non_invertible_fixed_alpha.r

max

Tue Jan 23 13:42:56 2018

```r
### This script implements example from page 1250 in Poole & Rafetery (2000), JASA
### Original code by Gabriel Mendes (Berkeley): http://discourse.mc-stan.org/t/bayesian-melding/3011
### Implements the (unormalised) exact target of the example, mainly to demonstrate correctness
##### Copyleft (or the one to blame): Luiz Max Carvalho (2018)
fixed_alpha <- '
functions{
real fZ_exact_lpdf(real z, real ax, real bx, real ay, real by){
// notice the lack of in-built check: ay/bx < x < by/ax
real k;
real L;
real U;
k = (bx-ax)*(by-ay);
L = max({ax, ay/z});
U = min({bx, by/z});
return(log(((U *fabs(U))- (L *fabs(L)))/(2*k))) ;
}
}
data{
real<lower=0, upper=1> alpha;
int<lower=0> M; // number of samples for method of moments
real<lower=0> max_X;
real<lower=0, upper=max_X> min_X;
real<lower=0> max_Y;
real<lower=0, upper=max_Y> min_Y;
}
parameters {
real<lower=min_Y/max_X, upper=max_Y/min_X> Z;
}
model{
target += alpha * uniform_lpdf(Z |0,5) + (1-alpha)*fZ_exact_lpdf(Z | min_X, max_X, min_Y, max_Y);
}
'
######################
library(rstan)
```

```
## Loading required package: ggplot2

## Loading required package: StanHeaders

## rstan (Version 2.16.2, packaged: 2017-07-03 09:24:58 UTC, GitRev: 2e1f913d3ca3)

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

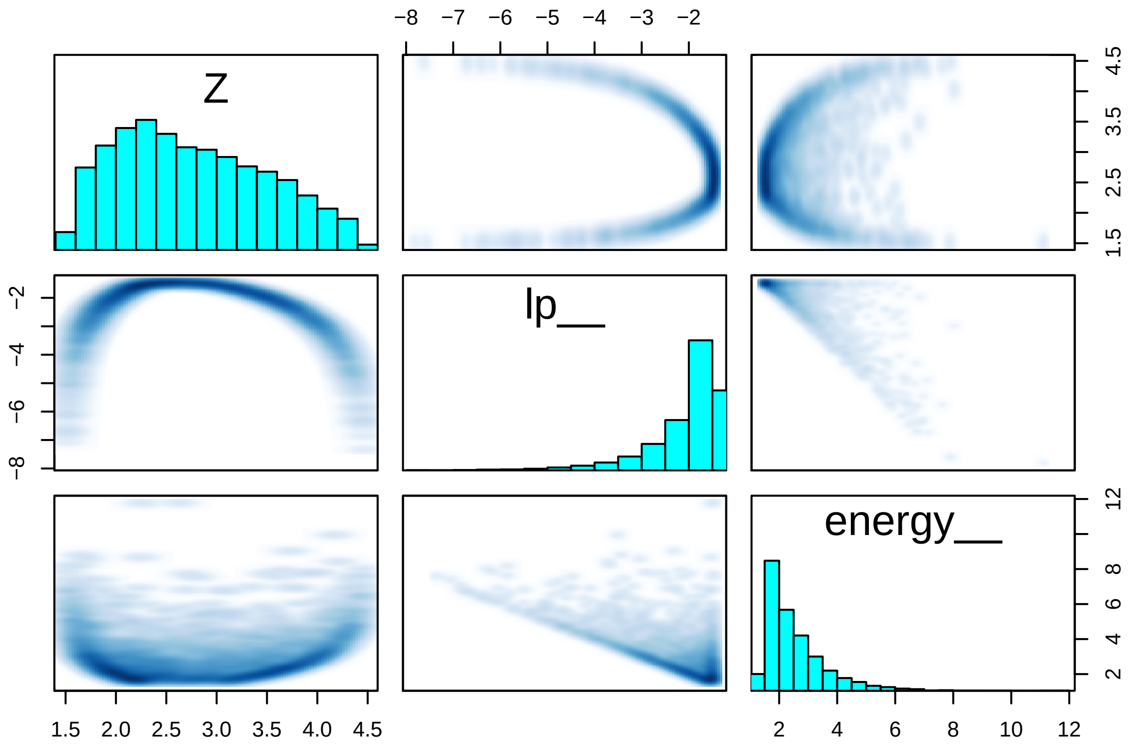
```r
options(mc.cores = parallel::detectCores())

fixed_alpha_run <- stan(model_code = fixed_alpha,
                        data = list(alpha = .5,
                                    M = 1000,
                                    min_X = 2, max_X = 4,
                                    min_Y = 6, max_Y = 9),
                        iter = 5000
)
```

```
## In file included from /home/max/R/x86_64-pc-linux-gnu-library/3.4/BH/include/boost/config.hpp:39:0,
##                  from /home/max/R/x86_64-pc-linux-gnu-library/3.4/BH/include/boost/math/tools/config
##                  from /home/max/R/x86_64-pc-linux-gnu-library/3.4/StanHeaders/include/stan/math/rev/
##                  from /home/max/R/x86_64-pc-linux-gnu-library/3.4/StanHeaders/include/stan/math/rev/
##                  from /home/max/R/x86_64-pc-linux-gnu-library/3.4/StanHeaders/include/stan/math/rev/
##                  from /home/max/R/x86_64-pc-linux-gnu-library/3.4/StanHeaders/include/stan/math/rev/
##                  from /home/max/R/x86_64-pc-linux-gnu-library/3.4/StanHeaders/include/stan/math.hpp:
##                  from /home/max/R/x86_64-pc-linux-gnu-library/3.4/StanHeaders/include/src/stan/model
##                  from file55d720956b2c.cpp:8:
## /home/max/R/x86_64-pc-linux-gnu-library/3.4/BH/include/boost/config/compiler/gcc.hpp:186:0: warning:
##  #  define BOOST_NO_CXX11_RVALUE_REFERENCES
##  ^
## <command-line>:0:0: note: this is the location of the previous definition
```

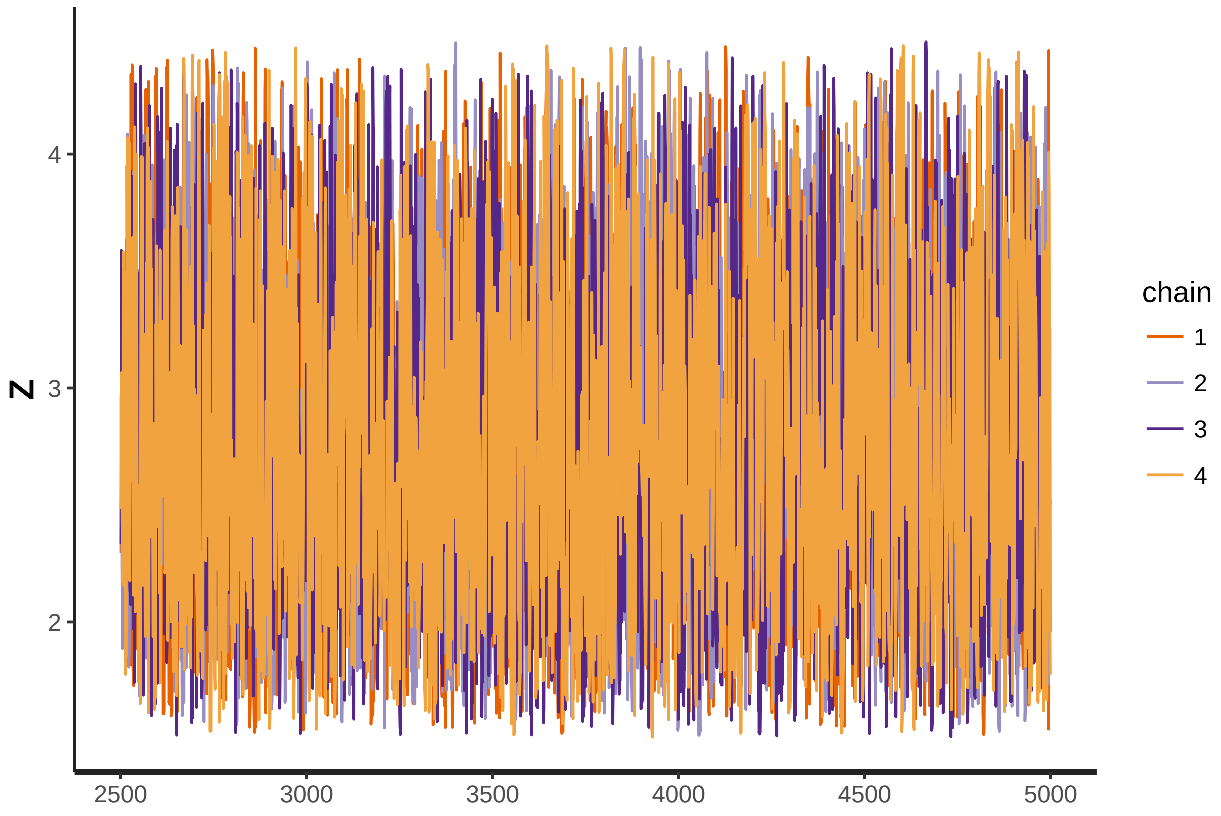
```
# init = list(
#   chain1 = list(Z = 7/3.2, X = 3.2, Y = 7),
#   chain2 = list(Z = 6.5/3.5,X = 3.5, Y = 6.5),
#   chain3 = list(Z = 7/3, X = 3, Y = 7),
#   chain4 = list(Z = 8/2.1, X = 2.1, Y = 8)
# )
######################
fixed_alpha_run
```

```
## Inference for Stan model: 3eb4164abdb17b48f9216db49b999764.
## 4 chains, each with iter=5000; warmup=2500; thin=1;
## post-warmup draws per chain=2500, total post-warmup draws=10000.
##
##       mean se_mean   sd  2.5%   25%   50%   75% 97.5% n_eff Rhat
## Z     2.77    0.01 0.73  1.64  2.17  2.69  3.33  4.24  3508    1
## lp__ -2.03    0.01 0.79 -4.32 -2.25 -1.71 -1.50 -1.46  3621    1
##
## Samples were drawn using NUTS(diag_e) at Tue Jan 23 13:43:54 2018.
## For each parameter, n_eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor on split chains (at
## convergence, Rhat=1).
```

```
pairs(fixed_alpha_run)
```

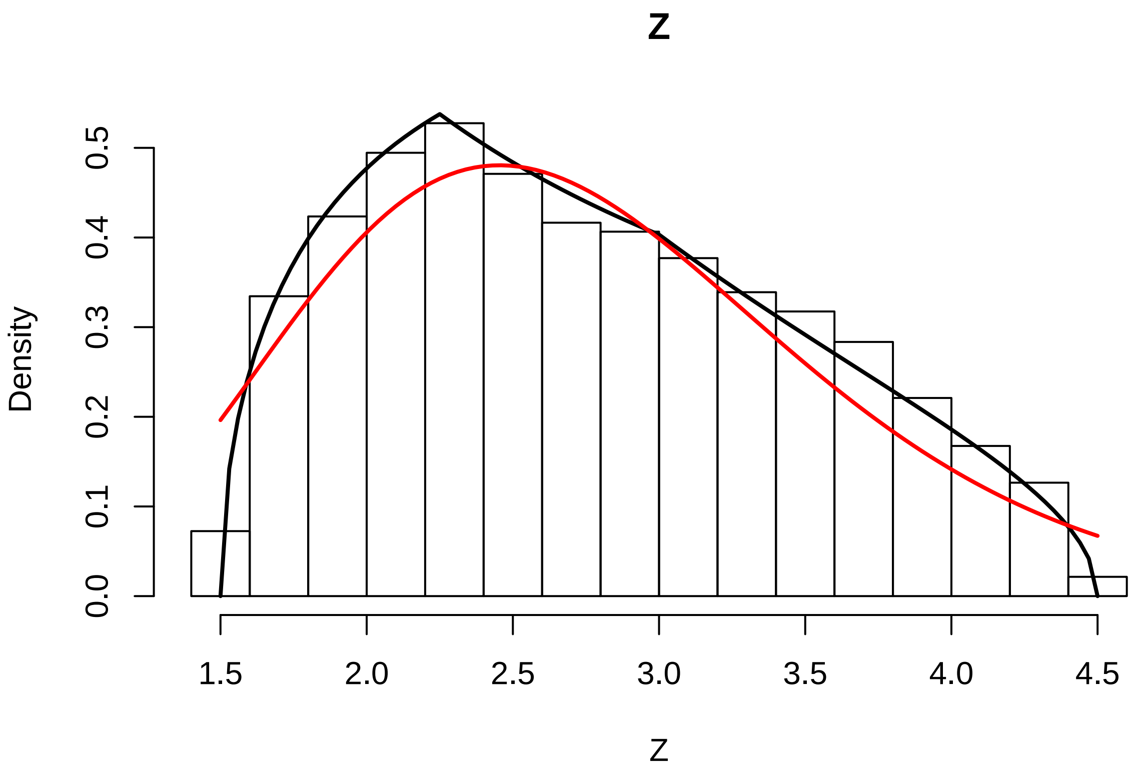


```
stan_trace(fixed_alpha_run)
```



```
source("../code/pooling_aux.r")
devtools::source_url("https://raw.githubusercontent.com/maxbiostat/CODE/b8473512151b0d205fd843bc291e45e
## SHA-1 hash of file is a1dfa5d771fdeb74b331d331462857416746eb31
```

```
dZ_exact <- function(x) dpoolnorm.positive(x = x, D = list(function(x) {dunif(x, 0, 5)},
                                                         function(x) {analytic_Z(x, ax = 2, bx = 4, a
                                           alphas = c(.5, .5)
)
dZ_approx <- function(x) dpoolnorm.positive(x = x, D = list(function(x) {dunif(x, 0, 5)},
                                                          function(x) {dgamma(x, 18.3, 7.05)}),
                                            alphas = c(.5, .5)
)

Z_samples <- extract(fixed_alpha_run, 'Z')$Z
hist(Z_samples,
     probability = TRUE, main = "Z", xlab = expression(Z))
curve(dZ_exact, 1.5, 4.5, lwd = 2, add = TRUE)
curve(dZ_approx, 1.5, 4.5, lwd = 2, col = 2, add = TRUE)
```



```
mu <- integrate( function(x) x * dZ_exact(x), 0 , Inf)
sq <- integrate( function(x) x^2 * dZ_exact(x), 0 , Inf)

mean(Z_samples); mu$value
```

```
## [1] 2.771368
```

```
## [1] 2.744048
```

```
var(Z_samples); sq$value-mu$value^2
```

```
## [1] 0.5346479
```

```
## [1] 0.5224814
```