Econometrics, Exercise 3

Demirol, Engelen & Kuschnig

26 Mai 2018

Part 1, Task 1

We have written a function as per requirements and will demonstrate its capabilities and usefulness over the course of this document. The definition is displayed here, for the full code please consult the appendix.

- **Parameters**: In its most basic form the function would only take the endogenous (Y) and exogenous (X) variables as parameters. Considering the importance of finding the right values for $\tau_{0,1}$ we should also be able to modify these parameters. According to most of the literature we have set the default value of τ_1 as $\tau_0 * 100$. For τ_0 we will utilise an automatic approach (see George, Sun & Ni 2006) as the default value, which we will discuss later. We also allow for varying the number of iterations - both in terms of values stored and burned. We adopt the non-informative priors from class but allow for fiddling by including them as parameters. Last but not least we provide an option for centering and scaling the endogenous and exogenous variables, which we will also discuss later.

- **Output**: For handing over all the results that we might need we use a list. The central element of the output should generally be the posterior inclusion probabilities, but mean values and variances of β and the mean σ^2 of the saved models should also prove helpful. We also include an element with metadata, i.e. parameters of the function call, with $\tau_{0,1}$ being the most interesting ones in the case of automatic estimation.

```
ssvs = function(y,
                X,
                tau0 = NULL,
                tau1 = tau0 * 100,
                save = 4000,
                burn = 1000,
                s_prior = 0.01,
                S_prior = 0.01,
                standardise = TRUE) {...}
```

Before testing the water we load the required dataset and decide on values for τ_0 and τ_1 (which we handle as a scalar for the former).

```
data(datafls)

y = datafls[, 1]
X = datafls[, 2:ncol(datafls)]

tau0s = c(1, 1e-2, 1e-5, 1e-15)
tau1_scales = c(100, 1000)
```

Then we run our function with the specified parameters. We do so three (ignoring the repetitions due to differing $\tau_{0,1}$) times:

- with the plain data, i.e. non-standardised

- with standardised data

- with standardised data and our automatic approach to setting $\tau_{0,1}$

```r
i = 1
for(scale in tau1_scales) {
  for(tau in tau0s) {
    pips[[i]] = ssvs(y, X, tau0 = tau, tau1 = tau * scale, standardise = FALSE)[[1]]
    names(pips)[i] = paste0(tau, " & *", scale)
    i = i + 1
  }
}

i = 1
for(scale in tau1_scales) {
  for(tau in tau0s) {
    pips_z[[i]] = ssvs(y, X, tau0 = tau, tau1 = tau * scale, standardise = TRUE)[[1]]
    names(pips_z)[i] = paste0(tau, " & *", scale)
    i = i + 1
  }
}

pips_auto = ssvs(y, X)[[1]]
```

We immediately take note of the fact that setting τ_0 to very small values (such as 1e-15) is probably best suited for simulating the outcome of a coinflip.
Pictured are the posterior inclusion probabilities of non-standardised and standardised run-throughs with τ_0 = 1e-15 and τ_1 = 100.

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.4467  0.4915  0.4988  0.5040  0.5130  0.6773
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.4680  0.4878  0.5025  0.5011  0.5082  0.5387
```

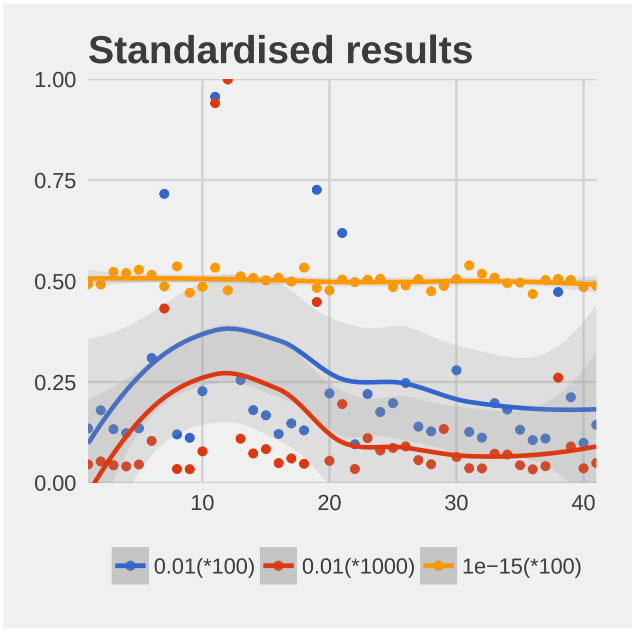
Furthermore we notice that choosing a bigger scalar for τ_1 leads to lower posterior inclusion probabilities, but comparable results overall. This is in line with the literature, where it is used to influence the scarcity of the model, i.e. amount of variables that should make it into the final model.
Pictured are the posterior inclusion probabilities of standardised run-throughs with $\tau_0 = 0.01$ and $\tau_1 = 100$ and 1000.

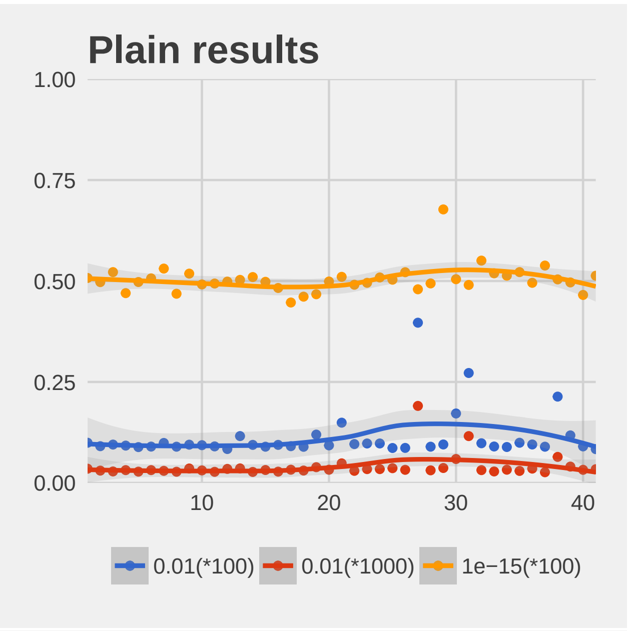
```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## 0.09575 0.12775 0.17575 0.25813 0.24750 0.99975
```

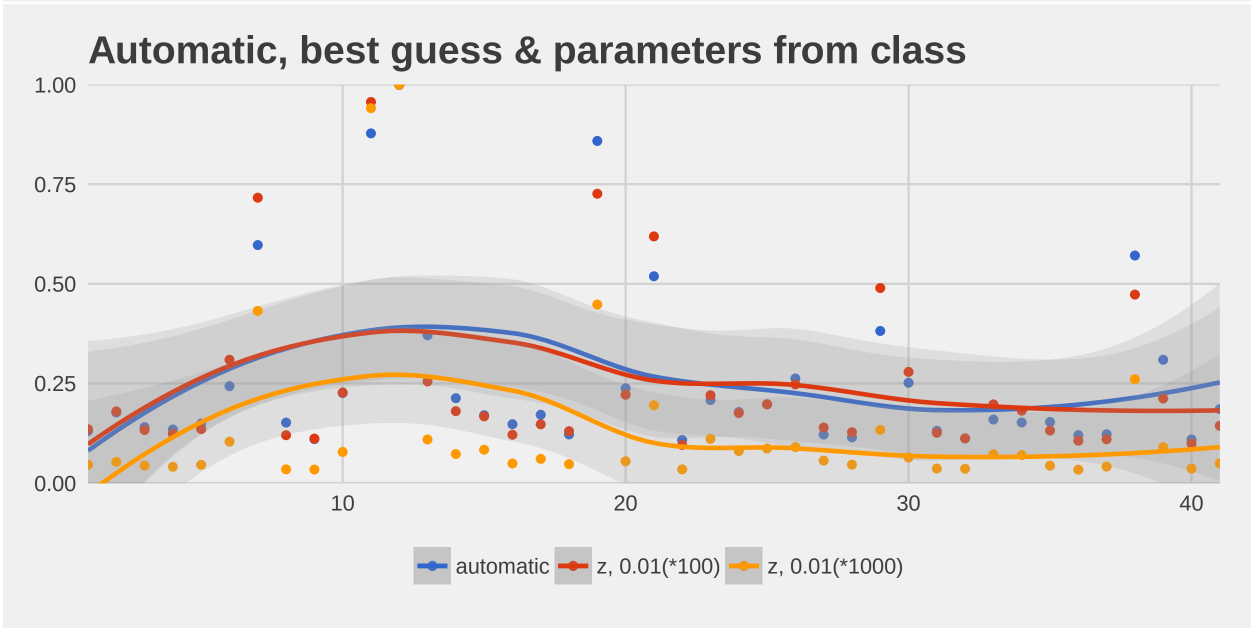
```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## 0.03375 0.04400 0.06075 0.13291 0.09050 0.99925
```

Processing the results graphically carries some further information:

- our coinflip-hypothesis is holding up
- standardising the data leads to more diverse and "nicer" posterior inclusion probabilities, with some variables making huge jumps
- the automatic approach of using OLS estimates of the variance for $\tau_{0,1}$ yields very similar results to our best guess of $\tau_0 = 0.01$ and $\tau_1 = 1$







Note: While fitting a line to the observations doesn't seem to make a lot of sense considering the data, we think it provides an easily noticeable and overall decent visual representation.

Part 2, Task 1

Here we try reproduce Table 11.1 (Koop 2015) with both our function and the BMS package (Feldkircher & Zeugner 2015). In the case of our function we will make use of the defaults, that is the automatic measure of τ_0 and standardisation.

```r
table_our = ssvs(y, X,
                 save = 200000, burn = 100000)
table_bms = bms(datafls,
                burn = 100000, iter = 200000, nmodel = 2000,
                mcmc = "bd", g = "UIP", mprior = "random",
                mprior.size = NA, user.int = T, start.value = NA,
                g.stats = T, logfile = F, logstep = 10000,
                force.full.ols = F, fixed.reg = numeric(0))
```

The results are definitely comparable - there's quite a bit of variation around the medium inclusion probabilities, but we observe a certain consensus on variables with very low and very high posterior inclusion probabilities. The posterior means are (apart from signs) not very similar - most likely due to the way we implemented standardisation. Due to us choosing τ_0 automatically the interpretation is not as straightforward, but we still see generally higher PIPs on the side of the bms function (which might suggest a lower scaling factor for τ_1). When comparing the PIPs graphically we see that our ssvs function places low PIPs around 20% on most variables, whereas the bms function produces a wider variety of PIPs.

	Custom			BMS		
Expl. Var.	**PIP**	**Post_Mean**	**Post_SD**	**PIP**	**Post_Mean**	**Post_SD**
Abslat	0.123	0.014	0.019	0.118	0.000	0.000
Age	0.144	-0.052	0.004	0.222	0.000	0.000
Area	0.113	0.003	0.007	0.098	0.000	0.000
BlMktPm	0.182	-0.075	0.004	0.472	-0.004	0.004
Brit	0.128	0.035	0.010	0.211	0.001	0.003
Buddha	0.172	0.071	0.005	0.315	0.003	0.006
Catholic	0.124	-0.020	0.018	0.187	0.000	0.003
CivlLib	0.155	-0.084	0.059	0.323	-0.001	0.001
Confucian	0.863	0.280	0.006	0.995	0.060	0.015
EcoOrg	0.210	0.093	0.006	0.571	0.001	0.001
English	0.150	-0.055	0.004	0.240	-0.002	0.004
EquipInv	0.568	0.211	0.008	0.911	0.135	0.064
EthnoL	0.239	0.106	0.008	0.396	0.005	0.007
Foreign	0.115	0.013	0.007	0.135	0.000	0.002
French	0.141	0.054	0.006	0.251	0.002	0.004
GDP60	0.999	-0.802	0.028	1.000	-0.016	0.003
HighEnroll	0.251	-0.117	0.014	0.389	-0.035	0.051
Hindu	0.516	-0.259	0.022	0.525	-0.035	0.042
Jewish	0.111	-0.012	0.004	0.097	0.000	0.004
LabForce	0.389	0.198	0.023	0.459	0.000	0.000
LatAmerica	0.234	-0.127	0.025	0.458	-0.005	0.006
LifeExp	0.903	0.542	0.035	0.971	0.001	0.000
Mining	0.373	0.131	0.004	0.734	0.029	0.022
Muslim	0.204	0.116	0.022	0.593	0.008	0.008
NequipInv	0.312	0.117	0.004	0.608	0.031	0.030
OutwarOr	0.149	-0.054	0.004	0.223	-0.001	0.002
PolRights	0.155	-0.084	0.048	0.230	0.000	0.001
Popg	0.132	0.030	0.018	0.115	0.009	0.072
PrExports	0.168	-0.075	0.014	0.178	-0.001	0.004
Protestants	0.206	-0.102	0.010	0.518	-0.006	0.007
PrScEnroll	0.223	0.116	0.020	0.403	0.008	0.011
PublEdupct	0.132	0.041	0.004	0.183	0.027	0.080
RevnCoup	0.113	0.011	0.004	0.095	0.000	0.001
RFEXDist	0.127	-0.035	0.006	0.164	0.000	0.000
RuleofLaw	0.248	0.123	0.015	0.657	0.008	0.007
Spanish	0.173	0.084	0.018	0.285	0.003	0.005
stdBMP	0.112	-0.014	0.004	0.116	0.000	0.000
SubSahara	0.592	-0.298	0.027	0.862	-0.015	0.008
WarDummy	0.149	-0.056	0.005	0.196	-0.001	0.002
WorkPop	0.124	-0.027	0.007	0.102	0.000	0.003
YrsOpen	0.164	0.069	0.013	0.364	0.004	0.007

Appendix

```r
#' @title Stochastic Search Variable Selection
#' @author Nikolas Kuschnig
#' @description Uses the Gibbs sampler to perform SSVS
#'
#' @param y The endogenous variable, must be convertible to a matrix.
#' @param X The explanatory variables, must be convertible to a matrix.
#' @param tau0 Number by which to scale an "unimportant" variable. Will use least squares estimates if
#' @param tau1 Number by which to scale an "important" variable. Defaults to 100 * tau0.
#' @param save Iterations to be stored for calculating means.
#' @param burn Iterations to be discarded before calculating means.
#' @param s_prior Prior accuracy, i.e. weight assigned to the prior
#' @param S_prior 1 / sigma^2
#' @param standardise A boolean determining whether to center and scale X & y.
#'
#' @return Returns a list containing the means of posterior: inclusion probability, mean and standard d
#' @export

ssvs = function(y,
                X,
                tau0 = NULL,
                tau1 = tau0 * 100,
                save = 4000,
                burn = 1000,
                s_prior = 0.01,
                S_prior = 0.01,
                standardise = TRUE) {
  y = matrix(y)
  X = as.matrix(X)

  N = nrow(y)
  K = ncol(X)

  # Standardise (i.e. center and scale) the data if desired
  if(standardise) {
    y = scale(y)
    X = scale(X)
  }

  # If no tau0 was supplied set up for OLS estimates, otherwise vectorise
  if(is.null(tau0)) {
    c0 = 0.1
    c1 = ifelse(length(tau1) == 0, 10, tau1)
  } else {
    tau0 = rep(tau0, K)
    tau1 = rep(tau1, K)
  }

  # get OLS stuff
  OLS = solve(crossprod(X)) %*% crossprod(X, y)
  SSE = as.numeric(crossprod(y - X %*% OLS))
  sigma_draw = as.numeric(SSE / (N - K))
```

```
V_beta_draw = sigma_draw * solve(crossprod(X))

# use OLS estimates for tau if no tau0 was supplied
if(is.null(tau0)) {
  tau0 <- c0 * sqrt(diag(V_beta_draw))
  tau1 <- c1 * sqrt(diag(V_beta_draw))
}

# Set up gamma, and storage matrices
gamma = matrix(1, K, 1)
V_prior = diag(as.numeric(gamma * tau1 + (1 - gamma) * tau0))

alpha_store = matrix(NA, save, K)
sigma_store = matrix(NA, save, 1)
V_beta_store = matrix(NA, save, K)
gamma_store = matrix(NA, save, K)

# Do the loop
for(i in 1:(save + burn)) {
  # Draw alpha
  V_post = solve(crossprod(X) / sigma_draw + diag(1 / diag(V_prior)))
  alpha_post = V_post %*% (crossprod(X, y) / sigma_draw)
  alpha_draw = alpha_post + t(chol(V_post)) %*% rnorm(K)

  # Determine inclusion based on alpha
  for(j in 1:K) {
    p0 = dnorm(alpha_draw[[j]], 0, sqrt(tau0[j]))
    p1 = dnorm(alpha_draw[[j]], 0, sqrt(tau1[j]))
    p11 = p1 / (p0 + p1)

    gamma[[j]] = ifelse(p11 > runif(1), 1, 0)
  }

  # Construct prior VC matrix
  V_prior = diag(as.numeric(gamma * tau1 + (1 - gamma) * tau0))

  # Draw sigma^2
  S_post = S_prior + crossprod(y - X %*% alpha_draw) / 2
  s_post = S_prior + N / 2
  sigma_draw = 1 / rgamma(1, s_post, S_post)
  V_beta_draw = diag(sigma_draw * solve(crossprod(X)))

  # Ignore the first n(=burn) iterations, store results afterwards
  if(i > burn) {
    alpha_store[i - burn, ] = alpha_draw
    sigma_store[i - burn, ] = sigma_draw
    V_beta_store[i - burn, ] = V_beta_draw
    gamma_store[i - burn, ] = gamma
  }
}

# Get means for the output
pip_mean = apply(gamma_store, 2, mean)
```

```r
  alpha_mean = apply(alpha_store, 2, mean)
  sigma_mean = apply(sigma_store, 2, mean)
  V_beta_mean = apply(V_beta_store, 2, mean)

  names(pip_mean) = names(alpha_mean) = names(V_beta_mean) = colnames(X)

  # Store the parameters used for the output
  meta_data = list("tau0" = tau0, "tau1" = tau1,
                   c("save" = save, "burn" = burn,
                   "s_prior" = s_prior, "S_prior" = S_prior,
                   "standardise" = standardise))

  out = list(pip_mean, alpha_mean, sigma_mean, V_beta_mean, meta_data)

  names(out) = c("pip", "post_mean", "post_std", "post_var_beta", "meta")

  return(out)
}
```