Higgins Method for AR(1) factor PD

0. Setup

Needed packages: (Maybe not all are needed)

```
#install.packages("tidyverse")
library("tidyverse")
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.2 --
## v ggplot2 3.4.0      v purrr   0.3.5
## v tibble  3.1.8      v dplyr   1.0.10
## v tidyr   1.2.1      v stringr 1.4.1
## v readr   2.1.3      v forcats 0.5.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
#install.packages("PEIP")
library("PEIP")
#install.packages("foreach")
library("foreach")
```

```
##
## Attaching package: 'foreach'
##
## The following objects are masked from 'package:purrr':
##
##     accumulate, when
```

```
#install.packages("doMC")
library("doMC")
```

```
## Loading required package: iterators
## Loading required package: parallel
```

```
#install.packages("expm")
library("expm")
```

```
## Loading required package: Matrix
##
## Attaching package: 'Matrix'
##
## The following objects are masked from 'package:tidyr':
##
##     expand, pack, unpack
##
##
## Attaching package: 'expm'
##
```

```
## The following object is masked from 'package:Matrix':
##
##     expm
```

```
#install.packages("dplyr")
library("dplyr")
```

This first command is to paralelise some computations. You must indicate the number of cores of your computer.

```
registerDoMC(2) # change the "2" by the n of cores of your computer.
set.seed(33204)
```

Preliminary Functions

The following function sends a message to my phone when the computation is finished (depending on the specification, it may take a lot of time). If you are running this on a mac, change my email for your phone number or email and enable this option in 'higgins_simul' to use this feature.

```
send_text = function(message){
    system(paste('osascript -e \'tell application "Messages"\' -e \'send "', message, '" to buddy "iren
}
```

This function is a 'groupby' kind of function, using tidyverse.

```
split_tibble <- function(tibble, col = 'col') tibble %>% split(., .[, col])
```

1. Simulating DGP

This function simply simulates the data according to the DGP processes described on the paper. It is inherited from a past notebook, so some features may not be used.

```
data_ary = function(beta=0, DGP = 1, N=50, T=50, rho=0.8){

  if (DGP == 3){lambda = 0.4} else{lambda = 1}
  if (DGP == 1 | DGP == 6) {Lambda = 0.4} else  if(DGP == 2 | DGP == 7){Lambda = 4} else{Lambda = 1}
  if(DGP == 8){sigmaeps = 10} else if (DGP == 6 | DGP == 7) {sigmaeps = 0.1} else{sigmaeps = 1}
  if(DGP == 5){sigmaeta = 10} else if (DGP == 4) {sigmaeta = 0.1} else{sigmaeta = 1}

  lambda1 = vector(); lambda2 = vector(); Lambda1 = vector(); Lambda2 = vector(); F1 = rnorm(T); F2 = rn
  eta = matrix(rnorm(N*T,0,sqrt(sigmaeta)),nrow = N); eps = matrix(rnorm(N*T,0,sqrt(sigmaeps)),nrow = N)

  ## Give value
  for (i in 1:N){
    lambda1[i] = lambda + 0.5 - 1*(i > floor(0.5*N)) # 1*N vector
    lambda2[i] = 0.5 - 1*(i > floor(0.5*N))          # 1*N vector
    Lambda1[i] = 0.5 - 1*(i > floor(0.5*N)) # 1*N vector
    Lambda2[i] = Lambda + 0.5 - 1*(i > floor(0.5*N))          # 1*N vector
  }


  X = cbind(Lambda1,Lambda2) %*% rbind(F1,F2) + eta # N*T matrix
  Y = matrix(0, N, T)
  Y[,1] = X[,1]*beta + cbind(lambda1,lambda2) %*% rbind(F1,F2)[,1] + eps[,1]
```

```
  for (t in 2:T){
    Y[,t] = rho*Y[,t-1] + X[,t]*beta + cbind(lambda1,lambda2) %*% rbind(F1,F2)[,t] + eps[,t]
  }

  y = c(t(Y)) # vector of length N*T
  x = c(t(X)) # vector of length N*T

  year = rep(1:T,N)
  region = rep(1:N,each = T)
  df = data.frame(cbind(year,region,y,x))
  return(df)
}
```

2. Implementing Higgins

2.1. Preliminary functions

First, we need to compute a function to create the W transformation matrix from the paper:

```
W = function(d){
  W=matrix(0,d,d)
  for(i in 2:d){
    W[i-1,i] = 1
  }
  return(W)
}
```

And then another to carry out the whole transformation from X to $\mathcal{X}$. Some of these functions look ugly but are generally simple operations within and between lists of matrices. This one obtains $\mathcal{Y}, \mathcal{X}$ matrices from the original dataframe.

```
transf_matrices = function(data,r=2){

  # obtainiing calX
  datalist = split_tibble(data, 'region')
  temp = lapply(datalist, function(x) t(matrix(x[,"x"])))
  calX = do.call("rbind",temp)
  temp = lapply(datalist, function(x) t(matrix(x[,"y"])))
  Y = do.call("rbind",temp)

  rhs=solve(sqrtm((t(calX)%*%(calX))))
  Qx = calX%*%rhs

  Ytrans = t(Qx)%*%Y
  Xtrans = t(Qx)%*%calX

  d = dim(Ytrans)[1]
  Ylag = Ytrans%*%W(d)
  X = cbind(Ylag,Xtrans)

  return(list(Y = Ytrans, Ylag = Ylag, Xtrans = Xtrans))
}
```

This one obtains the residuals.

```
res_hig = function(vec, Y1, Ylag, X){
  return(sum(diag((t((Y1 - vec[1]*Ylag - vec[2]*X))%*%(Y1 - vec[1]*Ylag - vec[2]*X)))))
}
```

Finally, we create the function to optimise numerically to obtain the estimated parameters. This function is critical and the numerical optimisation inside it may not be perfectly constructed. Some further work on it may be needed concerning this.

```
higgins_num_iteration = function(data, r=2, init_fact, init_F, Niter=3){
  Y = transf_matrices(data, r)$Y
  N = length(unique(data$region))
  T = dim(Y)[1]
  if (missing(init_fact)){
    init_fact = matrix(rnorm(T*r),T,r)
  }
  if (missing(init_F)){
    init_F =  matrix(rnorm(T*r),T,r)
  }

  # ITERATION:
  datatrans = transf_matrices(data)
  for (w in 1:Niter){
    Y1 = Y - init_fact%*%t(init_F)
    opt = optim(c(0.8,0), res_hig, Y1 = Y1, Ylag = datatrans$Ylag, X=datatrans$X)$par #alpha and beta
    residuals = Y - opt[1]*datatrans$Ylag - opt[2]*datatrans$Xtrans
    tr_residuals = (1/(N*T))*residuals%*%t(residuals)
    res = USV(tr_residuals)
    Lambda_hat = sqrt(N)*res$U[,1:r]
    F_hat = t((diag(res$S)%*%t(res$V))[1:r,]*(1/sqrt(N)))
    init_fact = Lambda_hat
    init_F = F_hat
  }

  # MSE  part

  beta= opt[2]

  # original data
  ## add t-1
  y_1 = c(NA, data$y[1:length(data$y)-1])
  data = cbind(data,y_1)
  data[data["year"]==1,]["y_1"] = NA
  data = na.omit(data)
  dflist <- split_tibble(data, 'region')
  dflist <- lapply(dflist, function(x) x[c("y","y_1","x")])
  # tranforms for estimating factors and loadings
  matz = as.matrix(do.call("cbind", dflist))
  matz = matz%*%t(matz)
  F_est = sqrt(dim(unique(data["year"]))[1])*eigen(matz)$vectors[,1:r]
  t = dim(unique(data["year"]))[1]
  rft = solve(t(F_est)%*%F_est)%*%t(F_est)
  M = diag(t) - F_est%*%rft
  reslist <- lapply(dflist, function(x) (x["y"]-F_est%*%rft%*%as.matrix(x["y"]-c(opt[1])*x["y_1"]-c(opt
```

```
  mse = do.call(sum,reslist) / (length(dflist)*t)

  return(list(opt=opt, mse=mse))
}
```

3. Simulation

3.1. Simulation Function

This function runs M simulations for specific DGP, N, T and parameters and returns the mean MSE (of the regression!) and mean biases of the parameters.

```
simulation_X_e = function(M, beta, DGP, N, T, rho=0.8, r=2){
  beta_row2 = numeric(M)
  mse_row = numeric(M)
  alpha_row = numeric(M)

  results = foreach(i=1:M) %dopar%{
    data=data_ary(beta=beta, DGP=DGP, N=N, T=T, rho=rho)
    res_prev = higgins_num_iteration(data,r)
    res = res_prev$opt
    mse = res_prev$mse
    beta_row2 = res[2] - beta
    alpha_row = res[1] - rho
    mse_row =
    return(list(alpha_row,beta_row2,mse))
  }

  beta_row2 = sapply(results, `[[`, 2)
  alpha_row = sapply(results, `[[`, 1)
  mse = sapply(results, `[[`, 3)
  return(list(alpha_bias = mean(alpha_row), beta_bias=mean(beta_row2), mse = mean(mse)))
}
```

This function just wraps everything constructing, for a given T, three dataframes with mean MSE (of the regression) and biases of the parameters and sends some messages to my phone. (That part is disabled behind a #, enable is desired.). Each row represents each 1,...,8 DGP and each column N=50,200.

```
higgins_simul = function(beta=0, rho=0.8,r=2, Niter=3000, T=4){ # T = 4 or T=10
  alphabias = data.frame(matrix(0,8,2)) # dim should be changed if needed
  betabias = data.frame(matrix(0,8,2))
  msemat =  data.frame(matrix(0,8,2))
  colnames(alphabias)=c(50,200)
  colnames(betabias)=c(50,200)
  colnames(msemat)=c(50,200) # N = 50 and 200
  rownames(alphabias)=1:8
  rownames(betabias)=1:8
  rownames(msemat)=1:8
  for(i in c("50","200")){
    # send_text(paste("INIT LOOP N=",i, " T=",T, " BETA=",beta," (type=higgins)", sep=""))
    for (j in 1:8){
      m = simulation_X_e(M=Niter, beta=beta, DGP=j, N=as.integer(i), T=T, rho=rho, r=r)
      alphabias[j, i] <- m$alpha_bias
```

```
      betabias[j, i] <- m$beta_bias
      msemat[j, i] <- m$mse
      mes = paste("Iteration ", j, " of 8. Value: \n", "--- Alpha bias =", m[1], " ---Beta bias =", m[2]
      # send_text(mes)
    }
  }
  # send_text(paste("ENDED CORRECTLY","____________________",sep="\n"))
  return(list(alpha_b = alphabias, beta_b= betabias, mse = msemat))
}
```

3.2. Simulation Results

Results for T=4; N=50,200; rho=0.8. It takes around 1h10 in my computer

```
#higgins_simul(beta=0,r=2)
```

Results for T=8; N=50,200; rho=0.8. It takes around 1h10 in my computer

```
#higgins_simul(beta=0,r=2, T=8)
```

Results for T=4; N=50,200; rho=0. It takes around 1h10 in my computer. (For some unknown reason, the results did not plot here, but I received them in my mobile phone.)

```
#higgins_simul(beta=0,r=2, T=4, rho=0)
```

Results for T=8; N=50,200; rho=0. It takes around 1h10 in my computer

```
#higgins_simul(beta=0,r=2, T=8, rho=0)
```