Data Extraction Evaluation

Matthew Leonawicz

December 24, 2014

1 Statistical sampling for spatial data extraction

1.1 Motivation: Data processing efficiency

We've gotten faster at SNAP, but so has our need for speed. What was once never bothered with (outside some of my own work), using statistical sampling to obtain results at no cost to validity, accuracy or precision compared to a census of our data, is now more relevant than ever. Before, we were content to let a process run in the background for hours and look at the results when done. There was little incentive to incorporate techniques like those laid out here. Now we have more types of data delivery and presentation, e.g., web applications, where it is intended for there to be a human watching and waiting for data processing to occur.

1.1.1 Assumptions, bad ones

An assumption I often encounter from those outside statistics, but involved in "big data" is that with today's processing power there is no reason not to use all the data. A corollary of this is that many statistical methods can be dispensed with, which is based on another assumption that this is what statistics basically exists for; to help us hobble along when we were in the stone age. However, both of these views are flawed. The latter suggests little knowledge of the broad uses of statistics. The former suggests little knowledge of statistics period, or the myriad ways data can be improperly analyzed and results interpreted.

1.1.2 Speed not for speed's sake

Making things go faster is perhaps the last area of application I would ever find for statistical methods, and since not a lot of traditional statistical analysis occurs at SNAP I do not want those untrained in statistics to get the wrong impression that speed improvements are all statistics is really good for. But it is relevant and beneficial in the context of some of our workflows, particularly my own. But I am also not the only one extracting and processing large amounts of data at SNAP. One use of statistics is data reduction. This is what I aim for when needing to "get things done faster," not really the speed itself. I'd rather see a decrease in computational time required for large data processing occur as a latent consequence of smart application of statistical methods than as something forced for its own sake. I will outline a simple and extremely common case.

1.2 Case study: sample mean

Example of population mean vs. sample mean for a typical map of SNAP's Alaska-Canada 2-km downscaled data. The sample mean converges in distribution to the population mean quite quickly.

```
no.knit <- if ("knitr" %in% names(sessionInfo()$otherPkgs)) FALSE else TRUE
library(raster)
library(microbenchmark)
library(ggplot2)
library(reshape2)
```

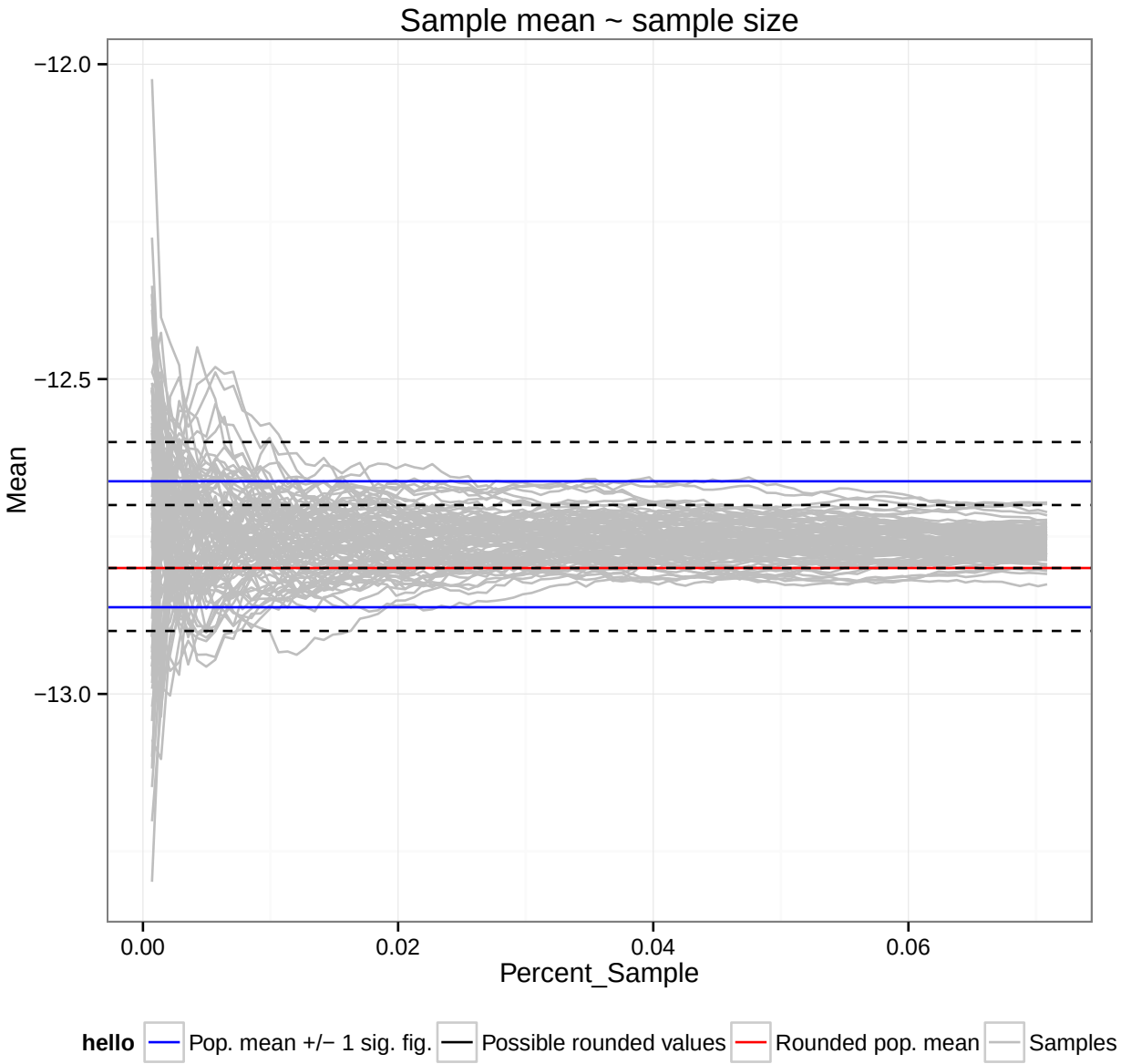
```
setwd("C:/github/DataExtraction/data")
# testfile <-
# 'Z:/Base_Data/ALFRESCO_formatted/ALFRESCO_Master_Dataset/ALFRESCO_Model_Input_Datasets/AK_CAN_Inputs/C
testfile <- "tas_mean_C_AR5_GFDL-CM3_rcp60_01_2062.tif"

r <- readAll(raster(testfile))  # force into memory so I/O time does not confound extraction time
v <- getValues(r)  # numeric vector
dat.ind <- Which(!is.na(r), cells = T)
d <- v[dat.ind]  # numeric vector of data values (drop NAs)
nd <- length(dat.ind)
```

```
# continue indexing v since this is how it will tend to occur in practice
# take mean of all cells
mean(v, na.rm = T)

## [1] -12.7623

# take mean of only the data cells
mean(v[dat.ind])

## [1] -12.7623

# take mean of only the data cells using sum and known length
sum(v[dat.ind])/nd

## [1] -12.7623

# take mean of data cells with .Internal
.Internal(mean(v[dat.ind]))

## [1] -12.7623

# take mean of data cells with .Internal sum, known length
.Primitive("sum")(v[dat.ind])/nd

## [1] -12.7623
```

```
mean.pop <- sum(d)/nd
mean.pop.out <- round(mean.pop, 1)  # round to one decimal place for temperature data
discrete.out <- round(seq(mean.pop, mean.pop + 0.4, by = 0.1) - 0.2, 1)
# median.pop <- median(d) median.pop.out <- round(median.pop, 1) # round to
# one decimal place for temperature data
bounds.round <- mean.pop.out + c(-0.05, 0.05)  # within rounding distance of the rounded population mean
bounds.signif <- mean.pop + c(-0.1, 0.1)  # bounds on the unrounded population mean at the significant d
# Use sample mean
n <- 1e+05
m <- 100
keep <- seq(1000, n, by = 1000)  # burn in and thin to facilitate visualization

set.seed(47)
d.sub <- replicate(m, sample(d, n, replace = F))
means <- data.frame(1:n, (1:n)/nd, apply(d.sub, 2, function(x, n) cumsum(x)/(1:n),
    n = n))
names(means) <- c("Size", "Percent_Sample", paste0("Sample_", c(paste0(0, 0:9),
```

```
    10:m)[1:m]))
means <- means[keep, ]
p <- data.frame(Size = keep, Percent_Sample = keep/nd, P_value = 1 - apply(means,
    1, function(x) length(which(x >= bounds.signif[1] & x < bounds.signif[2]))/length(x)))
means <- melt(means, id.vars = c("Size", "Percent_Sample"), variable.name = c("Sample"),
    value.name = "Mean")
p <- melt(p, id.vars = c("Size", "Percent_Sample"), variable.name = c("Type"),
    value.name = "Pval")

clr <- c(Samples = "gray", `Pop. mean +/- 1 sig. fig.` = "blue", `Rounded pop. mean` = "red",
    `Possible rounded values` = "black")
```

```
if (no.knit) png("../plots/mean_by_size.png", width = 2000, height = 1600, res = 200)
g <- ggplot(means, aes(x = Percent_Sample, y = Mean, group = Sample)) + theme_bw() +
    geom_line(aes(colour = "Samples")) + geom_hline(aes(yintercept = d, colour = "Rounded pop. mean"),
    data = data.frame(d = mean.pop.out)) + geom_hline(aes(yintercept = d, colour = c("Pop. mean +/- 1 si
    data = data.frame(d = bounds.signif)) + geom_hline(aes(yintercept = d, colour = "Possible rounded va
    , data = data.frame(d = discrete.out[2:5]), linetype = 2) + scale_colour_manual(name = "hello",
    values = clr) + theme(legend.position = "bottom") + labs(title = "Sample mean ~ sample size")
print(g)
```




```
if (no.knit) dev.off()
```

1.2.1 Justification

The difference between sampling vs. using all data in a map layer is minimal. It depends on various factors including but not limited to the statistic of interest, the spatial autocorrelation present in the map, and whether the entire map is of interest or just a particular region of a certain size.

In this example using the sample mean instead of the population mean, the difference is representative. The difference is also not particularly meaningful. It is also not final, as it tends to vanish anyway due to rounding to the nearest significant digits for the data after the statistic is computed. The difference in means can also be bounded arbitrarily even without the rounding to significant digits performed at the end.

In the case of the mean we are helped out by the weak law of large numbers and the central limit theorem. Consideration must also be given to the high level of spatial autocorrelation among pixels in the downscaled

raster maps. There is simply not as much data or information present as one might think and this drives the effective sample size.

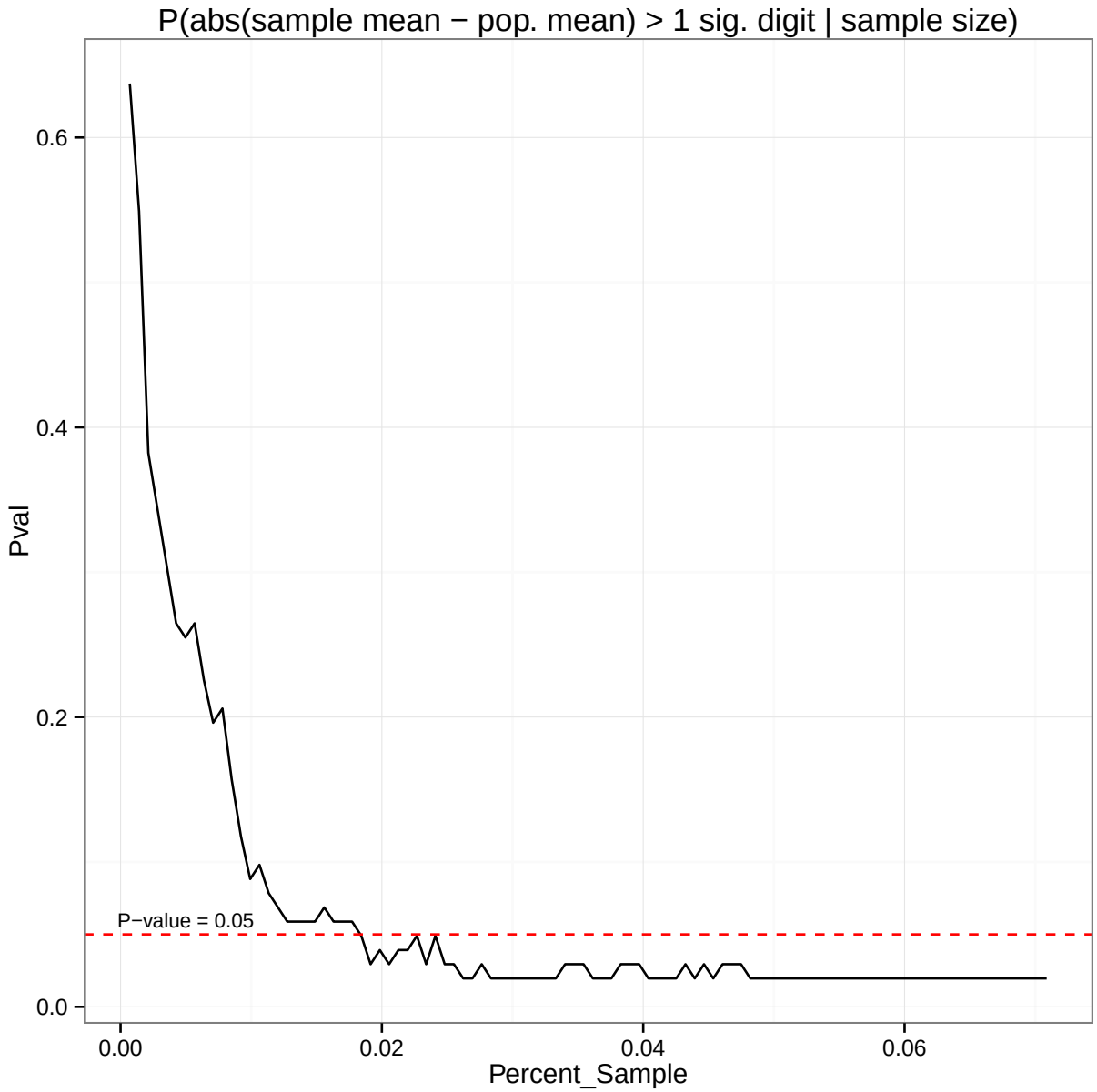
1.2.2 Results

All points? No point.

Using the sample mean is helpful as a data reduction strategy while not being harmful in terms of representativeness. The possible "tradeoff" itself appears to be largely a false dichotomy. There is no benefit to computing the mean of all pixels in the example map layer.

How many samples do we really need?

```
if (no.knit) png("../plots/pvalue_sigdig.png", width = 2000, height = 2000,
    res = 200)
g <- ggplot(p, aes(x = Percent_Sample, y = Pval, group = Type, colour = Type)) +
    theme_bw() + geom_line(colour = "black")
g <- g + geom_hline(aes(yintercept = 0.05, linetype = "P-value = 0.05"), colour = "red",
    linetype = 2) + annotate("text", 0.005, 0.05 * 1.2, label = "P-value = 0.05",
    size = 3)
g <- g + labs(title = "P(abs(sample mean - pop. mean) > 1 sig. digit | sample size)")
print(g)
```




```
if (no.knit) dev.off()
```

In this example even a two percent subsample of the original non-NA data cells is small enough to limit us to a five percent probability of obtaining a mean that differs from the mean computed on the full dataset by an amount equal to or greater than the smallest discrete increment possible (0.1 degrees Celsius for SNAP temperature data) based simply on the number of significant figures present. Furthermore, even for nominal sample sizes, the 0.05 probability is one almost strictly of minimal deviation (0.1 degrees). The probability that a sample mean computed on a subsample of the map layer deviates enough from the population mean to cause it to be rounded to two discrete incremental units from the population mean (0.2 degrees) is essentially zero (except if using crudely small sample sizes).

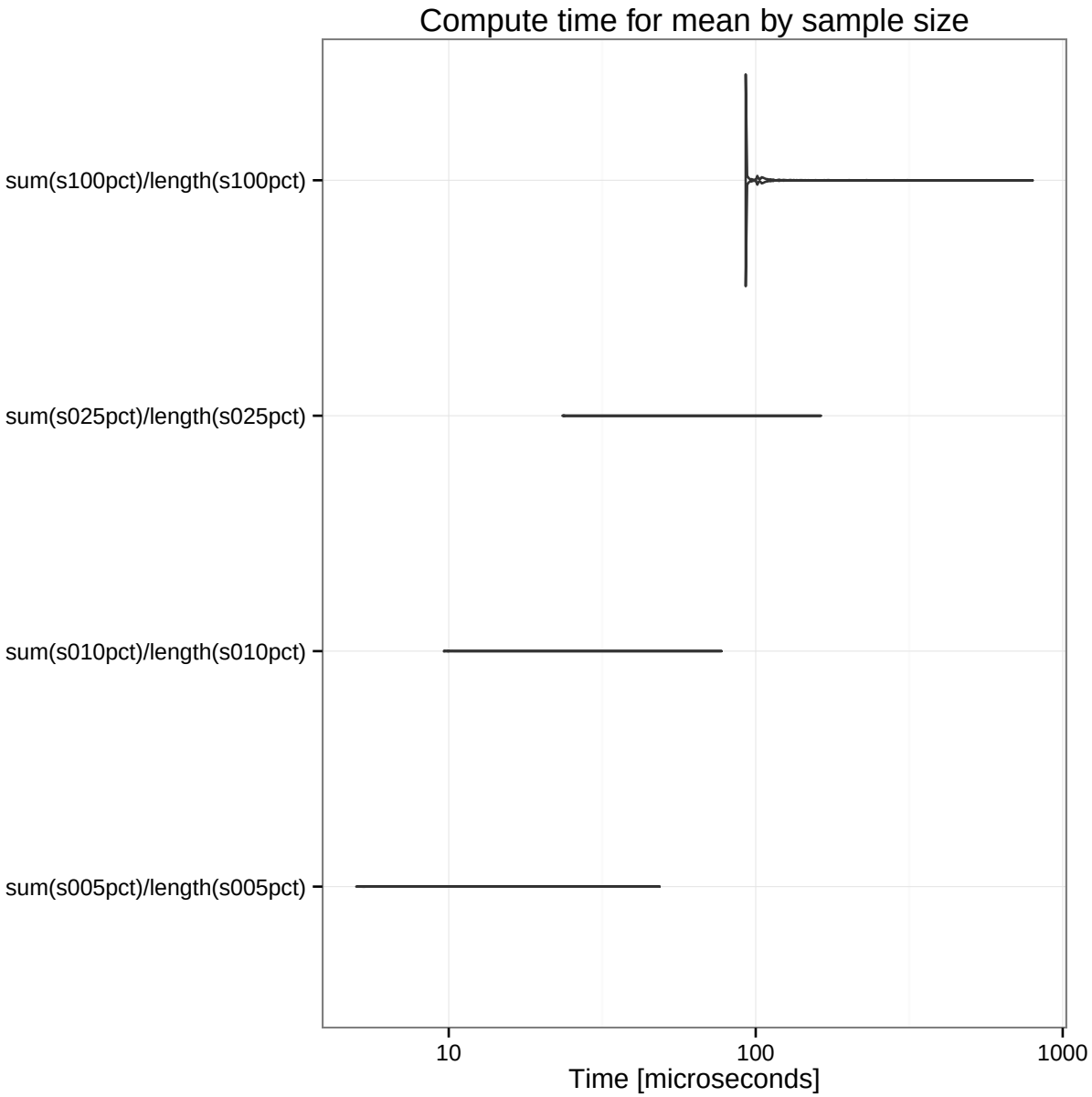
Although a two percent subsample appears sufficient for this criterion, lets use a five percent subsample for illustration. This is clearly overkill in this example since the p-value attenuates to the range of 0.019 to 0.029 by around 2.5 percent subsampling.

How much faster does this make things go?

Compute time for the mean is of course affected by the sample size.

```
# compute time for means for different sample size
s005pct <- d.sub[1:round((nrow(d.sub) * 0.05)), 1]
s010pct <- d.sub[1:round((nrow(d.sub) * 0.1)), 1]
s025pct <- d.sub[1:round((nrow(d.sub) * 0.25)), 1]
s100pct <- d.sub[, 1]
```

```
mb3 <- microbenchmark(sum(s005pct)/length(s005pct), sum(s010pct)/length(s010pct),
    sum(s025pct)/length(s025pct), sum(s100pct)/length(s100pct), times = 10000)
mb3

## Unit: microseconds
##                          expr    min     lq      mean median     uq
##  sum(s005pct)/length(s005pct)  4.976  5.288  5.329049  5.288  5.288
##  sum(s010pct)/length(s010pct)  9.642  9.642  9.919523  9.953  9.953
##  sum(s025pct)/length(s025pct) 23.326 23.326 23.703385 23.637 23.637
##  sum(s100pct)/length(s100pct) 91.746 92.057 92.653782 92.057 92.058
##       max neval
##    48.828 10000
##    77.440 10000
##   163.899 10000
##   807.050 10000

if (no.knit) png("../plots/benchmark3.png", width = 2000, height = 1600, res = 200)
autoplot(mb3) + theme_bw() + labs(title = "Compute time for mean by sample size",
    y = "Function")
```

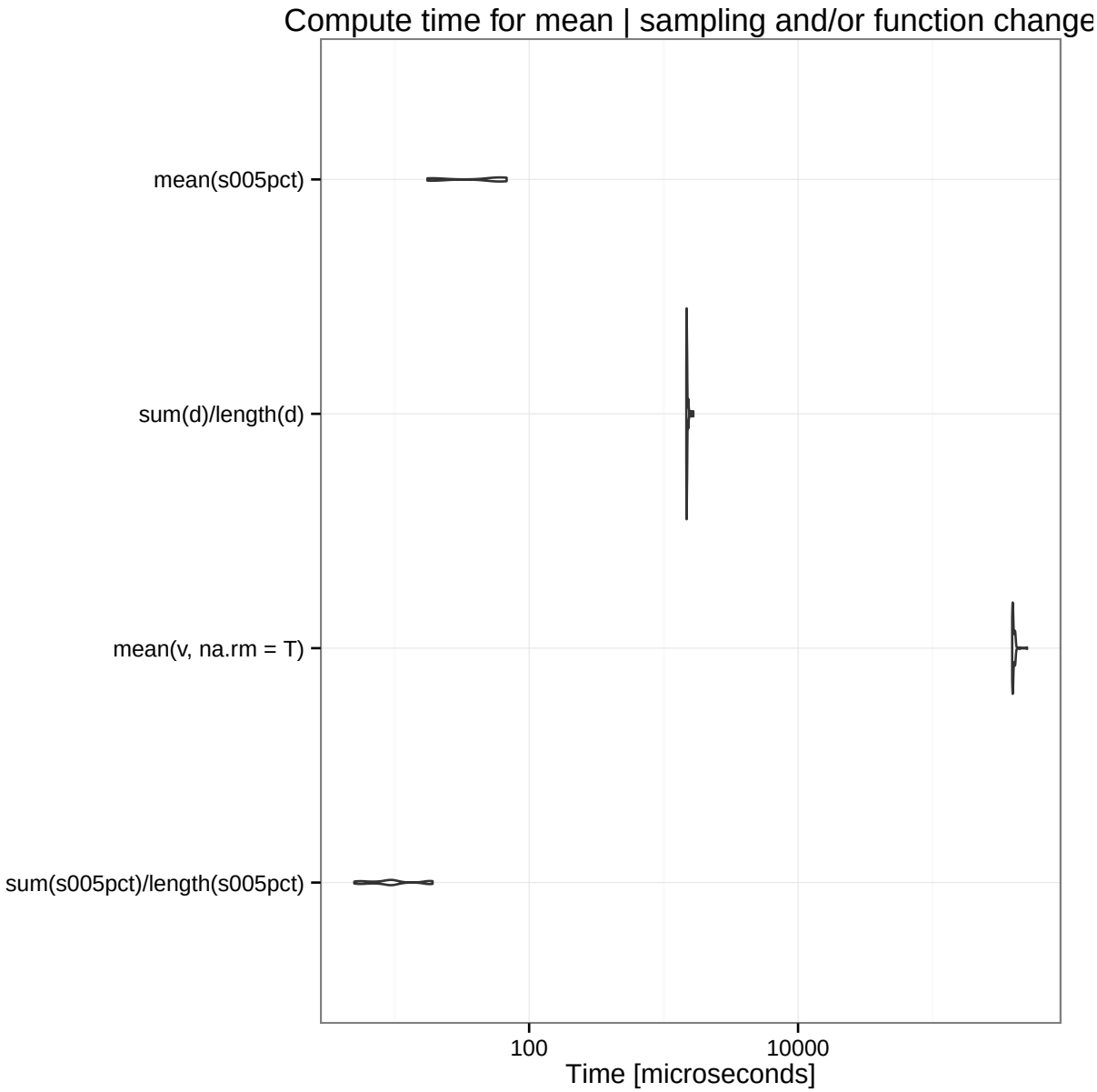



```
if (no.knit) dev.off()
```

Using optimal subsampling to estimate the mean achieves speed improvements orders of magnitude greater than what can be achieved through strictly algorithmic changes to how the mean is computed on the full dataset, though those help immensely as well, also by many orders of magnitude. Sampling is vastly more effective, but both approaches can be combined for maximum benefit.

```
mb4 <- microbenchmark(sum(s005pct)/length(s005pct), mean(v, na.rm = T), sum(d)/length(d),
    mean(s005pct), times = 100)
mb4

## Unit: microseconds
##                          expr        min          lq         mean
##  sum(s005pct)/length(s005pct)      4.977      6.9985     10.33559
```

```
##            mean(v, na.rm = T) 391108.905 394197.6155 401290.90893
##              sum(d)/length(d)   1476.325   1482.7000   1497.67516
##                 mean(s005pct)     17.417     19.7500     45.08691
##      median         uq        max neval
##       9.331     10.575     19.283   100
##  395102.475 408130.325 504864.841   100
##    1488.765   1498.095   1663.236   100
##      54.271     62.512     68.421   100

med <- print(mb4)$median

## Unit: microseconds
##                          expr        min          lq         mean
##  sum(s005pct)/length(s005pct)      4.977      6.9985     10.33559
##            mean(v, na.rm = T) 391108.905 394197.6155 401290.90893
##              sum(d)/length(d)   1476.325   1482.7000   1497.67516
##                 mean(s005pct)     17.417     19.7500     45.08691
##      median         uq        max neval
##       9.331     10.575     19.283   100
##  395102.475 408130.325 504864.841   100
##    1488.765   1498.095   1663.236   100
##      54.271     62.512     68.421   100

names(med) <- print(mb4)$expr

## Unit: microseconds
##                          expr        min          lq         mean
##  sum(s005pct)/length(s005pct)      4.977      6.9985     10.33559
##            mean(v, na.rm = T) 391108.905 394197.6155 401290.90893
##              sum(d)/length(d)   1476.325   1482.7000   1497.67516
##                 mean(s005pct)     17.417     19.7500     45.08691
##      median         uq        max neval
##       9.331     10.575     19.283   100
##  395102.475 408130.325 504864.841   100
##    1488.765   1498.095   1663.236   100
##      54.271     62.512     68.421   100

med <- med[c(1, 4:2)]

if (no.knit) png("../plots/benchmark4.png", width = 2000, height = 1600, res = 200)
autoplot(mb4) + theme_bw() + labs(title = "Compute time for mean | sampling and/or function change",
    y = "Function")
```

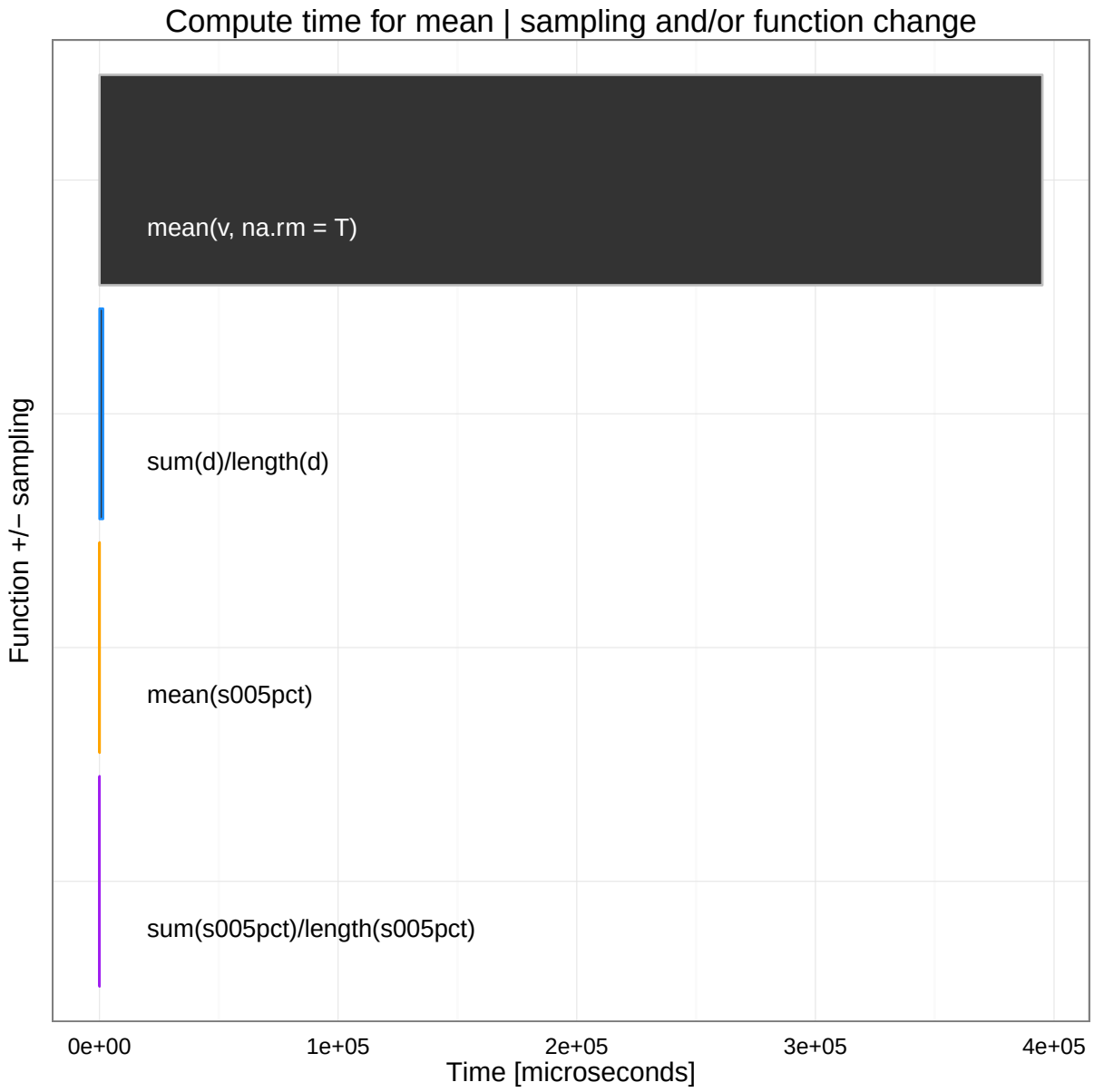



```
if (no.knit) dev.off()
```

Similar to above, below are the median compute times for the mean using (1) the full data while removing NAs, (2) the sum divided by the length after NAs removed, (3) the mean of a subsample, and (4) a combination of (2) and (3).

```
if (no.knit) png("../plots/benchmark4medians.png", width = 2000, height = 1000,
    res = 200)
ggplot(data.frame(x = names(med), y = med), aes(x = reorder(x, 1:length(x),
    function(z) z), y = y, colour = x)) + geom_bar(stat = "identity", size = 0.5,
    width = 0.9) + theme_bw() + theme(legend.position = "none", axis.ticks = element_blank(),
    axis.text.y = element_blank()) + scale_colour_manual(values = c("gray",
    "dodgerblue", "orange", "purple")[c(3, 1, 2, 4)]) + labs(title = "Compute time for mean | sampling a
    x = "Function +/- sampling", y = "Time [microseconds]") + annotate("text",
```

```
    x = (1:4) - 0.2, y = 20000, label = names(med), size = 4, hjust = 0, colour = c(rep("black",
        3), "white")) + coord_flip()
```

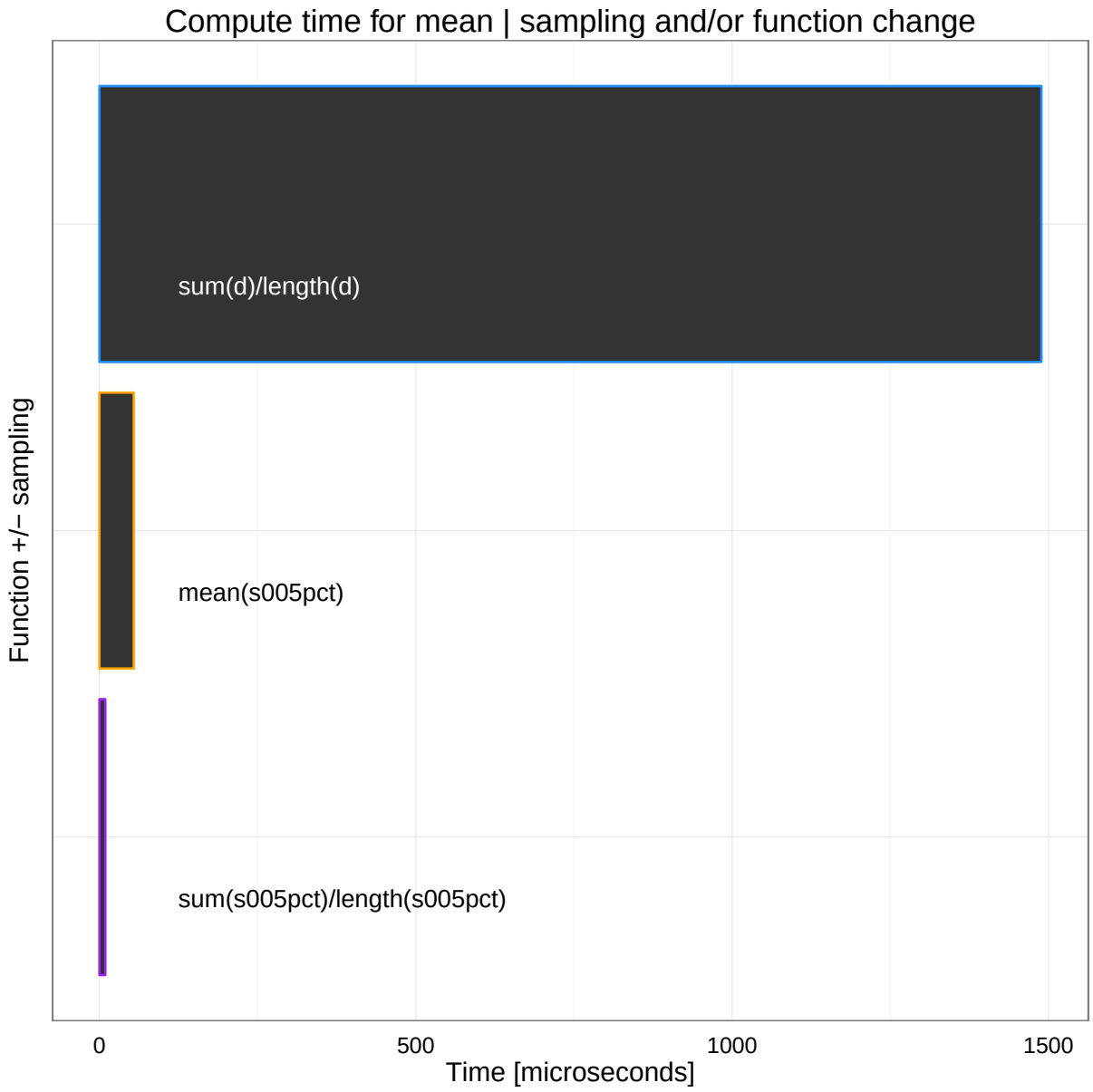



```
if (no.knit) dev.off()
```

Here is the same plot after removing the first bar to better show the relative compute time for the other three methods.

```
if (no.knit) png("../plots/benchmark4medians2.png", width = 2000, height = 1000,
    res = 200)
ggplot(data.frame(x = names(med)[-4], y = med[-4]), aes(x = reorder(x, 1:length(x),
    function(z) z), y = y, colour = x)) + geom_bar(stat = "identity", size = 0.5,
    width = 0.9) + theme_bw() + theme(legend.position = "none", axis.ticks = element_blank(),
    axis.text.y = element_blank()) + scale_colour_manual(values = c("dodgerblue",
```

```
    "orange", "purple")[c(2, 1, 3)]) + labs(title = "Compute time for mean | sampling and/or function ch
    x = "Function +/- sampling", y = "Time [microseconds]") + annotate("text",
    x = (1:(4 - 1)) - 0.2, y = 125, label = names(med)[-4], size = 4, hjust = 0,
    colour = c(rep("black", 3 - 1), "white")) + coord_flip()
```




```
if (no.knit) dev.off()
```

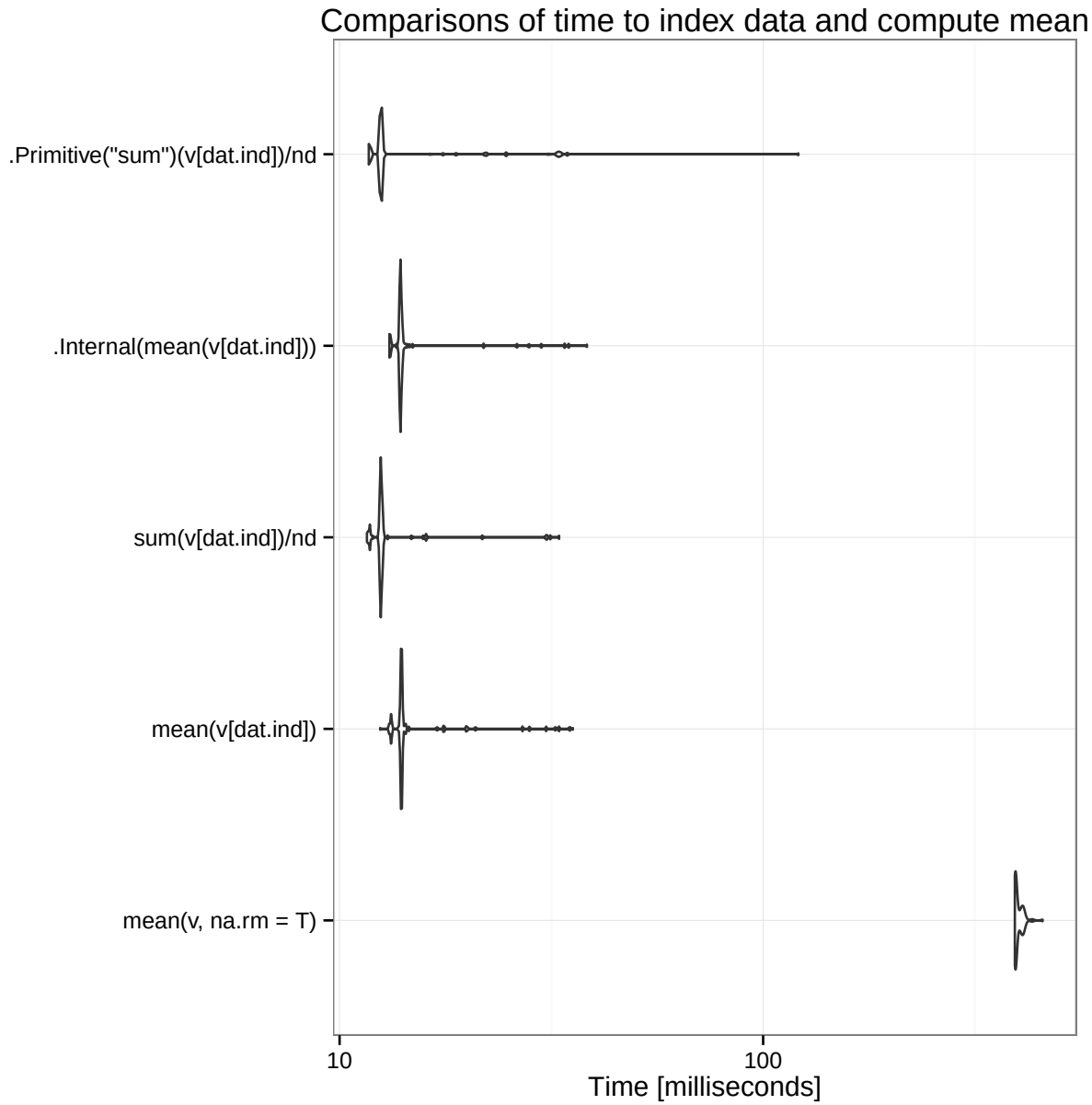
How does the benefit extend to extractions on maps at different extents, data heterogeneity, climate variables, or for other common statistics such as the standard deviation? These are open questions at the moment, but for one thing, I expect more samples are needed for precipitation than temperature. I also expect more samples needed to estimate parameters with higher moments.

1.3 Next steps

Combining sampling and data reduction methods while using the most efficient **R** functions can be particularly useful when processing large numbers of high-resolution geotiff raster layers. One thing I already do when extracting from many files by shapefile is I avoid extracting by shape more than once. I do it one time to obtain the corresponding raster layer cell indices. Then on all subsequent maps I extract by cell indices which is notably faster. Ultimately, there is much more room for speed improvements in terms of efficient use of statistics than in strictly programmatic corner-cutting.

The plots below benchmark different sample mean computations. Comparisons involve the sample mean of the entire data set and do not involve the main approach outlined above which focuses on efficiency gains by taking the mean of a smaller, representative sample. This provides some insight into how it is beneficial nonetheless to considering the right programmatic approach in conjunction with statistical efficiencies.

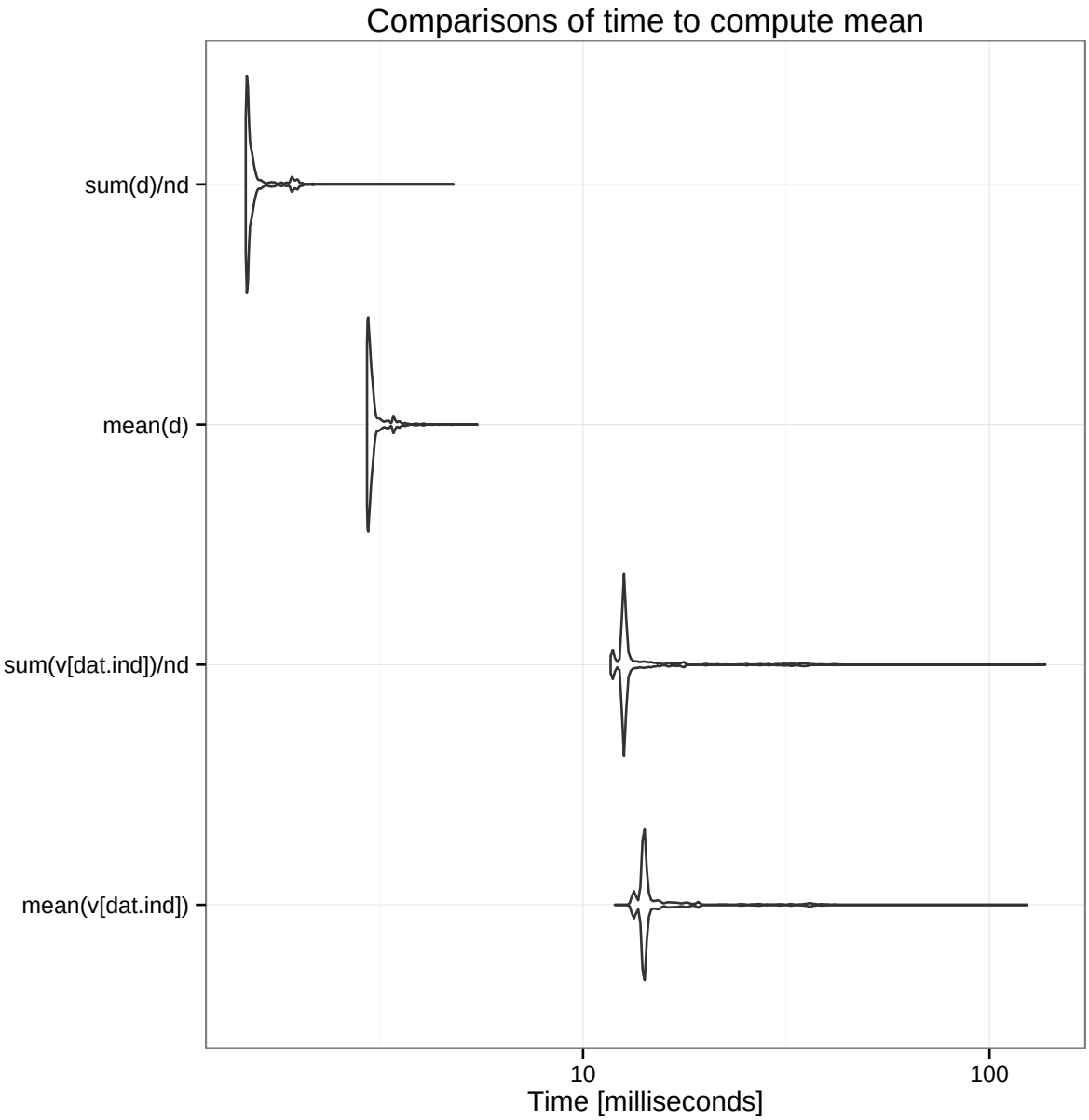
```
mb <- microbenchmark(mean(v, na.rm = T), mean(v[dat.ind]), sum(v[dat.ind])/nd,
    .Internal(mean(v[dat.ind])), .Primitive("sum")(v[dat.ind])/nd, times = 100)
mb

## Unit: milliseconds
##                               expr       min        lq      mean    median
##               mean(v, na.rm = T) 392.68724 394.05222 400.38345 394.85087
##                 mean(v[dat.ind])  12.42109  13.94157  15.43536  14.00035
##               sum(v[dat.ind])/nd  11.58107  12.45903  13.41523  12.51765
##       .Internal(mean(v[dat.ind]))  13.06766  13.87704  15.02636  13.92012
##  .Primitive("sum")(v[dat.ind])/nd  11.67064  12.46276  15.40434  12.52170
##         uq       max neval
##  406.63752 457.22097   100
##   14.06333  35.68089   100
##   12.59976  33.04919   100
##   14.01450  38.49203   100
##   12.58250 121.63734   100

if (no.knit) png("../plots/benchmark1.png", width = 2000, height = 1600, res = 200)
autoplot(mb) + theme_bw() + labs(title = "Comparisons of time to index data and compute mean",
    y = "Function")
```




```
if (no.knit) dev.off()
```

```
mb2 <- microbenchmark(mean(v[dat.ind]), sum(v[dat.ind])/nd, mean(d), sum(d)/nd,
    times = 1000)
mb2

## Unit: milliseconds
##                expr       min        lq      mean    median        uq
##    mean(v[dat.ind]) 11.839516 14.043738 16.112048 14.159742 14.775524
##  sum(v[dat.ind])/nd 11.639541 12.543934 14.728460 12.647030 13.069682
##             mean(d)  2.937099  2.956692  3.059285  2.990902  3.045327
##           sum(d)/nd  1.478502  1.490320  1.566962  1.502760  1.543657
##         max neval
```

```
##  125.081989  1000
##  137.833982  1000
##    5.506281  1000
##    4.805907  1000

if (no.knit) png("../plots/benchmark2.png", width = 2000, height = 1600, res = 200)
autoplot(mb2) + theme_bw() + labs(title = "Comparisons of time to compute mean",
    y = "Function")
```




```
if (no.knit) dev.off()
```