# Estimation of risk-sharing model with limited commitment

Mizuhiro Suzuki

# Table of contents

# Preface

This website showcases how to conduct a structural estimation of a risk-sharing model with limited commitment. It contains explanations on both theory and empirics. The code is largely based on the replication package of Laczó (2015). However, the purpose of this website is *not* to reproduce the paper's results. Actually, I simplified the estimation by, for example, reducing the parameters to estimate to focus on the core idea. Also I do not estimate the case with heterogeneous preferences, which was the main focus of the paper, because my focus here is to show the base risk-sharing model with limited commitment. The R code is greatly refactored, mainly for my learning purpose, which is another reason why some of the simulation results do not match although the original R script properly set the random seed for reproducibility.

Here I would like to thank Sarolta Laczó for providing the replication package to the journal website. I believe this is a good example of how such replication package can help people's understanding and learning, not just for verifying the reproducibility of studies.

Of course, any errors are my own. And any comments/feedback are greatly appreciated. If you use GitHub, please use the issue page of the GitHub repo. Otherwise, please send an email to mizuhiro.suzuki@gmail.com .

# 1 Data preparation

In this page, I show the data preparation process. The main goal in the data prep is to estimate the discretized income processes of households and the "village" (= average of households in a village).

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

## 1.1 Global settings

```r
set.seed(123)
```

## 1.2 Load datasets

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

### 1.2.1 Basic data processes of raw data

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

```r
createVillageAggregateByYear <- function(
    data,
    func,
    numVillages,
    villageIndicatorMatrix
    ) {
  do.call(
    rbind,
    map(
      seq(1, numVillages),
      ~ data[villageIndicatorMatrix[, .],] %>% func
      )
    )
  }

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

```r
vilMeanIncByYear <- createVillageAggregateByYear(incdat, colMeans, numVillages, villageInd
vilMeanConsByYear <- createVillageAggregateByYear(consdat, colMeans, numVillages, villageI
vilMeanLogCons <- createVillageAggregateByYear(log(consdat), colMeans, numVillages, villag
```

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

Since there is no saving in the model, the consumption and income should coincide. For this, I rescale the income so that the means of consumption and income are identical in each village.

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

```
incRescaledLow <- createVillageAggregate(
  incdatRescaled,
  function(x) quantile(x, 0.025, na.rm = TRUE),
  numVillages,
  villageIndicatorMatrix
  )
incRescaledHigh <- createVillageAggregate(
  incdatRescaled,
  function(x) quantile(x, 0.975, na.rm = TRUE),
  numVillages,
  villageIndicatorMatrix
  )
```

## 1.3 Estimate income processes

The main step of this data processing is to estimate the income process, first of households and second of the "village" (= average of households in a village). We want the income process of the village since, in estimation, we consider the risk-sharing transfers between a household and "the rest of the village" so that we can consider one-to-one transfers instead of one-to-n transfers.

I follow the estimation process of Laczó (2015), which is detailed in the paper's online appendix. First I estimate the income process of households by types, where the types are by income's mean and coefficient of variance (CV). Here, in total we have 4 types (high/low mean income X high/low CV income). If there were long enough income data, we would be able to estimate the income process of each household, but given the 6 observations per household for income in ICRISAT data, this is not feasible. Also, to mitigate the effect of outliers, income smaller than 2.5 percentile or larger than 97.5 percentile in each village is not used for income process estimation (but they are used for estimation in the risk sharing model).

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

### 1.3.1 Household

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

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

#### 1.3.1.1 Estimate AR(1) process

First I estimate the AR(1) process of households' income. In particular, the following AR(1) process is estimated:

$$y_{it} = (1 - \rho)\mu + \rho y_{i,t-1} + u_{it},$$

where $y_{it}$ is the income of a household $i$ in period $t$.

The parameters are given as

$$\mu = E(y_{it}) \rho = Cor(y_{it}, y_{i,t-1}) \sigma_u^2 = (1 - \rho^2) Var(y_{it}).$$

```r
householdIncMeanClassVec <- (
  householdIncMean > (villageIndicatorMatrix %*% vilIncMeanMedian)
  ) + 1
householdIncCVClassVec <- (
  householdIncCV > (villageIndicatorMatrix %*% vilIncCVMedian)
  ) + 1

incdatRescaledForAR1Estimation <- incdatRescaled
laggedIncdatRescaledForAR1Estimation <- cbind(
  NA,
  incdatRescaled[, seq(1, tnum - 1)]
  )

incdatRescaledForAR1Estimation[
  (incdat <= as.vector(villageIndicatorMatrix %*% incLow)) |
    (incdat >= as.vector(villageIndicatorMatrix %*% incHigh))] <- NA
laggedIncdatRescaledForAR1Estimation[
  (incdat <= as.vector(villageIndicatorMatrix %*% incLow)) |
    (incdat >= as.vector(villageIndicatorMatrix %*% incHigh))] <- NA

getDataByMeanCVClassByVillage <- function(
    village,
    data,
    meanClass,
    CVClass,
    meanClassVec,
    CVClassVec,
    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))
}
```

#### 1.3.1.2 Approximate the AR(1) process with Markov chain for income of households

Given the estimated parameters, I approximate the AR(1) process with discretization. For this, Laczó (2015) used Tauchen's method with a small modification that, instead of assigning the bounds of grid points with a parameter, the quantiles of income distributions are used to determine each grid point. To guarantee that the mean income at the steady state of the estimated process coincides with the actual mean income, the grid points are rescaled at the end.

```
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)

  transitionMatrix <- array(NA, c(numStates, numStates))

  for (currentState in 1:numStates) {
    transitionMatrix[currentState, 1] <- (
      pnorm(
        ((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
    %*% 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,
    householdIncMeanClassVec,
    householdIncCVClassVec,
    villageIndicatorMatrix
) {
  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,
        householdIncMeanClassVec, householdIncCVClassVec, villageIndicatorMatrix
        )
      laggeedIncdatRescaledMeanCVClass <- getDataByMeanCVClassByVillage(
        village, laggedData, incomeMeanClass, incomeCVClass,
        householdIncMeanClassVec, householdIncCVClassVec, villageIndicatorMatrix
        )
      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,
    numHouseholds = villageIndicatorMatrix[, village] %>% sum
    ))
}

householdAR1EstimationResult <- map(
  seq(1, numVillages),
  ~ estimateHouseholdIncomeTransitionProcessByVillage(
    .,
    numIncomeStatesHH,
    incdatRescaledForAR1Estimation,
    laggedIncdatRescaledForAR1Estimation,
    householdIncMeanClassVec,
    householdIncCVClassVec,
    villageIndicatorMatrix
    )
)
```

### 1.3.2 Village

Given the estimated household income processes, I estimate the income process of the "village". For this, I first simulate the average village income, using the parameters estimated above. In particular, I simulate the household income over 1000 periods, then compute the mean income in each period. After excluding the first 100 observations to use the income at the steady state, I estimate the parameters using the same method as for the household income.

#### 1.3.2.1 Simulate income process for village

```r
numVillageIncomeSimulations <- 1000
numVillageIncomeSimulationsPeriodDrop <- 100
```

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

sampleConditionalIncomeStateByMeanCVClass <- function(
    meanClass,
    CVClass,
    numStates,
    transitionMatrixArray,
    previousState
) {
  sample(
    seq(1, numStates),
    1,
    prob = transitionMatrixArray[meanClass, CVClass, previousState,]
  )
}
```

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

simulateHouseholdIncomeByMeanCVClass <- function(
    meanClass,
    CVClass,
    numStates,
    steadyStateProbArray,
    transitionMatrixArray,
    gridPointsArray,
    numSimulations = numVillageIncomeSimulations
) {
  incomeStateVec <- sampleAllIncomeStateByMeanCVClass(
    meanClass,
    CVClass,
    numStates,
    steadyStateProbArray,
    transitionMatrixArray)
  gridPointsArray[meanClass, CVClass, incomeStateVec]
}

simulateHouseholdIncomeByVillage <- function(
    village,
    meanClassVec,
    CVClassVec,
    numStates,
    householdAR1EstimationResult,
    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,
        gridPointsArrayVillage
        )
    )
  )

}
```

#### 1.3.2.2 Estimate the income process of the village

Given the simulated income data, I estimate the income process parameters and approximate the process with Tauchen's method.

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

  villageSimulatedIncMean <- colMeans(
    householdIncomeSimulationResult[
      , numVillageIncomeSimulationsPeriodDrop:numVillageIncomeSimulations
      ]
  )
  villageSimulatedLaggedIncMean <- colMeans(
    householdIncomeSimulationResult[
      , (numVillageIncomeSimulationsPeriodDrop - 1):(numVillageIncomeSimulations - 1)
      ]
  )

  villageAR1Parameters <- calculateAR1Parameters(
    villageSimulatedIncMean,
    villageSimulatedLaggedIncMean
  )

  villageAR1TauchenApproximation <- approximateAR1Tauchen(
    numStatesVillage, villageSimulatedIncMean,
    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,
      villageIndicatorMatrix,
      numVillageIncomeSimulationsPeriodDrop,
      numVillageIncomeSimulations
  )
)
```

## 1.4 Sanity check: compare against the parameters in Laczó (2015)

Just for a sanity check of my code, I compare the household AR(1) process parameters I derived against the ones provided in the appendix of Laczó (2015). The parameters in the tables below coincide to the parameters in the paper appendix, except the ones for "Low mean, high risk" in Shirapur. I reran the original R script and confirmed that the correct parameters are the ones that I am showing in the table below.

Since I refactored the origianl R script, I cannot reproduce the village parameters. To be clear, the author has set the random seed in the code, and the reason I cannot reproduce the village parameters is merely because of the difference in code structures.

|  | Low mean, low risk | Low mean, high risk | High mean, low risk | High mean, high risk |
|---|---|---|---|---|
| mu | 182.475 | 160.201 | 441.843 | 391.798 |
| rho | 0.189 | 0.626 | 0.550 | 0.365 |
| sigmau_squared | 49.088 | 68.506 | 128.741 | 188.458 |

```r
createParameterTableByVillage <- function(
    village,
    householdAR1EstimationResult
) {

  villageParameters <- array(NA, c(3, 4))
  colIndex <- 0
  for (incomeMeanClass in seq(1, 2)) {
    for (incomeCVClass in seq(1, 2)) {
      colIndex <- colIndex + 1
      villageParameters[, colIndex] <-
        householdAR1EstimationResult[[village]]$AR1ParametersArray[incomeMeanClass, income
    }
  }
  rownames(villageParameters) <- c(
    "mu",
    "rho",
    "sigmau_squared"
  )
  villageParameters %>%
    kbl(digits = 3) %>%
    kable_classic() %>%
    add_header_above(
      c(
        "",
        "Low mean, \n low risk",
        "Low mean, \n high risk",
        "High mean, \n low risk",
        "High mean, \n high risk"
      )
    )
}
createParameterTableByVillage(1, householdAR1EstimationResult)
```

|  | Low mean, low risk | Low mean, high risk | High mean, low risk | High mean, high risk |
|---|---|---|---|---|
| mu | 235.906 | 243.872 | 506.555 | 525.520 |
| rho | 0.491 | 0.285 | 0.846 | 0.520 |
| sigmau_squared | 45.808 | 79.111 | 122.302 | 237.918 |

|  | Low mean, low risk | Low mean, high risk | High mean, low risk | High mean, high risk |
|---|---|---|---|---|
| mu | 263.292 | 288.625 | 446.590 | 642.628 |
| rho | 0.318 | -0.150 | 0.160 | 0.199 |
| sigmau_squared | 78.793 | 126.987 | 129.785 | 271.997 |

```r
createParameterTableByVillage(2, householdAR1EstimationResult)
```

```r
createParameterTableByVillage(3, householdAR1EstimationResult)
```

## 1.5 Save data

```r
save.image("IntermediateData/allData.RData")
```

# 2 Full risk-sharing under homogeneous preferences

Here, first I review the theoretical result and then derive the likelihood function. Note that the model here estimates parameters *under the assumption* that the model follows the full risk-sharing model. This is in contrast to many empirical papers which *test* a full riks-sharing model (such as Townsend (1994)).

### 2.0.1 Settings

Suuposed that there are $N$ households in a village. Each household maximizes its expected lifetime utility:

$$E\left[\sum_{t=1}^{\infty} \delta^t u(c_{it})\right],$$

where $\delta$ is the discount factor, $u$ is instantaneous preference of households (note that here I assume homogeneous preference), and $c_{it}$ is consumption by household $i$ at time $t$. Assume that there is no saving.

For income, let $s^t = (s_1, \dots, s_t)$ the history of income states from time 1 to $t$, and let $y_{it}(s_t)$ be household $i$'s income at time $t$ and state $s_t$. Assume that the income process is a Markov process and is independent across households.

With these, I find the Pareto-optimal allocations. For this, a weighted sum of households' expected lifetime utilities is maximized. In other words, a social planner solves the following maximization problem:

$$\max_{\{c_{it}(s^t)\}} \quad \sum_i \lambda_i \sum_{t=1}^{\infty} \sum_{s^t} \delta^t \pi(s^t) u(c_{it}(s^t))$$

subject to the resource constraint

$$\sum_i c_{it}(s^t) \leq \sum_i y_{it}(s_t) \quad \forall s^t, \forall t,$$

where $\lambda_i$ is the Pareto weight of $i$, $\pi(s^t)$ is the probability that the history $s^t$ is realized, and $c_{it}(s^t)$ is $i$'s consumption under the history $s^t$.

### 2.0.2 Optimization condition

Solving this maximization problem, we can get a well-known result that, for any two households $i$ and $j$ in the village, the following holds:

$$\frac{u'(c_{jt}(s^t))}{u'(c_{it}(s^t))} = \frac{\lambda_i}{\lambda_j} \quad \forall s^t, \forall t.$$

If I assume the utility function to take the CRRA form, that is,

$$u(c_{it}) = \frac{c_{it}^{1-\sigma} - 1}{1-\sigma},$$

then

$$u'(c_{it}) = c_{it}^{-\sigma},$$

and thus I get

$$\frac{c_{jt}(s^t)^{-\sigma}}{c_{it}(s^t)^{-\sigma}} = \frac{\lambda_i}{\lambda_j} \quad \forall s^t, \forall t.$$

Dropping $s^t$ from the quation for simplicity and taking the logarithms, I obtain

$$\begin{aligned}
-\sigma \log(c_{jt}) + \sigma \log(c_{it}) &= \log(\lambda_i) - \log(\lambda_j) \\
\Leftrightarrow \log(c_{it}) &= \frac{1}{\sigma}\big(\log(\lambda_i) - \log(\lambda_j)\big) + \log(c_{jt}) \\
\Leftrightarrow N\log(c_{it}) &= \frac{1}{\sigma}\left(N\log(\lambda_i) - \sum_j \log(\lambda_j)\right) + \sum_j \log(c_{jt}) \\
\Leftrightarrow \log(c_{it}) &= \frac{1}{\sigma}\left(\log(\lambda_i) - \frac{1}{N}\sum_j \log(\lambda_j)\right) + \frac{1}{N}\sum_j \log(c_{jt}).
\end{aligned}$$

By defining the village-average consumption, $c_{vt}$, as $\log(c_{vt}) = \frac{1}{N}\sum_j \log(c_{jt})$, and taking the first difference from the equation above, I get

$$\log(c_{it}) - \log(c_{i,t-1}) = \log(c_{vt}) - \log(c_{v,t-1}).$$

### 2.0.3 Introducing measurement errors

Here I introduce the measurement error in consumption, which is assumed to be multiplicative and log-normally distributed. In particular, I denote observed consumption, $c_{it}^*$ as $c_{it}^* =$

$c_{it}\exp(\varepsilon^c_{it})$, where $\varepsilon_{it} \sim N(0,\gamma_c^2)$ is the measurement error which is i.i.d. across households and time. With this, the observed village-average consumption, $c^*_{vt}$ is so that

$$\begin{aligned}
\log(c^*_{vt}) &= \frac{1}{N}\sum_i \log(c^*_{it}) \\
&= \frac{1}{N}\sum_i \log(c_{it}) + \frac{1}{N}\sum_i \varepsilon^c_{it} \\
&= \log(c_{vt}) + \frac{1}{N}\sum_i \varepsilon^c_{it} \\
&= \log(c_{vt}) + \varepsilon^c_{vt},
\end{aligned}$$

where the village consumption measurement error is defined as $\varepsilon^c_{vt} \equiv \frac{1}{N}\sum_i \varepsilon^c_{it}$. Substituting these into $\log(c_{it}) - \log(c_{i,t-1}) = \log(c_{vt}) - \log(c_{v,t-1})$, I obtain

$$\log(c^*_{it}) - \log(c^*_{i,t-1}) = \log(c^*_{vt}) - \log(c^*_{v,t-1}) + \left(\varepsilon^c_{it} - \varepsilon^c_{vt} - (\varepsilon^c_{i,t-1} - \varepsilon^c_{v,t-1})\right).$$

### 2.0.4 Likelihood function

Since $\varepsilon_{it} \sim N(0,\gamma_C^2)$, I get $\varepsilon_{vt} \sim N(0,\frac{\gamma_C^2}{N})$, and note that $Cov(\varepsilon_{it},\varepsilon_{vt}) = \frac{\gamma_C^2}{N}$. With the independence across periods and households of measurement errors, I obtain

$$Var(\varepsilon^c_{it} - \varepsilon^c_{vt} - (\varepsilon^c_{i,t-1} - \varepsilon^c_{v,t-1})) = 2\left(\gamma_C^2 + \frac{\gamma_C^2}{N} - 2\frac{\gamma_C^2}{N}\right) = 2\gamma_C^2\left(1 - \frac{1}{N}\right).$$

Therefore, the likelihood function is

$$L(\theta) = \prod_i \prod_{t=2}^T f\left(\log\left(\frac{c^*_{it}}{c^*_{i,t-1}}\right) - \log\left(\frac{c^*_{vt}}{c^*_{v,t-1}}\right), \theta\right),$$

and the log likelihood function is

$$l(\theta) = \sum_i \sum_{t=2}^T \log\left(f\left(\log\left(\frac{c^*_{it}}{c^*_{i,t-1}}\right) - \log\left(\frac{c^*_{vt}}{c^*_{v,t-1}}\right), \theta\right)\right),$$

where $f(x,\theta)$ is the density function of $x \sim N\left(0, \sqrt{2\gamma_C^2\left(1-\frac{1}{N}\right)}\right)$, and $\theta$ is a parameter to estimate (here, $\theta = \gamma_C$). In estimation, I find the parameter to maximize $l(\theta)$.

Without assuming that the model is correctly specified but assuming that there is no autocorrelation, the standard error of the parameter is $A(\hat{\theta})^{-1}B(\hat{\theta})A(\hat{\theta})^{-1}$, where $A(\hat{\theta})$ is the Hessian matrix of the log-likelihood function and $B(\hat{\theta})$ is the outer product of scores (= gradient of log likelihood).

#### 2.0.4.1 Notes

- This model considers that any deviation from the full risk-sharing model is due to measurement errors in consumption.
- I derive the unconditional likelihood function, as opposed to Laczó (2015) who used a conditional likelihood function with simulation. The author clarifies her choice to use the simulation method in a footnote as follows: "Note that it is not necessary to use simulation to take into account measurement error in consumption at time t in the perfect risk-sharing case. I do it to be consistent with the LC case."

## 2.1 Code for estimation

This R script estimates the full risk sharing model.

```r
pacman::p_load(
  tidyverse,
  kableExtra,
  numDeriv
)
```

```r
load('IntermediateData/allData.RData')
```

```r
calculateLogLikelihoodEachObsFullHom <- function(
    param,
    currentCons,
    previousCons,
    currentVilLogCons,
    previousVilLogCons,
    numHouseholds
    ) {

  gammaC2 <- param

  logLikelihood <- dnorm(
    (
      log(currentCons / previousCons)
      - (currentVilLogCons - previousVilLogCons)
    ), sd = sqrt(2 * gammaC2 * (1 - 1 / numHouseholds)),
    log = TRUE
  )
  return(logLikelihood)
}

calculateScoreEachObsFullHom <- function(
    param,
    currentCons,
    previousCons,
    currentVilLogCons,
    previousVilLogCons,
    numHouseholds
) {
  grad(
    calculateLogLikelihoodEachObsFullHom,
    param,
    currentCons = currentCons,
    previousCons = previousCons,
    currentVilLogCons = currentVilLogCons,
    previousVilLogCons = previousVilLogCons,
    numHouseholds = numHouseholds
  )
}

calculateScoreFullHom <- function(
    param,
    consdatByVillage,
    vilMeanLogConsByVillage,
    numHouseholds,
    .tnum = tnum
) {

  numParameters <- length(param)

  scoreMatrix <- matrix(0, nrow = numParameters, ncol = numParameters)
  for (householdIndex in seq(1, numHouseholds)) {
    for (periodIndex in seq(2, .tnum)) {
      score <- calculateScoreEachObsFullHom(
          param,
          consdatByVillage[householdIndex, periodIndex],
          consdatByVillage[householdIndex, (periodIndex - 1)],
          vilMeanLogConsByVillage[periodIndex],
          vilMeanLogConsByVillage[(periodIndex - 1)],
          numHouseholds
        )
      scoreMatrix <- (
        scoreMatrix
        + outer(score, score)
      )
    }
  }
  return(scoreMatrix)
}

calculateLogLikelihoodFullHom <- function(
    param,
    consdatByVillage,
    vilMeanLogConsByVillage,
    numHouseholds,
    .tnum = tnum
    ) {

  logLikelihood <- 0
  for (householdIndex in seq(1, numHouseholds)) {
    for (periodIndex in seq(2, .tnum)) {
      logLikelihood <- (
        logLikelihood
        + calculateLogLikelihoodEachObsFullHom(
          param,
          consdatByVillage[householdIndex, periodIndex],
          consdatByVillage[householdIndex, (periodIndex - 1)],
          vilMeanLogConsByVillage[periodIndex],
          vilMeanLogConsByVillage[(periodIndex - 1)],
          numHouseholds
        )
      )
    }
  }

  return(- logLikelihood)
}

calculateStandardErrors <- function(
    estimationResult,
```

```r
    consdatByVillage,
    vilMeanLogConsByVillage,
    numHouseholds
) {
  (
    solve(estimationResult$hessian) %*%
    calculateScoreFullHom(
      estimationResult$par,
      consdatByVillage,
      vilMeanLogConsByVillage,
      numHouseholds
    ) %*%
    solve(estimationResult$hessian)
  ) %>% sqrt %>% diag
}
```

```r
estimateMLFullHom <- function(
    village,
    consdat,
    vilMeanLogCons,
    lowerBound,
    upperBound,
    villageIndicatorMatrix
) {

  numHouseholds <- villageIndicatorMatrix[, village] %>% sum
  consdatByVillage <- consdat[villageIndicatorMatrix[, village], ]
  vilMeanLogConsByVillage <- vilMeanLogCons[village, ]

  estimationResult <- optim(
    0.1,
    calculateLogLikelihoodFullHom,
    consdatByVillage = consdatByVillage,
    vilMeanLogConsByVillage = vilMeanLogConsByVillage,
    numHouseholds = numHouseholds,
    method = "L-BFGS-B",
    lower = lowerBound,
    upper = upperBound,
    control = list(trace = 0, maxit = 200),
    hessian = TRUE)
  standardErrors <- calculateStandardErrors(
    estimationResult,
    consdatByVillage,
    vilMeanLogConsByVillage,
    numHouseholds
  )

  return(list(
    parameter = estimationResult$par,
    logLikelihood = - estimationResult$value,
    standardErrors = standardErrors,
    optimizationResult = estimationResult$message
  ))
}

estimationResultFullHom <- map(
  seq(1, numVillages),
  ~ estimateMLFullHom(
      .,
      consdat,
      vilMeanLogCons,
      1e-3,
      10,
      villageIndicatorMatrix
  )
)
```

## 2.2 Estimation result

```
estimationFullHomTable <- do.call(
  cbind,
  map(
    seq(1, numVillages),
    ~ c(
      formatC(estimationResultFullHom[[.]]$parameter, digits = 3, format = "f"),
      str_interp('(${formatC(estimationResultFullHom[[.]]$standardErrors, digits = 3, form
      formatC(estimationResultFullHom[[.]]$logLikelihood, digits = 3, format = "f")
    )
  )
```

|  | Aurepalle | Kanzara | Shirapur |
|---|---|---|---|
| Variance of consumption measurement errors | 0.036 | 0.037 | 0.048 |
|  | (0.004) | (0.010) | (0.007) |
| Log likelihood | -13.808 | -20.166 | -36.422 |

```
)
colnames(estimationFullHomTable) <- c("Aurepalle", "Kanzara", "Shirapur")
rownames(estimationFullHomTable) <- c(
  "Variance of consumption measurement errors",
  "",
  "Log likelihood"
  )

estimationFullHomTable %>%
  kbl(digits = 3) %>%
  kable_classic()
```

# 3 Risk sharing with limited commitoment: model

Here I demonstrate how to compute value functions and consumption under risk-sharing with limited commitment. First I show the risk-sharing model with limited commitment, and then I show the calculation of value functions.

I consider constrained-efficient consumption allocations. The social planner solves the following problem:

$$\begin{aligned} &\max_{\{c_{it}(s^t)\}} \sum_i \lambda_i \sum_{t=1}^{\infty} \sum_{s^t} \delta^t \pi(s^t) u(c_{it}(s^t)) \\ \text{subject to} &\sum_i c_{it}(s^t) \leq \sum_i y_{it}(s_t) \quad \forall s^t, \forall t \\ &\sum_{r=t}^{\infty} \sum_{s^r} \delta^{r-t} \pi(s^r|s^t) u(c_{ir}(s^r)) \geq U_i^{aut}(s_t) \quad \forall s^t, \forall t, \forall i. \end{aligned}$$

Here, the income follows a Markov process and is independent across households. Notice the difference between the history of states up to period $t$ ($s^t$) and the state at period $t$ ($s_t$). The variable $\lambda_i$ is the Pareto weight of a household $i$. The last equation is the participation constraints (PCs), whose RHS is the value of autarky and the solution of the following Bellman equation:

$$U_i^{aut}(s_t) = u((1-\phi)y_{it}(s_t)) + \delta \sum_{s^{t+1}} \pi(s_{t+1}|s_t) U_i^{aut}(s_{t+1}),$$

where $\phi$ is the punishment of renege, which is a fraction of consumption each period. It is assumed that savings are absent.

Letting the multiplier on the PC of $i$ be $\delta^t \pi(s^t) \mu_i(s^t)$ and the multiplier on the aggregate resource constraint be $\delta^t \pi(s^t) \rho(s^t)$, the Lagrangian is

$$\sum_{t=1}^{\infty} \sum_{s^t} \delta^t \pi(s^t) \left\{ \sum_i \left[ \lambda_i u(c_{it}(s^t)) + \mu_i(s^t) \left( \sum_{r=t}^{\infty} \sum_{s^r} \delta^{r-t} \pi(s^r|s^t) u(c_{ir}(s^r)) - U_i^{aut}(s_t) \right) \right] + \rho(s^t) \left( \sum_i (y_{it}(s_t) - \right.\right.$$

With the recursive method in Marcet and Marimon (2019), this Lagrangian can be written as

$$\sum_{t=1}^{\infty}\sum_{s^t}\delta^t\pi(s^t)\left\{\sum_i\left[M_i(s^{t-1})u(c_{it}(s^t))+\mu_i(s^t)(u(c_{it}(s^t))-U_i^{aut}(s_t))\right]+\rho(s^t)\left(\sum_i(y_{it}(s_t)-c_{it}(s^t))\right)\right\},$$

where $M_i(s^t)=M_i(s^{t-1})+\mu_i(s^t)$ and $M_i(s^0)=\lambda$. The variable $M_i(s^t)$ is the current Pareto weight of household $i$ and is equal to its initial Pareto weight plus the sum of the Lagrange muliplier on its PCs along the history $s^t$.

From the Lagrangian, the optimality condition is

$$u'(c_{it}(s^t))M_i(s^t)=\frac{\rho(s^t)}{\delta^t\pi(s^t)},$$

and thus, for two households $i$ and $j$ ($i\neq j$),

$$u'(c_{it}(s^t))M_i(s^t)=u'(c_{jt}(s^t))M_j(s^t).$$

Taking logarithms and summing over households $j\neq i$, I get

$$\log\left(u'(c_{it}(s^t))\right)+\log\left(M_i(s^t)\right)=\frac{1}{N-1}\sum_{j\neq i}\log\left(u'(c_{jt}(s^t))\right)+\frac{1}{N-1}\sum_{j\neq i}\log\left(M_j(s^t)\right).$$

The pareto weights of $N$-households are intractable and the dimension of the value function becomes too large. Hence, I take the "one-versus-the-rest" approach. In particular, assume that "the rest of the village" has the same consumption ($c_{vt}(s^t)$) and Pareto weights ($M_v(s^t)$). Then the equation above becomes

$$\begin{aligned}\log\left(u'('c_{it}(s^t))\right)+\log\left(M_i(s^t)\right)&=\log\left(u'(c_{vt}(s^t))\right)+\log\left(M_v(s^t)\right)\\ \Leftrightarrow\frac{u'(c_{vt}(s^t))}{u'(c_{it}(s^t))}&=\frac{M_i(s^t)}{M_v(s^t)}\end{aligned}$$

Note that this is equivalent to the optimality condition that the ratio of marginal utilities between two "households", $i$ and $v$, equals the ratio of their Pareto weights.

Let $x_i(s^t)=\frac{M_i(s^t)}{M_v(s^t)}$, the relative Pareto weight of household $i$ under the history $s^t$. Then, the vector of relative weights $x(s^t)$ plays as a role as a co-state variable, and the solution consists of policy functions $x_{it}(s_t,x_{t-1})$ and $c_{it}(s_t,x_{t-1})$. That is, $x_{t-1}$ is a sufficient statistic for the history up to $t-1$. The optimality condition is

$$\frac{u'(c_{vt}(s_t,x_{t-1}))}{u'(c_{it}(s_t,x_{t-1}))}=x_{it}(s_t,x_{t-1})\quad\forall i.$$

Using the policy functions, the individual value functioon can be written recursively as

$$V_i(s_t, x_{t-1}) = u(c_{it}(s_t, x_{t-1})) + \delta \sum_{s_{t+1}} \pi(s_{t+1}|s_t) V_i(s_{t+1}, x_t(s_t, x_{t-1})).$$

The evolution of relative Pareto weights is fully characterized by state-dependent intervals, which give the weights in the case where PCs are binding (Ligon, Thomas, and Worrall (2002)).

## 3.1 Code

```
pacman::p_load(
  tidyverse,
  kableExtra,
  latex2exp
)
```

### 3.1.1 Utility functions

```
calculateUtility <- function(cons, sigma) {
  if (sigma != 1) {
    utility = (cons^(1 - sigma) - 1) / (1 - sigma)
  } else if (sigma == 1) {
    utility = log(cons)
  }
  return(utility)
}
calculateMarginalUtility <- function(cons, sigma) cons^(- sigma)
```

### 3.1.2 Value under autarky

```
calculateAutarkyValue <- function(
    incomeGridPoints,
    sigma,
    delta,
    punishment,
    incomeTransitionMatrix
) {

  autarkyValue <- numeric(length = length(incomeGridPoints))
  i <- 1
  diff <- 1
  while (diff > 1e-12) {
    autarkyValueNew <- (
      calculateUtility(incomeGridPoints * (1 - punishment), sigma)
      + delta * incomeTransitionMatrix %*% autarkyValue
    )
    diff <- max(abs(autarkyValueNew - autarkyValue))
    autarkyValue <- autarkyValueNew
    i <- i + 1
  }
  return(autarkyValue)
}
```

### 3.1.3 Create grid of relative Pareto weights

Here, I make the grid of relative Pareto weight of the household, $x(s^t)$, on which I compute the values (I use the notation $x(s^t)$ rather than $x_i(s^t)$ since there are only two households).

```
getRelativeParetoWeightsGridPoints <- function(
    sigma,
    punishment,
    householdIncomeGridPoints,
    villageIncomeGridPoints,
    numRelativeParetoWeights
    ) {

  minRelativeParetoWeights <- (
    calculateMarginalUtility(max(villageIncomeGridPoints), sigma)
    / calculateMarginalUtility(min(householdIncomeGridPoints * (1 - punishment)), sigma)
  )
  maxRelativeParetoWeights <- (
    calculateMarginalUtility(min(villageIncomeGridPoints * (1 - punishment)), sigma)
    / calculateMarginalUtility(max(householdIncomeGridPoints), sigma)
  )
  relativeParetoWeightsGridPoints <- exp(
    seq(
      log(minRelativeParetoWeights),
      log(maxRelativeParetoWeights),
      length.out = numRelativeParetoWeights)
    )
  return(relativeParetoWeightsGridPoints)
}
```

### 3.1.4 Calculate consumption on the grid points

Then, I compute consumptions of the household on these grid points. From the optimality condition and the CRRA utility functions, we obtain

$$
\begin{aligned}
\frac{c_{vt}^{-\sigma}}{c_{it}^{-\sigma}} &= x_t \\
\Leftrightarrow c_{vt} &= c_{it} x_t^{-1/\sigma} \\
\Leftrightarrow (N-1)c_{vt} &= (N-1)c_{it} x_t^{-1/\sigma} \\
\Leftrightarrow c_{it} + (N-1)c_{vt} &= c_{it}\left(1 + (N-1)x_t^{-1/\sigma}\right) \\
\Leftrightarrow c_{it} &= \frac{y_t}{\left(1 + (N-1)x_t^{-1/\sigma}\right)},
\end{aligned}
$$

where $y_t$ is the village aggregate income and the last transformation used the village wide equals the aggregate income due to the no saving assumption.

```
calculateHouseholdConsumption <- function(
  aggregateIncome,
  relativeParetoWeight,
  numHouseholds,
  sigma
) {
    aggregateIncome / (1 + (numHouseholds - 1) * (relativeParetoWeight^(- 1 / sigma)))
}
```

### 3.1.5 Caclulate values under full risk-sharing

Now, I compute the values under full risk-sharing, which will be used as the initial values in value function iterations under the limited commitment model. Note that, under full risk sharing, the consumption only depends on the aggregate resources and time-invariate relative Pareto weights. Hence, I numerically solve the following Bellman equation:

$$V_i^{full}(s_t, x) = u(c_{it}(s_t, x)) + \delta \sum_{s^{t+1}} \pi(s_{t+1}|s_t) V_i^{full}(s_{t+1}, x).$$

```
calculateValueFullRiskSharing <- function(
  incomeTransitionMatrix,
  aggregateIncomeGridPoints,
  delta,
  sigma,
  autarkyValueMatrix,
  consumptionOnRelativeParetoWeightGrid,
  numRelativeParetoWeights,
  numHouseholds
  ) {

  # Initial guess is expected utilities under autarky
  householdValueFullRiskSharing <- outer(
    autarkyValueMatrix[, 1], rep(1, numRelativeParetoWeights)
    )
  villageValueFullRiskSharing <- outer(
    autarkyValueMatrix[, 2], rep(1, numRelativeParetoWeights)
    )

  iteration <- 1
  diff <- 1
  while (diff > 1e-10 & iteration < 500) {
    householdValueFullRiskSharingNew <- (
      calculateUtility(consumptionOnRelativeParetoWeightGrid, sigma)
      + delta * incomeTransitionMatrix %*% householdValueFullRiskSharing
    )
    villageValueFullRiskSharingNew <- (
      calculateUtility(
        (aggregateIncomeGridPoints - consumptionOnRelativeParetoWeightGrid) / (numHouseho
        sigma
        )
      + delta * incomeTransitionMatrix %*% villageValueFullRiskSharing
    )

    diff <- max(
      max(abs(householdValueFullRiskSharing - householdValueFullRiskSharingNew)),
      max(abs(villageValueFullRiskSharing - villageValueFullRiskSharingNew))
      )
    householdValueFullRiskSharing <- householdValueFullRiskSharingNew
    villageValueFullRiskSharing <- villageValueFullRiskSharingNew
    iteration <- iteration + 1

  }

  return(list(
    householdValueFullRiskSharing = householdValueFullRiskSharing,
    villageValueFullRiskSharing = villageValueFullRiskSharing
    ))
}
```

### 3.1.6 Calculate values under risk-sharing with limited commitment

Next, I derive the state-dependent intervals of relative Pareto weights and calculate values under the model of limited commitment. To derive the intervals, I use the fact that at the limits of the intervals, the PCs are binding. For instance, to compute the lower limit $\underline{x}^h(s)$, where $h$ indicates $h$'th iteration, the PC of the household is binding:

$$u(c_i^h(s)) + \delta \sum_{s'} \pi(s'|s) V_i^{h-i}(s', \underline{x}^h(s)) = U_i^{aut}(s),$$

where the optimality condition is

$$\frac{u'(c_2^h(s))}{u'(c_1^h(s))} = \underline{x}^h(s).$$

Notice that, once the PC binds, the past history, which is summarized by $x_{t-1}$, does not matter. This property is called "amnesia" (Kocherlakota (1996)) or "forgiveness" (Ligon, Thomas, and Worrall (2002)). Since $\underline{x}^h(s)$ may not be on the grid $q$, linear interpolation is used to compute $V_i^{h-1}(s', \underline{x}^h(s))$.

Similarly, $\overline{x}^h(s)$ is computed using the binding PC of the village:

$$u(c_v^h(s)) + \delta \sum_{s'} \pi(s'|s) V_v^{h-i}(s', \overline{x}^h(s)) = U_v^{aut}(s),$$

where the optimality condition is

$$\frac{u'(c_2^h(s))}{u'(c_1^h(s))} = \overline{x}^h(s).$$

After deriving these limits of intervals,

1. for relative Pareto weights below $\underline{x}^h(s)$, compute consumption of the household based on $\underline{x}^h(s)$ and let its value be $U_i^{aut}$;
2. for relative Pareto weights above $\overline{x}^h(s)$, compute consumption of the household based on $\overline{x}^h(s)$ and let the value of village be $U_v^{aut}$;
3. for other relative Pareto weights, use them to compute consumption of the household and the values of the household and village.

By iterating these steps, we can calculate the value functions and limits of intervals.

```r
interpolateValueFunction <- function(
    relativeParetoWeight,
    relativeParetoWeightsGridPoints,
    valueFunctionMatrix
    ) {
  apply(
    valueFunctionMatrix,
    1,
    function(x) {
      approx(
        relativeParetoWeightsGridPoints,
        x,
        relativeParetoWeight,
        rule = 2
        )$y
    }
    )
}

calculateDiffLCRiskSharingAndAutarky <- function(
    relativeParetoWeight,
    relativeParetoWeightsGridPoints,
    delta,
    sigma,
    aggregateIncome,
    householdValueLCRiskSharing,
    villageValueLCRiskSharing,
    incomeTransitionProbVec,
    householdAutarkyValue,
    villageAutarkyValue,
    numHouseholds
    ) {
```

```r
  householdConsumption <- calculateHouseholdConsumption(
    aggregateIncome,
    relativeParetoWeight,
    numHouseholds,
    sigma
  )

  householdValueLCRiskSharingAtRelativeParetoWeight <- interpolateValueFunction(
    relativeParetoWeight,
    relativeParetoWeightsGridPoints,
    householdValueLCRiskSharing
    )
  villageValueLCRiskSharingAtRelativeParetoWeight <- interpolateValueFunction(
    relativeParetoWeight,
    relativeParetoWeightsGridPoints,
    villageValueLCRiskSharing
    )

  householdDiffLCRiskSharingAndAutarky <- (
    calculateUtility(householdConsumption, sigma)
    + delta * incomeTransitionProbVec %*% householdValueLCRiskSharingAtRelativeParetoWeigh
    - householdAutarkyValue
  ) %>% as.numeric
  villageDiffLCRiskSharingAndAutarky <- (
    calculateUtility((aggregateIncome - householdConsumption) / (numHouseholds - 1), sigma
    + delta * incomeTransitionProbVec %*% villageValueLCRiskSharingAtRelativeParetoWeight
    - villageAutarkyValue
  ) %>% as.numeric

  return(list(
    householdDiffLCRiskSharingAndAutarky = householdDiffLCRiskSharingAndAutarky,
    villageDiffLCRiskSharingAndAutarky = villageDiffLCRiskSharingAndAutarky
  ))
}

getClosestGridIndex <- function(
  point,
  gridPoints
) {
  closestGridIndex <- which.min(
    abs(point - gridPoints)
  )
  if (
    point
    > gridPoints[closestGridIndex]
    ) {
    closestGridIndex <- closestGridIndex + 1
  }
  return(closestGridIndex)
}

calculateValueLCRiskSharing <- function(
  valueFullRiskSharing,
  consumptionOnRelativeParetoWeightGrid,
  aggregateIncomeGridPoints,
  incomeTransitionMatrix,
  autarkyValueMatrix,
  relativeParetoWeightsGridPoints,
  numRelativeParetoWeights,
  delta,
  sigma,
  numIncomeStates,
  numHouseholds,
  iterationLimit,
  diffLimit
) {

  # Initial guess is expected utilities under full risk sharing
  householdValueLCRiskSharing <- valueFullRiskSharing$householdValueFullRiskSharing
  villageValueLCRiskSharing <- valueFullRiskSharing$villageValueFullRiskSharing

  diff <- 1
  iteration <- 1
  while ((diff > diffLimit) && (iteration <= iterationLimit)) {

    # First, ignore enforceability and just update the value functions
    # using the values at the previous iteration
    householdValueLCRiskSharingNew <- (
      calculateUtility(consumptionOnRelativeParetoWeightGrid, sigma)
      + delta * incomeTransitionMatrix %*% householdValueLCRiskSharing
    )
    villageValueLCRiskSharingNew <- (
    calculateUtility(
      (aggregateIncomeGridPoints - consumptionOnRelativeParetoWeightGrid) / (numHouseho
      sigma
      )
    + delta * incomeTransitionMatrix %*% villageValueLCRiskSharing
  )

  # Now check enforceability at each state
  for (incomeStateIndex in seq(1, numIncomeStates)) {
    householdAutarkyValue <- autarkyValueMatrix[incomeStateIndex, 1]
    villageAutarkyValue <- autarkyValueMatrix[incomeStateIndex, 2]

    if (any(householdValueLCRiskSharingNew[incomeStateIndex, ] <= householdAutarkyValue)
      villageValueLCRiskSharingNew[
        incomeStateIndex,
        householdValueLCRiskSharingNew[incomeStateIndex, ] <= householdAutarkyValue
      ] <- villageValueLCRiskSharingNew[
        incomeStateIndex,
        householdValueLCRiskSharingNew[incomeStateIndex, ] <= householdAutarkyValue
      ] %>% min
      householdValueLCRiskSharingNew[
        incomeStateIndex,
        householdValueLCRiskSharingNew[incomeStateIndex, ] <= householdAutarkyValue
      ] <- householdAutarkyValue
    }

    if (any(villageValueLCRiskSharingNew[incomeStateIndex, ] <= villageAutarkyValue)) {
      householdValueLCRiskSharingNew[
        incomeStateIndex,
        villageValueLCRiskSharingNew[incomeStateIndex, ] <= villageAutarkyValue
      ] <- householdValueLCRiskSharingNew[
        incomeStateIndex,
        villageValueLCRiskSharingNew[incomeStateIndex, ] <= villageAutarkyValue
      ] %>% min
      villageValueLCRiskSharingNew[
        incomeStateIndex,
        villageValueLCRiskSharingNew[incomeStateIndex, ] <= villageAutarkyValue
      ] <- villageAutarkyValue
    }
  }
```

```
    diff <- max(
      max(abs(householdValueLCRiskSharingNew - householdValueLCRiskSharing)),
      max(abs(villageValueLCRiskSharingNew - villageValueLCRiskSharing))
    )
    householdValueLCRiskSharing <- householdValueLCRiskSharingNew
    villageValueLCRiskSharing <- villageValueLCRiskSharingNew
    iteration <- iteration + 1
  }

  relativeParetoWeightBounds <- matrix(NA, nrow = numIncomeStates, ncol = 2)

  for (incomeStateIndex in seq(1, numIncomeStates)) {
    aggregateIncome <- aggregateIncomeGridPoints[incomeStateIndex]
    incomeTransitionProbVec <- incomeTransitionMatrix[incomeStateIndex,]
    householdAutarkyValue <- autarkyValueMatrix[incomeStateIndex, 1]
    villageAutarkyValue <- autarkyValueMatrix[incomeStateIndex, 2]

    if (
      calculateDiffLCRiskSharingAndAutarky(
        min(relativeParetoWeightsGridPoints),
        relativeParetoWeightsGridPoints,
        delta,
        sigma,
        aggregateIncome,
        householdValueLCRiskSharing,
        villageValueLCRiskSharing,
        incomeTransitionProbVec,
        householdAutarkyValue,
        villageAutarkyValue,
        numHouseholds
        )$householdDiffLCRiskSharingAndAutarky < 0) {
        relativeParetoWeightLowerBound <- uniroot(
          function(x) {calculateDiffLCRiskSharingAndAutarky(
          x,
          relativeParetoWeightsGridPoints,
          delta,
          sigma,
          aggregateIncome,
          householdValueLCRiskSharing,
          villageValueLCRiskSharing,
          incomeTransitionProbVec,
          householdAutarkyValue,
          villageAutarkyValue,
          numHouseholds
          )$householdDiffLCRiskSharingAndAutarky},
        c(min(relativeParetoWeightsGridPoints), max(relativeParetoWeightsGridPoints)),
        tol = 1e-10,
        maxiter = 300
        )$root
        } else {
          relativeParetoWeightLowerBound <- min(relativeParetoWeightsGridPoints)
        }

    if (
      calculateDiffLCRiskSharingAndAutarky(
        max(relativeParetoWeightsGridPoints),
        relativeParetoWeightsGridPoints,
        delta,
        sigma,
        aggregateIncome,
        householdValueLCRiskSharing,
        villageValueLCRiskSharing,
        incomeTransitionProbVec,
        householdAutarkyValue,
        villageAutarkyValue,
        numHouseholds
        )$villageDiffLCRiskSharingAndAutarky < 0) {
        relativeParetoWeightUpperBound <- uniroot(
          function(x) {calculateDiffLCRiskSharingAndAutarky(
          x,
          relativeParetoWeightsGridPoints,
          delta,
          sigma,
          aggregateIncome,
          householdValueLCRiskSharing,
          villageValueLCRiskSharing,
          incomeTransitionProbVec,
          householdAutarkyValue,
          villageAutarkyValue,
          numHouseholds
          )$villageDiffLCRiskSharingAndAutarky},
        c(min(relativeParetoWeightsGridPoints), max(relativeParetoWeightsGridPoints)),
        tol = 1e-10,
        maxiter = 300
        )$root
        } else {
          relativeParetoWeightUpperBound <- max(relativeParetoWeightsGridPoints)
        }
        relativeParetoWeightBounds[incomeStateIndex, 1] <- relativeParetoWeightLowerBound
        relativeParetoWeightBounds[incomeStateIndex, 2] <- relativeParetoWeightUpperBound
  }

  if (iteration == iterationLimit) {
    print("Reached the maximum limit of iterations!")
  }

  return(list(
    householdValueLCRiskSharing = householdValueLCRiskSharing,
    villageValueLCRiskSharing = villageValueLCRiskSharing,
    relativeParetoWeightBounds = relativeParetoWeightBounds))
}
```

```
solveLCRiskSharing <- function(
    delta,
    sigma,
    punishment,
    householdIncomeTransitionMatrix,
    householdIncomeGridPoints,
    villageIncomeTransitionMatrix,
    villageIncomeGridPoints,
    numIncomeStates,
    numHouseholds,
    numRelativeParetoWeights = 2000,
    iterationLimit = 100,
    diffLimit = 1e-8
) {

  incomeTransitionMatrix <- kronecker(
    villageIncomeTransitionMatrix,
    householdIncomeTransitionMatrix
    )

  incomeGridPointsMatrix <- as.matrix(expand.grid(
    householdIncomeGridPoints, villageIncomeGridPoints
    ))

  aggregateIncomeGridPoints <- (
    incomeGridPointsMatrix[, 1] + incomeGridPointsMatrix[, 2] * (numHouseholds - 1)
  )

  autarkyValueMatrix <- expand.grid(
    calculateAutarkyValue(
      householdIncomeGridPoints,
      sigma,
      delta,
      punishment,
      householdIncomeTransitionMatrix
    ),
    calculateAutarkyValue(
      villageIncomeGridPoints,
      sigma,
      delta,
      punishment,
      villageIncomeTransitionMatrix
    )
  )

  relativeParetoWeightsGridPoints <- getRelativeParetoWeightsGridPoints(
      sigma,
      punishment,
      householdIncomeGridPoints,
      villageIncomeGridPoints,
      numRelativeParetoWeights
      )

  consumptionOnRelativeParetoWeightGrid <- matrix(
    NA, nrow = numIncomeStates, ncol = numRelativeParetoWeights
    )
  for (incomeStateIndex in seq_along(aggregateIncomeGridPoints)) {
    for (relativeParetoWeightIndex in seq_along(relativeParetoWeightsGridPoints)) {
      consumptionOnRelativeParetoWeightGrid[
        incomeStateIndex,
        relativeParetoWeightIndex
        ] <- calculateHouseholdConsumption(
          aggregateIncomeGridPoints[incomeStateIndex],
          relativeParetoWeightsGridPoints[relativeParetoWeightIndex],
          numHouseholds,
          sigma
        )
      }
    }

  valueFullRiskSharing <- calculateValueFullRiskSharing(
    incomeTransitionMatrix,
    aggregateIncomeGridPoints,
    delta,
    sigma,
    autarkyValueMatrix,
    consumptionOnRelativeParetoWeightGrid,
    numRelativeParetoWeights,
    numHouseholds
    )

  valueLCRiskSharing <- calculateValueLCRiskSharing(
    valueFullRiskSharing,
    consumptionOnRelativeParetoWeightGrid,
    aggregateIncomeGridPoints,
    incomeTransitionMatrix,
    autarkyValueMatrix,
    relativeParetoWeightsGridPoints,
    numRelativeParetoWeights,
    delta,
    sigma,
    numIncomeStates,
    numHouseholds,
    iterationLimit,
    diffLimit
  )

  return(valueLCRiskSharing)
}
```

```
solveLCRiskSharingByVillage <- function(
    village,
    delta,
    sigma,
    punishment,
    householdAR1EstimationResult,
    villageAR1EstimationResult,
    numIncomeStates,
    numRelativeParetoWeights = 2000,
    iterationLimit = 100,
    diffLimit = 1e-8
) {

  numHouseholds <- householdAR1EstimationResult[[village]]$numHouseholds

  relativeParetoWeightBoundsArray <- array(
    NA,
    c(2, 2, numIncomeStates, 2)
  )

  for (meanClass in seq(1, 2)) {
    for (CVClass in seq(1, 2)) {
      householdIncomeTransitionMatrix <- (
        householdAR1EstimationResult[[village]]$transitionMatrixArray[meanClass, CVClass,
      )
      householdIncomeGridPoints <- (
        householdAR1EstimationResult[[village]]$gridPointsArray[meanClass, CVClass,]
      )
      villageIncomeTransitionMatrix <- (
        villageAR1EstimationResult[[village]]$transitionMatrix
      )
      villageIncomeGridPoints <- (
        villageAR1EstimationResult[[village]]$gridPoints
      )

      relativeParetoWeightBoundsArray[meanClass, CVClass, ,] <- solveLCRiskSharing(
        delta,
        sigma,
        punishment,
        householdIncomeTransitionMatrix,
        householdIncomeGridPoints,
        villageIncomeTransitionMatrix,
        villageIncomeGridPoints,
        numIncomeStates,
        numHouseholds,
        numRelativeParetoWeights,
        iterationLimit,
        diffLimit
      )$relativeParetoWeightBounds
    }
  }
  return(relativeParetoWeightBoundsArray)
}
```

```
save.image("IntermediateData/lc_hom_model_functions.RData")
```

### 3.1.7 Sanity test: replication of Figure 1 in Ligon, Thomas, and Worrall (2002)

For the sanity test of this function, I use it to replicate the Figure 1 in Ligon, Thomas, and Worrall (2002). Here I use the parameter values in the original paper. I choose the income process $(y_l, y_h) = (2/3, 4/3)$ and $(p_l, p_h) = (0.1, 0.9)$ for both households so that the mean is 1 and the ratio $y_l / y_h$ is 1/2 as in the paper. Also, the penalty under autarky is absent as in the original numerical exercise. Finally, I assume the CRRA utility functions:

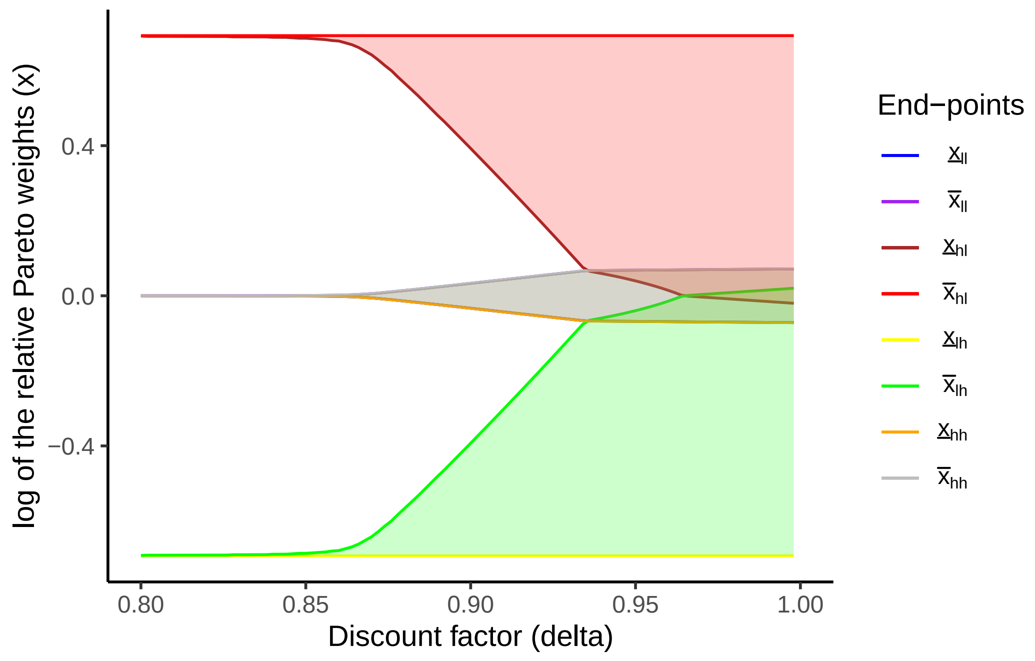
$$u(c_{it}) = \frac{c_{it}^{1-\sigma} - 1}{1 - \sigma}.$$

```
sigmaLTW <- 1.0
punishmentLTW <- 0.0

incomeTransitionMatrixLTW <- matrix(rep(c(0.1, 0.9), 2), nrow = 2, byrow = TRUE)
incomeGridPointsLTW <- c(2/3, 4/3)
numIncomeStatesLTW <- length(incomeGridPointsLTW) *  length(incomeGridPointsLTW)
numHouseholdsLTW <- 2

deltaVec <- seq(0.8, 0.999, by = 0.002)
```

```
LCRiskSharingResultLTW <- map(
  deltaVec,
  ~ solveLCRiskSharing(
    .,
    sigmaLTW,
    punishmentLTW,
    incomeTransitionMatrixLTW,
    incomeGridPointsLTW,
    incomeTransitionMatrixLTW,
    incomeGridPointsLTW,
    numIncomeStatesLTW,
    numHouseholdsLTW,
    numRelativeParetoWeights = 10000,
    iterationLimit = 1000,
    diffLimit = 1e-8
    )
)
```

```
createLCFigure <- function(
    deltaVec,
    incomeGridPoints,
    relativeParetoWeightBoundsArray
) {

  LCFigure <- ggplot() +
    geom_line(aes(deltaVec, log(relativeParetoWeightBoundsArray[1,1,]), color = "a")) +
    geom_line(aes(deltaVec, log(relativeParetoWeightBoundsArray[1,2,]), color = "b")) +
    geom_line(aes(deltaVec, log(relativeParetoWeightBoundsArray[2,1,]), color = "c")) +
    geom_line(aes(deltaVec, log(relativeParetoWeightBoundsArray[2,2,]), color = "d")) +
    geom_line(aes(deltaVec, log(relativeParetoWeightBoundsArray[3,1,]), color = "e")) +
    geom_line(aes(deltaVec, log(relativeParetoWeightBoundsArray[3,2,]), color = "f")) +
    geom_line(aes(deltaVec, log(relativeParetoWeightBoundsArray[4,1,]), color = "g")) +
    geom_line(aes(deltaVec, log(relativeParetoWeightBoundsArray[4,2,]), color = "h")) +
    coord_cartesian(
      xlim = c(0.8, 1.0),
      ylim = c(
        log(incomeGridPoints[1] / incomeGridPoints[2]),
        log(incomeGridPoints[2] / incomeGridPoints[1])
        )
      ) +
    geom_ribbon(aes(x = deltaVec,
                    ymin = log(relativeParetoWeightBoundsArray[1,1,]),
                    ymax = log(relativeParetoWeightBoundsArray[1,2,])),
                    fill = "blue", alpha = 0.2) +
    geom_ribbon(aes(x = deltaVec,
                    ymin = log(relativeParetoWeightBoundsArray[2,1,]),
                    ymax = log(relativeParetoWeightBoundsArray[2,2,])),
                   fill = "red", alpha = 0.2) +
    geom_ribbon(aes(x = deltaVec,
                   ymin = log(relativeParetoWeightBoundsArray[3,1,]),
                   ymax = log(relativeParetoWeightBoundsArray[3,2,])),
                   fill = "green", alpha = 0.2) +
    geom_ribbon(aes(x = deltaVec,
                   ymin = log(relativeParetoWeightBoundsArray[4,1,]),
                   ymax = log(relativeParetoWeightBoundsArray[4,2,])),
                   fill = "yellow", alpha = 0.2) +
    scale_color_manual(
      name = "End-points",
      values = c(
        "blue",
        "purple",
        "brown",
        "red",
        "yellow",
        "green",
        "orange",
        "gray"
        ),
      labels = unname(TeX(c(
        "$\\underline{x}_{ll}$",
        "$\\bar{x}_{ll}$",
        "$\\underline{x}_{hl}$",
        "$\\bar{x}_{hl}$",
        "$\\underline{x}_{lh}$",
        "$\\bar{x}_{lh}$",
        "$\\underline{x}_{hh}$",
        "$\\bar{x}_{hh}$"
        )))
      ) +
    xlab("Discount factor (delta)") +
    ylab("log of the relative Pareto weights (x)") +
    theme_classic()

  return(LCFigure)
}
```

```
relativeParetoWeightBoundsArrayLTW = array(
  NA,
  dim = c(numIncomeStatesLTW, 2, length(deltaVec))
  )

for (deltaIndex in seq_along(deltaVec)) {
  relativeParetoWeightBoundsArrayLTW[,,deltaIndex] <- (
    LCRiskSharingResultLTW[[deltaIndex]]$relativeParetoWeightBounds
  )
}

LCFigure <- createLCFigure(
    deltaVec,
    incomeGridPointsLTW,
    relativeParetoWeightBoundsArrayLTW
)

LCFigure
```



```
saveRDS(
  relativeParetoWeightBoundsArrayLTW,
  file.path('IntermediateData/relativeParetoWeightBoundsArrayLTW.rds')
)
saveRDS(
```

```r
  createLCFigure,
  file.path('IntermediateData/createLCFigure.rds')
)
saveRDS(
  LCFigure,
  file.path('IntermediateData/LCFigure.rds')
)
```

# 4 Interlude: Risk-sharing with static limited commitment

As opposed to the limited commitment model where the transfers depend on the past history, Coate and Ravallion (1993) consider a repeated game for a limited commitment model, where transfers are determined by the current state. Whereas the former is called a *dynamic* limited commitment model, the latter is called a *static* limited commitment model.

A static limited commitment model follows a similar update rule of relative Pareto weights, with one difference that, when participation constraints are not binding, the current relative Pareto weight becomes *the initial relative Pareto weight*. For more discussion, see Ligon, Thomas, and Worrall (2002).

Using this update rule, the static limited commitment model can be numerically solved.

```r
pacman::p_load(
  tidyverse,
  kableExtra,
  ggrepel,
  latex2exp
)
```

### 4.0.1 Load data and functions

```r
load('IntermediateData/lc_hom_model_functions.RData')
```

```r
relativeParetoWeightBoundsArrayLTW <- readRDS(
  file.path('IntermediateData/relativeParetoWeightBoundsArrayLTW.rds')
)
createLCFigure <- readRDS(
  file.path('IntermediateData/createLCFigure.rds')
)
LCFigure <- readRDS(
  file.path('IntermediateData/LCFigure.rds')
)
```

```r
calculateDiffStaticLCRiskSharingAndAutarky <- function(
    relativeParetoWeight,
    delta,
    sigma,
    aggregateIncome,
    householdValueLCRiskSharing,
    villageValueLCRiskSharing,
    incomeTransitionProbVec,
    householdAutarkyValue,
    villageAutarkyValue,
    numHouseholds
    ) {

  householdConsumption <- calculateHouseholdConsumption(
    aggregateIncome,
    relativeParetoWeight,
    numHouseholds,
    sigma
  )

  householdDiffLCRiskSharingAndAutarky <- (
    calculateUtility(householdConsumption, sigma)
    + delta * incomeTransitionProbVec %*% householdValueLCRiskSharing
    - householdAutarkyValue
  ) %>% as.numeric
  villageDiffLCRiskSharingAndAutarky <- (
    calculateUtility((aggregateIncome - householdConsumption) / (numHouseholds - 1), sigma
    + delta * incomeTransitionProbVec %*% villageValueLCRiskSharing
    - villageAutarkyValue
  ) %>% as.numeric

  return(list(
    householdDiffLCRiskSharingAndAutarky = householdDiffLCRiskSharingAndAutarky,
    villageDiffLCRiskSharingAndAutarky = villageDiffLCRiskSharingAndAutarky
  ))
}
```

```r
calculateValueStaticLCRiskSharing <- function(
  valueFullRiskSharing,
  consumptionOnRelativeParetoWeightGrid,
  aggregateIncomeGridPoints,
  incomeTransitionMatrix,
  autarkyValueMatrix,
  relativeParetoWeightsGridPoints,
  numRelativeParetoWeights,
  delta,
  sigma,
  numIncomeStates,
  numHouseholds,
  iterationLimit,
  diffLimit,
  initialRelativeParetoweight
) {

  # Initial guess is expected utilities under full risk sharing
  householdValueLCRiskSharing <- interpolateValueFunction(
      initialRelativeParetoweight,
      relativeParetoWeightsGridPoints,
      valueFullRiskSharing$householdValueFullRiskSharing
      )
  villageValueLCRiskSharing <- interpolateValueFunction(
      initialRelativeParetoweight,
      relativeParetoWeightsGridPoints,
      valueFullRiskSharing$villageValueFullRiskSharing
      )

  householdConsumptionAtInitialRelativeParetoWeight <- calculateHouseholdConsumption(
      aggregateIncomeGridPoints,
      initialRelativeParetoweight,
      numHouseholds,
      sigma
    )

  diff <- 1
  iteration <- 1
  while ((diff > diffLimit) && (iteration <= iterationLimit)) {

    # First, ignore enforceability and just update the value functions
    # using the values at the previous iteration
    householdValueLCRiskSharingNew <- (
      calculateUtility(householdConsumptionAtInitialRelativeParetoWeight, sigma)
      + delta * incomeTransitionMatrix %*% householdValueLCRiskSharing
    )
    villageValueLCRiskSharingNew <- (
      calculateUtility(
        (aggregateIncomeGridPoints - householdConsumptionAtInitialRelativeParetoWeight) /
        sigma
        )
      + delta * incomeTransitionMatrix %*% villageValueLCRiskSharing
    )

    relativeParetoWeightBounds <- matrix(NA, nrow = numIncomeStates, ncol = 2)

    # Now check enforceability at each state
    for (incomeStateIndex in seq(1, numIncomeStates)) {
      aggregateIncome <- aggregateIncomeGridPoints[incomeStateIndex]
      incomeTransitionProbVec <- incomeTransitionMatrix[incomeStateIndex,]
      householdAutarkyValue <- autarkyValueMatrix[incomeStateIndex, 1]
      villageAutarkyValue <- autarkyValueMatrix[incomeStateIndex, 2]

      if (
        calculateDiffStaticLCRiskSharingAndAutarky(
          min(relativeParetoWeightsGridPoints),
          delta,
          sigma,
          aggregateIncome,
          householdValueLCRiskSharing,
          villageValueLCRiskSharing,
          incomeTransitionProbVec,
          householdAutarkyValue,
          villageAutarkyValue,
          numHouseholds
          )$householdDiffLCRiskSharingAndAutarky < 0) {
          relativeParetoWeightLowerBound <- tryCatch({
            uniroot(
              function(x) {calculateDiffStaticLCRiskSharingAndAutarky(
              x,
              delta,
              sigma,
              aggregateIncome,
              householdValueLCRiskSharing,
              villageValueLCRiskSharing,
              incomeTransitionProbVec,
              householdAutarkyValue,
              villageAutarkyValue,
              numHouseholds
              )$householdDiffLCRiskSharingAndAutarky},
            c(min(relativeParetoWeightsGridPoints), max(relativeParetoWeightsGridPoints)),
            tol = 1e-10,
            maxiter = 300
            )$root
            }, error = function(e) {
              relativeParetoWeightLowerBound <- max(relativeParetoWeightsGridPoints)
            })
          } else {
            relativeParetoWeightLowerBound <- min(relativeParetoWeightsGridPoints)
          }

      if (
        calculateDiffStaticLCRiskSharingAndAutarky(
          max(relativeParetoWeightsGridPoints),
          delta,
          sigma,
          aggregateIncome,
          householdValueLCRiskSharing,
          villageValueLCRiskSharing,
          incomeTransitionProbVec,
          householdAutarkyValue,
          villageAutarkyValue,
          numHouseholds
          )$villageDiffLCRiskSharingAndAutarky < 0) {
          relativeParetoWeightUpperBound <- tryCatch({
            uniroot(
              function(x) {calculateDiffStaticLCRiskSharingAndAutarky(
              x,
              delta,
              sigma,
              aggregateIncome,
              householdValueLCRiskSharing,
              villageValueLCRiskSharing,
              incomeTransitionProbVec,
              householdAutarkyValue,
              villageAutarkyValue,
              numHouseholds
              )$villageDiffLCRiskSharingAndAutarky},
            c(min(relativeParetoWeightsGridPoints), max(relativeParetoWeightsGridPoints)),
            tol = 1e-10,
            maxiter = 300
            )$root
            }, error = function(e) {
              relativeParetoWeightUpperBound <- min(relativeParetoWeightsGridPoints)
            })
          } else {
            relativeParetoWeightUpperBound <- max(relativeParetoWeightsGridPoints)
          }

      relativeParetoWeightBounds[incomeStateIndex, 1] <- relativeParetoWeightLowerBound
      relativeParetoWeightBounds[incomeStateIndex, 2] <- relativeParetoWeightUpperBound

      householdConsumptionAtLowerBound <- calculateHouseholdConsumption(
          aggregateIncome,
          relativeParetoWeightLowerBound,
          numHouseholds,
          sigma
        )
      householdConsumptionAtUpperBound <- calculateHouseholdConsumption(
          aggregateIncome,
          relativeParetoWeightUpperBound,
          numHouseholds,
          sigma
        )

      if (householdValueLCRiskSharingNew[incomeStateIndex] < householdAutarkyValue) {
        householdValueLCRiskSharingNew[incomeStateIndex] <- householdAutarkyValue
        villageValueLCRiskSharingNew[incomeStateIndex] <- max(
          calculateUtility(
            (aggregateIncome - householdConsumptionAtLowerBound) / (numHouseholds - 1),
            sigma
            )
          + delta * incomeTransitionProbVec %*% villageValueLCRiskSharing,
          villageAutarkyValue
        )
      }
      if (villageValueLCRiskSharingNew[incomeStateIndex] < villageAutarkyValue) {
        villageValueLCRiskSharingNew[incomeStateIndex] <- villageAutarkyValue
```

```
        householdValueLCRiskSharingNew[incomeStateIndex] <- max(
          calculateUtility(
            householdConsumptionAtUpperBound,
            sigma
            )
          + delta * incomeTransitionProbVec %*% householdValueLCRiskSharing,
          householdAutarkyValue
        )
      }

    }

    diff <- max(
      max(abs(householdValueLCRiskSharingNew - householdValueLCRiskSharing)),
      max(abs(villageValueLCRiskSharingNew - villageValueLCRiskSharing))
    )
    householdValueLCRiskSharing <- householdValueLCRiskSharingNew
    villageValueLCRiskSharing <- villageValueLCRiskSharingNew
    iteration <- iteration + 1
  }


  if (iteration == iterationLimit) {
    print("Reached the maximum limit of iterations!")
  }

  return(list(
    householdValueLCRiskSharing = householdValueLCRiskSharing,
    villageValueLCRiskSharing = villageValueLCRiskSharing,
    relativeParetoWeightBounds = relativeParetoWeightBounds))
}
```

```
solveStaticLCRiskSharing <- function(
    delta,
    sigma,
    punishment,
    householdIncomeTransitionMatrix,
    householdIncomeGridPoints,
    villageIncomeTransitionMatrix,
    villageIncomeGridPoints,
    numIncomeStates,
    numHouseholds,
    initialRelativeParetoWeight,
    numRelativeParetoWeights = 2000,
    iterationLimit = 100,
    diffLimit = 1e-8
) {

  incomeTransitionMatrix <- kronecker(
    villageIncomeTransitionMatrix,
    householdIncomeTransitionMatrix
    )

  incomeGridPointsMatrix <- as.matrix(expand.grid(
    householdIncomeGridPoints, villageIncomeGridPoints
    ))

  aggregateIncomeGridPoints <- (
    incomeGridPointsMatrix[, 1] + incomeGridPointsMatrix[, 2] * (numHouseholds - 1)
  )

  autarkyValueMatrix <- expand.grid(
    calculateAutarkyValue(
      householdIncomeGridPoints,
      sigma,
      delta,
      punishment,
      householdIncomeTransitionMatrix
    ),
    calculateAutarkyValue(
      villageIncomeGridPoints,
      sigma,
      delta,
      punishment,
      villageIncomeTransitionMatrix
    )
  )

  relativeParetoWeightsGridPoints <- getRelativeParetoWeightsGridPoints(
      sigma,
      punishment,
      householdIncomeGridPoints,
      villageIncomeGridPoints,
      numRelativeParetoWeights
      )

  consumptionOnRelativeParetoWeightGrid <- matrix(
    NA, nrow = numIncomeStates, ncol = numRelativeParetoWeights
    )
  for (incomeStateIndex in seq_along(aggregateIncomeGridPoints)) {
    for (relativeParetoWeightIndex in seq_along(relativeParetoWeightsGridPoints)) {
      consumptionOnRelativeParetoWeightGrid[
        incomeStateIndex,
        relativeParetoWeightIndex
        ] <- calculateHouseholdConsumption(
          aggregateIncomeGridPoints[incomeStateIndex],
          relativeParetoWeightsGridPoints[relativeParetoWeightIndex],
          numHouseholds,
          sigma
        )
      }
    }

  valueFullRiskSharing <- calculateValueFullRiskSharing(
    incomeTransitionMatrix,
    aggregateIncomeGridPoints,
    delta,
    sigma,
    autarkyValueMatrix,
    consumptionOnRelativeParetoWeightGrid,
    numRelativeParetoWeights,
    numHouseholds
    )

  valueStaticLCRiskSharing <- calculateValueStaticLCRiskSharing(
    valueFullRiskSharing,
    consumptionOnRelativeParetoWeightGrid,
    aggregateIncomeGridPoints,
    incomeTransitionMatrix,
    autarkyValueMatrix,
    relativeParetoWeightsGridPoints,
    numRelativeParetoWeights,
    delta,
    sigma,
    numIncomeStates,
    numHouseholds,
    iterationLimit,
    diffLimit,
    initialRelativeParetoWeight
  )

  return(valueStaticLCRiskSharing)
}
```

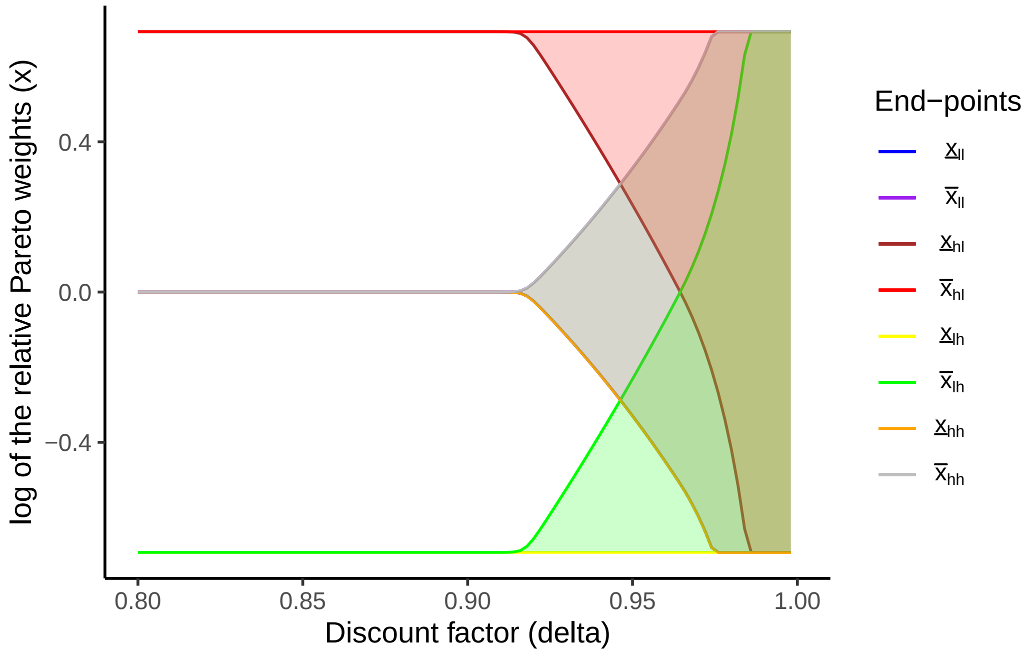
### 4.0.2 Create a figure of relative Pareto weight bounds

```
sigmaLTW <- 1.0
punishmentLTW <- 0.0

incomeTransitionMatrixLTW <- matrix(rep(c(0.1, 0.9), 2), nrow = 2, byrow = TRUE)
incomeGridPointsLTW <- c(2/3, 4/3)
numIncomeStatesLTW <- length(incomeGridPointsLTW) *  length(incomeGridPointsLTW)
numHouseholdsLTW <- 2

incomeTransitionMatrixLTW <- kronecker(
  incomeTransitionMatrixLTW,
  incomeTransitionMatrixLTW
  )
incomeGridPointsMatrixLTW <- as.matrix(expand.grid(
  incomeGridPointsLTW, incomeGridPointsLTW
  ))
aggregateIncomeGridPointsLTW <- (
  incomeGridPointsMatrixLTW[, 1] + incomeGridPointsMatrixLTW[, 2] * (numHouseholdsLTW - 1)
)

deltaVec <- seq(0.8, 0.999, by = 0.002)

initialRelativeParetoWeight <- 1
```

```
relativeParetoWeightBoundsArrayLTWStatic = array(
  NA,
  dim = c(numIncomeStatesLTW, 2, length(deltaVec))
  )
```

```
for (deltaIndex in seq_along(deltaVec)) {
  relativeParetoWeightBoundsArrayLTWStatic[,,deltaIndex] <- (
    staticLCRiskSharingResultLTW[[deltaIndex]]$relativeParetoWeightBounds
  )
}
```

```
staticLCFigure <- createLCFigure(
    deltaVec,
    incomeGridPointsLTW,
    relativeParetoWeightBoundsArrayLTWStatic
)
staticLCFigure
```



## 4.1 Comparison with a dynamic limited commitment model

Here, I compare the consumption volatility under static vs dynamic limited commitment. For illustartion, I generate 10 random incomes and explore how households transfer and consume in each period.

```r
numIncomeSimulations <- 10

set.seed(35)

# Sequence of income shocks
incomeRealization <- c(1, 2, 3, 4)
incomeRealizationLabel <- c("Low, Low", "High, Low", "Low, High", "High, High")
incomeSeq <- sample(
  incomeRealization,
  size = numIncomeSimulations,
  replace = TRUE,
  prob = incomeTransitionMatrixLTW[1,]
  )

householdIncomeRealization <- incomeGridPointsMatrixLTW[incomeSeq, 1]
villageIncomeRealization <- incomeGridPointsMatrixLTW[incomeSeq, 2]
(incomeRealizationVec <- incomeRealizationLabel[incomeSeq])
```

```
[1] "Low, High"  "High, High" "High, High" "High, Low"  "High, High"
[6] "High, High" "High, High" "High, High" "Low, Low"   "High, Low"
```

```r
createPlotTable <- function(
    delta,
    sigma,
    LCFigure,
    relativeParetoWeightBoundsMatrix,
    numIncomeSimulations,
    incomeSeq,
    incomeGridPointsMatrix,
    aggregateIncomeGridPoints,
    numHouseholds,
    incomeRealizationVec,
    householdIncomeRealization,
    villageIncomeRealization,
    dynamic = TRUE
    ) {

  # Vector of relative Pareto weights
  relativeParetoWeightVec <- rep(NA, numIncomeSimulations + 1)
  relativeParetoWeightVec[1] <- 1
```

```
# Relative Pareto weights on the history of income realizations
if (dynamic == TRUE) {
  for (timeIndex in seq(numIncomeSimulations)) {
    relativeParetoWeightBounds <- relativeParetoWeightBoundsMatrix[incomeSeq[timeIndex],
    if (relativeParetoWeightVec[timeIndex] < relativeParetoWeightBounds[1]) {
      relativeParetoWeightVec[timeIndex + 1] <- relativeParetoWeightBounds[1]
    } else if (relativeParetoWeightVec[timeIndex] > relativeParetoWeightBounds[2]) {
      relativeParetoWeightVec[timeIndex + 1] <- relativeParetoWeightBounds[2]
    } else {
      relativeParetoWeightVec[timeIndex + 1] <- relativeParetoWeightVec[timeIndex]
    }
  }
} else {
  for (timeIndex in seq(numIncomeSimulations)) {
    relativeParetoWeightBounds <- relativeParetoWeightBoundsMatrix[incomeSeq[timeIndex],
    if (relativeParetoWeightVec[1] < relativeParetoWeightBounds[1]) {
      relativeParetoWeightVec[timeIndex + 1] <- relativeParetoWeightBounds[1]
    } else if (relativeParetoWeightVec[1] > relativeParetoWeightBounds[2]) {
      relativeParetoWeightVec[timeIndex + 1] <- relativeParetoWeightBounds[2]
    } else {
      relativeParetoWeightVec[timeIndex + 1] <- relativeParetoWeightVec[1]
    }
  }
}

# Consumption and transfers, calculated based on x_vec
householdConsVec <- rep(NA, numIncomeSimulations + 1)
villageConsVec <- rep(NA, numIncomeSimulations + 1)
transferFromHHtoVilVec <- rep(NA, numIncomeSimulations + 1)
for (timeIndex in seq(2, numIncomeSimulations + 1)) {
  householdConsVec[timeIndex] <- calculateHouseholdConsumption(
    aggregateIncomeGridPoints[incomeSeq[timeIndex - 1]],
    relativeParetoWeightVec[timeIndex],
    numHouseholds,
    sigma
  )
  villageConsVec[timeIndex] <- (
    aggregateIncomeGridPoints[incomeSeq[timeIndex - 1]] - householdConsVec[timeIndex]
  )
  transferFromHHtoVilVec[timeIndex] <- (
    incomeGridPointsMatrix[incomeSeq[timeIndex - 1], 1] - householdConsVec[timeIndex]
  )
}

# Output table
tableOutput <- tibble(
  `Period` = seq(0, numIncomeSimulations),
  `ln(x)` = log(relativeParetoWeightVec),
  `Income shocks` = c(NA, incomeRealizationVec),
  `Net transfer (1 -> 2)` = transferFromHHtoVilVec,
  `Consumption (1)` = householdConsVec,
  `Consumption (2)` = villageConsVec,
  `Income (1)` = c(NA, householdIncomeRealization),
  `Income (2)` = c(NA, villageIncomeRealization)
)

# Output figure
plotOutput <- LCFigure +
  geom_vline(xintercept = delta) +
  geom_point(aes(rep(delta, numIncomeSimulations + 1), log(relativeParetoWeightVec))) +
  geom_label_repel(
    aes(rep(delta, numIncomeSimulations + 1), log(relativeParetoWeightVec)), label = seq
    box.padding = 0.35, point.padding = 0.5,
    max.overlaps = Inf
  )

# Table of mean and SD of consumption and income
summaryTableOutput <- tableOutput %>%
  select(
    c(
      "Consumption (1)",
      "Consumption (2)",
      "Income (1)",
      "Income (2)"
    )
  ) %>%
  summarise_all(
    list(
      mean = function(x) mean(x, na.rm = TRUE),
      sd = function(x) sd(x, na.rm = TRUE)
      )
    ) %>%
    pivot_longer(everything(), names_to = "variable", values_to = "value") %>%
    extract(variable, c("variable", "stat"),
            regex = "(.*)(_mean|_sd)") %>%
    pivot_wider(names_from = "stat", values_from = "value")
  colnames(summaryTableOutput) <- c("var", "Mean", "SD")

  return(list(tableOutput, plotOutput, summaryTableOutput))
}
```

I use $\delta = 0.95$ since both static and dynamic limited commitment models achieve partial risk sharing with this time discount factor.

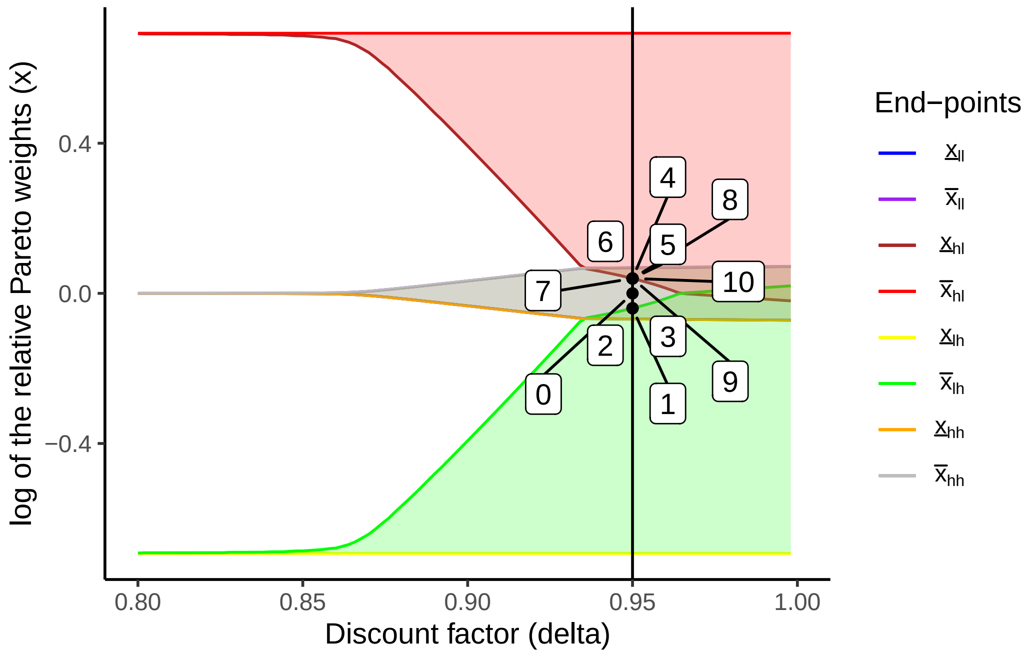
```
deltaIndex <- 76

dynamicLCResults <- createPlotTable(
    deltaVec[deltaIndex],
    sigmaLTW,
    LCFigure,
    relativeParetoWeightBoundsArrayLTW[, , deltaIndex],
    numIncomeSimulations,
    incomeSeq,
    incomeGridPointsMatrixLTW,
    aggregateIncomeGridPointsLTW,
    numHouseholdsLTW,
    incomeRealizationVec,
    householdIncomeRealization,
    villageIncomeRealization,
    dynamic = TRUE
    )

staticLCResults <- createPlotTable(
    deltaVec[deltaIndex],
    sigmaLTW,
    staticLCFigure,
    relativeParetoWeightBoundsArrayLTWStatic[, , deltaIndex],
    numIncomeSimulations,
    incomeSeq,
    incomeGridPointsMatrixLTW,
    aggregateIncomeGridPointsLTW,
    numHouseholdsLTW,
    incomeRealizationVec,
    householdIncomeRealization,
    villageIncomeRealization,
    dynamic = FALSE
    )
```

### 4.1.1 Results with dynamic limited commitment

From the figure and table below, we can see that the consumption and transfers depend on the past history. For instance, in periods 2 and 5, although the “High, High” income state is realized in both periods, transfers are from HH1 to HH2 in the period 2 and from HH2 to HH1 in the period 5. This is because, in period 1, HH1 experiences a negative shock, and hence it “pays back” to HH2 in the following period. The opposite happend in periods 4 and 5. This represents the “state-contingent loans” as in Udry (1994) or the “quasi-credit” as in Fafchamps (1999).

Also, in the period 4, HH2 experiences a bad shock and HH1’s participation constraint binds. At this point, they “forget” the past history: in other words, transfers and consumption do not depend on the past history any more. This is the “amnesia” property as in Kocherlakota (1996).

```
dynamicLCResults[[2]]
```



| Period | ln(x) | Income shocks | Net transfer (1 -> 2) | Consumption (1) | Consumption (2) | Income (1) |
|---:|---:|:---|---:|---:|---:|---:|
| 0 | 0.00 | NA | NA | NA | NA | NA |
| 1 | -0.04 | Low, High | -0.313 | 0.980 | 1.020 | 0.667 |
| 2 | -0.04 | High, High | 0.026 | 1.307 | 1.360 | 1.333 |
| 3 | -0.04 | High, High | 0.026 | 1.307 | 1.360 | 1.333 |
| 4 | 0.04 | High, Low | 0.313 | 1.020 | 0.980 | 1.333 |
| 5 | 0.04 | High, High | -0.026 | 1.360 | 1.307 | 1.333 |
| 6 | 0.04 | High, High | -0.026 | 1.360 | 1.307 | 1.333 |
| 7 | 0.04 | High, High | -0.026 | 1.360 | 1.307 | 1.333 |
| 8 | 0.04 | High, High | -0.026 | 1.360 | 1.307 | 1.333 |
| 9 | 0.04 | Low, Low | -0.013 | 0.680 | 0.653 | 0.667 |
| 10 | 0.04 | High, Low | 0.313 | 1.020 | 0.980 | 1.333 |

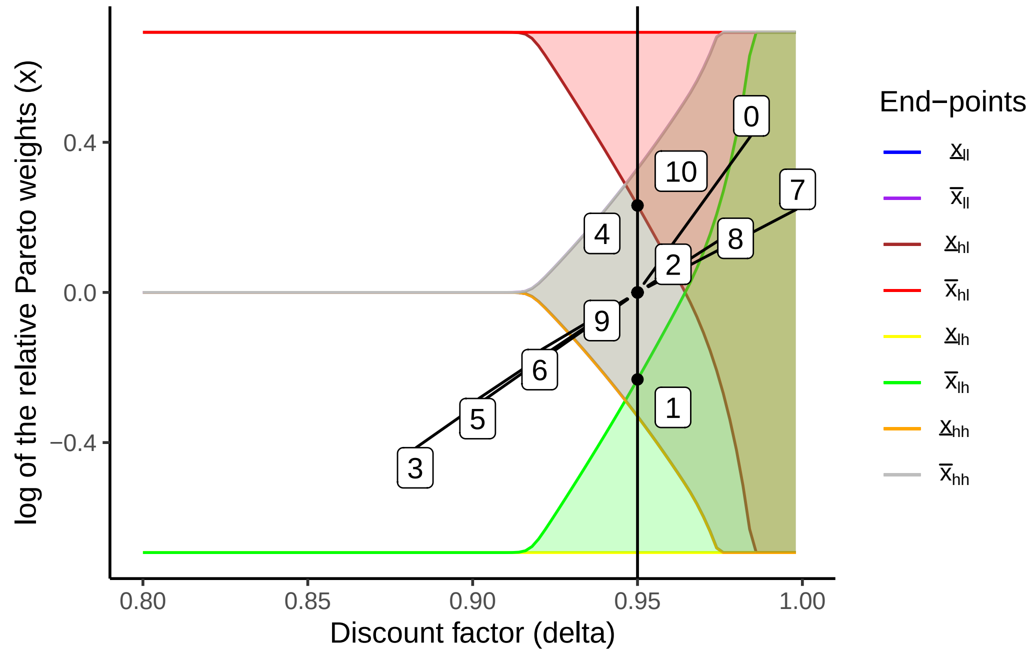
```r
tableDynamic <- dynamicLCResults[[1]]
tableDynamic %>%
  kbl(digits = 3) %>%
  kable_classic()
```

### 4.1.2 Results with dynamic limited commitment

In the static model, unlike in the model with a dynamic limited commitment, transfers and consumption do not depend on the past history, as shown in the figure and table below.

```r
staticLCResults[[2]]
```

| Period | ln(x) | Income shocks | Net transfer (1 -> 2) | Consumption (1) | Consumption (2) | Income (1) |
|---:|---:|:---|---:|---:|---:|---:|
| 0 | 0.000 | NA | NA | NA | NA | NA |
| 1 | -0.232 | Low, High | -0.218 | 0.885 | 1.115 | 0.667 |
| 2 | 0.000 | High, High | 0.000 | 1.333 | 1.333 | 1.333 |
| 3 | 0.000 | High, High | 0.000 | 1.333 | 1.333 | 1.333 |
| 4 | 0.232 | High, Low | 0.218 | 1.115 | 0.885 | 1.333 |
| 5 | 0.000 | High, High | 0.000 | 1.333 | 1.333 | 1.333 |
| 6 | 0.000 | High, High | 0.000 | 1.333 | 1.333 | 1.333 |
| 7 | 0.000 | High, High | 0.000 | 1.333 | 1.333 | 1.333 |
| 8 | 0.000 | High, High | 0.000 | 1.333 | 1.333 | 1.333 |
| 9 | 0.000 | Low, Low | 0.000 | 0.667 | 0.667 | 0.667 |
| 10 | 0.232 | High, Low | 0.218 | 1.115 | 0.885 | 1.333 |



```r
tableStatic <- staticLCResults[[1]]
tableStatic %>%
  kbl(digits = 3) %>%
  kable_classic()
```

|  | Mean (DLC) | SD (DLC) | Mean (SLC) | SD (SLC) |
|---|---|---|---|---|
| Consumption (1) | 1.175 | 0.236 | 1.178 | 0.236 |
| Consumption (2) | 1.158 | 0.237 | 1.155 | 0.253 |
| Income (1) | 1.200 | 0.281 | 1.200 | 0.281 |
| Income (2) | 1.133 | 0.322 | 1.133 | 0.322 |

### 4.1.3 Consumption volatility in the two models

The table below shows the income and consumption summary statistics. Since the income streams are identical in the two simulations, the statistics of the income are the same. In the dynamic limited commitment (DLC) model, the consumption volatility is smaller than that in the static limited commitment model (SLC). This demonstrates that, with dynamic limited commitment, households are better secured from income risks.

```r
tableCompare <- left_join(dynamicLCResults[[3]], staticLCResults[[3]], by = "var")
colnames(tableCompare) <- c("", "Mean (DLC)", "SD (DLC)", "Mean (SLC)", "SD (SLC)")
rownames(tableCompare) <- c(
  "Consumption (HH 1)",
  "Consumption (HH 2)",
  "Income (HH 1)",
  "Income (HH 2)"
)
```

```
Warning: Setting row names on a tibble is deprecated.
```

```r
tableCompare %>%
  kbl(digits = 3) %>%
  kable_classic()
```

# 5 Risk sharing with limited commitoment: estimation

### 5.0.1 Model

The main idea in the estimation is to find the parameters that explain the consumption pattern of a household:

$$c_{it} = \hat{c}_{it}(y_t, x_{t-1}; \theta),$$

where $\hat{c}_{it}$ is consumption predicted by the limited commitment risk sharing model with a given set of parameters, $\theta$. To explain the discrepancies from the observed consumption and the predicted consumption, I introduce multiplicative measurement errors in consumption, as I introduced in the full risk-sharing model. With this, I get the following relationship:

$$\log(c^*_{it}) = \log(\hat{c}_{it}(y_t, x_{t-1}; \theta)) + \varepsilon^c_{it},$$

where $\varepsilon^c_{it} \sim N(0, \gamma_C^2)$.

The previous-period relative Pareto weight, $x_{t-1}$, is derived by the ratio of the marginal utilities in the previous period:

$$x_{t-1} = \frac{c_{v,t-1}^{-\sigma}}{c_{i,t-1}^{-\sigma}} = \frac{c_{v,t-1}^{-\sigma}}{c_{i,t-1}^{*-\sigma} / \exp(\varepsilon^c_{i,t-1})},$$

where $c_{v,t} = \frac{1}{N}\sum_i c_{i,t}$.

Note that consumption measurement errors affect the consumption prediction, $\hat{c}$ through $x_{t-1}$. And the effect is non-linear in the measurement errors and does not have an analytical expression. Therefore, for estimation, I use the simulated maximum likelihood method.

### 5.0.2 Estimation steps

1. Randomly generate measurement errors, and denote their $s$'th simulations for $i$ at $t$ by $\varepsilon^c_{s,i,t}$. Then calculate "true" consumption of households and "village", $c_{s,i,t} = c^*_{i,t}/\exp(\varepsilon_{s,i,t})$ and $c_{s,v,t} = \exp\left(\frac{1}{N}\sum_i \log(c_{s,i,t})\right)$. The latter is calculated this way, instead of $c_{s,v,t} = \frac{1}{N}\sum_i c_{s,i,t}$, to be aligned with how the village consumption is calculated in the full risk-sharing model estimation.
2. Calculate the previous-period relative Pareto weight by $x_{s,t-1} = \frac{c_{s,v,t-1}^{-\sigma}}{c_{s,i,t-1}^{-\sigma}}$.

3. Predict the consumption given $x_{s,t-1}$ and $y_{s,t}$. For this, get $x_{s,t}$ based on $x_{s,t-1}$ and the bounds in relative Pareto weights given $\theta$ and $y_{s,t}$. Note that this updating rule approximates the $N$-household case by the "one-versus-the-other" approach.
4. Calculate the likelihood by $f(\log(c^*_{it}) - \log\left(\hat{c}_{it}(y_{s,t}, x_{s,t-1}; \theta)\right), \theta)$, where $f(x, \theta)$ is the density function of $N(0, \gamma_C^2)$.
5. Take the average of the likelihood across $s$: $\frac{1}{S}\sum_s f(\log(c^*_{it}) - \log\left(\hat{c}_{it}(y_{s,t}, x_{s,t-1}; \theta)\right), \theta)$.
6. Take a logarithm and sum it over $i$ and $t$.

For standard errors, I do not assume that the model is correctly specified. Without autocorrelation, the standard errors of the parameters is the squared roots of the diagonal elements of $A(\hat{\theta})^{-1}B(\hat{\theta})A(\hat{\theta})^{-1}$. Here, $A(\hat{\theta})$ is the Hessian matrix of the log-likelihood function and $B(\hat{\theta})$ is the outer product of scores.

## 5.1 Estimation code

```
pacman::p_load(
  tidyverse,
  kableExtra,
  pracma
)
```

### 5.1.1 Load data and functions

```
load('IntermediateData/allData.RData')
load('IntermediateData/lc_hom_model_functions.RData')
```

### 5.1.2 Simulate random measuremente errors

```
set.seed(123)
numSimulations <- 500
measurementErrorArray <- array(
  rnorm(hnum * tnum * numSimulations),
  dim = c(hnum, tnum, numSimulations)
  )
```

### 5.1.3 Functions to calculate likelihood

```
interpolateRelativeParetoWeightBound <- function(
    householdIncomeGridPoints,
    villageIncomeGridPoints,
    relativeParetoWeightBoundsVec,
    householdIncomeTrue,
    villageIncomeTrue
) {
  interp2(
    x = householdIncomeGridPoints,
    y = villageIncomeGridPoints,
    Z = matrix(
      relativeParetoWeightBoundsVec,
      nrow = length(villageIncomeGridPoints),
      byrow = TRUE
      ),
    xp = householdIncomeTrue %>%
      pmax(min(householdIncomeGridPoints)) %>%
      pmin(max(householdIncomeGridPoints)),
    yp = villageIncomeTrue %>%
      pmax(min(villageIncomeGridPoints)) %>%
      pmin(max(villageIncomeGridPoints)),
    method = "linear"
  )
}

updateRelativeParetoWeight <- function(
    previousRelativeParetoWeight,
    relativeParetoWeightLowerBound,
    relativeParetoWeightUpperBound
    ) {
  previousRelativeParetoWeight %>%
    pmax(relativeParetoWeightLowerBound) %>%
    pmin(relativeParetoWeightUpperBound)
}

calculateCurrentRelativeParetoWeightEachObsBySimLCHom <- function(
    village,
    meanClass,
    CVClass,
    householdAR1EstimationResult,
    villageAR1EstimationResult,
    householdIncomeTrue,
    villageIncomeTrue,
    previousRelativeParetoWeight,
    relativeParetoWeightBoundsArray
) {

    householdIncomeGridPoints <- (
      householdAR1EstimationResult[[village]]$gridPointsArray[meanClass, CVClass,]
    )
    villageIncomeGridPoints <- villageAR1EstimationResult[[village]]$gridPoints

    relativeParetoWeightLowerBound <- interpolateRelativeParetoWeightBound(
      householdIncomeGridPoints,
      villageIncomeGridPoints,
      relativeParetoWeightBoundsArray[meanClass, CVClass, , 1],
      householdIncomeTrue,
      villageIncomeTrue
      )

    relativeParetoWeightUpperBound <- interpolateRelativeParetoWeightBound(
      householdIncomeGridPoints,
      villageIncomeGridPoints,
      relativeParetoWeightBoundsArray[meanClass, CVClass, , 2],
      householdIncomeTrue,
      villageIncomeTrue
      )

    currentRelativeParetoWeight <- updateRelativeParetoWeight(
      previousRelativeParetoWeight,
      relativeParetoWeightLowerBound,
      relativeParetoWeightUpperBound
      )

    return(currentRelativeParetoWeight)
}

calculateLogLikelihoodForEachObsLCHom <- function(
    param,
    village,
    consdat,
    incdatRescaled,
    householdIncMeanClassVec,
    householdIncCVClassVec,
    householdAR1EstimationResult,
    villageAR1EstimationResult,
    measurementErrorArray,
    villageIndicatorMatrix,
    numIncomeStates,
    numSimulations
) {

  delta <- param[1]
  sigma <- param[2]
  punishment <- param[3]
  gammaC2 <- param[4]

  relativeParetoWeightBoundsArray <- solveLCRiskSharingByVillage(
      village,
      delta,
      sigma,
      punishment,
      householdAR1EstimationResult,
      villageAR1EstimationResult,
      numIncomeStates,
      numRelativeParetoWeights = 2000,
      iterationLimit = 100,
      diffLimit = 1e-8
  )

  consVillage <- consdat[villageIndicatorMatrix[, village], ]
  incdatRescaledVillage <- incdatRescaled[villageIndicatorMatrix[, village], ]
  measurementErrorArrayVillage <- measurementErrorArray[
    villageIndicatorMatrix[, village], ,
    ]
  numHouseholds <- householdAR1EstimationResult[[village]]$numHouseholds

  currentRelativeParetoWeightArray <- array(NA, dim = c(numHouseholds, tnum - 1, numSimula

  for (simulationIndex in seq(1, numSimulations)) {
    consTrue <- (
      consVillage
      / exp(measurementErrorArrayVillage[, , simulationIndex] * sqrt(gammaC2))
      )

    relativeParetoWeightMatrix <- (
      outer(
        rep(1, numHouseholds),
        calculateMarginalUtility(consTrue, sigma) %>% log %>% colMeans %>% exp
        )
      / calculateMarginalUtility(consTrue, sigma)
    )

    for (householdIndex in seq(1, numHouseholds)) {
      for (periodIndex in seq(2, tnum)) {

        meanClass <- householdIncMeanClassVec[villageIndicatorMatrix[, village]][household
        CVClass <- householdIncCVClassVec[villageIndicatorMatrix[, village]][householdInde

        householdIncomeTrue <- incdatRescaledVillage[householdIndex, periodIndex]
        villageIncomeTrue <- (incdatRescaledVillage %>% colMeans(na.rm = TRUE))[periodInde

        householdCons <- consVillage[householdIndex, periodIndex]
        previousRelativeParetoWeight <- relativeParetoWeightMatrix[householdIndex, (period

        currentRelativeParetoWeightArray[householdIndex, periodIndex - 1, simulationIndex]
          calculateCurrentRelativeParetoWeightEachObsBySimLCHom(
            village,
            meanClass,
            CVClass,
            householdAR1EstimationResult,
            villageAR1EstimationResult,
            householdIncomeTrue,
            villageIncomeTrue,
            previousRelativeParetoWeight,
            relativeParetoWeightBoundsArray
            )
        )
      }
    }
  }
  likelihoodArrayPerSim <- array(NA, dim = c(numHouseholds, tnum - 1, numSimulations))
  for (simulationIndex in seq(1, numSimulations)) {

    for (householdIndex in seq(1, numHouseholds)) {
      for (periodIndex in seq(2, tnum)) {
        logConsPredict <- (
          log(
            (incdatRescaledVillage[, periodIndex] %>% sum(na.rm = TRUE))
            / (
              (currentRelativeParetoWeightArray[, periodIndex - 1, simulationIndex]^(1 / s
                sum(na.rm = TRUE)
              )
            )
          + 1 / sigma * log(
            currentRelativeParetoWeightArray[householdIndex, periodIndex - 1, simulationIn
            )
        )
        likelihoodArrayPerSim[householdIndex, periodIndex - 1, simulationIndex] <- dnorm(
          log(consVillage[householdIndex, periodIndex])
          - logConsPredict,
          sd = sqrt(gammaC2)
        )
      }
    }
  }

  logLikelihoodArray <- array(NA, dim = c(numHouseholds, tnum - 1))
  for (householdIndex in seq(1, numHouseholds)) {
    for (periodIndex in seq(2, tnum)) {
      likelihoodSimMean <- likelihoodArrayPerSim[householdIndex, periodIndex - 1, ] %>% me
      logLikelihoodArray[householdIndex, periodIndex - 1] <- log(likelihoodSimMean %>% pma
    }
  }

  return(logLikelihoodArray)
}

calculateLogLikelihoodLCHom <- function(
    param,
    village,
    consdat,
    incdatRescaled,
    householdIncMeanClassVec,
    householdIncCVClassVec,
    householdAR1EstimationResult,
    villageAR1EstimationResult,
    measurementErrorArray,
    villageIndicatorMatrix,
    numIncomeStates,
    numSimulations
    ) {

  logLikelihoodForEachObs <- calculateLogLikelihoodForEachObsLCHom(
    param,
    village,
    consdat,
    incdatRescaled,
    householdIncMeanClassVec,
    householdIncCVClassVec,
    householdAR1EstimationResult,
    villageAR1EstimationResult,
    measurementErrorArray,
    villageIndicatorMatrix,
    numIncomeStates,
    numSimulations
  )

  logLikelihood <- sum(logLikelihoodForEachObs)

  return(- logLikelihood)
}
```

```
estimateMLLCHom <- function(
  initialParam,
  lowerBounds,
  upperBounds,
  village,
  consdat,
  incdatRescaled,
  householdIncMeanClassVec,
  householdIncCVClassVec,
  householdAR1EstimationResult,
  villageAR1EstimationResult,
  measurementErrorArray,
  villageIndicatorMatrix,
  numIncomeStates,
  numSimulations
) {

  optimRes <- optim(
    initialParam,
    calculateLogLikelihoodLCHom,
    village = village,
    consdat = consdat,
    incdatRescaled = incdatRescaled,
    householdIncMeanClassVec = householdIncMeanClassVec,
    householdIncCVClassVec = householdIncCVClassVec,
    householdAR1EstimationResult = householdAR1EstimationResult,
    villageAR1EstimationResult = villageAR1EstimationResult,
    measurementErrorArray = measurementErrorArray,
    villageIndicatorMatrix = villageIndicatorMatrix,
    numIncomeStates = numIncomeStates,
    numSimulations = numSimulations,
    method = "L-BFGS-B",
    lower = lowerBounds,
    upper = upperBounds,
    control = list(trace = 5, maxit = 200),
    hessian = TRUE
    )


  score <- jacobian(
    function(x) calculateLogLikelihoodForEachObsLCHom(
      x,
      village,
      consdat,
      incdatRescaled,
      householdIncMeanClassVec,
      householdIncCVClassVec,
      householdAR1EstimationResult,
      villageAR1EstimationResult,
      measurementErrorArray,
      villageIndicatorMatrix,
      numIncomeStates,
      numSimulations
    ) %>% as.vector,
    optimRes$par
  )

  standardErrors <- (
    (optimRes$hessian %>% solve)
    %*% reduce(
      map(
        seq(nrow(score)),
        ~ outer(score[., ], score[., ])
      ),
      function(x, y) x + y
    )
    %*% (optimRes$hessian %>% solve)
  ) %>% diag %>% sqrt

  return(list(optimRes = optimRes, standardErrors = standardErrors))
}
```

## 5.2 Estimation result

```
initialParam <- c(0.95, 3.0, 0.3, 0.03)
lowerBounds <- c(0.5, 1.0, 0.0, 1e-3)
upperBounds <- c(0.98, 5.0, 0.99, 1.0)

estimationLCHomRes <- map(
  seq(1, numVillages),
  ~ estimateMLLCHom(
    initialParam,
    lowerBounds,
    upperBounds,
    .,
    consdat,
    incdatRescaled,
    householdIncMeanClassVec,
    householdIncCVClassVec,
    householdAR1EstimationResult,
    villageAR1EstimationResult,
    measurementErrorArray,
    villageIndicatorMatrix,
    numIncomeStates,
    numSimulations
  )
)
```

```
createRegressionTable <- function(village) {

  estimationLCHomTable <- c()
  for (varIndex in seq_along(estimationLCHomRes[[village]]$optimRes$par)) {
    estimationLCHomTable <- c(
      estimationLCHomTable,
      formatC(estimationLCHomRes[[village]]$optimRes$par[varIndex], digits = 3, format = "
      str_interp(
        '(${formatC(estimationLCHomRes[[village]]$standardErrors[varIndex], digits = 3, fo
        )
      )
  }
  estimationLCHomTable <- c(
    estimationLCHomTable,
    formatC(-estimationLCHomRes[[village]]$optimRes$value, digits = 3, format = "f")
  )
  return(estimationLCHomTable)
}

estimationLCHomTable <- do.call(
  cbind,
  map(
    seq(1, numVillages),
    createRegressionTable
  )
)

colnames(estimationLCHomTable) <- c("Aurepalle", "Kanzara", "Shirapur")
rownames(estimationLCHomTable) <- c(
  "Discount factor",
  "",
  "Coef of RRA",
  "",
```

|  | Aurepalle | Kanzara | Shirapur |
|---|---|---|---|
| Discount factor | 0.980 | 0.748 | 0.661 |
|  | (0.025) | (0.030) | (0.232) |
| Coef of RRA | 2.998 | 2.905 | 2.756 |
|  | (0.078) | (0.013) | (0.066) |
| punishment parameter | 0.234 | 0.347 | 0.004 |
|  | (0.035) | (0.032) | (0.004) |
| Variance of consumption measurement errors | 0.035 | 0.028 | 0.059 |
|  | (0.005) | (0.004) | (0.009) |
| Log likelihood | -3.129 | -9.478 | -18.022 |

```
    "punishment parameter",
    "",
    "Variance of consumption measurement errors",
    "",
    "Log likelihood"
    )

  estimationLCHomTable %>%
    kbl(digits = 3) %>%
    kable_classic()
```

Coate, Stephen, and Martin Ravallion. 1993. "Reciprocity Without Commitment: Characterization and Performance of Informal Insurance Arrangements." *Journal of Development Economics* 40 (1): 1–24.

Fafchamps, Marcel. 1999. "Risk sharing and quasi-credit." *The Journal of International Trade & Economic Development* 8 (3): 257–78. https://doi.org/10.1080/09638199900000016.

Kocherlakota, N. R. 1996. "Implications of Efficient Risk Sharing without Commitment." *The Review of Economic Studies* 63 (4): 595–609. https://doi.org/10.2307/2297795.

Laczó, Sarolta. 2015. "Risk Sharing with Limited Commitment and Preference Heterogeneity: Structural Estimation and Testing." *Journal of the European Economic Association* 13 (2): 265–92.

Ligon, Ethan, Jonathan P. Thomas, and Tim Worrall. 2002. "Informal Insurance Arrangements with Limited Commitment: Theory and Evidence from Village Economies." *Review of Economic Studies* 69 (1): 209–44. https://doi.org/10.1111/1467-937X.00204.

Marcet, Albert, and Ramon Marimon. 2019. "Recursive Contracts." *Econometrica* 87 (5): 1589–1631.

Townsend, Robert M. 1994. "Risk and Insurance in Village India." *Econometrica* 62 (3): 539. https://doi.org/10.2307/2951659.

Udry, Christopher. 1994. "Risk and Insurance in a Rural Credit Market: An Empirical Investigation in North Nigeria." *Review of Economic Studies* 61 (3): 495–526.