BLACK RIHNO:

Overviews and FAQs

1 BLACK_RIHNO STRUCTURE 2

2 GENERAL NOTATION OF NOT SELF-EXPLAINING PARAMETERS 3

3 FAQS FOR BLACK RIHNO 4

3.1 *GENERAL* QUESTIONS 4

3.2 QUESTIONS FOR *BANK* 5

3.3 QUESTIONS FOR *ENVIRONMENT* 7

3.4 QUESTIONS FOR *NETWORK* 7

3.5 QUESTIONS FOR *STATE* 7

3.6 QUESTIONS FOR *UPDATER* 8

3.7 QUESTIONS FOR *GENERATE_BANKS* 8

3.8 QUESTIONS FOR *GENERATE_NETWORK* 8

4 TESTS 9

4.1 TEST FOR *BANK* 9

4.2 TEST FOR ENVIRONMENT 11

4.3 TEST FOR MEASUREMENTS 11

4.4 TEST FOR *NETWORK* 11

4.5 TEST FOR *PARAMETER* 12

4.6 TEST FOR *RUNNER* 12

4.7 TEST FOR *SHOCK* 12

4.8 TEST FOR *STATE* 12

4.9 TEST FOR TRANSACTION 12

4.10 TEST FOR *UPDATER* 12

1 Black_rihno structure

black_rihno.py

from environment import Environment

➔ Creates the environment for the simulation

from state import State

➔ Defines the state of the world the economy is in (interbank interest rate, bank's risk aversion, minimum required deposit rate, etc)

from parameters import Parameters

➔ Defines some general parameter needed for the simulation (number of banks, number of simulations, graph type, etc.)

from network import Network

➔ Sets up the network and exposures amongst banks

from bank import Bank

➔ Sets up banks and what they are doing: Investing, lending to each other, earning interests, etc.

➔ Main economic body of black rihno

from transaction import Transaction

➔ Sets up a function of transaction characteristics (value, type, maturity …)

from runner import Runner

➔ This initializes the run

from updater import Updater

➔ This is lubricant which keeps the system running from period to the other and loops over all banks

from shock import Shock

➔ Defines two types of shocks: solvency shock and liquidity shock

from measurement import Measurement

➔ Defines the histogram

br-generate_Banks.py

➔ This can be used to in order to generate several banks at once all having the same initial set of parameters

br-generate_networks.py

➔ generates a random network

br-make_tests.py

➔ this file is supposed to make a test run in order to check the single bits of black rhino

2 General Notation of not self-explaining parameters

assetNumber	# number of assets available to each bank (bank.py)
assetNumber	# total number of assets in the economy (state.py)
BC	# bank capital
collateralQuality	# central bank accepts as collateral
D	# deposits
E	# excess reserves
Ep	# the planned excess reserves
gammaBank	# fraction of interbank lending in overall balance sheet
I	# investment
Ip	# the planned optimal investment
lamb	# planned optimal portfolio structure of the bank
LC	# central bank loans
Lp	# the planned interbank loans; L > 0: excess supply of interbank liquidity
p	# probability of credit success
pBank	# bank's assumed credit success probability
Q	# the current liquidity position of the bank
r	# minimum required deposit rate
rb	# interbank interest rate
rd	# interest rate on deposits
rD	# required central bank deposits
rho	# interest charged on risky investment
rhoBank	# expected return of banks
scaleFactorHouseholds	# scaling factor for deposit fluctuations
sifiSurchargeFactor	# amount of extra capital that systemically important financial institutions have to hold
thetaBank	# bank's risk aversion parameter
V	# planned optimal portfolio volume of the bank
xiBank	# scaling factor for CRRA (constant relative risk aversion)

3 FAQs for Black Rihno

3.1 *General* questions

Q: Why do all files exists in two ways (bla.py and bla.pyc)?
A: *.pyc is a compiled python file which is the readable version for the computer. .py on the other hand is the normal "human-readable" version*

Q: What is the purpose of def __str__(self) and def print_state(self)
A: *This can be used for debugging purposes*

Q: What is NumSweeps?
A: *number of update steps, how many periods will the simulation have?*

Q: Why is "TimeOfDefault" always -1 ?
A: *Here -1 means never because it is never reached*

Q: Why can modules be imported also they are in different directories? For example in environment, networkx is imported although networkx is saved in a different directory…
A: *If one tries to execute the statement directly from environment it will not work, because Python will not find network. However, if one starts black_rihno networkx will be imported directly and the statement sys.path.append('src/') will tell black_rihno where to find environment. The same holds true for files such as "baseline-50" or "log"*

Q: Why does the following course not always overwrites the text?

```
text =  "<bank identifier='" + self.identifier + "'>\n"
text += "    <value name='active' value='" + str(self.active) + "'></value>\n"
text += " <parameter type='changing' name='pBank' value='" + str(self.p) +"'></parameter>\n"
```

A: *It actually does; or to be more prices it always adds the respective strings. In theory one could make all the statements in one. However this would be much more confusing.*

Q: Is the central bank and its balance sheet somewhere explicitly modeled?
A: *No, the central bank is nowhere modeled explicitly. In fact, banks are lending from central bank which is just providing the money "out of blue". However, as most central banks usually hold a policy of an open discount window in the short run, this is a valid assumption.*

Q: Where does the liquidity come from that banks provide to each other in the interbank market? Is there any constrain to the provision of liquidity?

A: *There is no explicit constrain. Banks are able to provide additional liquidity to the market via two channels, the interest channel and the deposit channel. Note however, that both of these channels are subject to a random walk.*

3.2 Questions for *Bank*

Q: What is the exact difference between Q # the current liquidity position of the bank and Lp # the planned interbank loans. Is Q just a dummy?
A: *Q is sort of a dummy. However, it is not a dummy in the strict sense (like "eye color") but it is a dummy that (due to the financial processes) floats around from to the other period and also within the period. It indicates if a bank needs liquidity or is short of liquidity. Lp on the other hand in a way results from Q but already quantifies the excess or shortage of liquidity.*

Q: Several variables already exist in "State" (such as xi; theata;…). Why is that?
A: *In a first version of black_rihno all parameters were defined in state.py. However it has proven that it has shown that the program runs more soundly if one also has some parameters in bank.py*

Q: There are several methods, lists or variables that are defined somewhere else / which is cannot find. Such as self.calculate_optimal_investment_volume(state), "B" or "DC" Where are they defined? Where do they come from?
A: *Unlike in C++ methods, variables, etc in Python does not need to be defined in a chronological order they can be defined anywhere in the code. If you do not find a variable or method right away you can search for it using the find-tool (strg+f).This is why for example all of the bank accounts can be defined at the end of the file, despite the fact that they are already called earlier. Note that in case of instancing the actual method or variable is stored in another file*

Q: Why does the sifi_surcharge factor # reduce central bank liabilities? / # and increase banking capital. And why is this supposed to trigger a liability change?
A: *By default the sifi-factor is set to 1.0 (which means this it has no influence). However, as the code is already predefined one has the possibility to apply this non-linear factor. Larger banks (in terms of assets and connectedness) will then be required to hold more capital. For more details about the sifi_surcharge_factor see the environement.py*
If the sifi-factor is applied it will be > 1 and thus increase the banking capital and lower the central bank liabilities

Q: Why is lastInsolvency = [0, -2]? I don't understand the comment? Why -2?
A: *In fact this is a bit confusing. Normally t=-1 means never, however as we are in this case actually referring to the previous period (before the current period) we had to use -2 in order to state clear that we mean never.*

Q: Where is the state.riskAversionAmplificationFactor? Where can we find how it is defined? And why are we rounding the numbers?
A: *As it is instanced from state.py it is also defined there. And we are rounding them to make them not to long*

Q: In several cases I do not know which values have been assigned to which variables or factor. Where for example can I find the exact value of def get_new_deposits(self, scaleFactor):,
A: *As in most of the cases, the scale factor is defined in the baseline-50.xml. This file provides actual numbers to a range of parameters of black_rihno.*

Q: What are all these numbers in this function "transaction.this_transaction" stand for? For example transaction.this_transaction("rD", self.identifier, -3, value, self.rb, 0, -1)
A: *If one compares it with the following function it might get a bit clearer bank.add_transaction("L", int(bank.identifier), int(counterparty.identifier), value, interest, maturity, timeOfDefault)*
Regarding the "counterparty.identifier" it probably makes sense to explain it a little bit more in detail, as this is nowhere explicitly stated in black_rihno. Each number simply refers to one sector in our economy:

-1	*household-sector*
-2	*firm-sector*
-3	*Central bank*
-4	*banking-sector (note that when referring to banks "self.identifier" is the equivalent of -4)*

Q: how to determine the optimal portfolio structure? Self.Lamb comes out of the blue
A: *In fact, the "optimal portfolio structure" is not a real matrix (which intuitively might be understood when reading "structure") it is the share of risky vs. risk-free assets in the portfolio (which means 0 < lamb < 1). Or to be more precisely, lamb refers to the risky part while (1 – lamb) means the risk-free share.*

Q: Why is in transfer_excess_reserves the planned Volume suddenly *self.gamma*(1.0-self.lamb)*self.V*
A: *Here we are referring to the risk-free part of the portfolio, as the central bank will only accept risk free assets as collateral.*

Q: Why is in self.Ip = round(float(self.gamma*(self.lamb)*self.V), 4) (self.lamb) in brackes, while in optimalVolume = round(float(self.gamma*self.lamb*self.V), 4) it is not???
A: *There is no difference, both is possible*

Q: "self.Lp = self.Lp + value" couldn't one also write self.Lp += value ?
A: *Yes one could*

Q: Why does the statement
maturity = int(round(random.random()*state.firmLoanMaturity, 1)) will generate loans are up to 2 years?
A: *This is because in baseline-50.xml firmLoanMaturity is assigned a value of 500. As one year has approximately 250 working days and the random-function will be between 0.0 and 1.0 loans will be up to 2 years*

Q: How does the fire-sale work? Is the price determined just endogenously by the size of a bank's offer?
A: *Yes it is. See Gabrieli and Georg (2013), mimeo*

3.3 Questions for *Environment*

Q: How does the following code work? The code in Python tutorial for ElementTree is somewhat different…

```
from xml.etree import ElementTree
xmlText = open(environmentFilename).read()

element = ElementTree.XML(xmlText)
self.identifier = element.attrib['title']

self.parameters.identifier = self.identifier
```

A:

Q: In the method "initialize" why does the initial_assets has a "self"??? Is that superfluous, because there is no other variable or function in the class environment
A:

3.4 Questions for *Network*

Q: What is nx.DiGraph?
A: *it is a pre-defined module from Networkx which is needed in order to set up the module*

Q: Why is Environment needed in Network? Network is already needed in Environment???
A: *These two object are cross referencing each other. Although network is a sub-object of the object environment both of them need each other to be created.*

3.5 Questions for *State*

Q: What is the difference between p.Bank and p.Financial? both are # bank's assumed credit success probability

A: *so far there is no difference. p.Financial was already created for future extension (differentiate between real and financial assets)*

3.6 Questions for *Updater*

Q: Why does the bank gets interest for its deposits, loans and central bank loans? Shouldn't they *pay* interest?

```
bank.Q += bank.get_interest("D")
bank.Q += bank.get_interest("L")
bank.Q += bank.get_interest("LC")
```

A: *See convention about interest payment. In bank.py the code "get_intrest" specifies the interest payments. The convention is that if the transaction goes to one-self the bank is the recipient of the payment (deposits, centeral bank loans) and thus has to pay interest. If the transaction goes to someone else (not a bank) banks will get an interest payment.*

```
if (transaction.transactionTo == self.identifier):
        volume = volume - float(transaction.transactionValue)*
        float(transaction.transactionInterest)
else:
        volume = volume + float(transaction.transactionValue)*
        float(transaction.transactionInterest)
```

3.7 Questions for *generate_Banks*

Q: I do not understand the code, what means the `fileName += "%01d" % (i,)` ?
A: *This code ensures that the file name of the first bank starts with 00; 01; 02 and so on. This simply makes it a bit more structure than file names of 1; 2; 3; … 10; 11; … 21;…*

3.8 Questions for *generate_Network*

Q: Here we are just defining a random network for the cases where `(sys.argv[2] == "ba"` or "random"?
A: *yes*

Q: What is the difference between `contracts = nx.DiGraph()` and `exposures = nx.DiGraph()`?
A: *While contracts refers to all **possible** mutual lines of credit exposures is the more narrow concept as it refers to all **actual** interbank exposures*

4 Tests

4.1 Test for *Bank*

- **# update_maturity**
  This test checks bank.update_maturity(). It is successful if the maturity of all transactions is reduced by one when the bank is printed the second time. Note that for investments also the time of default has to be reduced by one.

- **# get_interest**
  This test checks bank.initialize_standard_bank. It is successful if a standard Bank with 2 assets (I = 100), E = 90, D = 250 and pReal = 0,9 etc. has been created. See 'initialize_standard_bank' in bank.py for details.

- **# liquidate_due_transactions**
  This test checks bank.liquidate_due_transactions(). It is successful if the maturity of the investments is 0 and if the two investment the return their respective value (which should be 200 in total for the standard bank).

- **# def get_ new_deposits**
  This test checks bank.get_new_deposits. It returns the change of the deposits of the bank (250). With a scaleFactor of 0.02 this change should fluctuate randomly between +5 and -5.

- **# transfer_required_deposits**
  This test checks bank.transfer_required_deposits. First we delete the required deposits of the standard bank, afterwards we calculate the new rD using the respective function. For r=0.05 the output should be -12.5

- **# reduce_banking_capital**
  This test checks bank.reduce_banking_capital. It returns the reduced banking capital. Suppose the banking capital of our standard bank is reduced by 5, so the new BC will be 40 - 5 = 35

- **# check_solvency**
  This test checks bank.check_solvency. Within this test the required_capital_ratio is set extremely high so that our standard bank will fail to meet its capital requirement. Subsequently 'active' will be set to -1

- **# check_liquidity**
  This test checks bank.check_liquidity. Within this test Q is set < 0.0 so that our standard bank will become illiquid. Subsequently 'active' will be set to -1

- # **calculate_liquidity_demand**

  This test checks bank.calculate_liquidity_demand. It will calculate the liquidity demand for the standard bank according to its parameters and the formulas used in bank.calculate_liquidity_demand. As a reminder the parameters for the standard bank are:

  gamma = 0.8
  lamb = 0.5
  V = 250.0
  Q = 0.0

  The formulas for bank.calculate_liquidity_demand are:

  Ip = gamma * lamb * V = 0.8 * 0,5 * 250.0 = 100.0
  Ep = gamma * (1.0 - lamb) * V = 0.8 * 0,5 * 250.0 = 100.0
  Lp = Q - ((Ip-I) + (Ep-E)) = 0.0 – ((100-200) + (90-100))

  The result should be Lp = 90.0

- # **transfer_excess_reserves**

  This test checks bank.transfer_excess_reserves. It is successful if the excess reserves of our standard bank are increase. Under the assumption that our banks faces a liquidity surplus (available volume) of Q = +100.0 the balance sheet of our bank should get an additional transaction-position of E with a value of 100.0.

- # **get_account**

  This test checks bank.get_account. The purpose of this method is to establish an account for our bank which contains all kinds of assets and liabilities. The method simply adds all kinds of assets and stores them in one volume. As our Banks holds 300.0 assets (2* I = 100, E = 90, D = 250) and 300 liabilities the total volume of our account should be 600.0

- # **get_account_num_transactions**

  This test checks bank.get_account_num_transactions. The purpose of this method is to count the numbers of transaction for accounts banks hold. Our standard bank has 7 transactions by default. (2* I + E + rD + BC + D + LC). As long as our bank does not have e.g. an L the number of transactions should be 7.0

- # **add_transaction**

  This test checks bank.add_transaction. The most simple way to test this function is to assign an new transaction to our bank. Therefore, let's just assign the following transaction and check whether it has been added: (type = "D", fromID = -1, toID = bank.identifier, value = 10, interest = 0.09, maturity = 0, timeOfDefault = -1)

- # **change_deposits**

  This test checks bank.change_deposits. Suppose that the change in deposits = 10.0 then the new deposits should be 260.0

- # **initialize_standard_bank()**

This test checks bank.initialize_standard_bank(). It is successfull if a standart Bank with 2 asstets (I = 100), E = 90, D = 250 and pReal = 0,9 etc. has been created. See 'initialize_standard_bank' in bank.py for details.

4.2 Test for Environment

- # initialize
  Is the logging.info established?

- # read_environment_file
  This is a function from the standard Python library, which does not really needs to be tested

- # write_environment_file(file_name)

- # initialize_banks_from_files
  Print banks and compare

- # get_state
  Check length of state

- # apply_sifi_surcharge
  Look at 1 out of 3 banks and check if sifi surcharge has been applied

4.3 Test for Measurements

- # def do_measurement()
- # def do_histograms()
- # def write_histograms()
- # def write_histogram()

4.4 Test for *Network*

- # initialize_networks
  Maybe produce some graph that plots the network

- # do_interbank_trades
  Check the change in Lp for Bank and neighbor

- # remove_inactive_bank()

Check if an inactive bank still holds nodes

- # __str()__
- # write_network_of_exposures

4.5 Test for *Parameter*

- # add_parameter
  Check if it really adds parameter

4.6 Test for *Runner*

- # do_run

4.7 Test for *Shock*

- # do_shock
- # find_largest_bank()

4.8 Test for *State*

- # addInsolvencyToHistory(time)
  Trigger an insolvency and check if it adds to the history

4.9 Test for Transaction

- # this_transaction
- # write_transaction()

4.10 Test for *Updater*

- # do_update
- # do_update_phase1

- # do_update_phase2
- # do_update_phase3

- # find_active_banks()