Stan R Code

Nikita Kohli

2022-12-05

```
#Set work directory
setwd("C:/Users/nikit/Downloads/Rstan")

#Installation of RStan
#install.packages("StanHeaders", repos = c("https://mc-stan.org/r-packages/", getOption("repos")))
#install.packages("rstan", repos = c("https://mc-stan.org/r-packages/", getOption("repos")))

#Loading libraries
library(tidyr)
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(rstan)
```

```
## Loading required package: StanHeaders

##
## rstan version 2.26.13 (Stan version 2.26.1)

## 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)
## For within-chain threading using `reduce_sum()` or `map_rect()` Stan functions,
## change `threads_per_chain` option:
## rstan_options(threads_per_chain = 1)

## Do not specify '-march=native' in 'LOCAL_CPPFLAGS' or a Makevars file

##
## Attaching package: 'rstan'

## The following object is masked from 'package:tidyr':
##
##     extract
```

```r
library(tibble)
library(readr)
library(quadprog)
```

```r
#Demo: Simple iid Gaussian model
# Simulating some data
n = 100
y = rnorm(n,1.6,0.2)

# Running stan code
model = stan_model("demo.stan")
```

```
## Warning in readLines(file, warn = TRUE): incomplete final line found on 'C:
## \Users\nikit\Downloads\Rstan\demo.stan'
```

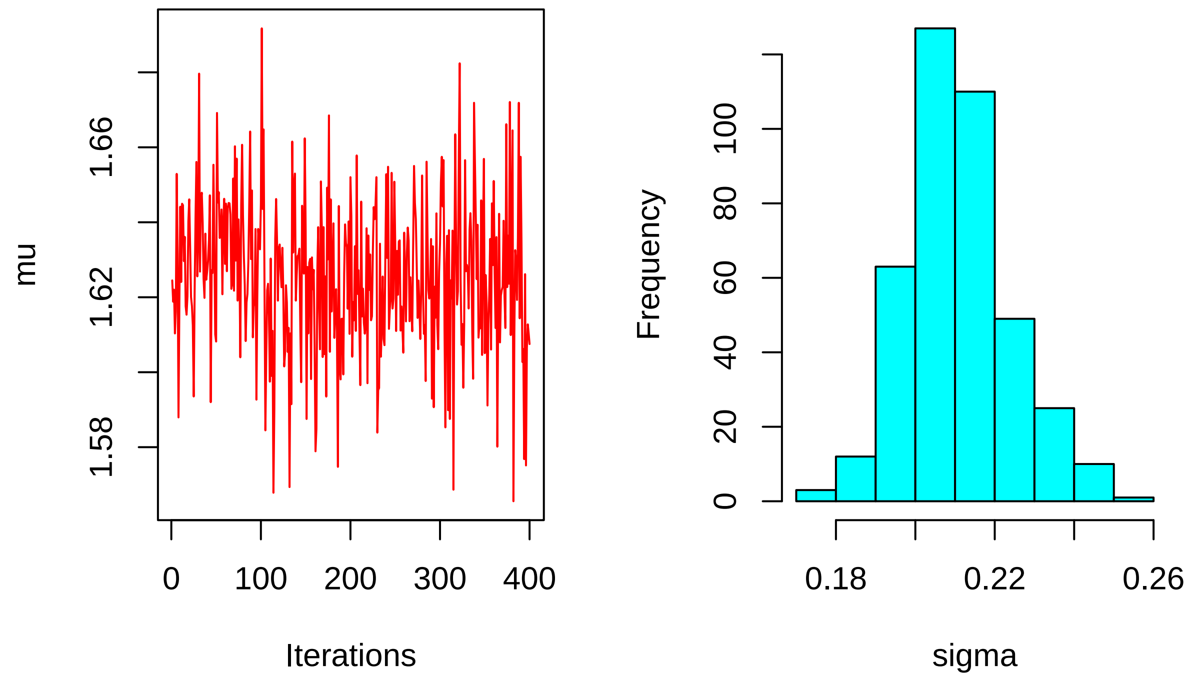
```r
fit = sampling(model,
               list(n=n,y=y),
               iter=200,
               chains=4,
               algorithm = "HMC",
               cores=4)
```

```
## Warning: The largest R-hat is 1.06, indicating chains have not mixed.
## Running the chains for more iterations may help. See
## https://mc-stan.org/misc/warnings.html#r-hat

## Warning: Bulk Effective Samples Size (ESS) is too low, indicating posterior means and medians may be
## Running the chains for more iterations may help. See
## https://mc-stan.org/misc/warnings.html#bulk-ess

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

```r
print(fit)
```

```
## Inference for Stan model: anon_model.
## 4 chains, each with iter=200; warmup=100; thin=1;
## post-warmup draws per chain=100, total post-warmup draws=400.
##
##         mean se_mean   sd   2.5%    25%    50%    75%  97.5% n_eff Rhat
## mu      1.63    0.00 0.02   1.58   1.61   1.63   1.64   1.66   146 1.03
## sigma   0.21    0.00 0.01   0.19   0.20   0.21   0.22   0.24   909 0.99
## lp__  104.46    0.07 0.91 101.90 104.12 104.80 105.08 105.28   152 1.05
##
## Samples were drawn using HMC(diag_e) at Tue Dec  6 14:09:07 2022.
## 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).
```

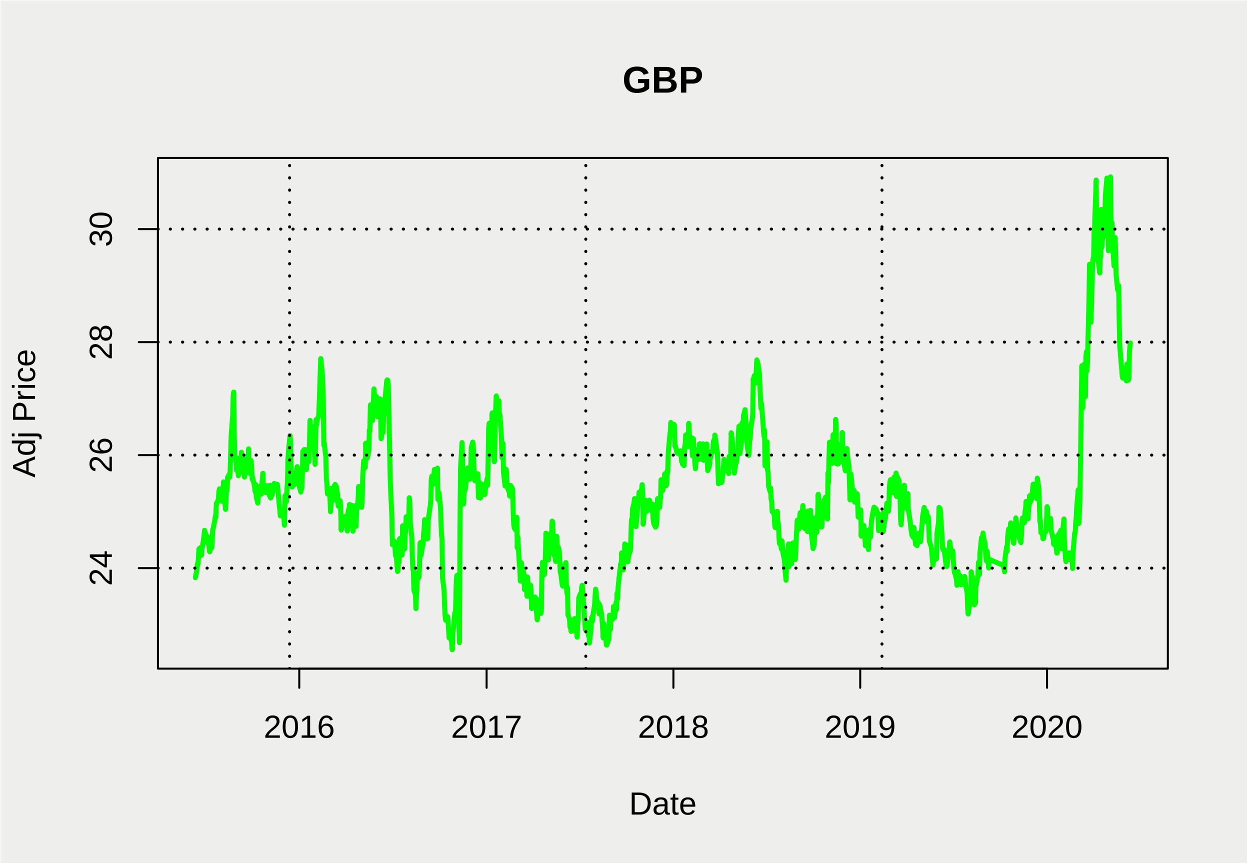
```r
params = extract(fit)
par(mfrow=c(1,2))
ts.plot(params$mu,xlab="Iterations",ylab="mu", col = "red") #inc_warmup = TRUE includes the burn-in val
hist(params$sigma,main="",xlab="sigma", col = "cyan")
```



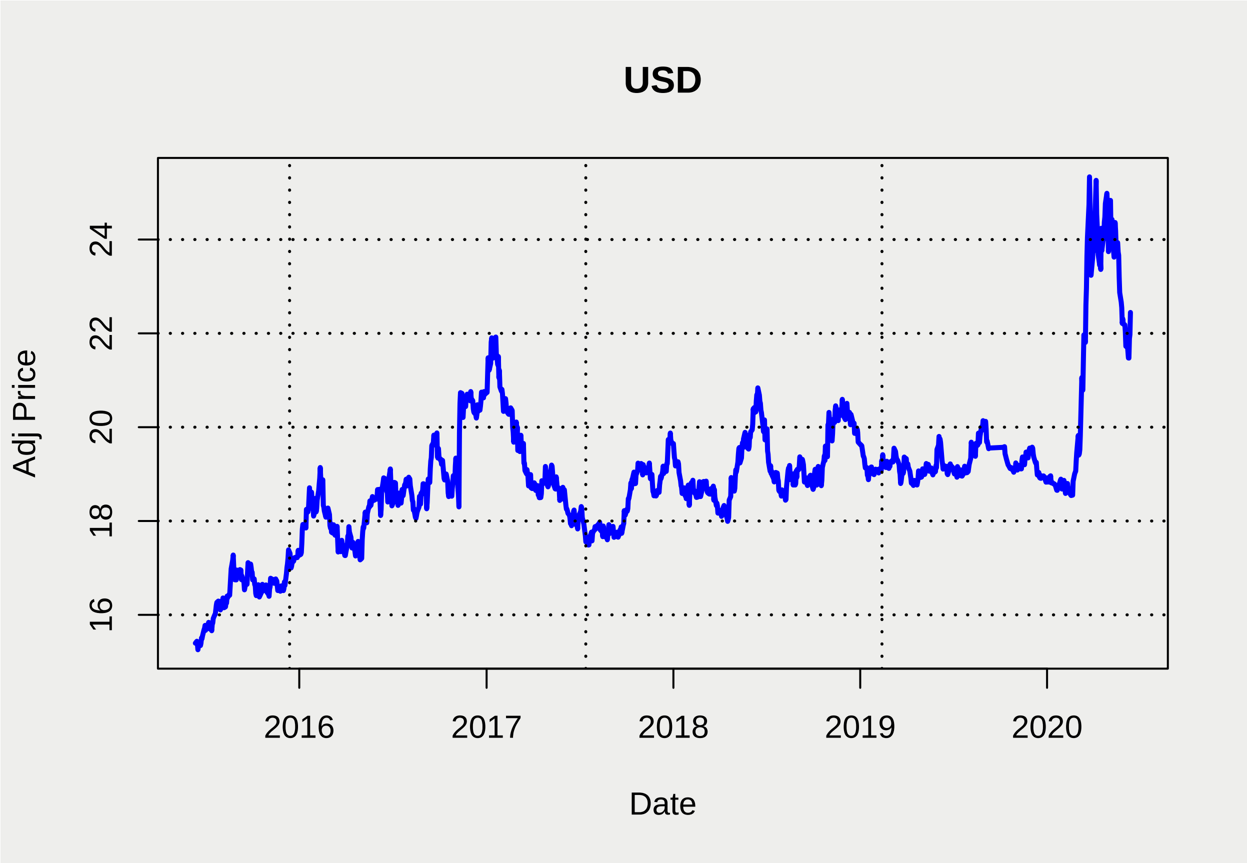

```r
#Portfolio Optimization
#Original Time Series Data
ex_rates <- read.csv('exchange_rates.csv', sep = ",", )
head(ex_rates)
```

```
##         Date eur_mxn usd_mxn gbp_mxn
## 1 2015-06-12 17.2722 15.3955 23.8298
## 2 2015-06-15 17.2819 15.4373 23.9679
## 3 2015-06-16 17.4038 15.3880 24.0718
## 4 2015-06-17 17.2966 15.2559 24.0628
## 5 2015-06-18 17.3400 15.3314 24.1718
## 6 2015-06-19 17.4310 15.3413 24.3398
```

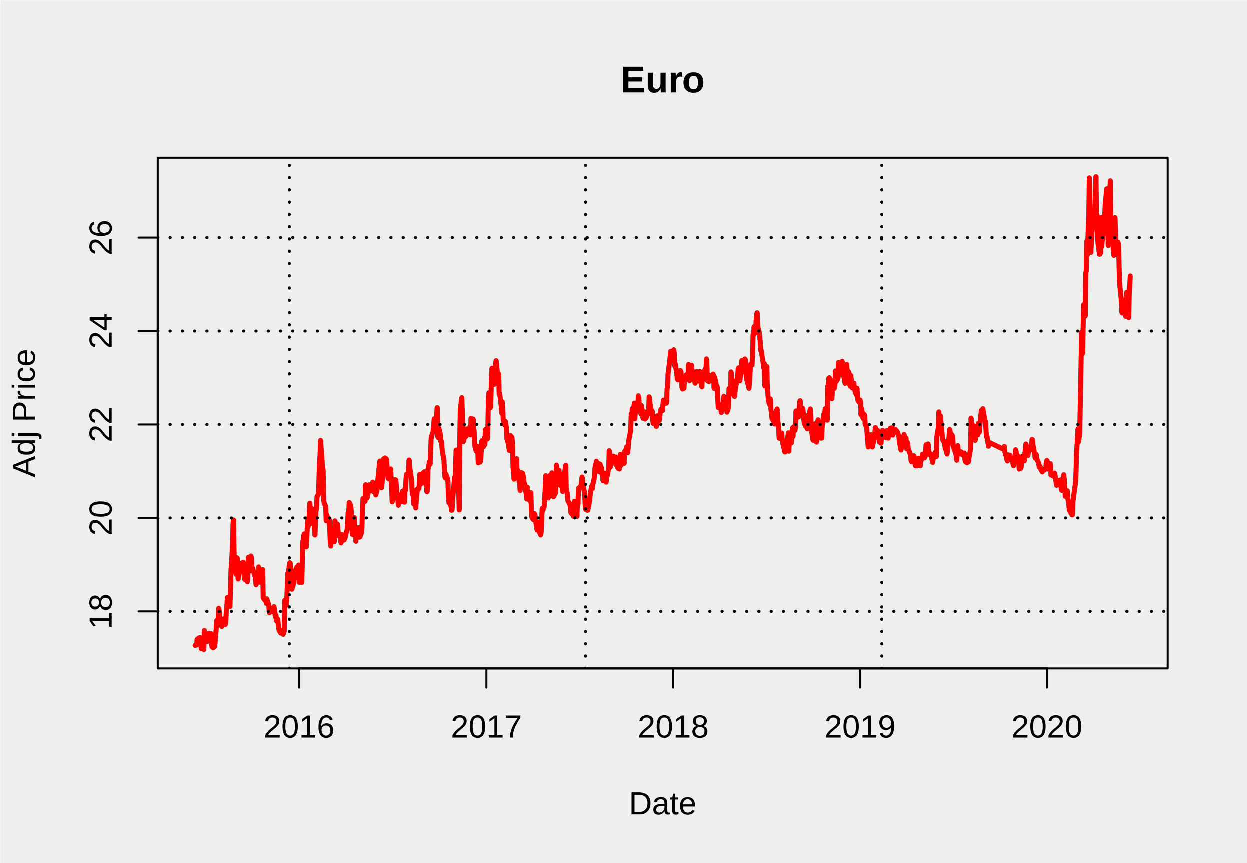
```r
#Visualizing the data set
#Converting to plot the time series plot
x = strptime(ex_rates$Date, '%Y-%m-%d')
par(bg = '#EEEEEC')
plot(x,
     ex_rates$gbp_mxn,
     col = 'green',
     type = 'l',
     lwd = 2.5,
     ylab = 'Adj Price',
     xlab = 'Date',
     main = 'GBP')
grid(col = 'black', lwd = 1.5)
```




```
plot(x,
     ex_rates$usd_mxn,
     col = 'blue',
     type = 'l',
     lwd = 2.5,
     ylab = 'Adj Price',
     xlab = 'Date',
     main = 'USD')
grid(col = 'black', lwd = 1.5)
```




```
plot(x,
     ex_rates$eur_mxn,
     col = 'red',
     type = 'l',
     lwd = 2.5,
     ylab = 'Adj Price',
     xlab = 'Date',
     main = 'Euro')
grid(col = 'black', lwd = 1.5)
```




```r
#Returns data (by taking log)
ret <- read.csv('log_ret.csv')
head(ret)
```

```
##         eur_mxn       usd_mxn       gbp_mxn
## 1  0.0005614385  0.0027113999  0.0057784951
## 2  0.0070288047 -0.0031986739  0.0043256375
## 3 -0.0061785653 -0.0086216714 -0.0003739929
## 4  0.0025060210  0.0049366994  0.0045196679
## 5  0.0052342589  0.0006455252  0.0069261648
## 6  0.0004645814  0.0002541834  0.0009732811
```

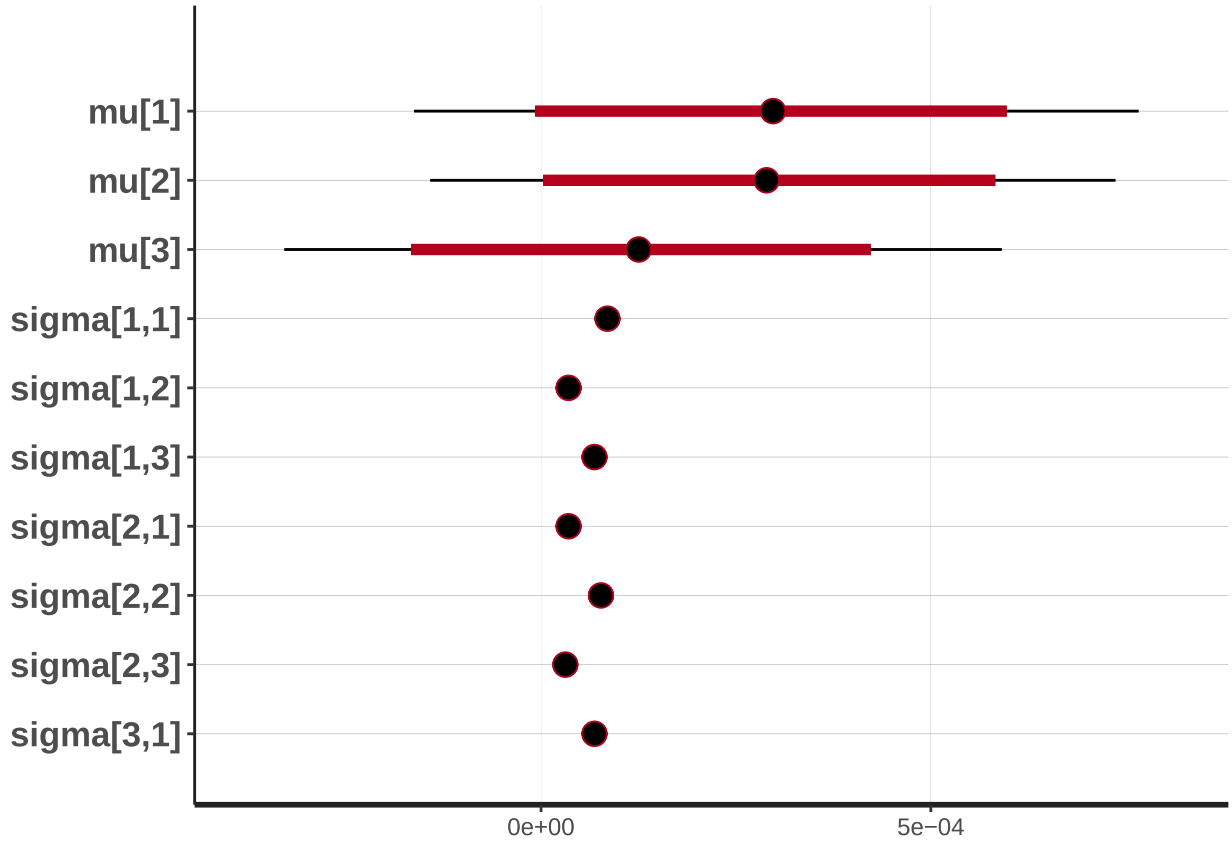
```r
#Mean of all the returns
mean_ret <- apply(ret, 2, mean)

#Covariance matrix
cov_mat <- cov(ret)

#Data for STAN
T <- nrow(ret)
N <- ncol(ret)
nu <- 12
tau <- 200 #considering this as 1/6th of T
data_stan <- list(
  T = T,
  N = N,
  nu = nu,
```

```r
  tau = 200,
  eta = mean_ret,
  R = as.matrix(ret),
  omega = cov_mat * (nu - N -1)
)

#Fitting the model
fit <- stan(
  file = "bay_port.stan",
  data = data_stan,
  chains = 4,
  warmup = 1000,
  iter = 2000,
  cores = 2
)
#Save the fitted model for future use because running takes a while
saveRDS(fit, 'stan_fit.rds')

fit <-readRDS('stan_fit.rds')

#Some diagnostics for posterior predictive values
plot(fit)
```

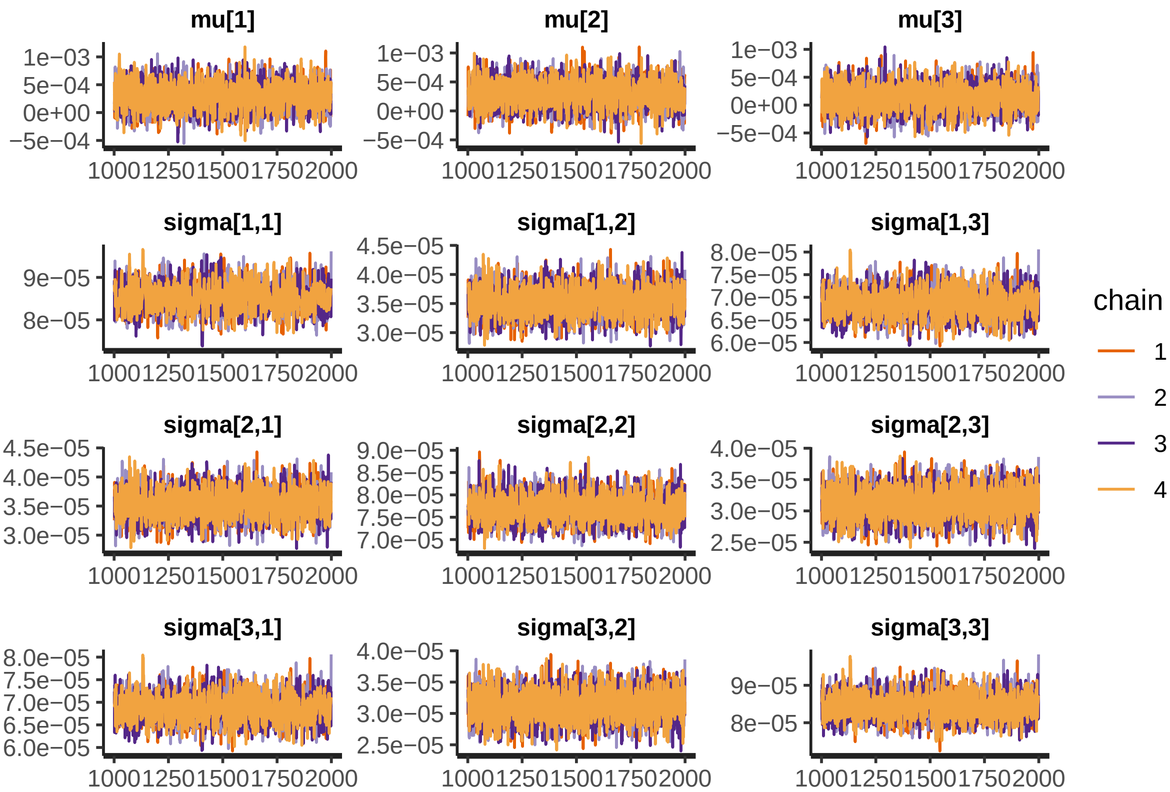
```
## 'pars' not specified. Showing first 10 parameters by default.
```

```
## ci_level: 0.8 (80% intervals)
```

```
## outer_level: 0.95 (95% intervals)
```

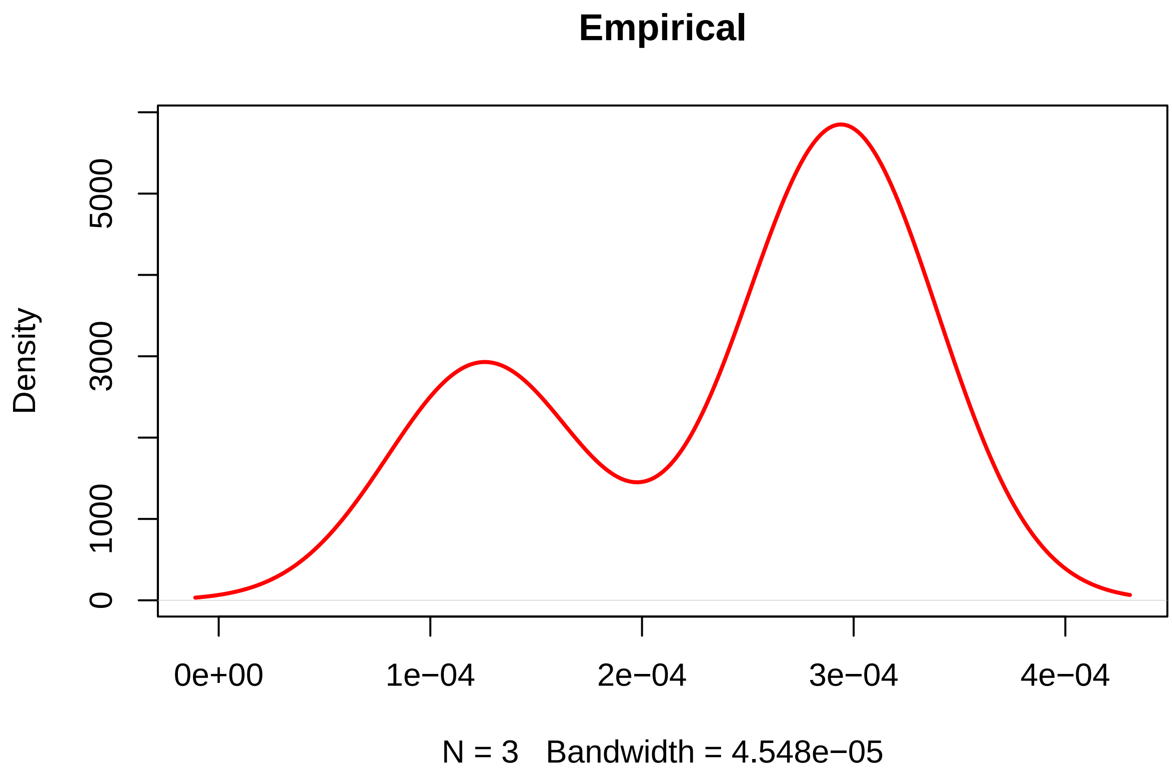



```
traceplot(fit, nrow = 4, pars = c('mu', 'sigma'))
```

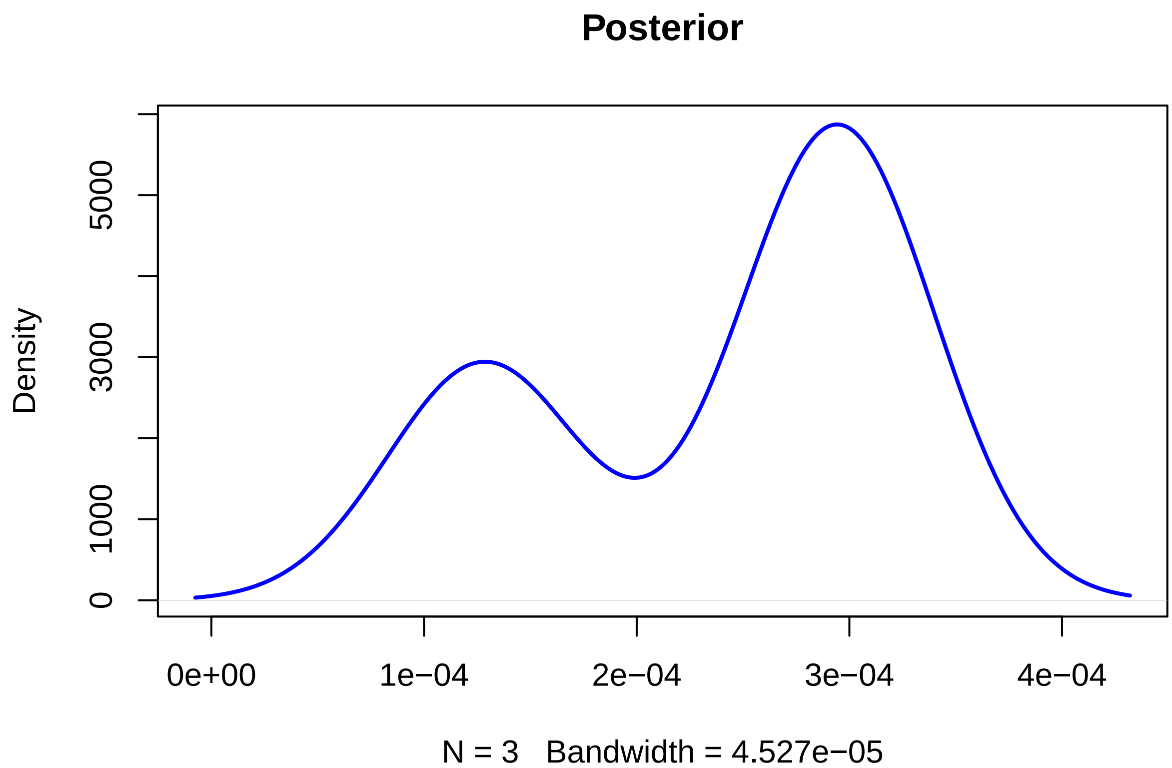



```
#Extract draws from the posterior
list_of_draws <- extract(fit)

#draws from the posterior distribution of sigma
sigma_post <- list_of_draws$sigma
sigma_post_new <- apply(sigma_post, c(2,3),mean)

#draws from the posterior distribution of mu
mu_post <- list_of_draws$mu
mu_post_new <- apply(mu_post, c(2), mean)
```

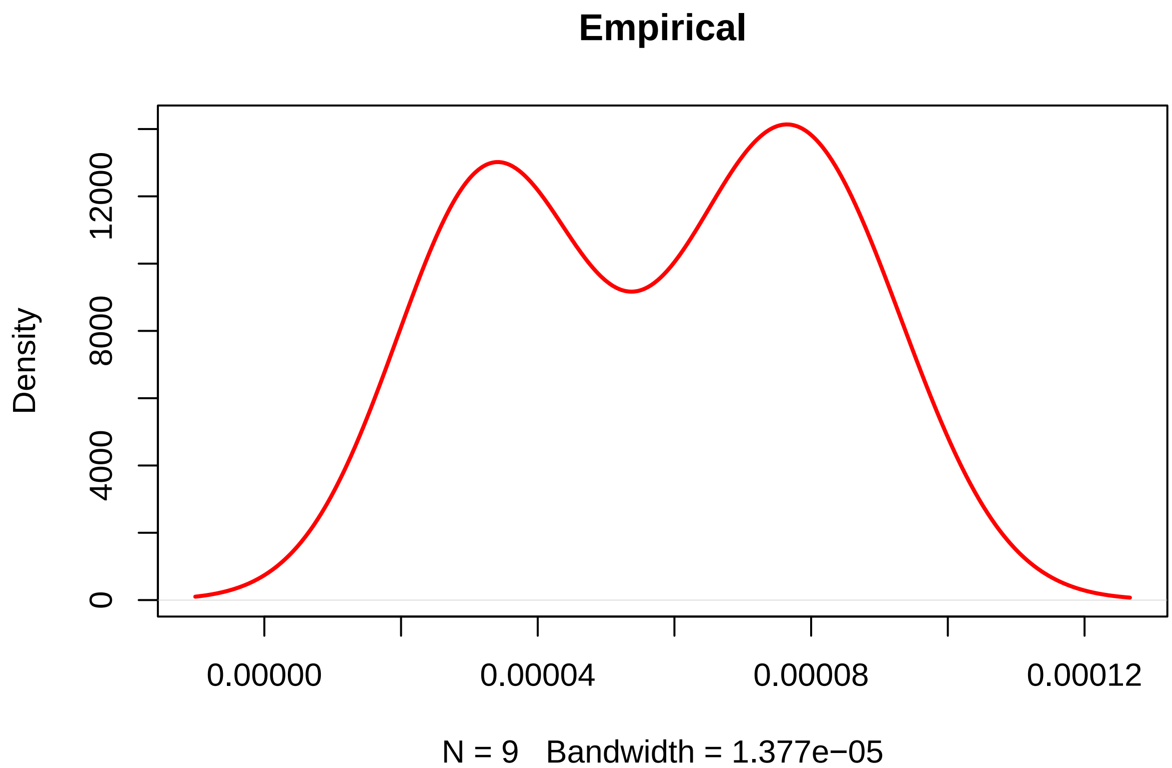
```
#For mean of the returns
#Prior distribution
plot(density(mean_ret),
      lwd = 2, col = "red",
     main = "Empirical")
```



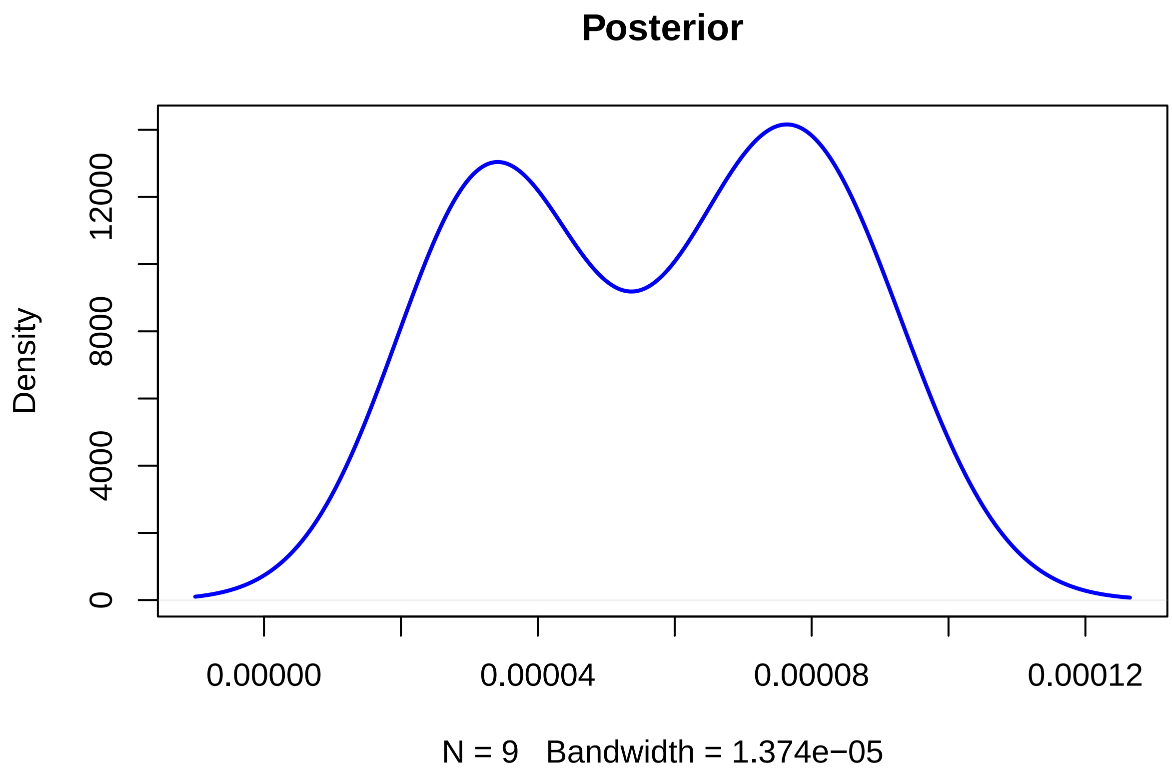

```
#Posterior distribution
plot(density(mu_post_new),
lwd = 2, col = 'blue',
main = "Posterior")
```




```
#For the variances of the returns
#Prior distribution
plot(density(cov_mat),
      lwd = 2, col = "red",
     main = "Empirical")
```




```
#Posterior distribution
plot(density(sigma_post_new),
lwd = 2, col = 'blue',
main = "Posterior")
```




```
M <- nrow(mu_post) #considering the whole data set

#target (annual) return (varying this to find optimal value)
annual_ret <- seq(0.02, 0.20, le = 100)

#This tibble will store the optimal weights
weights_opt <- tibble(eur = NULL,
                      usd = NULL,
                      gbp = NULL,
                      target = NULL,
                      sd = NULL)

#Number of assets
N <- ncol(mu_post)

#Solves the optimization and averages the solutions
#found for every target level
for(i in 1:M){
  #initialization
  mu_tib <-  c()
  var_opt <- c()

  #draw from mu
  mu_draw <- mu_post[i,]

  #draw from sigma
```

```r
  sig_draw <- sigma_post[i, ,]

  #Solves the optimization problem for each target value
  for(j in annual_ret){

    #Initial weights
    w = 252

    A <- matrix(0, nrow = N,ncol = 2)
    #sum of weights equals 1
    A[,1] <- 1

    #the target return constraint
    A[,2] <- mu_draw * w

    b0 <- c(1, j)
    sol <- solve.QP(2 * w * sig_draw, #due to the objective function
                    dvec = rep(0, N), #vector in the quadratic function
                    Amat = A, #matrix for constraints
                    bvec = b0, #default vector for zero in intercept
                    meq = 2) #number of equality constraints
    var_opt <- c(var_opt, sol$value )

    #update weights_opt tibble
    row <- c(sol$solution, j, sol$value)
    weights_opt <- rbind(weights_opt, row)
  }

}
names(weights_opt) <- c('eur',
                        'usd',
                        'gbp',
                        'target',
                        'std')

write_csv(weights_opt, "opt_weights.csv")

#tibble with the optimal weights
weights_opt <- read.csv('opt_weights.csv')
head(weights_opt)
```

```
##          eur       usd      gbp     target        std
## 1 -0.6640669 0.2932320 1.370835 0.02000000 0.02290541
## 2 -0.6448165 0.2984505 1.346366 0.02181818 0.02252433
## 3 -0.6255660 0.3036690 1.321897 0.02363636 0.02215237
## 4 -0.6063156 0.3088875 1.297428 0.02545455 0.02178953
## 5 -0.5870651 0.3141060 1.272959 0.02727273 0.02143581
## 6 -0.5678146 0.3193245 1.248490 0.02909091 0.02109121
```

```r
#Averages the values for each target value
mean_opt_w <- weights_opt %>%
  group_by(target) %>%
  mutate(mean_eur = mean(eur),
         mean_usd = mean(usd),
```

```r
         mean_gbp = mean(gbp),
         mean_std = mean(std)) %>%
  select(mean_eur,
         mean_usd,
         mean_gbp,
         target,
         mean_std) %>%
  unique()

par(bg = '#EEEEEC',
    mfcol = c(1,1))

#Plotting the percentage of the values
plot(100 * mean_opt_w$mean_std,
     100 * mean_opt_w$target,
     col = 'red', lwd = 2.5,
     main = 'Average Efficient Frontier',
     xlab = 'Standard Deviation %',
     ylab = 'Expected return %',
     type = 'l')
grid(col = 'black', lwd = 1.5)
```