VAR Example Batch Size

Deepak Bastola

October 08 2022

```
# install and load the necessary libraries

set.seed(16235, kind = "L'Ecuyer-CMRG" )
pkgs <- c("mvtnorm", "truncnorm", "invgamma","TruncatedNormal", "parallel",
          "Matrix", "ts.extend", "matrixcalc", "mAr", "parallel", "ggthemes")

 if(sum(as.numeric(!pkgs %in% installed.packages())) != 0) {
    installer <- pkgs[!pkgs %in% installed.packages()]
    for(i in 1:length(installer)) {
      install.packages(installer, dependencies = T)
      break()}
    sapply(pkgs, require, character = T)
  } else {
    sapply(pkgs, require, character = T)
  }
```

```
      mvtnorm       truncnorm        invgamma TruncatedNormal        parallel
         TRUE            TRUE            TRUE            TRUE            TRUE
       Matrix       ts.extend      matrixcalc             mAr        parallel
         TRUE            TRUE            TRUE            TRUE            TRUE
      ggthemes
         TRUE
```

```
# Function to simulate VAR model parameters
# for different dependence/correlation

sigphi <- function(p, rho){
  set.seed(1692, kind = "L'Ecuyer-CMRG" )
  omega <- Matrix(toeplitz(rev(seq(0.2,1, length.out = p)), sparse = TRUE))
  A <- matrix(rnorm(p*p,mean=0,sd=1), p, p)
  B <- A%*%t(A)
  m <- max(eigen(B)$values)
  phi0 <- B/(m+0.001)
  phik <- bdiag(rho*phi0)
  scratch <- diag((p)^2) - kronecker(phik,phik)
  V.s <- solve(scratch)%*%c(diag(p))
  V <- matrix(V.s, nrow = p, byrow = TRUE)
  Sigma <- solve(diag(p)-phik)%*%V + V%*%solve(diag(p)-phik) -V
  return(list(phi =phik, Sigma= Sigma))
}
```

```r
# Function to calculate the exact bias and variance

funBMexact <- function(b, x, y, r, c){
  n <- length(x)
  b <- floor(b)
  br <- floor(b/r)
  gamma0b1 <- 2*sum(x[2:b])
  gamma0b1r <- 2*sum(x[2:br])
  gamma0n1 <- 2*sum(x[2:n])
  gamma1b1 <- 2*sum((1:(b-1))*x[2:b])
  gamma1b1r <- 2*sum((1:(br-1))*x[2:br])
  gamma1n1 <- 2*sum(x[2:n]*seq(1, n-1))
  exact1 <-  (1/(1-c))* (x[1] + (((n/b)*gamma0b1 - gamma0n1)/(n/b -1)) +
                           (gamma1n1 - (n/b)^2*gamma1b1)/(n*(n-b)/b)) -
    (c/(1-c))* (x[1] + (((n/br)*gamma0b1r - gamma0n1)/(n/br -1)) +
                  (gamma1n1 - (n/br)^2*gamma1b1r)/(n*(n-br)/br))
  bias.exact <- y - exact1
  variance.exact <- (8*(y^2)*b/n) +  (6*(y^2)*b/(r*n)) + (8*y^2*(r-1)/(r*(r - b/n)))*(b^2/n^2)
  mse.exact <- bias.exact^2 + variance.exact
  return(mse.exact)
}
```

```r
# Function to calculate the proposed bias and variance for BM estimators

funBMi <- function(b, x, y, r, c){
  n <- length(x)
  b <- floor(b)
  br <- floor(b/r)
  gamma0b1 <- 2*sum(x[2:b])
  gamma0b1r <- 2*sum(x[2:br])
  gamma0n1 <- 2*sum(x[2:n])
  gamma1b1 <- 2*sum((1:(b-1))*x[2:b])
  gamma1b1r <- 2*sum((1:(br-1))*x[2:br])
  bias <- (1/(1-c))* ((n/(n-b))*(gamma0n1 - gamma0b1 + (gamma1b1)/b)) -
    (c/(1-c)* ((n/(n-br))*(gamma0n1 - gamma0b1r + (gamma1b1r)/br)))
  var <- (2*(y^2)*b/n) * (1/r + (r-1)/(r*(1-c)^2)) +
    (2*(y^2)*(b/n)^2) * ((2*c)/((1-c)^2*r^2) - 2/r)
  mse.bm <- bias^2 + var
  return(mse.bm)
}
```

```r
# Function to calculate the proposed bias and variance for OBM estimators

funOBMi <- function(b, x, y, r, c){
  b <- floor(b)
  n <- length(x)
  br <- floor(b/r)
  gamma0b1 <- 2*sum(x[2:b])
  gamma0nb <- 2*sum(x[2:(n-b+1)])
  gamma0nbr <- 2*sum(x[2:(n-br+1)])
  gamma0b1r <- 2*sum(x[2:br])
  gamma0n1 <- 2*sum(x[2:n])
  gamma1b1 <- 2*sum((1:(b-1))*x[2:b])
```

```
  gamma1b1r <- 2*sum((1:(br-1))*x[2:br])
  bias <- (1/(1-c))*(gamma0n1 - gamma0b1 + (n*gamma1b1)/(b*(n-b)) -
                       (b*n)/((n-b)*(n-b+1))*(gamma0b1 - gamma0nb)) -
    (c/(1-c)*(gamma0n1 - gamma0b1r + (n*gamma1b1r)/(br*(n-br)) -
                (br*n)/((n-br)*(n-br+1))*(gamma0b1r - gamma0nbr)))
  var <- ((4/3)*(y^2)*b/n) * (1/r + (r-1)/(r*(1-c)^2)) +
    ((4/3)*(y^2)*(b/n)^2) * ((2*c)/((1-c)^2*r^2) - 2/r)
  mse.obm <- bias^2 + var
  return(mse.obm)
}
```

```
# current
funCurrbm <- function(b, x, y, r, c){
  mse.bm.curr <- ((x/b)*(1-r*c)/(1-c))^2 + (8*(y^2)*b/n) +
    (6*(y^2)*b/(r*n)) + (8*y^2*(r-1)/(r*(r - b/n)))*(b^2/n^2)
  mse.bm.curr <- na.exclude(mse.bm.curr)
  return(mse.bm.curr)
}

funCurrobm <- function(b, x, y, r, c){
  mse.obm.curr <- ((x/b)*(1-r*c)/(1-c))^2 + ((4/3)*(y^2)*b/n) *
    (1/r + (r-1)/(r*(1-c)^2)) + ((4/3)*(y^2)*(b/n)^2) * ((2*c)/((1-c)^2*r^2) - 2/r)
  mse.obm.curr <- na.exclude(mse.obm.curr)
  return(mse.obm.curr)
}
```

```
# Function to calculate the optimal batch sizes

Batch_sizes <- function(n, p, Sigma, phi, rho, nrep){
  c=1/2
  omega = diag(p)
  chain <- lapply(1:nrep, function(j) as.matrix(mAr.sim(rep(0,p),
                                                        as.matrix(phi),
                                                        omega, N = n)))

  ar.chain <- lapply(1:nrep,
                     function(j) lapply(1:p,
                                        function(i) ar(chain[[j]][,i], order.max = NULL, method = "yw")))
  m <- lapply(1:nrep,
              function(j) sapply(1:p,
                                 function(i) ar.chain[[j]][[i]]$order))
  phi.i <- lapply(1:nrep,
                  function(j) sapply(1:p,
                                     function(i) ar.chain[[j]][[i]]$ar))
  sigma.e <- lapply(1:nrep,
                    function(j) sapply(1:p,
                                       function(i) ar.chain[[j]][[i]]$var.pred))
  Sigma.pilot <-lapply(1:nrep,
                       function(j) sapply(1:p,
                                          function(i) sigma.e[[j]][[i]]/(1 - sum(phi.i[[j]][[i]]))^2))

  # Find autocovariances using function ARMA.autocov() from ts.extend
  ar.autocovar <- lapply(1:nrep,
```

```
                        function(j) lapply(1:p, function(i)
                          ARMA.autocov(n = n, ar = phi.i[[j]][[i]], ma = 0, corr = FALSE)))

# Current batch size selection method

ll <- lapply(1:nrep,
             function(s) lapply(1:p,
                                function(j) lapply(1:m[[s]][[j]],
                                                   function(i) sapply(1:i,
                                                                      function(k)
                                                                        (k * ar.autocovar[[s]][[j]][a
ll.sum <- lapply(1:nrep,
                 function(k) sapply(1:p,
                                    function(j) sapply(1:m[[k]][[j]],
                                                       function(i) sum(ll[[k]][[j]][[i]])))))
t1 <- lapply(1:nrep,
             function(k) sapply(1:p,
                                function(j) sum(phi.i[[k]][[j]]*ll.sum[[k]][[j]])))
t2 <- lapply(1:nrep,
             function(k) sapply(1:p,
                                function(j) ((sigma.e[[k]][[j]] - ar.autocovar[[k]][[j]][1])/2)*
                                  sum(sapply(1:m[[k]][[j]], function(i) i*phi.i[[k]][[j]])))))
mult <- lapply(1:nrep,
               function(k) sapply(1:p, function(i) 1/(1 - sum(phi.i[[k]][[i]])))))
gamma.pilot <- lapply(1:nrep, function(k) -2*(t1[[k]] + t2[[k]])*mult[[k]])


# batch sizes for exact methods

b <- seq(10,1.5*n^(1/2))
a <- n/b


# use optim in a function

# batch size for proposed methods
b.bm.exact <- lapply(1:3,
                     function(k)
                       lapply(1:nrep,
                              function(j)
                                sapply(1:p,
                                       function(i) optim(par = c(40),
                                                         fn=funBMexact,
                                                         x = ar.autocovar[[j]][[i]],
                                                         y = diag(Sigma)[i],
                                                         r= k, c =1/2,
                                                         method = "Brent",
                                                         lower = c(6), upper=c(200)))))

# batch size for proposed methods
b.bm <- lapply(1:3,
               function(k)
                 lapply(1:nrep,
```

```
                        function(j)
                          sapply(1:p,
                                 function(i) optim(par = c(40),
                                                   fn=funBMi,
                                                   x = ar.autocovar[[j]][[i]],
                                                   y = Sigma.pilot[[j]][i],
                                                   r= k, c =1/2,
                                                   method = "Brent",
                                                   lower = c(6), upper=c(200)))))

# Geometric Mean

b.exact <- lapply(1:3,
                  function(j) lapply(1:nrep,
                                     function(i) exp(mean(log(unlist(b.bm.exact[[j]][[i]][1,])))))
b.bm.opt <- lapply(1:3,
                   function(j) lapply(1:nrep,
                                      function(i) exp(mean(log(unlist(b.bm[[j]][[i]][1,])))))


# batch size for current methods

ar.autocorr <- lapply(1:nrep,
                      function(k) lapply(1:p,
                                         function(j)
                                           ARMA.autocov(n = n, ar = phi.i[[k]][[j]], ma = 0, corr = TR
ubound <- 2*sqrt(log(n)/n)

rho.k <- lapply(1:nrep,
                function(k) sapply(1:n,
                                   function(i)
                                     max(abs(sapply(1:p, function(j) ar.autocorr[[k]][[j]][[i]])))))

# lag-based method
a.n <- 1:5

b.max <- lapply(1:nrep,
                function(k) sapply(1:p,
                                   function(i) which.max(ar.autocorr[[k]][[i]]<ubound)-1))

b.politis <- lapply(1:nrep,
                    function(k) sapply(1:p,
                                       function(i) if (all(sapply(a.n,
                                                                  function(j)
                                                                    ar.autocorr[[k]][[i]][which.max(ar
                                         {b.max[[k]][i] + 5}))

b.curr.bm <- lapply(1:3,
                    function(k)
                      lapply(1:nrep,
                             function(j)
                               sapply(1:p,
                                      function(i)
```

```
                                        optim(par = c(40),
                                              fn=funCurrbm,
                                              x = gamma.pilot[[j]][[i]],
                                              y = Sigma.pilot[[j]][i],
                                              r = k, c=c, method = "Brent",
                                              lower = c(6), upper = c(200)))))
  # geometric mean

  b.currbm.opt <- lapply(1:3,
                         function(j) lapply(1:nrep,
                                            function(i)
                                              exp(mean(log(unlist(b.curr.bm[[j]][[i]][1,])))))
  b.currbm.opt[[2]] <- lapply(1:nrep, function(i) exp(mean(log(unlist(b.politis[[i]])))))



  # Collect the batch sizes
  b.sizes <- lapply(1:3,
                    function(j)
                      lapply(1:nrep,
                             function(i)
                               c(b.exact[[j]][[i]], b.bm.opt[[j]][[i]], b.currbm.opt[[j]][[i]])))
  b.avg <- lapply(1:3, function(i) Reduce("+", b.sizes[[i]])/nrep)


  return(list("Batch Sizes" = b.sizes,
              "Batch Average" = b.avg,
              "Batch Lag-based" = b.politis,
              "Sigma" = Sigma,
              "No.of.Repetions" = nrep))
}

# Simulation setup

p <- 4
rho <- seq(0.80, 0.90, by = 0.01)
n <- 2e4
nrep = 20

foo <- lapply(1:length(rho), function(i) sigphi(p, rho[i]))
phi <- lapply(1:length(rho), function(i) foo[[i]][[1]])
Sigma <- lapply(1:length(rho), function(i) foo[[i]][[2]])

# Trial Simulation

phi = phi[[10]]
Sigma = Sigma[[10]]
r1 <- Batch_sizes(n, p=p, Sigma=Sigma, phi=phi, rho=rho[10], nrep = nrep)

# starttime <- Sys.time()
# sim2e4n <- lapply(1:length(rho),
#     function(i) Batch_sizes(n, p=p, Sigma=Sigma[[i]], phi=phi[[i]], rho=rho[i], nrep = nrep))
# endtime <- Sys.time()
```

```
# endtime - starttime
# save(sim2e4n, file = "2e4batch.Rda")
```