Asymptotic Tail Formulas For Gaussian Quantiles in R

Martin Mächler
Seminar für Statistik, ETH Zurich
Nov. 2022 LaTeX'ed December 1, 2022

Abstract

R's Gaussian quantile function `qnorm(p, ..)` has been based on the published algorithm AS 241 of Wichura (1988) which is fully accurate only on the regular scale for p down to the smallest double precision numbers > 0. When probabilities are used on the log scale, i.e., `qnorm(lp, log.p=TRUE)`, the argument is a log probability, and `lp`$= \log p \to -\infty$ when $p \to 0$, `qnorm()` using AS 241 has been very inaccurate in the very extreme tails.

We have derived asymptotic formulas for that case, using recursive plug-in to the asymptotic formula for $\Phi(x)$ (which `qnorm()` should invert).

Using these formulas for order $k = 0, 1, \ldots, 5$, for six different regions (adjacent intervals) allows to provide fully accurate `qnorm()` computations also on the log-scale. Pure R implementations of these are provided in our R package **DPQ**, functions `qnormAsymp()` and `qnormR()` and have also been prepared to be added to (the C code in `Rmathlib` in) the next version of R's `qnorm()`.

Keywords: asymptotic, approximation, extreme tail, Gaussian, Normal Quantile.

1. Gaussian Quantiles in R – okay on regular probability range

Gaussian or normal quantiles have been made available in R from the very beginning. Ross Ihaka (one of the two "fathers" of R) wrote the first version; visible in R's subversion (svn) repository, rev 574, dated Jan. 14, 1998 basically interfacing R with a C version of the published AS 111 algorithm (which was in Fortran 66 with `GOTO` etc), Beasley and Springer (1977), but improving AS 111 already by using a more accurate formula from Wichura for the "outer" tail (defined to have $p' := min(p, 1-p)$ close to zero, specifically, when $p' \in (10^{-300}, \epsilon_c]$, where ϵ_c, the computer epsilon, (= `DBL_EPSILON` in C's `math` library = R's `.Machine$double.eps` is nowadays always $\epsilon_c = 2^{-52} = 2.220446.. \cdot 10^{-16}$.

This first algorithm AS 111, e.g. in R 1.0.0, Feb.29, 2000, (svn rev 7639, 2000-01-18), the version of `<R>/src/nmath/qnorm.c` had contained the description

> *Compute the quantile function for the normal distribution.*
>
> *For small to moderate probabilities, algorithm referenced below is used to obtain an initial approximation which is polished with a final Newton step.*
>
> *For very large arguments, an algorithm of Wichura is used.*

and the reference to Beasley and Springer (1977).

Also, already before releasing R 1.0.0 on Feb. 29, 2000, we had introduced the `log.p` and `lower.tail` logical switches,

> r7615 | maechler | 2000-01-17 12:18:30 +0100 (Mo, 17 Jan 2000)
> add new argument lower.tail and log[p]; at first only to [dpq]pois()

Already a few months after releasing R 1.0.0 (June 6, svn r9464), I had switched `qnorm()` to use the more recent and accurate AS 241 with NEWS entry

```
o  qnorm() is now based on AS 241 instead of AS 111, and should
   give precise results up to 16 digits precision.
```

Algorithm AS 241 is by Wichura (1988) which contains the promise of 16 digits precision[1], the last sentence on p.477: ... *for* $10^{-316} < \min(p, 1-p)$. *The second routine, PPND16, is accurate to about 16 figures over the same range.*

Also in Wichura (1988),

$$r := \sqrt{-\log(\min(p, 1-p))} \quad (\Longleftrightarrow \min(p, 1-p) = e^{-r^2}). \tag{1}$$

For ease of notation, we assume $p < \frac{1}{2}$, for now, and hence the quantile $\texttt{qnorm(p)} = \Phi^{-1}(p) = \Phi^{-1}(\exp(-r^2))$ is negative. The "outermost" minimax rational approximation to $-\Phi^{-1}(p)$ used in AS 241 is in the interval $r \in (5, 27] \Longleftrightarrow r^2 \in (25, 729]$, or equivalently,

$$p \in [e^{-729}, e^{-25}) \approx [2.51 \cdot 10^{-317}, 1.389 \cdot 10^{-11}). \tag{2}$$

At first, the above seems sufficient, since indeed, the lower bound is already "de-normalized" in double precision, $e^{-27^2} = e^{-729} \approx 2.51 \cdot 10^{-317}$ is smaller than `DBL_XMIN` in C's `math` library = R's `.Machine$double.xmin`$= 2^{-1022} \approx 2.225 \cdot 10^{-308}$.

However, as mentioned above, in the R core team we had already seen that it is often advisable to work on the log–scale with probabilities. For that reason we had introduced the option `log.p = TRUE` for all our (cumulative) distribution and quantile functions. Now this changes the picture of "sufficient" approximation dramatically, as, indeed, on the log scale, the AS 241 algorithm only goes up to $\log p = r^2 = 729$, and then quickly loses precision (see below).

The goal of the remaining part of this paper is to describe our research for finding accurate approximations in these outermost tails.

1.1. DPQ's `qnormR()` documenting history

Note that in our **DPQ**, we do provide pure R code implementations of R's `qnorm()` in function `qnormR()` which has (almost[2]) the same arguments `p, mu = 0, sd = 1, lower.tail = TRUE, log.p = FALSE` as R's `qnorm()` and additionally `trace = 0, version = c("4.0.x", "2020-10-17", "2022-08-04")`, where the default `version = "4.0.x"` corresponds to R version up to 4.0.5 (2021-03-31) which uses basically the above AS 241, additionally treating extreme cases including $\pm$`Inf` and `NA, NaN` well. The newer versions are explained subsequently.

[1] 16 digits precision, i.e., about the usual IEEE 52-bit double precision ($\epsilon_c = 2^{-52} \approx 2.22 \cdot 10^{-16}$)

[2] 'mu' $\neq$ 'mean'

2. Accurate `qnorm(.., log.p=TRUE)`

In order to compare versions of `qnorm()` approximations with their "true" values, we use the fact that it, $x = \Phi^{-1}(p)$ =`qnorm(p)`, is defined as *inverse* of $p = \Phi(x)$ =`pnorm(x)` *and* we additionally assume that `pnorm(x)` is "fully accurate" which it basically is, also on the log–scale, demonstrably, e.g., using our CRAN pkg **Rmpfr** with its own very accurate `pnorm()`, but we are not providing the evidence here.

With this assumption, the error of `qnorm()` is the deviation from the identity $\Phi^{-1}(\Phi(x)) \equiv x$. If $x \neq 0$, the *relative* error is

$$\frac{\widehat{\Phi^{-1}}(\Phi(x)) - x}{x} = \widehat{\Phi^{-1}}(\Phi(x))/x - 1, \tag{3}$$

and we "define" the relative error of `qnorm()` as `qnorm(pnorm(x)) / x - 1` where we need to adjust for cases where x is (very close to) zero or not finite, etc. This is done by function `relErrV()` from package **sfsmisc**, shown in the appendix A, which takes care of all special or boundary cases.

And as a matter of fact, we will work in log-scale, hence using `log.p = TRUE` in both `pnorm()` and `qnorm()`, and we want to use positive numbers both for argument and result (and nicer formulae), so work with the *upper tail*, i.e., use `lower.tail = FALSE`. Consequently, instead of computing and inverting $\Phi(x)$, i.e., our `qnorm(.)` should compute the inverse of $\log(1 - \Phi(x))$.

```
> qs <- 2^seq( 0, 29, by=1/256) # => s >= 1.84
> lp <- pnorm(qs, lower.tail=FALSE, log.p=TRUE)
> s <- -lp # = -pnorm(..) = -log(1 - Phi(qs)) > 0
> require("DPQ") # -->  qnormR():
> qnrm    <- qnorm (-s, lower.tail=FALSE, log.p=TRUE)
> qnrm405 <- qnormR(-s, lower.tail=FALSE, log.p=TRUE, version= "4.0.x") # R <= 4.0.5
> qnrm410 <- qnormR(-s, lower.tail=FALSE, log.p=TRUE, version= "2020-10-17")
> Rver <- sfsmisc::shortRversion()
> if(getRversion() <= "4.0.5") { # our qnormR(.., version="4.0.x")
      cat(sprintf("%s, \"4.0.5\",\n   all.equal(*, tol=0): %s;  identical(): %s\n", Rver,
                  all.equal(qnrm, qnrm405, tolerance=0), identical(qnrm, qnrm405)))
      stopifnot(all.equal(qnrm, qnrm405, tolerance = 1e-12))
  } else if(getRversion() < "4.3") { # our qnormR(*, version="2020-10-17") matches:
      cat(sprintf("%s, \"4.1.0\",\n   all.equal(*, tol=0): %s;  identical(): %s\n", Rver,
                  all.equal(qnrm, qnrm410, tolerance=0), identical(qnrm, qnrm410)))
      stopifnot(all.equal(qnrm, qnrm410, tolerance = 1e-12))
  } else { # R version >= 4.3.x
      qnrm43 <- qnormR(-s, lower.tail=FALSE, log.p=TRUE, version = "2022")
      cat(sprintf("%s, >= 4.3.x,\n   all.equal(*, tol=0): %s;  identical(): %s\n", Rver,
                  all.equal(qnrm, qnrm43, tolerance=0), identical(qnrm, qnrm43)))
      rE6 <- qnorm(-1e6, log.p=TRUE)/-1414.2077829910174  - 1
      cat(sprintf("  rE(-1e6) = %g\n", rE6))
      if(abs(rE6) < 7e-16) # have R-devel with new 2022 code:
          stopifnot(all.equal(qnrm, qnrm43, tolerance = 1e-14))
  }

R 4.2.2 Patched 2022-11-23 r83388, "4.1.0",
   all.equal(*, tol=0): TRUE;  identical(): TRUE
```

Computing a version of the above (with larger range for s, starting from `qs <- 2^seq(0, 70, by=1/8)`) in R version 4.0.5 and plotting in log-log scale,

```
> plot(qnrm405 ~ s, type="l", log="xy", col=2, ylim = c(1, max(qs)), asp = 1,
       xaxt="n", yaxt="n"); require("sfsmisc"); eaxis(1); eaxis(2)
> lines(qs ~ s, col=(c4 <- adjustcolor(4, 1/4)), lwd=4)
> legend("top", c("qnorm(-s, lower.tail=FALSE, log.p=TRUE)", "true"),
         col=c(palette()[2], c4), lwd=c(1,4), bty="n")
```

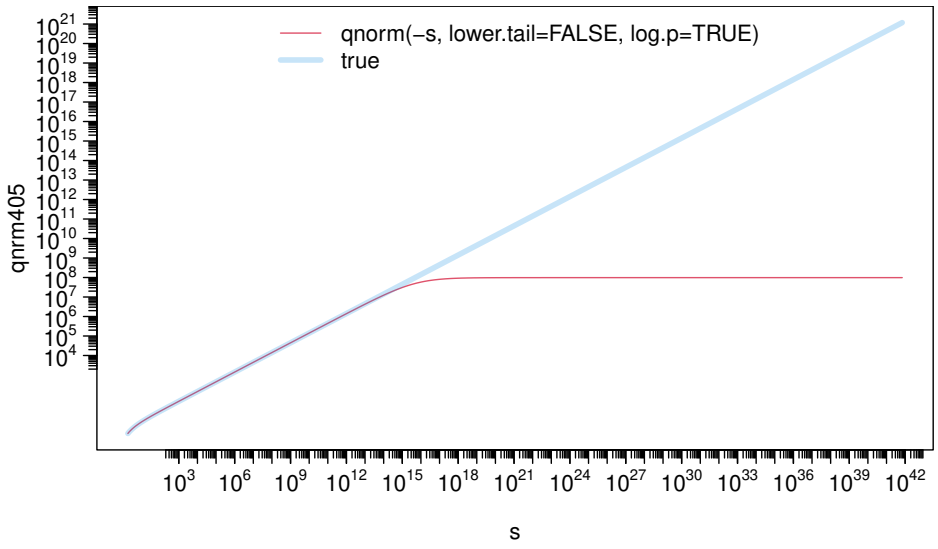


Figure 1: Extreme tail log-scale `qnorm(-s, ..)` in R 4.0.5 or earlier, i.e., up to 2021

looks good up to about 10^{12}, i.e., `qnorm()` coinciding with the true x `qs`, but beyond 10^{14} diverging for larger s, and for even larger s showing complete loss of accuracy as `qnorm(-s, *)` converges (to 98340296.6), even though the true function should go to $+\infty$. We will see that indeed, asymptotically, `qnorm`$(|s|, ..) \sim \sqrt{2|s|}$ which in log-log scale is a line (with intercept $\log\sqrt{2}$ and slope $1/2$).

Closer inspection, showing the relative errors in Figure 2 :

```
> if(!exists("version.txt"))  version.txt <- R.version.string
> ablaxis1 <- function(x) { abline(v = x^2, col=4, lty=2)
      axis(1, at=x^2, labels = substitute(X^2, list(X=x)), col=4, col.axis=4, line=-.61, cex=0.25)
> plot(abs(relE_qn) ~ s, type="l", log="xy",
       main = "inaccuracy of qnorm(-s, log.p=TRUE, lower.tail=F)", axes=FALSE)
> eaxis(1, nintLog = 13, sub10 = 2); eaxis(2); ablaxis1(x=27)
> mtext(version.txt, line = -0.8, cex=.8, adj = 0.75)
```

for

```
> relE_qn <- relErrV(qs, qnrm405) ; version.txt <- "R versions up to R 4.0.5"
```

From this, in September 2020, I started to investigate the visually obvious asymptotic behavior of the *correct* inversion of `qnorm(.)`, using the classical first order asymptotic

$$1 - \Phi(x) \sim \frac{\phi(x)}{x}, \quad \text{for } x \to \infty, \tag{4}$$

for the standard normal / Gaussian density $\phi(x) := \frac{1}{\sqrt{2\pi}} e^{-x^2/2}$ and cumulative distribution

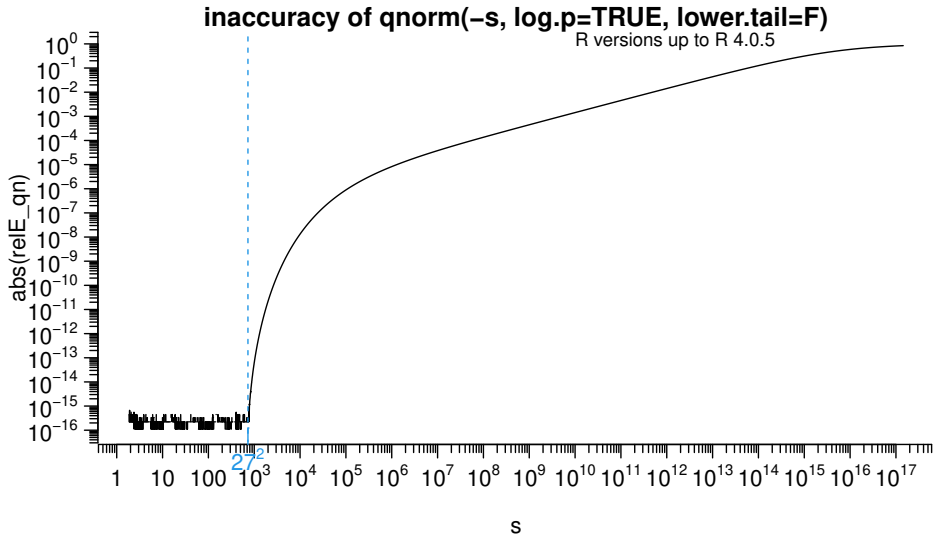


Figure 2: Relative error of qnorm() in extreme tails in R version before R 4.1.0

function $\Phi(x) := \int_{-\infty}^{x} \phi(t)\, dt$. On the log scale, this is equivalent to

$$\begin{aligned} \log(1 - \Phi(x)) &= \log \phi(x) - \log x + o(x), \\ &= -x^2/2 - 1/2 \log 2\pi - \log x + o(x) \\ &= -x^2/2 + o(x), \end{aligned} \tag{5}$$

i.e. $l_p := \log(1 - \Phi(x)) \approx -x^2/2$ for large x and hence,

$$x \approx \sqrt{-2l_p}, \tag{6}$$

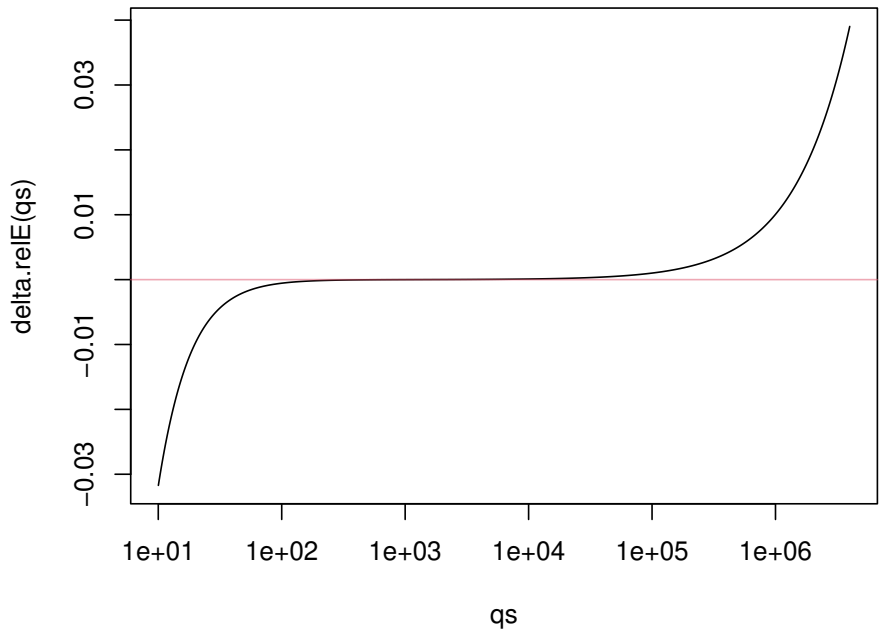
for large $|x|$ or large $|l_p| = -l_p =: s$ (using notation as in the R code above with `lp` and `s <- -lp`).

Consequently, a first order remedy against the "catastrophic" precision loss for extreme tail `qnorm()` was to use the above $\sqrt{2s}$ approximation for upper tail probabilities specified in log scale.

Computer experimentation was used to find a numerically optimal (for the double precision implementation of AS 241) cutoff.

Computing the absolute value of the relative error, (in R 4.0.x, *not* current R):

```
> delta.relE <- function(q, qNorm = function(...) qnormR(..., version = "4.0.x")) {
     lp <- pnorm(q, lower.tail=FALSE, log.p=TRUE) # <==>  q = true qnorm(lp, *)
     ## the "delta" of the two relative errors qnorm() vs sqrt(2*s) approx:
     abs(1 - qNorm(lp, lower.tail=FALSE, log.p=TRUE) / q) -
     abs(1 - sqrt(-2*lp) / q)
   }
> plot(delta.relE(qs) ~ qs, subset = 10 < qs & qs < 4e6, type="l", log="x")
> abline(h=0, col = adjustcolor(2, 1/2))
```



As this looks good, let us find the root location which then is the optimal cutoff, as it will minimize the absolute value of the relative error of computing `qnorm(..)`. At first:

```
> cutP. <- uniroot(function(logq) delta.relE(exp(logq)) , c(3, 13))
> exp(cutP.$root)

[1] 1153.223
```

then, getting more accurate once we approximately know the region:

```
> str(cP. <- uniroot(delta.relE, interval = c(1000, 1300), tol = 1e-12))

List of 5
 $ root      : num 1153
 $ f.root    : num 0
 $ iter      : int 7
 $ init.it   : int NA
 $ estim.prec: num 2.43e-09

> qC <- cP.$root # 1153.242
> (lpC <- pnorm(qC, lower.tail=FALSE, log.p=TRUE))

[1] -664991
```

so the optimal cutoff where to use the sqrt-approximation is at `lp`$= \log p = -664991$ or $r = \sqrt{-\log p} = 815.470$ (with r defined in (1)), and for convenience (round number), using the cutoff $r \geq 816$ in `qnorm()`, i.e., basically

```
  if(r >= 816)  value = sqrt(2) * r;
```

This consequently was added to the R (i.e., "R-devel") sources after more testing, a few weeks later

```
svn r79346 | maechler | 2020-10-17 21:42:17 +0200
```

to be in R 4.1.0 with NEWS entry

- `qnorm(<very large negative>, log.p=TRUE)` is now correct to at least five digits where it was catastrophically wrong, previously.

Indeed, `qnorm()` was now "first order accurate" even in the extreme tails, and in a plot such as Figure 1 one would not notice any inaccuracy. But then, there you'd visually only notice deviations in the order of 1 %, i.e, already visible in 2 digits accuracy. Looking at the relative errors directly, indeed shows the relative error being smaller than 10^{-5} and maximal at the cutoff $s = 816^2 = 665856$:

```
> relE_qn <- relErrV(qs, qnrm410); version.txt <- "R 4.1.0 to 4.2.x"
```

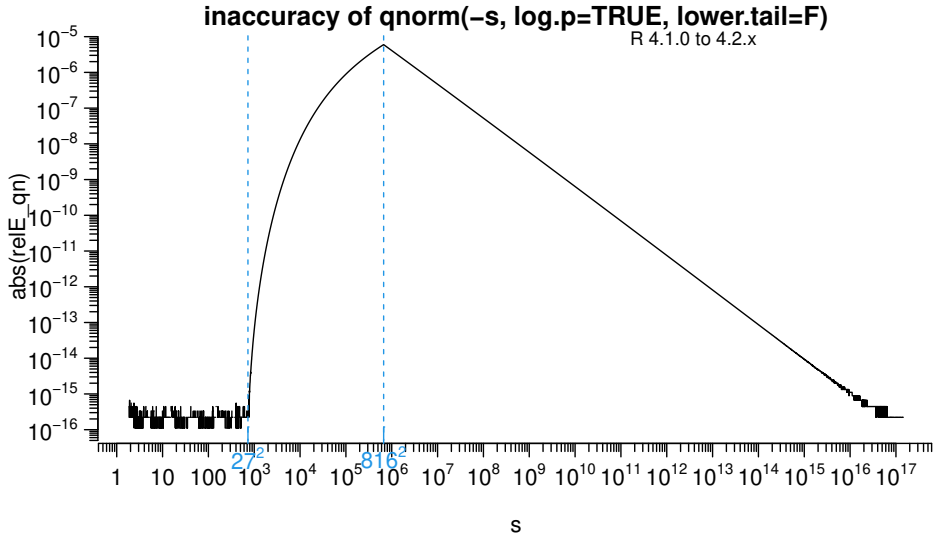


Figure 3: qnorm() relative error in extreme tails, R ver. 4.1.0–4.2.x, ca. 2021–'22

3. More accurate asymptotic `qnorm(., log.p=TRUE)`

In R, the function `qnorm()` and notably the underlying C API function `qnorm()`[3] are used in other places, not the least also to (approximately) compute quantiles of other distributions, such as the (non central) t.

In addition, $\Phi^{-1}(x)$ is a very smooth monotone function, it may naturally be desirable that `qnorm()` computes its values to the same full (double precision) accuracy as most other mathematically well defined functions in R.

Now, the classical simple first order asymptotic (4) for $\Phi(.)$, i.e., R's `pnorm()`, has been known to many more terms, also for a long time. Mills (1926) builds already on work by Laplace, said to have derived some of the two asymptotic series, in Abramowitz and Stegun (1972)[p. 932],

A. & S. (26.2.12).

$$1 - \Phi(x) = \Phi(-x) = \frac{\phi(x)}{x} \cdot \left(1 - \frac{1}{x^2} + \frac{1 \cdot 3}{x^4} + \cdots + \frac{(-1)^n \, 1 \cdot 3 \cdots (2n-1)}{x^{2n}}\right) + R_n, \quad (7)$$

[3] R's C API `qnorm()` is an alias for `qnorm5()` in the source file `<R>/src/nmath/qnorm.c`

where the remainder term R_n (which can be represented exactly as an integral) is smaller than the first neglected term. Note that A.&S. use notation $Q(x) \equiv 1-\Phi(x)$ and $Z(x) \equiv \phi(x)$, also in the subsequent asymptotic series which is slightly more accurate numerically (but without an explicit remainder term):

A. & S. (26.2.13).

$$1-\Phi(x) \sim \frac{\phi(x)}{x}\cdot\left(1-\frac{a_1}{x^2+2}+\frac{a_2}{(x^2+2)(x^2+4)}-\frac{a_3}{(x^2+2)(x^2+4)(x^2+6)}+\cdots\right), \quad (8)$$
$$\text{where } a_1 = a_2 = 1, a_3 = 5, a_4 = 9, a_5 = 129,$$

and the general coefficient a_n is defined via coefficients of a polynom expansion.

As previously, we need to use this in log-scale,

$$\begin{aligned} l_p = \log(1-\Phi(x)) &\approx \log(\phi(x)) - \log(x) + \log(1-q(x^2)), \\ &= -\frac{x^2}{2} - \frac{1}{2}\log(2\pi) - \log(x) + \log(1-q(x^2)), \quad (9) \\ \text{where } q(x^2) &= \frac{1}{x^2+2} - \frac{1}{(x^2+2)(x^2+4)} + \frac{5}{(x^2+2)(x^2+4)(x^2+6)} + \cdots, \\ &= \frac{1}{x^2+2}\Big(1-\frac{1}{x^2+4}\Big(1-\frac{1}{x^2+6}\Big(5-\frac{9}{x^2+8}+\cdots\Big)\Big)\Big), \quad (10) \end{aligned}$$

and l_p is the log probability (given as first argument to `qnorm()`), and we would like to *solve* for x, as in the simple 1^{st} order case in (5) and (6) above, but of course that is not possible. However, an amazingly versatile idea of recursive "plug-in" will work here: For the first step, we may neglect $q(x^2) \approx 1/(x^2+2)$ entirely as we know that $x^2 \approx 2s$ for relatively large s, and hence drop $\log(1-q(x^2)) \approx \log(1) = 0$, such that (-2) times eq. (9) becomes

$$\begin{aligned} -2l_p = 2s \approx x^2 + \log(2\pi) + 2\log(x) = \\ x^2 + \log(2\pi\, x^2), \end{aligned} \quad (11)$$

now subtracting the log term *and* replacing *its* x^2 by its asymptotic approximation $x_0^2 = 2s$ gives

$$2s - \log(2\pi\, 2s) \approx x^2, \quad \text{or, with}$$
$$x_0^2 := 2s, \quad (12)$$
$$x^2 \approx x_1^2 := 2s - \log(2\pi\, x_0^2) = 2s - \log(4\pi s), \quad (13)$$

and we do have a substantially better approximation, verified empirically in Fig. 4 below ($k = 0$ vs $k = 1$), where we show further steps, continuing our selective recursive plug-in of x^2 itself, now no longer neglecting $q(x^2)$ but still only using a first term, from (9),

$$\begin{aligned} -2\log(1-\Phi(x)) = 2s \approx x^2 + \log(2\pi\, x^2) - 2\log(1-q(x^2)) \approx \\ x^2 + \log(2\pi\, x^2) + 2q(x^2), \end{aligned} \quad (14)$$

where the 2nd "≈" is from $\log(1-q) \approx -q$ for $|q| \ll 1$ and $q(x^2) \approx 1/(x^2+2)$ is assumed to be very small here. Again solving for the first x^2 and replacing the other x^2 by our current best approximation x_1^2 leads to

$$x^2 \approx x_2^2 := 2s - \log(2\pi\, x_1^2) - 2/(x_1^2+2), \quad (15)$$

and continuing recursively, always taking one more term for $q(x^2)$, but no longer replacing $\log(1-q)$ by $-q$ but rather the fully accurate $\mathrm{log1p}(-q)$,

$$x^2 \approx x_3^2 := 2s - \log(2\pi\, x_2^2) + 2\,\mathrm{log1p}(-(1 - 1/(4 + x_2^2))/(2 + x_2^2)), \quad \text{and} \quad x^2 \approx \tag{16}$$

$$x_4^2 := 2s - \log(2\pi\, x_3^2) + 2\,\mathrm{log1p}(-(1 - (1 - 5/(6 + x_3^2))/(4 + x_3^2))/(2 + x_3^2)), \quad \text{and} \tag{17}$$

$$x_5^2 := 2s - \log(2\pi\, x_4^2) + 2\,\mathrm{log1p}(-(1 - (1 - (5 - 9/(8 + x_4^2))/(6 + x_4^2))/(4 + x_4^2))/(2 + x_4^2)). \tag{18}$$

Taking the square roots of these 6 approximations for x^2 for the inverse cumulative normal, $\Phi^{-1}(e^{-s})$, namely

$$\begin{aligned}
x_0(s) &= \sqrt{2s}, \quad \text{from (12)} \\
x_1(s) &= \sqrt{2s - \log(4\pi s)}, \quad \text{from (13)} \\
x_2(s) &= \sqrt{2s - \log(2\pi\, x_1^2) - 2/(x_1^2 + 2)}, \quad \text{from (15)} \\
x_3(s) &= \sqrt{2s - \log(2\pi\, x_2^2) + 2\,\mathrm{log1p}(r(x_2))}, \quad \text{see (16)} \\
x_4(s) &= \ldots\ldots, \quad x_5(s) = \ldots\ldots, \quad \text{see (17), (18).}
\end{aligned}$$

These $x_k(s)$ are provided as plain R function `qnormAsymp()`, in our **DPQ** package, specifically, $x_k(s) =$ `qnormAsymp(lp = -s, order = k)`[4]

```
> k.s <- 0:5; nks <- paste0("k=", k.s)
> qnAsym <- sapply(setNames(k.s, nks), function(k) qnormAsymp(lp=lp, order = k))
> relEasym <- apply(qnAsym, 2, relErrV, target = qs) # rel.errors for all
```

In Fig. 4 we depict the absolute values of their respective relative errors (in log-log scale against $s = -lp = -\log(1 - \Phi(x))$), and and then zoom in more closely in Figure 5:

[4] which is the short form; indeed, `qnormAsymp(lp = lp, order = k)` is identical to `qnormAsymp(p = lp, lower.tail=FALSE, log.p=TRUE, order = k)` .

```
> matplot(-lp, abs(relEasym), log="xy", type="l", lwd=2, axes=FALSE, xlab = quote(s == -lp))
> eaxis(1, sub10=2); eaxis(2, sub10=c(-2,2), nintLog=16); grid(col="gray75")
> legend("right", nks, col=1:6, lty=1:5, lwd=2, bty="n")
```

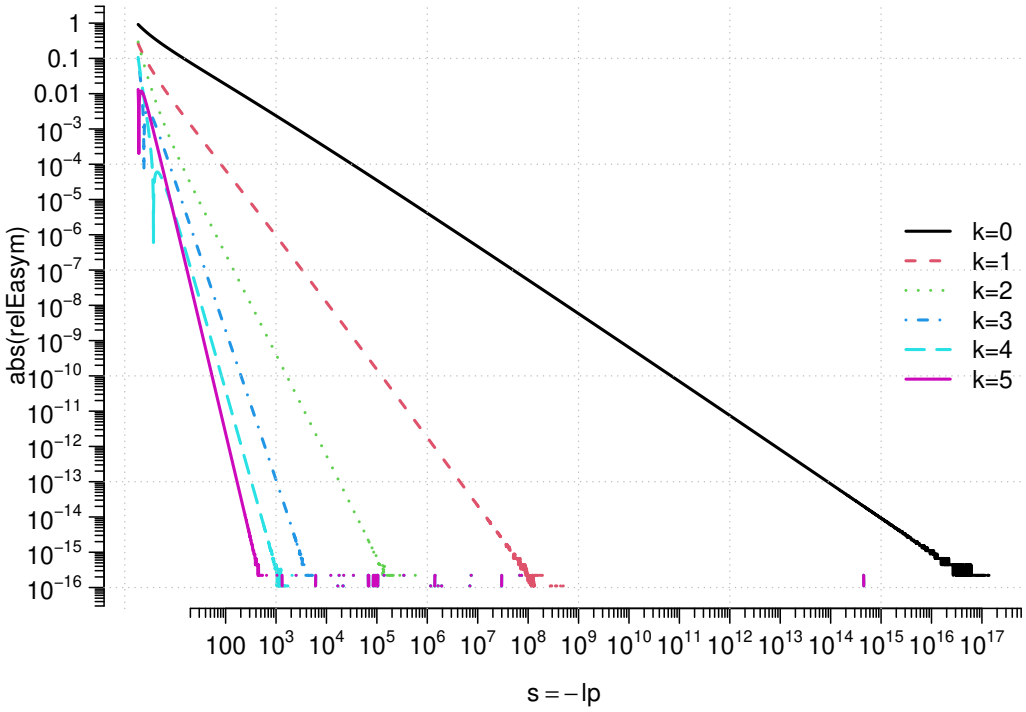


Figure 4: |relative errors| of asymptotic approximations in log-log scale

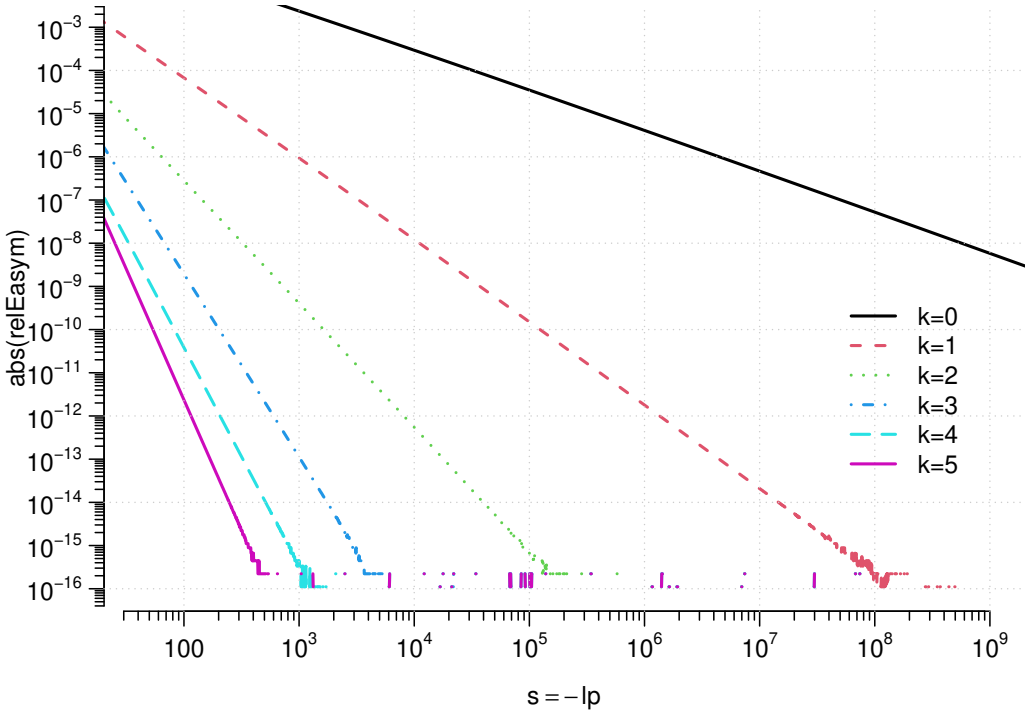


Figure 5: (Zoomed Fig. 4) |relative errors| of asymptotic approx. $x_0(s)$, $x_1(s)$, ..., $x_5(s)$

Fully accurate qnorm()

Our (package **DPQ**) `qnormR(... version = "2022-08")` now implements a (pure R implementation) also to be used in the next version of R, which uses "the optimal" asymptotic approximation $x_k(s)$ for $s = r^2 > 27^2$ and $k \in \{0, 1, \ldots, 5\}$ as defined above in (12)–(18). "Optimal" is defined as the smallest k which still provides full accuracy, e.g., when $s > 10^{18}$ clearly, $x_0(s) = \sqrt{2s}$ is sufficient and hence optimal in that sense.

Consequently, we have determined ("round number", approximate) **optimal cut points / regions for different approximation orders** k and found the following "round number" values,

k	5	4	3	2	1	0
$r \geq$	27	55	109	840	36000	6.4e8
$s = r^2 \geq$	729	3025	11880	705600	$1296{\cdot}10^6$	4.096e17

Table 1: optimal cutpoints to determine k to use $x_k(s)$ for $r = \sqrt{s} > 27$.

e.g. $k = 0$ is fully accurate and hence optimal and used for $r \geq 6.4e8 = 640 \cdot 10^6$, or equivalently, for $s = -\mathrm{lp} \geq 4.096e17$, where as for $r \in [55, 109) \iff s \in [3025, 11880)$ one needs (and uses) $k = 4$. These were determined using function `p.qnormAsy2()` in appendix B, for visualizing the optimal region for switching from $k-1$ to k, for $k = 1, 2, 3, 4, 5$.

To see how the final `qnormR()` is implemented, you can look at the `length-1` version `qnormR1()`[5].

4. Concluding Summary

We have derived asymptotic formulas for `qnorm(lp, lower.tail=FALSE, log.p=TRUE)`, i.e. $\Phi^{-1}(e^s) = \Phi^{-1}(e^{r^2})$ for large $s = r^2$, notably for $r > 27$ which is beyond the range where the published algorithm AS 241, Wichura (1988), is accurate.

For these formulas of order $k = 0, 1, \ldots, 5$, implemented in **DPQ**'s R function `qnormAsymp(*, order=k)` we have derived optimal regions, i.e., intervals for r, partitioning $(27, \infty)$ and implemented in R function `qnormR(*, version = "2022-08")` for reproducibility and to be used in (the C code in `Rmathlib` in) the next version of R's `qnorm()`.

5. Session Information

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

- R version 4.2.2 Patched (2022-11-23 r83388), `x86_64-pc-linux-gnu`
- Running under: `Fedora Linux 36 (Thirty Six)`
- Matrix products: default
- BLAS: `/u/maechler/R/D/r-patched/F36-64-inst/lib/libRblas.so`

[5] indeed, in the package source file `DPQ/R/norm_f.R`, `qnormR()` is defined to correctly vectorize in its main arguments `p`, `mu`, and `sd`, by `qnormR <- Vectorize(qnormR1, c("p", "mu", "sd"))`

- LAPACK: `/u/maechler/R/D/r-patched/F36-64-inst/lib/libRlapack.so`
- Base packages: base, datasets, grDevices, graphics, methods, stats, utils
- Other packages: DPQ 0.5-3, sfsmisc 1.1-14
- Loaded via a namespace (and not attached): compiler 4.2.2, tools 4.2.2

```
> unlist(packageDescription("DPQ")[c("Package", "Version", "Date")])
```

```
     Package      Version         Date
       "DPQ"      "0.5-3" "2022-12-01"
```

A. Function `relErrV()` (package sfsmisc)

To compute relative (approximation) errors, in a way that works correctly, also with `Inf`, `NA`, and `NaN`s, we make use of the function `relErrV()` from (our own) CRAN package **sfsmisc**, defined[6] as

```
> ## Componentwise aka "Vectorized" relative error:
> ## Must not be NA/NaN unless one of the components is  ==> deal with {0, Inf, NA}
> relErrV <- function(target, current, eps0 = .Machine$double.xmin) {
      n <- length(target <- as.vector(target))
      ## assert( <length current> is multiple of <length target>) :
      lc <- length(current)
      if(!n) {
          if(!lc) return(numeric()) # everything length 0
          else stop("length(target) == 0 differing from length(current)")
      } else if(!lc)
          stop("length(current) == 0 differing from length(target)")
      ## else n, lc  > 0
      if(lc %% n)
          stop("length(current) must be a multiple of length(target)")
      recycle <- (lc != n) # explicitly recycle
      R <- if(recycle)
               target[rep(seq_len(n), length.out=lc)]
           else
               target # (possibly "mpfr")
      R[] <- 0
      ## use *absolute* error when target is zero {and deal with NAs}:
      t0 <- abs(target) < eps0 & !(na.t <- is.na(target))
      R[t0] <- current[t0]
      ## absolute error also when it is infinite, as (-Inf, Inf) would give NaN:
      dInf <- is.infinite(E <- current - target)
      R[dInf] <- E[dInf]
      useRE <- !dInf & !t0 & (na.t | is.na(current) | (current != target))
      R[useRE] <- (current/target)[useRE] - 1
      ## preserve {dim, dimnames, names}  from 'current' :
      if(!is.null(d <- dim(current)))
          array(R, dim=d, dimnames=dimnames(current))
      else if(!is.null(nm <- names(current)) && is.null(names(R))) # not needed for mpfr
```

[6] currently; for updates, see https://github.com/mmaechler/sfsmisc/blob/master/R/relErr.R

```
        `names<-`(R, nm)
    else R
}
```

B. Function p.qnormAsy2() for showing optimal cutpoints

This function, currently also used in **DPQ**'s `example(qnormAsymp)`, was used by the author and may be used for reproducibility to visualize the five "cutpoint - regions" to switch from approximation $x_{k-1}(r)$ to $x_k(r)$, for $k = 1, \ldots, 5$ and $r = \sqrt{s} = \sqrt{-\log p}$, using

```
> r0 <- c(27, 55, 109, 840, 36000, 6.4e8) # <-- cutoffs  <--> in ../R/norm_f.R
> # use k =  5   4    3    2      1       0    e.g.  k = 0  good for r >= 6.4e8
> for(ir in 2:length(r0)) {
    p.qnormAsy2(r0[ir], k = 5 +2-ir) # k = 5, 4, ..
    if(interactive() && ir < length(r0)) {
        cat("[Enter] to continue: "); cat(readLines(stdin(), n=1), "\n") }
  }

> ## Zoom into each each cut-point region :
> p.qnormAsy2 <- function(r0, k, # use k-1 and k in region around r0
                          n = 2048, verbose=TRUE, ylim = c(-1,1) * 2.5e-16,
                          rr = seq(r0 * 0.5, r0 * 1.25, length = n), ...)
  {
    stopifnot(is.numeric(rr), !is.unsorted(rr), # the initial 'r'
              length(k) == 1L, is.numeric(k), k == as.integer(k), k >= 1)
    k.s <- (k-1L):k; nks <- paste0("k=", k.s)
    if(missing(r0)) r0 <- quantile(rr, 2/3)# allow specifying rr instead of r0
    if(verbose) cat("Around r0 =", r0,";  k =", deparse(k.s), "\n")
    lp <- (-rr^2) # = -r^2 = -s  <==> rr = sqrt(- lp)
    q. <- qnormR(lp, lower.tail=FALSE, log.p=TRUE, version="2022-08")# *not* depending on R ver!
    pq <- pnorm (q., lower.tail=FALSE, log.p=TRUE) # ~= lp
    ## the arg of pnorm() is the true qnorm(pq, ..) == q.  by construction
    r <- sqrt(- pq)
    stopifnot(all.equal(rr, r, tol=1e-15))
    qnAsy <- sapply(setNames(k.s, nks), function(ord)
                    qnormAsymp(pq, lower.tail=FALSE, log.p=TRUE, order=ord))
    relE <- qnAsy / q. - 1
    m <- cbind(r, pq, relE)
    if(verbose) {
      print(head(m, 9)); for(j in 1:2) cat(" ..........\n")
      print(tail(m, 4))
    }
    ## matplot(r, relE, type = "b", main = paste("around r0 = ", r0))
    matplot(r, relE, type = "l", ylim = ylim,
       main = paste("Relative error of qnormAsymp(*, k) around r0 = ", r0,
                    "for  k =", deparse(k.s)),
       xlab = quote(r == sqrt(-log(p))), ...)
    legend("topleft", nks, horiz = TRUE, col=1:2, lty=1:2, bty="n", lwd=2)
    for(j in seq_along(k.s))
      lines(smooth.spline(r, relE[,j]), col=adjustcolor(j, 2/3), lwd=4, lty="6132")
    cc <- "blue2"; lab <- substitute(r[0] == R, list(R = r0))
    abline(v  = r0, lty=2, lwd=2, col=cc)
    axis(3, at= r0, labels=lab, col=cc, col.axis=cc, line=-1)
```

```
  abline(h = (-1:1)*.Machine$double.eps, lty=c(3,1,3),
         col=c("green3", "gray", "tan2"))
  invisible(cbind(r = r, qn = q., relE))
}
```

References

Abramowitz M, Stegun IA (1972). *Handbook of Mathematical Functions*. Dover Publications, N. Y. URL https://en.wikipedia.org/wiki/Abramowitz_and_Stegun.

Beasley JD, Springer SG (1977). "Algorithm AS 111: The percentage points of the normal distribution." *Applied Statistics*, **26**(1), 118–121. doi:10.2307/2346889.

Mills JP (1926). "TABLE OF THE RATIO: AREA TO BOUNDING ORDINATE, FOR ANY PORTION OF NORMAL CURVE." *Biometrika*, **18**(3-4), 395–400. ISSN 0006-3444. doi:10.1093/biomet/18.3-4.395.

Wichura MJ (1988). "Algorithm AS 241: The Percentage Points of the Normal Distribution." *Applied Statistics*, **37**(3), 477–484. doi:10.2307/2347330.

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/~maechler

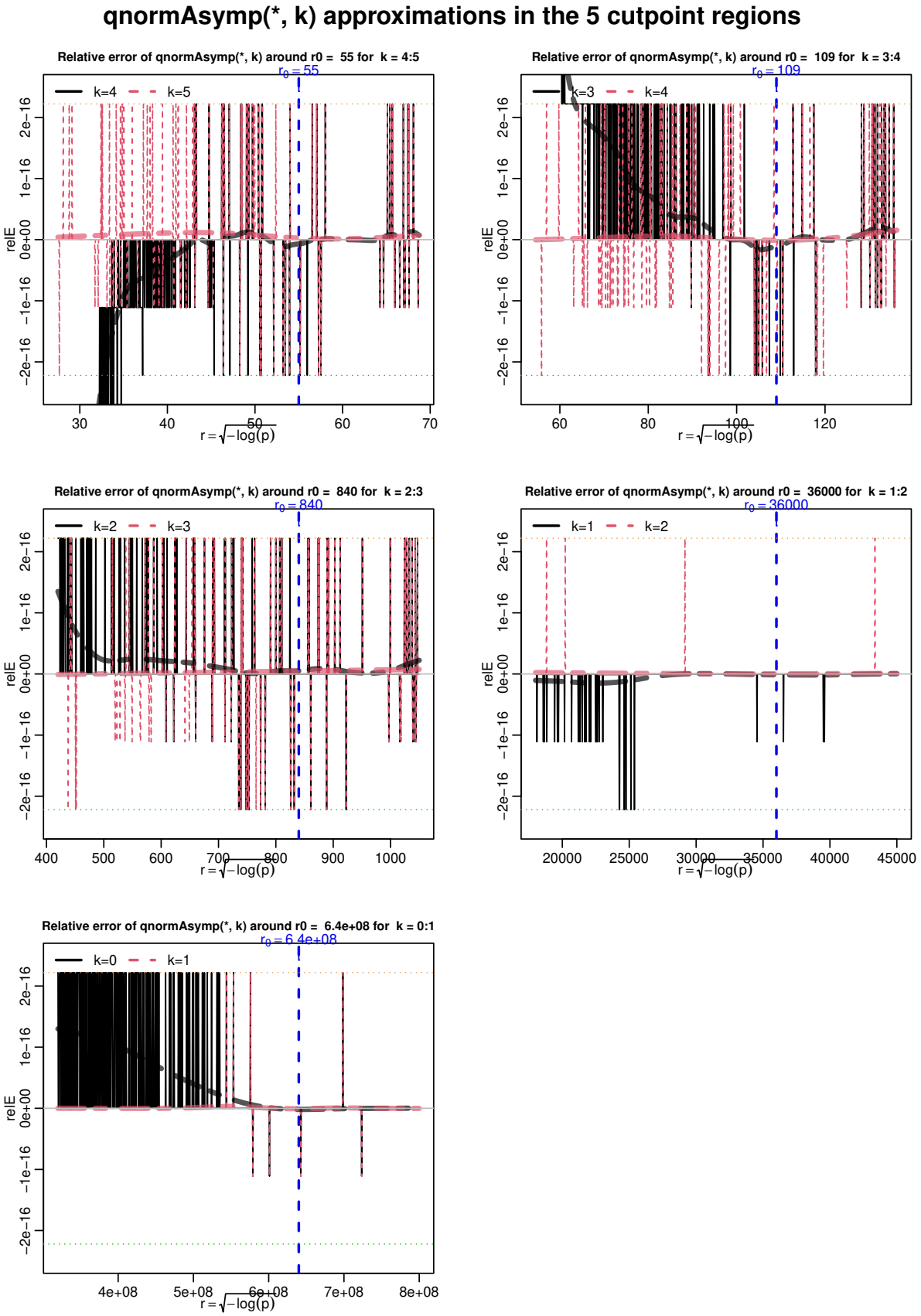


Figure 6: qnormAsymp(*, k) approximation in the 5 cutpoint regions:

```
r0 <- c(27, 55, 109, 840, 36000, 6.4e8)
for(ir in 2:length(r0)) p.qnormAsy2(r0[ir], k = 5 + 2-ir, ..)
```