# Implementation Multiple Time Series Analysis

Jacopo Lussetti

2025-04-02

## Companion Matrix function

```r
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:

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

```r
  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.

```r
  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.

```r
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
  ))
}
```

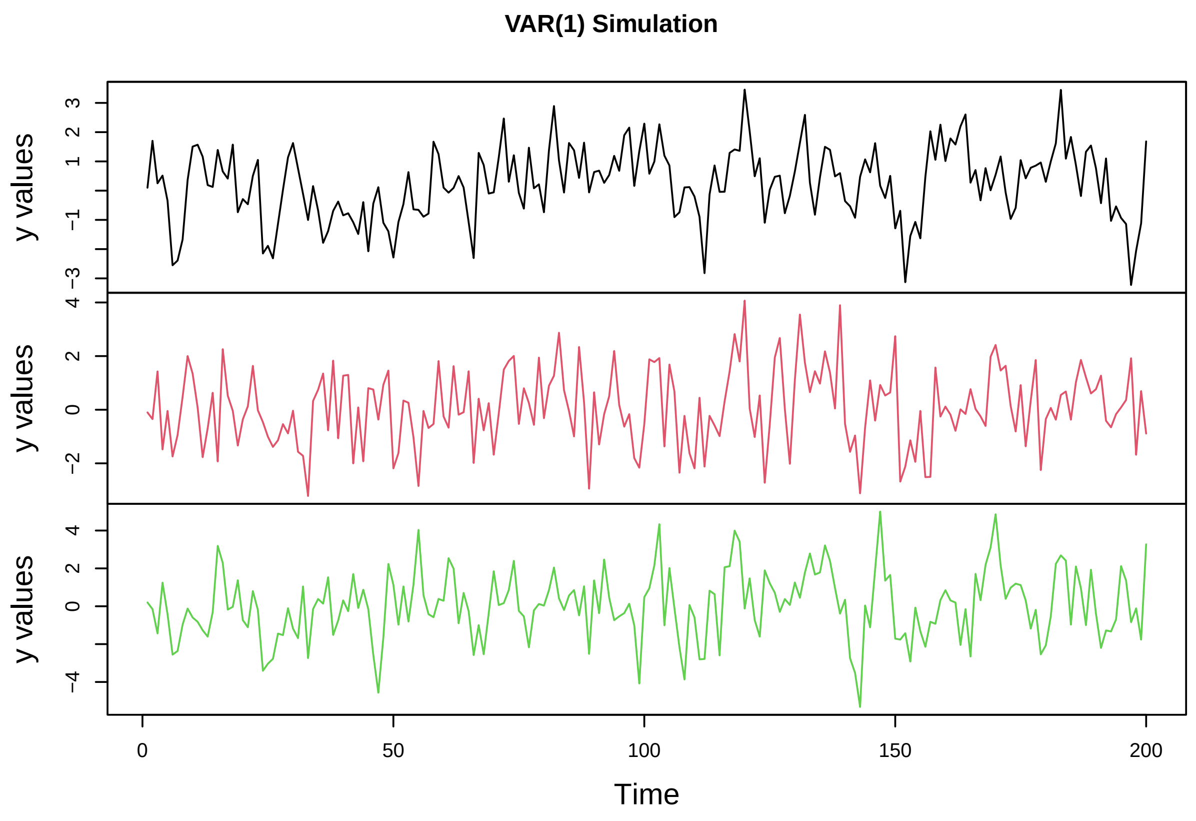
```
estimator<-function(Y, Z, p=1, method = c("standard", "qr", "lsfit")) {

    # 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

```r
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

```r
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

```r
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:

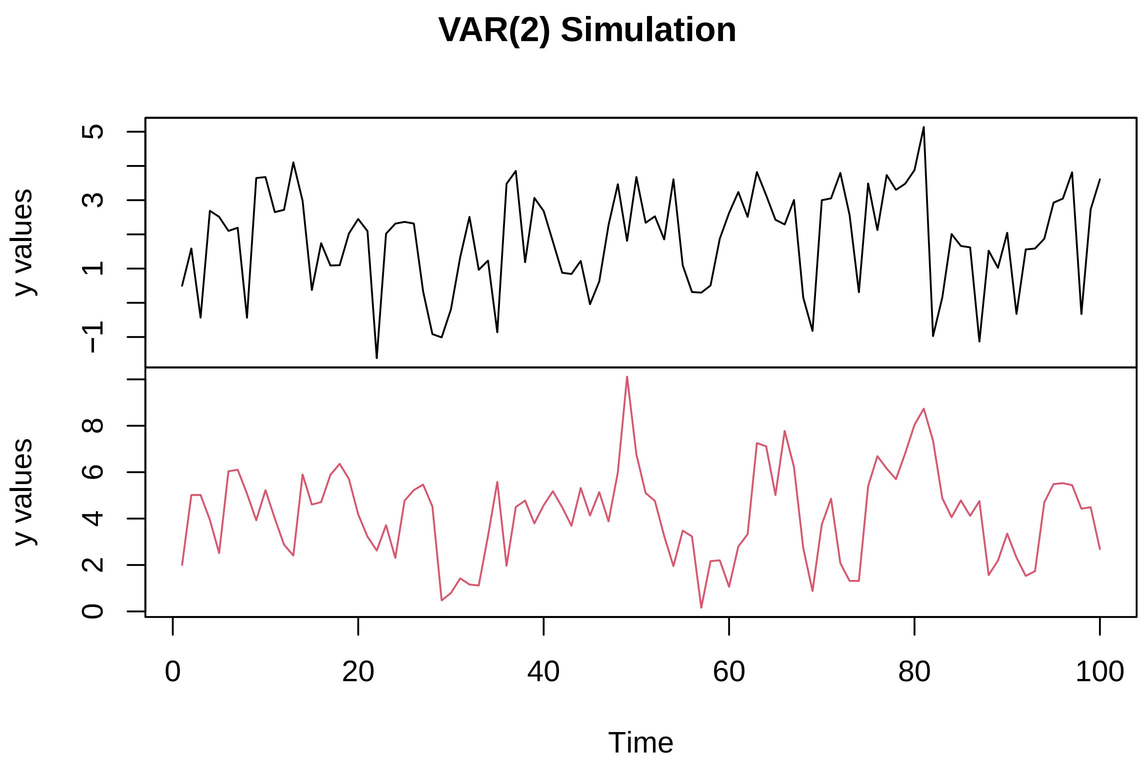
$$Y_t = \nu + \mathbf{A}Y_{t-1} + U_t$$

$$\text{where} \quad Y_t := \begin{bmatrix} y_t \\ y_{t-1} \\ \vdots \\ y_{t-p+1} \end{bmatrix}, \quad \nu := \begin{bmatrix} \nu \\ 0 \\ (Kp \times 1 \text{ zeros}) \end{bmatrix},$$

$$\mathbf{A} := \begin{bmatrix} A_1 & A_2 & \dots & A_{p-1} & A_p \\ I_K & 0 & \dots & 0 & 0 \\ 0 & I_K & \dots & 0 & 0 \\ \vdots & & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & I_K & 0 \end{bmatrix}, \quad U_t := \begin{bmatrix} u_t \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

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
```

## VAR(1) simulation with sparese coefficient Matrix

1. **VAR(1) Simulation:**
   Generate data from the process

$$y_t = Ay_{t-1} + u_t, \quad u_t \sim \mathcal{N}(0, \sigma^2 I).$$

   where A is a lower triangular matrix with sparsity on the top right.

2. **Estimation via LASSO:**
   Apply LASSO regression to estimate the autoregressive matrix $A$ from the simulated data through R funtion *glmnet*.

3. **Optimisation of the tuning parameter**
   Through cross-validation or information criterion, and we compare strategies

4. **Monte Carlo Repetition**

5. **Evaluation of Errors:**
   compute Type I and Type II error rates in detecting zero vs. nonzero coefficients.

6. **Increase dimension**
   We try larger matrices by increasing K, and we observe how LASSO's performance changes as dimension increases.

### VAR(1) Simulation

```r
set.seed(1234)
stab_test <- function(kp, A, tol = 1e-8)
{
  if (!is.matrix(A) || nrow(A) != ncol(A)) {
    stop("The matrix is not square")
  }
  eig <- eigen(A, only.values = TRUE)$values  # computing the eigenvalues

  for (i in 1:length(eig)) {
    if (Mod(eig[i]) >= 1 - tol) { # Mod also handles complex numbers
      return(FALSE)                # <-- fixed typo "returm"
    }
  }
  return(TRUE)
}


A_sim <- function(K, spar, sd, max_tries = 1000){ #paramete spar determine
  #how many elements will be set to zero
  tries <- 0
  repeat{
    A <- matrix(0L, K, K)
    idx_up <- which(upper.tri(A))
    n_up   <- length(idx_up)

    if (n_up > 0) {
      A[idx_up] <- rnorm(n_up, mean = 0, sd = sd)   #fill the upper
  #triangle with r.v. from a normal distribution
  #cancel a certain percentage of the elements
      nmiss <- round(n_up * spar)
      if (nmiss > 0) { # add sparsity
        zero_idx <- sample(idx_up, nmiss, replace = FALSE)
        A[zero_idx] <- 0
      }
    }

    # we apply the formula previously defined to check stability
    if (stab_test(K, A) == TRUE) return(A)
     # else try again
    tries <- tries + 1
    #to prevent infinite loop due to using repeat loop, we set a max number of
    #iterations
    if (tries >= max_tries) {
      stop("Could not generate a stable A within max_tries.")
    }
  }
}

#now we generate a stable upper triangle coef matrix with sparsity
K <- 5
sd_var<-0.1
A <- A_sim(K, spar = 0.1, sd = sd_var, max_tries = 10) # moderate sparsity, modest sd A
stab_test(K,A)
```

```
## [1] TRUE
```

Now we are able to generate a VAR(1) from the function specified at page 3

```
sigma_var<- diag(1,5,5)* sd_var
nu_var <- rep(0, K)
y_0 <- rep(0, K)
T_sim<-10000
var_1<- var_sim(AA=A,nu=nu_var,Sigma_u=sigma_var, nSteps=T_sim, y0=y_0)
```

## Estimation via LASSO

In this step we will apply lasso regression by applying the packages *glmnet*

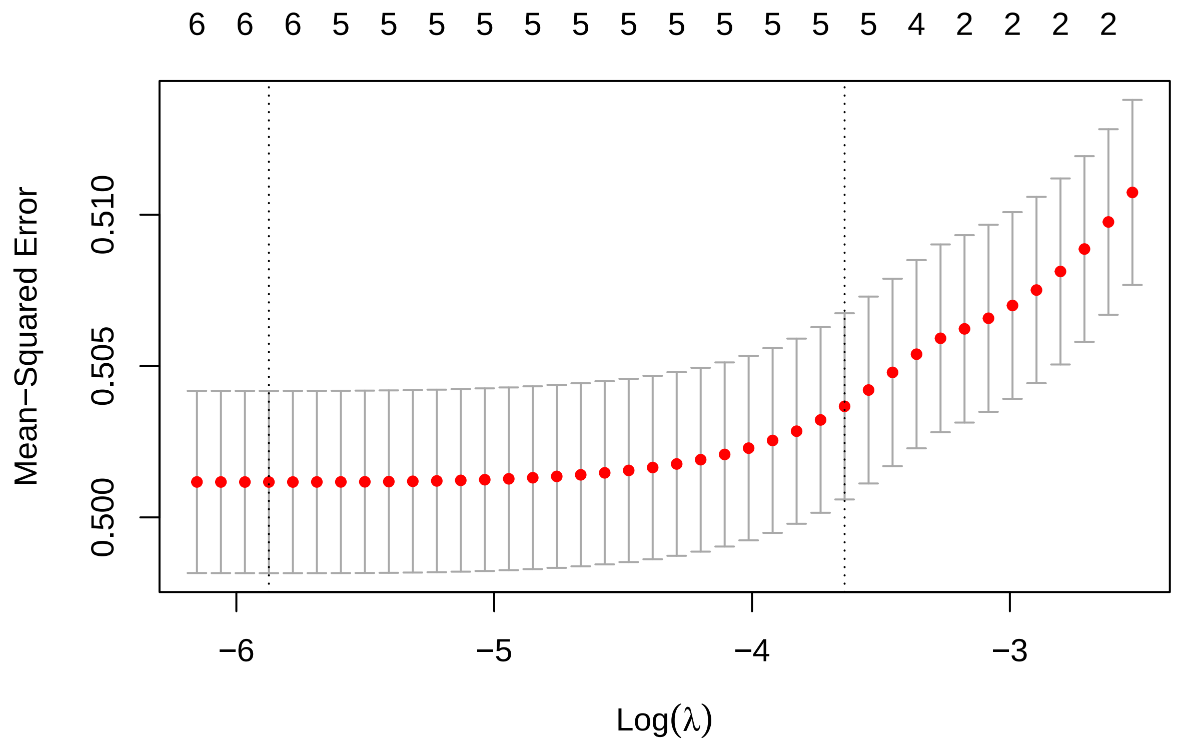
```
# Build lagged design (Z) and aligned response (Y)
Z <- as.matrix(var_1[- nrow(var_1),])  # predictors y_{t-1}
# Drop first row to align (remove the NA created by lag)
Y <- as.matrix(var_1[-1, , drop = FALSE])
# 70/30 split
n<-nrow(Z)
cut_off <- round(0.70 * n)
X_train <- Z[1:cut_off, , drop = FALSE]
Y_train <- Y[1:cut_off, , drop = FALSE]
X_test <- Z[(cut_off + 1):n, , drop = FALSE]
```

```r
Y_test <- Y[(cut_off + 1):n, , drop = FALSE]
# Manual CV over a simple lambda grid (avoid 0 exactly)
lambdas <- seq(0,1, 0.001)
val_error <- data.frame(lambda = lambdas, error = NA_real_)
for (i in seq_along(lambdas)) {
lmbd <- lambdas[i]
fit <- glmnet(X_train, Y_train, family = "mgaussian", lambda = lmbd)
pred <- predict(fit, newx = X_test, s = lmbd)[,,1]
mse_per_series <- colMeans((Y_test - pred)^2)
val_error$error[i] <- mean(mse_per_series)
}
best_lambda_1 <- val_error$lambda[which.min(val_error$error)]
best_lambda_1
```

```
## [1] 0.003
```

We try with default cross-validation from glmnet

```r
cv_fit <- cv.glmnet(
  x = as.matrix(Z),
  y = as.matrix(Y),
  family = "mgaussian"  # var us a mutlivariate regression
)

plot(cv_fit)
```



```r
best_lambda_2 <- cv_fit$lambda.1se # lambda.min brings results that are
#far not optimal, and have higher type I error
best_lambda_2
```

```
## [1] 0.02622498
```

Discrimination of lambda based on the mellow $C_p$ criteria, BIC and AIC defined as follows:

Table 1: Summary of model selection criteria based on degrees of freedom

| **Criterion** | **Formula** |
|---|---|
| $C_p$ | $\|y-\hat{\mu}\|^2 + 2\,\tau^2\,df$ |
| AIC | $N\log(2\pi\tau^2) + \dfrac{\|y-\hat{\mu}\|^2}{\tau^2} + 2\,df$ |
| AICc | $N\log\left(2\pi\,\dfrac{\|y-\hat{\mu}\|^2}{N}\right) + \dfrac{N-2N\,df}{N-df-1}$ |
| BIC | $N\log(2\pi\tau^2) + \dfrac{\|y-\hat{\mu}\|^2}{\tau^2} + (\log N)\,df$ |
| GCV | $\dfrac{1}{N}\,\dfrac{\|y-\hat{\mu}\|^2}{(1-df/N)^2}$ |

When we have normal residuals and no sparsity, the d.f can be computed as follows:

$$
\begin{aligned}
d.f. = \Sigma_{i=1}^{N}\frac{\text{cov}(\hat{\mu}_i, y_i)}{\sigma^2} \\
\text{tr}(X(X^TX)^{-1}X^T) = \\
\text{rank}(X)
\end{aligned}
$$

```r
criteria_df<-function(X,Y, p,steps){
  n<-nrow(X)
  #recall that the min lambda for gaussian is C'sigma sqrt(log(p)/n)
  C_i <- apply(X, 2, function(col) sqrt(sum(col^2)) / sqrt(n))
  C <- max(C_i)
    #we compute an initial estimator of sigma from the OLS
  beta_ols <- solve(t(X) %*% X) %*% t(X) %*% Y
  resid <- Y - X %*% beta_ols
  sigma_hat <- sqrt(sum(resid^2) / (n - p))
  lambda_0<- C * sigma_hat * sqrt(log(p) / n) # initial lambda
   #create a seq of candidate lambda with increments = steps
  lambda_seq<-seq(lambda_0, 10*lambda_0, by=steps)
  len_lambda<-length(lambda_seq)
  #create a data frame to store the results
  results<-data.frame(
   lambda_seq=lambda_seq,
   df=as.numeric(rep(0, len_lambda)),
   Cp=as.numeric(rep(0, len_lambda)),
   AIC=as.numeric(rep(0, len_lambda)),
   BIC=as.numeric(rep(0, len_lambda))
  )
  #we compute now the compute estimate by using glmnet
  for(i in 1:length(lambda_seq)){
    fit<-glmnet(
      X,
      Y,
      family = "mgaussian",
      lambda = lambda_seq[i]
    )
    #extract df
    df<-fit$df
    results$df[i]<-df
    #compute sd

    Y_hat<-predict(fit, newx=X)[,,1]
    RSS <- sum((Y - Y_hat)^2)
    sigma_hat<-RSS /(n-df)
    #compute criteria
    results$Cp[i]<-RSS + 2*sigma_hat*df
    results$AIC[i]<-n*log(2*pi*sigma_hat) + RSS/sigma_hat + 2*df
    results$BIC[i]<-n*log(2*pi*sigma_hat) + RSS/sigma_hat + (log(n))*df
  }
    return(results)
    }
test<-criteria_df(Z, Y, p=5, steps=0.0001)
index<-which.min(test$Cp)
print(test[index,])
```

```
##    lambda_seq df       Cp      AIC      BIC
## 8 0.003631403  4 5009.803 21465.74 21494.58
```

```r
lambda_criterion<-test$lambda_seq[index]
```

```r
#function to extract the coefficient
A_hat <- function(coef_list, K) {
  Ahat <- matrix(NA_real_, K, K)
  for (j in 1:K) {
    cj <- as.matrix(coef_list[[j]])
    Ahat[j, ] <- cj[rownames(cj) != "(Intercept)", 1]  # drop intercept
  }
  Ahat
}

#function to compute confusion matrices on true non zeros coefficients
metrics <- function(A_true, A_hat, tol = 5e-3) {
  S_true <- abs(A_true) > tol
  S_est  <- abs(A_hat)  > tol

  TP <- sum(S_true & S_est)
  FN <- sum(S_true & !S_est)
  FP <- sum(!S_true & S_est)
  TN <- sum(!S_true & !S_est)

  FPR <- if ((FP + TN) > 0) FP / (FP + TN) else NA_real_
  FNR <- if ((FN + TP) > 0) FN / (FN + TP) else NA_real_
  Prec <- if ((TP + FP) > 0) TP / (TP + FP) else NA_real_
  Rec  <- if ((TP + FN) > 0) TP / (TP + FN) else NA_real_
  F1   <- if (!is.na(Prec + Rec) && (Prec + Rec) > 0) 2 * Prec * Rec / (Prec + Rec) else NA_real_
  # coefficient MSE (Frobenius): mean squared diff over all K^2 entries
  MSE <- mean((A_true - A_hat)^2)

  c(TypeI = FPR, TypeII = FNR, Recall = Rec, Precision = Prec, F1 = F1, MSE = MSE)
}



#fit the model with the tuning parameters
lambda_cand<-c(best_lambda_1,best_lambda_2,lambda_criterion, 0.03)
fit_opt<-glmnet(
  x=as.matrix(Z),
  y = as.matrix(Y),
  family = "mgaussian",
  lambda =lambda_cand
)

results<-lapply(seq_along(lambda_cand), function(indx){
  lambda_i <- lambda_cand[indx]
  coef_list <- coef(fit_opt, s = lambda_i)
  A_hat<-A_hat(coef_list, K)
  metrix<-metrics(A, A_hat)
  list(
    summary  = data.frame(
    Lambda = lambda_i,
    TypeI = metrix["TypeI"],
    TypeII = metrix["TypeII"],
    Recall = metrix["Recall"],
    Precision = metrix["Precision"],
    F1 = metrix["F1"],
    MSE = metrix["MSE"]
  ),
  A_est=A_hat # we include the estimated matrix as well for separate analysis
  )

})
table_results<- lapply(results, `[[`, "summary") %>%
  do.call(rbind, .)
row.names(table_results)<-c("Manual CV", "glmnet CV", "Criterion df", "Lambda = 0.03")
table_results
```

```
##                     Lambda  TypeI    TypeII    Recall Precision        F1
## Manual CV     0.003000000 0.4375 0.0000000 1.0000000    0.5625 0.7200000
## glmnet CV     0.026224979 0.1250 0.1111111 0.8888889    0.8000 0.8421053
## Criterion df  0.003631403 0.4375 0.0000000 1.0000000    0.5625 0.7200000
## Lambda = 0.03 0.030000000 0.0000 0.2222222 0.7777778    1.0000 0.8750000
##                        MSE
## Manual CV     8.507898e-05
## glmnet CV     1.213101e-03
## Criterion df  9.278225e-05
## Lambda = 0.03 1.559946e-03
```

```
results[[2]]$A_est #visualise estiamte with lambda from cv.glmnet
```

```
##      [,1]          [,2]          [,3]         [,4]         [,5]
## [1,]    0 -0.0347751161  0.0025842415 -0.163961260 -0.014072232
## [2,]    0  0.0004757867  0.0137958978  0.024979587  0.005392964
## [3,]    0  0.0050128362  0.0015494216  0.035708504 -0.014148554
## [4,]    0  0.0020851592 -0.0006714499 -0.002712849 -0.025390209
## [5,]    0  0.0028118032 -0.0010083357  0.004904300  0.001267834
```

```
results[[4]]$A_est #visualise results from the arbitrary larger lambda
```

```
##      [,1]          [,2]          [,3]         [,4]          [,5]
## [1,]    0 -0.0229897174  0.0004214522 -0.152370485 -0.0089188408
## [2,]    0  0.0003173838  0.0022553981  0.023255741  0.0034022332
## [3,]    0  0.0033176857  0.0002547640  0.033196867 -0.0089766459
## [4,]    0  0.0013869488 -0.0001078720 -0.002524500 -0.0161024206
## [5,]    0  0.0018595483 -0.0001648456  0.004551683  0.0008046929
```

```
A
```

```
##      [,1]       [,2]       [,3]        [,4]        [,5]
## [1,]    0 -0.1207066 0.02774292 -0.23456977 -0.05747400
## [2,]    0  0.0000000 0.10844412  0.04291247  0.00000000
## [3,]    0  0.0000000 0.00000000  0.05060559 -0.05644520
## [4,]    0  0.0000000 0.00000000  0.00000000 -0.08900378
## [5,]    0  0.0000000 0.00000000  0.00000000  0.00000000
```

From the results we can see that the best option with the lowest type I error is glmnet CV function with lambda 1 s.d. away from the minimum lambda values.

## Monte Carlo Simulation

To do a MC simulation, I have defined a function that create n random seeds that will be used to generate different sparse true coefficient matrix. This will then used to generate various VAR(1) processes. For each replication, we will estimate the coefficient matrix using LASSO. To optimise the tuning parameter, we will use the built in function *cv.glmnet*, and select the value that is 1 s.d. away from the minimum. Finally, we will compute the Type I and Type II error rates for each replication, and summarize the results across all replications.

```
#define functions that returns
error_function <- function(tol, coef, tr_coef){  # coef = A_hat, tr_coef = A_true
  S_true <- abs(tr_coef) > tol
  S_est  <- abs(coef)    > tol

  TP <- sum(S_true & S_est)
  FN <- sum(S_true & !S_est)
  FP <- sum(!S_true & S_est)
  TN <- sum(!S_true & !S_est)

  FPR  <- FP/(FP+TN)
```

```r
  FNR  <-  FN/(FN+TP)
  Prec <-TP/(TP+FP)
  Rec  <-  TP/(TP+FN)
  F1   <- 2*Prec*Rec/(Prec+Rec)

  data.frame(TypeI = FPR, TypeII = FNR, Recall = Rec, Precision = Prec, F1 = F1)
}


MC_var_1 <- function(K, spar, sd, nu, max_tries = 1000, nrep = 100,
                     base_seed = 123, T_sim = 1000, tol = 5e-3) {

  seeds <- base_seed + seq_len(nrep) - 1

  # storage
  A_list     <- vector("list", nrep)   # store true A per replication
  Ahat_list  <- vector("list", nrep)   # store estimated A_hat
  nu_hat_list <- vector("list", nrep)  # store A_hat - A_true per replication

  results_comb <- data.frame(
    TypeI     = numeric(nrep),
    TypeII    = numeric(nrep),
    Recall    = numeric(nrep),
    Precision = numeric(nrep),
    F1        = numeric(nrep)
  )

  for (b in seq_len(nrep)) {
    set.seed(seeds[b])

    # (1) draw a stable sparse A
    A_true <- A_sim(K, spar = spar, sd = sd, max_tries = max_tries)

    # (2) simulate VAR(1)
    sigma_var <- diag(1, K, K)
    nu_var    <- rep(nu, length.out = K)
    y_0       <- rep(0, K)
    var_1 <- var_sim(AA = A_true, nu = nu_var, Sigma_u = sigma_var,
                     nSteps = T_sim, y0 = y_0)

    # (3) create lag matrices
    Y <- coredata(var_1)[-1, ]           # y_t
    Z <- coredata(var_1)[-nrow(var_1),] # y_{t-1}

    # (4) pick lambda via CV
    cv_fit <- cv.glmnet(x = as.matrix(Z), y = as.matrix(Y), family = "mgaussian")
    lam <- cv_fit$lambda.1se

    # (5) fit model and extract A_hat
    fit_best  <- glmnet(x = as.matrix(Z), y = as.matrix(Y),
                        family = "mgaussian", lambda = lam)
    coef_list <- coef(fit_best)
    A_est <- matrix(NA_real_, K, K)
    for (i in seq_len(K)) {
      ci <- as.matrix(coef_list[[i]])[-1, 1]   # drop intercept
      A_est[i, ] <- ci
    }

    # (6) compute difference (nu_hat = bias)
    nu_hat <- A_est - A_true

    # (7) metrics
    res_b <- error_function(tol = tol, coef = A_est, tr_coef = A_true)
    results_comb[b, ] <- res_b[1, ]

    # store
    A_list[[b]]     <- A_true
    Ahat_list[[b]]  <- A_est
    nu_hat_list[[b]] <- nu_hat
  }

  # summarize
  summary_df <- data.frame(
    metric = names(results_comb),
    mean   = sapply(results_comb, mean, na.rm = TRUE),
    sd     = sapply(results_comb, sd,   na.rm = TRUE),
    se     = sapply(results_comb, function(x) sd(x, na.rm = TRUE)/sqrt(sum(!is.na(x))))
  )

  # return
  list(
    results = results_comb,
    summary = summary_df,
    A_true  = A_list,
    A_hat   = Ahat_list,
    nu_hat  = nu_hat_list,
    seeds   = seeds
  )
}

  out_5 <- MC_var_1(K = 5, spar = 0.1, sd = 0.1, nu = 0, nrep = 1000, T_sim = 1000)
  out_5$summary
```

```
##              metric       mean         sd          se
## TypeI         TypeI 0.02788113 0.06979017 0.002206959
## TypeII       TypeII 0.87072619 0.23868982 0.007548035
## Recall       Recall 0.12927381 0.23868982 0.007548035
## Precision Precision 0.76271671 0.18370949 0.011038034
## F1               F1 0.54028985 0.16191247 0.009728377
```

## Increasing number of K

We first try with K=10

```r
out_10 <- MC_var_1(K = 10, spar = 0.1, sd = 0.1, nu = 0, nrep = 1000, T_sim = 1000)
out_10$summary
```

```
##              metric      mean         sd          se
## TypeI         TypeI 0.2296692 0.12353763 0.003906603
## TypeII       TypeII 0.3417093 0.19670050 0.006220216
## Recall       Recall 0.6582907 0.19670050 0.006220216
## Precision Precision 0.6723413 0.09137448 0.002912912
## F1               F1 0.6478061 0.10316453 0.003288765
```

We first try with K=15

```r
out_15 <- MC_var_1(K = 15, spar = 0.1, sd = 0.1, nu = 0, nrep = 1000, T_sim = 1000)
out_15$summary
```

```
##              metric      mean         sd           se
## TypeI         TypeI 0.3964835 0.10266690 0.0032466125
## TypeII       TypeII 0.1585117 0.07576975 0.0023960497
## Recall       Recall 0.8414883 0.07576975 0.0023960497
## Precision Precision 0.5983873 0.05093258 0.0016106297
## F1               F1 0.6951704 0.03118016 0.0009860034
```