInv(u,th), th, log = TRUE) +
              psiInvD1abs(u, th, log = TRUE)
      } else {
          d* psiDabs(d*psiInv(u,th), th) * psiInvD1abs(u,th)
      }
  }
```

Now, for the Frank copula (see Hofert and Mächler (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

nacopula, I had replaced the original `psiInv.0(u, th)` for $\psi_\theta^{-1}(u)$ by `psiInv.1()`, making use of `expm1()`. These and the derivative (9) originally were

```
> psiInv.0 <- function(u,theta) -log( (exp(-theta*u)-1) / (exp(-theta)-1) )
> psiInv.1 <- function(u,theta) -log(expm1(-u*theta) / expm1(-theta))
> psiInvD1abs.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 (10), simplified, for $k = 1$ (`degree = 1`),

```
> require("nacopula")# for polylog()
> psiDabs.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

```
> psiDabs.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 $\boldsymbol{u}_1, \boldsymbol{u}_2, \dots, \boldsymbol{u}_n$, ($\boldsymbol{u}_i \in [0,1]^d$) is defined as minimizer of the negative log-likelihood,

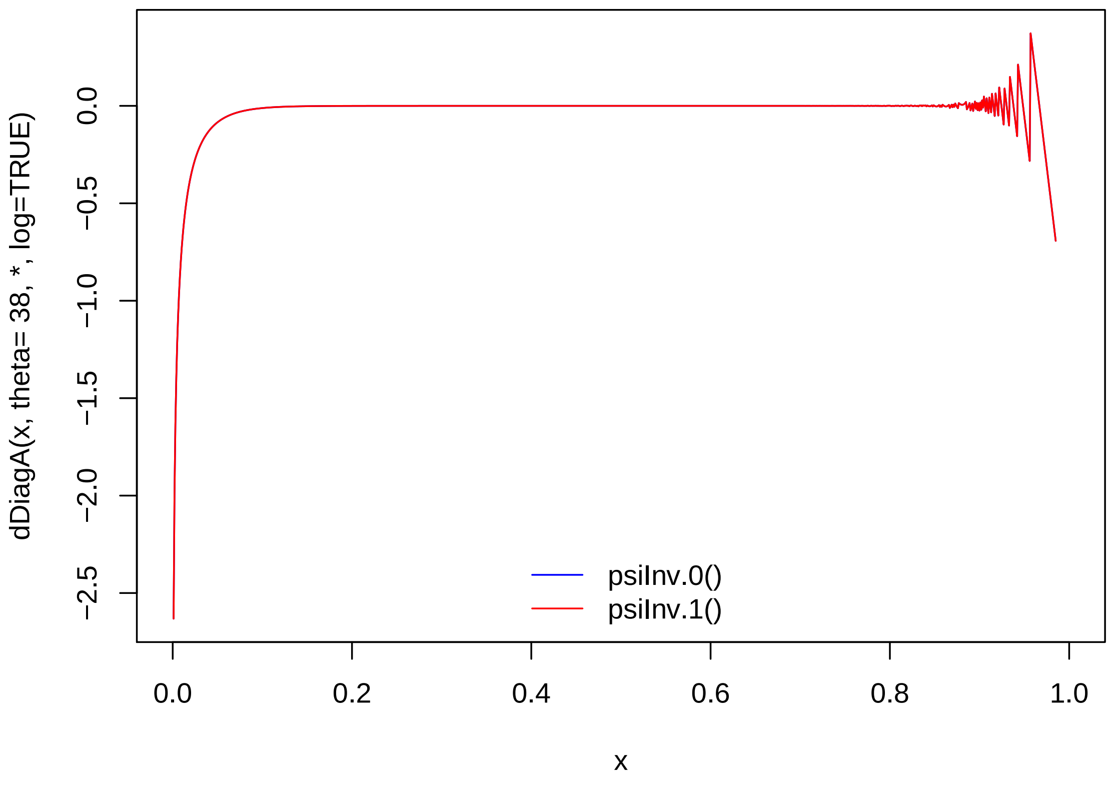
$$\hat\theta^D = \arg\min_\theta -l(\theta; \boldsymbol{u}_1, \dots, \boldsymbol{u}_n), \quad \text{where} \tag{13}$$

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

$$\tilde{u_i} = \max_{j=1,\dots,d} u_{i,j}. \tag{15}$$

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

```
> curve(dDiagA(x, th = 38, d = 2, psiInv = psiInv.0,
              psiDabs=psiDabs.1, psiInvD1abs=psiInvD1abs.1, log = TRUE),
        ylab = "dDiagA(x, theta= 38, *, log=TRUE)",
        0, 1, col = 4, n = 1000)
> ## and using the slightly better   psiInv.1  does not help here:
> curve(dDiagA(x, th = 38, d = 2, psiInv = psiInv.1,
              psiDabs=psiDabs.2, psiInvD1abs=psiInvD1abs.1, log = TRUE),
        add = TRUE, col = 2, n=1000)
> legend("bottom", c("psiInv.0()","psiInv.1()"),col=c(4,2), lty=1, bty="n")
```

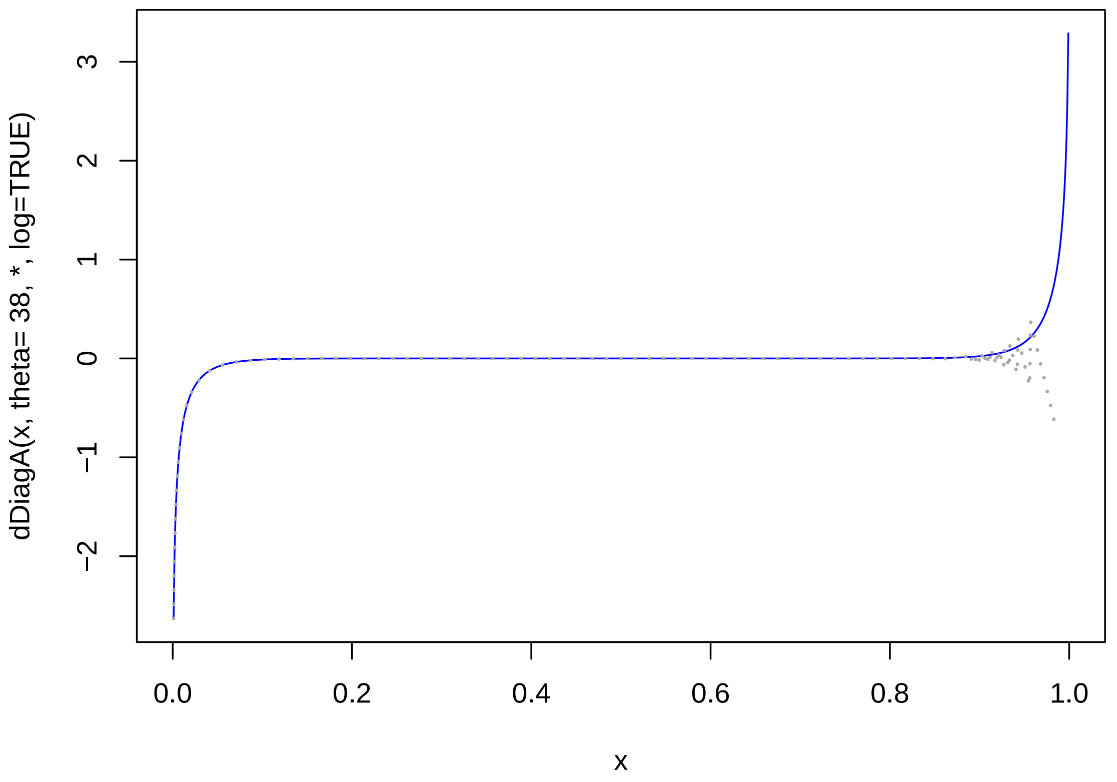


However, it's not hard to see that indeed our initial computation of Frank's ψ^{-1}, i.e, (8), $\psi_\theta^{-1}(u) = -\log\left(\frac{1-e^{-u\theta}}{1-e^{-\theta}}\right)$, suffers from "division cancellation" for "large" θ ($\theta = 38$ in ex.) when computed directly with `psiInv.0()`, see above,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

```
> psiInv.2 <- function(u,theta) -log1p((exp(-u*theta)-exp(-theta)) / expm1(-theta))
> curve(dDiagA(x, th = 38, d = 2, psiInv = psiInv.2,
               psiDabs=psiDabs.2, psiInvD1abs=psiInvD1abs.1, log = TRUE),
       ylab = "dDiagA(x, theta= 38, *, log=TRUE)",
       0, 1, col = 4, n = 1000)
> ## previously
> curve(dDiagA(x, th = 38, d = 2, psiInv = psiInv.1,
               psiDabs=psiDabs.2, psiInvD1abs=psiInvD1abs.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 $l()$:

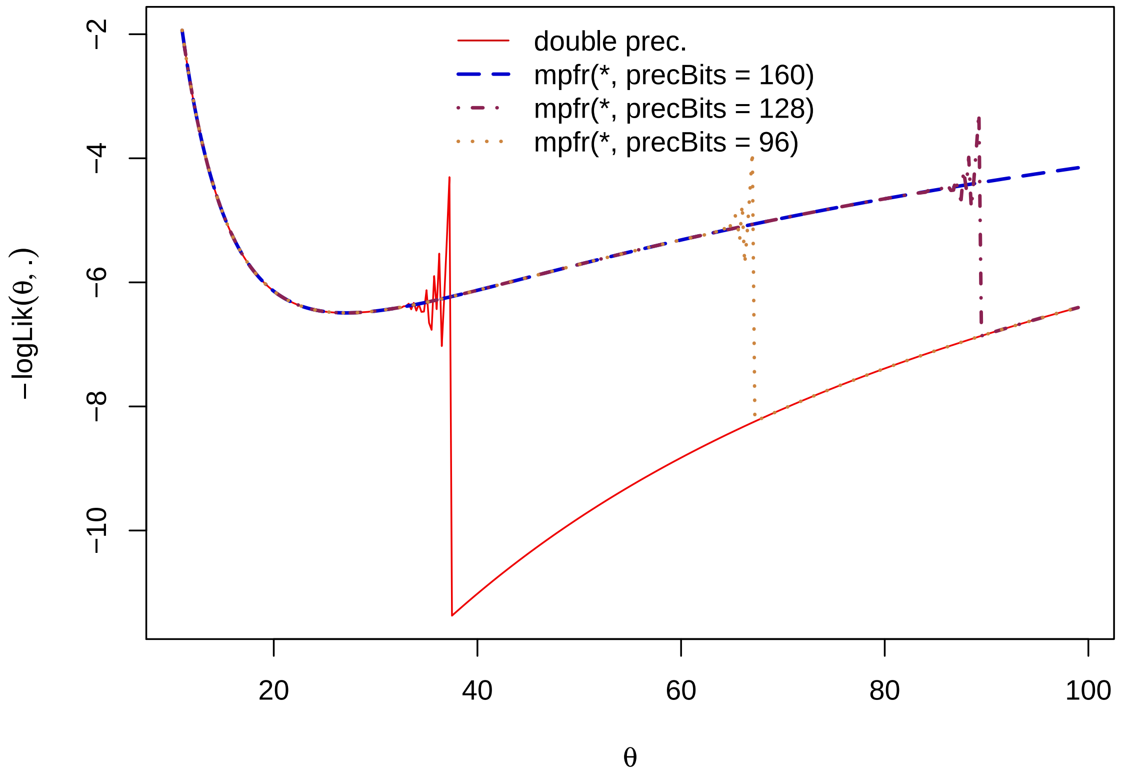
```
> d <- 5
> (theta <- copFrank@tauInv(tau = 0.75))

[1] 14.1385

> 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, psiInv = psiInv.2,
                  psiDabs=psiDabs.2, psiInvD1abs=psiInvD1abs.1,
                  log = TRUE))
```

Now, plot this negative log likelihood function in an interval θ, close to what is proposed **nacopula**'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 ...
```

```
Loading C code of R package 'Rmpfr': GMP using 64 bits per limb
```

```
> ## using three different precisions:
> MPrecBits <- c(160, 128, 96)
> mkNm <- function(bits) sprintf("%03d.bits", bits)
> ## As it takes a while seconds, cache the result:
> fnam <- sprintf("mlogL_mpfr_%s.rda", Sys.info()[["machine"]])
> if (!file.exists(fn <- file.path(copDDir,fnam))) {
    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
      })
    )) ## 91.226 0.013 91.506 [nb-mm icore 5]
      names(nllMP) <- mkNm(MPrecBits)
      save(nllMP, file = file.path(copSrcDDir, fnam))
  } else load(fn)
> 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's 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

```
    psiDabs(d*psiInv(u,th), th, log = TRUE) +
     psiInvD1abs(u,          th, log = TRUE)
```

currently, with the three functions

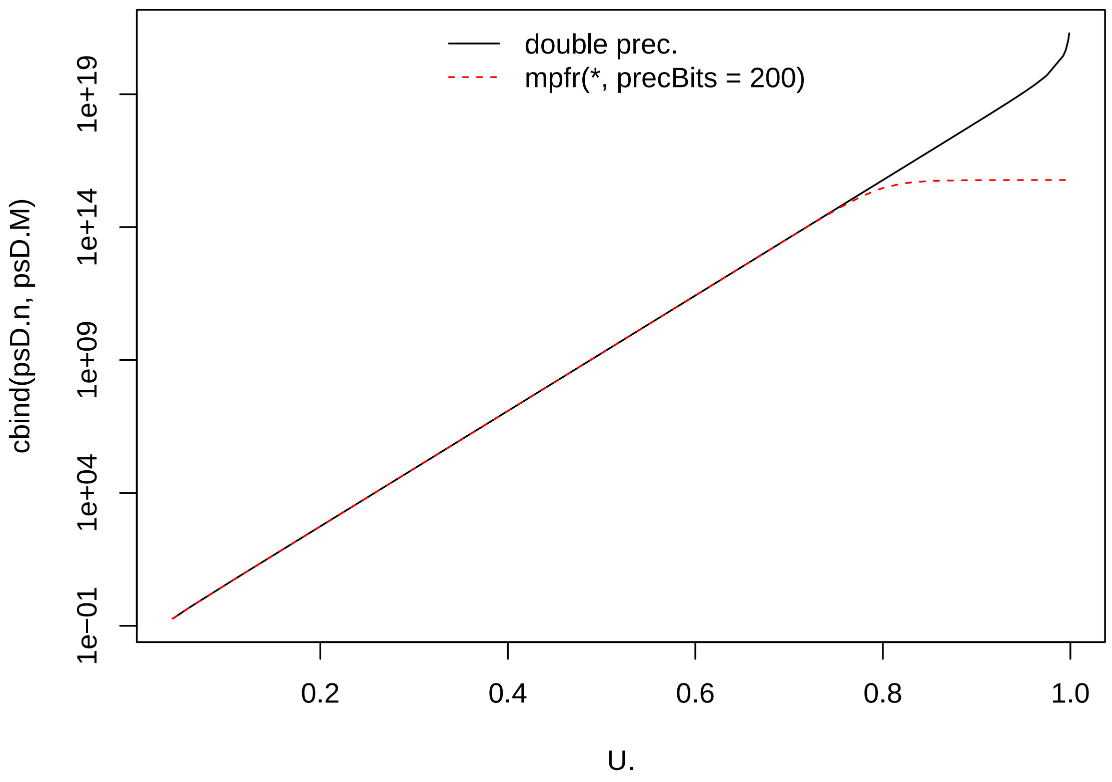
```
 psiInv = psiInv.2; psiDabs = psiDabs.2; psiInvD1abs = psiInvD1abs.1
```

where we have already tried to ensure that the `psiInv()` 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 `psiInv.0()` function instead the more stable `psiInv.2()` one:

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

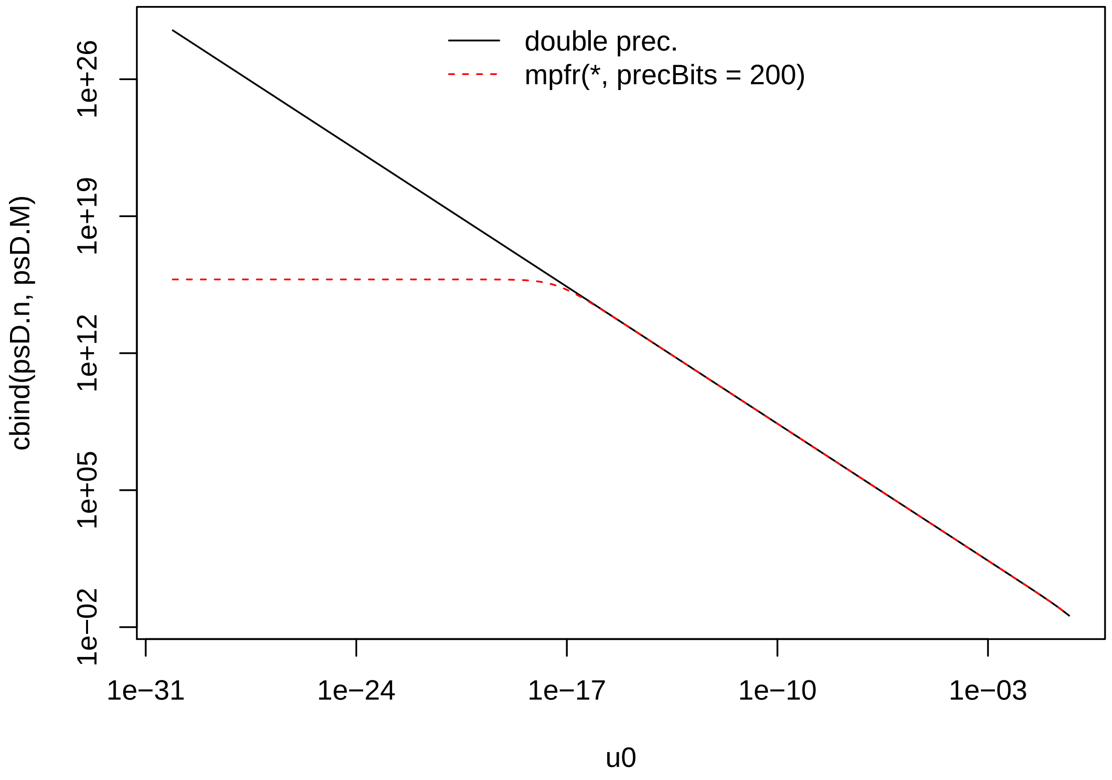
However, we can observe dramatic differences in `psiDabs.2()` ($= |\psi'(.)|$):

```
> psD.n <-             psiDabs.2(as.numeric(pI.U), 40)
> psD.M <- as.numeric(psiDabs.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= psiInv.2(U., theta)`$= \psi_\theta^{-1}(u)$.

```
> u0 <- 2^-(100:1)
> psD.n <-            psiDabs.2(u0, 40)
> psD.M <- as.numeric(psiDabs.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 `psiDabs.2()` which underflows for large theta, and hence should be replaced by the generally accurate

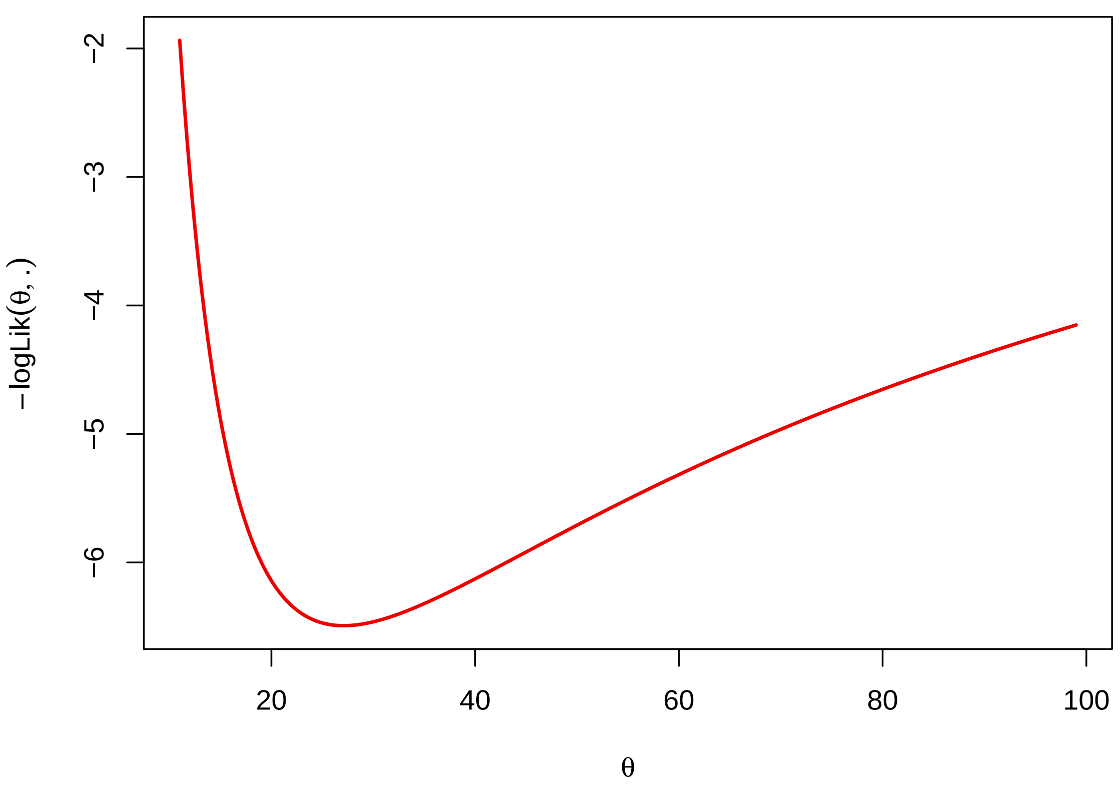
```
> log1mexpm <- function(a)
  {
      stopifnot(a >= 0)
      r <- a
      tst <- a <= log(2)
      r[ tst] <- log(-expm1(-a[ tst]))
      r[!tst] <- log1p(-exp(-a[!tst]))
      r
  }
```

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

```
> psiDabs.3 <- function(t, theta, log=FALSE) {
    w <- log1mexpm(theta) - t
    Li. <- if(log) w - log1mexpm(-w) else -exp(w)/expm1(w)
    if(log) Li. - log(theta) else Li. / theta
  }
```

Does this already solve the "diagonal likelihood" problem? Let's see, using a

```
> p.mlogL(th = seq(11, 99, by = 1/4),
          mlogL = (mlogL2 <- function(theta)
                   -sum(dDiagA(U., theta, d=d, psiInv = psiInv.2,
                               psiDabs = psiDabs.3, psiInvD1abs=psiInvD1abs.1,
                               log = TRUE))), lwd = 2)
```



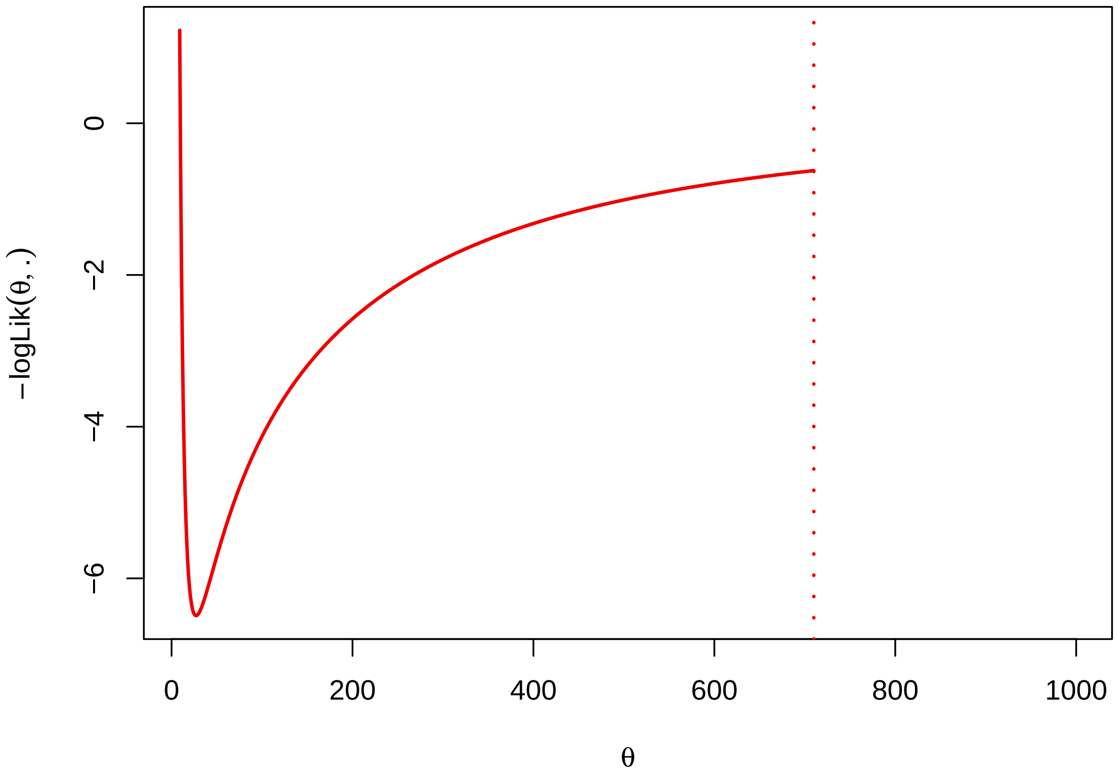
Yes, indeed, using a numerically stable version for `psiDabs()` did solve the numerical problem of computing, the "diagonal likelihood", and hence the `dmle()` ("diagonal MLE").

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, psiInv = psiInv.2, psiDabs = psiDabs.3,
                             psiInvD1abs = psiInvD1abs.1, log = TRUE)

[1] -Inf
```

which after closer inspection is from the `psiInvD1abs(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`:

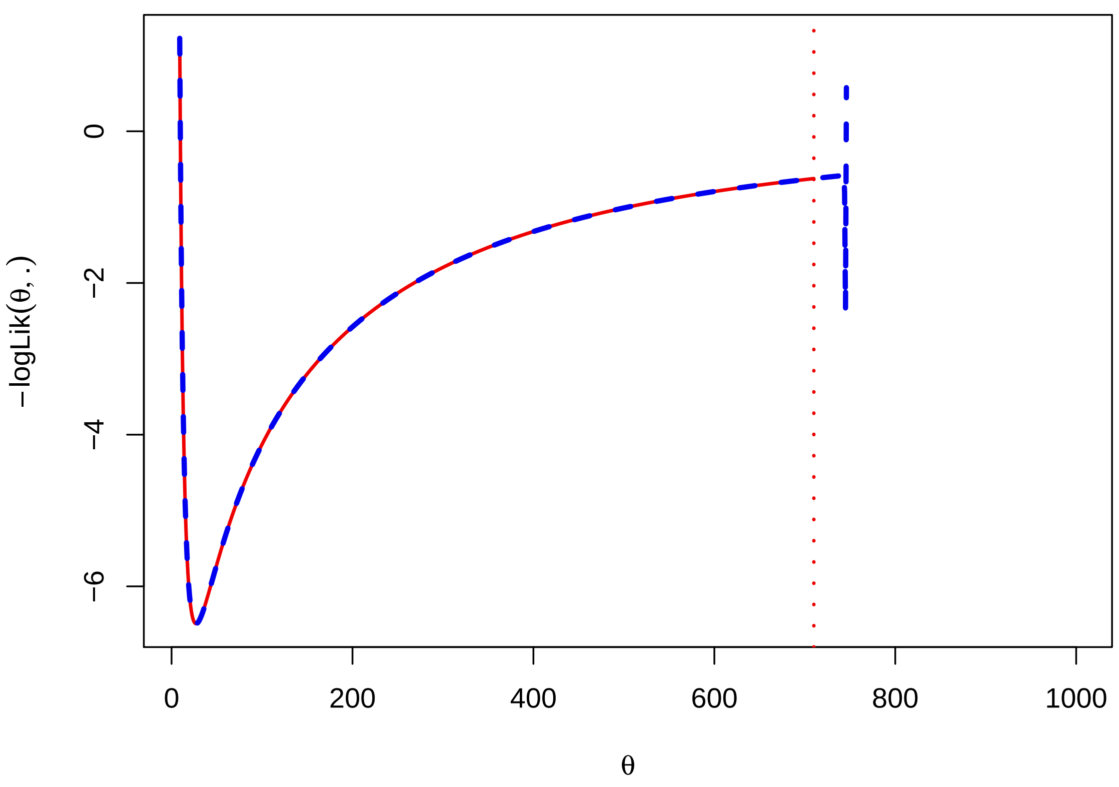
```
> psiInvD1abs.1(0.999, th = 715, log=TRUE)

[1] -Inf
```

However, as $\log(\mathrm{expm1}(y)) = \log(e^y - 1) = \log(e^y(1-e^{-y})) = y + \log(1-e^{-y})$, the "numerical stable" solution is to replace `log(expm1(y))` by `y + log1mexpm(y)`, such that we will use

```
> psiInvD1abs.2 <- function(u, theta, log = FALSE)
    if(log) log(theta)- {y <- u*theta; y + log1mexpm(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, psiInv= psiInv.2, psiDabs= psiDabs.3,
                               psiInvD1abs = psiInvD1abs.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 `psiDabs(d*psiInv(u,th), th, log = TRUE)`, as `psiInv(u,th) = psiInv(0.999, th=800)`=0 underflows to zero — by necessity: the correct value is smaller than the smallest representable double precision number:

```
> pI <- psiInv.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 `psiDabs()`, i.e., we start computing `log(psiInv(.))` directly (evading the underflow to 0 there).

. . . . . . to be continued.

3. Session Information

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

- R Under development (unstable) (2011-09-22 r57046), `x86_64-unknown-linux-gnu`
- Locale: `LC_CTYPE=C`, `LC_NUMERIC=C`, `LC_TIME=en_US.UTF-8`, `LC_COLLATE=C`, `LC_MONETARY=en_US.UTF-8`, `LC_MESSAGES=C`, `LC_PAPER=C`, `LC_NAME=C`, `LC_ADDRESS=C`, `LC_TELEPHONE=C`, `LC_MEASUREMENT=en_US.UTF-8`, `LC_IDENTIFICATION=C`
- Base packages: base, datasets, grDevices, graphics, methods, stats, stats4, utils
- Other packages: Rmpfr 0.4-3, nacopula 0.7-9-1

- Loaded via a namespace (and not attached): ADGofTest 0.1, gsl 1.9-9, stabledist 0.6-3, tools 2.14.0

4. Conclusion

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`