quarto

Jane Doe

12/25/22

Table of contents

Preface

This is a Quarto book.

To learn more about Quarto books visit https://quarto.org/docs/books.

```
1 + 1
```

```
[1] 2
```

```
2 + 2
```

```
[1] 4
```

1 Laczo (2015) R scripts for data preparation

1.1 Load packages

```r
pacman::p_load(
  tidyverse,
  Rtauchen
)
```

1.2 Load datasets

```r
load('Laczo2015/allest')
```

```r
numVillages <- max(villagedat)
villageIndicatorMatrix <- do.call(
  cbind,
  map(
    seq(1, numVillages),
    ~ villagedat[, 1] == .
  )
)
```

```r
createVillageAggregateByYear <- function(
    data,
    func,
    .numVillages = numVillages,
    .villageIndicatorMatrix = villageIndicatorMatrix
    ) {
  do.call(
    rbind,
    map(
      seq(1, .numVillages),
```

```r
      ~ data[.villageIndicatorMatrix[, .],] %>% func
      )
    )
  }

createVillageAggregate <- function(
    data,
    func,
    .numVillages = numVillages,
    .villageIndicatorMatrix = villageIndicatorMatrix
    ) {
    map_vec(
      seq(1, .numVillages),
      ~ data[.villageIndicatorMatrix[, .],] %>% func
    )
  }
```

```r
vilMeanIncByYear <- createVillageAggregateByYear(incdat, colMeans)
vilSumIncByYear <- createVillageAggregateByYear(incdat, colSums)
vilMeanInc <- createVillageAggregate(incdat, mean)
vilMeanConsByYear <- createVillageAggregateByYear(consdat, colMeans)
vilSumConsByYear <- createVillageAggregateByYear(consdat, colSums)
vilMeanCons <- createVillageAggregate(consdat, mean)
```

```r
vilMeanEduc <- createVillageAggregate(educdat, mean)
vilMeanWomenShare <- createVillageAggregate(propfdat, mean)
vilMeanAge <- createVillageAggregate(educdat, mean)
vilMeanLand <- createVillageAggregate(educdat, mean)
```

```r
incLow <- createVillageAggregate(incdat, function(x) quantile(x, 0.025, na.rm = TRUE))
incHigh <- createVillageAggregate(incdat, function(x) quantile(x, 0.975, na.rm = TRUE))
consLow <- createVillageAggregate(consdat, function(x) quantile(x, 0.025, na.rm = TRUE))
consHigh <- createVillageAggregate(consdat, function(x) quantile(x, 0.975, na.rm = TRUE))
```

```r
vilConsPerIncByYear <- vilMeanConsByYear / vilMeanIncByYear
incdatRescaled <- incdat * (villageIndicatorMatrix %*% vilConsPerIncByYear)
vilMeanIncByYearRescaled <- createVillageAggregateByYear(incdatRescaled, colMeans)
```

1.3 Estimate income processes

```
numIncomeStatesHH <- 8
numIncomeStatesVillage <- 5
numIncomeStatesCombination <- numIncomeStatesVillage * numIncomeStatesHH
```

1.3.1 Household

```
householdIncMean <- incdatRescaled %>% rowMeans
householdIncSD <- apply(incdatRescaled, 1, sd, na.rm = TRUE)
householdIncCV <- householdIncSD / householdIncMean

vilIncMeanMedian <- createVillageAggregate(
  as.matrix(householdIncMean),
  median
  )
vilIncCVMedian <- createVillageAggregate(
  as.matrix(householdIncCV),
  median
  )
```

1.3.1.1 Estimate AR(1) process approximated by a Markov chain

```
householdIncMeanClassVec <- (householdIncMean > (villageIndicatorMatrix %*% vilIncMeanMedi
householdIncCVClassVec <- (householdIncCV > (villageIndicatorMatrix %*% vilIncCVMedian)) +

laggedIncdatRescaled <- cbind(NA, incdatRescaled[, seq(1, tnum - 1)])
incdatRescaled[
  (incdat <= as.vector(villageIndicatorMatrix %*% incLow)) |
    (incdat >= as.vector(villageIndicatorMatrix %*% incHigh))] <- NA
laggedIncdatRescaled[
  (incdat <= as.vector(villageIndicatorMatrix %*% incLow)) |
    (incdat >= as.vector(villageIndicatorMatrix %*% incHigh))] <- NA

getDataByMeanCVClassByVillage <- function(
    village,
    data,
    meanClass,
```

```r
    CVClass,
    meanClassVec = householdIncMeanClassVec,
    CVClassVec = householdIncCVClassVec,
    .villageIndicatorMatrix = villageIndicatorMatrix
    ) {
  data[
    (meanClassVec == meanClass) &
      (CVClassVec == CVClass) &
      (.villageIndicatorMatrix[, village])
  ]
}

calculateAR1Parameters <- function(data, laggedData) {
  mu <- mean(data, na.rm = TRUE)
  rho <- cor(
    data,
    laggedData,
    use="complete.obs"
    )
  sigmau <- sqrt(var(data, na.rm = TRUE) * (1 - rho^2))

  return(list(mu = mu, rho = rho, sigmau = sigmau))
}
```

```r
calculateGridPoints <- function(numStates, data) {
  gridQuantile <- seq(0, 1, by = 1 / numStates)
  map_dbl(
    (gridQuantile[1:(length(gridQuantile) - 1)] + gridQuantile[2:length(gridQuantile)]) /
    ~ quantile(data, ., na.rm = TRUE)
  )
}

approximateAR1Tauchen <- function(numStates, data, mu, rho, sigma) {

  gridPoints <- calculateGridPoints(numStates, data)

  #transition probabilities
  transitionMatrix <- array(NA, c(numStates, numStates))
  for (currentState in 1:numStates) {
    transitionMatrix[currentState, 1] <- (
      pnorm(
```

```r
        ((gridPoints[2] + gridPoints[1]) / 2
         - (1 - rho) * mu - rho * gridPoints[currentState])
        / sigma
        )
      )
    transitionMatrix[currentState, numStates] <- 1 - pnorm(
      ((gridPoints[numStates] + gridPoints[numStates - 1]) / 2
       - (1 - rho) * mu - rho * gridPoints[currentState])
      / sigma
      )
    }
  for (currentState in 1:numStates) {
    for (nextState in 2:(numStates - 1)) {
        transitionMatrix[currentState, nextState] <- (
          pnorm(
          ((gridPoints[nextState + 1] + gridPoints[nextState]) / 2
           - (1 - rho) * mu - rho * gridPoints[currentState])
          / sigma
          )
        - pnorm(
          ((gridPoints[nextState] + gridPoints[nextState - 1]) / 2
           - (1 - rho) * mu - rho * gridPoints[currentState])
          / sigma
          )
          )
        }
    }
  return(list(transitionMatrix = transitionMatrix, gridPoints = gridPoints))
}

calculateSteadyStateProb <- function(transitionMatrix) {
  (
    eigen(t(transitionMatrix))$vector[, 1]
    / sum(eigen(t(transitionMatrix))$vector[, 1])
  )
}

rescaleGridPoints <- function(transitionMatrix, gridPoints, data) {
  steadyStateProb <- calculateSteadyStateProb(transitionMatrix)
  rescaleScalar <- as.numeric(
    mean(data, na.rm = TRUE) / gridPoints
```

```r
    %*% steadyStateProb
    )
  gridPointsRescaled <- gridPoints * rescaleScalar
  return(list(gridPointsRescaled = gridPointsRescaled, steadyStateProb = steadyStateProb))
}

approximateAR1TauchenWithRescaling <- function(numStates, data, mu, rho, sigma) {
  TauchenResult <- approximateAR1Tauchen(numStates, data, mu, rho, sigma)
  transitionMatrix <- TauchenResult$transitionMatrix
  gridPoints <- TauchenResult$gridPoints
  gridPointsRescaledResult <- rescaleGridPoints(
    transitionMatrix, gridPoints, data
    )
  gridPointsRescaled <- gridPointsRescaledResult$gridPointsRescaled
  steadyStateProb <- gridPointsRescaledResult$steadyStateProb

  return(list(
    transitionMatrix = transitionMatrix,
    gridPointsRescaled = gridPointsRescaled,
    steadyStateProb = steadyStateProb
    ))
}

estimateHouseholdIncomeTransitionProcessByVillage <- function(
    village,
    numStates,
    data,
    laggedData
) {
  gridPointsArray <- array(NA, c(2, 2, numStates))
  transitionMatrixArray <- array(NA, c(2, 2, numStates, numStates))
  steadyStateProbArray <- array(NA, c(2, 2, numStates))
  AR1ParametersArray <- array(NA, c(2, 2, 3))

  for (incomeMeanClass in seq(1, 2)) {
    for (incomeCVClass in seq(1, 2)) {
      incdatRescaledMeanCVClass <- getDataByMeanCVClassByVillage(
        village, data, incomeMeanClass, incomeCVClass
        )
      laggeedIncdatRescaledMeanCVClass <- getDataByMeanCVClassByVillage(
        village, laggedData, incomeMeanClass, incomeCVClass
```

```
        )
      AR1Parameters <- calculateAR1Parameters(incdatRescaledMeanCVClass, laggeedIncdatResc
      TauchenResult <- approximateAR1TauchenWithRescaling(
        numIncomeStatesHH, incdatRescaledMeanCVClass,
        AR1Parameters$mu, AR1Parameters$rho, AR1Parameters$sigmau
        )
      gridPointsArray[incomeMeanClass, incomeCVClass,] <- TauchenResult$gridPoints
      transitionMatrixArray[incomeMeanClass, incomeCVClass,,] <- TauchenResult$transitionM
      steadyStateProbArray[incomeMeanClass, incomeCVClass,] <- TauchenResult$steadyStatePr
      AR1ParametersArray[incomeMeanClass, incomeCVClass,] <- unlist(AR1Parameters)
    }
  }

  return(list(
    gridPointsArray = gridPointsArray,
    transitionMatrixArray = transitionMatrixArray,
    steadyStateProbArray = steadyStateProbArray,
    AR1ParametersArray = AR1ParametersArray
    ))
}

householdAR1EstimationResult <- map(
  seq(1, numVillages),
  ~ estimateHouseholdIncomeTransitionProcessByVillage(., numIncomeStatesHH, incdatRescaled
)
```

1.3.1.2 Markov chain for income of households

```
# https://github.com/tlamadon/rutils/blob/74fa1b13998781547cd485b1bcf8863b49530285/R/inc.u
rouwenhorst <- function(rho, sigma, mu = 0, n){
    stopifnot(n > 1)
    qu <- (rho + 1) / 2
    nu <- ((n - 1) / (1 - rho^2))^(1 / 2) * sigma
    P  <- matrix(c(qu, 1 - qu, 1 - qu, qu), nrow = 2, ncol = 2)
    if (n > 2) {
        for (i in 2:(n - 1)){
            zeros    <- rep(0, i)
            zzeros   <- rep(0, i + 1)
            P        <- (
              qu * rbind(cbind(P, zeros, deparse.level = 0), zzeros, deparse.level = 0)
```

```
          + (1 - qu) * rbind(cbind(zeros, P, deparse.level = 0), zzeros, deparse.level
         + (1 - qu) * rbind(zzeros, cbind(P, zeros, deparse.level = 0), deparse.level =
         + qu * rbind(zzeros, cbind(zeros, P, deparse.level = 0), deparse.level = 0)
          )
         P[2:i, ] <- P[2:i, ] / 2
       }
    }
    zgrid <- seq(from = mu / (1 - rho) - nu, to = mu / (1 - rho) + nu, length = n)
    return(list(Pmat = P,zgrid = zgrid))
}
```

1.3.2 Village

1.3.2.1 Simulate income process for village

```
numVillageIncomeSimulations <- 1000
numVillageIncomeSimulationsPeriodDrop <- 100

set.seed(123)
randomNumberMatrix <- matrix(runif(hnum * numVillageIncomeSimulations), nrow = hnum)

sample1stPeriodIncomeStateByMeanCVClass <- function(
    meanClass,
    CVClass,
    numStates,
    steadyStateProbArray
    ) {
  sample(
    seq(1, numStates),
    1,
    prob = steadyStateProbArray[meanClass, CVClass, ]
    )
}

sampleConditionalIncomeStateByMeanCVClass <- function(
    meanClass,
    CVClass,
    numStates,
    transitionMatrixArray,
    previousState
```

```r
) {
  sample(
    seq(1, numStates),
    1,
    prob = transitionMatrixArray[meanClass, CVClass, previousState,]
  )
}

sampleAllIncomeStateByMeanCVClass <- function(
    meanClass,
    CVClass,
    numStates,
    steadyStateProbArray,
    transitionMatrixArray,
    numSimulation = numVillageIncomeSimulations
) {
  incomeStateVector <- vector(mode = "integer", length = numSimulation)
  incomeStateVector[1] <- sample1stPeriodIncomeStateByMeanCVClass(
    meanClass,
    CVClass,
    numStates,
    steadyStateProbArray
    )
  for (period in seq(2, numSimulation)) {
    incomeStateVector[period] <- sampleConditionalIncomeStateByMeanCVClass(
      meanClass,
      CVClass,
      numStates,
      transitionMatrixArray,
      incomeStateVector[period - 1]
      )
  }
  return(incomeStateVector)
}

simulateHouseholdIncomeByMeanCVClass<- function(
    meanClass,
    CVClass,
    numStates,
    steadyStateProbArray,
    transitionMatrixArray,
```

```r
    gridPointsArray,
    numSimulation = numVillageIncomeSimulations
) {
  incomeStateVec <- sampleAllIncomeStateByMeanCVClass(
    meanClass,
    CVClass,
    numStates,
    steadyStateProbArray,
    transitionMatrixArray)
  gridPointsArray[meanClass, CVClass, incomeStateVec]
}

simulateHouseholdIncomeByVillage <- function(
    village,
    meanClassVec,
    CVClassVec,
    numStates,
    householdAR1EstimationResult,
    .villageIndicatorMatrix = villageIndicatorMatrix
) {

  meanClassVillage <- meanClassVec[villageIndicatorMatrix[, village]]
  CVClassVillage <- CVClassVec[villageIndicatorMatrix[, village]]
  steadyStateProbArrayVillage <- householdAR1EstimationResult[[village]]$steadyStateProbAr
  transitionMatrixArrayVillage <- householdAR1EstimationResult[[village]]$transitionMatrix
  gridPointsArrayVillage <- householdAR1EstimationResult[[village]]$gridPointsArray

  do.call(
    rbind,
    map2(
      meanClassVillage, CVClassVillage,
      ~ simulateHouseholdIncomeByMeanCVClass(
        .x, .y, numStates, steadyStateProbArrayVillage, transitionMatrixArrayVillage, grid
        )
    )
  )

}

simulatedHouseholdIncome <- map(
  seq(1, numVillages),
```

```
  ~ simulateHouseholdIncomeByVillage(
    .,
    householdIncMeanClassVec,
    householdIncCVClassVec,
    numIncomeStatesHH,
    householdAR1EstimationResult
    )
)

estimateVillagencomeTransitionProcessByVillage <- function(
    village,
    meanClassVec,
    CVClassVec,
    numStatesHH,
    numStatesVillage,
    householdAR1EstimationResult,
    .villageIndicatorMatrix = villageIndicatorMatrix,
    .numVillageIncomeSimulationsPeriodDrop = numVillageIncomeSimulationsPeriodDrop,
    .numVillageIncomeSimulations = numVillageIncomeSimulations
){
  householdIncomeSimulationResult <- simulateHouseholdIncomeByVillage(
    village,
    meanClassVec,
    CVClassVec,
    numStatesHH,
    householdAR1EstimationResult
  )

  villageSimulatedIncMean <- colMeans(
    householdIncomeSimulationResult[, .numVillageIncomeSimulationsPeriodDrop:.numVillageIn
  )
  villageSimulatedIncLogMean <- colMeans(
    householdIncomeSimulationResult[, .numVillageIncomeSimulationsPeriodDrop:.numVillageIn
  ) %>% log
  villageSimulatedLaggedIncLogMean <- colMeans(
    householdIncomeSimulationResult[
      , (.numVillageIncomeSimulationsPeriodDrop - 1):(.numVillageIncomeSimulations - 1)
      ]
  ) %>% log

  villageAR1Parameters <- calculateAR1Parameters(
```

```r
    villageSimulatedIncLogMean,
    villageSimulatedLaggedIncLogMean
  )

  villageAR1TauchenApproximation <- approximateAR1Tauchen(
    numStatesVillage, villageSimulatedIncLogMean,
    villageAR1Parameters$mu, villageAR1Parameters$rho, villageAR1Parameters$sigmau
  )

  villageIncomeGridPoints <- calculateGridPoints(numStatesVillage, villageSimulatedIncMean

  villageIncomeGridPointsRescaled <- rescaleGridPoints(
      villageAR1TauchenApproximation$transitionMatrix,
      villageIncomeGridPoints,
      villageSimulatedIncMean
      )

  return(list(
    transitionMatrix = villageAR1TauchenApproximation$transitionMatrix,
    gridPoints = villageIncomeGridPointsRescaled$gridPointsRescaled
  ))
}


villageAR1EstimationResult <- map(
  seq(1, numVillages),
  ~ estimateVillagencomeTransitionProcessByVillage(
      .,
      householdIncMeanClassVec,
      householdIncCVClassVec,
      numIncomeStatesHH,
      numIncomeStatesVillage,
      householdAR1EstimationResult
  )
)
```

1.4 Sanity check: compare against the parameters in the original paper