Implementation Multiple Time Series Analysis

Jacopo Lussetti

2025-04-02

Companion Matrix function

```
comp_mtrx <- function(AA){
    ## AA is a K x Kp matrix, so we are able to derive p in the following way
    K <- nrow(AA)
    Kp <- ncol(AA)
    p <- Kp/K

    # Create the empty companion matrix Kp x Kp
    C <- matrix(0, nrow=Kp, ncol=Kp)

    C[1:K,] <- AA

    # Add ones on the K-th sub-diagonal
    if (p>1)
        C[(K+1):Kp, 1:(Kp-K)] <- diag(Kp - K)
    return(C)
}
```

Autocovariance function

Equation (2.1.39) page 29 represents the formula to compute autocovariances of a *stable* VAR(p) Process. First and foremost we will then have to evaluate whether the process is stable

Stability check

A VAR(p) is a **stable process** if the condition (2.1.9.) holds:

$$det(I - \mathbf{A}z) \neq 0 \; if \; |z| < 1 \text{ where z are eigenvalues for A}$$

```
 var_roots<-function(AA){
   if(nrow(AA)==ncol(AA)){  # matrix is squared
     C<-AA
   }else{
     C<-comp_mtrx(AA) # transform form compact matrix into companion matrix
   }
   eig<-eigen(C)$values
   return(eig)
 }
```

Formula

After evaluating the conditions for stability, we proceed then defining the formula to compute contrivances. Th

$$\text{vec } \Gamma_Y(0) = (I_{(Kp)^2} - \mathbf{A} \otimes \mathbf{A})^{-1} \text{vec } \Sigma_U \tag{1}$$

```
autocov_fun<-function(A, Sigma_u,p=1){ # A for high-order var is combined matrix,
  K<-nrow(Sigma_u)
  Kp<-K * p
  #for var(1) is just A1
 if(p>1){
   #compute companion
   A<- comp_mtrx(A)
   #extend original sigma_u
  Sigma_U<-matrix(0, nrow=Kp, ncol=Kp)
  Sigma_U[1:K, 1:K]<-Sigma_u

 }else{
   Sigma_U<-Sigma_u
 }
 # compute the Kronecker product
  I<-diag(1, Kp^2)
  #compute vectorised Sigma_U
  vec_Sigma_U<-as.vector(Sigma_U)
  # compute the Autocovariance function
  vec_gamma_0<-solve(I - kronecker(A, A)) %*% vec_Sigma_U
  # reshape the result into a matrix
  Gamma_Y_0<-matrix(vec_gamma_0, nrow=Kp, ncol=Kp)

  return(Gamma_Y_0)
}
```

Equilibrium Points

Equilibrium points are defined by formula 2.1.10 at page 16

$$\mu := E(Y_t) = (I_{Kp} - \mathbf{A})^{-1}\nu \tag{2}$$

```
equilibrium<-function(A, nu){
  #check stability condition
  eig<-var_roots(A)
  if(any(Mod(eig)>=1)){
      stop("Trajectories are not stable")
  }
  Kp<-nrow(A)
  I_Kp<-diag(1,Kp)
  values<-solve(I_Kp-A) %*% nu
  return(values)
}
```

VAR(p) model

For this case we just consider the function simulating the trajectories, whereas equilibrium and autocovariance functions are treated separately.

```
  var_sim <- function(AA, nu, Sigma_u, nSteps, y0) {
    K <- nrow(Sigma_u)
    Kp <- ncol(AA)
    p <- Kp/K

    if (p > 1) {
        C <- comp_mtrx(AA) # form the  companion matrix of the var(p) process
    } else {
        C <- AA
    }
    y_t <- matrix(0, nrow =  nSteps, ncol=Kp) #trajectories matrix nSteps x Kp
    y_t[1, 1:Kp] <- y0 #add initial value to initiate the simulation
    noise <- mvrnorm(n = nSteps, mu = rep(0, K), Sigma = Sigma_u) #assuming that
    #residuals follow a multivariate normal distribution

    for (t in 2:nSteps) {
        y_t[t, ] <- C %*% y_t[t-1, ]
        y_t[t, 1:K] <- y_t[t, 1:K] + nu + noise[t,]
    }

    y_t <- zoo(y_t[,1:K], 1:nSteps)
    return(y_t)
  }
```

Estimates

Estimates of coefficents

We first define a formula to compute Z values, which are key parameters for both estimating coefficient & auto-correlation matrix.

```
par_estimate <- function(y_t, p=1) {
  nObs <- nrow(y_t)  # Number of observations
  K <- ncol(y_t)  # Number of variables
  T <- nObs - p  # Number of usable observations

  # y
  Y <- y_t[(p + 1):nObs, ]  # T x K matrix

  # Z
  Z <- matrix(1, nrow = T, ncol = (K * p + 1))  # Intercept + lagged values

  for (i in 1:p) {
    col_start <- 2 + (i - 1) * K
    col_end <- 1 + i * K
    Z[, col_start:col_end] <- y_t[(p + 1 - i):(nObs - i), ]
  }
```

```
  # Estimate coefficients using OLS: B_hat = (Z'Z)^(-1) Z'Y
  B_hat <- solve(t(Z) %*% Z) %*% t(Z) %*% Y  # (K*p + 1) x K matrix

  return(list(
    Y = Y,
    Z = Z,
    B_hat = B_hat,  # Estimated VAR parameters
    T = T
  ))
}
```

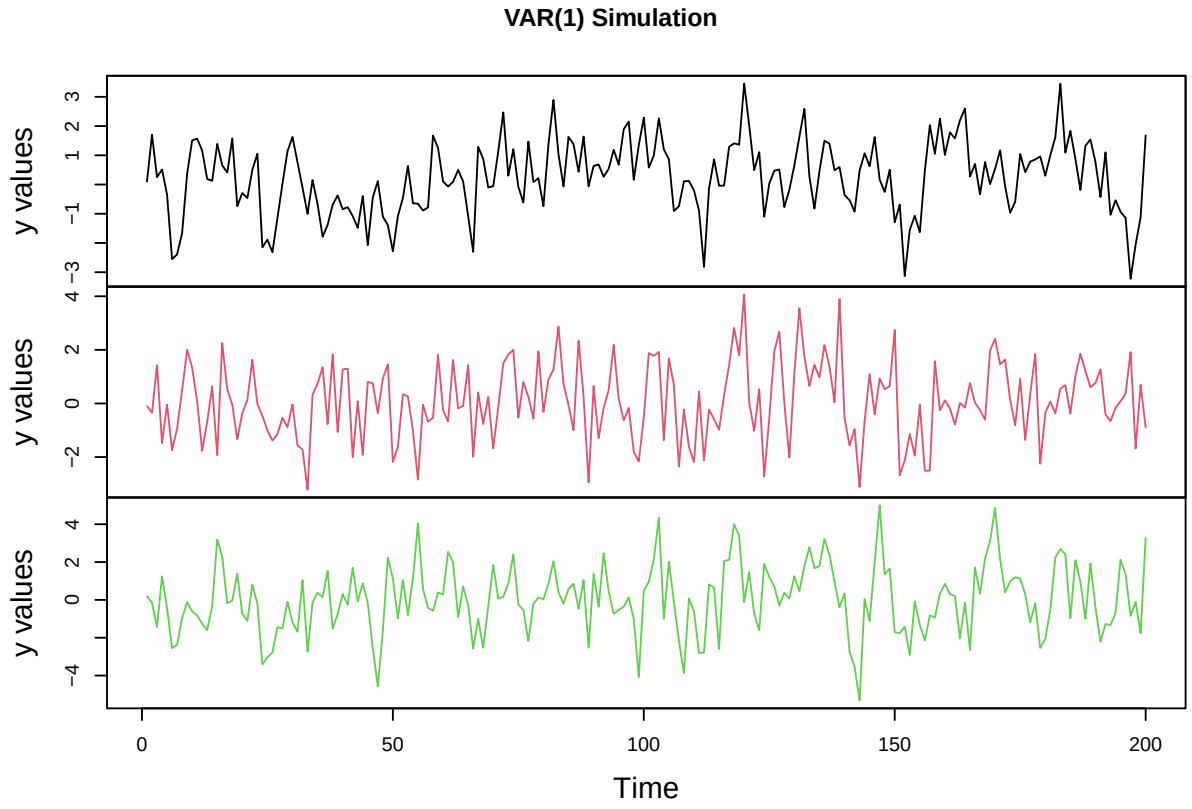
```
estimator<-function(Y, Z, p=1, method = c("standard", "qr", "lsfit")) {

    # Estimator
     method <- match.arg(method)
     if(method == "standard"){
        B_hat <- solve(t(Z) %*% Z, t(Z) %*% Y)
        B_hat <- t(B_hat)
     } else if(method == "qr"){
        qr_decomp <- qr(Z)
        B_hat <- qr.coef(qr_decomp, Y)
     } else if(method == "lsfit"){
       fit <- lsfit(Z, Y)
        B_hat <- fit$coef
     } else {
        stop("Unknown method")
     }
    return(list(nu=B_hat[,1], AA=B_hat[,-1]))
  }
```

Estimates of the autocovariance function

```
est_autocov <- function(y_t, Y, Z, T, p=1){
    K <- ncol(y_t)
    Kp <- K * p
    I_t <- diag(T)

    # QR decomposition of Z to avoid singularity issues
    qr_decomp <- qr(Z)
    Q <- qr.Q(qr_decomp)
    P_Z <- Q %*% t(Q)  # Projection matrix

    # Compute bias-corrected covariance
    bias_sigma <- 1/T * t(Y) %*% (I_t - P_Z) %*% Y

    # Degrees of freedom correction
    d.f. <- T / (T - Kp - 1)
    unbiased <- d.f. * bias_sigma  # Corrected covariance estimate

    return(unbiased)
}
```

Test A

```
set.seed(123)
  p<-1
  A<-matrix(c(0.5, 0.1, 0., 0., 0.1, 0.2, 0., 0.3, 0.3), nrow=3, ncol=3)
  nu<-matrix(c(0.05, 0.02, 0.04), nrow=3)
  Sigma_u <- matrix(c(1.0, 0.2, 0.3, 0.2, 2.0, 0.5, 0.3, 0.5, 3.0), nrow = 3, ncol = 3)
  nSteps <- 200
  y0 <- matrix(c(0.1, -0.1, 0.2),ncol=ncol(A))

#compute trajectories
y_t_a<-var_sim(A, nu, Sigma_u, nSteps, y0)
plot(y_t_a, main = "VAR(1) Simulation", xlab = "Time", ylab = "y values", col = 1:3, lty = 1)
```



Multivariate Least Squares Estimators

```
#estimate parameters
par_A<-par_estimate(y_t_a)
#estimate coefficient

test_A<-estimator(par_A$Y, par_A$Z)
auto_cov_A<-est_autocov(y_t_a, par_A$Y, par_A$Z, par_A$T)
```

Now we will compare original input for the simulation & estimated values

Coefficient Matrix

```
A_true <- A
A_est <- test_A$AA

diff_A <- A_true - A_est
print(diff_A)
```

```
##             [,1]        [,2]        [,3]
## [1,] -0.03817054 -0.03878324  0.01855267
## [2,] -0.07605193  0.07480504  0.04831467
## [3,] -0.15715522  0.05525142 -0.01811021
```

Intercept Matrix

```
nu_true <- nu  # True nu from the simulation
nu_est <- test_A$nu  # Estimated nu

diff_nu <- nu_true - nu_est  # Compute the difference
print(diff_nu)
```

```
##              [,1]
## [1,] -0.064343595
## [2,] -0.001342372
## [3,]  0.062053220
```

Covariance Matrix

```
Sigma_u_true <- Sigma_u  # True covariance matrix
Sigma_u_est <- auto_cov_A  # Estimated autocovariance

diff_Sigma_u <- Sigma_u_true - Sigma_u_est  # Compute the difference
print(diff_Sigma_u)
```

```
##            [,1]        [,2]        [,3]
## x.1 -0.09156416 -0.05131517 -0.07289564
## x.2 -0.05131517  0.27497818  0.05364027
## x.3 -0.07289564  0.05364027  0.32281594
```

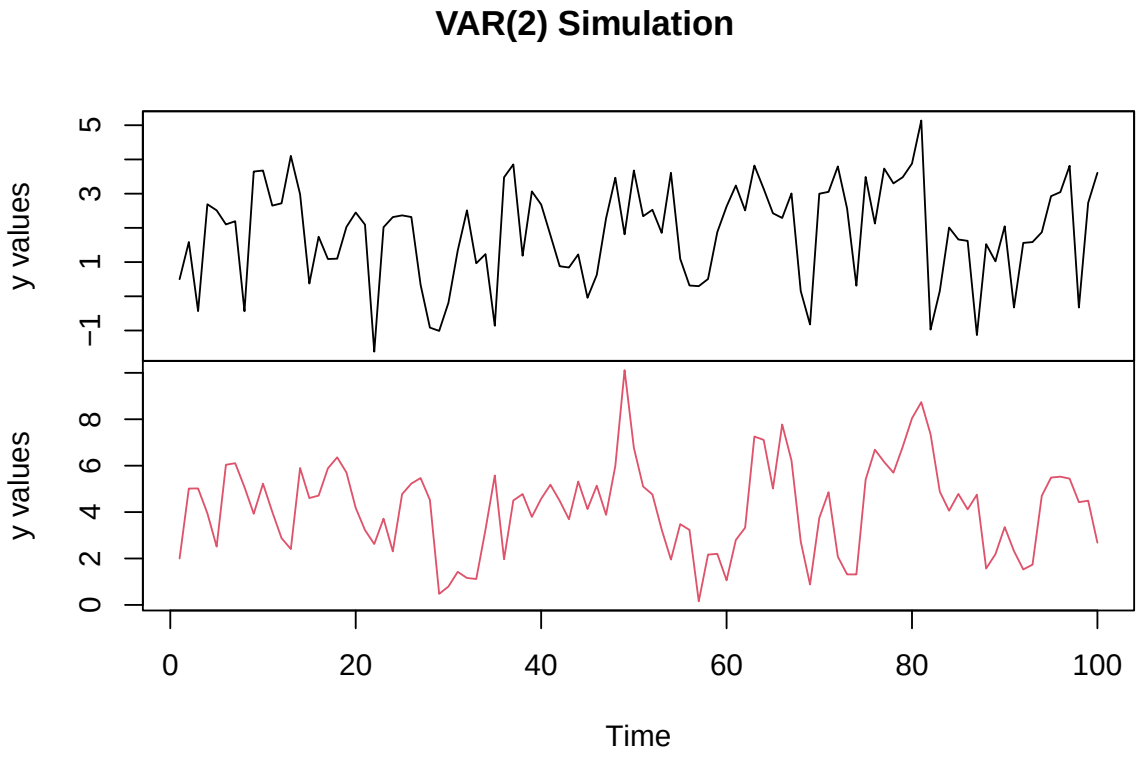
Test B

Kp-dimensional representation for VAR(p) is defined as following:

$$
\begin{aligned}
& Y_t = \nu + \mathbf{A} Y_{t-1} + U_t \\
\text{where} \quad & Y_t := \begin{bmatrix} y_t \\ y_{t-1} \\ \vdots \\ y_{t-p+1} \end{bmatrix}, \quad \nu := \begin{bmatrix} \nu \\ 0 \\ (Kp \times 1 \text{ zeros}) \end{bmatrix}, \\
& \mathbf{A} := \begin{bmatrix} A_1 & A_2 & \dots & A_{p-1} & A_p \\ I_K & 0 & \dots & 0 & 0 \\ 0 & I_K & \dots & 0 & 0 \\ \vdots & & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & I_K & 0 \end{bmatrix}, \quad U_t := \begin{bmatrix} u_t \\ 0 \\ \vdots \\ 0 \end{bmatrix}
\end{aligned}
$$

We first start with the simulation

```r
# Define parameters for Test B
set.seed(123)
p_B <- 2  # Number of lags for VAR(p)
nSteps_B <- 100
K_B <- 2  # Number of variables (size of y_t)
Sigma_u_B <- matrix(c(2, 0.3, 0.3, 3), nrow = 2, ncol = 2)
y0_B <- c(0.5, 2, 1, 5)
nu_int_B <- matrix(c(0.5, 0.9), nrow = 2)

A_1 <- matrix(c(0.5, 0.4, 0.1, 0.5), nrow = 2, ncol = 2)
A_2 <- matrix(c(0, 0.25, 0, 0), nrow = 2, ncol = 2)
AA <- cbind(A_1, A_2)

# Simulate time series for Test B
y_t_B <- var_sim(AA, nu_int_B, Sigma_u_B, nSteps_B, y0_B)
plot(y_t_B, main = "VAR(2) Simulation", xlab = "Time", ylab = "y values", col = 1:2, lty = 1)
```



and then estimates

```
# Estimate the parameters for Test B
par_B <- par_estimate(y_t_B, p = p_B)

# Estimate coefficients
test_B <- estimator(par_B$Y, par_B$Z)

# Display estimated coefficients for Test B
A_est_B <- test_B$AA
nu_est_B <- test_B$nu

# Estimate autocovariance matrix for Test B
auto_cov_B <- est_autocov(y_t_B, par_B$Y, par_B$Z, par_B$T, p = p_B)

# Display the estimated autocovariance matrix
auto_cov_B
```

```
##          [,1]      [,2]
## x.1 2.0481803 0.4062966
## x.2 0.4062966 2.2094853
```

Coefficient Matrix

```r
# True and estimated coefficient matrix for Test B
A_true_B <- AA  # True coefficient matrix
diff_A_B <- A_true_B - A_est_B  # Difference between true and estimated A

print("Difference in coefficient matrix (A):")
```

```
## [1] "Difference in coefficient matrix (A):"
```

```r
print(diff_A_B)
```

```
##           [,1]        [,2]       [,3]       [,4]
## [1,] 0.2530068  0.04676418 0.02766603 0.02474372
## [2,] 0.0581829 -0.08026600 0.16638178 0.11756736
```

Intercept Matrix

```r
# True and estimated intercept matrix for Test B
nu_true_B <- nu_int_B  # True intercept vector
diff_nu_B <- nu_true_B - nu_est_B  # Difference between true and estimated nu

print("Difference in intercept matrix (nu):")
```

```
## [1] "Difference in intercept matrix (nu):"
```

```r
print(diff_nu_B)
```

```
##           [,1]
## [1,] -0.8486765
## [2,] -0.5523156
```

Covariance Matrix

```r
# True and estimated covariance matrix for Test B
Sigma_u_true_B <- Sigma_u_B  # True covariance matrix
diff_Sigma_u_B <- Sigma_u_true_B - auto_cov_B  # Difference between true and estimated covariance

print("Difference in covariance matrix (Sigma_u):")
```

```
## [1] "Difference in covariance matrix (Sigma_u):"
```

```r
print(diff_Sigma_u_B)
```

```
##           [,1]       [,2]
## x.1 -0.04818035 -0.1062966
## x.2 -0.10629660  0.7905147
```

Refenence

Lütkepohl (2005)

Lütkepohl, Helmut. 2005. *New Introduction to Multiple Time Series Analysis*. Springer Books 978-3-540-27752-1. Springer. https://doi.org/10.1007/978-3-540-27752-1.