comparison_Gaussian.r

max

2021-03-13

```r
library(npowerPrioR)
```

```
## Loading required package: parallel
```

```
## Loading required package: mgcv
```

```
## Loading required package: nlme
```

```
## This is mgcv 1.8-33. For overview type 'help("mgcv-package")'.
```

```
## Loading required package: rstan
```

```
## Loading required package: StanHeaders
```

```
## Loading required package: ggplot2
```

```
## rstan (Version 2.21.2, GitRev: 2e1f913d3ca3)
```

```
## For execution on a local, multicore CPU with excess RAM we recommend calling
## options(mc.cores = parallel::detectCores()).
## To avoid recompilation of unchanged Stan programs, we recommend calling
## rstan_options(auto_write = TRUE)
```

```
## Loading required package: bridgesampling
```

```r
source("../Gaussian/data_Gaussian.r")

gs.data <- list(
  N0 = N_0,
  y0 = y_0,
  mu0 = mu_0,
  kappa0 = kappa_0,
  alpha0 = alpha_0,
  beta0 = beta_0,
  a_0 = 1
)

###
get_l_a0_gaussian <- function(y0, n0, alpha0, beta0, m0, k0, a_0){
  nstar <- a_0 * n0
  ybar <- mean(y0)
  s <- mean( (y0-ybar)^2 )
  kappa_n <- k0 + nstar
  alpha_n <- alpha0 + nstar/2
  beta_n <- beta0 + .5 * (nstar * s +  (k0 * nstar * (ybar - m0)^2 )/kappa_n )
  ans <- lgamma(alpha_n)-lgamma(alpha0)
  ans <- ans + alpha0 * log(beta0) - alpha_n * log(beta_n)
  ans <- ans + .5 *( log(k0) - log(kappa_n) )-nstar/2 * log(2*pi)
  return(ans)
}
############
l_a0 <- function(x) {
  get_l_a0_gaussian(
    y0 = gs.data$y0,
    n0 = gs.data$N0,
    alpha0 = gs.data$alpha0,
    beta0 = gs.data$beta0,
    m0 = gs.data$mu0,
    k0 = gs.data$kappa0,
    a_0 = x
  )
}
l_a0 <- Vectorize(l_a0)
########

maxA <- 1
prior <- stan_model("../Gaussian/stan/simple_Gaussian_prior.stan")
```

```
## Trying to compile a simple C file

## Running /usr/local/lib/R/bin/R CMD SHLIB foo.c
## gcc -I"/usr/local/lib/R/include" -DNDEBUG   -I"/home/max/R/x86_64-pc-linux-gnu-library/4.0/Rcpp/inclu
## In file included from /home/max/R/x86_64-pc-linux-gnu-library/4.0/RcppEigen/include/Eigen/Core:88,
##                  from /home/max/R/x86_64-pc-linux-gnu-library/4.0/RcppEigen/include/Eigen/Dense:1,
##                  from /home/max/R/x86_64-pc-linux-gnu-library/4.0/StanHeaders/include/stan/math/prim
##                  from <command-line>:
## /home/max/R/x86_64-pc-linux-gnu-library/4.0/RcppEigen/include/Eigen/src/Core/util/Macros.h:613:1: er
##   613 | namespace Eigen {
##       | ^~~~~~~~~
## /home/max/R/x86_64-pc-linux-gnu-library/4.0/RcppEigen/include/Eigen/src/Core/util/Macros.h:613:17: e
##   613 | namespace Eigen {
##       |                 ^
## In file included from /home/max/R/x86_64-pc-linux-gnu-library/4.0/RcppEigen/include/Eigen/Dense:1,
##                  from /home/max/R/x86_64-pc-linux-gnu-library/4.0/StanHeaders/include/stan/math/prim
##                  from <command-line>:
## /home/max/R/x86_64-pc-linux-gnu-library/4.0/RcppEigen/include/Eigen/Core:96:10: fatal error: complex
##    96 | #include <complex>
##       |          ^~~~~~~~~
## compilation terminated.
## make: *** [/usr/local/lib/R/etc/Makeconf:172: foo.o] Error 1
```

```
# direct method
J <- 20
epsilon <- 0.05

adaptive.time <- system.time(
  adaptive.ca0.estimates <- build_grid(compiled.model.prior = prior, eps = epsilon,
                                       M = maxA, J = J, v1 = 10, v2 = 10,
                                       stan.list = gs.data, pars = c("mu", "sigma"))
)

# VR2018
Delta.a <- 0.01
```

```
a0s.vr2018 <- seq(0, maxA, by = Delta.a)

vr2018.time <- system.time(
  vr2018.estimates <-  create_lc_df_derivOnly(a0_grid = a0s.vr2018,
                              compiled.model.prior = prior,
                              stan.list = gs.data, pars = c("mu", "sigma") )
)
```

```
## Warning: Tail Effective Samples Size (ESS) is too low, indicating posterior variances and tail quanti
## Running the chains for more iterations may help. See
## http://mc-stan.org/misc/warnings.html#tail-ess
```

```
write.csv(vr2018.estimates$result,
          file = "Gaussian_VR2018.csv", row.names = FALSE)
adaptive.time
```

```
##    user  system elapsed
##  18.109   0.096  18.232
```

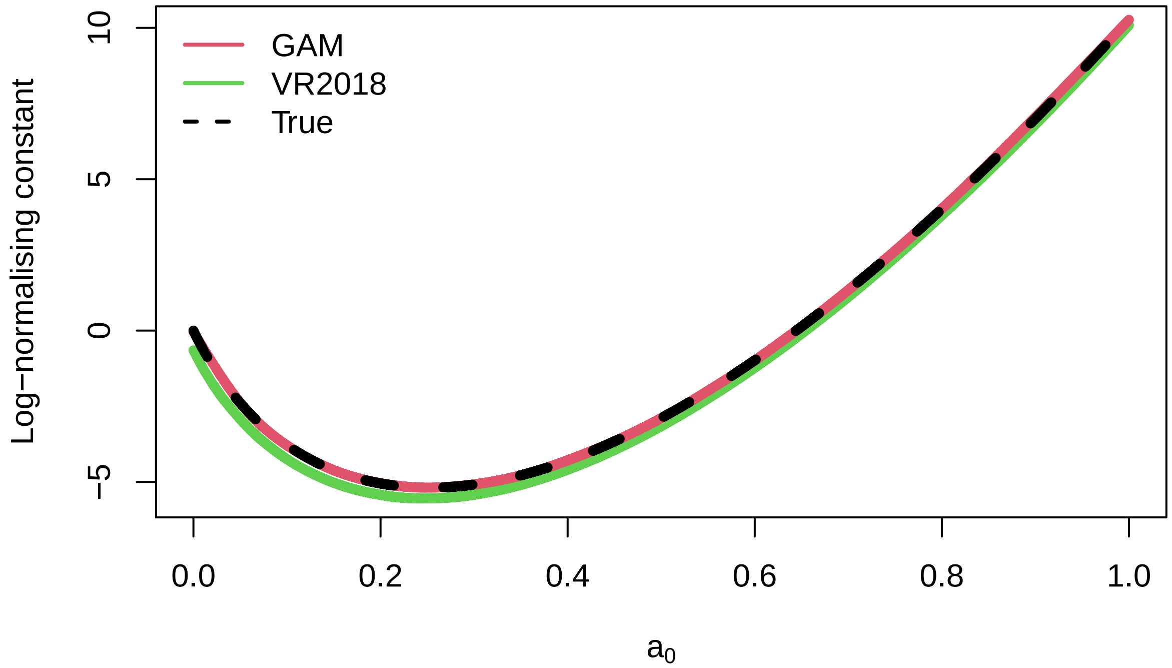
```
vr2018.time
```

```
##    user  system elapsed
##  18.004   0.080  18.156
```

```
###
## Now the approximations
adapt.gam <-  mgcv::gam(lc_a0 ~ s(a0, k = J), data = adaptive.ca0.estimates$result)
vr2018.estimates$result$la0_est <- cumsum(vr2018.estimates$result$deriv_lc) * Delta.a


## Finally, comparisons

K <- 20000
pred.a0s <- seq(0, maxA, length.out = K)

true.la0s <- l_a0(pred.a0s)

adaptive.preds <- predict(adapt.gam, newdata = data.frame(a0 = pred.a0s))

vr2018.preds <- approx(x = vr2018.estimates$result$a0,
                       y =  vr2018.estimates$result$la0_est,
                       xout =  pred.a0s)

plot(vr2018.preds, type = "l", lwd = 5,
     col = 3,
     xlab = expression(a[0]), ylab = "Log-normalising constant")
lines(pred.a0s, adaptive.preds, col = 2, lwd = 5)
lines(pred.a0s, true.la0s, lwd = 5, lty = 2, add = TRUE)
```

```
## Warning in plot.xy(xy.coords(x, y), type = type, ...): "add" is not a graphical
## parameter
```

```
legend(x = "topleft", legend =  c("GAM", "VR2018", "True"),
       col = c(2, 3, 1), lwd  = 2, lty = c(1, 1, 2), bty = 'n')
```



```r
preds.list <- list(
  adaptive = adaptive.preds,
  VR2018 = vr2018.preds$y
)

ntrue.la0s <- true.la0s

lapply(preds.list, function(pred) sqrt(mean( ( pred- ntrue.la0s)^2 )) )
```

```
## $adaptive
## [1] 0.02123131
##
## $VR2018
## [1] 0.3196206
```

```r
lapply(preds.list, function(pred) mean( abs( pred- ntrue.la0s) ))
```

```
## $adaptive
## [1] 0.0123095
##
## $VR2018
## [1] 0.3013275
```