Regularised Cov. Estimation

Jacopo Lussetti

2025-11-01

This report is aiming at replicating the methodology by Basu and Michailidis [2015] on estimating covariance matrices and their precision matrices for high-dimensional data.

Previous studies on the covariance estimator(Chen et al. [2013]) found out that the functional dependence measure scale with the spectral radius $\rho(A)$. We define functional dependence measure scale quantify the magnitude of time dependency between current and past observations. More formally,

$$\theta_{i,w,j} = \|Z_{ji} - Z'_{ji}\|_w = \left(\mathbb{E}|Z_{ji} - Z'_{ji}|^w\right)^{\frac{1}{w}} \tag{1}$$

where Z_{ji} are stationary process, and $Z'_{ji} = g(\mathcal{F}'_\rangle)$ with the filtration being defined as $\mathcal{F}^i(\dots e_{-1}, e'_0, e_1, \dots e_i)$, for e'_0 is an independent copy of the original innovation. Additionally, we also consider the Short-Range Dependence:

$$\Theta_{mw} = \max_{1\leq j\leq p} \sum_{\ell=m}^{\infty} \theta_{i,w,j} < \infty \tag{2}$$

In example 2.2 on stationary linear processes, he considers $\mathbf{z}_i = \sum_{m=0}^{\infty} A_m e_{i-m}$ with $\Sigma_e = \mathbb{E}(\mathbf{e}_i\mathbf{e}_i^T), \Sigma_{z_i} = \sum_{m=0}^{\infty} A_m \Sigma_e A_m^T$. We assume that e_{ij} i iid with mean 0, variance 1, $A_i = (a_{i,jk})_{1\leq j,k\leq q} \quad \text{s.t.} \max_{j\leq p} \sum_{k=1}^{p} a_{i,jk}^2 = \mathcal{O}(i^{-2-2\gamma})$. Then by the Rosenthal's theory:

$$\mathbb{E}\left[\left|\sum_{i=1}^{n} X_i\right|^p\right] \leq C_p \left(\sum_{i=1}^{n} \mathbb{E}[|X_i|^p] + \left(\sum_{i=1}^{n} \mathbb{E}[X_i^2]\right)^{p/2}\right)$$

In this setting, for the jth component we have $Z_{ji} = \sum_{m=0}^{\infty} \sum_{k=1}^{p} a_{m,jk} e_{k,i-m}$, and the coupled version differs only in the innovation at time 0:

$$Z_{ji} - Z'_{ji} = \sum_{k=1}^{p} a_{i,jk}(e_{k0} - e'_{k0}).$$

Applying Rosenthal's inequality to the independent summands $X_k = a_{i,jk}(e_{k0} - e'_{k0})$ gives

$$\mathbb{E}\left|\sum_k X_k\right|^w \leq C_w \left(\sum_k |a_{i,jk}|^w + \Big(\sum_k a_{i,jk}^2\Big)^{w/2}\right),$$

and therefore

$$\theta_{i,w,j} = \|Z_{ji} - Z'_{ji}\|_w \leq C_w \Big(\sum_{k=1}^{p} a_{i,jk}^2\Big)^{1/2}.$$

Under the assumption $\max_{j \leq p} \sum_{k=1}^{p} a_{i,jk}^2 = \mathcal{O}(i^{-2-2\gamma})$, we obtain $\theta_{i,w,j} = \mathcal{O}(i^{-1-\gamma})$ and hence

$$\Theta_{mw} = \max_{1 \leq j \leq p} \sum_{\ell=m}^{\infty} \theta_{\ell,w,j} = \mathcal{O}(m^{-\gamma}),$$

A special case of equation 1 is VAR(1) process $\mathbf{z}_i = A\mathbf{z}_{t-1} + \mathbf{e}_i$, where A is a real matrix with spectral norm $\rho(A) < 1$, and the functional dependence measure $\theta_{i,2q,j} = \mathcal{O}(\rho(A)^i)$

Exercise 1

##Data Generation To simulate the stochastic regression model, we will generate data from our code that allows to randomly generate an uppper triangle coefficient matrix. Then we will simulate the data with a VAR(1) model. Then we will proceed to use a Gaussian VAR(1) model \

$$X^t = AX^{t-1} + \epsilon^t, \quad \epsilon^t \overset{\text{i.i.d.}}{\sim} \mathcal{N}(0, \sigma^2 I_{Kp}), \quad \text{and} \quad \text{diag}(A) = 0.2$$

We first define a series of equations that we will use throughout the simulation. To ensure that l2-norm of the coefficient matrix is equal to a target value, we apply the following procedure.

```r
#we define first of all the various functions
##Simulation upper triangular matrix
### To ensure that process is stable, we need to check the abs of eigenvalue
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) {
      return(FALSE)
    }
  }
  return(TRUE)
}


#function to compute companion matrix
comp_mtrx <- function(AA,p ){ #AA is the cbind of covariate matrices
  K<-nrow(AA)
  Kp<-K*p
  C<-matrix(0, nrow=Kp, ncol=Kp)
  C[1:K,1:Kp]<-AA
  C[(K+1):Kp,1:(K*(p-1))]<- diag(1,nrow=K, ncol=K )
  return(C)
}

#
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)
}

## VAR(p) process Simulator
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)
}

#compute the largest eigenvalues of A^TA, used to compute l2 norm

lmda_max<-function(A,k){#k is the number of largest eigenvalues to compute
  s1<-svds(A,k=k, nu=0, nv=0)$d[1]
  return(s1)
}

#we then define a function to scale off-diagonal elements to ensure that
# ||A||=target
scaled_A_matrix<- function(A,target, tol=1e-6, max_k=1000){
  D<-diag(diag(A))
  B<-A-D
  #check the l2nrom of A, that is the max eigenvalue of A^TA
  #we use hte previousy defined function lmbda_max
  f<-function(k){
    M<-D + k*B
    lmda_max(M, k=1)-target
  }
  #we evaluate at k=0
  f0<-f(0)
  if(abs(f0)<tol){
    return(D)
  }
  k_low <- 0
  k_high <- 1
  fk_high <- f(k_high)
  iter <- 0
  while ((fk_high < 0) && (k_high < max_k)) {
    k_high <- k_high * 2
    fk_high <- f(k_high)
    iter <- iter + 1
    if (iter > 200) stop("Could not bracket the root: try increasing max_k or a different initial guess
  }
  if (fk_high < 0) stop("Failed to find k_high that yields operator norm >= target. Increase max_k.")
  root <- uniroot(f, lower = k_low, upper = k_high, tol = tol)$root
  A_scaled <- D + root * B
  return(A_scaled)
}
```

We will now replicate figure 1 from Basu and Michailidis [2015]. He provided the following values for the spectral radius $\rho(A)$ and fixed diagonal elements of the generating coefficient matrix A to generate the VAR(1) process.

α	$\rho(A)$
0.2	0.2
0.2	0.92
0.2	0.96
0.2	1
0.2	1.01
0.2	1.02
0.2	1.03

We will first compute a specific case, in particular for $\langle 0.2, 0.2\rangle$ to then generalise the code for the other cases.

```
K<-200
n_mc <- 200  # Number of Monte Carlo iterations

A_coef<-matrix(0L,nrow=K, ncol=K)
upper_indx<- which(row(A_coef)<col(A_coef))
A_coef[upper_indx]<-rnorm(length(upper_indx), mean=0, sd=0.2)
diag(A_coef)<-0.2
#check th specal norm
print(eigen(t(A_coef)%*%A_coef)$values[1]) #20 is too high, as by the simulation
```

```
## [1] 20.82245
```

```r
#it should be around 0.2
#we then proceed to rescale
prod_coef<-t(A_coef)%*%A_coef
rho_val<-max(abs(eigen(prod_coef)$values))
a<-0.2
scalar<-a/sqrt(rho_val)
scalar
```

```
## [1] 0.04382926
```

```r
#we compute A.scaled
A_scaled<-A_coef*scalar
#check the specrral norm
round(max(abs(eigen(t(A_scaled)%*%A_scaled)$values)),2)== 0.2 # we check whether
```

```
## [1] FALSE
```

```r
#the spectral norm is actually 0.2
#now we then generate the VAR(1) data and compute the response vector
nu<-rep(0,K)
y_0<-rep(0,K)
Cov_epsilon<-diag(0.2, K)#we assume homoskedasticity
pred<-var_sim(A_scaled, nu=nu, Sigma_u=Cov_epsilon, nSteps=200, y0=y_0)

#from a stochastic proces
#generate a sparse Kx1 beta vector to then apply the LASSO regression

beta_star<-rep(0,K)
#selectnaodmly a percentage of the coef to be different from zero
set.seed(1234)
nonzero<-K*0.3
beta_star[sample(1:K,nonzero)]<-rnorm(nonzero, mean=0, sd=0.5)

#generate gaussian inovation
epsilon_t<-rnorm(200, mean=0, sd=1)
#we generate response variables froma  linear model
y_t<- as.numeric(pred %*% beta_star) + epsilon_t
#we then compute the LASSO Regression from glmnet package
lasso_reg<-cv.glmnet(
  x=as.matrix(pred),
  y=y_t,
  type.measure="mse",
  alpha=1,
  nfolds=100,
  family="gaussian",
  grouped=FALSE
)
beta_lasso<-coef(lasso_reg)
#now we compute the lasso error
lasso_error<-sqrt(sum((beta_lasso[-1]-beta_star)^2))
lasso_error
```

```
## [1] 2.475876
```

```
#we compute Monte Carlo simulation to have asymptotic results
nsteps<-100
lasso_errors<-rep(0,nsteps)
A_coef_sim<-matrix(0L,nrow=K, ncol=K)
diag(A_coef_sim)<-0.2
for(i in 1:nsteps){
  upper_indx<- which(row(A_coef_sim)<col(A_coef_sim))
  A_coef_sim[upper_indx]<-rnorm(length(upper_indx), mean=0, sd=0.2)
  #now we normalised the coef matrix
  rho_val<-max(abs(eigen(prod_coef)$values))
  a<-0.2
  scalar<-a/sqrt(rho_val)
  A_scaled_sim<-A_coef_sim*scalar
  #simulate the data
  pred_sim<-var_sim(A_scaled_sim, nu=nu, Sigma_u=Cov_epsilon, nSteps=200, y0=y_0)
  #response variable
  y_t_sim<- as.numeric(pred_sim %*% beta_star) + rnorm(200, mean=0, sd=1)
  #LASSO regression
  lasso_reg_sim<-cv.glmnet(
    x=as.matrix(pred_sim),
    y=y_t_sim,
    type.measure="mse",
    alpha=1,
    nfolds=100,
    family="gaussian",
    grouped=FALSE
  )
  beta_lasso_sim<-coef(lasso_reg_sim)
  #LASSO error
  lasso_errors[i]<-sqrt(sum((beta_lasso_sim[-1]-beta_star)^2))
}
mu_02_02<-mean(lasso_errors)
mu_02_02
```

```
## [1] 2.342756
```

Now that we have define the pipeline, we will able to generalised to the other cases amd for different number of observations

```
K<-200
n_mc<-100
rho_A<-0.2 # fixed, as we have triangular matrix
spctr_norm <- c(0.2, 0.92, 0.96, 1, 1.01, 1.02, 1.03)
pairs<-paste0("(",rho_A,",",spctr_norm,")")
n_values <- c(20, 40, 50, 80, 100, 150, 200, 300, 400, 500, 600) #diff level
#of timepoints
Lasso_error<-matrix(NA, nrow=length(pairs), ncol=length(n_values))
colnames(Lasso_error)<-n_values
rownames(Lasso_error)<-pairs

#fixed initial values to generate VAR(1) data#
```

```r
nu <- rep(0, K)
y_0 <- rep(0, K)
Cov_epsilon <- diag(0.2, K)

#Fixed true betas

beta_star <- rep(0, K)
nonzero <- round(K * 0.4)
beta_star[sample(1:K, nonzero)] <- runif(nonzero,2,3)

#now we start the loop

for (i in seq_along(spctr_norm)) {
  # i is the index; norm_A is the target spectral measure for this row
  norm_A <- spctr_norm[i]

  for (j in seq_along(n_values)) {
    n <- n_values[j]
    lasso_iter <- rep(NA, n_mc)

    # we first generate a diagonal A matrix
    # (we will re-fill the two-upper band inside the MC loop so each MC draw differs)

    for (m in 1:n_mc) {
      # compute the index for the two-upper triangular
      A_coef <- diag(rho_A, K, K)
      two_upper_indx<- which(col(A_coef)== row(A_coef)+2, arr.ind = TRUE)
      # we sample gamma from a normal distribution
      A_coef[ two_upper_indx] <- rnorm(nrow(two_upper_indx), mean = 0, sd = 2)
      #change here to have more var of VAR process

      # we now scale to ensure that spectral measure is equal to what we need
      #we scale by the spectral radius which is common 0.2
      #we need to compute the sqrt of the max eigenvalue for (A^TA)
      prod<-t(A_coef)%*% A_coef #A^TA
      rho_ATA<-max(abs(eigen(prod)$values))
      #rescale

      scalar<- sqrt(norm_A/rho_ATA)
      A_scaled<-A_coef * scalar


      # Step 2: Simulate VAR(1) data with n = n_values[j] time steps
      pred_sim <- var_sim(A_scaled, nu = nu, Sigma_u = Cov_epsilon,
                          nSteps = n,
                          y0 = y_0)

      # Step 3: Generate response
      y_t_sim <- as.numeric(pred_sim %*% beta_star) + rnorm(n, mean = 0,
                                                            sd = 3)
      #we asssume normality of the error term
      # Step 4: Fit LASSO
      lasso_reg_sim <- cv.glmnet(
        x = as.matrix(pred_sim),
        y = y_t_sim,
        type.measure = "deviance",
        intercept=FALSE,
        alpha = 1,
        nfolds = 5,
        family = "gaussian",
        grouped = FALSE
      )
      beta_lasso_sim <- coef(lasso_reg_sim, s = "lambda.min")
      beta_hat <- as.numeric(beta_lasso_sim[-1])
      lasso_iter <- sqrt(sum((beta_hat - beta_star)^2))
    } # end m loop

    # Store the average error over Monte Carlo repetitions
    Lasso_error[i, j] <- mean(lasso_iter, na.rm = TRUE)

  } # end j loop
} # end i loop

Lasso_error
```

```
##                   20       40       50       80      100      150      200
## (0.2,0.2)  27.45106 22.49035 22.77984 22.06649 20.88234 14.93459 12.24623
## (0.2,0.92) 22.49035 22.49035 21.86302 21.10915 21.33288 17.33256 10.25399
## (0.2,0.96) 22.49035 23.70565 22.23601 21.91210 21.37127 15.33626 10.73865
## (0.2,1)    22.49035 22.04774 23.74905 21.20809 21.09741 15.57816 13.63860
## (0.2,1.01) 22.68366 22.49035 22.69119 22.26605 21.32168 17.54797 10.84788
## (0.2,1.02) 24.25694 22.43880 21.52446 21.01031 20.81282 18.80716 10.29030
## (0.2,1.03) 22.49035 23.55366 22.49035 21.67173 20.00189 15.89199 11.58997
##                  300      400      500      600
## (0.2,0.2)  6.595095 5.211143 4.549302 3.921228
## (0.2,0.92) 7.123377 4.801682 3.892079 3.755516
## (0.2,0.96) 6.528694 4.444216 4.272681 3.403214
## (0.2,1)    6.887346 5.088138 4.635230 3.986484
## (0.2,1.01) 6.128968 5.287163 4.354028 3.796825
## (0.2,1.02) 6.571215 5.240484 4.091626 3.855648
## (0.2,1.03) 6.983003 4.710130 4.099023 3.418696
```

now we compute the simulation for all the cases

```
K<-200
n_mc<-100
rho_A<-0.2 # fixed, as we have triangular matrix
spctr_norm <- c(0.2, 0.92, 0.96, 1, 1.01, 1.02, 1.03)
pairs<-paste0("(",rho_A,",",spctr_norm,")")
n_values <- c(20, 40, 50, 80, 100, 150, 200, 300, 400, 500, 600) #diff level
#of timepoints
Lasso_error<-matrix(NA, nrow=length(pairs), ncol=length(n_values))
colnames(Lasso_error)<-n_values
rownames(Lasso_error)<-pairs

#fixed initial values to generate VAR(1) data#
```

```r
nu <- rep(0, K)
y_0 <- rep(0, K)
Cov_epsilon <- diag(0.2, K)

#Fixed true betas

beta_star <- rep(0, K)
nonzero <- round(K * 0.4)
beta_star[sample(1:K, nonzero)] <- runif(nonzero,2,3)

#now we start the loop

for (i in seq_along(spctr_norm)) {
  # i is the index; norm_A is the target spectral measure for this row
  norm_A <- spctr_norm[i]

  for (j in seq_along(n_values)) {
    n <- n_values[j]
    lasso_iter <- rep(NA, n_mc)



    for (m in 1:n_mc) {
      # compute the index for the two-upper triangular
      A_coef <- diag(rho_A, K, K)
      two_upper_indx<- which(col(A_coef)== row(A_coef)+2, arr.ind = TRUE)
      # we sample gamma from a normal distribution
      A_coef[ two_upper_indx] <- rnorm(nrow(two_upper_indx), mean = 0, sd = 2)
      A_scaled <- scaled_A_matrix(A_coef, target = norm_A)

      pred_sim <- var_sim(A_scaled, nu = nu, Sigma_u = Cov_epsilon,
                          nSteps = n,
                          y0 = y_0)

      # Step 3: Generate response
      y_t_sim <- as.numeric(pred_sim %*% beta_star) + rnorm(n, mean = 0,
                                                             sd = 3)
      #we asssume normality of the error term
      # Step 4: Fit LASSO
      lasso_reg_sim <- cv.glmnet(
        x = as.matrix(pred_sim),
        y = y_t_sim,
        type.measure = "deviance",
        intercept=FALSE,
        alpha = 1,
        nfolds = 5,
        family = "gaussian",
        grouped = FALSE
      )
      beta_lasso_sim <- coef(lasso_reg_sim, s = "lambda.1se")
      beta_hat <- as.numeric(beta_lasso_sim[-1])
      lasso_iter <- sqrt(sum((beta_hat - beta_star)^2))
    } # end m loop
    # Store the average error over Monte Carlo repetitions
    Lasso_error[i, j] <- mean(lasso_iter, na.rm = TRUE)

  } # end j loop
} # end i loop

Lasso_error
```

```
##                  20       40       50       80      100      150       200
## (0.2,0.2)  22.27558 22.27558 21.98841 22.27558 19.86004 17.45390 13.738905
## (0.2,0.92) 22.27558 22.27558 22.27558 22.16174 22.27558 17.20078 13.107619
## (0.2,0.96) 22.27558 22.27558 22.07721 21.70955 22.27558 20.39346 12.711138
## (0.2,1)    22.27558 23.08156 22.27558 20.47348 20.04253 18.20859  9.054558
## (0.2,1.01) 22.27558 21.45026 22.27558 22.27558 19.21472 19.90843 11.533542
## (0.2,1.02) 22.27558 22.27558 22.31854 22.12725 21.86363 17.96892  9.254011
## (0.2,1.03) 22.20125 22.27558 22.19614 21.57246 20.03028 19.49354 11.110448
##                 300      400      500      600
## (0.2,0.2)  7.132546 5.620852 4.139953 3.861417
## (0.2,0.92) 6.984664 5.211779 4.518421 3.704195
## (0.2,0.96) 7.992455 4.552820 4.827080 3.909078
## (0.2,1)    7.248344 4.397211 4.778971 4.626483
## (0.2,1.01) 6.778725 6.114303 4.660584 3.509663
## (0.2,1.02) 7.312270 5.015085 4.352393 4.246484
## (0.2,1.03) 7.488470 5.346329 4.232370 3.804003
```

Visualisation

```
# Convert matrix to data frame and tidy it
Lasso_error_df <- as.data.frame(Lasso_error, stringsAsFactors = FALSE)
Lasso_error_df$alpha_rho <- rownames(Lasso_error_df)

Lasso_error_long <- Lasso_error_df %>%
  pivot_longer(
    cols = -c(alpha_rho),  # Use norm_A (not rho) for ||A||
    names_to = "n_obs",
    values_to = "Lasso_Error"
  ) %>%
  mutate(n_obs = as.integer(n_obs))

# Define color and linetype palettes
color_palette <- c(
  "(0.2,0.2)"   = "black",
  "(0.2,0.92)"  = "red",
  "(0.2,0.96)"  = "grey",
  "(0.2,1)"     = "purple",
  "(0.2,1.01)"  = "cyan",
  "(0.2,1.02)"  = "#FA008F",
  "(0.2,1.03)"  = "yellow"
)

line_types <- c(
```

```r
  "(0.2,0.2)"   = "solid",
  "(0.2,0.92)"  = "dashed",
  "(0.2,0.96)"  = "dashed",
  "(0.2,1)"     = "dotdash",
  "(0.2,1.01)"  = "dotdash",
  "(0.2,1.02)"  = "dotdash",
  "(0.2,1.03)"  = "dotdash"
)

# Plot
g <- ggplot(
  data = Lasso_error_long,
  mapping = aes(x = n_obs, y = Lasso_Error)
) +
  geom_line(aes(color=factor(alpha_rho), linetype=factor(alpha_rho)), size=0.5) +
  scale_color_manual(
    values = color_palette

  ) +
  scale_linetype_manual(
    values = line_types,
  ) +
  labs(
    x="n",
    y=TeX("$\\|\\hat{\\beta}- \\beta^*\\|_2$"),
    color = TeX("$\\rho(A),\\,\\|A\\|_2$")
  )+
  expand_limits(x=700)+
  theme_minimal()+
  theme(
    panel.grid.major = element_blank(),
    panel.grid.minor = element_blank(),
    # make sure axes lines are visible
    axis.line = element_line(colour = "black", linewidth = 0.5),
    axis.ticks = element_line(colour = "black"),

    # add full black border around the plot panel
    panel.border = element_rect(colour = "black", fill = NA, linewidth = 0.8),

    # move legend to top-right
    legend.position = c(0.98, 0.98),
    legend.justification = c("right", "top"),

    legend.key.size = unit(0.5, 'cm'),
    legend.key.width = unit(0.5, 'cm'),
    legend.title = element_text(size=8),
    # make legend text a bit smaller
    legend.text = element_text(size = 6)
  )+
  guides(linetype = "none")
```

#Exercise 2

for the second graph we have the following situation:

$$\text{VAR}(2) = X_j^t = 2\alpha X_{j-1}^t - \alpha^2 X_{j-2}^t + \xi^t$$

We assume there is no cross-dependence, and the data is centered. and the number of predictors is 500. To ensure that $\Gamma_X(0) = 1$, we need to compute the covariance matrix for the residuals as follow. We consider the definition of Autocovariance of Stable VAR(p) process from Lütkepohl [2005], in particular the *Yule-Walker equations*:

$$\Gamma_Y(0) = \mathbf{A}\Gamma_Y(0)\mathbf{A}^T + \Sigma_V, \text{ where } A = \begin{bmatrix} A_1 & A_2 \\ I_K & 0 \end{bmatrix}, \quad \Sigma_V = \begin{bmatrix} \Sigma_\epsilon & 0 \\ 0 & 0 \end{bmatrix}$$

In our case, we assume that $\Gamma_X(0) = I_K$, so we can rearrange the previous function as follow:

$$\Sigma_V = I_{Kp} - A\Gamma_Y(0)A^T = I_{Kp} - AA^T$$

Unfortunately, for values closer to α, the covariance matrix wont be positive definitive, but we could rely on the fact that there is no cross-dependence, implying that the autocovariance is determined by the scaled residual variance. We considered the VAR(1) representation of the VAR(2) process:

$$X_t = \alpha_i X_{t-1} + \xi_t$$

$$\begin{aligned}
\xi_t &= X_t - 2\alpha X_{t-1} + \alpha^2 X_{t-2} \\
&= \underbrace{(1 - 2\alpha + \alpha^2 L^2)}_{(1-\alpha L)^2} X_t \qquad\qquad \text{where } L = \text{Lag Operator} \\
X_t &= (1 - \alpha L)^{-2} X_t \\
(1 - \alpha L)^{-2} &= \left[\sum_{j=0}^{\infty} (\alpha L)^j\right]^2 \\
&= \sum_{j=0}^{\infty} (j+1)(\alpha L)^j \\
X_t &= \underbrace{\sum_{j=0}^{\infty} (j+1)(\alpha)^j \xi_{t-j}}_{\text{Wold Representation}} \\
\Gamma_X(0) = \text{Var}(X_t) &= \sigma_\xi \underbrace{\sum_{j=0}^{\infty} (j+1)^2 (\alpha)^j}_{(m^2-2m+1)r^m} \qquad\qquad \sum_{j=0}^{\infty} r^m = \frac{1}{1-r} \\
\sum_{j=0}^{\infty} m r^{m-1} &= \frac{1}{(1-r)^2} \to m r^m = \frac{r}{(1-r)^2} \qquad\qquad \sum_{j=0}^{\infty} m^2 r^m = \frac{r + r^2}{(1-r)^3} \\
\underbrace{\Gamma_X(0)}_{\mathbb{I}} &= \sigma_\xi \frac{(1+\alpha^2)}{(1-\alpha^2)^3} \\
\sigma_\xi &= \frac{(1-\alpha^2)^3}{(1+\alpha^2)} \mathbb{I}
\end{aligned}$$

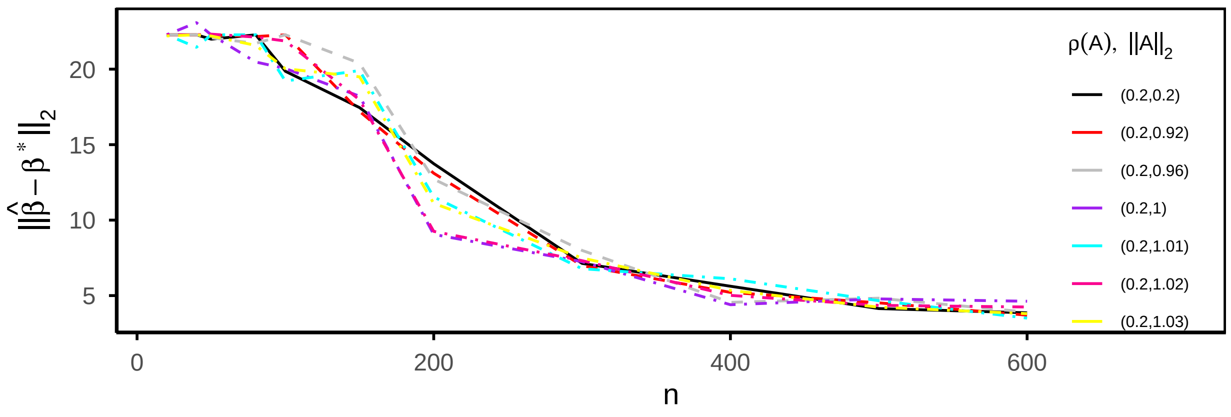
```r
#we start listing all constant and paramters
alpha <- c(0.1, 0.3, 0.5,0.6, 0.7, 0.8, 0.9, 0.95)
n_values <- c(200, 400, 700, 1000, 1300, 1500)
K <- 500
n_mc <- 50

nu <- rep(0, K)
x_0 <- rep(0, K * 2)

# we set sparsity in the true beta for both
nonzero_first_lag  <- round(K * 0.4)
nonzero_second_lag <- round(K * 0.4)
#store lasso error
Lasso_error_ex_2<-matrix(NA,nrow=length(alpha), ncol=length(n_values))

for(a_idx in seq_along(alpha)) {
  alpha_i <- alpha[a_idx]

  S <- (1 + alpha_i)^2 / (1 - alpha_i^2)^3
  sigma_2 <- 1 / S
  Sigma_V <- diag(sigma_2, nrow = K, ncol = K)

  A1 <- diag(2 * alpha_i, K, K)
  A2 <- diag(-(alpha_i^2), K, K)
  AA <- cbind(A1, A2)
  AA_comp <- comp_mtrx(AA, 2)

  if(max(abs(eigen(AA_comp, only.values = TRUE)$values)) >= 1) {
    warning("Alpha ", alpha_i, ": companion matrix spectral radius >= 1 (process may be unstable)")
  }

  for(n_idx in seq_along(n_values)) {
    #check how long it takes for each loop
    start_time <- Sys.time()
    n <- n_values[n_idx]

    errs <- numeric(n_mc)
    nnz  <- integer(n_mc)
    tol_level_vec <- numeric(n_mc)


    #we generate the true parameters for the stochastic regression
    set.seed(2025)
    beta_star <- numeric(2 * K)
    beta_star[sample(1:K, nonzero_first_lag)] <- runif(nonzero_first_lag, 2, 3)
    beta_star[sample((K + 1):(2 * K), nonzero_second_lag)] <- runif(nonzero_second_lag, 2, 3)

    for(mc in 1:n_mc) {
      # simulate X_t for this MC iteration
      X_t <- matrix(0, nrow = n, ncol = 2 * K)
      X_t[1, ] <- x_0
      noise <- mvrnorm(n = n, mu = rep(0, K), Sigma = Sigma_V)
      #VAR(1) process
      for(t in 2:n) {
        X_t[t, ] <- AA_comp %*% X_t[t - 1, ]
        X_t[t, 1:K] <- X_t[t, 1:K] + nu + noise[t, ]
      }

      y_t_sim <- as.numeric(X_t %*% beta_star) + rnorm(n, mean = 0, sd = 3)

      lasso_reg_sim <- cv.glmnet(
        x = as.matrix(X_t),
        y = as.numeric(y_t_sim),
        intercept = FALSE, #we remove the intercept as we have
        #nu=0 in the original model.
        alpha = 1, #this ensure we applied a LASSO regerssion
        nfolds = 8,
        grouped = FALSE
      )

      coef_hat <- as.numeric(coef(lasso_reg_sim, s = "lambda.min")[-1])
      errs[mc] <- sqrt(sum((coef_hat - beta_star)^2))


    } # end MC loop
    end_time <- Sys.time()
    elapsed <- round(as.numeric(difftime(end_time, start_time, units = "mins")), 2)
    valid_errs <- errs[!is.na(errs)]
    Lasso_error_ex_2[ a_idx,n_idx]<-mean(valid_errs, na.rm = TRUE)
    cat("alpha=", alpha_i, " n=", n, " MC valid=", length(valid_errs),
        " median_err=", round(median(valid_errs, na.rm = TRUE), 4),
        " mean_err=", round(mean(valid_errs, na.rm = TRUE), 4),
        " sd=", round(sd(valid_errs, na.rm = TRUE), 4), "\n")

    cat("  quantiles (0.1,0.25,0.5,0.75,0.9): ",
        paste(round(quantile(valid_errs, probs = c(.1, .25, .5, .75, .9),
                             na.rm = TRUE), 3), collapse = ", "), "\n")

    cat("Execution time for each (alpha,n) combination", elapsed)


  } # end n_values loop

} # end alpha loop
```

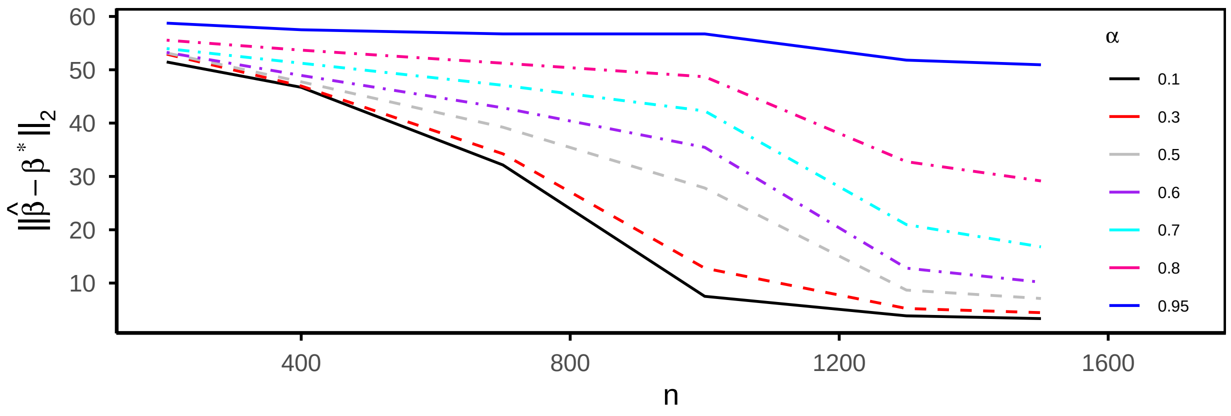
```
## alpha= 0.1  n= 200  MC valid= 50  median_err= 50.8446  mean_err= 51.4644  sd= 1.3083
##   quantiles (0.1,0.25,0.5,0.75,0.9):  50.193, 50.479, 50.845, 52.321, 53.349
## Execution time for each (alpha,n) combination 0.48alpha= 0.1  n= 400  MC valid= 50  median_err= 46.6
##   quantiles (0.1,0.25,0.5,0.75,0.9):  45.627, 46.038, 46.65, 47.287, 47.751
## Execution time for each (alpha,n) combination 0.85alpha= 0.1  n= 700  MC valid= 50  median_err= 32.0
##   quantiles (0.1,0.25,0.5,0.75,0.9):  29.99, 31.04, 32.068, 33.466, 34.275
## Execution time for each (alpha,n) combination 1.55alpha= 0.1  n= 1000  MC valid= 50  median_err= 7.5
##   quantiles (0.1,0.25,0.5,0.75,0.9):  6.702, 7.099, 7.528, 7.826, 8.384
## Execution time for each (alpha,n) combination 2.49alpha= 0.1  n= 1300  MC valid= 50  median_err= 3.8
##   quantiles (0.1,0.25,0.5,0.75,0.9):  3.629, 3.709, 3.86, 3.974, 4.072
## Execution time for each (alpha,n) combination 3.09alpha= 0.1  n= 1500  MC valid= 50  median_err= 3.3
##   quantiles (0.1,0.25,0.5,0.75,0.9):  3.17, 3.239, 3.326, 3.45, 3.543
## Execution time for each (alpha,n) combination 3.37alpha= 0.3  n= 200  MC valid= 50  median_err= 52.94
##   quantiles (0.1,0.25,0.5,0.75,0.9):  51.603, 52.26, 52.941, 53.502, 54.25
## Execution time for each (alpha,n) combination 0.56alpha= 0.3  n= 400  MC valid= 50  median_err= 46.99
##   quantiles (0.1,0.25,0.5,0.75,0.9):  45.689, 46.288, 46.995, 47.48, 48.169
## Execution time for each (alpha,n) combination 0.95alpha= 0.3  n= 700  MC valid= 50  median_err= 34.24
##   quantiles (0.1,0.25,0.5,0.75,0.9):  32.273, 33.175, 34.246, 35.459, 36.13
## Execution time for each (alpha,n) combination 1.73alpha= 0.3  n= 1000  MC valid= 50  median_err= 12.0
##   quantiles (0.1,0.25,0.5,0.75,0.9):  11.136, 11.811, 12.621, 13.619, 14.332
## Execution time for each (alpha,n) combination 2.61alpha= 0.3  n= 1300  MC valid= 50  median_err= 5.2
##   quantiles (0.1,0.25,0.5,0.75,0.9):  4.958, 5.066, 5.21, 5.449, 5.511
## Execution time for each (alpha,n) combination 3.06alpha= 0.3  n= 1500  MC valid= 50  median_err= 4.4
##   quantiles (0.1,0.25,0.5,0.75,0.9):  4.179, 4.299, 4.469, 4.586, 4.748
## Execution time for each (alpha,n) combination 3.95alpha= 0.5  n= 200  MC valid= 50  median_err= 53.1
##   quantiles (0.1,0.25,0.5,0.75,0.9):  51.965, 52.457, 53.137, 53.671, 54.528
## Execution time for each (alpha,n) combination 0.56alpha= 0.5  n= 400  MC valid= 50  median_err= 47.7
##   quantiles (0.1,0.25,0.5,0.75,0.9):  46.549, 47.039, 47.729, 48.285, 48.781
## Execution time for each (alpha,n) combination 1.01alpha= 0.5  n= 700  MC valid= 50  median_err= 39.2
##   quantiles (0.1,0.25,0.5,0.75,0.9):  37.741, 38.306, 39.28, 39.948, 40.582
## Execution time for each (alpha,n) combination 1.67alpha= 0.5  n= 1000  MC valid= 50  median_err= 27.8
##   quantiles (0.1,0.25,0.5,0.75,0.9):  25.638, 27.045, 27.867, 28.791, 29.71
## Execution time for each (alpha,n) combination 2.58alpha= 0.5  n= 1300  MC valid= 50  median_err= 8.6
##   quantiles (0.1,0.25,0.5,0.75,0.9):  8.243, 8.393, 8.693, 9.005, 9.14
## Execution time for each (alpha,n) combination 3.6alpha= 0.5  n= 1500  MC valid= 50  median_err= 7.08
##   quantiles (0.1,0.25,0.5,0.75,0.9):  6.734, 6.893, 7.08, 7.35, 7.496
## Execution time for each (alpha,n) combination 4.55alpha= 0.6  n= 200  MC valid= 50  median_err= 53.1
##   quantiles (0.1,0.25,0.5,0.75,0.9):  51.897, 52.474, 53.191, 53.993, 54.76
## Execution time for each (alpha,n) combination 0.62alpha= 0.6  n= 400  MC valid= 50  median_err= 48.7
##   quantiles (0.1,0.25,0.5,0.75,0.9):  47.838, 48.538, 48.78, 49.315, 50.123
## Execution time for each (alpha,n) combination 1.01alpha= 0.6  n= 700  MC valid= 50  median_err= 42.6
##   quantiles (0.1,0.25,0.5,0.75,0.9):  41.703, 42.044, 42.676, 43.613, 44.235
## Execution time for each (alpha,n) combination 1.83alpha= 0.6  n= 1000  MC valid= 50  median_err= 35.
##   quantiles (0.1,0.25,0.5,0.75,0.9):  34.071, 34.674, 35.378, 36.357, 36.924
## Execution time for each (alpha,n) combination 2.09alpha= 0.6  n= 1300  MC valid= 50  median_err= 12.8
##   quantiles (0.1,0.25,0.5,0.75,0.9):  12.139, 12.386, 12.82, 13.281, 13.48
## Execution time for each (alpha,n) combination 3.13alpha= 0.6  n= 1500  MC valid= 50  median_err= 10.
##   quantiles (0.1,0.25,0.5,0.75,0.9):  9.588, 9.819, 10.179, 10.483, 10.728
## Execution time for each (alpha,n) combination 3.64alpha= 0.7  n= 200  MC valid= 50  median_err= 53.8
##   quantiles (0.1,0.25,0.5,0.75,0.9):  52.684, 53.118, 53.875, 54.65, 55.531
## Execution time for each (alpha,n) combination 0.44alpha= 0.7  n= 400  MC valid= 50  median_err= 50.9
##   quantiles (0.1,0.25,0.5,0.75,0.9):  50.216, 50.739, 50.999, 51.547, 52.31
## Execution time for each (alpha,n) combination 0.77alpha= 0.7  n= 700  MC valid= 50  median_err= 46.9
##   quantiles (0.1,0.25,0.5,0.75,0.9):  46.017, 46.498, 46.959, 47.674, 48.265
## Execution time for each (alpha,n) combination 1.32alpha= 0.7  n= 1000  MC valid= 50  median_err= 42.0
##   quantiles (0.1,0.25,0.5,0.75,0.9):  41.258, 41.715, 42.064, 42.795, 43.455
## Execution time for each (alpha,n) combination 2.01alpha= 0.7  n= 1300  MC valid= 50  median_err= 21.0
##   quantiles (0.1,0.25,0.5,0.75,0.9):  19.794, 20.33, 21.07, 21.596, 22.089
## Execution time for each (alpha,n) combination 3.72alpha= 0.7  n= 1500  MC valid= 50  median_err= 16.9
##   quantiles (0.1,0.25,0.5,0.75,0.9):  15.642, 16.144, 16.997, 17.444, 17.659
## Execution time for each (alpha,n) combination 4.03alpha= 0.8  n= 200  MC valid= 50  median_err= 55.5
##   quantiles (0.1,0.25,0.5,0.75,0.9):  53.548, 54.375, 55.552, 56.499, 57.312
## Execution time for each (alpha,n) combination 0.4alpha= 0.8  n= 400  MC valid= 50  median_err= 53.57
##   quantiles (0.1,0.25,0.5,0.75,0.9):  52.594, 52.951, 53.574, 54.189, 54.814
## Execution time for each (alpha,n) combination 0.68alpha= 0.8  n= 700  MC valid= 50  median_err= 51.3
##   quantiles (0.1,0.25,0.5,0.75,0.9):  50.196, 50.696, 51.316, 51.748, 52.355
## Execution time for each (alpha,n) combination 1.15alpha= 0.8  n= 1000  MC valid= 50  median_err= 48.
##   quantiles (0.1,0.25,0.5,0.75,0.9):  47.451, 47.9, 48.621, 49.5, 50.028
## Execution time for each (alpha,n) combination 1.79alpha= 0.8  n= 1300  MC valid= 50  median_err= 32.
##   quantiles (0.1,0.25,0.5,0.75,0.9):  31.467, 32.121, 32.737, 33.374, 34.091
## Execution time for each (alpha,n) combination 3.29alpha= 0.8  n= 1500  MC valid= 50  median_err= 29.
##   quantiles (0.1,0.25,0.5,0.75,0.9):  27.704, 28.295, 29.296, 29.795, 30.768
## Execution time for each (alpha,n) combination 72.58alpha= 0.9  n= 200  MC valid= 50  median_err= 56.
##   quantiles (0.1,0.25,0.5,0.75,0.9):  54.226, 55.125, 56.917, 58.76, 60.925
## Execution time for each (alpha,n) combination 0.39alpha= 0.9  n= 400  MC valid= 50  median_err= 55.8
##   quantiles (0.1,0.25,0.5,0.75,0.9):  54.486, 55.002, 55.883, 56.752, 57.854
## Execution time for each (alpha,n) combination 0.64alpha= 0.9  n= 700  MC valid= 50  median_err= 54.6
##   quantiles (0.1,0.25,0.5,0.75,0.9):  53.535, 53.975, 54.635, 55.826, 56.551
## Execution time for each (alpha,n) combination 1.07alpha= 0.9  n= 1000  MC valid= 50  median_err= 54.
##   quantiles (0.1,0.25,0.5,0.75,0.9):  52.909, 53.249, 54.167, 54.99, 56.011
## Execution time for each (alpha,n) combination 1.62alpha= 0.9  n= 1300  MC valid= 50  median_err= 45.
##   quantiles (0.1,0.25,0.5,0.75,0.9):  44.727, 45.065, 45.907, 46.85, 47.435
## Execution time for each (alpha,n) combination 3.15alpha= 0.9  n= 1500  MC valid= 50  median_err= 44.
##   quantiles (0.1,0.25,0.5,0.75,0.9):  42.914, 43.435, 44.231, 45.019, 45.687
## Execution time for each (alpha,n) combination 3.72alpha= 0.95  n= 200  MC valid= 50  median_err= 58.
##   quantiles (0.1,0.25,0.5,0.75,0.9):  54.537, 56.078, 58.23, 60.443, 63.534
## Execution time for each (alpha,n) combination 0.37alpha= 0.95  n= 400  MC valid= 50  median_err= 57.
##   quantiles (0.1,0.25,0.5,0.75,0.9):  55.189, 55.797, 57.088, 58.298, 60.041
## Execution time for each (alpha,n) combination 0.61alpha= 0.95  n= 700  MC valid= 50  median_err= 56.
##   quantiles (0.1,0.25,0.5,0.75,0.9):  54.456, 55.725, 56.567, 57.556, 59.476
## Execution time for each (alpha,n) combination 0.98alpha= 0.95  n= 1000  MC valid= 50  median_err= 56
##   quantiles (0.1,0.25,0.5,0.75,0.9):  54.353, 55.735, 56.732, 57.625, 58.969
## Execution time for each (alpha,n) combination 1.46alpha= 0.95  n= 1300  MC valid= 50  median_err= 51
##   quantiles (0.1,0.25,0.5,0.75,0.9):  50.47, 51.007, 51.812, 52.645, 53.771
## Execution time for each (alpha,n) combination 2.58alpha= 0.95  n= 1500  MC valid= 50  median_err= 50
##   quantiles (0.1,0.25,0.5,0.75,0.9):  49.584, 50.213, 50.697, 51.658, 52.28
## Execution time for each (alpha,n) combination 2.99
```

```r
#Convert names from the graph
colnames(Lasso_error_ex_2)<-as.character(n_values)
rownames(Lasso_error_ex_2)<-as.character(alpha)
# Convert matrix to data frame and tidy it
Lasso_error_df2 <- as.data.frame(Lasso_error_ex_2, stringsAsFactors = FALSE)
Lasso_error_df2$alpha <- rownames(Lasso_error_df2)

Lasso_error_long <- Lasso_error_df2 %>%
  pivot_longer(
    cols = -c(alpha),
    names_to = "n_obs",
    values_to = "Lasso_Error"
  ) %>%
  mutate(n_obs = as.integer(n_obs))

# Define color and linetype palettes
color_palette <- c(
  "0.1"   = "black",
  "0.3"  = "red",
  "0.5"  = "grey",
  "0.6"     = "purple",
  "0.7"  = "cyan",
  "0.8"  = "#FA008F",
  "0.90"  = "yellow",
  "0.95"  = "blue"
)

line_types <- c(
  "0.1"   = "solid",
  "0.3"  = "dashed",
  "0.5"  = "dashed",
  "0.6"     = "dotdash",
  "0.7"  = "dotdash",
  "0.8"  = "dotdash",
  "0.90"  = "dotdash",
  "0.95"  ="solid"
)

# Plot
p <- ggplot(
  data = Lasso_error_long,
  mapping = aes(x = n_obs, y = Lasso_Error)
) +
  geom_line(aes(color=factor(alpha), linetype=factor(alpha)), size=0.5) +
  scale_color_manual(
    values = color_palette

  ) +
  scale_linetype_manual(
    values = line_types,
  ) +
  scale_x_continuous(breaks=seq(0,1600, by=400))+
  scale_y_continuous(breaks = seq(0,60, by=10))+
  expand_limits(x=1700)+
  labs(
    x="n",
    y=TeX("$\\|\\hat{\\beta}- \\beta^*\\|_2$"),
    color = TeX("$\\alpha$")
  )+
  theme_minimal()+
  theme(
    panel.grid.major = element_blank(),
    panel.grid.minor = element_blank(),
    # make sure axes lines are visible
    axis.line = element_line(colour = "black", linewidth = 0.5),
    axis.ticks = element_line(colour = "black"),

    # add full black border around the plot panel
    panel.border = element_rect(colour = "black", fill = NA, linewidth = 0.8),

    # move legend to top-right
    legend.position = c(0.98, 0.98),
    legend.justification = c("right", "top"),
    legend.key.size = unit(0.5, 'cm'),
    legend.key.width = unit(0.5, 'cm'),
    legend.title = element_text(size=8),

    legend.text = element_text(size = 6),


  )+
  guides(linetype = "none")
```

```
#combine graphs
gp<-grid.arrange(g,p,nrow=2)
```

```
## Warning: Removed 6 rows containing missing values or values outside the scale range
## (‘geom_line()‘).
```





```
gp
```

```
## TableGrob (2 x 1) "arrange": 2 grobs
##   z     cells    name           grob
## 1 1 (1-1,1-1) arrange gtable[layout]
## 2 2 (2-2,1-1) arrange gtable[layout]
```

On the top figure, there exist a cross-sectional dependence defined by the VAR coefficient vectors. We saw at the beginning of the report that the functional dependence scale is of the same order as $\rho(A)$, implying that $\theta \leq c \cdot \rho(A)$ with sufficiently large n. In the picture however we see that even though we fixed $\rho(A)$, by increasing l2 norm of A the estimation error decreases more slowly with increasing sample size n. This indicates that the estimation error is sensitive to the magnitude of the coefficients in the matrix A. Potentially A can slightly exceed 1 as well.

On the second example we have a situation that in some cases the assumption $\|A\| < 1$ might not hold. Assmuing that there is no cross-dependence, the graph shows different convergence speeds depending on α, even though all processes are stable.We notice that for certain level of alpha, we have that with large sample size, the effect of dependence is significantly reduced.

References

Sumanta Basu and George Michailidis. Regularized estimation in sparse high-dimensional time series models. *The Annals of Statistics*, pages 1535–1567, 2015.

Xiaohui Chen, Mengyu Xu, and Wei Biao Wu. Covariance and precision matrix estimation for high-dimensional time series. *The Annals of Statistics*, 41(6):2994–3021, 2013.

Helmut Lütkepohl. *New Introduction to Multiple Time Series Analysis.* Number 978-3-540-27752-1 in Springer Books. Springer, March 2005. ISBN ARRAY(0x677fb3a0). doi: 10.1007/978-3-540-27752-1. URL https://ideas.repec.org/b/spr/sprbok/978-3-540-27752-1.html.