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

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

```r
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

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

```r
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
  ))
}
```

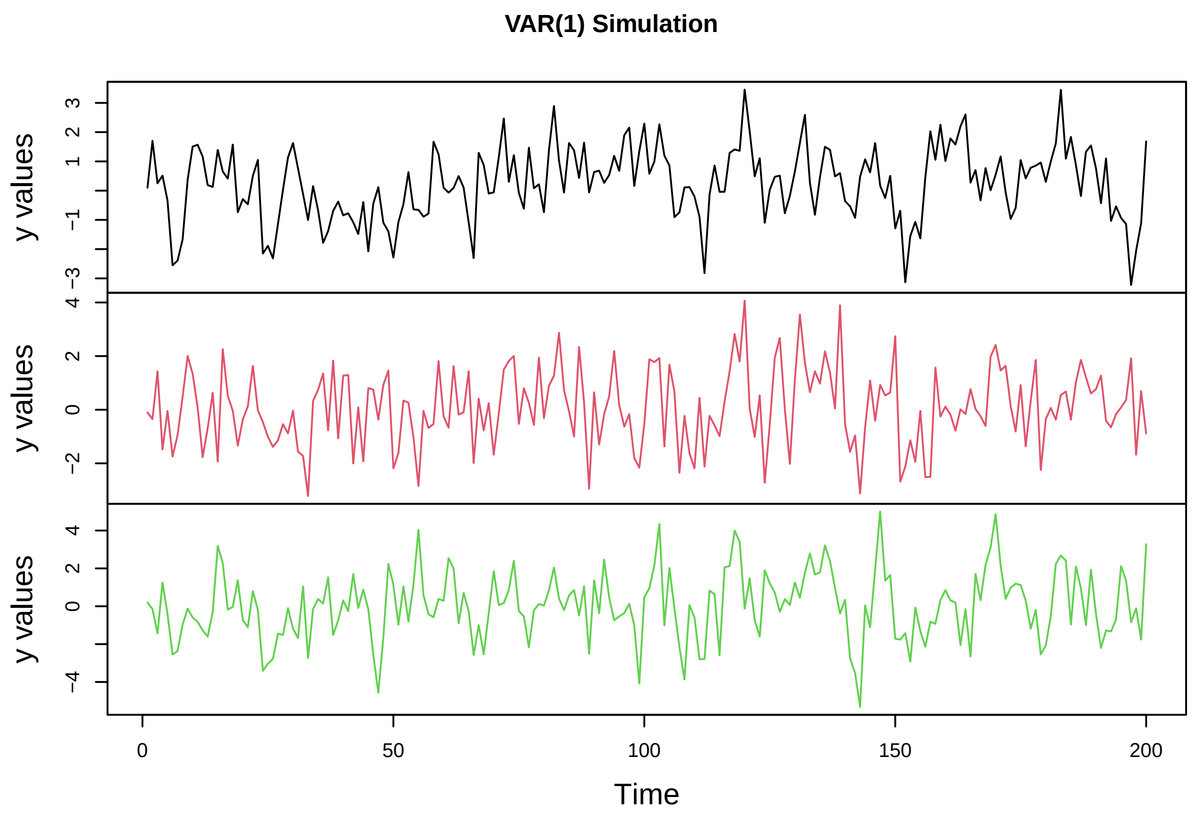
```r
estimator<-function(Y, Z, p=1, method = c("standard", "qr", "lsfit")) {

    # 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

```r
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

```r
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

```r
#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:

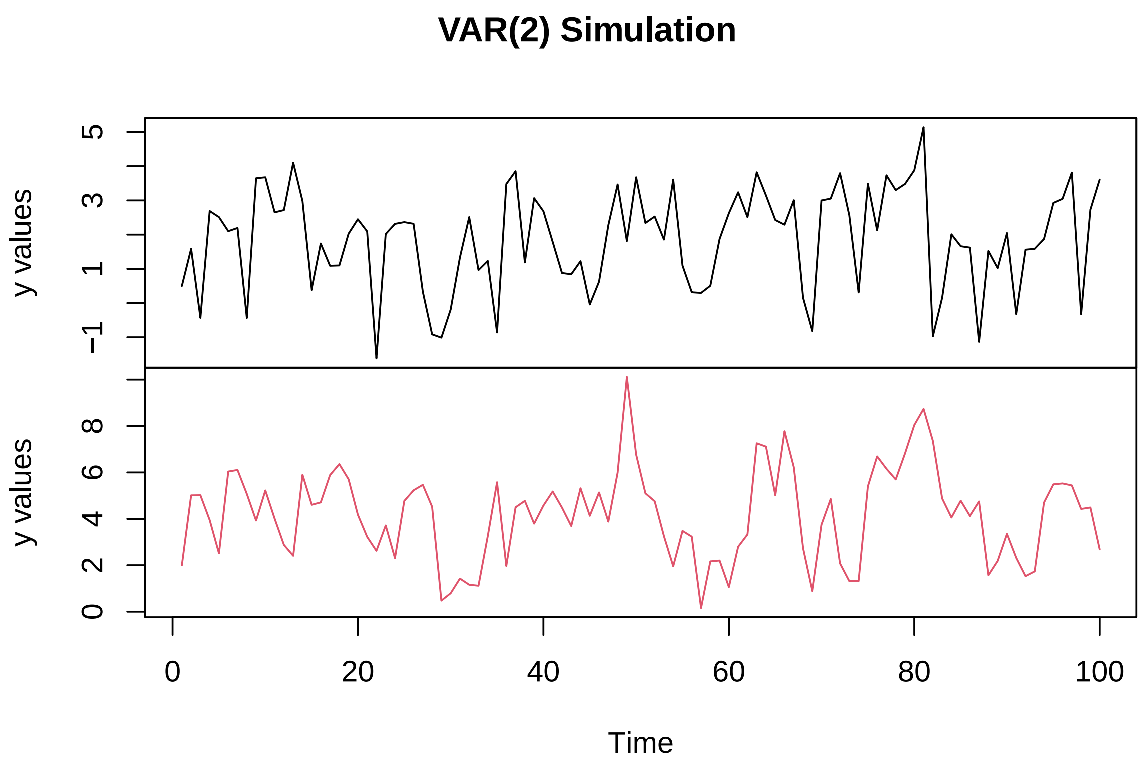
$$Y_t = \nu + \mathbf{A}Y_{t-1} + U_t$$

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

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

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){
  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
      }
    }

    # check whether the model is stable (INSIDE the repeat loop)
    if (stab_test(K, A) == TRUE) return(A)

    tries <- tries + 1
    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 = 0.3, 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

```r
sigma_var<- diag(1,5,5)* sd_var
nu_var <- rep(1, 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*

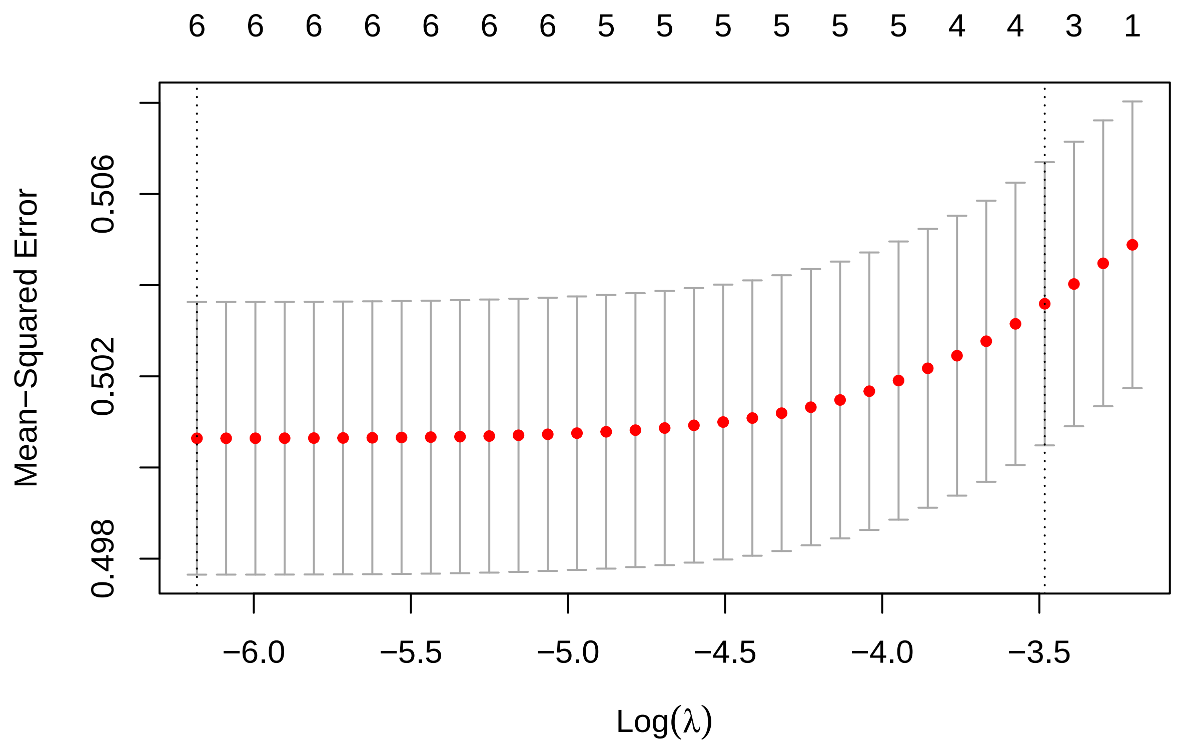
```r
# Build lagged design (Z) and aligned response (Y)
Z <- coredata(lag(var_1, k = -1))   # predictors y_{t-1}
Y_full <- coredata(var_1)                 # responses  y_t

# Drop first row to align (remove the NA created by lag)
Y <- Y_full[-1, , drop = FALSE]

# If univariate, coerce to matrix to allow 2D subsetting later
if (is.null(dim(Z))) Z <- matrix(Z, ncol = 1)
if (is.null(dim(Y))) Y <- matrix(Y, ncol = 1)

# Chronological split (70/30)
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]
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] 0.002
```

We try with default cross-validation from glmnet

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

plot(cv_fit)
```



```
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] 0.03071886
```

```
fit_opt<-glmnet(
  x=as.matrix(Z),
  y = as.matrix(Y),
  family = "mgaussian",
  lambda = c(best_lambda_1,best_lambda_2)
)
A_hat_114<-coef(fit_opt, s=best_lambda_1)
A_hat_129<- coef(fit_opt, s = best_lambda_2)
#we merge estimates in unique A_hat
A_hat_matrix_1<-matrix(NA, nrow=K, ncol=K)
for(i in 1:5){
  ci <- as.matrix(A_hat_114[[i]])           # ith response
lag_rows <- rownames(ci) != "(Intercept)"  # drop intercept
A_hat_matrix_1[i, ] <- ci[lag_rows, 1]
}
A_hat_matrix_1
```

```
##                 [,1]          [,2]         [,3]         [,4]         [,5]
## [1,] -0.0001786303 -0.1182278661  0.025777383 -0.008025843 -0.059850965
## [2,]  0.0057772827  0.0179550374  0.090117131  0.029666248  0.015684294
## [3,] -0.0063304819 -0.0028146054 -0.003984287  0.046454109 -0.007805003
## [4,] -0.0142927874  0.0009652047 -0.003934388 -0.008665567 -0.101623272
## [5,]  0.0026490078  0.0150541075 -0.008829143 -0.023946439 -0.012265350
```

```
A_hat_matrix_2<-matrix(NA, nrow=K, ncol=K)
for(i in 1:5){
ci <- as.matrix(A_hat_129[[i]])           # ith response
lag_rows <- rownames(ci) != "(Intercept)"  # drop intercept
A_hat_matrix_2[i, ] <- ci[lag_rows, 1]
}
A_hat_matrix_2
```

```
##      [,1]          [,2]          [,3] [,4]         [,5]
## [1,]    0 -0.0303206270  0.0010677323    0 -0.014539411
## [2,]    0  0.0047089761  0.0037204063    0  0.003675622
## [3,]    0 -0.0005223395 -0.0001484643    0 -0.001940271
## [4,]    0  0.0002349195 -0.0001534088    0 -0.024641372
## [5,]    0  0.0037579949 -0.0003701004    0 -0.002947628
```

Now we check

```
# --- Support function
tol <- 1e-3
S_true  <- abs(A)                   > tol
S_est_1 <- abs(A_hat_matrix_1)    > tol
S_est_2 <- abs(A_hat_matrix_2)    > tol

# --- Confusion counts
TP_1 <- sum(S_true & S_est_1)
FN_1 <- sum(S_true & !S_est_1)
FP_1 <- sum(!S_true & S_est_1)
TN_1 <- sum(!S_true & !S_est_1)

TP_2 <- sum(S_true & S_est_2)
FN_2 <- sum(S_true & !S_est_2)
FP_2 <- sum(!S_true & S_est_2)
TN_2 <- sum(!S_true & !S_est_2)

# --- Rates (guard divisions) ---
FPR_1 <- if ((FP_1+TN_1)>0) FP_1/(FP_1+TN_1) else NA_real_
FPR_2 <- if ((FP_2+TN_2)>0) FP_2/(FP_2+TN_2) else NA_real_

FNR_1 <- if ((FN_1+TP_1)>0) FN_1/(FN_1+TP_1) else NA_real_
FNR_2 <- if ((FN_2+TP_2)>0) FN_2/(FN_2+TP_2) else NA_real_

Prec_1 <- if ((TP_1+FP_1)>0) TP_1/(TP_1+FP_1) else NA_real_
Prec_2 <- if ((TP_2+FP_2)>0) TP_2/(TP_2+FP_2) else NA_real_

Rec_1  <- if ((TP_1+FN_1)>0) TP_1/(TP_1+FN_1) else NA_real_
Rec_2  <- if ((TP_2+FN_2)>0) TP_2/(TP_2+FN_2) else NA_real_

f1_1 <- if (!is.na(Prec_1+Rec_1) && (Prec_1+Rec_1)>0) 2*Prec_1*Rec_1/(Prec_1+Rec_1) else NA_real_
```

```
f1_2 <- if (!is.na(Prec_2+Rec_2) && (Prec_2+Rec_2)>0) 2*Prec_2*Rec_2/(Prec_2+Rec_2) else NA_real_

# Result Tables
results <- data.frame(
  Model     = c(paste0("Lambda=", best_lambda_1), paste0("Lambda=",best_lambda_2)),
  TypeI     = c(FPR_1, FPR_2),
  TypeII    = c(FNR_1, FNR_2),
  Recall    = c(Rec_1, Rec_2),
  Precision = c(Prec_1, Prec_2),
  F1        = c(f1_1, f1_2)
)
results
```

```
##                       Model     TypeI    TypeII    Recall Precision        F1
## 1              Lambda=0.002 0.8888889 0.0000000 1.0000000 0.3043478 0.4666667
## 2 Lambda=0.0307188553843202 0.2777778 0.2857143 0.7142857 0.5000000 0.5882353
```

From results we can see that using the largest lambda s.t. the error is within one standard error of the minimum (lambda.1se) gives better results in terms of type I error, precision and F1 score.

Monte Carlo Simulation

```
#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)
  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 = 1e-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
```

```r
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
}
```

```r
  # 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 = 0.3, sd = 0.1, nu = 0, nrep = 1000, T_sim = 1000)
  out_5$summary
```

```
##              metric       mean         sd          se
## TypeI         TypeI 0.03232281 0.08866886 0.002803955
## TypeII       TypeII 0.91208571 0.21879279 0.006918836
## Recall       Recall 0.08791429 0.21879279 0.006918836
## Precision Precision 0.53499294 0.16863644 0.013168294
## F1               F1 0.51109846 0.15494049 0.012098819
```

Increasing number of K

We first try with K=10

```r
out_10 <- MC_var_1(K = 10, spar = 0.3, sd = 0.1, nu = 0, nrep = 1000, T_sim = 1000)
out_10$summary
```

```
##              metric      mean         sd          se
## TypeI         TypeI 0.2644698 0.17375825 0.005494718
## TypeII       TypeII 0.4534531 0.27667048 0.008749089
## Recall       Recall 0.5465469 0.27667048 0.008749089
## Precision Precision 0.5022286 0.09156717 0.003059044
## F1               F1 0.5208759 0.10498692 0.003507367
```

We first try with K=15

```r
out_15 <- MC_var_1(K = 15, spar = 0.3, sd = 0.1, nu = 0, nrep = 1000, T_sim = 1000)
out_15$summary
```

```
##              metric      mean         sd          se
## TypeI         TypeI 0.5133506 0.13765024 0.004352883
## TypeII       TypeII 0.1590695 0.11020043 0.003484844
## Recall       Recall 0.8409305 0.11020043 0.003484844
## Precision Precision 0.4458865 0.04892344 0.001547095
## F1               F1 0.5762555 0.03667695 0.001159827
```