Estimators of Binary Spatial Autoregressive Models: A Monte Carlo Study
online appendix

Raffaella Calabrese
Essex Business School
University of Essex
rcalab@essex.ac.uk

Johan A. Elkink
School of Politics & International Relations
and Geary Institute
University College Dublin
jos.elkink@ucd.ie

January 17, 2014

Online appendix to R. Calabrese and J.A. Elkink, "Estimators of Binary Spatial Autoregressive Models: A Monte Carlo Study", *Journal of Regional Science*, forthcoming.

Contents

1 Estimators

1.1 Wrapper function

Part of this function is based on code in the built-in `lm()` function in R.

```
bsar <- function(formula, data, adj.matrix, method = "RIS", weights, subset,
    na.action, start = NULL, etastart, mustart, offset, control = list(...),
    model = TRUE, x = FALSE, y = TRUE, contrasts = NULL, debug = FALSE, ...)
{
    ## Name all parameters
    call <- match.call()

    ## If 'data' parameter not given, take calling environment
    if (missing(data))
        data <- environment(formula)

    mf <- match.call(expand.dots = FALSE)
    m <- match(c("formula", "data", "subset", "weights", "na.action",
        "etastart", "mustart", "offset"), names(mf), 0L)
    mf <- mf[c(1L, m)]
    mf$drop.unused.levels <- TRUE
    mf[[1L]] <- as.name("model.frame")
    mf <- eval(mf, parent.frame())
    mt <- attr(mf, "terms")

    Y <- model.response(mf, "any")
    if (length(dim(Y)) == 1L) {
        nm <- rownames(Y)
        dim(Y) <- NULL
        if (!is.null(nm))
            names(Y) <- nm
    }

    X <- if (!is.empty.model(mt))
        model.matrix(mt, mf, contrasts)
    else matrix(, NROW(Y), 0L)

    W <- adj.matrix

    weights <- as.vector(model.weights(mf))
    if (!is.null(weights) && !is.numeric(weights))
        stop("'weights' must be a numeric vector")
    if (!is.null(weights) && any(weights < 0))
        stop("negative weights not allowed")
    offset <- as.vector(model.offset(mf))
    if (!is.null(offset)) {
        if (length(offset) != NROW(Y))
            stop(gettextf(
                "number of offsets is %d should equal %d (number of obs.)",
                length(offset), NROW(Y)), domain = NA)
    }
    mustart <- model.extract(mf, "mustart")
```

```
    etastart <- model.extract(mf, "etastart")

    if (method == "EM")
        fit <- em_estimator(Y, X, W, control, debug = debug)
    else if (method == "PS")
        fit <- ps_estimator(Y, X, W, control, debug = debug)
    else if (method == "KM")
        fit <- km_estimator(Y, X, W, control, debug = debug)
    else if (method == "RIS")
        fit <- ris_estimator(Y, X, W, control, debug = debug)
    else if (method == "Gibbs")
        fit <- gibbs_estimator(Y, X, W, control, debug = debug)
    else
        stop(sprintf("Method '%s' not yet implemented", method))

    fit
}
```

1.2 Expectation-Maximization

```
require(MASS)

em_estimator <- function(y, X, W, control = list(...), debug = 0)
{
    con <- list(
        optim.max.iterations = 1000,
        em.max.iterations = 1000,
        em.converge.threshold = 1e-4,
        rho.init = 0.2
    )
    con[names(control)] <- control

    ## Initial values are computed by ordinary probit model and
    ## 1 for standard deviation and 0 for rho
    init <- c(coef(glm(y ~ 0 + X, family = binomial(link = "probit"))),
                    con$rho.init)

    ## Outside of ll.ar function for efficiency reasons
    n <- length(y)
    k <- dim(X)[2]
    I <- diag(n)

    ## Loglikelihood function of the AR model
    ll.ar <- function(theta, y, X, W) {

        Xb <- X %*% theta[1:k]
        p <- -1 + 2 * pnorm(theta[k+1])
```

```
    A <- -p*W
    diag(A) <- 1
    e <- A %*% y - Xb

    ll <- - (1/2) * t(e) %*% (e)  + log(det(A))

    ll
}

# Necessary for first iteration
Ey <- X %*% init[1:k]
theta <- init

s1 <- y == 1

iter <- 1
repeat {

    theta.prev <- theta

    ## Run optim using expected Y values (M-step)
    out <- optim(theta, ll.ar,
                 control = list(fnscale = -1,
                        maxit = con$optim.max.iterations),
                 y = Ey, X = X, W = W)
    theta <- out$par

    ## Calculate expected Y values (E-step)
    p <- -1 + 2 * pnorm(theta[k+1])
    Xb <- X %*% theta[1:k]
    A <- -p * W
    diag(A) <- 1
    Ai <- solve(A)
    sdV <- sqrt(rowSums(Ai ^ 2) )

    AiXb <- Ai %*% Xb

    pnEy <- pnorm(AiXb / sdV)
    Ey <- ifelse(s1,
            AiXb + sdV * dnorm(AiXb / sdV) / ifelse(pnEy > 0,
                    pnEy, 1e-7),
            AiXb - sdV * dnorm(AiXb / sdV) / ifelse(pnEy < 1,
                    1 - pnEy, 1e-7))

    ## Break loop if convergence condition satisfied
    iter <<- iter + 1
    if (sum(abs(theta - theta.prev)) < con$em.converge.threshold ||
```

```
                iter >= con$em.max.iterations) break;
    }

    se = rep(NA, k+1)

    list(beta = theta[1:k],
         rho = -1 + 2 * pnorm(theta[k+1]),
         beta.se = rep(NA, k),
         rho.se = NA,
         loglik = ll.ar(theta, y, X, W))
}
```

1.3 Gibbs

```
library(MASS)
library(truncnorm)

gibbs_estimator <- function(y, X, W, control = list(...), debug = FALSE)
{
    con <- list(
        q = 100,
        c = .1,
        R = 3000,
        run.in = 500,
        rho.init = 0.2
    )
    con[names(control)] <- control

    with(con, {

        n <- length(y)
        k <- dim(X)[2]
        In <- diag(rep(1, n))
        Ik <- diag(rep(1, k))
        ik <- rep(1, k)

        ## Initial values
        rho <- rho.init
        P <- v <- rep(1, n)
        A <- In - rho * W
        sigma2 <- 1
        z <- y

        densities <- matrix(NA, nrow = R, ncol = k + 1)

        rchisq.sigma2 <- rchisq(R, n)
        rchisq.v <- matrix(rchisq(R * n, q + 1), R, n)
```

```
runif.rho <- runif(R)

s0 <- y == 0
s1 <- y == 1
n0 <- sum(s0)
n1 <- n - n0

f <- function(rho, y, Xb, sigma2, W, P) {
    A <- -rho * W
    diag(A) <- 1
    log(det(A)) - sum(((A %*% y - Xb) * P)^2) / (2 * sigma2)
}

rho.count.acceptance <- 0

for (i in 1:R)
{
    ## update beta
    ytil <- A %*% z
    xstar <- P %*% t(ik) * X
    ystar <- P * ytil
    Hinv <- solve(t(xstar) %*% xstar / sigma2)
    b0 <- Hinv %*% (t(xstar) %*% ystar / sigma2)
    bhat <- mvrnorm(1, b0, Hinv)
    Xb <- X %*% bhat

    ## update sigma2
    sigma2 <- 1

    ## update v
    v <- ((ytil - Xb)^2 / sigma2 + q) / rchisq.v[i,]
    P <- 1/sqrt(v)

    ## update z
    mu <- solve(A) %*% Xb
    h <- diag(as.vector(P)) %*% A / sqrt(sigma2)
    covinv <- t(h) %*% h
    var.z <- 1 / diag(covinv)
    resid <- z - mu
    mu.z <- mu - var.z * (covinv %*% resid) + resid
    z[s1] <- rtruncnorm(n1, mean = mu.z[s1],
                sd = sqrt(var.z[s1]), a = 0)
    z[s0] <- rtruncnorm(n0, mean = mu.z[s0],
                sd = sqrt(var.z[s0]), b = 0)

    ## update rho using metropolis-hastings
    repeat {
```

```
            rhonew <- rho + c * rnorm(1)
            if (rhonew > -.9999 & rhonew < 1) break
        }

        f.old <- f(rho, z, Xb, sigma2, W, P)
        f.new <- f(rhonew, z, Xb, sigma2, W, P)
        ratio <- exp(f.new - f.old)

        if (runif.rho[i] < ratio) {

            rho <- rhonew
            A <- -rho * W
            diag(A) <- 1
            rho.count.acceptance <- rho.count.acceptance + 1
        }

        ## Tune c based on acceptance rate
        rho.tune.acceptance <- rho.count.acceptance / i

        if (rho.tune.acceptance > .6)
           c <- min(c * 1.1, .7)
        else
           c <- c / 1.1

        densities[i,] <- c(bhat, rho)
    }

    theta.hat <- apply(densities[(run.in+1):R,], 2, mean)
    theta.se <- apply(densities[(run.in+1):R,], 2, sd)

    list(beta = theta.hat[1:k],
         rho = theta.hat[k+1],
         beta.se = theta.se[1:k],
         rho.se = theta.se[k+1],
         densities = densities)
  })
}
```

1.4 Recursive Importance Sampler

```
ris_estimator <- function(y, X, W, control = list(...), debug = FALSE)
{
  con <- list(
    R = 1000,
    store.density = FALSE,
    optim.max.iterations = 1000,
    optim.reltol = .0025
```

```
)
con[names(control)] <- control

with(con, {

  n <- length(y)
  k <- dim(X)[2]
  Z <- diag(as.vector(1 - 2 * y))
  In <- diag(n)

  ## Random draw from importance density function, given
  ## an upper bound, using antithetical sampling
  random.draw <- function(upper.bound)
  {
    q <- runif(R/2)
    q <- c(q, 1-q)

    qnorm(q * pnorm(upper.bound))
  }

  iter <- 0

  densities <- NULL

  ll <- function(par)
  {
    iter <<- iter + 1

    rho <-  -1 + 2 * pnorm(par[k+1])

    beta <- par[1:k]

    Ai <- -rho * W
    diag(Ai) <- 1
    A<-solve(Ai)

    omega <- Z %*% A %*% t(A) %*% t(Z)
    V <- -Z %*% A %*% X %*% beta

    B <- solve(chol(solve(omega)))

    Eta0 <- Eta <- matrix(NA, nrow = n, ncol = R)

    Eta0[n,] <- rep(1/B[n,n] * V[n], R)
    Eta[n,] <- random.draw(Eta0[n,])

    for (i in (n-1):1)
```

```
      {
        Eta0[i,] <- 1/B[i,i] * (V[i] - t(B[i,(i+1):n]) %*% Eta[(i+1):n,])
        if (i > 1)
            Eta[i,] <- random.draw(Eta0[i,])
      }

      pnEta0 <- pnorm(Eta0)
      a <- 1 / mean(pnEta0)
      ll <- log(mean(apply(pnEta0 * a, 2, prod))) - log(1/R) - n * log(a)

      ll
    }

    par <- optim(c(rep(0,k), 0), ll, control=list(fnscale = -1,
                reltol = optim.reltol, maxit = optim.max.iterations))

    list(beta = par$par[1:k],
         rho = -1 + 2 * pnorm(par$par[k+1]),
         beta.se = rep(NA, k),
         rho.se = NA,
         densities = densities)
  })
}
```

1.5 Generalized Method of Moments

```
ps_estimator <- function(y, X, W, control = list(...), debug = FALSE)
{
    n <- length(y)
    k <- dim(X)[2]
    Z <- cbind(1, X, W %*% X, W %*% W %*% X, W %*% W %*% W %*% X)
    ll <- function(theta)
    {
        beta <- theta[1:k]
        rho <- theta[k+1]
        B <- -rho * W
        diag(B) <- 1
        A <- solve(B)
        v <- sqrt(diag(A %*% t(A)))
        g <- A %*% X %*% beta
        G <- g / v
        u <- dnorm(G) * (y - pnorm(G)) / max((pnorm(G) *
                                    (1 - pnorm(G))), 1e-16)
        S <- 1/n * t(Z) %*% u
        (Q <- t(S) %*% S)
    }
```

```
    init.values <- coef(glm(y ~ 0 + X, family = binomial(link = "probit")))
    par <- constrOptim(c(init.values, 0), ll, NULL,
             ui = rbind(c(0,0,-1), c(0,0,-1), c(0,0,1)), ci = c(-1,-1,-1))

    list(beta = par$par[1:k],
         rho = par$par[k+1],
         beta.se = rep(NA, k),
         rho.se = NA)
}
```

1.6 Linearized Generalized Method of Moments

```
library(Matrix)
library(MASS)

invlogit <- function(x) { 1 / (1 + exp(-x)) }

km_estimator <- function(y, X, W, control = list(...), debug = 0)
{
    k <- dim(X)[2]
    n <- length(y)

    ## Step 1
    theta <- c(coef(glm(y ~ 0 + X, family = binomial(link = "logit"))), 0)
    b <- theta[1:k]
    rho <- theta[k+1]
    P <- invlogit(X %*% b)
    Gb <- apply(X, 2, function(x) { P * (1 - P) * x })
    Grho <- P * (1 - P) * (W %*% X %*% b)
    G <- cbind(Gb, Grho)
    u = y - invlogit(X %*% b)

    ## Step 2
    Z <- cbind(1, X, W %*% X, W %*% W %*% X, W %*% W %*% W %*% X)
    Ghat <- Z %*% ginv(t(Z) %*% Z) %*% t(Z) %*% G
    Gbhat <- Ghat[,1:k]
    Grhohat <- Ghat[,k+1]
    bhat <- ginv(t(Gbhat) %*% Gbhat) %*% t(Gbhat) %*% (u + Gb %*% b)
    rhohat <- ginv(t(Grhohat) %*% Grhohat) %*% t(Grhohat) %*% (u + Gb %*% b)

    theta <- c(bhat, rhohat)

    list(beta = theta[1:k],
         rho = theta[k+1],
         beta.se = rep(NA, k),
         rho.se = NA,
         loglik = NA)
```

```
}
```

2 Monte Carlo setup

2.1 Parameter settings

```
## Path variables
work <- "/work/"
data.dir <- paste(work, "data/", sep = "")
libs.dir <- paste(work, "Rlibs/", sep = "")
output.dir <- paste(work, "output/", sep = "")
tmp.dir <- paste(work, "tmp/", sep = "")

## Monte Carlo options
mu.x <- 2
sd.x <- 4
b0   <- 4
b1   <- -2
sd.e <- 1
rho  <- c(0, .1, .45, .8)
N    <- c(50, 500, 1500)

## Prepare parameters matrix
params <- expand.grid(rho, N)
colnames(params) <- c("rho", "N")
params <- as.data.frame(params)

params$mu.x <- mu.x
params$sd.x <- sd.x
params$b0   <- b0
params$b1   <- b1
params$sd.e <- sd.e

params$id <- 1:dim(params)[1]

write.csv(params, file = paste(output.dir, "parameters.csv", sep = ""),
                              row.names = FALSE)
```

2.2 Data generation

```
## Environment variables
methods <- c(Sys.getenv("MC_METHOD"))
sample.size <- as.integer(Sys.getenv("MC_N"))
subset.start <- as.integer(Sys.getenv("MC_START"))
subset.end <- as.integer(Sys.getenv("MC_END"))
```

```
process.id <- as.integer(Sys.getenv("MC_ID"))

## Path variables
work <- "/work/"
data.dir <- paste(work, "data/", sep = "")
output.dir <- paste(work, "output/", sep = "")
tmp.dir <- paste(work, "tmp/", sep = "")

## Monte Carlo options
params <- read.csv(paste(output.dir, "parameters.csv", sep = ""))
lbl <- sprintf("%s_%s_%d", Sys.info()["nodename"], format(Sys.time(),
                    "%Y%m%d_%H%M%S"), process.id)

## Libraries required
library(rlecuyer)

## Function to generate random W
## Follows description of algorithm by Beron & Vijverberg (2004)
generate.W <- function(ncases)
{
    x <- runif(ncases)
    y <- runif(ncases)

    if (ncases == 50)  d <- .21
    else if (ncases == 500) d <- .06
    else if (ncases == 1500) d <- .036
    else stop("Do not know d to generate W!")

    D <- as.matrix(dist(cbind(x, y)))
    W <- D < d
    diag(W) <- 0

    rs <- rowSums(W)
    W[rs > 0, rs > 0] <- W[rs > 0, rs > 0] / rs[rs > 0]

    W
}

## Main monte carlo loop
streamName <- paste("str", process.id, sep = "")
.lec.CreateStream(streamName)
.lec.SetSeed(streamName, sample(1:60000, 6, replace = TRUE) * process.id)
.lec.CurrentStream(streamName)

p <- params[params$N == sample.size, ]

for (s in subset.start:subset.end) {
```

```
    for (i in 1:dim(p)[1]) {

        ## Generate exogenous variables
        x  <- rnorm(p[i,"N"], p[i,"mu.x"], p[i,"sd.x"])
        Xb <- cbind(1, x) %*% c(p[i,"b0"], p[i,"b1"])
        e  <- rnorm(p[i,"N"], 0, p[i,"sd.e"])
        W  <- generate.W(p[i,"N"])

        ## Generate Y
        A       <- -p[i,"rho"] * W
        diag(A) <- 1
        A       <- solve(A)

        y.star <- A %*% Xb + A %*% e
        y      <- ifelse(y.star > 0, 1, 0)

        ## Save the data
        save(y, y.star, x, e, W, p, s, i, file =
              sprintf("%sdata_%05d_%04d.Rdata", data.dir, s, p[i, "id"]))
    }
}

.lec.CurrentStreamEnd()
```

Note that a version with `rlogis()` instead of `rnorm()` was used for the generation of `e` for the alternative simulations for the linearized GMM estimator.

2.3 Simulation run

```
## Environment variables
methods <- c(Sys.getenv("MC_METHOD"))
sample.size <- as.integer(Sys.getenv("MC_N"))
subset.start <- as.integer(Sys.getenv("MC_START"))
subset.end <- as.integer(Sys.getenv("MC_END"))
process.id <- as.integer(Sys.getenv("MC_ID"))

## Path variables
work <- "/work/"
data.dir <- paste(work, "data/", sep = "")
output.dir <- paste(work, "output/", sep = "")
tmp.dir <- paste(work, "tmp/", sep = "")


## Libraries required
library(rlecuyer)
```

```
## Parameter values
params <- read.csv(sprintf("%sparameters.csv", output.dir))

report <- function(set, id, N, rho, est, e)
{
    cat(sprintf("FAIL: %s id=%d config=%d N=%d rho=%.2f: %s",
            est, s, id, N, rho, e),
        file = sprintf("%serrors_%s.txt", output.dir, lbl), append = TRUE)
}

## Main monte carlo loop
lbl <- sprintf("%s_%s_%d", Sys.info()["nodename"], format(Sys.time(),
                    "%Y%m%d_%H%M%S"), process.id)

empty.estimate <- list(beta = c(NA, NA), rho = NA,
         beta.se = c(NA, NA), rho.se = NA)

streamName <- paste("str", process.id, sep = "")
.lec.CreateStream(streamName)
.lec.SetSeed(streamName, sample(1:600, 6, replace = TRUE) * process.id)
.lec.CurrentStream(streamName)

p <- params[params$N == sample.size,]

processing.order <- sample(1:dim(p)[1], dim(p)[1], replace = FALSE)

for (s in subset.start:subset.end) {
    for (i in processing.order) {

        load(sprintf("%sdata_%05d_%04d.Rdata", data.dir, s, p[i,"id"]))

        for (m in methods) {

            est.t <- system.time(tryCatch({
                if (m != "probit") {
                    est <<- bsar(y ~ x, adj.matrix = W, method = m)
                } else {
                    pbmodel <- glm(y ~ x, family = binomial(link = "probit"))
                    est <<- list(beta = coef(pbmodel),
                                 rho = NA,
                                 beta.se = sqrt(diag(vcov(pbmodel))),
                                 rho.se = NA,
                                 loglik = NA)
                }
            }, error = function(e) {
                est <<- empty.estimate
                report(s, p[i,"id"], p[i,"N"], p[i,"rho"], m, e)
```

```
            }))[3]

            cat(paste(c(s, p[i,"id"], p[i,"N"], p[i,"rho"], m, est$beta,
                    est$rho, est.t, lbl), collapse = ","),"\n", sep = "",
                file = sprintf("%sestimates_%s.csv", output.dir, lbl),
                append = TRUE)
        }
    }
}

.lec.CurrentStreamEnd()
```