Benchmarking `rtn()`'s Performance

Jonathan Olmsted
jpolmstedgmail.com

October 15, 2013

This brief document shows some performance benchmarks of **RcppTN**'s `rtn()` compared to other truncated Normal distribution RNG's in R. The other functions considered come from the R packages **truncnorm** and **msm**.

1 Three different RNG's

Broadly speaking, calls to the three different R functions are similar. In the simplest case (no truncation), they even identical return values.

```
set.seed(1)
RcppTN::rtn()

## [1] -0.6265

set.seed(1)
msm::rtnorm(n = 1)

## [1] -0.6265

set.seed(1)
truncnorm::rtruncnorm(n = 1)

## [1] -0.6265
```

But, this is not true in general. Differences in return values result from the use of different algorithms and different implementations of the same algorithm. First, consider a standard Normal distribution truncated below 4 and above 4.1. Here, the output from the **RcppTN** package and the **truncnorm** package agree.

```
set.seed(1)
RcppTN::rtn(.mean = 0, .sd = 1, .low = 4, .high = 4.1)

## [1] 4.027

set.seed(1)
msm::rtnorm(n = 1, mean = 0, sd = 1, lower = 4, upper = 4.1)
```

```
## [1] 4.034

set.seed(1)
truncnorm::rtruncnorm(n = 1, mean = 0, sd = 1, a = 4, b = 4.1)

## [1] 4.027
```

Yet, in the case of truncation below 5 without any truncation from above, the output from the **RcppTN** and the **msm** package agree. Again, this is just a result of how each sampler is implemented using R's base RNG functionality. None of these return values is incorrect, per se, but it is worth noting that the functions do not produce identical output, even if they are all valid RNG's for the same distribution.

```
set.seed(1)
RcppTN::rtn(.mean = 0, .sd = 1, .low = 5, .high = Inf)

## [1] 5.145

set.seed(1)
msm::rtnorm(n = 1, mean = 0, sd = 1, lower = 5, upper = Inf)

## [1] 5.145

set.seed(1)
truncnorm::rtruncnorm(n = 1, mean = 0, sd = 1, a = 5, b = Inf)

## [1] 5.151
```

2 Compiled Code is faster than Interpreted Code

```
library(RcppTN)
library(truncnorm)
library(msm)
library(rbenchmark)
sizes <- c(1e1, 1e3, 1e5)
lows <- c(-1, 5, -Inf, 4, 4, -Inf, 50)
highs <- c(1, Inf, 10, 7, 4.1, Inf, 100)
```

Both **RcppTN** and **truncnorm** use compiled code for their RNG. However, the RNG in **msm** is written in R. As a result, the performance cost that one would expect manifests in even the a simple case where the standard Normal distribution is truncated below at -1 and above at 1. Here, a naive Accept-Reject sampler works perfectly fine. Yet, the C(++)-based implementations are over 20 times faster in drawing samples of size 1,000

```
s <- sizes[2]
lows[1]

## [1] -1

highs[1]

## [1] 1

s

## [1] 1000

benchmark(
    "rtn" = rtn(.mean = rep(0, s),
    .low = rep(lows[1], s),
    .high = rep(highs[1], s)
    ),
    "rtruncnorm" = rtruncnorm(n = s,
    a = rep(lows[1], s),
    b = rep(highs[1], s)
    ),
    "rtnorm" = rtnorm(n = s,
    lower = rep(lows[1], s),
    upper = rep(highs[1], s)
    ),
    replications = 100,
    columns = c("test", "elapsed", "relative")
    )

##         test elapsed relative
## 1        rtn   0.015    1.000
## 3     rtnorm   0.276   18.400
## 2 rtruncnorm   0.019    1.267
```

A similarly large performance cost due to writing the RNG in R is seen in a slightly harder case: a standard Normal distribution truncated below at 4 and above at 4.1. The sample size is still 1,000.

```
lows[5]

## [1] 4

highs[5]

## [1] 4.1

s

## [1] 1000
```

```
benchmark(
    "rtn" = rtn(.mean = rep(0, s),
    .low = rep(lows[5], s),
    .high = rep(highs[5], s)
    ),
    "rtruncnorm" = rtruncnorm(n = s,
    a = rep(lows[5], s),
    b = rep(highs[5], s)
    ),
    "rtnorm" = rtnorm(n = s,
    lower = rep(lows[5], s),
    upper = rep(highs[5], s)
    ),
    replications = 100,
    columns = c("test", "elapsed", "relative")
    )
```

```
##         test elapsed relative
## 1        rtn   0.014    1.000
## 3     rtnorm   0.369   26.357
## 2 rtruncnorm   0.019    1.357
```

For this reason, the `rtnorm()` function from the **msm** package is excluded from subsequent analysis. Not only is it assumed that it will be the slowest for the different sample sizes and truncation bounds considered, but it will just take too long to build the vignette if it is included.

3 RcppTN vs. truncnorm

The RNG's in **RcppTN** and **truncnorm** are written in **Rcpp**-based C++ and C, respectively. However, they implement different mathematical algorithms. The former uses Robert (1995) and the latter uses Geweke (1991). To compare the R-level performance of the two, a more complete set of conditions is considered.

```
library(microbenchmark)
for (case in 1:length(lows)) {
    cat("======================\n")
    cat("Lower Bound:", lows[case], "\n")
    cat("Upper Bound:", highs[case], "\n\n")
    for (s in sizes) {
        cat("  [ Sample Size per Call:", s,  "]\n")
        out <- microbenchmark(
            rtn = rtn(.mean = rep(0, s),
            .low = rep(lows[case], s),
            .high = rep(highs[case], s)
            ),
            rtruncnorm = rtruncnorm(n = s,
            a = rep(lows[case], s),
```

```
            b = rep(highs[case], s)
            ),
            times = 100L
            )
        print(out)
        cat("\n")
    }
    cat("======================\n\n")
}
```

```
## ======================
## Lower Bound: -1
## Upper Bound: 1
##
##   [ Sample Size per Call: 10 ]
## Unit: microseconds
##        expr   min    lq median    uq    max neval
##         rtn 10.61 11.69  13.22 13.74  26.74   100
##  rtruncnorm 19.58 20.38  20.75 21.18 100.00   100
##
##   [ Sample Size per Call: 1000 ]
## Unit: microseconds
##        expr   min    lq median    uq    max neval
##         rtn 127.8 132.7  134.7 136.0 1044.0   100
##  rtruncnorm 177.9 180.3  181.5 182.8  240.5   100
##
##   [ Sample Size per Call: 1e+05 ]
## Unit: milliseconds
##        expr   min    lq median    uq   max neval
##         rtn 12.06 12.23  13.02 13.43 51.80   100
##  rtruncnorm 15.73 16.12  16.85 17.14 54.25   100
##
## ======================
##
## ======================
## Lower Bound: 5
## Upper Bound: Inf
##
##   [ Sample Size per Call: 10 ]
## Unit: microseconds
##        expr   min    lq median    uq   max neval
##         rtn 10.25 11.54  12.59 13.33 53.30   100
##  rtruncnorm 18.57 19.39  19.84 20.34 57.41   100
##
##   [ Sample Size per Call: 1000 ]
## Unit: microseconds
##        expr   min    lq median    uq   max neval
##         rtn 111.0 113.4  114.9 116.5 151.4   100
```

```
##  rtruncnorm 122.1 124.8  125.9 127.3 472.5   100
##
##   [ Sample Size per Call: 1e+05 ]
## Unit: milliseconds
##        expr   min    lq median    uq   max neval
##         rtn 10.06 10.32  11.08 11.45 49.16   100
##  rtruncnorm 10.39 10.45  11.18 11.34 51.18   100
##
## ======================
##
## ======================
## Lower Bound: -Inf
## Upper Bound: 10
##
##   [ Sample Size per Call: 10 ]
## Unit: microseconds
##        expr    min    lq median    uq   max neval
##         rtn  9.668 10.92  12.08 12.62 23.01   100
##  rtruncnorm 18.190 18.98  19.38 19.92 90.61   100
##
##   [ Sample Size per Call: 1000 ]
## Unit: microseconds
##        expr    min     lq median    uq   max neval
##         rtn  97.21  99.05  100.6 101.4 107.1   100
##  rtruncnorm 110.15 111.37  112.3 113.2 475.5   100
##
##   [ Sample Size per Call: 1e+05 ]
## Unit: milliseconds
##        expr   min    lq median    uq   max neval
##         rtn 8.663 8.879  9.562 9.952 47.42   100
##  rtruncnorm 8.971 9.012  9.170 9.848 46.30   100
##
## ======================
##
## ======================
## Lower Bound: 4
## Upper Bound: 7
##
##   [ Sample Size per Call: 10 ]
## Unit: microseconds
##        expr   min    lq median    uq   max neval
##         rtn 10.44 11.60  12.94 13.79 23.20   100
##  rtruncnorm 19.49 20.06  20.65 21.05 91.22   100
##
##   [ Sample Size per Call: 1000 ]
## Unit: microseconds
##        expr   min    lq median    uq   max neval
```

```
##         rtn 141.5 144.1  145.2 146.5 175.5   100
##  rtruncnorm 171.4 173.3  174.6 176.2 521.5   100
##
##   [ Sample Size per Call: 1e+05 ]
## Unit: milliseconds
##        expr   min    lq median    uq   max neval
##         rtn 13.12 13.85  14.02 14.55 52.65   100
##  rtruncnorm 15.08 15.22  15.31 16.03 52.24   100
##
## ======================
##
## ======================
## Lower Bound: 4
## Upper Bound: 4.1
##
##   [ Sample Size per Call: 10 ]
## Unit: microseconds
##        expr   min    lq median    uq   max neval
##         rtn 10.21 11.39  12.35 13.33 23.50   100
##  rtruncnorm 19.39 20.21  20.67 21.22 88.36   100
##
##   [ Sample Size per Call: 1000 ]
## Unit: microseconds
##        expr   min    lq median    uq   max neval
##         rtn 117.5 119.7  121.2 122.3 477.7   100
##  rtruncnorm 180.3 183.5  185.0 186.1 222.1   100
##
##   [ Sample Size per Call: 1e+05 ]
## Unit: milliseconds
##        expr   min    lq median    uq   max neval
##         rtn 10.76 11.09  11.63 11.97 50.24   100
##  rtruncnorm 16.06 16.30  16.65 17.20 54.23   100
##
## ======================
##
## ======================
## Lower Bound: -Inf
## Upper Bound: Inf
##
##   [ Sample Size per Call: 10 ]
## Unit: microseconds
##        expr    min    lq median    uq   max neval
##         rtn  9.855 11.15  12.01 13.10 23.71   100
##  rtruncnorm 18.116 19.45  19.89 20.32 87.06   100
##
##   [ Sample Size per Call: 1000 ]
## Unit: microseconds
```

```
##        expr    min     lq median    uq   max neval
##         rtn  96.89  98.59  100.0 101.8 444.1   100
##  rtruncnorm 107.35 108.96  109.7 110.6 142.9   100
##
##   [ Sample Size per Call: 1e+05 ]
## Unit: milliseconds
##        expr   min    lq median    uq   max neval
##         rtn 8.463 9.350  9.505 9.868 48.06   100
##  rtruncnorm 8.549 8.772  9.379 9.774 45.89   100
##
## ======================
##
## ======================
## Lower Bound: 50
## Upper Bound: 100
##
##   [ Sample Size per Call: 10 ]
## Unit: microseconds
##        expr   min    lq median    uq   max neval
##         rtn 10.56 11.61  13.01 13.62 53.81   100
##  rtruncnorm 20.24 21.08  21.55 22.01 57.43   100
##
##   [ Sample Size per Call: 1000 ]
## Unit: microseconds
##        expr   min    lq median    uq   max neval
##         rtn 130.2 132.2  133.4 135.1 144.8   100
##  rtruncnorm 270.8 274.3  275.8 277.2 636.0   100
##
##   [ Sample Size per Call: 1e+05 ]
## Unit: milliseconds
##        expr   min    lq median    uq   max neval
##         rtn 11.99 12.82  13.20 15.12 54.85   100
##  rtruncnorm 25.22 25.43  26.43 27.92 68.63   100
##
## ======================
```

The motivation for **RcppTN** isn't speed, but the **Rcpp**-based implementation performs quite-well. For larger sample sizes (e.g., $\geq 10^6$), `rtn()` does not necessarily keep its efficiency edge.