2022-11_FET_BM_Homework 1

Fabio Enrico Traverso

2022-11-07

Task 1

The setup is such that with the echo=TRUE option all the code will be printed.

To perform the homework, the following packages are used:

```
library(bayesm)
library(data.table)
library(knitr)
library(flextable)
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.2 --
## v ggplot2 3.3.6      v purrr   0.3.5
## v tibble  3.1.8      v dplyr   1.0.10
## v tidyr   1.2.1      v stringr 1.4.1
## v readr   2.1.3      v forcats 0.5.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::between()   masks data.table::between()
## x purrr::compose()   masks flextable::compose()
## x dplyr::filter()    masks stats::filter()
## x dplyr::first()     masks data.table::first()
## x dplyr::lag()       masks stats::lag()
## x dplyr::last()      masks data.table::last()
## x purrr::transpose() masks data.table::transpose()
```

```
set.seed(2516)
set_flextable_defaults(fonts_ignore=TRUE)
library(compiler)
runireg_c=cmpfun(runireg) # a bit faster
```

To loop over the number of observations, I consructed a vector of increasing numbers.

```
nobs <- c(5,10,100,1000,10000)

# storing matrices
b_b1 <- matrix(0, length(nobs), 3) # m rows, 3 columns
sd_b1 <- matrix(0, length(nobs), 3)
b_ols <- b_b1
sd_ols <- b_b1

iteration <- 0
for (n in nobs){
  print(n)
  iteration <- iteration + 1
```

```r
  # Data Generating Process
  X <- cbind(rep(1,n),runif(n),runif(n))
  beta <- c(-1,3,6)
  y <- (X %*% beta + rnorm(n, mean = 0, sd = 1))# set up y

  # parameters of the simulation
  R <- 80000
  Mcmc1 <- list(R=R,keep=1, nprint = 0)
  Prior <- list(nu=0, ssq=0)

  # run regressions
  ## bayes
  out_b1 <- runireg_c(Data=list(y=y,X=X), Prior = Prior,Mcmc=Mcmc1)
  b_draw1 <-out_b1$betadraw
  # for every sample size the mean and the standard deviation are computed
  b_b1[iteration,] <- apply(b_draw1, 2, mean)
  sd_b1[iteration,] <- apply(b_draw1, 2, sd)

  ## ols
  out_ols <- lm(y ~ X[,-1])
  summary(out_ols)
  b_ols[iteration,] <- matrix(out_ols$coefficients, 1, 3)
  sd_ols[iteration,] <- matrix(sqrt(diag(vcov(out_ols))), 1, 3)
}
```

n	b1	sd_b1	b2	sd_b2	b3	sd_b3	ols1	sd_ols1	ols2	sd_ols2	ols3	sd_ols3
5	0.8138	0.9320	0.8634	0.9457	3.9846	1.1027	0.7276	1.0770	0.9001	1.0876	4.1042	1.2662
10	-1.3110	1.1061	3.8865	1.4348	6.5866	2.1180	-1.4903	1.1623	4.0626	1.4990	6.9303	2.2264
100	-1.5812	0.2405	3.6514	0.3239	6.3709	0.3186	-1.5874	0.2412	3.6552	0.3257	6.3786	0.3196
1,000	-1.0264	0.0828	3.1496	0.1123	5.9221	0.1088	-1.0272	0.0833	3.1495	0.1118	5.9232	0.1095
10,000	-0.9764	0.0265	3.0252	0.0341	5.9153	0.0344	-0.9764	0.0264	3.0253	0.0341	5.9152	0.0344

Both estimators converge to the true parameters, although the bayesian one presents a lower standard error, which implies it is more efficient than OLS.

Task 2

We expect that due to multicollinearity, the OLS estimator is not feasible in this setup. In particular, the matrix $X'X$ is not invertible. However, the Bayesian estimator does not require the $X'X$ to be invertible and will likely estimate a virtually identical distribution for the two coefficients.

```r
nobs <- 20000
noby <- n
b_b <- matrix(0, length(nobs), 3)
sd_b <- matrix(0, length(nobs), 3)
b_ols <- b_b
sd_ols <- b_b

#Data Generating Process
hilf= runif(nobs)
```

```r
X <- cbind(rep(1,n),hilf, hilf)
beta <- c(-1,3,6)
y <- (X %*% beta + rnorm(n, mean = 0, sd = 1))# set up y

# set up sampling
R <- 80000 # as in the lecture
Mcmc1 <- list(R=R,keep=1, nprint = 0)
Prior <- list(nu=0, ssq=0) # beta = 0 as default

# run regressions
## bayes
out_b <- runireg_c(Data=list(y=y,X=X), Prior = Prior,Mcmc=Mcmc1)
b_draw <-out_b$betadraw
b_b[1,] <- apply(b_draw, 2, mean) # get mean for each coeff.
sd_b[1,] <- apply(b_draw, 2, sd) # standard deviation

## ols
out_ols <- lm(y ~ X[,-1]) # without intercept in x, is added automatically
summary(out_ols)
b_ols[1,] <- matrix(out_ols$coefficients, 1, 3)
sd_ols[1,] <- matrix(sqrt(diag(vcov(out_ols))), 1, 3)
```

As discussed above, multicollinearity is “solved” bz the program by eliminating one of the two regressors.

```r
res <- data.frame(n = nobs, b1 = b_b[,1], sd_b1 = sd_b[,1],
                            b2 = b_b[,2], sd_b2 = sd_b[,2],
                            b3 = b_b[,3], sd_b3 = sd_b[,3],
                            ols1 = b_ols[,1], sd_ols1 = sd_ols[,1],
                            ols2 = b_ols[,2], sd_ols2 = sd_ols[,2],
                            ols3 = b_ols[,3], sd_ols3 = sd_ols[,3])

round(res,4) %>% regulartable() %>% autofit() %>% fit_to_width(max_width = 6.5)
```

n	b1	sd_b1	b2	sd_b2	b3	sd_b3	ols1	sd_ols1	ols2	sd_ols2	ols3	sd_ols3
20,000	-1.0099	0.014	4.5457	7.0609	4.497	7.0608	-1.01	0.014	9.0429	0.0244		

```r
kable(summary(b_draw), digits = 4)
```

Summary of Posterior Marginal Distributions Moments mean std dev num se rel eff sam size 1 -1.0 0.014 5.1e-05 0.94 72000 2 4.5 7.068 2.6e-02 0.99 72000 3 4.5 7.068 2.6e-02 0.99 72000

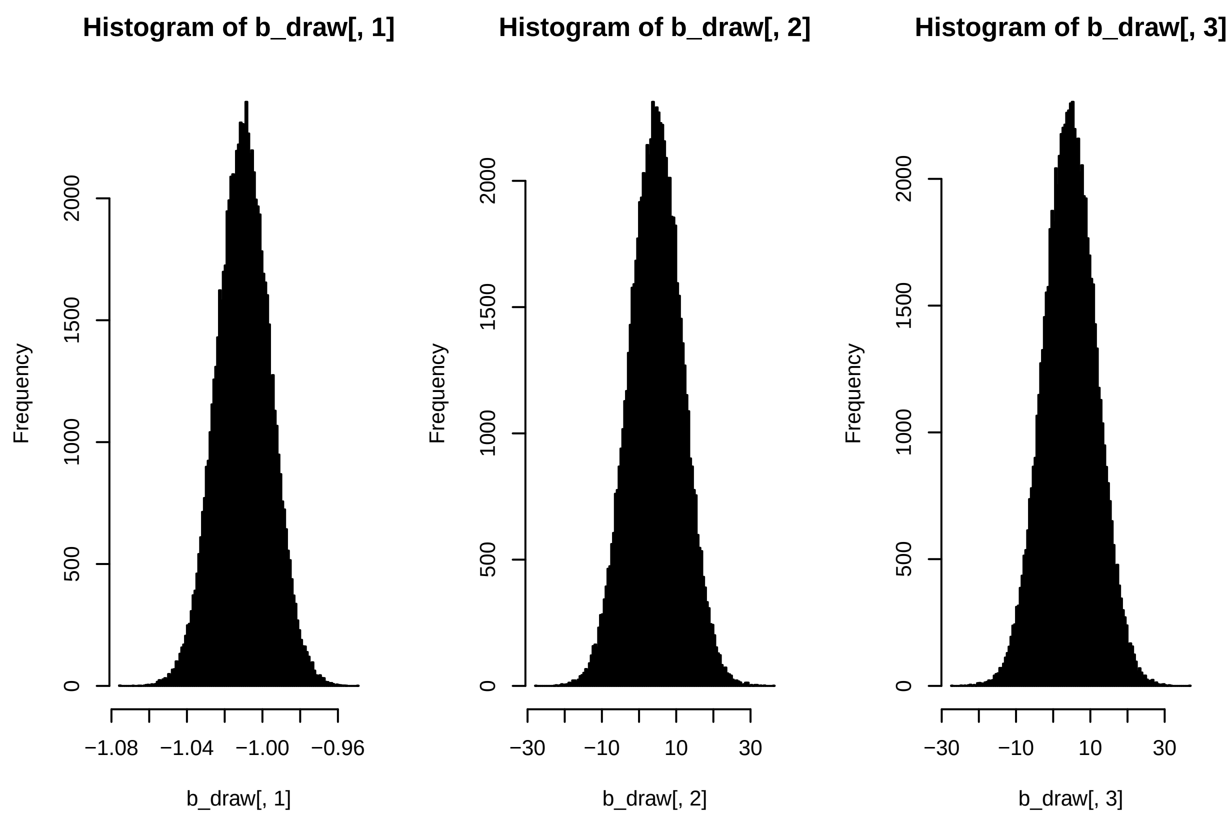
Quantiles 2.5% 5% 50% 95% 97.5% 1 -1.0 -1.0 -1.0 -0.99 -0.98 2 -9.4 -7.1 4.6 16.13 18.41 3 -9.4 -7.1 4.5 16.15 18.42 based on 72000 valid draws (burn-in=8000)

mean	std dev	num se	rel eff	sam size
-1.0099	0.0140	0.0001	0.9387	72000
4.5433	7.0681	0.0261	0.9852	72000
4.4994	7.0679	0.0261	0.9854	72000

```r
kable(t(apply(b_draw,2,quantile)), digits = 4)
```

0%	25%	50%	75%	100%
-1.0756	-1.0193	-1.0099	-1.0005	-0.9499
-27.5110	-0.2165	4.5723	9.3227	36.1505
-27.0989	-0.2784	4.4680	9.2601	36.5901

Finally, the distribution of the bayesian estimator confirms the hypotheses: the perfectly collinear regression coefficients' posterior distributions are indeed close to identical.




```
## null device
##           1
```

Task 3

```r
nobs <- c(5,10,100,1000,10000) # different ns

# matrices in which results are stored
b_b3 <- matrix(0, length(nobs), 3) # m rows, 3 columns
sd_b3 <- matrix(0, length(nobs), 3)
b_pro <- b_b3
sd_pro <- b_b3
```

```r
iteration <- 0

for (n in nobs){
  print(n)
  iteration <- iteration + 1
  # set up DGP
  #n <- 5
  X <- cbind(rep(1,n),runif(n),runif(n))
  beta <- c(-1,3,6)
  y <- (X %*% beta + rnorm(n, mean = 0, sd = 1))# set up y
  y <- ifelse(y<0,0,1)

  # set up sampling
  R <- 80000 # as in the lecture
  Mcmc1 <- list(R=R,keep=1, nprint = 0)
  #Prior <- list(nu=0, ssq=0) # beta = 0 as default

  # run regressions
  ## bayes
  out_b3 <- runireg_c(Data=list(y=y,X=X),Mcmc=Mcmc1)
  b_draw3 <-out_b3$betadraw
  b_b3[iteration,] <- apply(b_draw3, 2, mean) # get mean for each coeff.
  sd_b3[iteration,] <- apply(b_draw3, 2, sd) # standard deviation

  ## ols
  out_pro <- glm(y ~ X[,-1], family=binomial(link="probit")) # without intercept in x, is added automat
  summary(out_pro)
  b_pro[iteration,] <- matrix(out_pro$coefficients, 1, 3)
  sd_pro[iteration,] <- matrix(sqrt(diag(vcov(out_pro))), 1, 3)
}
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

Having access to linear observations is better than the probit ones. Note infact that the estimates in the first table show that the bayesian estimators for the probit data are not consistent, whereas the probit estimator does converge, although not fast, to the true parameters. The comparison in Table 2 also highlights that the bayes linear estimator is also converging to the true parameters, whereas the probit one does not. Hence, it is preferrable to have access to linear data.

n	b1	sd_b1	b2	sd_b2	b3	sd_b3	pro1	sd_pro1	pro2	sd_pro2	pro3	sd_pro3
5	0.9663	0.0738	0.0417	0.1078	0.0122	0.0712	6.5528	33,333.3010	0.0000	48,966.1561	0.0000	31,258.7002
10	0.9944	0.0224	0.0032	0.0354	0.0070	0.0430	6.5528	13,093.3502	0.0000	20,479.1630	0.0000	25,151.8009
100	0.5150	0.0686	0.2914	0.0990	0.4554	0.1002	-1.9912	0.7065	3.5501	1.1881	7.1637	2.0896
1,000	0.7446	0.0176	0.1698	0.0229	0.2329	0.0228	-0.8760	0.2306	3.1026	0.4556	4.8595	0.6508
10,000	0.7260	0.0056	0.1705	0.0074	0.2627	0.0073	-1.1037	0.0769	2.9588	0.1402	6.3300	0.2658

n	b1.1	sd_b1.1	b2.1	sd_b2.1	b3.1	sd_b3.1	b1.3	sd_b1.3	b2.3	sd_b2.3	b3.3	sd_b3.3
5	0.8138	0.9320	0.8634	0.9457	3.9846	1.1027	0.9663	0.0738	0.0417	0.1078	0.0122	0.0712
10	-1.3110	1.1061	3.8865	1.4348	6.5866	2.1180	0.9944	0.0224	0.0032	0.0354	0.0070	0.0430
100	-1.5812	0.2405	3.6514	0.3239	6.3709	0.3186	0.5150	0.0686	0.2914	0.0990	0.4554	0.1002
1,000	-1.0264	0.0828	3.1496	0.1123	5.9221	0.1088	0.7446	0.0176	0.1698	0.0229	0.2329	0.0228
10,000	-0.9764	0.0265	3.0252	0.0341	5.9153	0.0344	0.7260	0.0056	0.1705	0.0074	0.2627	0.0073