```
# Notes ---------------------------------------------------------------
------------------------------
# Date: 28.12.2018
# Author: Michal Stachnio


# 1) Will need to deal with the fact initial quantities can be decimal
values. It shouldn't be the case
#    I should therefore round it up and either adjust the assets size
to fit the values, or redistribute the sizes so that it works.
# 2) Important: When the asset reduction takes place, it creates a new
equilibrium price value for the asset. This means it does not react
the same way for a given volume sold!
# Packages ------------------------------------------------------------
---------------------------------

library(igraph)

# Fucntions -----------------------------------------------------------
----------------------

NamingString = function(String, startingValue, size) {
  nameString = matrix("0", nrow = 1, ncol = size, byrow = TRUE)
  for (i in 1:size) {
    nameString[, i] = paste(String, i - 1 + startingValue, sep = "") #
sep="" means that there is no space between w and the number
  }
  return (nameString)
}

NamingRows = function(String, data, startingValue) {

  # Input Variables:
  # String        = value in "" that will be the name that will be
iterated
  # data          = data structure that will have a new name for rows
  # startingValue = allows to start with 0 or 1 etc...

  dataNames = matrix("0", nrow = nrow(data), ncol = 1, byrow = TRUE)
  for (i in 1:nrow(dataNames)) {
    dataNames[i, ] = paste(String, i - 1 + startingValue, sep = "") #
sep="" means that there is no space between w and the number
  }
  rownames(data) = c(dataNames)
  return (data)
}

NamingCols = function(String, data, startingValue) {
  dataNames = matrix("0", nrow = 1, ncol = ncol(data), byrow = TRUE)
  for (i in 1:ncol(dataNames)) {
    dataNames[, i] = paste(String, i - 1 + startingValue, sep = "") #
sep="" means that there is no space between w and the number
  }
  colnames(data) = c(dataNames)
```

```
  return (data)
}

# Generating the bipartiate graph -----------------------------------
--------------------------------------------------------

Generategraph = function(n_Banks, m_Assets, linkProbabiliy) {
  # Generates the graph
  graph = sample_bipartite(n_Banks, m_Assets, p=linkProbabiliy)
  # Changes the form of the vertex for the graph
  V(graph)$shape <- c("square", "circle")[V(graph)$type+1]
  # Deletes the label
  V(graph)$label <- c(NamingString("bank",1,n_Banks),
NamingString("asset",1,m_Assets))

  return(graph)
}

# System Rules -------------------------------------------------------
-----

# Liquidation asset size
# Positive Delta_A means the bank needs to liquidate assets
LiquidationQuantity = function(gamma, target_Leverage, balanceSheet,
n_Banks, banks_buyback_parameter) {
  delta_asset = matrix(0, nrow = n_Banks, ncol = 1)
  for (i in 1:n_Banks) {
   # The below might not be necessary. The bankruptcy condition should
perhaps be done somewhere else
    # if (balanceSheet[i, 1] + balanceSheet[i, 2] - balanceSheet[i, 3]
< 0 ) {
    #  delta_asset[i] = balanceSheet[i, 1]
    # } else {
    if (banks_buyback_parameter == 1) {
      delta_asset[i] = gamma[i] * balanceSheet[i, 1] * ((1 -
(target_Leverage[i] * balanceSheet[i, 4])/balanceSheet[i, 1]))
    } else {
      if (target_Leverage[i] < (balanceSheet[i, 1] / balanceSheet[i,
4])) {
        delta_asset[i] = gamma[i] * balanceSheet[i, 1] * ((1 -
(target_Leverage[i] * balanceSheet[i, 4])/balanceSheet[i, 1]))
      }
    }
  }
  return(delta_asset)
}

# Bankruptcy Check
BankruptcyCheck = function(balanceSheet, delta_asset) {
  for (i in 1:n_Banks) {
    if (balanceSheet[ ,1] + balanceSheet[ ,2] - balanceSheet[ ,3] < 0
) {
      delta_asset[i] = balanceSheet[ i,1]
    }
```

```
  }
  return(delta_asset)
}

# Price Impact
Price_Impact = function(decision_Volume_Traded, liquidity_factor,
m_Assets, p_0, System_Q, daily_market_volume, net_Volume_Traded) {

  # External trade is the demand for the asset outside of the
financial system
  external_Trade = matrix(System_Q * daily_market_volume, nrow =
m_Assets, ncol = 1)
  external_Trade = NamingRows("asset", external_Trade, 1)

  net_Volume_Traded = matrix(0, nrow = m_Assets, ncol = 1)
  net_Volume_Traded = rowSums(decision_Volume_Traded) + external_Trade

  delta_price = matrix(0, nrow = m_Assets, ncol = 1)
  delta_price = log(liquidity_factor * net_Volume_Traded/System_Q *
(exp(1) - 1) + 1 ) * p_0

  p_t1 = p_0 * exp(liquidity_factor * ((net_Volume_Traded +
rowSums(decision_Volume_Traded))/System_Q)) #*
net_Volume_Traded/abs(net_Volume_Traded)
  # for (i in 1:m_Assets) {
  #   if (p_t1[i] < 0 ) {
  #     p_t1[i] = 0
  #   }
  # }
  return(list("price" = p_t1, "net_Volume_Traded" =
net_Volume_Traded))
}

# Liquidation Schedules ----------------------------------------------
--------------

# In case of bankruptcy liquidation condition:


# Approach 1: Pro-Rata Liquidation

# IMPORTANT: will need to check if the bankruptcy condition is
fulfilled !!
Prorata_Liquidation = function(delta_asset, q_t, p_t, n_Banks,
m_Assets, decision_Volume_Traded, balanceSheet) {
  # Define the net volume vector

  # The part that checks if a bank is bankrupt
  bank_index_array = 1:n_Banks
  for (i in 1:n_Banks) {
    if (balanceSheet[i, 1] + balanceSheet[i, 2] - balanceSheet[i, 3] <
0) {
      decision_Volume_Traded[, i] = -t(q_t[i, ])
      bank_index_array = bank_index_array[!bank_index_array %in% i]
    }
```

```
  }


  for (i in bank_index_array) {
    leftover_delta_asset = delta_asset[i]
    # counts how many assets are held by each bank
    count_Asset_Held = length(which(q_t[i, ] > 0))
    if (delta_asset[i] == 0) {
      per_Asset_Liquidation_Quantity = 0
    } else {
      per_Asset_Liquidation_Quantity = delta_asset[i] /
count_Asset_Held
    }

    repeat{
      test = 0
      for (j in which(q_t[i, ] > 0)) {
        # If a bank does not have enough of one asset to liquidate on
a pro-rata basis,
        # it liquidates all it has and then adapts how much it
liquidates of other assets.
        if (q_t[i, j] * p_t[j] < per_Asset_Liquidation_Quantity) {
          # all the position in the asset is liquidated
          decision_Volume_Traded[j, i] = - q_t[i, j]
          leftover_delta_asset = delta_asset[i] +
rowSums(t(decision_Volume_Traded[, i]))
          count_Asset_Held = count_Asset_Held - 1
          per_Asset_Liquidation_Quantity = leftover_delta_asset /
count_Asset_Held
        } else {
          test = test + 1
        }
      }
      if (test == length(which(q_t[i,] > 0))) {
        break
      }
    }
    # If there are some assets for which the bank can fully liquidate,
it liquidates equal amounts:
    if (count_Asset_Held > 0) {
      # Identifies the index of the assets where the bank still holds
a position:
      assign_Vector = which((q_t[i, ] - decision_Volume_Traded[, i]) >
0)
      for (q in assign_Vector) {
        decision_Volume_Traded[assign_Vector, i] = -
leftover_delta_asset / count_Asset_Held
      }
    }
  }
  return(decision_Volume_Traded)
}

# Approach 2: Liquidation based on bank holdings:
```

```
Equity = function(decision_Volume_Traded, q_t, p_0) {
  equityValue = q_t %*% Price_Impact(decision_Volume_Traded,
liquidity_factor, m_Assets, p_0, System_Q, daily_market_volume)$price
+ balanceSheet[, 2] - balanceSheet[, 3]

  # Constraints:


  return(equityValue)
}



BankHolding_Liquidation = function(delta_asset, q_t, p_t, n_Banks,
m_Assets, decision_Volume_Traded, balanceSheet) {

}

# System actualisation rules ----------------------------------------
---------------------
System_Update = function(method_selection, gamma, asset_t, equity_t,
target_Leverage, q_t, p_t, n_Banks, m_Assets, liquidity_factor, p_0,
                          balanceSheet, banks_buyback_parameter) {
  # choose 1, 2, 3, 4 for each method
  # Defines the quantities to be liquidated.

  # 1) Defines how much assets bank need to liquidate
  delta_asset = LiquidationQuantity(gamma, target_Leverage,
balanceSheet, n_Banks, banks_buyback_parameter)

  # 2) Defines the liquidation schedule
  decision_Volume_Traded = matrix(0, nrow = m_Assets, ncol = n_Banks)
  decision_Volume_Traded = NamingRows("asset", decision_Volume_Traded,
1)
  decision_Volume_Traded = NamingCols("bank", decision_Volume_Traded,
1)

  if (method_selection == 1) {
    decision_Volume_Traded = Prorata_Liquidation(delta_asset, q_t,
p_t, n_Banks, m_Assets, decision_Volume_Traded, balanceSheet)
  } else if (method_selection == 2) {

  } else if (method_selection == 3) {

  } else {

  }

  # 3) Calculate the price impact of decision_Volume_Traded
  p_t = Price_Impact(decision_Volume_Traded, liquidity_factor,
m_Assets, p_0, System_Q, daily_market_volume)

  # 4) Settle the cash of the trade
  balanceSheet[, 2] = balanceSheet[, 2] + t(t(p_t) %*% -
decision_Volume_Traded)
```

```
  # 5) Reduce the quantities of holdings of banks
  q_t = q_t + t(decision_Volume_Traded)

  # 6) reupdate the asset values
  balanceSheet[, 1] = q_t %*% p_t

  # 7) Calculate the equity
  balanceSheet[, 4] = balanceSheet[, 1] + balanceSheet[, 2] -
balanceSheet[, 3]

  # The funciton should return: the prices, the balance sheets, the
quantities,

  return(list("balanceSheet" = balanceSheet, "Quantities" = q_t,
"Prices" = p_t))
}

# On top of this:
# - monitoring of the bankruptcy rate, the delta asset sold, the total
equity of the sstem
# - do a loop that stops once there are no more bankruptcies
# - make a loop so that it does multiple simulations and collects
aggregate data on the simulations
# -


# Execution Code -----------------------------------------------------
-------

# Parameters
n_Banks = 10
m_Assets = 4
linkProbabiliy = 0.5
gamma = matrix(data = 0.1, nrow = n_Banks, ncol = 1)
liquidity_factor = matrix(0, nrow = m_Assets, ncol = 1)
  for (i in 1:m_Assets) {
    liquidity_factor[i] = 0.5
  }
daily_market_volume = 0.1
assetReduction = 0.1 # Intial system shock
banks_buyback_parameter = 0


# Generating the balance Sheet
asset_0 = 80
cash_0 = 20
liabilities_0 = 96
intial_Price = matrix(data = 1, nrow = m_Assets, ncol = 1)



# PART 1: Setting up the system

# Plots the graph, changes the color of the vertexes depending on the
```

```
type
graph1 = Generategraph(n_Banks, m_Assets, linkProbabiliy)


# Making sure each bank has at least one asset:
assetChoiceVector = 1:m_Assets
for (i in 1:n_Banks) {
  if (sum(graph1[i, ]) == 0) {
    assetChoice = sample(assetChoiceVector, 1)
    graph1 = graph1 + edge(c(i, assetChoice + n_Banks))
  }
}
# Making sure each asset has at least one bank:
bankChoiceVector = 1:n_Banks
for (i in 1:m_Assets) {
  if (sum(graph1[, i + n_Banks]) == 0) {
    bankChoice = sample(bankChoiceVector, 1)
    graph1 = graph1 + edge(c(i + n_Banks, assetChoice))
  }
}


# Create the k_i vector that has the degrees of each bank
k_i = matrix(data = NA, ncol = 1, nrow = n_Banks)
for (i in 1:n_Banks) {
  k_i[i] = sum(graph1[i])
}
k_i = NamingRows("bank", k_i, 1)

# Create the l_j vector that has the degrees of each asset
l_j = matrix(data = NA, ncol = 1, nrow = m_Assets)
for (j in 1:m_Assets) {
  l_j[j] = sum(graph1[j+n_Banks])
}
l_j = NamingRows("asset", l_j, 1)

balanceSheet = matrix(data = NA, nrow = n_Banks, ncol = 4)
colnames(balanceSheet) = c("Assets", "Cash", "Liabilities", "Equity")
balanceSheet = NamingRows("bank", balanceSheet, 1)
balanceSheet[ ,1] = asset_0
balanceSheet[ ,2] = cash_0
balanceSheet[ ,3] = liabilities_0
equity_0 = asset_0 + cash_0 - liabilities_0
balanceSheet[ ,4] = equity_0


# defining the initial price of assets:
p_0 = matrix(data = intial_Price, nrow = m_Assets, ncol = 1)
p_0 = NamingRows("asset", p_0, 1)

# The matrix of asset quantities owned
q_0 = matrix(data = 0, nrow = n_Banks, ncol = m_Assets )
q_0 = NamingRows("bank", q_0, 1)
q_0 = NamingCols("asset", q_0, 1)
```

```
# Assigns the initial quantities owned by each bank of all assets
for (i in 1:n_Banks) {
  for (j in 1:m_Assets) {
    if (graph1[i, j+ n_Banks] == 1) {
      q_0[i, j] = balanceSheet[ i,1] / (k_i[i] * p_0[j])
    }
  }
}
System_Q = rowSums(t(q_0))

# Defining the termporal variables:
q_t = q_0
p_t = p_0
k_t = k_i
balanceSheet[, 1] = asset_0
equity_t = balanceSheet[, 4]
target_Leverage = asset_0 / balanceSheet[, 4] # Setting the target
leverage at the initial leverage level

# Part 2: Doing the simulation

# Shocking the system via price reduction
assetChoice = sample(assetChoiceVector, 1)
p_t[assetChoice] = (1 - assetReduction) * p_t[assetChoice]

# Updating asset values
balanceSheet[, 1] = q_t %*% p_t

# Updating equity values
balanceSheet[, 4] = balanceSheet[, 1] + balanceSheet[, 2] -
balanceSheet[, 3]

numberIterations = 20


asset_evolution = matrix(0, nrow = numberIterations, ncol =1)
prices_evolution = matrix(0, nrow = numberIterations, ncol = m_Assets)
# Simulating one system
for (i in 1:numberIterations) {
  system_Update_Values = System_Update(method_selection = 1, gamma,
balanceSheet[, 1], balanceSheet[, 4], target_Leverage, q_t, p_t,
n_Banks, m_Assets, liquidity_factor, p_0,
                balanceSheet, banks_buyback_parameter)
  asset_evolution[i] = sum(balanceSheet[, 1])
  prices_evolution[i,] = t(p_t)

  balanceSheet = system_Update_Values$balanceSheet
  q_t = system_Update_Values$Quantities
  p_t = system_Update_Values$Prices

}
plot(graph1, vertex.color=c("orange", "green")[1 + (V(graph1)$type ==
"TRUE")],
     vertex.size = 5, vertex.label = NA)
plot(asset_evolution)
```