Eurace@Unibi Model v1.0 User Manual

Herbert Dawid
Simon Gemkow
Philipp Harting
Sander van der Hoog
Michael Neugart

February 3, 2011

Contents

Preface

This is the documentation of the Eurace@Unibi model, written at Bielefeld University. It describes the modules that were developed by the Bielefeld Team for the Eurace model, and that subsequently have entered into the Eurace@Unibi model.

The Eurace@Unibi model presented here is based on the agent-based macroeconomic simulation platform developed within the EURACE project (aka the Eurace Simulator). After the completion of the EURACE project in November 2009 the group consisting of Herbert Dawid, Simon Gemkow, Philipp Harting, Michael Neugart and Sander van der Hoog has extended and altered the model and its implementation substantially in numerous directions leading to the current version of the model (version 1.0 of February 2011).

The Eurace@Unibi model should therefore not be identfied to the Eurace Simulator. It contains several extensions and simplifications with respect to the Eurace model. A list of features that differ is the following:

1. Consumption goods producers use a Leontief production function. [In the Eurace model a Cobb-Douglas production function is used.]

2. Investment goods producers produce heterogeneous vintages of capital, with newer vintages having a higher productivity than older ones. [In the Eurace model there is just a single vintage of capital the productivity of which increases over time.]

3. Credit market: The banks pay a positive interest on deposits. The Central Bank pays the base rate on overnight deposits of the banks, and banks have to pay the base rate on their Central Bank debt (overdrafts on the standing facility). [In the Eurace model banks do not pay interest on deposits, nor does the Central Bank pay interest to banks.]

4. The financial market is highly stylized and simplified. There is a single risky asset that is a stock market index. Transactions follow from a rationing scheme, and the price mechanism is a cautious adjustment based on the excess demand for index shares. Dividends are paid on shares of the market index. Firms are not allowed to issue new shares, hence the number of outside shares remains constant. [In the Eurace model there is a multi-asset market with shares of individual firms, banks and for government bonds. A centralized Clearinghouse manages all trades by a clearing price mechanism. Firms are allowed to issue new shares to raise equity.]

5. Balance sheets. The structure of the balance sheets has been completely overhauled and checked rigorously. A list of 33 rules for stock-flow consistency has been checked as a means of model verification (see Chapter 2).

6. Bankruptcy code. The code for firm bankruptcy has been extended to ensure that bankrupt firms are able to resurface (see Chapter 6).

General acknowledgement to our Eurace partners.
We are thankful for the collaboration with our partners in the Eurace project, without whom we would not have been able to develop the model in such detail. We have had many discussions over the years that helped us clarify many issues surrounding the integration of models, testing and validation, the implementation in the FLAME framework, and issues of model development in general. The usual disclaimer applies. The authors of this documentation are solely responsible for any ommissions or failures in the documentation.

Chapter 1

About the documentation

Why we have written this user manual.

We have often heard the comment about agent-based models that it is difficult to understand precisely what is going on. The complaint is that the model is too detailed, contains too many variables and parameters (is over-specified and over-parametrized) and is not described properly or in sufficient detail. We can identify several reasons for this.

The first reason is that the agent-based research community lacks clear standards how to communicate and document the models. There does not yet exist a set of guidelines on how to build, describe, analyse and evaluate agent-based models (Squazzoni (2010)). What seems to be needed is a common modelling methodology and a set of best-practices that leads to a *lingua franca* for describing agent-based models. An attempt in this direction is given by the *'Dahlem guidelines for agent-based model documentation* (Wolf et al. (2013)) which was the result of a meeting of several groups engaged in building economic agent-based models.

A second reason may be that some agent-based models, and in particular the Eurace model, are simply too complex or too complicated to be intelligible for any human being. Humans are not computers and the human mind has inherent limitations on the amount of information it can hold. A person cannot parse and compile the code of a model and understand it, so the model must be communicated by some other means. In addition there is the need to explain the model at different levels of description for different purposes.

The lowest level of description is understanding the model without necessarily being able to use it oneself. For example, for the purpose of comparing two models in the same domain. The second level is to actually be able to use the code, and the third level is the ability to extent and alter it. Ideally, documentation should exist at each level of description but with varying degrees of detail.

Our approach to the documentation problem is to take a modular view. It is basically a divide-and-conquer approach, which we have come to dub the *general to specific approach*, in which the model is divided into several logical components according to the contexts of activities (e.g., markets). We document the behavior of each component separately, and then describe how the modules are connected, possibly in a hierarchy.

Our aim with this User Manual is to provide a documentation of the model that gives as much detail as possible, without burdening the reader with too many technical details. This means we do not just provide the source code, because in our opinion this would not provide the reader with sufficient information on how the model works. Therefore we describe the model using a mixture of explanatory text in natural language, mathematics, pseudocode and diagrams.

We hope this documentation will provide the reader with an in-depth understanding of the model and that it will be found useful as a tool for macroeconomic analysis, also by different research groups than the ones originally developing it.

1.1 Levels of documentation

Following the general practice of software documentation there are different levels at which documentation can be written, depending on the purpose and the level of expertise of the audience.

1. Requirements documentation gives a specification of what the software does or should do. The Requirements documentation is written at the beginning of the software development project and updated as it proceeds.

2. Architecture design documentation states the general requirements for the existence of routines, but does not specify the routines themselves. A good architecture document is short on details but thick on explanation.

3. Technical documentation (developers manual) is the most detailed form of documentation. It consists of text accompanying the code that describes its intended operation, that should allow developers of the code to understand how to use, apply and extend the existing code.

4. The user documentation (user manual) describes each feature of the program, but in less detail than the technical developers manual. It should be sufficiently detailed to understand and use the code, but not necessarily to alter and extend it.

5. Reference manual. This manual gives a complete reference to all entities that are used in the code: variables, functions, global constants and data structures are all listed.

Remark

For the Eurace@Unibi model the documentation currently consists of a User manual and a Reference manual.

1.2 Criteria and organisation

The following principles have guided the organisation of the text for the manuals:

- Tutorial approach: the description should be useful for a new user. We therefore have written the user manual as a walkthrough of the code, with accompanying text explaining what the code does.

- Thematic approach: the chapters of the user manual have been organised in a hierarchical way, according to the components of the software model. The modules mostly represent the economic markets, but can also refer to a particular context. For example, the firm's financial management routines are not part of a market context, but form a collection of routines that describe the firm's behavior in a certain role.

- Reference approach: In the Reference Manual we have collected tables per agent and per module. These tables contain the memory variables and functions of the agents, as well as the model constants with their default values, a list of all messages with data content, and the abstract datatypes. Furthermore, the Reference Manual has an alphabetical index to easily cross-reference the items.

We have adopted several criteria to guide our writing effort for the documentation:

1. Accuracy. The documentation should be thorough and up to date. It should accurately reflect the implementation, but it should also be concise enough so as not to burden the reader with too many unecessary details.

2. Human-readability, consistency and simplicity. Humans are not computers, so we can not expect them to be able to read source code and understand what the model does. We therefore have adopted a combination of text and source code to give the reader a walkthrough, a guide to the code. One of the objectives is to help new users to understand the code.

3. Self-documenting code. We have strived to make the code itself as clear as possible by using descriptive names for variables and functions as much as possible. For example, there exists a variable total_debt_installment_payment instead of the shorter form tot_debt_pymnt such that it is clear from the name what is meant by the variable.

4. Auto-generated documentation. The FLAME framework contains several tools that help us to automatically generate (parts of) the documentation. One of these is the output of LaTeX-tables of all memory variables, functions, constants, and data structures. These are found in the reference manual. Another very useful documentation feature are the stategraphs that are found at the end of each chapter in the user manual. These graphs are automatically generated by FLAME from the xml code.

Chapter 2

Balance Sheets and the System of National Accounts

2.1 Stock-flow consistent modelling

An important part of the testing and verification process will be the verification of the internal consistency of the model. For this task we need a stock-flow consistent (SFC) model, that can be defined as:

> "[...] models that identify economic agents with the main social categories/institutional sectors of actual capitalist economies – thoroughly describe these agents' short-period behaviors and consistently model the 'period by period' balance sheet dynamics implied by the latter." (Macedo e Silva and Dos Santos (2008, p. 2))

Using a SFC model we need to check that all monetary flows are accounted for, and that all changes to stock variables are consistent with these flows. This can be accomplished by tracking the time evolution of the balance sheets across the different sectors of the economy. This could be done by constructing a Social Accounting Matrix (SAM) that contains all the monetary flows and changes to the balance sheet between the beginning and end of an accounting period. A SAM consists of a double-entry accounting system in which each flow comes from somewhere and goes to somewhere. It shows how the balance sheets of the different economic sectors (agents) are interlinked, and it also shows how the period-by-period balance sheets change dynamically over time. Such an accounting system at the macro level provides us with a number of accounting identities that should always hold and this can be tested by an external invariant detector such as Daikon.

This provides us with a solid and economically well-founded methodology to test the consistency of the model and it increases the credibility that can be attached to the model's results. Thereby it is not only part of the testing and verification procedure, but is also part of the accreditation process. It will help to raise the acceptability and trust in the model.

2.2 Overview of balance sheets

Below we list for each agent type the items on its balance sheet. The cash flows indicated only relate to the financing activities.

Household: Refer to Tables 2.1 and 2.2.

Household labour income consists of a wage (W^h) or unemployment benefits (U^h), and they can receive government subsidies (S^h) or transfers (Tr^h). their expenditures consist of consumption (C^h), taxes (T^h) and a possible restitution payment (R^h).[1]

Households can have bank deposits (M^h) and receive interest on deposits ($r^b M^h$). They do not take out bank loans. They can purchase government bonds (n_g^h) and private equity shares of firms (n_f^h). They receive interest on the government bonds ($+r^g n_g^h$) and dividends on the shares ($\sum_f n_f^h$).

Firm - Consumer goods and investment goods: Refer to Tables 2.3 and 2.4.

The income of firms consists of sales revenues ($p_C^f R^f$), subsidies (S^f) and transfers (Tr^f). Their expenditures include the wage bill (W^f), investment costs (I^f), energy costs En^f, tax payment (T^f), debt installment and interest payments, and dividend payments.

Firms can have bank deposits (M^f, at a single bank) and bank loans (D_b^f, at multiple banks). They receive interest on the deposits ($r^b M^f$), and have to pay interest on each loan ($r^b L_b^f$). They have equity shares (N^f) on which they pay dividends ($d^f N^f$). They they do not purchase government bonds, or shares of other firms.

Bank: Refer to Tables 2.5 and 2.6.

Banks also have outstanding equity shares on which they pay dividends (N^b, $d^b N^b$). They have a portfolio of outstanding loans to firms (L_f^b) on which they receive interest ($+r_f^b L_f^b$) and debt instalment payments (ΔL^b). They do not purchase government bonds, and they do not purchase shares in other firms or banks. The banks have a standing facility with the Central Bank from which they can draw advances freely (D^b), on which they have to pay the base interest rate to the Central Bank ($-r^c D^b$). The Central Bank pays the same base interest on overnight deposits of the bank's liquid reserves ($r^c M^b$). The interest rate that the bank pays on household and firm deposits is lower than the base rate, while the interest on loans to firms is higher than the base rate ($r^b < r^c < r_f^b$). The deposits interest rate r^b is determined as a mark-down on the base rate. The interest on loans r_f^b depends on the firm-specific balance sheet, in particular on the probability of debt default.

Government: Refer to Tables 2.7 and 2.8.

The Government income consists of taxes and restitution payments (T^i, R^h). Government expenditures are: unemployment benefits (U^h), subsidies (S^i, $i = h, f$) and transfers (Tr^i, $i = h, f$).

The government deposits its liquid funds (its payment account) at the Central Bank (M^g). If there are any changes to the payment account of the government (i.e. withdrawals to pay for unemployment benefits or subsidies) this is recorded as a change in the stock of the asset M^g ($-\Delta M^g$), with a counterpart liability on the balance sheet of the Central Bank ($+\Delta M^g$). The government also has a standing facility at the Central Bank that allows it to have a negative payment account (D^g). The government has a liability that is given by the stock of currently outstanding government bonds (N^g) on which it pays coupons ($-r^g N^g$).

[1]The household repays a restitution payment to the Government if it received a monthly unemployment benefit at the start of the month, and was reemployed during the same month.

Central Bank: Refer to Tables 2.9 and 2.10.

The Central Bank can purchase government bonds (n_g^c) as a means to finance the budget deficit. It receives coupons ($+r^g n_g^c$). The Central Bank gives advances to the banks ($-\Delta D^b$), on which the banks have to pay an interest ($+r^c D^b$). The Central Bank pays interest on the overnight deposits by banks ($+r^c M^b$), but not for governments. Since the Central Bank is not allowed to make a profit, its revenues from government bonds and bank advances ($+r^g n_g^c$, $+r^c D^b$) are distributed to the government in the form of a dividend ($-Div^c$). In case of multiple governments, the total dividend payment is equally divided among the governments.

Fiat money is created automatically when banks or governments draw on their standing facilities (ΔD^g,ΔD^b), and is being destroyed whenever these funds return to the Central Bank ($\Delta D^b(-)$).

2.3 Detailed balance sheets for each agent type

Household balance sheet (h)

Table 2.1: Household balance sheet.

Assets	**Liabilities**
M^h: liquidity deposited at a given *bank* $+W^h + U^h + Tr^h$ $-C^h - T^h - R^h$ $+r^b M^h$ $+r^g n_g^h$ $+\sum d^f n_f^h + d^b n_b^h$ $-(p_f \Delta n_f^h + p_b \Delta n_b^h + p_g \Delta n_g^h)$ n_g^h: government bonds holdings $+\Delta n_g^h$ n_f^h, n_b^h: equity shares holdings of firm f and bank b $+\Delta n_f^h + \Delta n_b^h$	(none)

Explanation:

- Financial wealth (liquidity + asset wealth):

$$W = M^h + \sum_{f\in\{firms\}} n_f^h p_f + \sum_{b\in\{banks\}} n_b^h p_b + \sum_{g\in\{governments\}} n_g^h p_g$$

- p_f, p_b: daily price of equity shares issued by firm f and bank b, respectively

- p_g: daily price of the bond issued by government g

Table 2.2: Household cash flow.

Ingoing	**Outgoing**
W^h Wage	C^h Consumption
U^h Unempl. benefits	T^h Tax payment
S^h Subsidies	R^h Restitution payment
Tr^h Transfers	
$r^b M^h$ Interest on deposits	
$r^g n^h_g$ Coupons on gov bonds	
$\sum d^f n^h_f + d^b n^h_b$ Dividend income	
$\Delta n^h_f(+)$ Asset sales	$\Delta n^h_f(-)$ Asset purchases
Total income	Total expenses

Table 2.3: Firm balance sheet.

Assets	**Liabilities**
M^f: liquidity deposited at a given *bank*	D^f_b: debts to *banks*
$+p^f_C R^f + S^f + Tr^f$	$+\Delta D^f_b(+)$
$-W^f - I^f - En^f - T^f$	$-\Delta D^f_b(-)$
$+\Delta D^f_b$	
$+r^b M^f - r^b D^f_b$	
$-d^f N^f$	
I^f_m: value of local inventories at *malls*	E^f: equity
$-R^f$	
$+p^f_C X^f$	
K^f: value of physical capital	
$+I^f$	

Firm balance sheet (f)

Explanation:

- M^f, I^f_m updated daily following the firm's cash flow and sales in the local malls.

- K^f, and D^f_b updated monthly on the ideosyncratic day of the month to act.

- E^f: equity updated monthly (on the last day of the month) according to:

$$E^f = M^f + \sum_{m \in \{malls\}} I^f_m + K^f - \sum_{b \in \{banks\}} D^f_b$$

Table 2.4: Firm cash flow (CGP and IGP differ only in the item Investment costs or Energy costs).

Ingoing	**Outgoing**
$p_C^f R^f$ Sales revenues	W^f Labour costs
S^f Subsidies	I^f Investment costs
Tr^f Transfers	En^f Energy costs (IGP)
X^f Output	T^f Tax payment
$\Delta D_b^f(+)$ New credit from banks	$\Delta D_b^f(-)$ Debt installment payments
$r^b M^f$ Interest on deposits	$r^b D_b^f$ Interest payment
	$d^f N^f$ Dividend payment
Total income	Total expenses

- p_C^f: firm price level of consumption goods.
- p_K: single price of capital goods (single IGFirm).
- N^f: Total number of outstanding shares.
- d^f: Dividend per share.

Bank balance sheet (b)

Table 2.5: Bank balance sheet.

Assets	**Liabilities**
M^b: liquidity (cash reserves) (deposited at the *central bank*) $+\Delta M^h + \Delta M^f$ $-r^b(M_b^h + M_b^f)$ $+\Delta L^b(-)$ $-\Delta L^b(+)$ $+\sum r_f^b L_f^b$ $-r^c D^b$ $+r^c M^b$ $-T^b$ $-d^b N^b$ $+\Delta D^b(+)$ $-\Delta D^b(-)$	M_h^b: households' deposits at the bank $+\Delta M^h$
	M_f^b: firms' deposits at the bank $+\Delta M^f$ (withdrawals/deposits)
	D^b: standing facility (debts to the *central bank*) $+\Delta D^b(+)$ $-\Delta D^b(-)$
L_f^b: outstanding loans to firms $-\Delta L^b(-)$ $+\Delta L^b(+)$	E^b: equity $-r^b(M_b^h + M_b^f)$ $+\sum r_f^b L_f^b$ $-r^c D^b$ $+r^c M^b$ $-T^b - d^b N^b$

Table 2.6: Bank cash flow.

Ingoing	**Outgoing**
ΔM^h, ΔM^f Deposits mutation	
$\Delta L^b(-)$ Firm loan installments	$\Delta L^b(+)$ New loans to firms
$\sum r_f^b L_f^b$ Firm interest payments	
$\Delta D^b(+)$ New ECB debt (standing facility)	$\Delta D^b(-)$ Reduction ECB debt
$r^c M^b$ Interest received from ECB	$r^c D^b$ Interest payment on ECB debt
$r^b(M_b^h + M_b^f)$ Interest on deposits	$d^b N^b$ Dividend payout
	T^b Tax payment
Total income	Total expenses

Explanation:

- M_h^b, M_f^b, L_f^b updated daily following the private sector deposits changes and the credit market outcomes.
- N^f: Total number of outstanding shares.
- d^f: Dividend per share.
- Cash reserves M^b and equity E^b updated daily following the bank's cash flows.
- The balance sheet identity reads:

$$M^b = D^b + \sum_{h \in \{households\}} M_h^b + \sum_{f \in \{firms\}} M_f^b + E^b - \sum_{f \in \{firms\}} L_f^b$$

ECB interest on bank cash reserves:

- Interest received: the bank receives interest on overnight deposits from the Central Bank: $r^c M^b$.
- Interest paid: the bank pays interest on its ECB debt: $r^c D^b$.

Loans to firms are created ex nihilo:

- When a new credit is created it reduces the bank's liquidity. New loans are added to the asset L_f^b (loans to firms) and subtracted from the asset M^b (cash reserves).
- When credit is repaid it is added to the bank's cash reserves. Debt installments are a reduction from the asset L_f^b, and credited to the bank's liquidity M^b (cash reserves).
- Note that the bank providing the loan may not be the same as the bank at which a firm has deposits.

ECB debt and cash reserves: limited by a minimum cash reserve requirement

- There is a minimum cash reserve ratio R^{min} that bank's have to respect.
- If cash reserves M^b drop below the minimum $R^{min}(M_f^b + M_h^b)$ (based on current deposit levels) then M^b is set equal to the minimum cash reserve level: $M^b = R^{min}(M_f^b + M_h^b)$. The bank's ECB debt D^b is automatically increased by the difference: $\Delta D^b(+)$ is added to bank liquidity (cash reserves).
- If M^b superceedes the minimum cash reserve requirement and the bank has positive ECB debt D^b, then ECB debt is partially or totally repaid. $\Delta D^b(-)$ is subtracted from the bank's cash reserves, and the debt of the bank is reduced.
- As a consequence, also the balance sheet of the Central Bank is reduced (see Central Bank balance sheet).

Government balance sheet (g)

Table 2.7: Government balance sheet.

Assets	**Liabilities**
M^g: liquidity deposited at the central bank $+\sum_{i\in h,f,b} T^i + \sum_h R^h$ $-\sum_{i\in h,f} S^i - \sum_{i\in h,f} Tr^i - \sum_h U^h$ $-r^g N^g$ $+\Delta D^g$ $+\Delta n_g^c p^g$	D^g: standing facility with the central bank $+\Delta D^g$ N^g: number of outstanding bonds $+\Delta N^g$

Table 2.8: Government cash flow.

Ingoing	**Outgoing**
Cash flow from public sector activities:	
$\sum_{i\in h,f,b} T^i$ Tax revenues	$\sum_{i\in h,f} S^i$ Subsidy payments
	$\sum_{i\in h,f} Tr^i$ Transfer payments
$\sum_h R^h$ Restitution payment	$\sum_h U^h$ Unemployment benefit payments
Cash flow from deficit financing:	
ΔD^g Monetization of deficit	$r^g N^g$ Bond coupon payments
$\Delta n_g^c p^g$ Bond financing of deficit	
Total income	Total expenses

Explanation:

- Revenues: taxes on corporate profits and household labour and capital income;
- Expenses: unemployment benefits, subsidies and transfers.
- Restitution payment: Unemployed workers receive a monthly unemployment benefit. If they become re-employed on a different day than they were fired on, they should restitute a proportion of the monthly benefit already received.

Central Bank balance sheet (c)

Table 2.9: Central Bank balance sheet.

Assets	**Liabilities**
M^c: liquidity/cash $\Delta M^g + \Delta M^b$ $-r^c M^b$ $+r^c L_b^c$ $+r^g n_g^c$	M_g^c: Governments liquidity $+\Delta M^g$ $+\Delta n_g^c p^g$
L_b^c: loans to banks $-\Delta D^b(-)$ $+\Delta D^b(+)$	M_b^c: Bank cash reserves ΔM^b $+\Delta D^b(+)$
L_g^c: loans to governments ΔD^g	M^c: fiat money (due to QE, bank bailouts) ΔD^g $-\Delta D^b(-)$ $+\Delta D^b(+)$
n_g^c: Government bonds (QE) $\Delta n_g^c p^g$	E^c: equity $-Div^c$

Table 2.10: Central Bank cash flow.

Ingoing	**Outgoing**
ΔM^g Gov cash deposits	ΔD^g Gov fiat money (standing facility)
ΔM^b Bank cash deposits	$\Delta D^b(+)$ Bank new ECB debt (standing facility)
$r^c L_b^c$ Interest from banks	$r^c M^b$ Interest to banks
$\Delta D^b(-)$ Bank debt reduction	$\Delta n_g^c p^g$ Gov bond purchases
$+r^g n_g^c$ Govmnt coupons	Div^c Dividend payment to govs
Total income	Total expenses

Explanation:

- With quantitative easing (QE), the central bank purchases government bonds using money it creates *ex nihilo* (fiat money), and so expands its balance sheet.

- Since the Central Bank is not allowed to make a profit, its revenues from government bonds and bank advances are distributed to the government in the form of a dividend.

- In case of multiple governments, the total dividend payment is equally divided among the governments.

Bank bailouts:

- New bank loans do not come from ECB liquidity/cash reserves, but are directly financed by creating fiat money. Therefore, ΔD^b is added to the asset L_b^c (loans to banks), and to the liabilities M^c (fiat money) and M_b^c (Bank cash reserves). The "missing" fourth item is found on the asset side of the bank's balance sheet, that receives the high powered money ($\Delta D^b(+)$).

Chapter 3

Consumption Goods Market Documentation

3.1 Messages

wage_payment

- Message from consumption goods producers to households. It contains the monthly wage that an individual household earns in the current production cycle.

quality_price_info_1

- Message from malls to household. Before a household starts her weekly shopping at the mall, she gets information from the mall about the range goods available at the beginning of the current day and the corresponding prices.
- Due to the asynchronized consumption of the of households the mall sends this message every day.

quality_price_info_2

- Message from malls to household. If a household was rationed in the regular shopping she has the opportunity to pass a second purchasing process. Since the offer has changed after the first purchasing loop the mall sends a updated information message.

update_mall_stock

- Message sent from firms to malls. After the production the output is distributed among the malls. The update_mall_stock message contains the delivery volume for an individual mall.
- The mall uses the information in order to update the current mall stock.

consumption_request_1

- This message is sent from households to malls. After a household has made her purchasing decision she sends the desired quantity of the selected consumption good to the mall.
- Before starting to serve the households the malls collect all orders first.

accepted_consumption_1

- Message sent from malls to households containing the amount of the consumption good that an individual household gets from the mall.
- After the mall has collected all consumption requests it checks whether the demand for each good can be satisfied without rationing. In case of excess demand the accepted delivery quantities are lower than the desired quantities.

consumption_request_2

- This message (sent from households to malls) contains the requested amount of the good that is selected in a second consumption decision making in case of rationing in the first purchasing process.

accepted_consumption_2

- This message contains the accepted consumption quantity if a household has been rationed in the first consumption loop.

sales

- The message is sent from malls to consumption goods producing firms at the end of each day. It contains information concerning the daily sales and whether or not the mall stock is sold out.

specific_skill_update

- This message is sent from firms to its employees. It contains the current value of the productivity of firm's capital stock, which is needed for the specific skill adaptation.

3.2 Firm

3.2.1 Activities

The (consumption goods producing) firms performs the following activities at the consumption goods market:

- The firm computes the planned production quantity.
- It determines the required input factors for producing the planned output.
- After passing the factor markets it produces and distributes the output among the malls.

3.2.2 Functions

Firm_calc_production_quantity

In this function, the firms compute the planned production quantity and the planned delivery volumes for the malls. The determination of the planned delivery volume of firm i for mall r is based on a standard inventory rule with uncertain demand. The inventory rule is a classical newsboy problem with a critical (inventory) stock $Y_{i,r,t}$ up to which the mall is refilled at the beginning of each selling cycle. The demand is estimated using a standard linear regression model.

The (estimated) demand of firm i in mall r of the last τ periods is given by $\left\{\hat{D}_{i,r,t-\tau}, ..., \hat{D}_{i,r,t-1}\right\}$.

$Y_{i,r,t}$ is chosen such that the firm expects to be able to satisfy the market demand with some probability $1-\chi$, where χ is the stock-out probability. $Y_{i,r,t}$ is computed by

$$Y_{i,r,t} = \hat{a}_{i,r,t} + (\tau+1)\hat{b}_{i,r,t} + \bar{q}_{1-\chi} \cdot \sqrt{\hat{\delta}_{i,r,t}}, \tag{3.1}$$

where $\hat{a}$ and $\hat{b}$ are linear regression coefficients, $\hat{\delta}$ is the estimated demand variance, and $\bar{q}_{1-\chi}$ is the $1-\chi$-quantile of the standard normal distribution.

For $\hat{b}$ we have

$$\hat{b}_{i,r,t} = \frac{\tau \sum_{s=1}^{\tau} s\hat{D}_{i,r,t-\tau+s} - \frac{1}{2}(\tau(\tau+1))\hat{D}_{i,r,t-\tau+s}}{\frac{1}{6}(\tau^2(\tau+1)(2\tau+1)) - \frac{1}{4}(\tau^2(\tau+1)^2)}, \tag{3.2}$$

and for $\hat{a}$

$$\hat{a}_{i,r,t} = \frac{1}{\tau}\sum_{s=1}^{\tau} \hat{D}_{i,r,t-\tau+s} - \frac{1}{2}\hat{b}_{i,r,t}(\tau+1), \tag{3.3}$$

and for the variance

$$\delta^2 = \frac{1}{\tau-1}\sum_{s=1}^{\tau}(\hat{D}_{i,r,t-\tau+s} - (\hat{a}_{i,r,t} + s \cdot \hat{b}_{i,r,t}))^2. \tag{3.4}$$

```
//Compute the estimators for each mall

for(i=0; i< MALLS_SALES_STATISTICS.size; i++)
{
   sum_1=0;
   sum_2=0;

   for (j=0;j<FIRM_PLANNING_HORIZON; j++)
   {
     sum_1+= (FIRM_PLANNING_HORIZON + 1 -
       MALLS_SALES_STATISTICS.array[i].sales.array[j].period)*
       MALLS_SALES_STATISTICS.array[i].sales.array[j].sales;

     sum_2+=  MALLS_SALES_STATISTICS.array[i].sales.array[j].sales;
   }

   regressor = (FIRM_PLANNING_HORIZON * sum_1 -
    0.5*FIRM_PLANNING_HORIZON*(FIRM_PLANNING_HORIZON+1)*sum_2)/
    (1/6.0*pow(FIRM_PLANNING_HORIZON,2)*(FIRM_PLANNING_HORIZON+1)*
    (2*FIRM_PLANNING_HORIZON+1)-
    1/4.0*(pow(FIRM_PLANNING_HORIZON,2)*
    pow((FIRM_PLANNING_HORIZON+1),2)));

   intercept =  1/(1.0*FIRM_PLANNING_HORIZON)*sum_2 -
    0.5*regressor*(FIRM_PLANNING_HORIZON+1);

    variance = 0;

    for(j=0; j< FIRM_PLANNING_HORIZON; j++)
    {
     variance+=
       pow(MALLS_SALES_STATISTICS.array[i].sales.array[j].sales-
      (intercept+ (FIRM_PLANNING_HORIZON + 1 -
      MALLS_SALES_STATISTICS.array[i].sales.array[j].period)
      * regressor),2)/(FIRM_PLANNING_HORIZON-1);
    }
    for(k=0; k< LINEAR_REGRESSION_ESTIMATORS.size ;k++)
    {
     if(MALLS_SALES_STATISTICS.array[i].mall_id==
      LINEAR_REGRESSION_ESTIMATORS.array[k].mall_id)
    {
      LINEAR_REGRESSION_ESTIMATORS.array[k].intercept=intercept;
      LINEAR_REGRESSION_ESTIMATORS.array[k].regressor=regressor;
      LINEAR_REGRESSION_ESTIMATORS.array[k].variance=variance;
    }
  }
}

/*Setting the critical values*/
for(int i = 0; i < CURRENT_MALL_STOCKS.size;i++)
{
    for(int j = 0; j < LINEAR_REGRESSION_ESTIMATORS.size;j++)
    {
        if(CURRENT_MALL_STOCKS.array[i].mall_id==
         LINEAR_REGRESSION_ESTIMATORS.array[j].mall_id)
        {
           CURRENT_MALL_STOCKS.array[i].critical_stock =
             LINEAR_REGRESSION_ESTIMATORS.array[j].intercept
             + (1+FIRM_PLANNING_HORIZON)*
             LINEAR_REGRESSION_ESTIMATORS.array[j].regressor
             + QUANTIL_NORMAL_DISTRIBUTION*
             pow(LINEAR_REGRESSION_ESTIMATORS.array[j].variance,0.5);
        }
    }
}
```

From the newsboy rule it follows for the desired delivery volume for mall r :

$$\tilde{D}_{i,r,t} = \begin{cases} 0 & \text{if } SL_{i,r,t} \geq Y_{i,r,t} \\ Y_{i,r,t} - SL_{i,r,t} & \text{if } SL_{i,r,t} < Y_{i,r,t}, \end{cases} . \tag{3.5}$$

```
/*checking whether or not the current mall stocks are
below the critical values (sS-Rule). If this is
the case refill the stock up to the max stock */

for(int i = 0; i < CURRENT_MALL_STOCKS.size; i++)
{
   if(CURRENT_MALL_STOCKS.array[i].current_stock <=
      CURRENT_MALL_STOCKS.array[i].critical_stock)
   {
     /*If stocks are left at the beginning of a new production cycle
     * then firms produce the difference between these stocks and the
     * critical stock for this mall*/

     prod_vol = CURRENT_MALL_STOCKS.array[i].critical_stock -
       CURRENT_MALL_STOCKS.array[i].current_stock;

     PLANNED_DELIVERY_VOLUME.array[i].mall_id =
       CURRENT_MALL_STOCKS.array[i].mall_id;
```

```
    PLANNED_DELIVERY_VOLUME.array[i].quantity  = prod_vol;
    production_volume = production_volume + prod_vol;
  }
  else
  /*If  stocks are higher than the critical value
   no production takes place for this mall*/
  {
    PLANNED_DELIVERY_VOLUME.array[i].mall_id=
    CURRENT_MALL_STOCKS.array[i].mall_id;
    PLANNED_DELIVERY_VOLUME.array[i].quantity= 0;
  }
}
```

Finally, in order to avoid a highly volatile production quantity, the planned total production volume is smoothed. It is computed by a linear combination of the sum of planned delivery volumes for the malls and the mean planned production quantities of the last N periods,

$$\tilde{Q}_{i,t} = \lambda \sum_{R} \tilde{D}_{i,r,t} + (1-\lambda) \cdot \frac{1}{N} \cdot \sum_{l=1}^{N} \tilde{Q}_{i,t-l}. \tag{3.6}$$

```
/*Smoothing of production quantity in
order to avoid high fluctuations*/

double mean_production_quantity=0;
for(int i = 0; i < LAST_PLANNED_PRODUCTION_QUANTITIES.size; i++)
{
  mean_production_quantity +=
    LAST_PLANNED_PRODUCTION_QUANTITIES.array[i];
}

  mean_production_quantity = mean_production_quantity/
  LAST_PLANNED_PRODUCTION_QUANTITIES.size;

  PLANNED_PRODUCTION_QUANTITY = LAMBDA*production_volume +
  (1-LAMBDA)*mean_production_quantity;

  //Set planned production value that
  //is retained in memory during the month:
  PLANNED_OUTPUT = PLANNED_PRODUCTION_QUANTITY;
```

Firm_calc_input_demands

The first action of this function is to update the range of investment goods vintages and the corresponding prices. Therefore, the firm reads a message sent by the IG firm that contains this information.

```
//Clean the array of vintages
for(i=TECHNOLOGY_VINTAGES.size -1; i>=0;i--)
{
  remove_adt_technology_vintages(&TECHNOLOGY_VINTAGES, i);
}

/*Getting information about the offered vintages*/
START_PRODUCTIVITY_MESSAGE_LOOP

  add_adt_technology_vintages(&TECHNOLOGY_VINTAGES
  ,productivity_message->cap_productivity,
  productivity_message->cap_good_price,0.0);
FINISH_PRODUCTIVITY_MESSAGE_LOOP
```

Consumption goods producers need physical capital and labor to produce the consumption goods. A firm i has a capital stock $K_{i,t}$ that is composed of different vintages of the production technology $\mathbb{V} = \{V\}_1^{Vmax}$,

$$K_{i,t} = \sum_{V=1}^{Vmax} K_{i,t}^{V}. \tag{3.7}$$

The accumulation of physical capital by a consumption goods producer follows

$$K_{i,t+1} = \sum_{V=1}^{Vmax} (1-\delta) K_{i,t}^{V} + \sum_{V=1}^{Vmax} I_{i,t}^{V} \tag{3.8}$$

where δ is the depreciation rate and $I_{i,t}^{V} \geq 0$ is the gross investment in vintage V.

```
/*Depreciation of the capital stock*/
double depreciation;

TOTAL_CAPITAL_DEPRECIATION_UNITS=0;
TOTAL_VALUE_CAPITAL_STOCK=0;
TOTAL_UNITS_CAPITAL_STOCK=0;
EFFECTIVE_CAPITAL_STOCK = 0.0;

for(i=0;i<CAPITAL_STOCK_VINTAGES.size;i++)
{
  depreciation = CAPITAL_STOCK_VINTAGES.array[i].amount
  *DEPRECIATION_RATE;

  TOTAL_CAPITAL_DEPRECIATION_UNITS += depreciation;
  CAPITAL_STOCK_VINTAGES.array[i].amount-=depreciation;
  TOTAL_UNITS_CAPITAL_STOCK+=
    CAPITAL_STOCK_VINTAGES.array[i].amount;

  EFFECTIVE_CAPITAL_STOCK +=
    CAPITAL_STOCK_VINTAGES.array[i].amount*
    CAPITAL_STOCK_VINTAGES.array[i].productivity;

  for(j=0;j<TECHNOLOGY_VINTAGES.size;j++)
  {
  if(CAPITAL_STOCK_VINTAGES.array[i].productivity==
    TECHNOLOGY_VINTAGES.array[j].productivity)
    {
    TOTAL_CAPITAL_DEPRECIATION_VALUE += ^
      TECHNOLOGY_VINTAGES.array[j].price*depreciation;

    TOTAL_VALUE_CAPITAL_STOCK+=
      TECHNOLOGY_VINTAGES.array[j].price*
        CAPITAL_STOCK_VINTAGES.array[i].amount;

    }
  }
}
```

The production technology in the consumption goods sector is represented by a Leontief type production function with complementarities between the qualities of the different vintages of the investment good and the specific skill level of employees for using these types of technologies. Vintages are deployed for production in descending order by using the best vintage first. For each vintage the effective productivity is determined by the minimum of its productivity and the average level of relevant specific skills of the workers. Accordingly, output for a consumption goods producer is given by

$$Q_{i,t} = \sum_{V=1}^{Vmax} \min\left[K_{i,t}^{V}, \max\left[0, L_{i,t} - \sum_{k=V+1}^{Vmax} K_{i,t}^{k}\right]\right] \cdot \min\left[A^{V}, B_{i,t}\right], \tag{3.9}$$

where A^V is the productivity of vintage V and $B_{i,t}$ denotes the average specific skill level in firms.

Let $\tilde{Q}_{i,t}$ be the planned production quantity of firm i in t and $\hat{Q}_{i,t}$ the feasible output that can be produced with the current capital stock. This potential output is computed according to

$$\hat{Q}_{i,t} = \sum_{V=1}^{Vmax} (1-\delta) K_{i,t}^{V} \cdot \min\left[A^{V}, B_{i,t}\right]. \tag{3.10}$$

Two cases have to be considered for the factor demand determination:

1. If $\hat{Q}_{i,t} \geq \tilde{Q}_{i,t}$: In that case the desired output can be produced with the current capital stock and no additional investments are necessary. We have $I_{i,t} = 0$ and the labor input is computed by taking the labor productivity of the last month into account:

$$\tilde{L}_{i,t} = \tilde{Q}_{i,t} \cdot \frac{L_{i,t-1}}{Q_{i,t-1}.} \tag{3.11}$$

2. If $\hat{Q}_{i,t} < \tilde{Q}_{i,t}$: Here we have positive investments $I_{i,t} > 0$; the amount depends on the outcome of the vintage choice (see next section). If V is the selected vintage, the investment volume is

$$I_{i,t} = \frac{\tilde{Q}_{i,t} - \hat{Q}_{i,t}}{\min\left[A^{V}, B_{i,t}\right]} \tag{3.12}$$

and the labor demand

$$\tilde{L}_{i,t} = K_{i,t-1}(1-\delta) + I_{i,t}. \tag{3.13}$$

```
/*------------Determination of input needs:-----------------------*/

/*1. Compute the feasible output given the capital stock*/

double feasible_output=0.0;
int employees_needed_last_month;

for(i=0;i<CAPITAL_STOCK_VINTAGES.size;i++)
{
  feasible_output+=CAPITAL_STOCK_VINTAGES.array[i].amount*
  min(CAPITAL_STOCK_VINTAGES.array[i].productivity
    ,MEAN_SPECIFIC_SKILLS);
}

/*2. Check if additional investments are necessary*/
if(feasible_output >= PLANNED_PRODUCTION_QUANTITY)
{
  DEMAND_CAPITAL_STOCK=0.0;
  VINTAGE_CHOICE_TAKEN = 0;

  if(NO_EMPLOYEES>0)
  {
    employees_needed_last_month  = NO_EMPLOYEES;
    EMPLOYEES_NEEDED =
      round_double_to_int(PLANNED_PRODUCTION_QUANTITY/OUTPUT *
      employees_needed_last_month);
  }else
  {
    EMPLOYEES_NEEDED
      round_double_to_int(PLANNED_PRODUCTION_QUANTITY/
      MEAN_SPECIFIC_SKILLS);
  }

  NEEDED_CAPITAL_STOCK = EMPLOYEES_NEEDED;
}else
{
  /*Technology choice*/
  /*For each vintage compute...*/

  VINTAGE_CHOICE_TAKEN = 1;

  double sum_eff_productivites[TECHNOLOGY_VINTAGES.size];
  double productivity_price_ratio[TECHNOLOGY_VINTAGES.size];

  double specific_skills;

  for(i=0;i<TECHNOLOGY_VINTAGES.size;i++)
  {
  /* ...the sum of discounted effective productivities.
  This means the min of the productivity of the capital good
  and the mean specific skills where the
  later converges to the A_i,t */

  specific_skills = MEAN_SPECIFIC_SKILLS;
  sum_eff_productivites[i] = 0;

  for(j=0;j<24;j++)
  {
    /*Update the specific skill: depends on the actual specific skill,
    the gap between the actual specific skills and the actual
    productivity of the employer,
    and the general skill which determines the speed of
    closing the this gap.*/

    specific_skills = max(specific_skills, specific_skills+
      (TECHNOLOGY_VINTAGES.array[i].productivity-specific_skills)
      *((1-pow(0.5,1/(20+0.25*(AVERAGE_G_SKILL-1)*(4-20))))));

    sum_eff_productivites[i]+= pow(1/(1+DISCONT_RATE),j)*
      min(TECHNOLOGY_VINTAGES.array[i].productivity,specific_skills);
    }

    productivity_price_ratio[i]= sum_eff_productivites[i]/
      TECHNOLOGY_VINTAGES.array[i].price;
    TECHNOLOGY_VINTAGES.array[i].sum_effective_productivities =
      sum_eff_productivites[i];
```

```
  }
  /*A Logit model used for vintage choice*/

  double sum=0;
  double logit[TECHNOLOGY_VINTAGES.size];

  if(DAY>=TRANSITION_PHASE)
  {
    /*Summing for logit denominator*/
    for(i=0; i<TECHNOLOGY_VINTAGES.size; i++)
    {
      sum += exp(GAMMA_LOGIT_VINTAGE_CHOICE*log(productivity_price_ratio[i]));
    }

    /*Computing the logits*/
    for(i=0; i<TECHNOLOGY_VINTAGES.size; i++)
    {
      logit[i]= exp(GAMMA_LOGIT_VINTAGE_CHOICE*log(productivity_price_ratio[i]))/sum;
      logit_for_print_debug[i]=logit[i];
    }

    /*random generator and selection of logit*/
    double rnd_number = (double)random_int(0,100)/100.0;

    for(i=0; i<TECHNOLOGY_VINTAGES.size; i++)
    {
      if(rnd_number<logit[i])
      {
        VINTAGE_SELECTED = i;
        break;
      }else
      {
        if(i<TECHNOLOGY_VINTAGES.size -1)
        logit[i+1]+=logit[i];
      }
    }
  }

  /*Imput factor determination*/

  DEMAND_CAPITAL_STOCK = (PLANNED_PRODUCTION_QUANTITY-feasible_output)
    /min(TECHNOLOGY_VINTAGES.array[VINTAGE_SELECTED].productivity,
    MEAN_SPECIFIC_SKILLS);
  EMPLOYEES_NEEDED = round_double_to_int(TOTAL_UNITS_CAPITAL_STOCK
   + DEMAND_CAPITAL_STOCK);
  NEEDED_CAPITAL_STOCK = EMPLOYEES_NEEDED;
}
```

Firm_calc_production_quantity_2

After passing the firm financial management role and, if an external financing is needed, entering the credit and financial market, the firm has calculated its total liquidity requirements and how much of these are covered by the internal and externally obtained resources. Because financial obligations have a higher priority than achieving the production plan, the firm has to reduce the production plan if the resources are not sufficient to satisfy both.

The firm compares the available resources for production with the money that is needed for the production plan. If the available resources do not cover the planned expenditures the firm has to reduce the production quantity decrementally as long as the financial requirements are sufficient.

$$\tilde{Q}_{i,t} = \begin{cases} \tilde{Q}_{i,t}, & \text{if } FinProd^{NEEDED}_{i,t}(\tilde{Q}) < FinProd_{i,t} \\ \tilde{Q}_{i,t} \cdot (1 - ADAP\_PROD), & \text{else.} \end{cases} \tag{3.14}$$

```
double decrement;
double diff;
diff = PLANNED_PRODUCTION_QUANTITY;

//Here we set a fraction of the planned production quantity
decrement = ADAPTION_PRODUCTION_VOLUME_DUE_TO_INSUFFICIENT_FINANCES*
PLANNED_PRODUCTION_QUANTITY;

if( FINANCIAL_RESOURCES_FOR_PRODUCTION>0.0)
{
  while (PLANNED_PRODUCTION_COSTS > FINANCIAL_RESOURCES_FOR_PRODUCTION)
  {
    PLANNED_PRODUCTION_QUANTITY -= decrement;
    Firm_calc_input_demands_2();
  }
}else
{
  PLANNED_PRODUCTION_QUANTITY=0.0;
  Firm_calc_input_demands_2();
}

//Compute the diff
diff -= PLANNED_PRODUCTION_QUANTITY;

//Set planned production value that is retained in memory during the month:
PLANNED_OUTPUT = PLANNED_PRODUCTION_QUANTITY;
```

Firm_calc_input_demands_2

In this function the firm recalculates the input factor demand if the production volume has to be decremented in case of financial shortcomings. The function is repetitively called by Firm_calc_production_quantity_2 at each time when the firm decrements the output. The source code of this function is very close to the code of Firm_calc_input_demands.

Firm_send_capital_demand

This function sends a message to the capital goods producer containing the capital good demand $K^{demand}_{i,t}$.

```
if ( (NEEDED_CAPITAL_STOCK > 0) && (NEEDED_CAPITAL_STOCK >
TOTAL_UNITS_CAPITAL_STOCK))
{
  add_capital_good_request_message(ID,DEMAND_CAPITAL_STOCK);
}
```

Firm_receive_capital_goods

In this function the consumption goods producing firm receives the capital goods delivery from the investment goods producer. It reads a message containing the amount of delivered capital goods $\bar{I}^V_{i,t}$ and it updates the capital stock accordingly:

$$K^V_{i,t} = \hat{K}^V_{i,t} + \bar{I}^V_{i,t}, \tag{3.15}$$

The value of the new capital stock is computed by

$$K^{Value}_{i,t} = \sum_{v=1}^{V_{max}} p^{Inv}_{v,t} \cdot K^v_{i,t}. \tag{3.16}$$

The nominal investment in the physical capital for that production period is the capital bill $I_{i,t} = \bar{I}^V_{i,t} \cdot p^{Inv}_{v,t}$. The acquisition costs of the investment are distributed among the length of the repayment period for external financing, T^{LOANS}, regardless if the investment was externally or internally financed. This means that in the next T^{LOANS} periods a fraction $\frac{1}{T^{LOANS}}$ of the investment $I_{i,t}$ is apportioned as capital costs for each period.

```
CAPITAL_COSTS = 0.0;
// EFFECTIVE_INVESTMENTS is set to 0 in Firm_calc_input_demands

START_CAPITAL_GOOD_DELIVERY_MESSAGE_LOOP

/*Determine the weighted average productivity of the total capital stock*/
/*Update of productivity*/
/*Update of current value of capital stock*/

 for(i=0;i<CAPITAL_STOCK_VINTAGES.size;i++)
{
  if(capital_good_delivery_message->productivity==
     CAPITAL_STOCK_VINTAGES.array[i].productivity)
  {
     CAPITAL_STOCK_VINTAGES.array[i].amount +=
        capital_good_delivery_message->capital_good_delivery_volume;

     TOTAL_UNITS_CAPITAL_STOCK+=capital_good_delivery_message->
        capital_good_delivery_volume;

     EFFECTIVE_INVESTMENTS = capital_good_delivery_message->
        capital_good_delivery_volume *
     CAPITAL_STOCK_VINTAGES.array[i].productivity;

     TOTAL_VALUE_CAPITAL_STOCK+=
        capital_good_delivery_message->capital_good_price*
        capital_good_delivery_message->capital_good_delivery_volume;
     EFFECTIVE_CAPITAL_STOCK += capital_good_delivery_message->
        capital_good_delivery_volume*CAPITAL_STOCK_VINTAGES.array[i].productivity;
     flag=1;
     break;
  }
}

/*If new capital vintage is purchased add new vintage to array*/
if(flag==0)
{
  add_adt_capital_stock_vintages(&CAPITAL_STOCK_VINTAGES,
    capital_good_delivery_message->capital_good_delivery_volume ,
     TECHNOLOGY_VINTAGES.array[VINTAGE_SELECTED].productivity);

  TOTAL_UNITS_CAPITAL_STOCK+=capital_good_delivery_message
    ->capital_good_delivery_volume;

  /*Sorting the array from low to high*/
  for(i=0;i<CAPITAL_STOCK_VINTAGES.size;i++)
  {
    for(j=0;j<i;j++)
    {
    if(CAPITAL_STOCK_VINTAGES.array[i].productivity
      <CAPITAL_STOCK_VINTAGES.array[j].productivity)
      {
        double x =CAPITAL_STOCK_VINTAGES.array[i].productivity;
        double y =CAPITAL_STOCK_VINTAGES.array[i].amount;
        CAPITAL_STOCK_VINTAGES.array[i].productivity=
          CAPITAL_STOCK_VINTAGES.array[j].productivity;
        CAPITAL_STOCK_VINTAGES.array[i].amount= CAPITAL_STOCK_VINTAGES.array[j].amount;
        CAPITAL_STOCK_VINTAGES.array[j].productivity=x;
        CAPITAL_STOCK_VINTAGES.array[j].amount=y;
      }
    }
  }

  /*Computing the capital bill*/
  CAPITAL_COSTS += capital_good_delivery_message
    ->capital_good_delivery_volume* TECHNOLOGY_VINTAGES.array[VINTAGE_SELECTED].price;

  capital_good_price = capital_good_delivery_message->capital_good_price;
  capital_good_delivery_volume = capital_good_delivery_message
    ->capital_good_delivery_volume;

  FINISH_CAPITAL_GOOD_DELIVERY_MESSAGE_LOOP

  if(CAPITAL_COSTS>0.0)
  {
  add_financing_capital(&CAPITAL_FINANCING,CAPITAL_COSTS/
    CONST_INSTALLMENT_PERIODS,CONST_INSTALLMENT_PERIODS);
}
```

Firm_execute_production

This function computes the realized output of the firm i, $Q_{i,t}$. If the planned output $\tilde{Q}_{i,t} \neq 0$, the output is computed according to the production function

$$Q_{i,t} = \sum_{V=1}^{Vmax} \min\left[K_{i,t}^{V}, \max\left[0, L_{i,t} - \sum_{k=V+1}^{Vmax} K_{i,t}^{k}\right]\right] \cdot \min\left[A^{V}, B_{i,t}\right] \tag{3.17}$$

otherwise the production quantity is $Q_{i,t} = 0$. Additionally, the productivity of the deployed capital stock is computed. We have:

$$A_{i,t} = \frac{\sum_{i=1}^{V_{max}} A^{v} \cdot \tilde{K}_{i,t}^{v}}{\sum_{i=1}^{V_{max}} \tilde{K}_{i,t}^{v}}, \tag{3.18}$$

where $\tilde{K}_{i,t}^{v}$ is the amount of capital units of vintage v deployed for the production in t.

```
if(PLANNED_PRODUCTION_QUANTITY != 0)
{

  double sum_already_used_vintages=0.0;
  double technology =0.0;
  PRODUCTION_QUANTITY=0.0;

  for(i=CAPITAL_STOCK_VINTAGES.size -1;i>=0;i--)
  {
    /*Here the production quantity is computed*/
    PRODUCTION_QUANTITY+= min(CAPITAL_STOCK_VINTAGES.array[i].amount,
      max(0,NO_EMPLOYEES-sum_already_used_vintages))
      *min(CAPITAL_STOCK_VINTAGES.array[i].productivity ,MEAN_SPECIFIC_SKILLS);

    /*Here we compute the productivity of the used capital stock.
    This is sent to workers thus the specific skills converges to this value*/

    technology += min(CAPITAL_STOCK_VINTAGES.array[i].amount,max(0,NO_EMPLOYEES
      -sum_already_used_vintages))
    *CAPITAL_STOCK_VINTAGES.array[i].productivity;

    /*Sum up the used vintages:*/
    sum_already_used_vintages+=min(CAPITAL_STOCK_VINTAGES.array[i].
    amount,max(0,NO_EMPLOYEES-sum_already_used_vintages));
  }
  if(sum_already_used_vintages>0)
  {
    technology = technology/sum_already_used_vintages;
    TECHNOLOGY = technology;
  }
  if(TOTAL_UNITS_CAPITAL_STOCK)
  {
    UTILIZATION_CAPACITY = sum_already_used_vintages/TOTAL_UNITS_CAPITAL_STOCK;
  }else
  {
    UTILIZATION_CAPACITY=0.0;
  }

}
else
{
  PRODUCTION_QUANTITY=0.0;
  UTILIZATION_CAPACITY=0.0;

}
```

Firm_calc_pay_costs

This function computes and pays the production costs and sets the price of the good by applying a mark-up pricing rule.

1. The firm computes the total labor costs as the sum of the individual worker wages,

$$C_{i,t}^{Lab} = \sum_{\forall k \in W_i} w_{k,t} \tag{3.19}$$

   and sends a message to the workers with the individual wage payment.

```
LABOUR_COSTS=0.0;

for(int i=0; i<EMPLOYEES.size; i++)
{
  LABOUR_COSTS += EMPLOYEES.array[i].wage;

  add_wage_payment_message(ID,
    EMPLOYEES.array[i].id,EMPLOYEES.array[i].wage,
    TECHNOLOGY, MEAN_SPECIFIC_SKILLS);
}
```

2. Capital cost accounting: Determining the capital costs $I_{i,t}$ and the the costs of using firm's capital stock, $C^{Cap}_{i,t}$. These calculatory capital costs are the sum of $\frac{1}{T^{LOANS}}$th of the investments carried out within the last T^{LOANS} periods (see also function Firm_receive_capital_goods),

$$C^{Cap}_{i,t} = \sum_{l=0}^{t-T^{LOANS}} \frac{1}{T^{LOANS}} \cdot I_{t-l}. \tag{3.20}$$

```
add_pay_capital_goods_message(ID,CAPITAL_COSTS);

if(PRODUCTION_QUANTITY!=0 )
{
  CALC_CAPITAL_COSTS = 0;
  for(int i = 0; i<CAPITAL_FINANCING.size; i++)
  {
    if(CAPITAL_FINANCING.array[i].nr_periods_before_repayment==0)
    {
      remove_financing_capital(&CAPITAL_FINANCING, i);
      i--;
    }else
    {
      CAPITAL_FINANCING.array[i].nr_periods_before_repayment--;
      CALC_CAPITAL_COSTS+= CAPITAL_FINANCING.array[i].financing_per_month;
    }
  }
}
```

3. The labour costs, nominal investments and calculatory capital costs are used to compute the total amount of transactions related to production in that period, the production costs and unit costs. By multiplying the unit costs with the mark-up, the firm sets the price of its consumption good. The total transactions or expenditures related to production are

$$Exp^{Prod}_{i,t} = C^{Lab}_{i,t} + I_{i,t}, \tag{3.21}$$

the total production costs are

$$C_{i,t} = C^{Lab}_{i,t} + C^{Cap}_{i,t}. \tag{3.22}$$

The unit costs additionally take into account the interest payments $Int_{i,t}$, thus it holds

$$c_{i,t} = (C_{i,t} + Int_{i,t})/Q_{i,t}. \tag{3.23}$$

For the price we have then

$$p_{i,t} = (1 + \mu)c_{i,t}. \tag{3.24}$$

```
UNIT_COSTS=(LABOUR_COSTS + CALC_CAPITAL_COSTS + TOTAL_INTEREST_PAYMENT)
/ PRODUCTION_QUANTITY;

PRICE_LAST_MONTH = PRICE;
PRICE = UNIT_COSTS*(1 + MARK_UP);
    }

PRODUCTION_COSTS = CAPITAL_COSTS + LABOUR_COSTS;

CALC_PRODUCTION_COSTS= LABOUR_COSTS + CALC_CAPITAL_COSTS;

PAYMENT_ACCOUNT -= PRODUCTION_COSTS;

remove_double(&LAST_PLANNED_PRODUCTION_QUANTITIES,0);
add_double(&LAST_PLANNED_PRODUCTION_QUANTITIES,PLANNED_PRODUCTION_QUANTITY);
```

Firm_send_goods_to_mall

In this function the firm distributes the output among the malls. It sends a messages to the malls containing the volume and the current price. Since the realized output does not necessarily correspond to the planned quantity the firms determines the actual delivery quantities proportionally to the intended volumes:

$$D_{i,r,t} = \frac{Q_{i,t}}{\sum_R \tilde{D}_{i,r,t}} \cdot \tilde{D}_{i,r,t}. \tag{3.25}$$

```
NOMINAL_EXPORTS = 0.0;
REAL_EXPORTS = 0.0;
double delivery_volume=0;

for(int i=0; i<PLANNED_DELIVERY_VOLUME.size; i++)
{
  delivery_volume+=PLANNED_DELIVERY_VOLUME.array[i].quantity;
}
for(int i = 0; i < PLANNED_DELIVERY_VOLUME.size; i++)
{
  for(int j = 0; j < DELIVERY_VOLUME.size; j++)
  {
    if(DELIVERY_VOLUME.array[j].mall_id ==
      PLANNED_DELIVERY_VOLUME.array[i].mall_id)
    {
      /*If planned prod vol > realized prod vol -> curtain the delivery vol*/
      if(delivery_volume>PRODUCTION_QUANTITY)
      {
        DELIVERY_VOLUME.array[j].quantity =
        PRODUCTION_QUANTITY / delivery_volume *
        PLANNED_DELIVERY_VOLUME.array[i].quantity;
      }else /*otherwise planned = realized del vol*/
          if(PRODUCTION_QUANTITY > delivery_volume &&
            delivery_volume !=0)
          {
            DELIVERY_VOLUME.array[j].quantity = PRODUCTION_QUANTITY /
            delivery_volume*PLANNED_DELIVERY_VOLUME.array[i].quantity;
          }else if(PRODUCTION_QUANTITY > delivery_volume &&
            delivery_volume ==0)
          {
                DELIVERY_VOLUME.array[j].quantity =
                PRODUCTION_QUANTITY/(DELIVERY_VOLUME.size);
            }
            else
            {
                DELIVERY_VOLUME.array[j].quantity =
                PLANNED_DELIVERY_VOLUME.array[i].quantity;
            }
            DELIVERY_VOLUME.array[j].quality=
                PLANNED_DELIVERY_VOLUME.array[j].quality;
            DELIVERY_VOLUME.array[j].price=
                PLANNED_DELIVERY_VOLUME.array[j].price;
            add_update_mall_stock_message(DELIVERY_VOLUME.array[j].mall_id ,ID,
                DELIVERY_VOLUME.array[j].quantity ,QUALITY,PRICE);
            }
        }
    }
}
```

Firm_calc_revenue

The firm receives messages containing information on the daily revenues at each mall. This data is used to compute monthly mall sales, which is required for the production planning.

```
REVENUE_PER_DAY=0.0;
TOTAL_SOLD_QUANTITY=0.0;

/*calc the daily revenue and sum up the monthly revenue*/
START_SALES_MESSAGE_LOOP

for(int i=0; i< SOLD_QUANTITIES.size; i++)
{
    if(sales_message->mall_id == SOLD_QUANTITIES.array[i].mall_id)
    {
        SOLD_QUANTITIES.array[i].sold_quantity +=
        sales_message->revenue/PRICE;
        REVENUE_PER_DAY += sales_message->revenue;
        CUM_REVENUE += sales_message->revenue;
        TOTAL_SOLD_QUANTITY+=sales_message->revenue/PRICE;
    }
}
//Update mall stocks
for(int i=0; i< CURRENT_MALL_STOCKS.size; i++)
{
    if(sales_message->mall_id == CURRENT_MALL_STOCKS.array[i].mall_id)
    {
        CURRENT_MALL_STOCKS.array[i].current_stock= sales_message->current_stock;
    }
}
FINISH_SALES_MESSAGE_LOOP

PAYMENT_ACCOUNT += REVENUE_PER_DAY
```

Firm_compute_sales_statistics

For the inventory rule applied in Firm_calc_production_quantity, firms need for each mall the sold quantities of the last T periods. For that purpose they remove the sold quantity of period

$t - T$ and store the current sold quantity. If a mall stock is completely sold out then in order to estimate the real demand the sold quantity is adapted by a percentage of the sales of the last month.

```
int remove_index;

for(int j=0; j < MALLS_SALES_STATISTICS.size;j++)
{
  for(int k = 0; k < MALLS_SALES_STATISTICS.array[j].sales.size; k++)
  {
    if(MALLS_SALES_STATISTICS.array[j].sales.array[k].period==
      FIRM_PLANNING_HORIZON)
    {
      remove_index = k;
      remove_data_type_sales(&(MALLS_SALES_STATISTICS.array[j].sales),
        remove_index);
      k--;
     }else
     {
       MALLS_SALES_STATISTICS.array[j].sales.array[k].period++;
     }
  }
}
for(int i=0; i< SOLD_QUANTITIES.size;i++)
{
  for(int j=0; j<MALLS_SALES_STATISTICS.size; j++)
  {
    if(MALLS_SALES_STATISTICS.array[j].mall_id ==
      SOLD_QUANTITIES.array[i].mall_id)
    {
       if(SOLD_QUANTITIES.array[i].stock_empty==0)
       {
         add_data_type_sales(&(MALLS_SALES_STATISTICS.array[j].sales), 1 ,
           SOLD_QUANTITIES.array[i].sold_quantity);
       }
       else
       {
          add_data_type_sales(&(MALLS_SALES_STATISTICS.array[j].sales), 1 ,
             SOLD_QUANTITIES. array[i].sold_quantity*
             (1 + ADAPTION_DELIVERY_VOLUME));
       }
       SOLD_QUANTITIES.array[i].sold_quantity=0;
       SOLD_QUANTITIES.array[i].stock_empty=0;
    }
  }
}
```

Firm_update_specific_skills_of_workers

In this function the firm updates the employee array concerning the individual specific skills.

```
START_SPECIFIC_SKILL_UPDATE_MESSAGE_LOOP

for(int i=0; i<EMPLOYEES.size;i++)
{
  if(specific_skill_update_message->id==EMPLOYEES.array[i].id)
  {
    EMPLOYEES.array[i].specific_skill=
    specific_skill_update_message->specific_skills;
  }
}
FINISH_SPECIFIC_SKILL_UPDATE_MESSAGE_LOOP
```

3.3 Household

3.3.1 Activities

The activities of a household in the consumption goods market are:

- The household determines the monthly and weekly consumption budgets.
- The household visits a outlet mall in order to

    1. Collect information on the range of provided goods and their prices
    2. decide and purchase a good for the weekly consumption.

3.3.2 Functions

Household_determine_consumption_budget

In this function the household computes the monthly consumption budget. Households have a target wealth income ratio and if the current income wealth ratio matches the target value then the consumption budget equals the mean income of a certain number of previous periods. In case of a deviation of the current compared to the target ratio, the consumption budget deviates from the mean income as long as the two values are coincident.

```

/*Compute the wealth income ratio*/

WEALTH_INCOME_RATIO_ACTUAL = WEALTH/ MEAN_NET_INCOME;

asset_wealth = ASSETSOWNED.units*ASSETSOWNED.lastprice;
CONSUMPTION_BUDGET = MEAN_NET_INCOME +
  CARROL_CONSUMPTION_PARAMETER*PAYMENT_ACCOUNT +
  TRADING_ACTIVITY * CARROL_CONSUMPTION_PARAMETER *
  (asset_wealth - WEALTH_INCOME_RATIO_TARGET*MEAN_NET_INCOME);

/* Simple case: no selling of assets is possible, so do not consume
more than your liquid assets */
if ((TRADING_ACTIVITY==0)&&(CONSUMPTION_BUDGET > PAYMENT_ACCOUNT))
  CONSUMPTION_BUDGET = PAYMENT_ACCOUNT;

CONSUMPTION_BUDGET_IN_MONTH = CONSUMPTION_BUDGET;
```

A household goes shopping once a week but the consumption budget is determined after a household has received the wage/ unemployment benefit. The total monthly budget is equally distributed among the four weeks of a month such that the household can spend the same amount of money in each week.

```

WEEKLY_BUDGET = CONSUMPTION_BUDGET/4;
WEEK_OF_MONTH = 4;
```

Household_rank_and_buy_goods_1

The household goes shopping to the closest located mall, where the household gets an overview concerning the range of provided goods at the mall and the prices.

```
/*Household reads quality price info messages sent by malls   */
START_QUALITY_PRICE_INFO_1_MESSAGE_LOOP

add_mall_quality_price_info(&mall_quality_price_info_list ,
quality_price_info_1_message->mall_id ,
 quality_price_info_1_message->firm_id ,
 quality_price_info_1_message->mall_region_id ,
quality_price_info_1_message->quality , quality_price_info_1_message->price ,
quality_price_info_1_message->available );

FINISH_QUALITY_PRICE_INFO_1_MESSAGE_LOOP
```

The household tries to spend the weekly budget for one of the provided goods. The decision making relies on a logit model which takes the prices of the goods as an important factor of households selection decision into account. The strength of the price sensitivity is thereby represented by a parameter. The probability for choosing good i from the set of provided goods $G_{r,week_t}$ is

$$\text{Prob}_{k,i,t} = \frac{\exp(\lambda_k^{Cons} \cdot (-1)\ln(p_{i,t}))}{\sum_{j\in G_{r,week_t}} \exp(\lambda_k^{Cons} \cdot (-1)\ln(p_{j,t}))}. \tag{3.26}$$

```
/*Sum of weighted exponents of quality price ratios*/
    for(i = 0; i < mall_quality_price_info_list.size;i++)
    {
      sum_weighted_qual_pric_ratios += (mall_quality_price_info_list.array[i]
      .available) * exp(log(mall_quality_price_info_list.array[i].price)*GAMMA);
    }



    /* Compute logits and add on temporary logit array   */
    for(i = 0; i < mall_quality_price_info_list.size;i++)
    {
      logit = (mall_quality_price_info_list.array[i].available) *
      exp(log(mall_quality_price_info_list.array[i].price)*GAMMA) /
      sum_weighted_qual_pric_ratios;

      logit = logit * 100;

      if(logit > 0)
      {
        add_logit_firm_id(&logit_firm_id_list , logit ,
        mall_quality_price_info_list.array[i].firm_id);
      }

    }


    if(sum_weighted_qual_pric_ratios > 0)
    {
      MALL_COMPLETELY_SOLD_OUT = 0;
      int random_number = random_int(0,100);
      j=0;
      int x =0, index_selected_good=j;
      for(j = 0; j < logit_firm_id_list.size;j++)
      {

        /*if random number <= logit then select the corresponding good */
        if((random_number < logit_firm_id_list.array[j].logit) && (x != 1))
        {
          ORDER_QUANTITY[0].firm_id = logit_firm_id_list.
          array[j].firm_id;

          x = 1;
          index_selected_good= j;
        }
        /*else sum logits and go to the next iteration step */
        else
        {
          if((j < logit_firm_id_list.size -1) )
          {
            logit_firm_id_list.array[j+1].logit =
            logit_firm_id_list.array[j+1].logit+
            logit_firm_id_list.array[j].logit;
          }
        }
      }
```

After the household has decided for one good she sends an order message to the mall.

$$OQ^1_{k,i,r,week_t} = \frac{B^{Cons}_{k,week_t}}{p_{i,t}}. \tag{3.27}$$

```
    /*This computes and stores the order quantity of the selected good and
    saves the price in memory */
    ORDER_QUANTITY[0].quantity = WEEKLY_BUDGET/ mall_quality_price_info_list
    .array[index_selected_good].price;

    ORDER_QUANTITY[0].price = mall_quality_price_info_list.
    array[index_selected_good].price;

    /*The consumption request message is sent */
    add_consumption_request_1_message(
    mall_quality_price_info_list.array[index_selected_good].mall_id ,ID,
    ORDER_QUANTITY[0].firm_id ,
    ORDER_QUANTITY[0].quantity);

  }
  else
  {
    EXPENDITURES=0;
    MALL_COMPLETELY_SOLD_OUT =1;
    ORDER_QUANTITY[0].quantity=0;
    ORDER_QUANTITY[0].price=0;
    ORDER_QUANTITY[0].firm_id=0;
  }
```

Household_receive_goods_read_rationing

In this function the household gets the quantity of the ordered consumption good which has been accepted by the malls. In case of an excess demand for the ordered good the mall has rationed the requested quantities and the received quantity is smaller than the ordered.

The expenditure for the consumption is subtracted from the weekly budget, such that in case of no rationing the left over budget is 0. Otherwise, the remaining budget is used for a second consumption loop where the household tries to spend the budget for another good.

```
EXPENDITURES = 0;

/*Household reads messages containing the realized consumption of the first round*/
START_ACCEPTED_CONSUMPTION_1_MESSAGE_LOOP

  RATIONED = accepted_consumption_1_message->rationed;

  /*Update of Budget  */
  WEEKLY_BUDGET = WEEKLY_BUDGET - accepted_consumption_1_message
    ->offered_consumption_volume * ORDER_QUANTITY[0].price;

  EXPENDITURES =accepted_consumption_1_message
    ->offered_consumption_volume * ORDER_QUANTITY[0].price;

  RECEIVED_QUANTITY[0].quantity = accepted_consumption_1_message
    ->offered_consumption_volume;

  RECEIVED_QUANTITY[0].firm_id = ORDER_QUANTITY[0].firm_id;

  FINISH_ACCEPTED_CONSUMPTION_1_MESSAGE_LOOP
```

Household_rank_and_buy_goods_2

If the household has been rationed in the first shopping loop, she tries to spend the remaining budget in a second loop. The purchasing decision is analog to the decision making in the first loop. First, the household updates the information concerning the availability of the goods.

```
/*The updated quality price message is read  */
    START_QUALITY_PRICE_INFO_2_MESSAGE_LOOP

add_mall_quality_price_info(&mall_quality_price_info_list ,
quality_price_info_2_message->mall_id ,
        quality_price_info_2_message->firm_id ,
        quality_price_info_2_message->mall_region_id ,
        quality_price_info_2_message->quality ,
        quality_price_info_2_message->price ,
        quality_price_info_2_message->available);

    FINISH_QUALITY_PRICE_INFO_2_MESSAGE_LOOP
```

Then, she chooses one good according to the same logit model as in function Household_rank_and_buy_goods_1.

```
/*Sum of weighted exponents of quality price ratios   */
    for(i = 0;i < mall_quality_price_info_list.size; i++)
    {
      sum_weighted_qual_pric_ratios +=(mall_quality_price_info_list
      .array[i].available) *
      exp(log(mall_quality_price_info_list.array[i].price)*GAMMA);
    }

    /*This computes the logits  */
    for(i = 0; i < mall_quality_price_info_list.size; i++)
    {
      logit = (mall_quality_price_info_list.array[i].available) *
      exp(log(mall_quality_price_info_list.array[i].price)*GAMMA) /
      sum_weighted_qual_pric_ratios;

      logit = logit*100;

      add_logit_firm_id(&logit_firm_id_list, logit,
      mall_quality_price_info_list.array[i].firm_id);
    }

    if(sum_weighted_qual_pric_ratios >0)
    {
      int random_number = random_int(0,100);
      j = 0;
      int x = 0, index_selected_good=j;

      for(j = 0; j < logit_firm_id_list.size;j++)
      {
        /*if random number <= logit then select the corresponding good */
        if((random_number < logit_firm_id_list.array[j].logit)
        && (x!=1))
        {
          ORDER_QUANTITY[1].firm_id = logit_firm_id_list
          .array[j].firm_id; //Selected Good

          x =1;
          index_selected_good= j;
        }
        /*else sum logits and go to next iteration step */
        else
        {
          if((j < logit_firm_id_list.size-1) )
          {
            logit_firm_id_list.array[j+1].logit =
            logit_firm_id_list.array[j+1].logit+
            logit_firm_id_list.array[j].logit;
          }
        }
      }

      /*This computes the order quantity  and store the price */
      ORDER_QUANTITY[1].quantity = WEEKLY_BUDGET/
      mall_quality_price_info_list.array[index_selected_good].price;

      ORDER_QUANTITY[1].price = mall_quality_price_info_list
      .array[index_selected_good].price;

      /*Sending the second consumption request message  */
      add_consumption_request_2_message(
      mall_quality_price_info_list.array[index_selected_good].mall_id,
      ID,ORDER_QUANTITY[1].firm_id,
      ORDER_QUANTITY[1].quantity);

    }
    else
    {
      ORDER_QUANTITY[1].quantity = 0;
      ORDER_QUANTITY[1].firm_id= 0;
      ORDER_QUANTITY[1].price= 0;
    }
```

Household_receive_goods_read_rationing_2

Here, a household gets the accepted consumption good delivery for the second loop, and computes the expenditures and the remaining budget.

```
if(RATIONED ==1)
  {
    /*Read the message about accepted consumption */
    START_ACCEPTED_CONSUMPTION_2_MESSAGE_LOOP

        RATIONED = accepted_consumption_2_message->rationed;

        RECEIVED_QUANTITY[1].quantity=
         accepted_consumption_2_message->offered_consumption_volume;

        RECEIVED_QUANTITY[1].firm_id =
        ORDER_QUANTITY[1].firm_id;

    FINISH_ACCEPTED_CONSUMPTION_2_MESSAGE_LOOP
  }
  else
  {
    RECEIVED_QUANTITY[1].quantity=0.0;
    RECEIVED_QUANTITY[1].firm_id =0;
  }

  WEEKLY_BUDGET = WEEKLY_BUDGET - RECEIVED_QUANTITY[1].quantity
  *ORDER_QUANTITY[1].price ;

  EXPENDITURES += RECEIVED_QUANTITY[1].quantity * ORDER_QUANTITY[1].price ;
```

Household_handle_leftover_budget

The purpose of this function is

1. to subtract the consumption expenditures from the payment account
2. to handle the left over budget if the household was rationed in both shopping loops. How the household proceeds, depends on the week of the months when the rationing occurs. In the last week of the month (i.e. at the next activation day the household determines the budget for the following month) there is no need for a special treatment since the left over money remains at the payment account and goes trough the wealth into the determination of the next monthly budget. For the other months, the left over budget is equally distributed among the remaining weekly budgets.

```
CONSUMPTION_BUDGET -= EXPENDITURES;
    MONTHLY_EXPENDITURES += EXPENDITURES;

  if(WEEK_OF_MONTH !=1)
    {
      PAYMENT_ACCOUNT -= EXPENDITURES;
      WEEK_OF_MONTH--;
      WEEKLY_BUDGET = CONSUMPTION_BUDGET / WEEK_OF_MONTH;
    }
    else
    {
      PAYMENT_ACCOUNT =PAYMENT_ACCOUNT - EXPENDITURES;
      WEEK_OF_MONTH--;
    }
    //set rationed back to zero:
    RATIONED = 0;

    add_bank_account_update_message(ID, BANK_ID, PAYMENT_ACCOUNT);
```

3.4 Mall

3.4.1 Activities

The mall activities at the consumption goods market are:

- regional market platform for the consumption goods.
- The malls store local inventories of all consumption goods firm. When a firm finishes its production, it distributes the output over all malls.
- The mall collects all consumption requests, and handles the sale of goods. If necessary it rations the consumption.
- It collects the revenues and sends them to the consumption goods firm.

3.4.2 Functions

Mall_update_mall_stock

At that day when consumption goods producers finish their production processes and deliver the goods to the malls, they send a message to the malls containing the individual delivery volume , such that the mall can update firm's local inventory stock and the current goods price.

```
START_UPDATE_MALL_STOCK_MESSAGE_LOOP

  // Message filter used: if(a.id==m.mall_id)

    for(int j=0; j < CURRENT_STOCK.size; j++)
    {

      if(update_mall_stock_message->firm_id==
      CURRENT_STOCK.array[j].firm_id)
      {
        CURRENT_STOCK.array[j].stock=CURRENT_STOCK.array[j].
        stock + update_mall_stock_message->quantity;

        CURRENT_STOCK.array[j].firm_id=
        update_mall_stock_message->firm_id;

        CURRENT_STOCK.array[j].quality=
        update_mall_stock_message->quality;

        CURRENT_STOCK.array[j].price=
        update_mall_stock_message->price;
        //Counting the daily imports
        if(REGION_ID != CURRENT_STOCK.array[j].region_id)
        {

          daily_imports_nominal += update_mall_stock_message->quantity*
          update_mall_stock_message->price;
          daily_imports_real += update_mall_stock_message->quantity;

        }
      }
    }

  FINISH_UPDATE_MALL_STOCK_MESSAGE_LOOP
```

Mall_send_quality_price_info_1

The mall sends a daily message to the household that informs the consumers about the available range of goods. This message contains the information whether a good is available, and its corresponding price.

```
int i;
  int available;
  for(i=0;i<CURRENT_STOCK.size;i++)
  {
    if(CURRENT_STOCK.array[i].stock > 0)
    {
      available= 1;
    }else
    {
      available= 0;
    }
    add_quality_price_info_1_message(ID,REGION_ID,
    CURRENT_STOCK.array[i].firm_id ,
    CURRENT_STOCK.array[i].quality ,
    CURRENT_STOCK.array[i].price , available);
  }
```

Mall_update_mall_stocks_sales_sales_1

The first action of this function is the the mall collects all consumption requests related to the first shopping loop.

```
START_CONSUMPTION_REQUEST_1_MESSAGE_LOOP
        add_consumption_request(&consumption_request_list ,
        consumption_request_1_message->worker_id ,
        consumption_request_1_message->firm_id ,
        consumption_request_1_message->quantity );
FINISH_CONSUMPTION_REQUEST_1_MESSAGE_LOOP
```

By summing all individual requests the mall determines for each of the provided goods the total demand in the first shopping loop. If there is a excess demand (summed demand>supply) for a good, the mall has to ration the accepted quantities. Therefore, it computes a quota indicating how much of the total demand is covered by the actual supply. Each demander for a rationed good receives the percentage of her ordered quantity that corresponds to the rationing quota.

The mall sends messages to the costumers containing the actual delivery volumes and computes the firm specific sales after shopping round 1, and finally it subtracts the sold quantities from the local inventories.

At the end the mall sends a new message to households with the updated mall stocks.

```
/*Aggregation of demand per firm*/
  for(int i = 0; i < CURRENT_STOCK.size;i++)
  {
    aggregated_demand=0.0;
    for(int j = 0; j < consumption_request_list.size; j++)
    {
      if(CURRENT_STOCK.array[i].firm_id ==
      consumption_request_list.array[j].firm_id)
      {
        aggregated_demand+= consumption_request_list.array[j].quantity;
      }
    }
    /*If aggregated demand > current stock . Rationing*/
    if(aggregated_demand> CURRENT_STOCK.array[i].stock)
    {

      rationing_rate= CURRENT_STOCK.array[i].stock/ aggregated_demand;

      for(int k=0; k<consumption_request_list.size;k++)
      {
        if(CURRENT_STOCK.array[i].firm_id ==
        consumption_request_list.array[k].firm_id)
        {
          /*Send accepted consumption volume*/
          add_accepted_consumption_1_message(ID,
          consumption_request_list.array[k].worker_id,
          consumption_request_list.array[k].quantity*
          rationing_rate, 1);
        }
      }
          CURRENT_STOCK.array[i].sales = CURRENT_STOCK.array[i]
          .stock*CURRENT_STOCK.array[i].price;

          /*mall stock completely sold out*/
          CURRENT_STOCK.array[i].stock=0.0;
    }
    else /*Otherwise no rationing*/
    {
      for(int k=0; k<consumption_request_list.size;k++)
      {
        if(CURRENT_STOCK.array[i].firm_id ==
        consumption_request_list.array[k].firm_id)
        { /*Send accepted consumption volume*/
          add_accepted_consumption_1_message(ID,
          consumption_request_list.array[k].worker_id,
          consumption_request_list.array[k].quantity, 0);
        }
      }
        /*Calc and store revenues per firm*/
          CURRENT_STOCK.array[i].sales =
          aggregated_demand *CURRENT_STOCK.array[i].price ;
          /*remaining mall stock*/
          CURRENT_STOCK.array[i].stock-=aggregated_demand;
    }
  }

  free_consumption_request_array(&consumption_request_list);

  /*Send second price info*/
  int i;
  int available;
  for(i=0;i<CURRENT_STOCK.size;i++)
  {

    if(CURRENT_STOCK.array[i].stock > 0)
    {
      available= 1;
    }else
    {
      available= 0;
    }

    add_quality_price_info_2_message(ID,REGION_ID,
    CURRENT_STOCK.array[i].firm_id,
    CURRENT_STOCK.array[i].quality,
    CURRENT_STOCK.array[i].price,available);

  }
```

Mall_update_mall_stocks_sales_sales_2

Just like in Mall_update_mall_stocks_sales_sales_1, the mall receives a message with information about the order quantities of households, sums them up and determines, if necessary, a rationing quota. It then sends the accepted consumption goods to the households and computes the sales.

```
  /*Read the request*/
START_CONSUMPTION_REQUEST_2_MESSAGE_LOOP
// Message filter used: if(a.id==m.mall_id)

    add_consumption_request(&consumption_request_list,
      consumption_request_2_message->worker_id,
    consumption_request_2_message->firm_id,
    consumption_request_2_message->quantity );

FINISH_CONSUMPTION_REQUEST_2_MESSAGE_LOOP

  /*Aggregation of demand*/
for(int i = 0; i < CURRENT_STOCK.size;i++)
{
  aggregated_demand=0;
  for(int j = 0; j < consumption_request_list.size; j++)
  {
    if(CURRENT_STOCK.array[i].firm_id ==
    consumption_request_list.array[j].firm_id)
    {
      aggregated_demand+= consumption_request_list.array[j].quantity;
    }
  }
  /*If agg demand > mall stocks . Rationing*/
  if(aggregated_demand > CURRENT_STOCK.array[i].stock)
  {
    rationing_rate= CURRENT_STOCK.array[i].stock/ aggregated_demand;
```

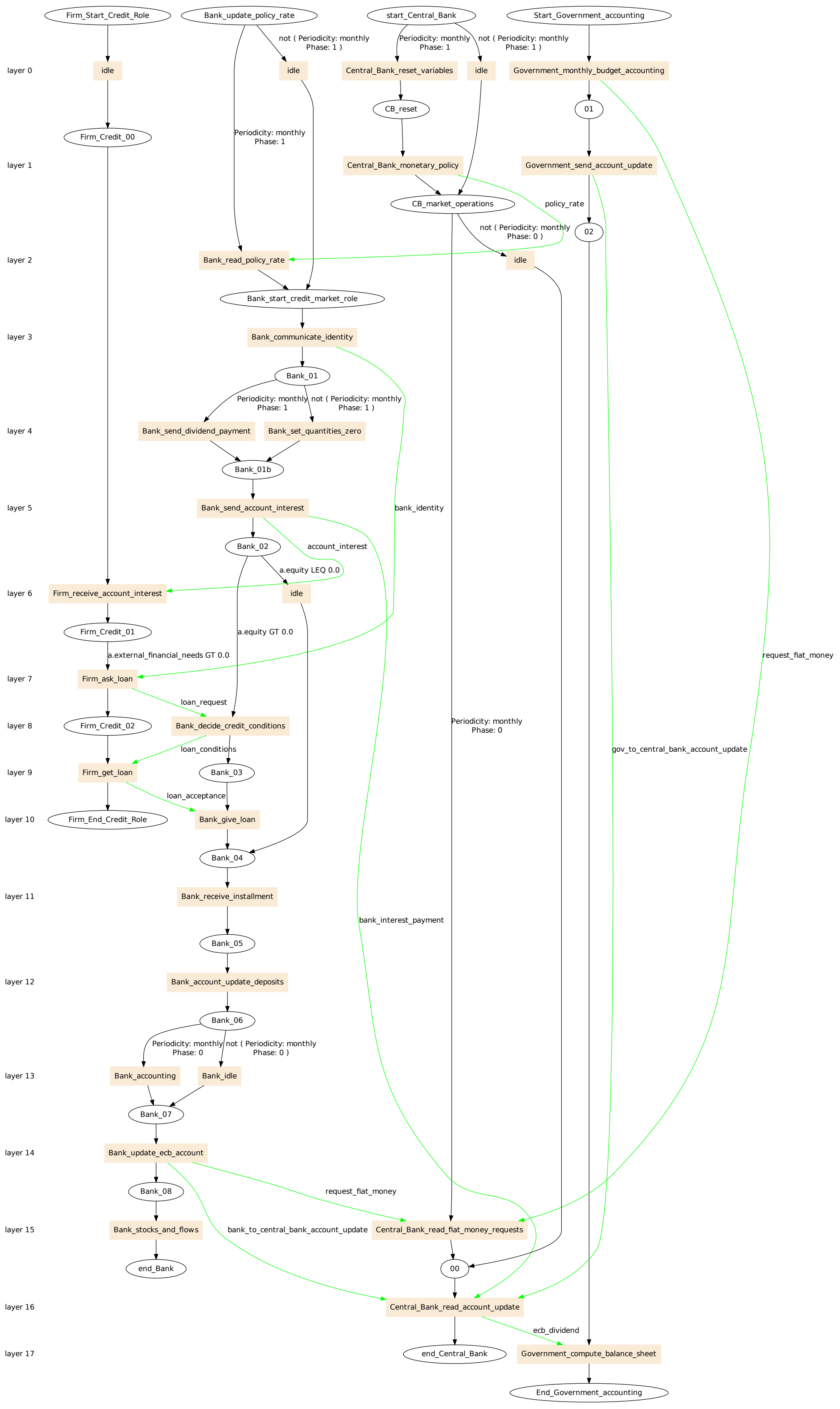
```
      for(int k=0; k<consumption_request_list.size;k++)
      {
         if(CURRENT_STOCK.array[i].firm_id ==
         consumption_request_list.array[k].firm_id)
         {
            add_accepted_consumption_2_message(ID,
            consumption_request_list.array[k].worker_id,
            consumption_request_list.array[k].quantity*
            rationing_rate, 1);
         }
      }
      /*Revenues and final mall stock*/

           CURRENT_STOCK.array[i].sales += CURRENT_STOCK.array[i]
           .stock*CURRENT_STOCK.array[i].price;

           CURRENT_STOCK.array[i].stock=0;
   }
   else /*Otherwise no rationg*/
   {
   for(int k=0; k<consumption_request_list.size;k++)
      {
         if(CURRENT_STOCK.array[i].firm_id ==
         consumption_request_list.array[k].firm_id)
         {
            add_accepted_consumption_2_message(ID,
            consumption_request_list.array[k].worker_id,
            consumption_request_list.array[k].quantity, 0);
         }
      }
      /*revenues and final stocks*/
           CURRENT_STOCK.array[i].sales += aggregated_demand *
           CURRENT_STOCK.array[i].price;

           CURRENT_STOCK.array[i].stock-=aggregated_demand;
   }
}
free_consumption_request_array(&consumption_request_list);
```

Mall_pay_firm

Here, the mall sends a daily message to the consumption goods producers containing the revenue and the current mall stock.

```
TOTAL_SUPPLY=0.0;
   int stock_empty;
   for(int i=0; i<CURRENT_STOCK.size; i++)
   {
      TOTAL_SUPPLY+=CURRENT_STOCK.array[i].stock;
      if(CURRENT_STOCK.array[i].stock ==0)
      {
         stock_empty =1;
      }else
      {
         stock_empty =0;
      }
      add_sales_message(ID, CURRENT_STOCK.array[i].firm_id,
    CURRENT_STOCK.array[i].sales,stock_empty,CURRENT_STOCK.array[i].stock);
  }
```

3.5 Stategraph


Firm_Start_Credit_Role
Bank_update_policy_rate
start_Central_Bank
Start_Government_accounting
not (Periodicity: monthly Phase: 1)
Periodicity: monthly Phase: 1
not (Periodicity: monthly Phase: 1)
layer 0
idle
idle
Central_Bank_reset_variables
idle
Government_monthly_budget_accounting
CB_reset
01
Firm_Credit_00
Periodicity: monthly Phase: 1
layer 1
Central_Bank_monetary_policy
Government_send_account_update
CB_market_operations
policy_rate
not (Periodicity: monthly Phase: 0)
02
layer 2
Bank_read_policy_rate
idle
Bank_start_credit_market_role
layer 3
Bank_communicate_identity
Bank_01
Periodicity: monthly Phase: 1
not (Periodicity: monthly Phase: 1)
layer 4
Bank_send_dividend_payment
Bank_set_quantities_zero
Bank_01b
layer 5
Bank_send_account_interest
bank_identity
Bank_02
account_interest
a.equity LEQ 0.0
layer 6
Firm_receive_account_interest
idle
Firm_Credit_01
a.equity GT 0.0
a.external_financial_needs GT 0.0
request_fiat_money
layer 7
Firm_ask_loan
loan_request
Periodicity: monthly Phase: 0
layer 8
Firm_Credit_02
Bank_decide_credit_conditions
loan_conditions
gov_to_central_bank_account_update
layer 9
Firm_get_loan
Bank_03
loan_acceptance
layer 10
Firm_End_Credit_Role
Bank_give_loan
Bank_04
layer 11
Bank_receive_installment
bank_interest_payment
Bank_05
layer 12
Bank_account_update_deposits
Bank_06
Periodicity: monthly Phase: 0
not (Periodicity: monthly Phase: 0)
layer 13
Bank_accounting
Bank_idle
Bank_07
layer 14
Bank_update_ecb_account
request_fiat_money
Bank_08
layer 15
Bank_stocks_and_flows
bank_to_central_bank_account_update
Central_Bank_read_fiat_money_requests
end_Bank
00
layer 16
Central_Bank_read_account_update
ecb_dividend
layer 17
end_Central_Bank
Government_compute_balance_sheet
End_Government_accounting

Chapter 4

Labour Market Documentation

4.1 Firm

4.1.1 Activities

Firms are active on the labour market after the production planning but before the production takes place. Firms perform the following activities on a monthly base:

- Lay off workers if number of employees is bigger than the actual labour demand.
- Employ workers if number of employees is smaller than the actual labour demand. Determine wage offers, post vacancies, and select among applicants.

4.1.2 Functions

Firm calculate specific skills and wage offer

1. The firms calculate the average specific skills of each general skill group $b^{gen} \in \{1, 2, ...5\}$ in order to determine wage offers for each group. Therefore in a first step the specific skills $b_{j,k,t}$ with $k \in \{1; 2; ...5\}$ of each general skill group are summed up.

```
//For each employee.
for(int n = 0; n < EMPLOYEES.size; n++)
{
    /*For each general skill level (1-5)*/
    switch(EMPLOYEES.array[n].general_skill)
    {
       case 1:/*If employee has general skill level 1
        add specific skill to sum1.*/
        sum_1 = sum_1 + EMPLOYEES.array[n].specific_skill;
        break;
       .
       .
       .
       case 5:/*If employee has general skill level 5
        add specific skill to sum5.*/
        sum_5 = sum_5 + EMPLOYEES.array[n].specific_skill;
        break;
    }
}
```

2. Afterwards the average specific skills of each general skill group are calculated. If there is no employee in one of the general skill groups the average specific skills of this group is replaced by the average specific skills in the population. This information is sent by the Eurostat agent. The general skill depending wage offers are determined by the product of the base wage offer $w^O_{i,t}$ which is the price of one unit of specific skills times the effective productivity $min\left[A_{i,t}, \bar{B}_{i,k,t}\right]$ of the skill group k. The effective productivity is the minimum of the technolgy $A_{i,t}$ and the average specific skills of the general skill group $\bar{B}_{i,k,t}$.

$$w^O_{k,t} = w^O_{i,t} \times min\left[A_{i,t}, \bar{B}_{i,k,t}\right]$$

```
/*For each general skill group:
 calculate the average specific skills*/
for(m = 1; m < 6;m++)
{
    switch(m)
    {
       /*If general skill level 1*/
       case 1:
        /*If there are employees with general skill
         level 1 replace the average specific skills of
         the population by the skills inside the firm.*/
        if(NO_EMPLOYEES_SKILL_1 > 0)
        {
            /*Calculate average specific skill group 1.*/
            AVERAGE_S_SKILL_OF_1 = sum_1/NO_EMPLOYEES_SKILL_1;
        }

        WAGE_OFFER_FOR_SKILL_1 =
        WAGE_OFFER* min(TECHNOLOGY, AVERAGE_S_SKILL_OF_1);
        break;
       .
       .
       .
      case 5:
        /*If there are employees with general skill
         level 5 replace the average specific skills of
         the population by the skills inside the firm.*/
        if(NO_EMPLOYEES_SKILL_5 > 0)
        {
            /*Calculate average specific skill group 5.*/
            AVERAGE_S_SKILL_OF_5 =
            sum_5/NO_EMPLOYEES_SKILL_5;
        }

        WAGE_OFFER_FOR_SKILL_5 =
        WAGE_OFFER*min(TECHNOLOGY, AVERAGE_S_SKILL_OF_5);
        break;
    }
}
```

Firm send redundancies

If the firms have more employees than needed they have to lay off employees. The number of redundancies is the difference between the number of employees and the number of needed employees. The number of needed employees is determined in the function **Firm calc input**

demands in the Consumption Goods Market. The employees are fired randomly (alternative: lowest specific skills first) and are notified via a firing message.

```
no_redundancies = NO_EMPLOYEES - EMPLOYEES_NEEDED;

for(int i = 0; i < (no_redundancies); i++)
{
    /*Firing: randomly*/
    j = random_int(0,(EMPLOYEES.size -1));
    add_firing_message(ID, EMPLOYEES.array[j].id);

    /*Firing: lowest specific skill*/
    /*j = EMPLOYEES.size -1;

    //Send quitting to employee.
    add_firing_message(ID, EMPLOYEES.array[j].id);*/

    //Adjust the workforce.
    switch(EMPLOYEES.array[j].general_skill)
    {
        /*If employee has skill level 1 reduce the
        number of employees with skill level 1 by 1.*/
        case 1:
            NO_EMPLOYEES_SKILL_1--;
            break;
        .
        .
        .
        /*If employee has skill level 5 reduce the number
        of employees with skill level 5 by 1.*/
        case 5:
            NO_EMPLOYEES_SKILL_5--;
            break;
    }

    remove_employee(&EMPLOYEES, j);
    NO_EMPLOYEES--;
}
```

Firm send random redundancies

Additionally to the workers dismissed because of a necessary adjustment of the workforce, the firms fire a random number of workers. This reflects the normal fluctuations of workers in a firm. The number of fired workers is a random number that is equally distributed between the boundaries *LOWER_BOUND_FIRING* and *UPPER_BOUND_FIRING*. The boundaries are constants during the simulation. The workers are notified via a firing message.

```
//Draw a random number.
int random_num = random_int(LOWER_BOUND_FIRING,UPPER_BOUND_FIRING);

/*The random number is the percentage of employees
who are laid off.*/
int no_redundancies = (random_num*NO_EMPLOYEES)/100;

for(int i = 0; i < (no_redundancies); i++)
{
    /*Firing: randomly*/
    j = random_int(0,(EMPLOYEES.size -1));
```

```
        add_firing_message(ID, EMPLOYEES.array[j].id);

        //Send quitting to employee.
        add_firing_message(ID, EMPLOYEES.array[j].id);*/

        //Adjust the workforce.
        switch(EMPLOYEES.array[j].general_skill)
        {
            /*If employee has skill level 1 reduce the
            number of employees with skill level 1 by 1.*/
            case 1:
                NO_EMPLOYEES_SKILL_1--;
                break;
            .
            .
            .
            /*If employee has skill level 5 reduce the number
            of employees with skill level 5 by 1.*/
            case 5:
                NO_EMPLOYEES_SKILL_5--;
                break;
        }

        remove_employee(&EMPLOYEES, j);
        NO_EMPLOYEES--;
    }
```

Firm send vacancies

In the case that firms want to hire new workers they calculate the number of vacancies v which is the difference between the number of needed employees and the number of employees. The number of needed employees is determined in the function **Firm calc input demands** in the Consumption Goods Market. Firms send one vacancy message which is read by unemployed households containing wage offers $w^0_{i,k,t}$ for each general skill group.

```
    /*Number of vacancies/additional employees wanted.*/
    VACANCIES = EMPLOYEES_NEEDED - NO_EMPLOYEES;

    /*Send vacancy message with wage offers.*/
    add_vacancies_message(ID, REGION_ID,
    WAGE_OFFER_FOR_SKILL_1, WAGE_OFFER_FOR_SKILL_2,
    WAGE_OFFER_FOR_SKILL_3, WAGE_OFFER_FOR_SKILL_4,
    WAGE_OFFER_FOR_SKILL_5);
```

Firm read job applications send job offer or rejection

In the first step of this function firms receive the applications from unemployed workers containing their workerID, specific skills, and the general skill level.

```
    START_JOB_APPLICATION_MESSAGE_LOOP

        /* Read job application messages for this Firm. */
        if(job_application_message->firm_id == ID)
        {
            /*Add application to a list (array).*/
            add_job_application(&job_application_list,
            job_application_message->worker_id,
            job_application_message->region_id,
            job_application_message->general_skill,
            job_application_message->specific_skill);
        }

    FINISH_JOB_APPLICATION_MESSAGE_LOOP

    //Number of applicants.
    no_applications = job_application_list.size;
```

After counting the number of applications there are two cases:

1. In the first case the number of applications is equal or smaller than the number of vacancies. In this case the firms send job offers to each applicant on the application list.

```
if(no_applications <= VACANCIES)
{
  for(i = 0; i < (job_application_list.size); i++)
  {
    /*For each genral skill level (1-5) */
    switch(job_application_list.array[i].general_skill)
    {
      /*If general skill level of the applicant is 1 send
       job offer with wage offer for general skill level 1*/
      case 1:
        add_job_offer_message(ID,
        job_application_list.array[i].worker_id,
        REGION_ID,WAGE_OFFER_FOR_SKILL_1);
        break;
        .
        .
        .
      /*If general skill level of the applicant is 5 send
       job offer with wage offer for general skill level 5*/
      case 5:
        add_job_offer_message(ID,
        job_application_list.array[i].worker_id,
        REGION_ID,WAGE_OFFER_FOR_SKILL_5);
        break;
    }
  }
}
```

2. In the second case the number of applicants is bigger than the number of vacancies. In this case the firms send as many job offers to applicants as it has vacancies to fill. The selection is based on a standard logit model that takes into account the general (or optionally the specific) skill level. Applicants with high general skills have a higher probability to be chosen.

```
if(no_applications > VACANCIES)
{
  //For each vacancy.
  for(i = 0; i< VACANCIES; i++)
  {
    logit_array  logit_applications_list;
    init_logit_array(&logit_applications_list);
```

```
/*Computing the denominator of the logit.*/
double denominator_logit = 0;
double logit = 0.0;
double sum_of_logits;
double random_number;

for(j = 0; j< job_application_list.size;j++)
{
  denominator_logit+=exp(LOGIT_PARAMETER_GENERAL_SKILLS*
  job_application_list.array[j].general_skill +
  LOGIT_PARAMETER_SPECIFIC_SKILLS*
  job_application_list.array[j].specific_skill);
}

/*Compute the logits and store them at the temporary array
 logit_applications_list.*/

if(denominator_logit >0)
{
  for(j = 0; j< job_application_list.size;j++)
  {
    logit = exp(LOGIT_PARAMETER_GENERAL_SKILLS*
    job_application_list.array[j].general_skill +
    LOGIT_PARAMETER_SPECIFIC_SKILLS*
    job_application_list.array[j].specific_skill)
    /denominator_logit;

    add_logit(&logit_applications_list, 100*logit,
    job_application_list.array[j].worker_id,
    job_application_list.array[j].general_skill);
  }
}

/*Draw a random number.*/
random_number =  (double)random_int(0,100);

sum_of_logits = 0;
for(j = 0; j< logit_applications_list.size;j++)
{
  sum_of_logits +=
  logit_applications_list.array[j].logit_value;

  if(random_number < sum_of_logits)
  {
    /*If condition is true, the firm chooses this
     worker and send a job offer with the
     corresponding wage offer.*/

    switch(logit_applications_list.array[j].general_skill)
    {
      /*If general skill level of the applicant is 1
       send job offer with wage offer for
       general skill level 1.*/
      case 1:
        add_job_offer_message(ID,
        logit_applications_list.array[j]
        .worker_id,REGION_ID, WAGE_OFFER_FOR_SKILL_1);
        break;
      .
```

```
                .
                .
                /*If general skill level of the applicant is 5
                 send job offer with wage offer for
                 general skill level 5.*/
                case 5:
                   add_job_offer_message(ID,
                   logit_applications_list.array[j]
                   .worker_id ,REGION_ID, WAGE_OFFER_FOR_SKILL_5);
                   break;
             }

             /*The chosen worker has to be removed from
              the application list.*/
             for(k = 0; k< job_application_list.size;k++)
             {
                if(job_application_list.array[k].worker_id ==
                logit_applications_list.array[j].worker_id)
                {
                   remove_job_application(&job_application_list ,k);
                   break;
                }
             }
             break;
          }
       }
       /* Free the job application dynamic array */
       free_logit_array(&logit_applications_list);
    }
 }
```

Firm read job responses

The firms read the accepted job offers and adds the worker and his characteristics (ID, wage, general, and specific skill) to its workforce.

```
START_JOB_ACCEPTANCE_MESSAGE_LOOP

   /* Read job acceptance messages for this Firm */
   if(job_acceptance_message->firm_id == ID)
   {
      /*Reduce number of vacancies and
      increase number of employees*/
      VACANCIES--;
      NO_EMPLOYEES++;

         switch(job_acceptance_message->general_skill)
         {
            /*If new employee has general skill level 1*/
            case 1:
               add_employee(&EMPLOYEES,
               job_acceptance_message->worker_id ,
               job_acceptance_message->region_id ,
               WAGE_OFFER_FOR_SKILL_1,
               job_acceptance_message->general_skill ,
               job_acceptance_message->specific_skill);

               NO_EMPLOYEES_SKILL_1++;
               break;
               .
               .
               .
               /*If new employee has general skill level 5*/
            case 5:
               add_employee(&EMPLOYEES,
               job_acceptance_message->worker_id,
               job_acceptance_message->region_id,
               WAGE_OFFER_FOR_SKILL_5,
               job_acceptance_message->general_skill,
               job_acceptance_message->specific_skill);

               NO_EMPLOYEES_SKILL_5++;
               break;
         }
   }
```

Firm update wage offer

If the firms have still vacancies to fill and the number of vacancies is higher than a threshold $\bar{v}$ they increase the wage offer for each skill group by φ hence $w^O_{i,k,t+1} = w^O_{i,k,t}(1+\varphi_i)$. $\bar{v}$ and φ are constant during the simulation.

```
    if(VACANCIES > MIN_VACANCY)
    {
        //Increase base wage offer for the next period.
        WAGE_OFFER = WAGE_OFFER*(1+WAGE_UPDATE);

        //Increase actual wage offers for the second loop.
        WAGE_OFFER_FOR_SKILL_1 =
        WAGE_OFFER_FOR_SKILL_1*(1+WAGE_UPDATE);
        .
        .
        .
        WAGE_OFFER_FOR_SKILL_5 =
        WAGE_OFFER_FOR_SKILL_5*(1+WAGE_UPDATE);
    }
```

Remark: The last four functions (Firm send vacancies to Firm update wage offer) constitute a loop which is ran two times each month on the activation days of the firms.

Firm compute mean wage specific skills

After the firms finished their basic labour market activities they calculate the mean wage and specific skills mainly for activities (production) in the Consumption Goods Market.

```
    /*Sum up the wage and the specific skills of each worker.*/
    for(i=0;i<EMPLOYEES.size;i++)
    {
      ave_specific_skills += EMPLOYEES.array[i].specific_skill;
      ave_general_skills += EMPLOYEES.array[i].general_skill;
    }

    /*If the firm has no employees*/
    double no_employees = (double) NO_EMPLOYEES;
    if(no_employees==0)
    {
      MEAN_WAGE = WAGE_OFFER;
      MEAN_SPECIFIC_SKILLS = AVERAGE_S_SKILL_OF_1;
    }

    /*If the firm has  employees calculate the mean wage and mean specific  skills.*/
    else
    {
      MEAN_WAGE = ave_wage /( no_employees);
      MEAN_SPECIFIC_SKILLS =ave_specific_skills/no_employees;
    }
```

4.2 Household

4.2.1 Activities

Households perform the following activities in the labour market:

- If employed: receive a wage from the current employer, read firing message.
- If unemployed: receive unemployment benefits from the government, send job applications to firms that have opened vacancies.

4.2.2 Functions

Household read firing message

If households/workers are employed they have to check if they are fired by reading firing messages on the activation day of the employer. Fired workers set EMPLOYEE_FIRM_ID to -1 to indicate that they are unemployed.

```
    /* Check for firing message */
    START_FIRING_MESSAGE_LOOP

        /*If employee is fired*/
        if(firing_message->worker_id == ID)
        {
            EMPLOYEE_FIRM_ID = -1;
            LAST_LABOUR_INCOME = WAGE;
            WAGE = 0;
        }
    FINISH_FIRING_MESSAGE_LOOP
```

Household unemployed read job vacancies and send applications

The probability of an unemployed worker to search for a new job on the actual day/iteration is

```
p = NUMBER_APPLICATIONS/(20*APPLICATIONS_PER_DAY)
```

Number_applications is the the number of applications an unemployed worker is allowed to send in one month while Application_per_day is the maximum number on one day. They are constant during the simulation. In the model 20 days are one month.

1. The unemployed households are searching for a new job. Therefore they read vacancy messages from firms and compare the wage offer with its reservation wage $w^R_{k,t}$. If the vacancy is not in the domestic region the household has to take commuting costs into consideration. The information about the vacancies are stored in a vacancy-list if the wage offer (net of commuting costs) is higher than reservation wage.

```
START_VACANCIES_MESSAGE_LOOP

    /*Unemployed check the wage offer for their skill group.*/
    if(GENERAL_SKILL == 1)
    {
        wage_offer =
        vacancies_message->firm_wage_offer_for_skill_1;
    }
    .
    .
    .
    if(GENERAL_SKILL == 5)
    {
        wage_offer =
        vacancies_message->firm_wage_offer_for_skill_5;
    }

    /*Wage offer has to be equal or higher
    than the reservation wage.*/
    if(wage_offer >= WAGE_RESERVATION)
    {
        /*Same region: Firm and Household.*/
        if(REGION_ID == vacancies_message->region_id)
        {
            add_vacancy(&vacancy_list,
            vacancies_message->firm_id,
            vacancies_message->region_id,
            wage_offer);
        }
        else /*Different regions: Firm and Household.*/
        {
            /*For every neighboring region*/
            for(i = 0;i < NEIGHBORING_REGION_IDS.size; i++)
            {
                /*If vacancy is in a neighboring region.*/
                if(vacancies_message->region_id
                ==NEIGHBORING_REGION_IDS.array[i])
                {
                  //Commuting costs.
                  if((wage_offer-REGION_COST)
                    >= WAGE_RESERVATION)
                  {
                    add_vacancy(&vacancy_list,
                    vacancies_message->firm_id,
                    vacancies_message->region_id,
                    (wage_offer - REGION_COST));
                  }
                  break;
                }
            }
        }
    }
FINISH_VACANCIES_MESSAGE_LOOP
```

2. The household sends an exogenously determined number of applications (applications_per_day) to randomly chosen vacancies from the vacancy list.

```
if(vacancy_list.size > APPLICATIONS_PER_DAY)
{
    /*Remove vacancies from the list randomly until
    the list contains as many vacancies as a
    household can send applications*/
    while(vacancy_list.size > NUMBER_APPLICATIONS)
    {
        j = random_int(0, (vacancy_list.size -1));
        remove_vacancy(&vacancy_list, j);
    }
}

/*If the vacancy list is bigger than zero then send
a job application to every vacancy on the list.*/
for(i = 0; i < (vacancy_list.size); i++)
{
    add_job_application_message(ID,
    vacancy_list.array[i].firm_id,
    REGION_ID, GENERAL_SKILL, SPECIFIC_SKILL);
}
```

Household read job offers send response

1. The households receive the job offers containing the ID and the regionID of the employer and the wage offer. If they receive more than one job offer they rank the incoming offers regarding the wage offer (net of commuting costs) accept the job offer with highest wage.

```
START_JOB_OFFER_MESSAGE_LOOP

    /* Read job offer.*/
    if(job_offer_message->worker_id == ID)
    {
        /*Job offers of firms in the same region*/
        if(REGION_ID == job_offer_message->region_id)
        {
            add_job_offer(&job_offer_list,
            job_offer_message->firm_id,
            job_offer_message->region_id,
            job_offer_message->wage_offer);
        }
        else/*Job offers of firms in different regions*/
        {
            add_job_offer(&job_offer_list,
            job_offer_message->firm_id,
            job_offer_message->region_id,
            (job_offer_message->wage_offer -
            REGION_COST);
        }
    }

FINISH_JOB_OFFER_MESSAGE_LOOP
```

2. If they receive more than one job offer they rank the incoming offers regarding the wage offer (net of commuting costs) accept the job offer with highest wage. They send a job acceptance message containing their ID, general, and specific skills.

```
    if(job_offer_list.size > 0)
    {
        add_job_acceptance_message(ID,
        job_offer_list.array[0].firm_id,
        REGION_ID, GENERAL_SKILL, SPECIFIC_SKILL);

        EMPLOYEE_FIRM_ID = job_offer_list.array[0].firm_id;
        EMPLOYER_REGION_ID = job_offer_list.array[0].region_id;
        DAY_OF_MONTH_RECEIVE_INCOME = DAY%MONTH;

        /*Update some memory variables because of the new job*/
        if(REGION_ID == job_offer_list.array[0].region_id)
        {
          WAGE = job_offer_list.array[0].wage_offer;
          WAGE_RESERVATION = WAGE;
        }
        else
        {
          /*Commuting costs. Add commuting costs to wage offer.
          Commuting costs were substracted from the wage offer
          when they were ranked.*/
          WAGE =
          (job_offer_list.array[0].wage_offer + REGION_COST);

          /*Reservation wage is the wage net of commuting costs*/
          WAGE_RESERVATION = WAGE - REGION_COST;
        }
    }
```

Household update wage reservation

If the household is still unemployed it adjusts the reservation wage downwards by a certain fraction $w^R_{k,t+1} = w^R_{k,t}(1+\psi)$. ψ is constant during the simulation. The reservation wage cannot be lower than the unemployment benefits.

```
    WAGE_RESERVATION = WAGE_RESERVATION -
    WAGE_RESERVATION*WAGE_RESERVATION_UPDATE;

    /* Don't let wage reservation be
    below the current unemployment benefit pct.*/
    if(WAGE_RESERVATION < LAST_LABOUR_INCOME*UNEMPLOYMENT_BENEFIT_PCT)
    {
      WAGE_RESERVATION = LAST_LABOUR_INCOME*UNEMPLOYMENT_BENEFIT_PCT;
    }
```

REMARK: The last three functions (Household unemployed read vacancies and send applications to Household update wage reservation) constitute a loop which is ran two times a day when the unemployed household is searching for a new job.

Household send unemployment notification

If households are unemployed they they send an unemployment notification message to the government on the day when they received their last wage in each month. If their last wage is higher than the mean wage in the economy (or region) they receive a fraction unemployment_benefit_pct of their last wage. The parameter unemployment_benefit_pct is constant during the simulation. If their last wage is lower than the mean wage they recive 50% of the mean wage.

```
/*Compute the individual unemployment benefit payment
as a fraction of the last labour income
if unemployment benefit is larger than the mean wage:*/

if(LAST_LABOUR_INCOME*UNEMPLOYMENT_BENEFIT_PCT >
REGION_WIDE_MEAN_WAGE*0.5 )
{
    UNEMPLOYMENT_PAYMENT =
    LAST_LABOUR_INCOME*UNEMPLOYMENT_BENEFIT_PCT;
}
else
{
    //if unemployment benefit is below the
    mean wage: pay 0.5 * MEAN_WAGE
    UNEMPLOYMENT_PAYMENT =
    REGION_WIDE_MEAN_WAGE*0.5;
}

/*Add unemployment_benefit message */
add_unemployment_notification_message(GOV_ID,
UNEMPLOYMENT_PAYMENT );
HOUSEHOLD_INFLOWS_CALENDAR.unemployment_benefit +=
UNEMPLOYMENT_PAYMENT;

/*Add unemployment_benefit to account */
PAYMENT_ACCOUNT += UNEMPLOYMENT_PAYMENT;


TOTAL_INCOME= UNEMPLOYMENT_PAYMENT +
CUM_TOTAL_DIVIDENDS + MONTHLY_BOND_INTEREST_INCOME;


//Set the benefit reception day
DAY_OF_MONTH_RECEIVE_BENEFIT = DAY_OF_MONTH_RECEIVE_INCOME;
```

Household receive wage

If households are employed they receive the wage on the activation day of the employer in each month. The wage messages contain also the productivity of the employer.

```
/*Household reads the wage messages if employed.*/
START_WAGE_PAYMENT_MESSAGE_LOOP

    /*Set wage and reservation wage.*/
    WAGE = wage_payment_message->payment;
    WAGE_RESERVATION = WAGE;

    HOUSEHOLD_INFLOWS_CALENDAR.wage +=
    wage_payment_message->payment;

    /*Calculate the total income.*/
    TOTAL_INCOME= wage_payment_message->payment +
    CUM_TOTAL_DIVIDENDS +
    MONTHLY_BOND_INTEREST_INCOME;

    /*Add wage to payment account.*/
    PAYMENT_ACCOUNT += wage_payment_message->payment;

    /*Store the productivity of the employer in order
    to update the specific skills later.*/

    CURRENT_PRODUCTIVITY_EMPLOYER =
    wage_payment_message-> productivity;

FINISH_WAGE_PAYMENT_MESSAGE_LOOP
```

Household update specific skills

If households are employed they update their specific skills on the activation day of the employer in each month. They send the updated specific skills to their employer. The specific skills evolve according to

$$b_{w,t+1} = b_{w,t} + \chi(b_w^{gen}) \cdot (A_{i,t} - b_{w,t}) \, ,$$

where $b_{w,t}$ are the specific skills of the worker, b_w^{gen} is the general skill level, $\chi()$ is a function which increases with the general skill level, and $A_{i,t}$ is the average quality/productivity of the capital stock. The function χ is increasing in the general skill level of the worker. The higher the genarel skill level the faster is the learning process. Negative learning, i.e. a decrease of the specific skill level, is not possible.

```
    /*If the specific skill of household is lower than
    the productivity of the employer
    - no negative effect: no forgetting.*/
    if(SPECIFIC_SKILL < CURRENT_PRODUCTIVITY_EMPLOYER)
    {
      /*Update the specific skill: depends on the actual
      specific skill, the gap between the actual specific
      skills and the actual productivity of the employer,
      and the general skill which determines the speed of
      closing the this gap.*/

      SPECIFIC_SKILL = SPECIFIC_SKILL +
      (CURRENT_PRODUCTIVITY_EMPLOYER - SPECIFIC_SKILL)*
      ((1-pow(0.5,1/(20+0.25*(GENERAL_SKILL-1)*(4-20)))));
    }

    /*Send specific skill to employer.*/
     add_specific_skill_update_message(ID,
     EMPLOYEE_FIRM_ID, SPECIFIC_SKILL);
```

4.3 Stategraph

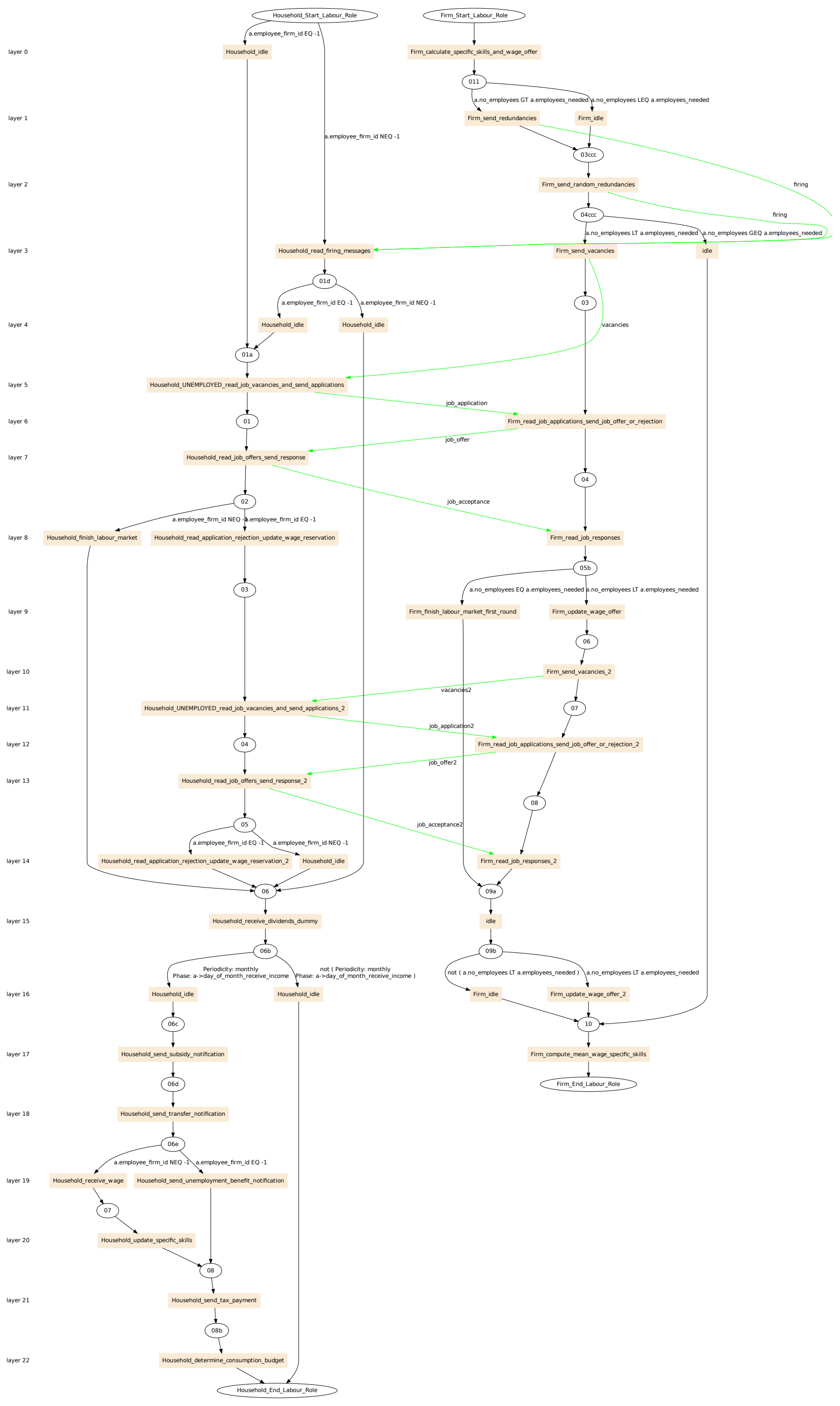

Household_Start_Labour_Role
Firm_Start_Labour_Role
a.employee_firm_id EQ -1
layer 0
Household_idle
Firm_calculate_specific_skills_and_wage_offer
011
a.no_employees GT a.employees_needed
a.no_employees LEQ a.employees_needed
layer 1
Firm_send_redundancies
Firm_idle
a.employee_firm_id NEQ -1
03ccc
layer 2
Firm_send_random_redundancies
firing
04ccc
firing
a.no_employees LT a.employees_needed
a.no_employees GEQ a.employees_needed
layer 3
Household_read_firing_messages
Firm_send_vacancies
idle
01d
a.employee_firm_id EQ -1
a.employee_firm_id NEQ -1
03
layer 4
Household_idle
Household_idle
vacancies
01a
layer 5
Household_UNEMPLOYED_read_job_vacancies_and_send_applications
job_application
layer 6
01
Firm_read_job_applications_send_job_offer_or_rejection
job_offer
layer 7
Household_read_job_offers_send_response
04
02
job_acceptance
a.employee_firm_id NEQ -1
a.employee_firm_id EQ -1
layer 8
Household_finish_labour_market
Household_read_application_rejection_update_wage_reservation
Firm_read_job_responses
05b
03
a.no_employees EQ a.employees_needed
a.no_employees LT a.employees_needed
layer 9
Firm_finish_labour_market_first_round
Firm_update_wage_offer
06
layer 10
Firm_send_vacancies_2
vacancies2
layer 11
Household_UNEMPLOYED_read_job_vacancies_and_send_applications_2
07
job_application2
layer 12
04
Firm_read_job_applications_send_job_offer_or_rejection_2
job_offer2
layer 13
Household_read_job_offers_send_response_2
08
05
job_acceptance2
a.employee_firm_id EQ -1
a.employee_firm_id NEQ -1
layer 14
Household_read_application_rejection_update_wage_reservation_2
Household_idle
Firm_read_job_responses_2
06
09a
layer 15
Household_receive_dividends_dummy
idle
06b
09b
Periodicity: monthly
Phase: a->day_of_month_receive_income
not (Periodicity: monthly
Phase: a->day_of_month_receive_income)
not (a.no_employees LT a.employees_needed)
a.no_employees LT a.employees_needed
layer 16
Household_idle
Household_idle
Firm_idle
Firm_update_wage_offer_2
06c
10
layer 17
Household_send_subsidy_notification
Firm_compute_mean_wage_specific_skills
06d
Firm_End_Labour_Role
layer 18
Household_send_transfer_notification
06e
a.employee_firm_id NEQ -1
a.employee_firm_id EQ -1
layer 19
Household_receive_wage
Household_send_unemployment_benefit_notification
07
layer 20
Household_update_specific_skills
08
layer 21
Household_send_tax_payment
08b
layer 22
Household_determine_consumption_budget
Household_End_Labour_Role

Figure 4.1: Stategraph for the labour market.

Chapter 5

Financial Management Documentation

5.1 Firm

5.1.1 Activities

The Firm's main activities in the Financial Management role are:

1. Computing financial payments and external financing; interfacing between the real and financial economy.
2. Checking for bankruptcy: insolvency and illiquidity (see the bankruptcy documentation).
3. Performing accounting routines.

Note that the external financing is performed only once at the start of the production cycle and involves both the financing of the financial commitments inherited from the previous production cycle and the liquidity needs for the upcoming production cycle this month. We have chosen to finance these two categories simultaneously to prevent the firms from having to enter the credit market twice during one cycle.

In the stategraph the functions of the firm are separated into two main branches:

1. Active branch: the firm is not bankrupt and starts the normal production routine (see the left and centre branch in the stategraph).
2. Inactive branch: the firm is one of the bankruptcy states and executes its bankruptcy routines (see the right-most branch in the stategraph).

When the firm is active, its activites are separated into three parts:

Part 1: Production planning and external financing

1. Compute financial needs for production.
2. Compute financial needs to pay for previous financial commitments: debt installments, interests, and tax payments from the previous cycle.
3. Enter the credit market for external financing of these financial liquidity needs.

Part 2: Post-external financing After the external financing is finished, we check for illiquidity bankruptcy. This occurs if insufficient external financing could be obtained from the credit market. Two possible continuations exist:

1. Enter the bankruptcy illiquidity state immediately and do not start production (see the left-most branch in the stategraph).

2. Proceed with normal operations of the production cycle (see centre branch in the stategraph).

Part 3: Post-production and post-selling.

- The firm performs end-of-production-cycle accounting.

- Compute the income statement, balance sheet and dividends

- Check for insolvency bankrupty.

5.1.2 Functions

Firm compute total liquidity needs

This function is to determine the liquidity needs for production and for financing the inherited financial commitments of the previous cycle.

```
PRODUCTION_LIQUIDITY_NEEDS = PLANNED_PRODUCTION_COSTS;
FINANCIAL_LIQUIDITY_NEEDS = TOTAL_INTEREST_PAYMENT
                            + TOTAL_DEBT_INSTALLMENT_PAYMENT
                            + TAX_PAYMENT;

//Check if additional external financial needs are required
TOTAL_FINANCIAL_NEEDS = PRODUCTION_LIQUIDITY_NEEDS
                        + FINANCIAL_LIQUIDITY_NEEDS
                        + TOTAL_DIVIDEND_PAYMENT;

//CASE 1: No external financing needed
if (PAYMENT_ACCOUNT >= TOTAL_FINANCIAL_NEEDS)
{
    EXTERNAL_FINANCIAL_NEEDS = 0.0;
}
//CASE 2: External financing needed
else
{
    EXTERNAL_FINANCIAL_NEEDS = TOTAL_FINANCIAL_NEEDS - PAYMENT_ACCOUNT;
}
```

Firm execute financial payments

The following financial payments are executed:

1. tax payment

2. debt installments and interest payments on all outstanding loans

3. dividend payment

All payments are subtracted from the payment account and the appropriate messages are send. After these payments have been exectued, the payment account can be used to pay for factor inputs for production.

Tax payment

```
add_tax_payment_message(GOV_ID, TAX_PAYMENT);
PAYMENT_ACCOUNT -= TAX_PAYMENT;
```

Debt installments and interest payments

- reduce the number of debt periods remaining
- reduce the residual value-at-risk with the var per installment
- decrease loan value with the principle repaid (installment amount)
- per loan: send a message to the bank at which the loan was obtained

After this, check if the loan has reached maturity, and if so, remove from the loan portfolio.

```
for (i=LOANS.size-1; i>-1; i--)
{
    LOANS.array[i].nr_periods_before_repayment -= 1;
    LOANS.array[i].residual_var -= LOANS.array[i].var_per_installment;

    //decrease payment_account with the installment payment
    PAYMENT_ACCOUNT -= LOANS.array[i].installment_amount;

    //decrease the value of the loan with the installment_payment:
    LOANS.array[i].loan_value -= LOANS.array[i].installment_amount;

    //pay interest
    temp_interest=(LOANS.array[i].interest_rate/12.0)
                 * LOANS.array[i].loan_value;
    PAYMENT_ACCOUNT -= temp_interest;

    add_installment_message(LOANS.array[i].bank_id,
                            LOANS.array[i].installment_amount,
                            temp_interest,
                            LOANS.array[i].var_per_installment);

    if (LOANS.array[i].nr_periods_before_repayment==0)
        remove_debt_item(&LOANS, i);
}
```

Firm compute financial payments

This function serves to compute the previous financial commitments:

- total debt installment payments on all loans
- total interest payments on all loans (each loan has a different interest rate)

```
for (i=0; i<LOANS.size; i++)
{
    TOTAL_INTEREST_PAYMENT              += (LOANS.array[i].interest_rate/12.0)
                                           * LOANS.array[i].loan_value;
    TOTAL_DEBT_INSTALLMENT_PAYMENT   += LOANS.array[i].installment_amount;
}
```

Firm compute income statement

```
//Compute net earnings
EARNINGS = CUM_REVENUE - TOTAL_INTEREST_PAYMENT - CALC_PRODUCTION_COSTS;

if (EARNINGS>0.0)
    TAX_PAYMENT = TAX_RATE_CORPORATE * EARNINGS;
else
    TAX_PAYMENT = 0.0;

PREVIOUS_NET_EARNINGS = NET_EARNINGS;
NET_EARNINGS = EARNINGS - TAX_PAYMENT;

//continue balance sheet (data pertaining to the period that just ended)
PREVIOUS_EARNINGS_PER_SHARE = EARNINGS_PER_SHARE;
if (CURRENT_SHARES_OUTSTANDING>0)
    EARNINGS_PER_SHARE = NET_EARNINGS/CURRENT_SHARES_OUTSTANDING;
```

Firm compute dividends

The dividend payout rule is: as long as earnings are positive, keep earnings per share constant. This yields the following dividend rule:

- If previous dividend was positive: the total divided payment increases with same rate as the earnings per share.
- If previous dividend was zero, and net earnings are positive: set new divided payment equal to a given fraction of net earnings.
- If net earnings are negative: set divided payment to zero.

```
if ((PREVIOUS_EARNINGS_PER_SHARE>0.0)&&(PREVIOUS_SHARES_OUTSTANDING>0))
{
    TOTAL_DIVIDEND_PAYMENT *= (EARNINGS_PER_SHARE/PREVIOUS_EARNINGS_PER_SHARE);
}

//Determine total_dividend if it was zero, and there are positive net earnings.
if(TOTAL_DIVIDEND_PAYMENT<1e-6 && NET_EARNINGS>0.0)
{
    TOTAL_DIVIDEND_PAYMENT = CONST_DIVIDEND_EARNINGS_RATIO * NET_EARNINGS;
}

//In case of negative earnings:
if (EARNINGS<0.0)
    TOTAL_DIVIDEND_PAYMENT = 0.0;
```

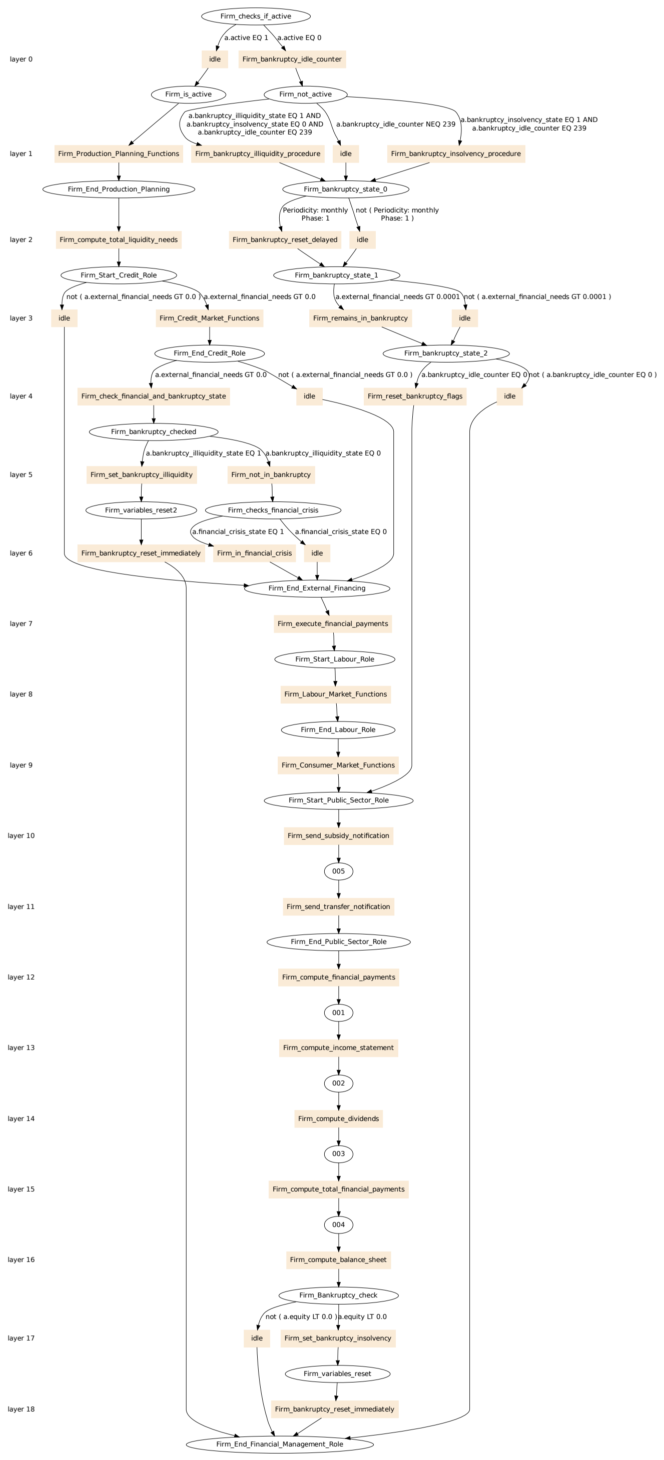
Firm compute balance sheet

We compute these values:

- total value capital stock
- total value inventory stock
- total assets
- equity
- financial indicators

```
TOTAL_VALUE_LOCAL_INVENTORY=0.0;
for (i=0; i<CURRENT_MALL_STOCKS.size; i++)
{
    TOTAL_VALUE_LOCAL_INVENTORY += PRICE_INDEX*CURRENT_MALL_STOCKS.array[i].current_stock;
}
TOTAL_ASSETS                = PAYMENT_ACCOUNT + TOTAL_VALUE_CAPITAL_STOCK
                                + TOTAL_VALUE_LOCAL_INVENTORY;
EQUITY                      = TOTAL_ASSETS - TOTAL_DEBT;
DEBT_EQUITY_RATIO           = TOTAL_DEBT/EQUITY;
DEBT_EARNINGS_RATIO         = TOTAL_DEBT/NET_EARNINGS;
EQUITY_ASSET_RATIO          = EQUITY/TOTAL_ASSETS;
```

5.2 Stategraph



Firm_checks_if_active
a.active EQ 1
a.active EQ 0
layer 0
idle
Firm_bankruptcy_idle_counter
Firm_is_active
Firm_not_active
a.bankruptcy_illiquidity_state EQ 1 AND a.bankruptcy_insolvency_state EQ 0 AND a.bankruptcy_idle_counter EQ 239
a.bankruptcy_idle_counter NEQ 239
a.bankruptcy_insolvency_state EQ 1 AND a.bankruptcy_idle_counter EQ 239
layer 1
Firm_Production_Planning_Functions
Firm_bankruptcy_illiquidity_procedure
idle
Firm_bankruptcy_insolvency_procedure
Firm_End_Production_Planning
Firm_bankruptcy_state_0
Periodicity: monthly Phase: 1
not (Periodicity: monthly Phase: 1)
layer 2
Firm_compute_total_liquidity_needs
Firm_bankruptcy_reset_delayed
idle
Firm_Start_Credit_Role
Firm_bankruptcy_state_1
not (a.external_financial_needs GT 0.0)
a.external_financial_needs GT 0.0
a.external_financial_needs GT 0.0001
not (a.external_financial_needs GT 0.0001)
layer 3
idle
Firm_Credit_Market_Functions
Firm_remains_in_bankruptcy
idle
Firm_End_Credit_Role
Firm_bankruptcy_state_2
a.external_financial_needs GT 0.0
not (a.external_financial_needs GT 0.0)
a.bankruptcy_idle_counter EQ 0
not (a.bankruptcy_idle_counter EQ 0)
layer 4
Firm_check_financial_and_bankruptcy_state
idle
Firm_reset_bankruptcy_flags
idle
Firm_bankruptcy_checked
a.bankruptcy_illiquidity_state EQ 1
a.bankruptcy_illiquidity_state EQ 0
layer 5
Firm_set_bankruptcy_illiquidity
Firm_not_in_bankruptcy
Firm_variables_reset2
Firm_checks_financial_crisis
a.financial_crisis_state EQ 1
a.financial_crisis_state EQ 0
layer 6
Firm_bankruptcy_reset_immediately
Firm_in_financial_crisis
idle
Firm_End_External_Financing
layer 7
Firm_execute_financial_payments
Firm_Start_Labour_Role
layer 8
Firm_Labour_Market_Functions
Firm_End_Labour_Role
layer 9
Firm_Consumer_Market_Functions
Firm_Start_Public_Sector_Role
layer 10
Firm_send_subsidy_notification
005
layer 11
Firm_send_transfer_notification
Firm_End_Public_Sector_Role
layer 12
Firm_compute_financial_payments
001
layer 13
Firm_compute_income_statement
002
layer 14
Firm_compute_dividends
003
layer 15
Firm_compute_total_financial_payments
004
layer 16
Firm_compute_balance_sheet
Firm_Bankruptcy_check
not (a.equity LT 0.0)
a.equity LT 0.0
layer 17
idle
Firm_set_bankruptcy_insolvency
Firm_variables_reset
layer 18
Firm_bankruptcy_reset_immediately
Firm_End_Financial_Management_Role

Figure 5.1: Stategraph.

Chapter 6

Bankruptcy Documentation

Acknowledgement

This text was written in collaboration with Marco Raberto, Andrea Teglio and Saul Desiderio.

This work was carried out in conjunction with the EURACE project (EU IST FP6 STREP grant 035086) which is a consortium lead by S. Cincotti (Università di Genova), H. Dawid (Universitaet Bielefeld), C. Deissenberg (Université de la Méditerranée), K. Erkan (TUBITAK National Research Institute of Electronics and Cryptology), M. Gallegati (Università Politecnica delle Marche), M. Holcombe (University of Sheffield), M. Marchesi (Università di Cagliari), C. Greenough (STFC - Rutherford Appleton Laboratory).

6.1 Introduction

The Eurace model considers two types of bankruptcy:

- Illiquidity bankruptcy: firm equity is positive but, after the ACM role and the AFM role, the firm is still unable to pay its financial commitments, i.e., taxes, debt instalments and interests; The firm is illiquid and has to find funds to resurface.
- Insolvency bankruptcy: the firm equity becomes negative. The firm is insolvent and has to perform a debt-to-equity transformation to become solvent again.

Both types of bankruptcy may occur in the Eurace economy. At first, we thought to address the two types of bankruptcy in the same way. However, during further discussions, it was clear that there were fundamental differences, e.g., the necessity of debt renegotiation in the insolvency bankruptcy case, which could not have been combined with the illiquidity bankruptcy in a unified bankruptcy model.

6.1.1 Effects on the markets

As soon as the firm declares bankruptcy, this has the following effects:

- All production activities are suspended.
- All workers are fired.
- The capital stock remains in place (and does not depreciate).
- The firm may enter into debt renegotiations with the banks at which it has outstanding loans, and some part of the loans may be written down (bad debt).

During the bankruptcy period:

- The firm remains idle for some time; this is an exogenous period of one year after which the firm is re-activated.
- The period can also be made endogenous, depending on some event-based activity of the firm, related to the financial market activities. For example the raising of new equity may take some time. In this case, the firm does not restart until all financial conditions are met.
- There is a memory variable in the firm: 'active' that takes on binary values 0 or 1. It is set to 1 when the firm's production role is active. When a firm is *'inactive'* it executes its bankruptcy procedures.

6.2 Bankruptcy in the full-fledged version of Eurace

Consider the following variable names for each firm:

- K: nominal value of physical capital at market prices
- I: market value of mall inventories
- C: payment account

- L: total loan
- $T_A = K + I + C$ (total assets)
- $E = T_A - L$ (equity)

Consider two parameters ω and ℓ, which are constant both in time and among firms; ω sets the debt renegotiation factor and ℓ the target leverage ratio of the restructured firm after bankruptcy.

6.2.1 Insolvency bankruptcy

The insolvency bankruptcy process of a firm consists of three main steps:

1. stop any business activity, fire all workers and cancel all equity shares among present shareholders;
2. debt renegotiation with the bank(s) at which the firm has current loans;
3. recapitalization of the firm by the issuance of new shares.

Debt renegotiation is addressed by re-scaling L, which is greater than T_A in the insolvency bankruptcy case, to a new value L^* lower than T_A, i.e., a value compatible with firms normal operations,

$$L^* = \omega T_A \quad \text{with} \quad \omega < 1\,. \tag{6.1}$$

Accordingly, we state that each bank j lending to the bankrupted firm is subject to a write-off w_j, i.e. a reduction of its assets (and consequently of its equity) proportional to the value of the loan L_j granted to the bankrupted firm. Given that the total amount of write-off among banks must be $L - L^*$ and that $\sum_j L_j = L$, we have:

$$w_j = \frac{L - L^*}{L} L_j\,. \tag{6.2}$$

Recapitalization is addressed by the issue of new shares. The rationale for raising new capital is twofold. First, it is intended to further strengthen the financial structure of the firm after the debt renegotiation, second, it is a simple way to allocate the claims on the equity capital of the restructured firm among new shareholders. The second issue could have been addressed by letting the creditor banks be the new shareholders, as a partial compensation for the write-off in their balance sheets, but this possibility was judged an unuseful complication and has been discarded. Furthermore, the chosen solution may set an endogenous idle time for the firm vefore restarting normal business operations, by letting the restart of production activity be contingent on having raised a sufficient amount of new capital on the stock market.

The sufficient amount of money is set by ℓ. Given the total firm assets before recapitalization T_A and the amount of money raised in the stock market m, ℓ sets the target leverage value of the restructured firm, i.e.,

$$\ell = \frac{L^*}{E^*}\,, \tag{6.3}$$

where $E^* = T_A + m - L^*$. The value of m is then derived accordingly, i.e.,

$$m = \max\left(0, \frac{L^*}{\ell} + L^* - T_A\right). \tag{6.4}$$

6.2.2 Illiquidity bankruptcy

The second form of bankruptcy is the illiquidity bankruptcy. The illiquidity bankruptcy process of a firm consists of two main steps:

1. stopping any business and firing of all workers (shareholders now retain their equity shares).
2. recapitalization of the bankrupt firm by the issue of new shares.

The illiquidity bankruptcy case is then similar in some respect to the insolvency bankruptcy case. The main difference is that here, L being already lower then T_A, the firm does not need to renegotiate its debt. Further, existing shareholders are no longer wiped out. However, due to the liquidity crisis which causes the firm to be unable to pay its financial commitments, business operations are stopped until the firm is able to raise a sufficient amount of funds m in the stock market by means of the issuing of new shares. We state that m must be equal to α times the difference between the firms financial commitments and its payment account, where α is a parameter (the target liquidity ratio).

6.3 Scaled down version of Eurace@Unibi

In the simplified version used in the Eurace@Unibi model we do treat both types of bankruptcy similarly. This has to do with the fact that in the simplified version of the financial market there are no individual firm stocks, only a market index. We do not allow firms to issue new shares, and no shares are cancelled. This has several consequences for the treatment of bankruptcy in the scaled-down version:

1. Existing shareholders retain their shares in the insolvency bankruptcy case. They continue to receive dividends when the firm resurfaces if it becomes solvent again.
2. Debt renegotiation with the bank (bad debt write-downs) also occurs in the case of illiquidity bankruptcy.
3. There is no recapitalization by issuing new shares in either insolvency or illiquidity bankruptcy.

6.4 Functions

6.4.1 Firm check financial and bankruptcy state

The first check for bankruptcy occurs inside the function Firm_check_financial_and_bankruptcy_state, which sets the following flags:

```
BANKRUPTCY_ILLIQUIDITY_STATE = 0 or 1
FINANCIAL_CRISIS_STATE = 1 or 0
```

```
    //Check bankrupcy condition for illiquidity case
    if (PAYMENT_ACCOUNT < TOTAL_FINANCIAL_NEEDS)
    {
        //Code: check if payment account is also less than financial payments
        if (PAYMENT_ACCOUNT >= TOTAL_INTEREST_PAYMENT
                + TOTAL_DEBT_INSTALLMENT_PAYMENT + TAX_PAYMENT-1e-5)
        {
            //Financial crisis condition
            FINANCIAL_CRISIS_STATE=1;
        }else
    {
            BANKRUPTCY_ILLIQUIDITY_STATE=1;
        }
    }
```

If the condition BANKRUPTCY_ILLIQUIDITY_STATE==0 is true, the execution flow continues to the function Firm_in_financial_crisis to resolve the financial crisis by reducing dividends. Otherwise, BANKRUPTCY_ILLIQUIDITY_STATE==1 and the flow goes to the function Firm_set_bankruptcy_illiquidity.

6.4.2 Firm in financial crisis

```
int Firm_in_financial_crisis()
{
    double payment_account_after_compulsory_payments;

    payment_account_after_compulsory_payments = PAYMENT_ACCOUNT
            - (TOTAL_INTEREST_PAYMENT + TOTAL_DEBT_INSTALLMENT_PAYMENT
                    + TAX_PAYMENT);

    //Try to resolve the crisis by lowering dividends
    TOTAL_DIVIDEND_PAYMENT = max(0, payment_account_after_compulsory_payments
            - PLANNED_PRODUCTION_COSTS);

    //Set flag if resolved:
    if (PAYMENT_ACCOUNT >= TOTAL_INTEREST_PAYMENT
            + TOTAL_DEBT_INSTALLMENT_PAYMENT + TAX_PAYMENT
            + TOTAL_DIVIDEND_PAYMENT)
    {
        FINANCIAL_CRISIS_STATE=0;
        BANKRUPTCY_STATE=0;
    }
```

6.4.3 Firm set bankruptcy illiquidity

This function sets the following flags:

```
ACTIVE=0
BANKRUPTCY_IDLE_COUNTER = CONST_BANKRUPTCY_IDLE_PERIOD
BANKRUPTCY_INSOLVENCY_STATE  = 0
BANKRUPTCY_ILLIQUIDITY_STATE = 1

//Send msg to malls to remove inventory stock:
add_bankruptcy_illiquidity_message(ID);
```

and then goes directly to the end of the iteration. In the next iteration the flag BANKRUPTCY_ILLIQUIDITY_STATE=1 causes the actual bankruptcy procedure to be executed for the case of the illiquidity bankruptcy. See the function Firm_bankruptcy_illiquidity_procedure for details.

6.4.4 Firm set bankruptcy insolvency

The second check for bankruptcy occurs later on in the iteration, after the function Firm_compute_balance_sheet has been executed. That function computes the value of equity. If equity< 0 the function Firm_set_bankruptcy_insolvency is run, which sets the following flags:

```
ACTIVE=0
BANKRUPTCY_IDLE_COUNTER = CONST_BANKRUPTCY_IDLE_PERIOD
BANKRUPTCY_INSOLVENCY_STATE  = 1
BANKRUPTCY_ILLIQUIDITY_STATE = 0

//Send msg to malls to remove inventory stock:
add_bankruptcy_insolvency_message(ID);
```

and then goes directly to the end of the iteration. In the next iteration the flag BANKRUPTCY_INSOLVENCY_STATE=1 causes the actual bankruptcy procedure to be executed for the case of the insolvency bankruptcy. See the function Firm_bankruptcy_insolvency_procedure for details.

6.4.5 Firm bankruptcy insolvency/illiquidity procedure

The procedure consists of the following steps:

1. Effect on the credit market by renegotiating all bank loans:
   (a) Debt-to-equity transformation: The old debt is rescaled to a fraction of the current total assets.
   (b) Defaulting on a fraction of each bank loan by the write_off_ratio. This is the bad debt that should be written off from the bank's balance sheet.
2. Effect on the labour market by firing all workers.
3. Effect on the goods market: all local inventory stock at the mall is destroyed.

The first step is to set the new target debt as a ratio of the resulting total assets:

```
//We only perform these calculations once, at the start of the bankruptcy procedure
if (BANKRUPTCY_IDLE_COUNTER == CONST_BANKRUPTCY_IDLE_PERIOD - 1)
{
  //Recompute total assets after inventory stock has been deleted
   TOTAL_ASSETS = TOTAL_VALUE_CAPITAL_STOCK + PAYMENT_ACCOUNT;

  //Set the target debt
  target_debt = DEBT_RESCALING_FACTOR*TOTAL_ASSETS;
  write_off_ratio = (TOTAL_DEBT - target_debt)/TOTAL_DEBT;
}
```

The DEBT_RESCALING_FACTOR is an environment constant that is the same for all firms and is set globally. It sets the ratio of target_debt/total_assets at the beginning of the debt-to-equity transformation procedure. This constant is not a decision variable since it is exogenously given.

We assume there is a debt renegotiation with the bank. For each loan, the firm computes the fraction of the loan that should be written-off from the bank's balance sheet. The write_off_ratio is given by the fraction $(L - L^*)/L$ of the loan value L_j.

Listing 6.1: Bankruptcy procedure, step 1: Effect on credit market.

```
//We only perform these calculations once, at the start of the bankruptcy procedure
if (BANKRUPTCY_IDLE_COUNTER == CONST_BANKRUPTCY_IDLE_PERIOD - 1)
{
  //Effect on credit market:
  //Renegotiating debt: refunding credit, computing bad debt
  for (i=0; i<LOANS.size; i++)
  {
      //Computing bad debt
      bad_debt = write_off_ratio*LOANS.array[i].loan_value;

    //Compute value at risk to be written off:
      writeoff_var = write_off_ratio *
              LOANS.array[i].var_per_installment * LOANS.array[i].nr_periods_before_repayment;

      LOANS.array[i].var_per_installment = (1-write_off_ratio)
                              *LOANS.array[i].var_per_installment;
      LOANS.array[i].loan_value = (1-write_off_ratio)*LOANS.array[i].loan_value;

      LOANS.array[i].installment_amount = LOANS.array[i].loan_value
                          /LOANS.array[i].nr_periods_before_repayment;

    //Send the bankruptcy_message to write off the bad debt
    //add_bankruptcy_message(firm_id, bank_id, bad_debt, credit_refunded, writeoff_var);
    add_bankruptcy_message(ID, LOANS.array[i].bank_id, bad_debt, credit_refunded, writeoff_var)
     }
  //Recompute balance sheet after the write-off
  TOTAL_DEBT = target_debt;
  EQUITY = TOTAL_ASSETS - TOTAL_DEBT;
}
```

Listing 6.2: Bankruptcy procedure Step 2: Effect on labour market.

```
//Effect on labour market
//Firing all employees
    for (i=0;i<EMPLOYEES.size;i++)
    {
            add_firing_message(ID, EMPLOYEES.array[i].id);
            remove_employee(&EMPLOYEES, i);
    }
```

Firm remains in bankruptcy

- Function to signal that a firm remains in bankruptcy if the debt write-off is not enough to get the equity positive. The firm cannot issue new shares on the financial market.

6.5 Stategraph

See the Financial Management documentation.

Chapter 7

Credit Market Documentation

Acknowledgement

The description of the credit market is largely adapted from documentation provided by our colleagues at the Universitá Polytechnica de la Marche in Ancona (UPM).

7.1 Bank

7.1.1 Activities

Banks perform the following activities in the credit market:

- Manage payment accounts for households and firms; the deposits pay an interest rate slightly below the base rate.
- Provide credit to firms, at an interest rate that is slightly above the base rate and depends on the firm's credit worthiness.
- Access a standing facility at the Central Bank, allowing them to borrow when their cash reserves become negative. The bank pays the base rate on overnight loans, and receives the base rate on overnight deposits at the Central Bank.

7.1.2 Functions

Bank read policy rate

The ECB base interest rate is a constant in the model. The banks use this base rate to set two interest rates:

- the interest rate on firm loans: the bank uses the ECB base rate plus a firm specific markup that depends on the firms balance sheet.
- the interest rate on deposits from households and firms.

```
DEPOSIT_INTEREST_RATE = (1-ECB_INTEREST_RATE_MARKDOWN) * ECB_INTEREST_RATE;
```

Bank send account interest

Step 1. The bank has to pay daily interest at the base rate on its outstanding ECB debt, respectively it receives interest at the base rate on its overnight deposits at the ECB. It sends a message to the ECB to this affect.

```
    //Pay interests to ecb on ECB debt of day before
     int_to_ecb   = ECB_DEBT*ECB_INTEREST_RATE/240.0;

    //Receive interest on cash held at ECB
     int_from_ecb  = CASH*ECB_INTEREST_RATE/240.0;

    //Flow accounting
     ECB_INTEREST_PAYMENT = int_to_ecb - int_from_ecb;
     BANK_OUTFLOWS_CALENDAR.ecb_interest_payment += int_to_ecb - int_from_ecb;

    //Subtract interest paid to ECB on ECB debt
     PROFITS[0] -= int_to_ecb;
     CASH       -= int_to_ecb;
     EQUITY     -= int_to_ecb;

    //add interest received from ECB on deposits
     PROFITS[0] += int_from_ecb;
     CASH       += int_from_ecb;
     EQUITY     += int_from_ecb;

    //send interest to ECB
     add_bank_interest_payment_message(int_to_ecb - int_from_ecb);
```

Step 1. The bank pays a daily deposit interest rate (base rate minus a markdown) on household and firm accounts by sending the account_interest message. The bank does not have knowledge of each individual account, but if agents know the interest rate they know the new level of their own account.

```
  //Pay daily deposit interest
     interest = (1/240.0)*DEPOSIT_INTEREST_RATE*DEPOSITS;
     CASH    -= interest;
     EQUITY   -= interest;

    //Send interest
     add_account_interest_message(ID, DEPOSIT_INTEREST_RATE);
```

Bank decide credit conditions

Every day the banks receive messages from firms with loan requests. For each individual request the bank records several firm characteristics: equity (e), debt (d), and credit demand (c).

If the bank has no debt with the Central Bank and meets the Basel II capital requirements (the value-at-risk is below some threshold level), it decides on the credit conditions for the applying firm. This consists of the amount of credit provided and the interest rate on the new loan. The bank's ability to provide credit is constrained by the minimal capital requirement (defines the available excess VaR) and by the minimal cash reserve ratio (defines the available excess liquidity).

```
EXCESS_VAR         = ALFA*EQUITY - VALUE_AT_RISK;
EXCESS_LIQUIDITY   = CASH - MIN_CASH_RESERVE_RATIO*DEPOSITS;
```

```
START_LOAN_REQUEST_MESSAGE_LOOP
  equity = loan_request_message->equity;
  debt = loan_request_message->total_debt;
  credit_requested = loan_request_message->external_financial_needs;

  MIN_INTEREST = ECB_INTEREST_RATE;
  bankruptcy_prob = 1-exp(-(debt+credit_request)/equity);
  new_value_at_risk = bankruptcy_prob*credit_requested;

  //1. VaR of requested loan is within available excess VaR of bank
    if ( new_value_at_risk <= EXCESS_VAR )
    {
        credit_allowed  = credit_requested;
        EXCESS_VAR     -= new_value_at_risk;
    }else
    {
      //2. VaR of requested loan exceeds available excess VaR of bank
        credit_allowed = EXCESS_VAR/bankruptcy_prob;
    }
    interest = MIN_INTEREST + BANK_GAMMA[0]*new_value_at_risk
          * 0.01*random_double(0.0,1.0);

  //Send response to firm with credit conditions
  //constrained to the excess liquidity still available
  if (EXCESS_LIQUIDITY>credit_allowed)
  {
    //credit_allowed = credit_allowed;
    EXCESS_LIQUIDITY -= credit_allowed;
  }
  else
  {
    credit_allowed = EXCESS_LIQUIDITY;
    EXCESS_LIQUIDITY =0.0;
  }
  //Only make loan offers for positive credit request
  if (credit_allowed > 1e-6)
  {
    value_at_risk_of_loan = r*(credit_allowed/c);
    add_loan_conditions_message(loan_request_message->firm_id, ID, i, credit_allowed,
value_at_risk_of_loan);          }
FINISH_LOAN_REQUEST_MESSAGE_LOOP
```

Bank give loan

The bank provides a loan to all firms that have accepted the loan conditions (firms may have applied to multiple banks for the same loan, so they either accept or reject the loan conditions, see the Firm functions). If the loan is granted, the bank updates its balance sheet (loans are provided out of cash reserves), their loan portfolio (total amount of credit outstanding) and the value-at-risk is increased with the loan specific value.

```
START_LOAN_ACCEPTANCE_MESSAGE_LOOP
 if(loan_acceptance_message->bank_id==ID)
 {
  CASH          -= loan_acceptance_message->credit_amount_taken;
  TOTAL_CREDIT  += loan_acceptance_message->credit_amount_taken;
  VALUE_AT_RISK += loan_acceptance_message->loan_total_var;
 }
FINISH_LOAN_ACCEPTANCE_MESSAGE_LOOP
```

Bank receive installment

Loan installments are received as cash payments, so the bank accounts them on its cash reserves (liquidity). The interest is accounted on the profit and equity of the bank.

Step 1. With daily frequency banks receive messages with interest and debt installment payments from debtor firms. Internal bank variables and the balance sheet are updated accordingly.

```
START_INSTALLMENT_MESSAGE_LOOP
   if(installment_message->bank_id==ID)
   {
      //Add debt installment
      CASH += installment_message->installment_amount;

       //Add interest:
      CASH      += installment_message->interest_amount;
      PROFITS[0]   += installment_message->interest_amount;
      EQUITY      += installment_message->interest_amount;

      //Subtract debt installment (payment of principle)
      TOTAL_CREDIT   -= installment_message->installment_amount;
      VALUE_AT_RISK    -= installment_message->var_per_installment;
   }
FINISH_INSTALLMENT_MESSAGE_LOOP
```

Step 2. A second purpose of this function is to record if some of the borrowing firms have entered bankruptcy. If this happens the client firm defaults on its debt and the bank suffers a bad debt that has to be written down on the bank's balance sheet.

```
START_BANKRUPTCY_MESSAGE_LOOP
   if(bankruptcy_message->bank_id==ID)
   {
       total_bad_debt   += bankruptcy_message->bad_debt;
      EQUITY        -= bankruptcy_message->bad_debt;
      TOTAL_CREDIT   -= bankruptcy_message->bad_debt;
      VALUE_AT_RISK    -= bankruptcy_message->residual_var;
   }
FINISH_BANKRUPTCY_MESSAGE_LOOP
```

Bank account update deposits

At the end of ever iteration (daily) the banks need to update the balance of the payment accounts of households and firms according to their transactions that occurred during the day. At the end of every day all money in the private sector is therefore deposited in the banking system.

```
DEPOSITS=0;
START_BANK_ACCOUNT_UPDATE_MESSAGE_LOOP
     if (bank_account_update_message->bank_id==ID)
     {
        DEPOSITS += bank_account_update_message->payment_account;
     }
```

```
FINISH_BANK_ACCOUNT_UPDATE_MESSAGE_LOOP

if(DEPOSITS - old_deposits >= 0)
{
  BANK_INFLOWS_CALENDAR.deposit_inflow += (DEPOSITS - old_deposits);
}
else
{
  BANK_OUTFLOWS_CALENDAR.deposit_outflow += -(DEPOSITS - old_deposits);
}
//Add deposit mutation to cash reserves
CASH += (DEPOSITS - old_deposits);
```

Bank accounting

At the end of the month the bank does basic accounting.

Step 1. Computing profits, set dividend payments, and pay taxes.

```
if (PROFITS[1]>0)
{
    growth=( (PROFITS[0]-PROFITS[1])/PROFITS[1] );
}
else
    growth=0;

q=BANK_GAMMA[0];
c=BANK_GAMMA[1];

//Behavioral parameters:
BANK_GAMMA[1]=    q;
BANK_GAMMA[0]=(q+(BANK_LAMBDA*(q-c)*growth)+0.01* random_double(0.0,1.0);
if (BANK_GAMMA[0]<0.02)
{
  BANK_GAMMA[0]=0.02;
}
```

Step 2. Pay taxes and dividends.

```
//Dividend rate
if (EQUITY<VALUE_AT_RISK/ALFA)
     BANK_DIVIDEND_RATE = 0.0;
else
    BANK_DIVIDEND_RATE = 1.0;

if (PROFITS[0]>0)
{
      TAXES      = TAX_RATE_CORPORATE*PROFITS[0];
      PROFITS[0] -= TAXES;
      EQUITY     -= TAXES;
      CASH       -= TAXES;

      add_tax_payment_message(GOV_ID, TAXES);

    TOTAL_DIVIDEND_PAYMENT = BANK_DIVIDEND_RATE*PROFITS[0];
}
```

Bank send dividend payment

The bank sends its monthly dividend payments to the Clearinghouse, who centrally pays dividends on the market index (see financial market documentation).

```
    add_dividend_info_message(ID, TOTAL_DIVIDEND_PAYMENT);
    EQUITY -= TOTAL_DIVIDEND_PAYMENT;
    CASH -= TOTAL_DIVIDEND_PAYMENT;
```

Bank update ecb account

If the bank's cash outflows exceeds its cash inflow it may occur that the cash reserves drop below the minimin cash reserve requirement and the bank is illiquid. It is then forced to resort to the Central Bank and rely on a standing facility, i.e. ECB debt to replenish its cash reserves. We assume the Central Bank is fully commodating the banks' demand for liquidity and a flow of money from the Central bank to such illiquid commercial banks is generated.

```
    // Procedure to add ECB debt daily
  EXCESS_LIQUIDITY = CASH - MIN_CASH_RESERVE_RATIO*DEPOSITS;

  if (EXCESS_LIQUIDITY<0.0)
  {
    //Monetary base is increased
    ECB_DEBT    += -EXCESS_LIQUIDITY;
    CASH     += -EXCESS_LIQUIDITY;
    EXCESS_LIQUIDITY = 0.0;
  }
  // Procedure to reduce ECB debt
  else if ( (ECB_DEBT>0.0)&& (EXCESS_LIQUIDITY>0.0) )
  {
    //Case 1: Sufficient excess liquidity to fully repay ECB debt
    a=1.0;
    if(EXCESS_LIQUIDITY>=ECB_DEBT)
    {
      CASH  -= a*ECB_DEBT;
      ECB_DEBT = (1-a)*ECB_DEBT;
      EXCESS_LIQUIDITY = CASH - MIN_CASH_RESERVE_RATIO*DEPOSITS;
    }
    //Case 2: Insufficient excess liquidity: repay partially, deplete excess
    b=1.0;
    if(EXCESS_LIQUIDITY<ECB_DEBT)
    {
      CASH     -= b*EXCESS_LIQUIDITY;
      ECB_DEBT  -= b*EXCESS_LIQUIDITY;
      EXCESS_LIQUIDITY = CASH - MIN_CASH_RESERVE_RATIO*DEPOSITS;
    }
  }
```

7.2 Firm

Firms need to enter the credit market whenever their internal resources are insufficient to cover their planned expenditures. They switch from the financial management role to the credit role.

7.2.1 Activities

Firms (and IGFirm) perform the following activities in the credit market:

- Access a bank account to deposit all liquid funds, on which they receive the deposit interest rate.
- Request credit from banks, on which they pay an interest rate that is firm specific and can differ per loan. A firm can request loans at multiple banks, and decides to accept or reject a loan depending on the credit conditions.

7.2.2 Functions

Firm receive account interest

Firms receive the deposit interest rate on their money holdings. Although the interest is an annual rate, they receive a daily interest depending on the daily level of the payment account.

```
START_ACCOUNT_INTEREST_MESSAGE_LOOP
  if(account_interest_message->bank_id == BANK_ID)
    interest_rate = account_interest_message->interest_rate;
FINISH_ACCOUNT_INTEREST_MESSAGE_LOOP

// Add daily interest on deposits: (1/240) of the interest rate
  interest = (1/240.0)*interest_rate*PAYMENT_ACCOUNT;
  PAYMENT_ACCOUNT += interest;
```

Firm ask loan

Whenever a firm needs external financing, it searches for the active banks that are able to lend to them. This function creates the list of potential lending banks that the firm can contact. It creates the firm-bank credit network in which every firm can contact up to N_b banks.

```
//Search for active banks' name
START_BANK_IDENTITY_MESSAGE_LOOP
    add_potential_lender(&SET_OF_LENDERS, bank_identity_message->bank_id ,0);
    NUMBER_OF_BANKS_ASKED++;
FINISH_BANK_IDENTITY_MESSAGE_LOOP

if (EXTERNAL_FINANCIAL_NEEDS>0.0)
{
  connected=0;

  //Create bank network for this firm
  while(connected<NUMBER_OF_BANKS_TO_APPLY)
  {
    j= rand() % CONST_NUMBER_OF_BANKS; //choose banks randomly
    add_loan_request_message(ID, SET_OF_LENDERS.array[j].bank_name, EQUITY,
                                 TOTAL_DEBT, EXTERNAL_FINANCIAL_NEEDS);
    connected++;
  }
}
```

Firm get loan

This is the firm's procedure for obtaining a new loan.

Step 1. Firms receive loan conditions from all banks to which is sen a loan request message.

```
//Read messages from banks
START_LOAN_CONDITIONS_MESSAGE_LOOP
  if (loan_conditions_message->firm_id==ID)
  {
      bk = loan_conditions_message->bank_id;
      interest_array[bk] = loan_conditions_message->proposed_interest_rate;
      credit_offer_array[bk] = loan_conditions_message->amount_offered_credit;
      rate_order_array[bk] = loan_conditions_message->bank_id;
      value_at_risk_array[bk] = loan_conditions_message->value_at_risk;
  }
FINISH_LOAN_CONDITIONS_MESSAGE_LOOP
```

Step 2. Firms examine the loan conditions stated by banks and sort them according to interest rates (lowest interest first):

```
for(i=0;i<CONST_NUMBER_OF_BANKS-1;i++)
{
  for(k=i+1; k<CONST_NUMBER_OF_BANKS; k++)
  {
    if (interest_array[i]>interest_array[k])
    {
      aux=interest_array[i];
      interest_array[i]=interest_array[k];
      interest_array[k]=aux;

      aux=credit_offer_array[i];
      credit_offer_array[i]=credit_offer_array[k];
      credit_offer_array[k]=aux;

      aux=value_at_risk_array[i];
      value_at_risk_array[i]=value_at_risk_array[k];
      value_at_risk_array[k]=aux;

      n1=rate_order_array[i];
      rate_order_array[i]=rate_order_array[k];
      rate_order_array[k]=n1;
    }
  }
}
```

Step 3. Accept loans in order of rank, lowest interest rate first, and continue to take out loans until the external financial needs are satisfied, or the loan offers are exhausted.

```
//Travers the banks according to the order in the rate_order_array,
//obtain a loan if credit_demand >= credit_offer
for(i=0; i<NUMBER_OF_BANKS_ASKED; i++)
{
  if (rate_order_array[i]!=-1)
  {
    credit_demand = EXTERNAL_FINANCIAL_NEEDS - total_credit_taken;
    //Accept the credit:
    if (credit_demand >= credit_offer_array[i])
```

```
      {
        credit_accepted = credit_offer_array[i];
      }
      else
      {
        credit_accepted=credit_demand;
      }
      //Set loan values:
      total_credit_taken += credit_accepted;
      bank_id         = rate_order_array[i];
      loan_value       = credit_accepted;
      interest_rate      = interest_array[i];
      installment_amount  = credit_accepted/CONST_INSTALLMENT_PERIODS;
      interest_amount    = interest_rate*installment_amount;

      residual_var     = value_at_risk_array[i]
                     *(credit_accepted/credit_offer_array[i]);
      var_per_installment = residual_var/CONST_INSTALLMENT_PERIODS;
      bad_debt       = 0.0;
      nr_periods_before_repayment = CONST_INSTALLMENT_PERIODS+1;
    }
  }
```

Step 4. Firms add to their loan portfolio a new 'debt' item with all the relevant attributes of the loan they are about to accept:

```
if (credit_accepted >0.0)
{
  add_debt_item(&LOANS, bank_id, loan_value, interest_rate, installment_amount,
  var_per_installment, residual_var, bad_debt, nr_periods_before_repayment);
}
```

Step 5. Finally, the firm communicates the decision to accept the loan to the issuing bank:

```
add_loan_acceptance_message(bank_id, credit_accepted, residual_var);
```

7.3 Central Bank

7.3.1 Activities

The Central Banks performs the following activities in the credit market:

- Manage payment accounts for the banks and Governments.
- Provide standing facilities for the banks. We assume the Central Bank fully accommodates the banks' requests for liquidity.
- Pay interest on banks' overnight deposits.
- Provide fiat money to government (in case of direct monetization of debt)

7.3.2 Functions

Central Bank read fiat money requests

CB search for liquidity requests from commercial banks and Government:

```
START_REQUEST_FIAT_MONEY_MESSAGE_LOOP
   FIAT_MONEY_GOVS += request_fiat_money_message->nominal_value;
   FIAT_MONEY      += request_fiat_money_message->nominal_value;
FINISH_REQUEST_FIAT_MONEY_MESSAGE_LOOP
```

Central Bank read account update

Step 1. Update the current account of all banks at the Central Bank.

```
START_BANK_TO_CENTRAL_BANK_ACCOUNT_UPDATE_MESSAGE_LOOP
  ECB_DEPOSITS += bank_to_central_bank_account_update_message->payment_account;

  //Search for the correct account and update the value
  for (i=0;i<ACCOUNTS_BANKS.size;i++)
  {
    if(ACCOUNTS_BANKS.array[i].id == bank_to_central_bank_account_update_message->id)
    {
      ACCOUNTS_BANKS.array[i].payment_account =
        bank_to_central_bank_account_update_message->payment_account;
    }
  }
FINISH_BANK_TO_CENTRAL_BANK_ACCOUNT_UPDATE_MESSAGE_LOOP
```

Step 2. Add interest payments from bank to ECB. This is the net interest between interest on ECB debt and interest on Bank cash reserves held as overnight deposits at ECB.

```
interest = 0.0;
START_BANK_INTEREST_PAYMENT_MESSAGE_LOOP
  interest += bank_interest_payment_message->bank_interest_amount;
FINISH_BANK_INTEREST_PAYMENT_MESSAGE_LOOP

CASH += interest;
```

Step 3. Update the current account of all Governments at the Central Bank.

```
START_GOV_TO_CENTRAL_BANK_ACCOUNT_UPDATE_MESSAGE_LOOP
  ECB_DEPOSITS += gov_to_central_bank_account_update_message->payment_account;

  //Search for the correct account and update the value
  for (i=0;i<ACCOUNTS_GOVS.size;i++)
  {
    if(ACCOUNTS_GOVS.array[i].id == gov_to_central_bank_account_update_message->id)
    {
      ACCOUNTS_GOVS.array[i].payment_account =
        gov_to_central_bank_account_update_message->payment_account;
    }
  }
FINISH_GOV_TO_CENTRAL_BANK_ACCOUNT_UPDATE_MESSAGE_LOOP
```

Step 4. In case of positive ECB equity, the ECB makes a profit, and should distribute this evenly among governments.

```
no_govs = TOTAL_REGIONS/NO_REGIONS_PER_GOV;

//ECB pays out all CASH as dividends
dividend_per_gov = CASH/no_govs;

//ECB adds dividends to Governments' payment accounts
for (j=0; j<ACCOUNTS_GOVS.size; j++)
  ACCOUNTS_GOVS.array[j].payment_account += dividend_per_gov;

//ECB sends ecb_dividend_payment_msg, Governments adjust payment accounts
add_ecb_dividend_message(dividend_per_gov);
```

7.4 Stategraph

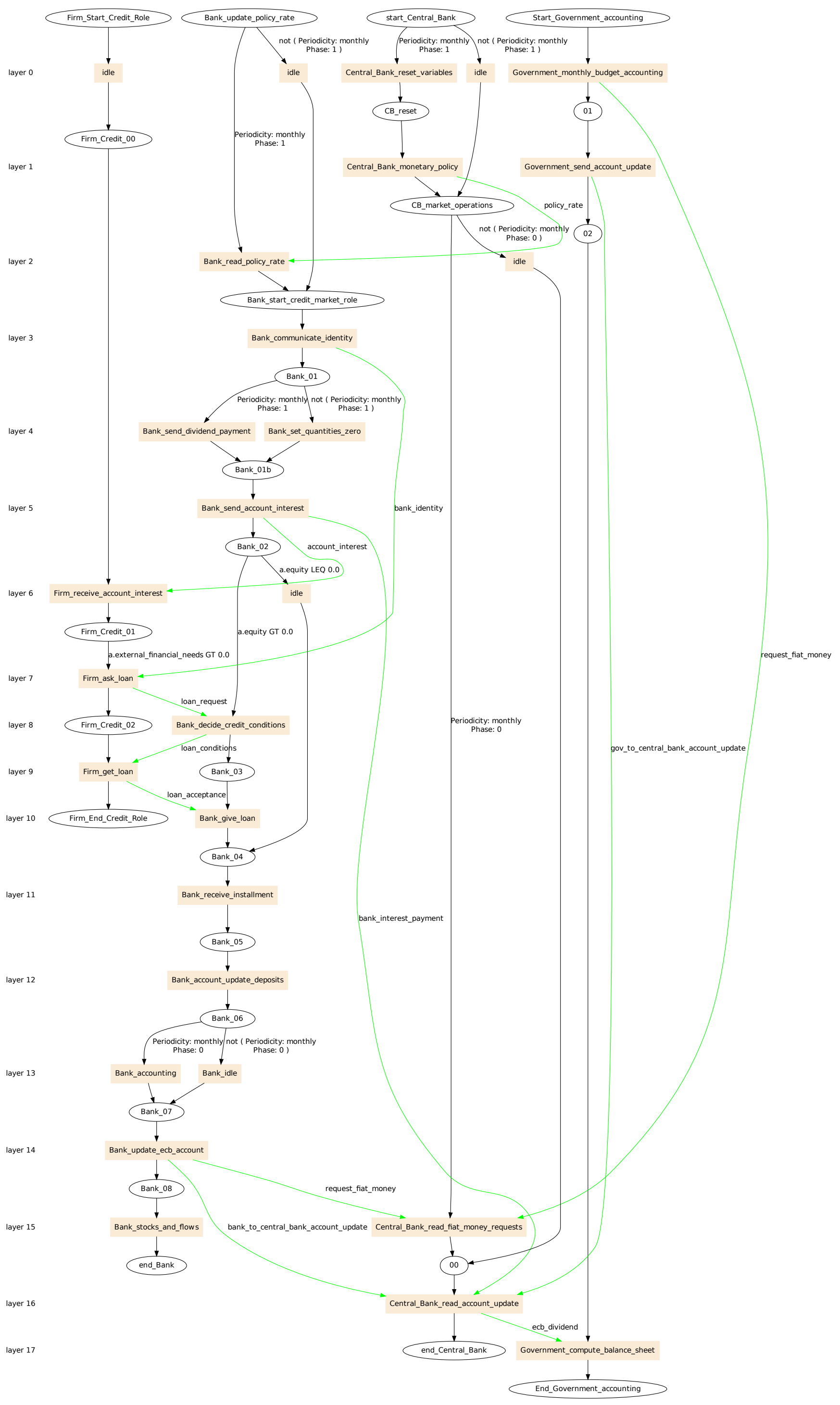

Firm_Start_Credit_Role
Bank_update_policy_rate
start_Central_Bank
Start_Government_accounting
not (Periodicity: monthly Phase: 1)
Periodicity: monthly Phase: 1
not (Periodicity: monthly Phase: 1)
layer 0
idle
idle
Central_Bank_reset_variables
idle
Government_monthly_budget_accounting
CB_reset
01
Firm_Credit_00
Periodicity: monthly Phase: 1
layer 1
Central_Bank_monetary_policy
Government_send_account_update
CB_market_operations
policy_rate
not (Periodicity: monthly Phase: 0)
02
layer 2
Bank_read_policy_rate
idle
Bank_start_credit_market_role
layer 3
Bank_communicate_identity
Bank_01
Periodicity: monthly Phase: 1
not (Periodicity: monthly Phase: 1)
layer 4
Bank_send_dividend_payment
Bank_set_quantities_zero
Bank_01b
layer 5
Bank_send_account_interest
bank_identity
Bank_02
account_interest
a.equity LEQ 0.0
layer 6
Firm_receive_account_interest
idle
Firm_Credit_01
a.equity GT 0.0
a.external_financial_needs GT 0.0
request_fiat_money
layer 7
Firm_ask_loan
loan_request
layer 8
Firm_Credit_02
Bank_decide_credit_conditions
Periodicity: monthly Phase: 0
loan_conditions
gov_to_central_bank_account_update
layer 9
Firm_get_loan
Bank_03
loan_acceptance
layer 10
Firm_End_Credit_Role
Bank_give_loan
Bank_04
layer 11
Bank_receive_installment
bank_interest_payment
Bank_05
layer 12
Bank_account_update_deposits
Bank_06
Periodicity: monthly Phase: 0
not (Periodicity: monthly Phase: 0)
layer 13
Bank_accounting
Bank_idle
Bank_07
layer 14
Bank_update_ecb_account
Bank_08
request_fiat_money
layer 15
Bank_stocks_and_flows
bank_to_central_bank_account_update
Central_Bank_read_fiat_money_requests
end_Bank
00
layer 16
Central_Bank_read_account_update
ecb_dividend
layer 17
end_Central_Bank
Government_compute_balance_sheet
End_Government_accounting

Figure 7.1: Stategraph for the credit market.

Chapter 8

Financial Market Documentation

8.1 Messages

Household order message:

- Households send trading orders to the financial market on their activation day.
- Every day the Clearinghouse reads the trading orders, computes the aggregate demand and supply for index shares, performs market rationing, and sends back order status messages to the individual traders.

Dividend message:

- Firms determine their total dividend payment on their day of month to act (first day of their production cycle).
- Firms send the dividend info message (containing the total dividend) to the Clearinghouse on the day of month to act. The Clearinghouse then aggregates the total dividends during the calendar month and computes the dividend that is paid on the market index shares.
- Since the subjective month and the calendar month differ by maximum 19 days, there is asynchronicity between the time at which firms subtract the dividend payments from their bank account (on their day of month to act) and the time at which households add the dividends to their bank accounts (on teh first day of the calendar month).

Market index info message:

- First day of the calendar month: Given the index composition, the Clearinghouse computes the dividend per index share, given the current information it has on the firms' dividend payment.
- On the first day of the calendar month the Clearinghouse sends the index info message containing the dividend per share.
- Households read this message and given the number of index shares they own, add the total amount of dividend to their bank account.

Market index price message:

- The Clearinghouse sends a daily price info message containing the price of the index shares.
- Households read this message on their activation day, before they revise their planned asset portfolio. This daily index price affects the current market value of the current asset portfolio, which counts as wealth, and hence influences the asset allocation decision.

8.2 Constants

trading_activity Swithes on/off trading activity in index shares:

- trading_activity=0: trading of shares is off. The asset trading branch is not executed. Household do receive dividends on the index shares they own.
- trading_activity=1: trading of shares is on. Households read the index price and submit orders on their monthly activation day only.

trading_random Sets the asset allocation decision to random orders or to orders according to the logit choice model.

- trading_random=1: Households allocate a random fraction (0,1) of their asset budget to the index. The remainder is deposited on the bank account.
- trading_random=0: households use a method to determine the fraction of their asset budget to allocate to the index.

index_price_adj Parameter for index price adjustment. Sets the speed of adjustment in the price updating mechanism.

- index_price_adj=1: Scaling parameter.
- index_price_adj_min=0.90: floor level on price decrease -10%.
- index_price_adj_max=1.10: ceiling level on price increase $+10\%$.

8.3 Firm

8.3.1 Activities

- Firms ($f = 1, ..., F$) have a fixed number of outstanding equity shares, but do not issue new shares. The outside supply of shares is therefore zero.
- Firms pay out a dividend on their shares y_t^f that depends on their profits.

8.3.2 Functions

Firm execute financial payments

Runs monthly, on the firm's activation day.

- Firms send a dividend_info message that contains the total dividend payment to the Clearinghouse. This message is sent in the function Firm_execute_financial_payments that is in the financial management of the firm, at the same location where the firm subtracts the total dividend payment from the payment account.

```
    //send total_dividend_payment and decrease payment_account
    add_dividend_info_message(ID, TOTAL_DIVIDEND_PAYMENT);
    PAYMENT_ACCOUNT -= TOTAL_DIVIDEND_PAYMENT;
```

8.4 Households

8.4.1 Activities

- Households save or withdraw deposits from their bank acount.
- Households ($h = 1, ..., H$) can invest in equity shares by submitting buy and sell orders to the Clearing house.
- Households decide their asset allocation after their consumption budget is determined: the total monetary value they want to hold in their asset portfolio (s, a) is determined in the consumption model. This leads to re-adjustment of their current asset portfolio.
- Households receive a dividend income depending on the number of index shares they hold in portfolio.

8.4.2 Functions

Household receive index info

Runs monthly, on first day of the calendar month.

- Set the monthly consumption counter to zero at the first day of the month.
- Set the deposit interest rate to ECB base rate.
- Read the index info message from the Clearinghouse, add dividend income to bank account.

```
RECEIVED_DIVIDEND=0;

//Household sets its monthly consumption counter to zero at the first day of the month
if(DAY%MONTH==1)
    MONTHLY_CONSUMPTION_EXPENDITURE = 0.0;

//Set deposit rate to ECB base rate
RISK_FREE_RATE = ECB_INTEREST_RATE;

START_INDEX_INFO_MESSAGE_LOOP
    dividend = index_info_message->dividend_per_share*ASSETSOWNED.units;
FINISH_INDEX_INFO_MESSAGE_LOOP
```

Household receive index price

Runs monthly, on household's activation day.

- Read index info message from Clearinghouse.
- Store current index price and the moving average price.

```
START_INDEX_PRICE_MESSAGE_LOOP
    ASSETSOWNED.lastprice = index_price_message->price;
    ASSETSOWNED.moving_avg_price = index_price_message->moving_avg_price;
FINISH_INDEX_PRICE_MESSAGE_LOOP
```

Household revises expected portfolio

Runs monthly, on household's activation day.

Input: payment_account, consumption_budget.

Purpose: Determine the value of the planned asset portfolio, to be allocated among the risk-free and risky asset (can be negative if shares need to be sold to accomodate for the consumption budget).

Step 1. Compute the value of the planned asset portfolio (deposits + index).

```
portfolio_budget = PAYMENT_ACCOUNT
                   + ASSETSOWNED.lastprice*ASSETSOWNED.units
                   - CONSUMPTION_BUDGET;
```

Step 2. Compute composition of the new planned portfolio, using mean-variance maximization framework.

1. Update beliefs on excess return $E_{ht}[\rho_{t+1}] - r$, and volatility $V_{ht}[\rho_{t+1} - r]$.

```
        BELIEFS.expected_return
        BELIEFS.expected_volatility
```

2. Set the optimal proportion π_{ht} of the portfolio budget to invest in the index (the risky asset). This is the net investment, the remainder stays in deposits.

$$\pi_{ht} = \frac{E_{ht}[\rho_{t+1}] - r}{V_{ht}[\rho_{t+1} - r]} \tag{8.1}$$

$$= p_t \frac{E_{ht}[p_{t+1} + y_{t+1} - Rp_t]}{V_{ht}[p_{t+1} + y_{t+1} - Rp_t]} = p_t z_t. \tag{8.2}$$

```
BELIEFS.fraction_to_invest = (BELIEFS.expected_return - RISK_FREE_RATE)/
                              BELIEFS.expected_volatility;
```

3. Add restrictions on trade:
   - Investors cannot go short: $\pi_{ht} \geq 0$.
   - Investors cannot have a negative payment account (borrow from a bank to buy shares): $\pi_{ht} \leq 1$.

This implies that the proportion to invest is restricted to lie between 0 and 1:

$$\hat{\pi}_{ht} = \min\{\max\{0, \pi_{ht}\}, 1\}. \tag{8.3}$$

The solution to the asset allocation problem is now that the investor invests $(1 - \hat{\pi}_{ht})W_{ht}$ in the risk-free asset and $\hat{\pi}_{ht}W_{ht}$ in the risky asset. Note that, in principle, any type of belief formation procedure can be substituted for the beliefs $E_{ht}[\rho_{t+1}]$ and $V_{ht}[\rho_{t+1}]$.

Step 3. Demand for index shares: Set monetary value of the planned asset portfolio (planned value of index shares after transactions).

1. Option 3a: Mean-Variance: the proportion of the portfolio budget to be invested in the index is derived by Mean-Variance theory.

```
monetary_value = BELIEFS.fraction_to_invest*portfolio_budget;
```

2. Option 3b: Random value: a random fraction of the portfolio_budget is to be invested in the index.

```
monetary_value = random_unif()*portfolio_budget;
```

3. Option 3c: The full portfolio_budget is to be invested in the index, no deposits (this is a testing option).

```
monetary_value =  portfolio_budget;
```

Step 4. Net demand for index shares: Transform the monetary value to a net investment, based on the current market value of the asset portfolio.

```
net_investment = monetary_value - ASSETSOWNED.lastprice*ASSETSOWNED.units;
planned_shares = (int) (monetary_value/ASSETSOWNED.lastprice);
net_shares     = (int) (net_investment/ASSETSOWNED.lastprice);
```

Step 5. Check if units to sell is larger than units owned; if so, sell all units in portfolio.

```
if ((net_investment<0.0) && (ASSETSOWNED.units + net_shares<0))
    net_investment = -ASSETSOWNED.lastprice*ASSETSOWNED.units;
```

Step 6. Send market order: negative value means sell, positive means buy.

```
add_order_message(ID, net_investment);
```

Household update portfolio

Step 1. Read order status message

Step 2. Update assetsowned, payment account, and wealth

```
ASSETSOWNED.units    += order_status_message->quantity;
PAYMENT_ACCOUNT      -= order_status_message->value;
WEALTH = PAYMENT_ACCOUNT + ASSETSOWNED.lastprice*ASSETSOWNED.units;
```

Step 3. Determine cash flow of the transaction and add to flow variables (for stock-flow consistency checks).

```
if(order_status_message->quantity < 0)
 HOUSEHOLD_INFLOWS_CALENDAR.asset_sales += -(order_status_message->value);
else
 HOUSEHOLD_OUTFLOWS_CALENDAR.asset_purchases += (order_status_message->value);
```

8.5 Clearinghouse

8.5.1 Activities

- The Clearinghouse determines the dividend per share of the market index, based on the information it receives from the firms and banks on the total dividend payments.
- Clearinghouse asset price mechanism. The price of a share in the market index depends on the asset trading of households and is computed by the Clearinghouse. No equilibrium pricing is used, but a cautious price adjustment process with ceilings and floors on the price growth rate (this reflects so called circuit breakers).
- Clearinghouse asset transaction mechanism. The Clearinghouse determines transactions in the market index shares. A simple proportional rationing mechanism is used. There is conservation of the total number of index units bought and sold.

8.5.2 Functions

ClearingHouse send index price

Runs daily, at start of iteration.

- Sends the index price message with the current index price and moving average price (this is computed by the ClearingHouse).

ClearingHouse send index info

Runs monthly, on first day of the calendar month.

- Start of calendar month: given monthly counter of total dividends, compute dividend per share for the stock market index.
- Send index info message with dividend per share and the current share price.

ClearingHouse receive orders

Runs daily.

- Reset the pending orders array.
- Add all incoming orders to the pending orders array.

```
reset_order_array(&PENDING_ORDERS);

START_ORDER_MESSAGE_LOOP
     quantity = (int)((order_message->value)/STOCK_INDEX.price);
     add_order(&PENDING_ORDERS, order_message->trader_id, order_message->value, quantity);
FINISH_ORDER_MESSAGE_LOOP
```

ClearingHouse compute transactions

Runs daily. Process the pending orders array, produce the processed orders array.

Step 1. Compute aggregate demand and supply (in units):

- Transform requested value to units: integer conversion may cause some orders to now have 0 quantity.
- If total demand or supply is zero: Add zero processed orders for all, and exit.

```
//Process the pending_orders array, produce processed_orders
     reset_order_array(&PROCESSED_ORDERS);

//Step 1: Compute aggregates
     demand = 0;
     supply = 0;

//Step 1a: Transform requested value to units
  for (i=0; i<PENDING_ORDERS.size; i++)
  {

     value = PENDING_ORDERS.array[i].value;

     //Transform requested value to units: set PENDING_ORDERS.array[i].quantity
     if (value>0.0)
     {
          demand += (int)(value/STOCK_INDEX.price);
          PENDING_ORDERS.array[i].quantity = (int)(value/STOCK_INDEX.price);
     }
     else
     {
          supply += -1*(int)(value/STOCK_INDEX.price);
          PENDING_ORDERS.array[i].quantity = (int)(value/STOCK_INDEX.price);
     }
  }

//Step 1b: Check for zeros and break
if (demand==0 || supply ==0)
{
     DSRATIO = 1;

     //Add zero orders for all orders
     for (i=0; i<PENDING_ORDERS.size; i++)
     {
          trader_id = PENDING_ORDERS.array[i].trader_id;
          add_order(&PROCESSED_ORDERS, trader_id, 0.0, 0);
     }
}
```

Step 2. If both demand and supply are positive, continue with rationing of the long side of the market:

- Set Demand/Supply ratio.
- Compute transacted buy/sell units.
- Add new processed order to processed orders array.

```
if (demand>0 && supply>0)
{
    //Demand/Supply ratio
    DSRATIO = (double)demand/(double)supply;

    //Demand rationing
    if (DSRATIO>1.0)
    for (i=0; i<PENDING_ORDERS.size; i++)
    {
        //Trader is buyer
        if (PENDING_ORDERS.array[i].quantity>0)
        {
        quantity = (int)((1/DSRATIO)*PENDING_ORDERS.array[i].quantity);
        sum_units_bought += quantity;
        }

        //Trader is seller
        if (PENDING_ORDERS.array[i].quantity<=0)
        {
        quantity = PENDING_ORDERS.array[i].quantity;
        sum_units_sold += -quantity;
        }

        //Add new processed order
        value = quantity*STOCK_INDEX.price;
        sum_value += value;
        sum_quantity += quantity;
        add_order(&PROCESSED_ORDERS, PENDING_ORDERS.array[i].trader_id, value, quantity);
    }
    else
    //Supply rationing
    for (i=0; i<PENDING_ORDERS.size; i++)
    {
        //Trader is buyer
        if (PENDING_ORDERS.array[i].quantity>0)
        {
        quantity = PENDING_ORDERS.array[i].quantity;
        sum_units_bought += quantity;
        }
        //Trader is seller
        if (PENDING_ORDERS.array[i].quantity<=0)
        {
        quantity = (int)(DSRATIO*PENDING_ORDERS.array[i].quantity);
        sum_units_sold += -quantity;
        }

        //Add new processed order
        value = quantity*STOCK_INDEX.price;
        sum_value += value;
        sum_quantity += quantity;
        add_order(&PROCESSED_ORDERS, PENDING_ORDERS.array[i].trader_id, value, quantity);
    }
}
```

Step 3. Post-rationing diagnosis: check conservation of units bought and sold. See the auxiliary functions for details. The algorithm monotonically converges to a state in which no further rationing is required.

```
if (sum_value > 1e-1 || sum_value < -1e-1 || sum_quantity!=0)
{
    if (sum_quantity>0)
        ClearingHouse_correct_rationing_positive(sum_quantity);
    if (sum_quantity<0)
        ClearingHouse_correct_rationing_negative(sum_quantity);
    //Check consistency:
        ClearingHouse_diagnosis();
}
```

ClearingHouse send transaction info

Runs daily.

- Send order status messages for all orders in the processed orders array.

ClearingHouse update price

Runs daily.

Step 1. Update the price of the stock market index using the latest supply/demand ratio, subject to the ceilings and floors on price changes.

The price of the market index is updated using a cautious price adjustment based on the ratio between demand and supply, limited by floors and ceilings on the growth rate of prices.

$$\Pi = \left(\frac{D_t}{S_t}\right)^{\lambda}, \tag{8.4}$$

$$p_{t+1} = \begin{cases} rp_t, & \Pi \leq r, \\ \Pi p_t, & r \leq \Pi \leq R, \\ Rp_t, & R \leq \Pi. \end{cases} \tag{8.5}$$

where $\lambda > 0$ is a price adjustment speed, r and R are lower and upper bounds on the price growth rate, respectively. Typical values are $\lambda = 0.5$, $r = 0.90$ and $R = 1.10$.

```
//Update the price of the stock market index using the latest supply/demand data.
    coeff = pow(DSRATIO, INDEX_PRICE_ADJ);

    if (coeff > INDEX_PRICE_ADJ_MAX)
        coeff = INDEX_PRICE_ADJ_MAX;
    else if (coeff < INDEX_PRICE_ADJ_MIN)
        coeff = INDEX_PRICE_ADJ_MIN;

    STOCK_INDEX.price = coeff*STOCK_INDEX.price;
```

Step 2. Update the price history and compute the new moving average price (being sent to the households in the index info message at the start of the next iteration).

```
//Shift the price history and determine the moving average:

    //remove the oldest price at the front of the list, reduce array size by 1
    remove_double(&STOCK_INDEX.price_history, 0);

    //add new price at the end of the list, increase array size by 1
    add_double(&STOCK_INDEX.price_history, STOCK_INDEX.price);

    sum=0.0;
    for (i=0;i<STOCK_INDEX.price_history.size;i++)
        sum += STOCK_INDEX.price_history.array[i];

    if(STOCK_INDEX.price_history.size>0)
        STOCK_INDEX.moving_avg_price = sum/STOCK_INDEX.price_history.size;
```

ClearingHouse receive dividend info

Runs daily, at end of iteration.

- Every day (end of iteration): Read dividend info messages from firms during the month and add to a monthly counter of total dividends for the stock market index.

8.5.3 ClearingHouse auxiliary functions

ClearingHouse correct rationing positive(int total rationing)

- Goal: Correct the overshooting of the first rationing round.
- The positive side of the market needs to be rationed by a given amount of units R =total_rationing.

Step 1. Traverse the PROCESSED_ORDERS array, and ration subsequent long orders by a random amount.

Step 2. Draw a uniformly random integer $x \in [1, R]$.

Step 3. Ration the buy order to $B_h := B_h - x$ units (or to 0 if $B_h < x$) and decrease total_rationing to $R := R - x$.

Step 4. Continue as long as there is rationing left-over or the end of PROCESSED_ORDERS is reached.

```
void ClearingHouse_correct_rationing_positive(int total_rationing)
{
    int i;
    int x, rationed_units;

  i=0;
  while ((total_rationing>0)&&(i<PROCESSED_ORDERS.size))
  {
    //Only consider positive orders:
    if(PROCESSED_ORDERS.array[i].quantity>0)
    {
        //Ration this order by random x units, if the order has at least x units
        x=random_int(1,total_rationing);

        if (x<=PROCESSED_ORDERS.array[i].quantity)
        {
            // ation this order by random x
            PROCESSED_ORDERS.array[i].quantity -= x;
            rationed_units = x;
            if (PRINT_DEBUG_AFM_CH) printf("case_1\t");
        }
        else
        {
            //Fully ration this order to 0:
            rationed_units = PROCESSED_ORDERS.array[i].quantity;
            PROCESSED_ORDERS.array[i].quantity =0;
        }
        total_rationing -= rationed_units;
    }//order-loop
    i++;
  }//while-loop
```

ClearingHouse correct rationing negative(int total rationing)

- Goal: Correct the overshooting of the first rationing round.
- The negative side of the market needs to be rationed by a given amount of units R =total_rationing.

Step 1. Traverse the PROCESSED_ORDERS array, and ration subsequent long orders by a random amount.

Step 2. Draw a uniformly random integer $x \in [1, R]$.

Step 3. Ration the sell order to $S_h := S_h + x$ units (or to 0 if $x \leq S_h < 0$) and decrease total_rationing to $R := R - x$.

Step 4. Continue as long as there is rationing left-over or the end of PROCESSED_ORDERS is reached.

```
void ClearingHouse_correct_rationing_negative(int total_rationing)
{
    int i;
    int x, rationed_units;

  i=0;
  while ((total_rationing<0)&&(i<PROCESSED_ORDERS.size))
  {
    //Only consider negative orders:
    if(PROCESSED_ORDERS.array[i].quantity<0)
    {
        //Ration this order by random x units, if the order has at least x units
        x=random_int(1,-total_rationing);

        if (x<= -PROCESSED_ORDERS.array[i].quantity)
        {
            // ation this order by random x: a negative order is increased
            PROCESSED_ORDERS.array[i].quantity += x;
            rationed_units = x;
        }
        else
        {
            //Fully ration this order to 0:
            rationed_units = -PROCESSED_ORDERS.array[i].quantity;
            PROCESSED_ORDERS.array[i].quantity =0;
        }

        //increase to-ration units towards 0:
        total_rationing += rationed_units;
    }//order-loop
    i++;
  }//while-loop
```

8.6 Stategraph

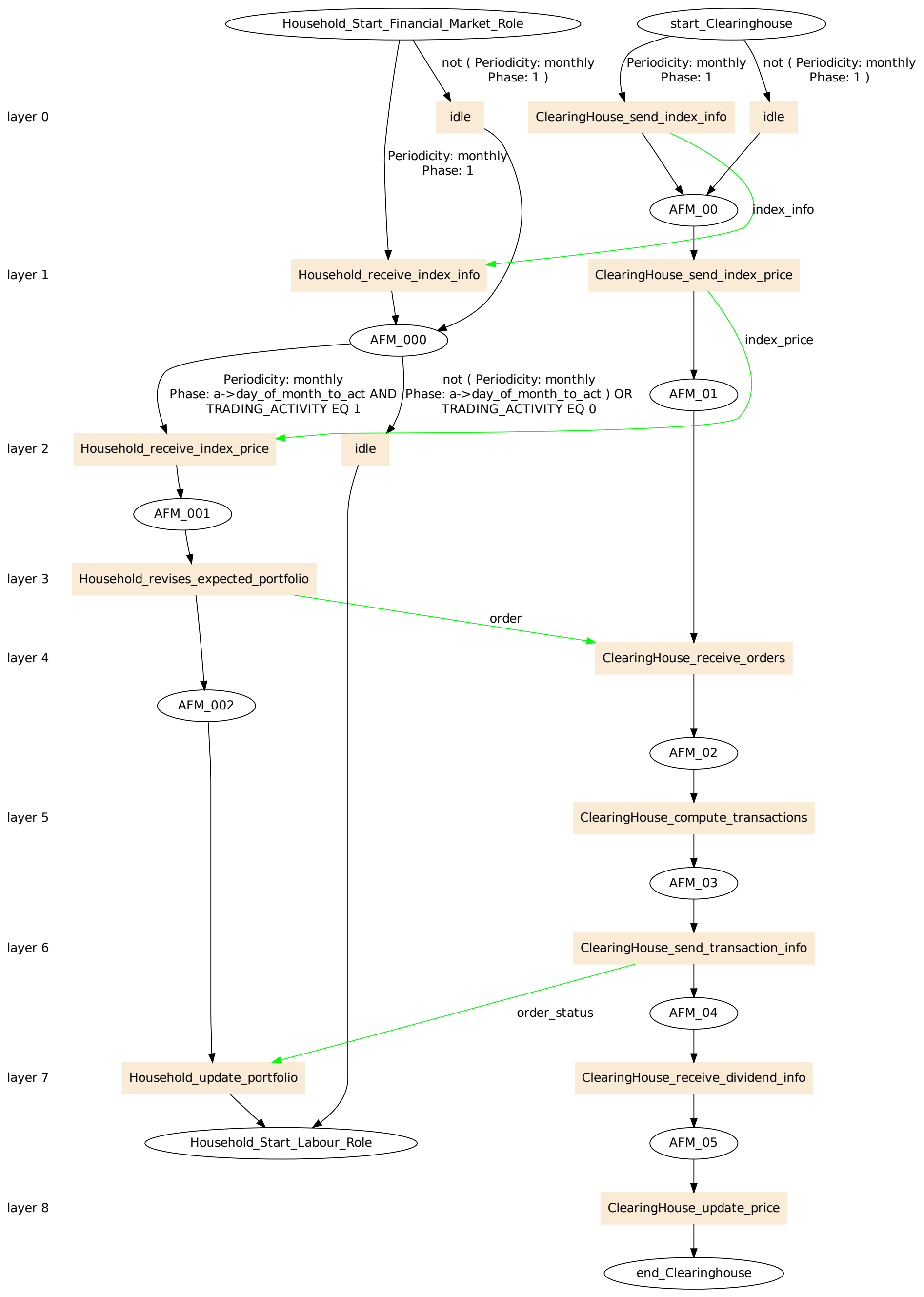

Household_Start_Financial_Market_Role
start_Clearinghouse
not (Periodicity: monthly Phase: 1)
Periodicity: monthly Phase: 1
not (Periodicity: monthly Phase: 1)
layer 0
idle
ClearingHouse_send_index_info
idle
Periodicity: monthly Phase: 1
AFM_00
index_info
layer 1
Household_receive_index_info
ClearingHouse_send_index_price
AFM_000
index_price
Periodicity: monthly Phase: a->day_of_month_to_act AND TRADING_ACTIVITY EQ 1
not (Periodicity: monthly Phase: a->day_of_month_to_act) OR TRADING_ACTIVITY EQ 0
AFM_01
layer 2
Household_receive_index_price
idle
AFM_001
layer 3
Household_revises_expected_portfolio
order
layer 4
ClearingHouse_receive_orders
AFM_002
AFM_02
layer 5
ClearingHouse_compute_transactions
AFM_03
layer 6
ClearingHouse_send_transaction_info
order_status
AFM_04
layer 7
Household_update_portfolio
ClearingHouse_receive_dividend_info
Household_Start_Labour_Role
AFM_05
layer 8
ClearingHouse_update_price
end_Clearinghouse

Figure 8.1: Stategraph for the financial market.

Chapter 9

Government Documentation

9.1 Messages

- Data for Government message: start of the year.

  At the start of the year Eurostat sends multiple data_for_Government messages, one for each region, that contains the GDP and other macrodata for that region. The Governments read these messages only for those regions that belong to them, and compute their national GDP. The national GDP is then used by the Governments to compute their budget forecast and to set policies. The message contains region data:

  ```
  region_id
  gdp
  mean_wage
  ```

- Eurostat send macro data message: start of the year. At the start of the year Eurostat sends one message that contains global economic data: economy-wide GDP, inflation and unemployment rate. All Governments can read this message.

- Policy announcement message: start of the year. At the start of the year, the Government announces new tax rates and new percentages for subsidies and transfers. Agents read these messages at the start of the year.

- Unemployment notification messages: daily. Send as soon as a household becomes unemployed. Government needs to read these messages daily.

- Subsidy and transfer notification messages: daily. At the start of the agent's own month, each firm and household determines whether it wants to make use of a certain announced subsidy or transfer. If an agent makes use of a policy, it sends the notification message on the 1st day of its month. Government needs to read these messages daily, since agents have different activation days.

9.2 Eurostat

9.2.1 Activities

9.2.2 Functions

Eurostat send data

At the start of the year Eurostat sends multiple data_for_Government messages (one for each region) that contains the GDP and other macrodata for each region. The Governments read these messages only for those regions that are associated to them, and compute their national GDP. The national GDP is then used by the Governments to compute their budget forecast and to set policies.

9.3 Firm, Household, Bank

9.3.1 Activities

Households:

- Households pay personal income tax on wage and dividend income.
- Households decide whether or not to apply for a subsidy or transfer payments, and unemployment benefits when unemployed.

Firms:

- Firms pay corporation tax on income.
- Firms decide whether or not to apply for a subsidy or transfer payments.

Banks:

- Banks pay corporation tax on profits.

9.3.2 Functions

Read policy announcements

At the start of the year all agents read the Government policy announcements (only from their own Government), and store that information in memory variables for later use.

Send subsidy/transfer/unemployment notification

When an agent applies for any payment (benefits, subsidies, or transfers) it sends a notification message to its Government that contains only the ID of the Government and the payment amount (tax, unemployment benefit, or subsidy payment). These notification messages have a daily frequency. From the name of the notification message it becomes clear what type of payment is referred to, and since the amount is known by both sides the government can now compute sums for the separate payment items. The list of notification messages is (can be expand as needed):

```
add_unemployment_notification_message(gov_id, last_income)
add_hh_subsidy_notification_message(gov_id, subsidy_payment)
add_firm_subsidy_notification_message(gov_id, subsidy_payment)
add_hh_transfer_notification_message(gov_id, transfer_payment)
add_firm_transfer_notification_message(gov_id, transfer_payment)
```

Send tax payment

All agents in the private sector pay taxes to Government. However, these payments are not synchronized at the end of a calendar month, since the firms have different days of the month to start production.

- Firms: Tax payment occurs at the end of the production cycle, which includes the selling of output i.e. after one month has passed since production has commenced.
- Households: Tax payment occurs at the moment households receive their income, which consists of labour wage, dividends, and interest on bank deposits.
- Banks: Tax payment occurs at the end of the calander month.

```
add_tax_payment_message(gov_id, tax_payment)
```

9.4 Government

9.4.1 Activities

- Government mainly has redistributive functions, such as taxation (income and corporate taxes, social security contributions) and unemployment benefit payments.
- The provisional Government budget is given by the total sum of subsidies and unemployment benefits.
- The Government uses a naive expectation at the start of the year: the projected Government income is equal to last year's income times 1 + an anticipated growth rate; the projected Government expenditures are equal to last year's expenditures times 1 + an anticipated growth rate.
- Any surplus is deposited at the central bank. Any deficit is covered by loans from the central bank.
- the Government also distributes subsidies. Subsidies are conditional, they must be used for the specific purpose under which they are granted. E.g., a firm can receive a subsidy to pay for internal training schemes that will raise the general skills of (part of) its employees.
- The subsidies are non-discriminatory: they are available to all firms or households. This does not preclude the possibility of regional policies: some subsidies are available in one region, not in the other.
- Government performs budget accounting: end of the month.

- Bond market opens: daily. The Government tries to finance its budget deficit of the previous month during the following month, in which the bond market is open each day, and the Government can spread out its monthly liquidity needs across 20 days. If after 20 days the Government has not managed to attain all of its liquidity requirements the remainder is carried over to the following month.

- Government policy making: Government decides on new policies together with the budgeting at the end of the year (i.e., new tax rates, unemployment benefit percentage, subsidy percentage, transfer lump-sum payment). The announcement of the new policies occurs at the start of the new year.

- Government policies remain unaltered during the year (total government consumption and investments, tax rates, subsidies and transfers all remain fixed) and may only be changed at the end of the year.

- Government decides monetary policy: end of the year. After computing the budget deficit, the Government tries to finance it, either through bond financing or by money financing (quantitative easing). In the case of bond financing, the Government first issues bonds on the bond market and tries to sell to households. If this fails, the remaining bonds can be sold to the Central Bank. This involves the issue_bonds_to_ecb message. In the case of direct money financing, the Government instructs the Central Bank to print fiat money. This involves the request_fiat_money message.

- The Government computes its budget deficit once per month, but enters the bond market on a daily basis. This is due to liquidity reasons since it will be probable that if the Government enters the bond market only once per month there is insufficient demand for the bonds, and the Government may fail to attain its liquidity target. So basically the monthly budget deficit will be financed by bonds on a monthly basis, but there is a smoothing across the month.

- Consequently, the Government runs the budget accounting function each month, instead of each year, to determine the monthly budget deficit.

- By default, the budget deficits are 100% financed by bonds. This due to the fact that in the EU context, the national Central Banks have lost the authority to control the money stock, so budget deficits can no longer be financed by money creation. However, the Government does have a limited standing facility with the Central Bank on which it can draw freely during the month. This can effectively be considered as temporary money creation. The Government repays the borrowed amount to the Central Bank as soon as it has financed all its monthly expenditures by issuing bonds. If due to rationing on the bond market the Government does not manage to finance all its liquidity needs the remainder is carried over to the next month. In the meantime, the Government has a standing facility at the Central Bank that functions as a buffer account to finance ongoing payments.

9.4.2 Functions

Government read data from Eurostat

At the start of the year the Government(s) read the regional and economy-wide macroeconomic data from Eurostat, to determine the policy for the upcoming year.

Step 1. The government reads all regional data for those regions that are in its list of regions.

```
GDP=0.0;
COUNTRY_WIDE_MEAN_WAGE=0.0;

START_DATA_FOR_GOVERNMENT_MESSAGE_LOOP
    for (i=0; i<NO_REGIONS_PER_GOV; i++)
    {
        if(data_for_government_message->region_id==LIST_OF_REGIONS.array[i])
        {
            //Read region mean wage
            COUNTRY_WIDE_MEAN_WAGE += data_for_government_message->mean_wage;

            //Read region GDP
            GDP += data_for_government_message->gdp;
        }
    }
FINISH_DATA_FOR_GOVERNMENT_MESSAGE_LOOP

//Set country-wide mean wage as avg of region's mean wages
COUNTRY_WIDE_MEAN_WAGE = COUNTRY_WIDE_MEAN_WAGE/NO_REGIONS_PER_GOV;

//Set GDP growth rate
if (old_gdp > 0.0)
    GDP_GROWTH = GDP/old_gdp;
else GDP_GROWTH = 1.0;
```

Step 2. The government reads the global economic data to retrieve the economy-wide inflation and unemployment rates:

```
START_EUROSTAT_SEND_MACRODATA_MESSAGE_LOOP
    INFLATION_RATE = eurostat_send_macrodata_message->inflation;
    UNEMPLOYMENT_RATE = eurostat_send_macrodata_message->unemployment_rate;
FINISH_EUROSTAT_SEND_MACRODATA_MESSAGE_LOOP
```

Government set policy

The Government sets new policies based on the previous year's budget deficit, national GDP, and its forecasted next year's GDP.

Step 1. Set a GDP forecast equal to an extrapolation of the previous year's GDP growth rate. Set an income forecast assuming that next year's income will grow at the same rate as last year's GDP, and set an expenditure budget allowing yearly expenditure to grow with the GDP growth rate. Finally, determine the budget balance forecast.

```
GDP_FORECAST = GDP_GROWTH*GDP;
YEARLY_INCOME_FORECAST = GDP_GROWTH*YEARLY_INCOME;
YEARLY_EXPENDITURE_BUDGET = GDP_GROWTH*YEARLY_EXPENDITURE;
BUDGET_BALANCE_FORECAST = YEARLY_INCOME_FORECAST - YEARLY_EXPENDITURE_BUDGET;
```

Step 2. Determine government consumption and investment, as fractions of GDP.

```
YEARLY_CONSUMPTION_BUDGET = GOV_POLICY_GDP_FRACTION_CONSUMPTION * GDP_FORECAST;
MONTHLY_CONSUMPTION_BUDGET = YEARLY_CONSUMPTION_BUDGET/12;

YEARLY_INVESTMENT_BUDGET = GOV_POLICY_GDP_FRACTION_INVESTMENT * GDP_FORECAST;
MONTHLY_INVESTMENT_BUDGET = YEARLY_INVESTMENT_BUDGET/12;
```

Government send policy announcements

At the start of the year the Government announces new policies by sending a general policy_announcement_message. The message contains the id of the Government, information for each type of policy (benefit, transfer, subsidy), and whether the payment is a percentage or a lump-sum payment. The variables in the announcement message are (can be expanded as needed):

```
gov_id
tax_rate_corporate
tax_rate_hh_labour
tax_rate_hh_capital
tax_rate_vat
unemployment_benefit_pct
hh_subsidy_pct
firm_subsidy_pct
hh_transfer_payment
firm_transfer_payment
```

The global environment constant gov_policy_unemployment_benefit_pct can be fixed, but the Government can also use the variable unemployment_benefit_pct to announce an endogenous policy that deviates from the global policy parameter.

Government read subsidy notification

Computation of the monthly and yearly totals of subsidy payments to households and firms. The Government computes total payments by looping over these notification messages each day and also computes the monthly and yearly sums:

```
tax_payment_message(gov_id, tax_payment)
unemployment_notification_message(gov_id, last_income)
hh_subsidy_notification_message(gov_id, subsidy_payment)
firm_subsidy_notification_message(gov_id, subsidy_payment)
hh_transfer_notification_message(gov_id, transfer_payment)
firm_transfer_notification_message(gov_id, transfer_payment)
```

Step 1. Sum over all household subsidy payments (as reported by households in the message) and compute the total subsidy payments to households.

```
sum=0;
START_HH_SUBSIDY_NOTIFICATION_MESSAGE_LOOP
    if(hh_subsidy_payment_notification_message->gov_id==ID)
        sum+=hh_subsidy_payment_notification_message->subsidy_payment;
FINISH_HH_SUBSIDY_NOTIFICATION_MESSAGE_LOOP

MONTHLY_SUBSIDY_PAYMENT += sum;
YEARLY_SUBSIDY_PAYMENT += sum;
PAYMENT_ACCOUNT -= sum;
```

Step 2. Sum over all firm subsidy payments and compute the total subsidy payment to firms.

```
sum=0;
START_FIRM_SUBSIDY_NOTIFICATION_MESSAGE_LOOP
    if(firm_subsidy_payment_notification_message->gov_id==ID)
        sum+=firm_subsidy_payment_notification_message->subsidy_payment;
FINISH_FIRM_SUBSIDY_NOTIFICATION_MESSAGE_LOOP
```

```
MONTHLY_SUBSIDY_PAYMENT += sum;
YEARLY_SUBSIDY_PAYMENT += sum;
PAYMENT_ACCOUNT -= sum;
```

Government read transfer notification

Computation of the monthly and yearly totals of the transfer payments.

Step 1. Sum over all household transfer notifications and compute the total transfer payments to households.

```
sum=0;
START_HH_TRANSFER_NOTIFICATION_MESSAGE_LOOP
     if(hh_transfer_notification_message->gov_id==ID)
          sum++;
FINISH_HH_TRANSFER_NOTIFICATION_MESSAGE_LOOP

MONTHLY_TRANSFER_PAYMENT += sum*HH_TRANSFER_PAYMENT;
YEARLY_TRANSFER_PAYMENT += sum*HH_TRANSFER_PAYMENT;
PAYMENT_ACCOUNT -= sum*HH_TRANSFER_PAYMENT;
```

Step 2. Sum over all firm transfer notifications and compute the total transfer payments to firms.

```
sum=0;
START_FIRM_TRANSFER_NOTIFICATION_MESSAGE_LOOP
     if(firm_transfer_notification_message->gov_id==ID)
          sum++;
FINISH_FIRM_TRANSFER_NOTIFICATION_MESSAGE_LOOP

MONTHLY_TRANSFER_PAYMENT += sum*FIRM_TRANSFER_PAYMENT;
YEARLY_TRANSFER_PAYMENT += sum*FIRM_TRANSFER_PAYMENT;
PAYMENT_ACCOUNT -= sum*FIRM_TRANSFER_PAYMENT;
```

Government read unemployment benefit notifications

(Related: Household send unemployment benefit notifications)
Function to compute monthly and yearly totals of the unemployment benefit payments. Monthly counter of the unemployment messages.

- When households become unemployed *during* the month, they send an unemployment_notification message to their Government at the beginning of their own private month, on the day on which they normally would have received their next wage. The message contains their last received wage. Their Government reads this message, computes the cost of the entitlement and also computes the total monthly payments for unemployment benefits. The total value of all unemployment benefits received that day is subtracted from the Government's payment account, and added to the total monthly and yearly sums.
- The Government does not need to send an unemployment benefit payment message to each unemployed household individually. Instead, the households receive the information at the start of the year containing the net replacement rate (the percentage of their last earned wage). The value of the unemployment benefit can then be computed by the households themselves without any need for a message interaction.

- From then onwards, the household receives the benefit on the same day as it normally would have received its wage from the firm, by sending the unemployment notification message on the same every month.
- A problem occurs when a household gets re-employed by a different firm: the new firm's day of the month to pay wages may differ from the household's day it receives unemployment benefits. This can be resolved by letting the household repay the excess in benefits as a restitution payment to the Government, given by:

  ```
  restitution_payment = (1/20)*((day_of_month_receive_benefit + (20 - day_of_month_receive_n
  * unemployment_benefit
  ```

  Example 1: A household is fired on day 5 (the last day the firm paid the wage was on day 4). It receives its monthly unemployment benefit on day 5 each month. It is re-employed on day 14 and so it starts earning a wage on day 14. It already received a month worth of benefits. It now has to repay that part starting from day 14 up until day 4 the next month. That is: $5 + (20 - 14) = 11$ out of 20 days. The household repays the Government (11/20) of the received benefit in additional taxes at the end of the month.
  Example 2: The same household is re-employed on day 4. Then it needs to repay (1/20): $5 + (20 - 4) = 21$, but we take $21(mod 20) = 1$.
  Example 3: The same household is re-employed on day 3. Then it needs to repay (2/20): $5 + (20 - 3) = 22$, but we take $22(mod 20) = 2$.
  So the correct code is:

  ```
  restitution_payment = (1/20)*((day_of_month_receive_benefit + (20 - day_of_month_receive_n
  * unemployment_benefit
  ```

- If the household remains unemployed, it should resend the unemployment notification message on the same day of the month. In effect, we force the unemployed households to re-apply for unemployment benefits every month.

Step 1. Compute the individual unemployment benefit payment as a fraction of the last labour income (the net replacement rate). If the computed unemployment benefit is greater than the mean wage, this amount is used as the benefit payment. Otherwise, the benefit payment is set equal to half the mean wage.

```
NUM_UNEMPLOYED = 0;
MONTHLY_UNEMPLOYMENT_BENEFIT_PAYMENT=0.0;

sum=0.0;
START_UNEMPLOYMENT_NOTIFICATION_MESSAGE_LOOP
    NUM_UNEMPLOYED++;

    if(unemployment_notification_message->last_labour_income*UNEMPLOYMENT_BENEFIT_PCT
        > COUNTRY_WIDE_MEAN_WAGE*0.5 )
    {
        unemployment_payment = unemployment_notification_message->last_labour_income*U
    }
    else
    {
        unemployment_payment =  COUNTRY_WIDE_MEAN_WAGE*0.5;
    }
    sum += unemployment_payment;
FINISH_UNEMPLOYMENT_NOTIFICATION_MESSAGE_LOOP
```

Step 2. The total benefit payments are computed and subtracted from the payment account.

```
MONTHLY_UNEMPLOYMENT_BENEFIT_PAYMENT += sum;
YEARLY_UNEMPLOYMENT_BENEFIT_PAYMENT += sum;
PAYMENT_ACCOUNT -= sum;
```

Government read tax payments

At the end of the subjective month all agents send their tax_payment message, read by the Government daily and added to the Government's payment account. The monthly tax revenues are added to the yearly tax revenues. At the end of the month, the monthly tax revenues are reset to zero to restart the count for the next month.

```
//Reset the monthly tax counter
MONTHLY_TAX_REVENUES =0.0;

START_TAX_PAYMENT_MESSAGE_LOOP
    MONTHLY_TAX_REVENUES += tax_payment_message->tax_payment;
FINISH_TAX_PAYMENT_MESSAGE_LOOP

PAYMENT_ACCOUNT += MONTHLY_TAX_REVENUES;
YEARLY_TAX_REVENUES += MONTHLY_TAX_REVENUES;
```

Government monthly budget accounting

At the end of the year the Government does some essential budget accounting: it computes the total unemployment benefit payments, total transfer payments, total bond coupon payments, total interest payment on the public debt, total Government investment and consumption. Then it computes the budget deficit and executes the monetary policy rule to determine the money financing and bond financing amounts. Next, it executes the fiscal policy rule to adjust the tax rates for the following year. The fiscal policy rule is currently very simple: if the payment account of the Government was negative, increase taxes a bit, if it was positive, decrease taxes a bit.

Step 1. Compute the monthly income, expenditures, and budget balance.

```
YEARLY_TAX_REVENUES += MONTHLY_TAX_REVENUES;

    in = MONTHLY_TAX_REVENUES;
    MONTHLY_INCOME = in;

    out = MONTHLY_UNEMPLOYMENT_BENEFIT_PAYMENT +
    MONTHLY_TRANSFER_PAYMENT +
    MONTHLY_BOND_COUPON_PAYMENT +
    MONTHLY_INVESTMENT_EXPENDITURE +
    MONTHLY_CONSUMPTION_EXPENDITURE;

    MONTHLY_EXPENDITURE = out;

    MONTHLY_BUDGET_BALANCE = in - out;
```

Step 2. Subtract the monthly budget balance from the cumulated deficit. In case the balance is a surplus the cumulative deficit decreases, otherwise it increases. The total debt is computed as the face value of all outstanding government bonds.

```
CUMULATED_DEFICIT -= MONTHLY_BUDGET_BALANCE;
TOTAL_DEBT = BOND.nr_outstanding*BOND.face_value;
```

Step 3. Set a monetary policy rule to finance the current deficit.

```
TOTAL_MONEY_FINANCING=0;
TOTAL_BOND_FINANCING=0;
if (PAYMENT_ACCOUNT<0)
{
    TOTAL_MONEY_FINANCING = GOV_POLICY_MONEY_FINANCING_FRACTION
                            * abs(PAYMENT_ACCOUNT);
    TOTAL_BOND_FINANCING = (1-GOV_POLICY_MONEY_FINANCING_FRACTION)
                            * abs(PAYMENT_ACCOUNT);
}
```

Step 4. The part of the deficit that is directly financed by fiat money requires a message to the Central Bank.

```
add_request_fiat_money_message(TOTAL_MONEY_FINANCING);
PAYMENT_ACCOUNT += TOTAL_MONEY_FINANCING;
```

Government send account update

Government sends the payment account value to the Central Bank.

Government compute balance sheet

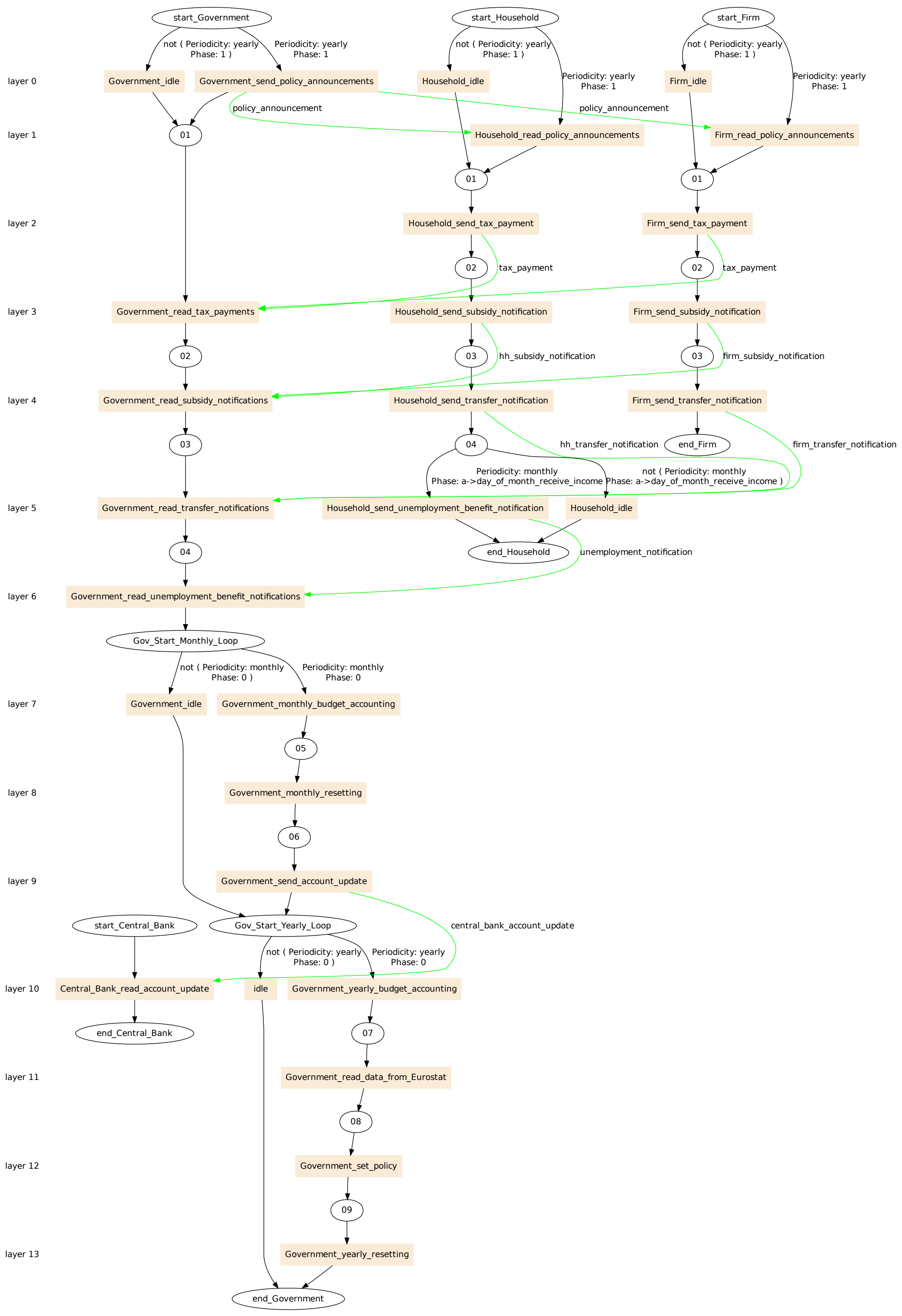
Monthly computation of the Government balance sheet.

Government yearly budget accounting

At the end of the year the Government computes the yearly income, expenditures, and budget balance.

```
in = YEARLY_TAX_REVENUES;
YEARLY_INCOME = in;

out = YEARLY_UNEMPLOYMENT_BENEFIT_PAYMENT +
YEARLY_TRANSFER_PAYMENT +
YEARLY_BOND_COUPON_PAYMENT +
YEARLY_INVESTMENT_EXPENDITURE +
YEARLY_CONSUMPTION_EXPENDITURE;
YEARLY_EXPENDITURE = out;

YEARLY_BUDGET_BALANCE = in - out;
```

9.5 Stategraph


start_Government
start_Household
start_Firm
not (Periodicity: yearly Phase: 1)
Periodicity: yearly Phase: 1
layer 0
Government_idle
Government_send_policy_announcements
Household_idle
Firm_idle
policy_announcement
layer 1
01
Household_read_policy_announcements
Firm_read_policy_announcements
layer 2
Household_send_tax_payment
Firm_send_tax_payment
02
tax_payment
layer 3
Government_read_tax_payments
Household_send_subsidy_notification
Firm_send_subsidy_notification
03
hh_subsidy_notification
firm_subsidy_notification
layer 4
Government_read_subsidy_notifications
Household_send_transfer_notification
Firm_send_transfer_notification
04
hh_transfer_notification
end_Firm
firm_transfer_notification
Periodicity: monthly Phase: a->day_of_month_receive_income
not (Periodicity: monthly Phase: a->day_of_month_receive_income)
layer 5
Government_read_transfer_notifications
Household_send_unemployment_benefit_notification
Household_idle
end_Household
unemployment_notification
layer 6
Government_read_unemployment_benefit_notifications
Gov_Start_Monthly_Loop
not (Periodicity: monthly Phase: 0)
Periodicity: monthly Phase: 0
layer 7
Government_idle
Government_monthly_budget_accounting
05
layer 8
Government_monthly_resetting
06
layer 9
Government_send_account_update
start_Central_Bank
Gov_Start_Yearly_Loop
central_bank_account_update
not (Periodicity: yearly Phase: 0)
Periodicity: yearly Phase: 0
layer 10
Central_Bank_read_account_update
idle
Government_yearly_budget_accounting
end_Central_Bank
07
layer 11
Government_read_data_from_Eurostat
08
layer 12
Government_set_policy
09
layer 13
Government_yearly_resetting
end_Government

Chapter 10

Model initialization

10.1 Population initialization

In general there are several considerations that constrain the initialization of the agent's state variables. First and foremost, we cannot initialize the variables completely at random. This would violate the internal logic of the model, since in order to obtain a working simulation we have to initialize the agent's balance sheets according to the criterion of stock-flow consistency. This means that we are constrained to set the initial values such that the balance sheet relationships between agents hold. If the balance sheets would be inconsistent from the start they would remain so throughout the entire simulation.

The second consideration is that we start with reasonable or plausible values. This is in order to alleviate the initial transient effects that any initialization invariably has. In our experience large path dependencies can be generated by such initial transients, so it is important to carefully consider the interdependencies between the initial values.

10.1.1 IGFirm values

Step 0. IGFirm constant initialization values

```
IGFirm.wage_offer            1.0
IGFirm.capital_good_store    100
IGFirm.productivity          1.7
IGFirm.payment_account       0.0
```

Step 1. IGFirm capital goods price and unit costs

```
IGFirm.capital price         initial_capital_price_wage_ratio * IGFirm.wage_offer
IGFirm.unit_costs            initial_capital_price_wage_ratio  * IGFirm.wage_offer
```

Argumentation:

- *capital price*: The initial capital price is assumed to be in a fixed relation to the initial wage that is on average paid in the economy. Even if the IG firm does not employ workers it has a memory variable *wage offer* that is used only for the initialization.

- *unit costs*: The price setting of the IG firm is a combination of value and cost based pricing. For the cost based price component the IG firm takes virtual unit costs into account, i.e. a variable called *unit costs* that is a proxy for the costs which would arise in the production process. Since the costs usually change over time (mainly due to increasing labor costs) this change has to be incorporated in the evolution of the unit costs. In order to have a stable capital goods price in the first months, the unit costs have to be initialized at the same level as the initial capital goods price.

10.1.2 Firm values

Step 0. Firm constant initialization values

```
Firm.mean_specific_skills         1.5
Firm.wage_offer                   1.0
Firm.technology                   1.5
```

Step 1. Firm output and units of capital stock needed

```
Firm.output                       (Nr.Household/Nr.Firm)
                                  * min(Firm.technology, Firm.mean_specific_skills)
Firm.total_units_capital_stock    Firm.output/min(technology, mean_specific_skills)
```

Argumentation:

- *output*: We set the output of a firm at a level such that the total labor demand that is needed for producing the cumulated output would correspond to full employment.
- *total units capital stock*: The capital stock is set to have a sufficient capital stock in order that the initial production quantity can be produced without additional capital investments.

Step 2. Set firm balance sheet: asset side.

```
Firm.actual_cap_price             IGFirm.capital price
Firm.total_value_capital_stock    Firm.actual_cap_price*total_units_capital_stock
Firm.payment_account              1.0*total_value_capital_stock
Firm_total_value_local_inventory  0.0
Firm.total_assets                 payment_account + total_value_capital_stock
                                                  + total_value_local_inventory
```

Argumentation:

- *total value capital stock*: This is an asset on the balance sheet of the firm.
- *payment account*: The value of the firm's payment account is set to equal the value of its capital stock, such that the firm has sufficient liquidity in the first month to start repaying the initial loan that was inherited from historical investments.
- *total value local inventory*: The firm has no initial inventory stock.

Step 3. Set firm balance sheet: liability side.

```
const_firm_leverage               2.0
Firm.initial_loan                 (const_firm_leverage/(1+const_firm_leverage))
                                      * Firm.total_assets
Firm.total_debt                   Firm.initial_loan
Firm.equity                       Firm.total_assets - Firm.total_debt
```

Argumentation:

- *initial loan*: We start the firms with an initial loand in order to approximate plausible leverage ratios. This will alleviate initial transient effects. The initial loan is set according to a constant leverage ratio of 2.0. This implies that the initial loan is (2/3) of total assets and equity is (1/3) of total assets.

Step 4. Capital financing of the existing capital stock.

```
installment_periods           24
depreciation_rate             0.01
Firm.capital_financing[24]    total_units_capital_stock * depreciation_rate
                              * IGFirm.capital_good_price/(2*installment_periods)
```

Argumentation:

- *capital financing per month*: We assume that the firm has invested in capital during its history before day 0 (the start of the simulation). The investments are exactly that amount which is necessary to compensate for the monthly depreciation of capital such that the capital stock remains constant. In order to stabilize the simulation with regard to bankruptcies at the beginning we deviate from the usual assumption that the loan obtained for the investments has to be repaid in the standard repayment period of a loan. Instead we allow the initial loan to be repaid in twice the length of time (24 months).

10.1.3 Household values

Step 0. Household constant initialization values

```
wealth_income_ratio_target        16.67
target_savings_rate               0.10
carrol_consumption_parameter      0.01
Household.employee_firm_id        −1 (unemployed)
```

Argumentation:

- *wealth income ratio target*: Household consumption and savings behavior is driven by a target wealth income ratio.
- *target savings rate*: We assume a target savings rate of 10%.
- *carrol consumption parameter*: The constant from buffer stock saving theory. If total wealth (liquid and illiquid, i.e. payment account plus asset wealth) exceeds the target wealth level, the households consumes an extra 1% of the excess and saves the rest.
- *employee firm id*: We assume all households are initially unemployed.

Step 1. Set Household wage and income

```
Household.wage_reservation        min(Firm.technology, specific_skill)*1.0
Household.mean_net_income         Household.wage_reservation
```

Argumentation:

- *wage reservation*: The reservation wage is set equal to the firms wage offer, such that households accept job offers in the first month.
- *mean net income*: Mean net income is set equal to the reservation wage.

Step 2. Set Household balance sheet: asset side.

```
Household.payment_account         Household.mean_net_income
                                  * target_savings_rate/carrol_consumption_parameter
Household.assetsowned_units       Clearinghouse.nr_shares/Households
Household.assetsowned_price       10/units
Household.wealth                  payment_account + assetsowned_units*assetsowned_price
```

Argumentation:

- *payment account*: Households have an initial payment account equal to 15 monthly wages, to represent a plausible savings buffer.
- *assetsowned*: The households asset portfolio is scaled to yield a wealth level that is reasonable. Each household is endowed with an equal number of index shares, with a price such that the total value of the initial portfolio is 10 (each household has risky-asset wealth equal to 10 monthly mean wages).
- *wealth*: Households' initial total wealth consists of liquid money holdings in the payment account and illiquid asset holdings. Together with the payment account of 15 monthly wages the initial total wealth of each household is 25.

10.1.4 Clearinghouse values

Step 1. Clearinghouse initialization of stock market index

```
Clearinghouse.index
    nr_shares          (F+IG)*H
    price              10*H/nr_shares
    weight             1/(F+IG)
```

Argumentation:

- *number of shares*: The number of index shares is scaled to the total number of firms and households.
- *price*: The price of index shares is chosen such that each household initially has an asset portfolio with a value of 10.
- *weight*: The weight of the firms in the index share is uniform. This is needed to compute the dividend per index share from the total dividend payment of the firms.

10.2 Realization of initial values

```
installment_periods                12
initial_capital_price_wage_ratio   30.0
```

```
IGFirm.capital price               30
IGFirm.wage_offer                  1.0
IGFirm.capital_good_store          100
IGFirm.productivity                1.7
IGFirm.payment_account             0.0
IGFirm.unit_costs                  30
```

```
Firm.output                        30
Firm.total_units_capital_stock     20
Firm.total_value_capital_stock     600
Firm.payment_account               600
Firm.total_assets                  1200
Firm.initial_loan                  800
Firm.debt_installments[24]         33.3
Firm.equity                        400
Firm.capital_financing[24]         0.25
```

```
Household.mean_net_income          1.5
Household.payment_account          15
Household.wealth                   25
Household.wealth_income_ratio_actual  16.67
Household.consumption_budget       1.5
Household.assetsowned
             price                 0.12345679
             units                 81
```

```
Clearinghouse.index
    nrFirm                         80
    nrIGFirm                       1
    nrHousehold                    1600
    nr_shares                      129600
    price                          0.12345679
```

Chapter 11

Model validation

11.1 Validation rules

To validate the internal consistency of the model we list 33 rules that we have successfully tested. On the one hand these rules are balance sheet accounting identities, and on the other they are conservation rules for material quantities and monetary values. Having thus validated the model we are confident that the EURACE model is stock-flow consistent, and can form a solid basis for further extensions in the future.

Examples of validation rules

- Balance sheet accounting identities can be devised across agents and used to validate the model.

- Examples:

  - Total amount of loans on the balance sheets of the banks equals total debts on the balance sheet of the firms:

$$\sum_f \sum_b L_f^b = \sum_f \sum_b D_b^f$$

  - Total money holdings on the balance sheets of households equals total household deposits on the balance sheets of the banks (summed across all banks):

$$\sum_h M^h = \sum_b \sum_h M_h^b$$

  - Total government bonds equals total bond holdings by all households, plus bond holdings of the Central Bank (due to QE):

$$n^g = \sum_h n_g^h + n_g^c$$

Monetary aggregates and invariants

- In the EURACE model we have a monetary conservation rule (invariant quantity):

$$\underbrace{\Big(\sum_h M^h + \sum_f M^f\Big)}_{\text{private sector deposits}} + \underbrace{\Big(\sum_b M^b\Big)}_{\text{bank cash reserves}} + \underbrace{\Big(\sum_g M^g + M^c\Big)}_{\text{public sector deposits}}$$

$$=$$

$$\underbrace{\Big(M^c + \sum_b D^c_b + \sum_g L^c_g\Big)}_{\text{fiat money}} + \underbrace{\Big(\sum_b \sum_f L^b_f\Big)}_{\text{credit money}}$$

11.2 List of tested rules

Balance sheet identities

Rule 1

Firm balance sheet: assets and liabilities.

firm_payment_account+firm_total_value_local_inventory+firm_total_value_capital_stock=firm_total_debt+firm_equity

Rule 2

IGFirm balance sheet: assets and liabilities.

igfirm_payment_account+igfirm_total_value_local_inventory+igfirm_total_value_capital_stock=igfirm_total_debt+igfirm_equity

Rule 3

Bank balance sheet: assets and liabilities.

bank_cash+bank_total_credit=bank_deposits+bank_equity+bank_ecb_debt

Rule 4

Government balance sheet: assets and liabilities.

gov_payment_account=gov_value_bonds_outstanding+gov_ecb_debt+gov_equity

Rule 5

Central bank balance sheet: assets and liabilities.

The issued fiat money to government(s) equals the total value of bond purchases by the ECB (open market operations). Note that the value of bond holdings is valued at the face value, while the bonds were purchased against the market bond price. The difference in value is subsumed in the ECB equity.

ecb_cash+ecb_gov_bond_holdings+ecb_fiat_money_banks=ecb_payment_account_banks+ecb_payment_account_govs+ecb_fiat_money+ecb_equity

Internal acounts: updating stocks with flows

Rule 6

Firm internal accounting of monetary flows.

firm_net_inflow=firm_payment_account_day_20-firm_payment_account_day_1

Rule 7

IGFirm internal accounting of monetary flows.

igfirm_net_inflow=igfirm_payment_account_day_20-igfirm_payment_account_day_1

Rule 8

Household internal accounting of monetary flows.

household_net_inflow=household_payment_account_day_20-household_payment_account_day_1

Rule 9

Bank internal accounting of monetary flows.

bank_net_inflow=bank_cash_day_20-bank_cash_day_1

Rule 10

Government internal accounting of monetary flows.

gov_net_inflow=gov_payment_account_day_20-gov_payment_account_day_1

Aggregate monetary flows between agents (or across sectors)

Banking sector

Rule 11

Payment accounts: aggregate bank deposits equals the sum of payment accounts in agent memory.

bank_deposits=firm_payment_account+igfirm_payment_account+hh_payment_account

Rule 12

Credit money: aggregate Bank credit outstanding equals total loans to firms.

bank_credit=firm_total_debt+igfirm_total_debt

Rule 13

Net inflows: the change in banks deposits equals the sum of the agents net inflows

bank_net_deposit_inflow=firm_net_inflow+igfirm_net_inflow+household_net_inflow

Rule 14

Debt installment payments by firms to banks and the payments received by banks are equal. firm_debt_installments+igfirm_debt_installment=bank_received_loan_installment

Rule 15

Interest payments by firms to banks are equal to interest payments received.

firm_interest_payment+igfirm_interest_payment=bank_received_interest_payment

Real sector

Rule 16

Dividend payments send and received are equal.

firm_dividend_payment+igfirm_dividend_payment+bank_dividend_payment=household_total_dividends

Rule 17

Consistency between IGFirm sales revenues and Firm capital costs.

firm_capital_costs=igfirm_revenue

Rule 18

Consistency between IGFirm and Firm labour costs and Households received wages.

household_wage=firm_labour_costs+igfirm_labour_costs

Rule 19

Consistency between Firm revenues and Households consumption expenditures

household_consumption_expenditure=firm_revenue

Eurostat

Rule 20

Definition of GDP.

eurostat_gdp=eurostat_investment_value+household_consumption_expenditure+gov_consumption_expenditure

Rule 21

Check the number of active firms:

eurostat_no_firms=firm_active+igfirm_active+eurostat_no_firm_bankruptcies

Rule 22

Material quantity conservation rule: Eurostat total sold quantity compared with firm data on number of goods sold (in volume).

eurostat_sold_quantity=firm_total_sold_quantity_volume

Rule 23

Investments in monetary value: Eurostat data (aggregated across the firms' investment costs) equals the IGFirm revenues.

eurostat_investment_value=igfirm_revenues

Government

Rule 24

Taxes paid equal tax revenues.

gov_tax_revenue=firm_tax_payment+igfirm_tax_payment+household_tax_payment+bank_tax_payment

Rule 25

Consistency of unemployment benefits paid by governments and received by households.

gov_benefit_payment-gov_restitution_payment=household_unemployment_benefit-household_restitution_payment

Rule 26

Total number of shares outstanding equals the total number of shares in household and ECB portfolios.

household_nr_assets+ecb_nr_gov_bonds=firm_outstanding_shares+igfirm_outstanding_shares+bank_outstanding_shares+gov_nr_bonds_oustanding

Central Bank

Rule 27

Deposits at ECB are consistent with memory contents of bank and government.

bank_cash+gov_payment_account=ecb_payment_account_banks+ecb_payment_account_govs

Rule 28

ECB fiat money is by definition the sum of the fiat money created for governments and for banks.

ecb_fiat_money=ecb_fiat_money_govs+ecb_fiat_money_banks

Rule 29

ECB fiat money banks is by definition the fiat money created for banks when they take on ECB debt.

ecb_fiat_money_banks=bank_ecb_debt

Rule 30

The interest payments by banks on central bank loans equals the ECB's interest receipts.

bank_ecb_interest_payment=ecb_bank_interest

Rule 31

The ECB dividend payment to Governments equals the dividend received.

ecb_dividend_payment=gov_ecb_dividend

Rule 32

Monetary conservation rule: credit money plus fiat money. All deposits in the banking sector plus bank equity, plus all idle cash balances held in the public sector (government payment account and ECB cash), equals all the credit money created by the banks plus all the fiat money created by the central bank.

bank_deposits+bank_equity+gov_payment_account+ecb_cash=bank_credit+ecb_fiat_money

Rule 33

Simplified monetary conservation rule: fiat money. Leaving out bank deposits, bank equity, ECB cash and bank total credit. The idle cash balances of bank and government equals the total fiat money created for the government.

bank_cash+gov_payment_account=ecb_fiat_money_govs

Why certain rules fail

Note that there is a slight difference between the LHS and RHS of rule 20 which compares the monthly consumption expenditures of households to Eurostat GDP. The difference is maximally 0.66. We assume this is a statistical error in aggregating over 1600 household values.

```
Iteration 800: OK
                    LHS                     RHS
--------------------------------------------------------------
             0.660142000002          <      1.0
--------------------------------------------------------------
eurostat gdp                         = 1977.246489
household consumption expenditure    = 1977.906631
```

The rules 32 and 33 that measure the total outstanding fiat and credit money in the economy fail due to this difference in the monetary flows, but the error is accumulatived in the stocks over time. It takes on the value 37.7 at iteration 800. If in every month there is a flow difference of 0.66, then after 40 months this accumulates to 26.7, which is very close to the observed difference of 37.7.

```
Iteration 800 :   FAIL
                  LHS                              RHS
______________________________________________________________
            37.721527                 >=           1.0
bank_deposits                         =     18762.661212
bank_equity                           =      2243.575997
gov_payment_account                   =      1382.172985
ecb_cash                              =         0.000000
bank_total_credit                     =     13194.607750
ecb_fiat_money                        =      9231.523971
```

```
firm_payment_account + firm_total_value_local_inventory + firm_total_value_capital_stock = firm_total_debt + firm_equity
igfirm_payment_account + igfirm_total_value_local_inventory + igfirm_total_value_capital_stock = igfirm_total_debt + igfirm_equity
bank_cash + bank_total_credit = bank_deposits + bank_equity + bank_ecb_debt
gov_payment_account = gov_value_bonds_outstanding + gov_ecb_debt + gov_equity
ecb_cash + ecb_gov_bond_holdings + ecb_fiat_money_banks = ecb_payment_account_banks + ecb_payment_account_govs + ecb_fiat_money + ecb_equity
firm_net_inflow = firm_payment_account_day_20 − firm_payment_account_day_1
igfirm_net_inflow = igfirm_payment_account_day_20 − igfirm_payment_account_day_1
household_net_inflow = household_payment_account_day_20 − household_payment_account_day_1
bank_net_inflow = bank_cash_day_20 − bank_cash_day_1
gov_net_inflow = gov_payment_account_day_20 − gov_payment_account_day_1
bank_deposits = firm_payment_account + igfirm_payment_account + hh_payment_account
bank_credit = firm_total_debt + igfirm_total_debt
bank_net_deposit_inflow = firm_net_inflow + igfirm_net_inflow + household_net_inflow
firm_debt_installments + igfirm_debt_installment = bank_received_loan_installment
firm_interest_payment + igfirm_interest_payment = bank_received_interest_payment
firm_dividend_payment + igfirm_dividend_payment + bank_dividend_payment = household_total_dividends
firm_capital_costs = igfirm_revenue
household_wage = firm_labour_costs + igfirm_labour_costs
household_consumption_expenditure = firm_revenue
eurostat_gdp = eurostat_investment_value + household_consumption_expenditure + gov_consumption_expenditure
eurostat_no_firms = firm_active + igfirm_active + eurostat_no_firm_bankruptcies
eurostat_sold_quantity = firm_total_sold_quantity_volume
eurostat_investment_value = igfirm_revenues
gov_tax_revenue = firm_tax_payment + igfirm_tax_payment + household_tax_payment + bank_tax_payment
gov_benefit_payment − gov_restitution_payment = household_unemployment_benefit − household_restitution_payment
household_nr_assets + ecb_nr_gov_bonds = firm_outstanding_shares + igfirm_outstanding_shares + bank_outstanding_shares + gov_nr_bonds_oustand
bank_cash + gov_payment_account = ecb_payment_account_banks + ecb_payment_account_govs
ecb_fiat_money = ecb_fiat_money_govs + ecb_fiat_money_banks
ecb_fiat_money_banks = bank_ecb_debt
bank_ecb_interest_payment = ecb_bank_interest
ecb_dividend_payment = gov_ecb_dividend
bank_deposits + bank_equity + gov_payment_account + ecb_cash = bank_credit + ecb_fiat_money
bank_cash + gov_payment_account = ecb_fiat_money_govs
```

11.3 Validation output

We show the rule validation output for every rule, for iterations $80, 320, 560, 800$.

```
RULE 1 : abs ( firm_payment_account + firm_value_local_inventory
 + firm_value_capital_stock − firm_total_debt − firm_equity ) < PRECISION


           ——— [ Iteration 80 : OK ] ———

                    LHS                               RHS
    ——————————————————————————————————————————————————————————————
        3.30000002577e−05        <                    1.0
                   0.0033%                        99.9967%
    ——————————————————————————————————————————————————————————————

     firm_payment_account                 =        1743.418907
     firm_value_local_inventory           =        1848.125816
     firm_value_capital_stock             =       16950.137440
     firm_total_debt                      =       12939.987931
     firm_equity                          =        7601.694199


           ——— [ Iteration 320 : OK ] ———

                    LHS                               RHS
    ——————————————————————————————————————————————————————————————
        3.99998589273e−06        <                    1.0
                   0.0004%                        99.9996%
    ——————————————————————————————————————————————————————————————

     firm_payment_account                 =        3304.877863
     firm_value_local_inventory           =        1585.217715
     firm_value_capital_stock             =       15042.124810
     firm_total_debt                      =       12940.009153
     firm_equity                          =        6992.211239


           ——— [ Iteration 560 : OK ] ———

                    LHS                               RHS
    ——————————————————————————————————————————————————————————————
        4.70000040877e−05        <                    1.0
                   0.0047%                        99.9953%
    ——————————————————————————————————————————————————————————————

     firm_payment_account                 =        2330.215346
     firm_value_local_inventory           =        1808.648398
     firm_value_capital_stock             =       13356.469262
     firm_total_debt                      =       12980.886231
     firm_equity                          =        4514.446822


           ——— [ Iteration 800 : OK ] ———

                    LHS                               RHS
    ——————————————————————————————————————————————————————————————
        1.30000048557e−05        <                    1.0
                   0.0013%                        99.9987%
    ______________________________________________________

    firm_payment_account                =       1759.440843
    firm_value_local_inventory          =       1816.642874
    firm_value_capital_stock            =      11944.331841
    firm_total_debt                     =      13194.607752
    firm_equity                         =       2325.807793

  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

RULE 2 : abs (  igfirm_payment_account  +  igfirm_value_local_inventory
+  igfirm_value_capital_stock  -  igfirm_total_debt  -  igfirm_equity )  <  PRECISION

              ----- [ Iteration 80 :  OK ] -----

                    LHS                              RHS
    ______________________________________________________
                     0.0        <                    1.0
                 0.0000%                       100.0000%
    ______________________________________________________

    igfirm_payment_account              =          0.000000
    igfirm_value_local_inventory        =          0.000000
    igfirm_value_capital_stock          =          0.000000
    igfirm_total_debt                   =          0.000000
    igfirm_equity                       =          0.000000

              ----- [ Iteration 320 :  OK ] -----

                    LHS                              RHS
    ______________________________________________________
                     0.0        <                    1.0
                 0.0000%                       100.0000%
    ______________________________________________________

    igfirm_payment_account              =          0.000000
    igfirm_value_local_inventory        =          0.000000
    igfirm_value_capital_stock          =          0.000000
    igfirm_total_debt                   =          0.000000
    igfirm_equity                       =          0.000000

              ----- [ Iteration 560 :  OK ] -----

                    LHS                              RHS
    ______________________________________________________
                     0.0        <                    1.0
                 0.0000%                       100.0000%
    ______________________________________________________

    igfirm_payment_account              =          0.000000
    igfirm_value_local_inventory        =          0.000000
    igfirm_value_capital_stock          =          0.000000
    igfirm_total_debt                   =          0.000000
    igfirm_equity                       =          0.000000
```

```
            ——— [ Iteration 800 : OK ] ———

                    LHS                               RHS
     ___________________________________________________________
                    0.0          <                    1.0
                0.0000%                          100.0000%
     ___________________________________________________________

     igfirm_payment_account          =         0.000000
     igfirm_value_local_inventory    =         0.000000
     igfirm_value_capital_stock      =         0.000000
     igfirm_total_debt               =         0.000000
     igfirm_equity                   =         0.000000

  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

RULE 3 : abs ( bank_cash + bank_total_credit − bank_deposits − bank_ecb_debt
− bank_equity ) < PRECISION

            ——— [ Iteration 80 : OK ] ———

                    LHS                               RHS
     ___________________________________________________________
       9.99997610052e−07         <                    1.0
                0.0001%                           99.9999%
     ___________________________________________________________

     bank_cash                       =      7732.945957
     bank_total_credit               =     12939.987927
     bank_deposits                   =     18435.183686
     bank_ecb_debt                   =         0.000000
     bank_equity                     =      2237.750197

            ——— [ Iteration 320 : OK ] ———

                    LHS                               RHS
     ___________________________________________________________
       1.81898940355e−12         <                    1.0
                0.0000%                          100.0000%
     ___________________________________________________________

     bank_cash                       =      9181.479966
     bank_total_credit               =     12940.009149
     bank_deposits                   =     19883.738862
     bank_ecb_debt                   =         0.000000
     bank_equity                     =      2237.750253

            ——— [ Iteration 560 : OK ] ———

                    LHS                               RHS
     ___________________________________________________________
       4.09272615798e−12         <                    1.0
                0.0000%                          100.0000%
     ___________________________________________________________

     bank_cash                       =      8595.186580
```

```
    bank_total_credit              =        12980.886226
    bank_deposits                  =        19337.916998
    bank_ecb_debt                  =            0.000000
    bank_equity                    =         2238.155808


             ——— [ Iteration 800 :  OK ] ———

                     LHS                          RHS
   ____________________________________________________
      1.00000033854e−06       <                  1.0
                0.0001%                      99.9999%
   ____________________________________________________

    bank_cash                      =         7811.629460
    bank_total_credit              =        13194.607750
    bank_deposits                  =        18762.661212
    bank_ecb_debt                  =            0.000000
    bank_equity                    =         2243.575997
 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
RULE 4 : abs (  gov_payment_account  −  gov_value_bonds_outstanding  −  gov_ecb_debt
−  gov_equity )  <  PRECISION


             ——— [ Iteration 80 :  OK ] ———

                     LHS                          RHS
   ____________________________________________________
                    0.0       <                  1.0
                0.0000%                     100.0000%
   ____________________________________________________

    gov_payment_account            =          −212.079110
    gov_value_bonds_outstanding    =         3545.500000
    gov_ecb_debt                   =            0.000000
    gov_equity                     =        −3757.579110


             ——— [ Iteration 320 :  OK ] ———

                     LHS                          RHS
   ____________________________________________________
      1.81898940355e−12       <                  1.0
                0.0000%                     100.0000%
   ____________________________________________________

    gov_payment_account            =           −63.143590
    gov_value_bonds_outstanding    =        10773.300000
    gov_ecb_debt                   =            0.000000
    gov_equity                     =       −10836.443590


             ——— [ Iteration 560 :  OK ] ———

                     LHS                          RHS
   ____________________________________________________
      1.81898940355e−12       <                  1.0
```

```
                 0.0000%                         100.0000%
    ____________________________________________________________

    gov_payment_account                     =        606.808140
    gov_value_bonds_outstanding             =      12410.100000
    gov_ecb_debt                            =          0.000000
    gov_equity                              =     -11803.291860

            ——— [ Iteration 800 :  OK ] ———

                          LHS                            RHS
    ____________________________________________________________
        1.81898940355e-12       <                        1.0
                 0.0000%                         100.0000%
    ____________________________________________________________

    gov_payment_account                     =       1382.172985
    gov_value_bonds_outstanding             =      12410.100000
    gov_ecb_debt                            =          0.000000
    gov_equity                              =     -11027.927015

  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

RULE 5 : abs (  ecb_cash  +  ecb_gov_bond_holdings  +  ecb_fiat_money_banks
-  ecb_payment_account_banks  -  ecb_payment_account_govs  -  ecb_fiat_money  -  ecb_equity )
<  PRECISION

            ——— [ Iteration 80 :  OK ] ———

                          LHS                            RHS
    ____________________________________________________________
                     0.0        <                        1.0
                 0.0000%                         100.0000%
    ____________________________________________________________

    ecb_cash                                =          0.000000
    ecb_gov_bond_holdings                   =        345.500000
    ecb_fiat_money_banks                    =          0.000000
    ecb_payment_account_banks               =       7732.945957
    ecb_payment_account_govs                =       -212.079110
    ecb_fiat_money                          =       7540.836535
    ecb_equity                              =     -14716.203382

            ——— [ Iteration 320 :  OK ] ———

                          LHS                            RHS
    ____________________________________________________________
                     0.0        <                        1.0
                 0.0000%                         100.0000%
    ____________________________________________________________

    ecb_cash                                =          0.000000
    ecb_gov_bond_holdings                   =       7573.300000
    ecb_fiat_money_banks                    =          0.000000
    ecb_payment_account_banks               =       9181.479967
    ecb_payment_account_govs                =        -63.143590
    ecb_fiat_money                  =       9143.066199
    ecb_equity                      =       −10688.102576


          ——— [ Iteration 560 :  OK ] ———

                 LHS                            RHS
  ─────────────────────────────────────────────────────
     1.81898940355e−12      <                   1.0
              0.0000%                        100.0000%
  ─────────────────────────────────────────────────────

    ecb_cash                        =          0.000000
    ecb_gov_bond_holdings           =       9210.100000
    ecb_fiat_money_banks            =          0.000000
    ecb_payment_account_banks       =       8595.186580
    ecb_payment_account_govs        =        606.808140
    ecb_fiat_money                  =       9231.523971
    ecb_equity                      =        −9223.418691


          ——— [ Iteration 800 :  OK ] ———

                 LHS                            RHS
  ─────────────────────────────────────────────────────
     1.00000033854e−06      <                   1.0
              0.0001%                        99.9999%
  ─────────────────────────────────────────────────────

    ecb_cash                        =          0.000000
    ecb_gov_bond_holdings           =       9210.100000
    ecb_fiat_money_banks            =          0.000000
    ecb_payment_account_banks       =       7811.629460
    ecb_payment_account_govs        =       1382.172985
    ecb_fiat_money                  =       9231.523971
    ecb_equity                      =        −9215.226417

 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

RULE 6 : abs (  firm_net_inflow  −  firm_payment_account_day_20  +  firm_payment_account_day_1
<  PRECISION


          ——— [ Iteration 80 :  OK ] ———

                 LHS                            RHS
  ─────────────────────────────────────────────────────
     7.99999975243e−06      <                   1.0
              0.0008%                        99.9992%
  ─────────────────────────────────────────────────────

    firm_net_inflow                 =        202.854799
    firm_payment_account_day_20     =       1743.418907
    firm_payment_account_day_1      =       1540.564116


          ——— [ Iteration 320 :  OK ] ———

                 LHS                            RHS
  ----------------------------------------------------------------
      3.99999817091e-06       <                      1.0
               0.0004%                           99.9996%
  ----------------------------------------------------------------

    firm_net_inflow                     =          63.881017
    firm_payment_account_day_20         =        3304.877863
    firm_payment_account_day_1          =        3240.996842

             ----- [ Iteration 560 :  OK ] -----

                      LHS                               RHS
  ----------------------------------------------------------------
      6.99999918652e-06       <                      1.0
               0.0007%                           99.9993%
  ----------------------------------------------------------------

    firm_net_inflow                     =          -77.922143
    firm_payment_account_day_20         =        2330.215346
    firm_payment_account_day_1          =        2408.137482

             ----- [ Iteration 800 :  OK ] -----

                      LHS                               RHS
  ----------------------------------------------------------------
      3.00000033349e-06       <                      1.0
               0.0003%                           99.9997%
  ----------------------------------------------------------------

    firm_net_inflow                     =          66.101219
    firm_payment_account_day_20         =        1759.440843
    firm_payment_account_day_1          =        1693.339621

..................................................................

RULE 7 : abs ( igfirm_net_inflow - igfirm_payment_account_day_20
+ igfirm_payment_account_day_1 ) < PRECISION

             ----- [ Iteration 80 : OK ] -----

                      LHS                               RHS
  ----------------------------------------------------------------
                      0.0     <                      1.0
               0.0000%                           100.0000%
  ----------------------------------------------------------------

    igfirm_net_inflow                   =            0.000000
    igfirm_payment_account_day_20       =            0.000000
    igfirm_payment_account_day_1        =            0.000000

             ----- [ Iteration 320 :  OK ] -----

                      LHS                               RHS
  ----------------------------------------------------------------
                      0.0     <                      1.0
```

```
                0.0000%                           100.0000%
  ______________________________________________________________


  igfirm_net_inflow                      =          0.000000
  igfirm_payment_account_day_20          =          0.000000
  igfirm_payment_account_day_1           =          0.000000


            ---- [ Iteration 560 :  OK ] ----

                     LHS                             RHS
  ______________________________________________________________
                     0.0        <                    1.0
                0.0000%                           100.0000%
  ______________________________________________________________


  igfirm_net_inflow                      =          0.000000
  igfirm_payment_account_day_20          =          0.000000
  igfirm_payment_account_day_1           =          0.000000


            ---- [ Iteration 800 :  OK ] ----

                     LHS                             RHS
  ______________________________________________________________
                     0.0        <                    1.0
                0.0000%                           100.0000%
  ______________________________________________________________


  igfirm_net_inflow                      =          0.000000
  igfirm_payment_account_day_20          =          0.000000
  igfirm_payment_account_day_1           =          0.000000

. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

RULE 8 : abs (  household_net_inflow  -  household_payment_account_day_20
+  household_payment_account_day_1 )  <  PRECISION


            ---- [ Iteration 80 :  OK ] ----

                     LHS                             RHS
  ______________________________________________________________
       8.30000353744e-05        <                    1.0
                0.0083%                           99.9917%
  ______________________________________________________________


  household_net_inflow                   =          4.732117
  household_payment_account_day_20       =      16691.764841
  household_payment_account_day_1        =      16687.032641


            ---- [ Iteration 320 :  OK ] ----

                     LHS                             RHS
  ______________________________________________________________
       5.00003079651e-06        <                    1.0
                0.0005%                           99.9995%
  ______________________________________________________________
```

```
    household_net_inflow                =          25.835734
    household_payment_account_day_20    =       16578.861019
    household_payment_account_day_1     =       16553.025280


            ——— [ Iteration 560 :  OK ] ———

                    LHS                              RHS
   ————————————————————————————————————————————————————————
       1.99999340111e−06      <                   1.0
                 0.0002%                       99.9998%
   ————————————————————————————————————————————————————————

    household_net_inflow                =          73.080692
    household_payment_account_day_20    =       17007.701652
    household_payment_account_day_1     =       16934.620962


            ——— [ Iteration 800 :  OK ] ———

                    LHS                              RHS
   ————————————————————————————————————————————————————————
       1.60000199685e−05      <                   1.0
                 0.0016%                       99.9984%
   ————————————————————————————————————————————————————————

    household_net_inflow                =          −10.519701
    household_payment_account_day_20    =       17003.220363
    household_payment_account_day_1     =       17013.740080

 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

RULE 9 : abs ( bank_net_inflow − bank_cash_day_20 + bank_cash_day_1 ) < PRECISION


            ——— [ Iteration 80 :  OK ] ———

                    LHS                              RHS
   ————————————————————————————————————————————————————————
       9.99999429041e−07      <                   1.0
                 0.0001%                       99.9999%
   ————————————————————————————————————————————————————————

    bank_net_inflow                     =         207.586918
    bank_cash_day_20                    =        7732.945957
    bank_cash_day_1                     =        7525.359040


            ——— [ Iteration 320 :  OK ] ———

                    LHS                              RHS
   ————————————————————————————————————————————————————————
       1.99999885808e−06      <                   1.0
                 0.0002%                       99.9998%
   ————————————————————————————————————————————————————————

    bank_net_inflow                     =          89.702946
    bank_cash_day_20                    =        9181.479966
    bank_cash_day_1                     =       9091.777022

             ---- [ Iteration 560 :  OK ] ----

                     LHS                            RHS
    ------------------------------------------------------
                     0.0       <                    1.0
                0.0000%                       100.0000%
    ------------------------------------------------------

    bank_net_inflow                     =         -19.160433
    bank_cash_day_20                    =       8595.186580
    bank_cash_day_1                     =       8614.347013

             ---- [ Iteration 800 :  OK ] ----

                     LHS                            RHS
    ------------------------------------------------------
                     0.0       <                    1.0
                0.0000%                       100.0000%
    ------------------------------------------------------

    bank_net_inflow                     =         27.395144
    bank_cash_day_20                    =       7811.629460
    bank_cash_day_1                     =       7784.234316

  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

RULE 10 : abs (  gov_net_inflow  -  gov_payment_account_day_20  +  gov_payment_account_day_1  )
<  PRECISION


             ---- [ Iteration 80 :  OK ] ----

                     LHS                            RHS
    ------------------------------------------------------
          9.99999976159e-07    <                    1.0
                0.0001%                        99.9999%
    ------------------------------------------------------

    gov_net_inflow                      =        -149.997474
    gov_payment_account_day_20          =        -212.079110
    gov_payment_account_day_1           =         -62.081635

             ---- [ Iteration 320 :  OK ] ----

                     LHS                            RHS
    ------------------------------------------------------
           7.1054273576e-15    <                    1.0
                0.0000%                       100.0000%
    ------------------------------------------------------

    gov_net_inflow                      =         -79.655002
    gov_payment_account_day_20          =         -63.143590
    gov_payment_account_day_1           =         16.511412
```

```
            —— [ Iteration 560 : OK ] ——

                   LHS                          RHS
    ______________________________________________________
                   0.0        <                 1.0
               0.0000%                      100.0000%
    ______________________________________________________

    gov_net_inflow                   =      18.681237
    gov_payment_account_day_20       =     606.808140
    gov_payment_account_day_1        =     588.126903

            —— [ Iteration 800 : OK ] ——

                   LHS                          RHS
    ______________________________________________________
                   0.0        <                 1.0
               0.0000%                      100.0000%
    ______________________________________________________

    gov_net_inflow                   =        −28.055276
    gov_payment_account_day_20       =    1382.172985
    gov_payment_account_day_1        =    1410.228261
```

. . . .

```
RULE 11 : abs ( firm_payment_account_day_20 + igfirm_payment_account_day_20 +
household_payment_account_day_20 − bank_deposits ) < PRECISION
```

```
            —— [ Iteration 80 : OK ] ——

                   LHS                          RHS
    ______________________________________________________
      6.19999882474e−05       <                 1.0
               0.0062%                       99.9938%
    ______________________________________________________

    firm_payment_account_day_20       =     1743.418907
    igfirm_payment_account_day_20     =        0.000000
    household_payment_account_day_20  =    16691.764841
    bank_deposits                     =    18435.183686

            —— [ Iteration 320 : OK ] ——

                   LHS                          RHS
    ______________________________________________________
      2.00000358745e−05       <                 1.0
               0.0020%                       99.9980%
    ______________________________________________________

    firm_payment_account_day_20       =     3304.877863
    igfirm_payment_account_day_20     =        0.000000
    household_payment_account_day_20  =    16578.861019
    bank_deposits                     =    19883.738862
```

```
          ---- [ Iteration 560 :  OK ] ----

                 LHS                            RHS
   -----------------------------------------------------
    1.09139364213e-11       <                   1.0
              0.0000%                       100.0000%
   -----------------------------------------------------

    firm_payment_account_day_20         =       2330.215346
    igfirm_payment_account_day_20       =          0.000000
    household_payment_account_day_20    =      17007.701652
    bank_deposits                       =      19337.916998

          ---- [ Iteration 800 :  OK ] ----

                 LHS                            RHS
   -----------------------------------------------------
    5.99999839324e-06       <                   1.0
              0.0006%                        99.9994%
   -----------------------------------------------------

    firm_payment_account_day_20         =       1759.440843
    igfirm_payment_account_day_20       =          0.000000
    household_payment_account_day_20    =      17003.220363
    bank_deposits                       =      18762.661212
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

RULE 12 : abs ( bank_total_credit - firm_total_debt - igfirm_total_debt ) <
PRECISION

          ---- [ Iteration 80 :  OK ] ----

                 LHS                            RHS
   -----------------------------------------------------
    3.99998862122e-06       <                   1.0
              0.0004%                        99.9996%
   -----------------------------------------------------

    bank_total_credit                   =      12939.987927
    firm_total_debt                     =      12939.987931
    igfirm_total_debt                   =          0.000000

          ---- [ Iteration 320 :  OK ] ----

                 LHS                            RHS
   -----------------------------------------------------
    3.99998862122e-06       <                   1.0
              0.0004%                        99.9996%
   -----------------------------------------------------

    bank_total_credit                   =      12940.009149
    firm_total_debt                     =      12940.009153
    igfirm_total_debt                   =          0.000000
```

```
            —— [ Iteration 560 :  OK ] ——

                  LHS                            RHS
    ─────────────────────────────────────────────────────
       4.99999259773e-06       <                 1.0
                 0.0005%                     99.9995%
    ─────────────────────────────────────────────────────

    bank_total_credit                 =      12980.886226
    firm_total_debt                   =      12980.886231
    igfirm_total_debt                 =          0.000000


            —— [ Iteration 800 :  OK ] ——

                  LHS                            RHS
    ─────────────────────────────────────────────────────
       1.99999158212e-06       <                 1.0
                 0.0002%                     99.9998%
    ─────────────────────────────────────────────────────

    bank_total_credit                 =      13194.607750
    firm_total_debt                   =      13194.607752
    igfirm_total_debt                 =          0.000000
  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

RULE 13 : abs ( firm_net_inflow + igfirm_net_inflow + household_net_inflow −
 bank_net_deposit_inflow ) < PRECISION


            —— [ Iteration 80 :  OK ] ——

                  LHS                            RHS
    ─────────────────────────────────────────────────────
       1.29999998819e-05       <                 1.0
                 0.0013%                     99.9987%
    ─────────────────────────────────────────────────────

    firm_net_inflow                   =        202.854799
    igfirm_net_inflow                 =          0.000000
    household_net_inflow              =          4.732117
    bank_net_deposit_inflow           =        207.586929


            —— [ Iteration 320 :  OK ] ——

                  LHS                            RHS
    ─────────────────────────────────────────────────────
       8.00000003665e-06       <                 1.0
                 0.0008%                     99.9992%
    ─────────────────────────────────────────────────────

    firm_net_inflow                   =         63.881017
    igfirm_net_inflow                 =          0.000000
    household_net_inflow              =         25.835734
    bank_net_deposit_inflow           =         89.716743
```

```
            ——— [ Iteration 560 : OK ] ———

                     LHS                              RHS
      ________________________________________________________
         2.99999989295e−06       <                  1.0
                   0.0003%                        99.9997%
      ________________________________________________________

      firm_net_inflow                  =          −77.922143
      igfirm_net_inflow                =           0.000000
      household_net_inflow             =          73.080692
      bank_net_deposit_inflow          =           −4.841448


            ——— [ Iteration 800 : OK ] ———

                     LHS                              RHS
      ________________________________________________________
         2.00000001627e−06       <                  1.0
                   0.0002%                        99.9998%
      ________________________________________________________

      firm_net_inflow                  =          66.101219
      igfirm_net_inflow                =           0.000000
      household_net_inflow             =          −10.519701
      bank_net_deposit_inflow          =          55.581516
 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

RULE 14 : abs ( firm_debt_installment + igfirm_debt_installment
− bank_received_loan_installment ) < PRECISION


            ——— [ Iteration 80 : OK ] ———

                     LHS                              RHS
      ________________________________________________________
                       0.0       <                  1.0
                   0.0000%                        100.0000%
      ________________________________________________________

      firm_debt_installment            =           0.000000
      igfirm_debt_installment          =           0.000000
      bank_received_loan_installment   =           0.000000


            ——— [ Iteration 320 : OK ] ———

                     LHS                              RHS
      ________________________________________________________
                     1e−06       <                  1.0
                   0.0001%                        99.9999%
      ________________________________________________________

      firm_debt_installment            =           0.002316
      igfirm_debt_installment          =           0.000000
      bank_received_loan_installment   =           0.002315
```

```
            ———— [ Iteration 560 : OK ] ————

                    LHS                              RHS
    ——————————————————————————————————————————————————————
        9.99999999252e−07        <                1.0
                  0.0001%                       99.9999%
    ——————————————————————————————————————————————————————

    firm_debt_installment                =          9.228251
    igfirm_debt_installment              =          0.000000
    bank_received_loan_installment       =          9.228252


            ———— [ Iteration 800 : OK ] ————

                    LHS                              RHS
    ——————————————————————————————————————————————————————
        9.99999983264e−07        <                1.0
                  0.0001%                       99.9999%
    ——————————————————————————————————————————————————————

    firm_debt_installment                =         66.613417
    igfirm_debt_installment              =          0.000000
    bank_received_loan_installment       =         66.613418

  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

RULE 15 : abs ( firm_interest_payment + igfirm_interest_payment
− bank_received_interest_payment ) < PRECISION


            ———— [ Iteration 80 : OK ] ————

                    LHS                              RHS
    ——————————————————————————————————————————————————————
                      0.0        <                1.0
                  0.0000%                      100.0000%
    ——————————————————————————————————————————————————————

    firm_interest_payment                =          0.000000
    igfirm_interest_payment              =          0.000000
    bank_received_interest_payment       =          0.000000


            ———— [ Iteration 320 : OK ] ————

                    LHS                              RHS
    ——————————————————————————————————————————————————————
        6.77626357803e−21        <                1.0
                  0.0000%                      100.0000%
    ——————————————————————————————————————————————————————

    firm_interest_payment                =          0.000031
    igfirm_interest_payment              =          0.000000
    bank_received_interest_payment       =          0.000031


            ———— [ Iteration 560 : OK ] ————
```

```
                      LHS                                   RHS
      ____________________________________________________________________
                    1e-06          <                        1.0
                0.0001%                                 99.9999%
      ____________________________________________________________________

      firm_interest_payment                  =             0.116475
      igfirm_interest_payment                =             0.000000
      bank_received_interest_payment         =             0.116476

                 ----- [ Iteration 800 :  OK ] -----

                      LHS                                   RHS
      ____________________________________________________________________
         2.00000000006e-06         <                        1.0
                0.0002%                                 99.9998%
      ____________________________________________________________________

      firm_interest_payment                  =             0.826099
      igfirm_interest_payment                =             0.000000
      bank_received_interest_payment         =             0.826101

  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

RULE 16 : abs (  firm_dividend_payment  +  igfirm_dividend_payment  +  bank_dividend_payment
-  household_total_dividends )   <  PRECISION

                 ----- [ Iteration 80 :  OK ] -----

                      LHS                                   RHS
      ____________________________________________________________________
         0.000389999999982         <                        1.0
                0.0390%                                 99.9610%
      ____________________________________________________________________

      firm_dividend_payment                  =            19.350010
      igfirm_dividend_payment                =             0.000000
      bank_dividend_payment                  =             0.000000
      household_total_dividends              =            19.350400

                 ----- [ Iteration 320 :  OK ] -----

                      LHS                                   RHS
      ____________________________________________________________________
         0.000202000000364         <                        1.0
                0.0202%                                 99.9798%
      ____________________________________________________________________

      firm_dividend_payment                  =            42.916598
      igfirm_dividend_payment                =             0.000000
      bank_dividend_payment                  =             0.000000
      household_total_dividends              =            42.916800

                 ----- [ Iteration 560 :  OK ] -----
```

```
                       LHS                                      RHS
        ─────────────────────────────────────────────────────────────
            0.000103999999386        <                   1.0
                      0.0104%                        99.9896%
        ─────────────────────────────────────────────────────────────

        firm_dividend_payment                  =          25.830504
        igfirm_dividend_payment                =           0.000000
        bank_dividend_payment                  =           0.000000
        household_total_dividends              =          25.830400

                  ——— [ Iteration 800 : OK ] ———

                       LHS                                      RHS
        ─────────────────────────────────────────────────────────────
            0.000531000000087        <                   1.0
                      0.0531%                        99.9469%
        ─────────────────────────────────────────────────────────────

        firm_dividend_payment                  =           4.117869
        igfirm_dividend_payment                =           0.000000
        bank_dividend_payment                  =           0.000000
        household_total_dividends              =           4.118400

  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

RULE 17 : abs ( firm_capital_costs − igfirm_revenue ) < PRECISION

                  ——— [ Iteration 80 : OK ] ———

                       LHS                                      RHS
        ─────────────────────────────────────────────────────────────
                          0.0        <                   1.0
                      0.0000%                       100.0000%
        ─────────────────────────────────────────────────────────────

        firm_capital_costs                     =           0.000000
        igfirm_revenue                         =           0.000000

                  ——— [ Iteration 320 : OK ] ———

                       LHS                                      RHS
        ─────────────────────────────────────────────────────────────
                          0.0        <                   1.0
                      0.0000%                       100.0000%
        ─────────────────────────────────────────────────────────────

        firm_capital_costs                     =           0.000000
        igfirm_revenue                         =           0.000000

                  ——— [ Iteration 560 : OK ] ———

                       LHS                                      RHS
        ─────────────────────────────────────────────────────────────
```

```
                    0.0        <                    1.0
               0.0000%                        100.0000%
---------------------------------------------------------

firm_capital_costs                 =           0.000000
igfirm_revenue                     =           0.000000


          ----- [ Iteration 800 :  OK ] -----

                   LHS                            RHS
---------------------------------------------------------
                    0.0        <                    1.0
               0.0000%                        100.0000%
---------------------------------------------------------

firm_capital_costs                 =           0.000000
igfirm_revenue                     =           0.000000
```

. . . .

```
RULE 18 : abs (  household_wage  -  firm_labour_costs  -  igfirm_labour_costs )  <
PRECISION


          ----- [ Iteration 80 :  OK ] -----

                   LHS                            RHS
---------------------------------------------------------
      0.000274000007266        <                    1.0
               0.0274%                         99.9726%
---------------------------------------------------------

household_wage                     =        1273.394386
firm_labour_costs                  =        1273.394660
igfirm_labour_costs                =           0.000000


          ----- [ Iteration 320 :  OK ] -----

                   LHS                            RHS
---------------------------------------------------------
        2.7000000955e-05       <                    1.0
               0.0027%                         99.9973%
---------------------------------------------------------

household_wage                     =        1622.990863
firm_labour_costs                  =        1622.990836
igfirm_labour_costs                =           0.000000


          ----- [ Iteration 560 :  OK ] -----

                   LHS                            RHS
---------------------------------------------------------
      9.00000100046e-06        <                    1.0
               0.0009%                         99.9991%
---------------------------------------------------------
```

```
    household_wage                      =        1941.965082
    firm_labour_costs                   =        1941.965073
    igfirm_labour_costs                 =           0.000000

              ——— [ Iteration 800 :  OK ] ———

                      LHS                              RHS
   ____________________________________________________________
       2.00000226869e-06        <                      1.0
                 0.0002%                           99.9998%
   ____________________________________________________________

    household_wage                      =        1903.243356
    firm_labour_costs                   =        1903.243358
    igfirm_labour_costs                 =           0.000000

 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

RULE 19 : abs (  household_consumption_expenditure  -  firm_revenue )  <  PRECISION

              ——— [ Iteration 80 :  OK ] ———

                      LHS                              RHS
   ____________________________________________________________
         0.0667990000006        <                      1.0
                 6.2616%                           93.7384%
   ____________________________________________________________

    household_consumption_expenditure =        1505.850472
    firm_revenue                        =        1505.783673

              ——— [ Iteration 320 :  OK ] ———

                      LHS                              RHS
   ____________________________________________________________
          0.610714000002        <                      1.0
                37.9157%                           62.0843%
   ____________________________________________________________

    household_consumption_expenditure =        1758.049286
    firm_revenue                        =        1757.438572

              ——— [ Iteration 560 :  OK ] ———

                      LHS                              RHS
   ____________________________________________________________
          0.479193999999        <                      1.0
                32.3956%                           67.6044%
   ____________________________________________________________

    household_consumption_expenditure =        1910.539720
    firm_revenue                        =        1910.060526

              ——— [ Iteration 800 :  OK ] ———
```

```
                 LHS                              RHS
    ________________________________________________________
          0.660147000003         <                 1.0
                39.7644%                        60.2356%
    ________________________________________________________

    household_consumption_expenditure =        1977.906631
    firm_revenue                      =        1977.246484

 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

RULE 20 : abs (  eurostat_gdp  −  eurostat_investment_value
−  household_consumption_expenditure  −  gov_consumption_expenditure  )  <  PRECISION


            ——— [ Iteration 80 :  OK ] ———

                 LHS                              RHS
    ________________________________________________________
         0.0668010000006         <                 1.0
                 6.2618%                        93.7382%
    ________________________________________________________

    eurostat_gdp                      =        1505.783671
    eurostat_investment_value         =           0.000000
    household_consumption_expenditure =        1505.850472
    gov_consumption_expenditure       =           0.000000


            ——— [ Iteration 320 :  OK ] ———

                 LHS                              RHS
    ________________________________________________________
          0.610710000002         <                 1.0
                37.9156%                        62.0844%
    ________________________________________________________

    eurostat_gdp                      =        1757.438576
    eurostat_investment_value         =           0.000000
    household_consumption_expenditure =        1758.049286
    gov_consumption_expenditure       =           0.000000


            ——— [ Iteration 560 :  OK ] ———

                 LHS                              RHS
    ________________________________________________________
          0.479192999999         <                 1.0
                32.3956%                        67.6044%
    ________________________________________________________

    eurostat_gdp                      =        1910.060527
    eurostat_investment_value         =           0.000000
    household_consumption_expenditure =        1910.539720
    gov_consumption_expenditure       =           0.000000


            ——— [ Iteration 800 :  OK ] ———
```

```
                    LHS                                  RHS
    ------------------------------------------------------------
           0.660142000002        <                   1.0
                 39.7642%                         60.2358%
    ------------------------------------------------------------

    eurostat_gdp                           =      1977.246489
    eurostat_investment_value              =         0.000000
    household_consumption_expenditure      =      1977.906631
    gov_consumption_expenditure            =         0.000000

. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

RULE 21 : abs (  eurostat_active_firms  -  firm_active  -  igfirm_active )  <  PRECISION

            ---- [ Iteration 80 :  OK ] ----

                    LHS                                  RHS
    ------------------------------------------------------------
                      0.0        <                   1.0
                  0.0000%                        100.0000%
    ------------------------------------------------------------

    eurostat_active_firms                  =        81.000000
    firm_active                            =        80.000000
    igfirm_active                          =         1.000000

            ---- [ Iteration 320 :  OK ] ----

                    LHS                                  RHS
    ------------------------------------------------------------
                      0.0        <                   1.0
                  0.0000%                        100.0000%
    ------------------------------------------------------------

    eurostat_active_firms                  =        80.000000
    firm_active                            =        79.000000
    igfirm_active                          =         1.000000

            ---- [ Iteration 560 :  OK ] ----

                    LHS                                  RHS
    ------------------------------------------------------------
                      0.0        <                   1.0
                  0.0000%                        100.0000%
    ------------------------------------------------------------

    eurostat_active_firms                  =        79.000000
    firm_active                            =        78.000000
    igfirm_active                          =         1.000000

            ---- [ Iteration 800 :  OK ] ----

                    LHS                                  RHS
```

```
   ________________________________________________________
                  0.0       <               1.0
             0.0000%                  100.0000%
   ________________________________________________________

   eurostat_active_firms          =       72.000000
   firm_active                    =       71.000000
   igfirm_active                  =        1.000000

 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

RULE 22 : abs ( eurostat_sold_quantity − firm_total_sold_quantity_volume ) <
PRECISION


            ——— [ Iteration 80 : OK ] ———

                  LHS                            RHS
   ________________________________________________________
     1.99999931283e−06      <               1.0
             0.0002%                   99.9998%
   ________________________________________________________

   eurostat_sold_quantity         =     2418.120105
   firm_total_sold_quantity_volume =    2418.120103


            ——— [ Iteration 320 : OK ] ———

                  LHS                            RHS
   ________________________________________________________
     4.54747350886e−13      <               1.0
             0.0000%                  100.0000%
   ________________________________________________________

   eurostat_sold_quantity         =     2940.147077
   firm_total_sold_quantity_volume =    2940.147077


            ——— [ Iteration 560 : OK ] ———

                  LHS                            RHS
   ________________________________________________________
     1.00000079328e−06      <               1.0
             0.0001%                   99.9999%
   ________________________________________________________

   eurostat_sold_quantity         =     3150.449715
   firm_total_sold_quantity_volume =    3150.449714


            ——— [ Iteration 800 : OK ] ———

                  LHS                            RHS
   ________________________________________________________
     2.99999874187e−06      <               1.0
             0.0003%                   99.9997%
   ________________________________________________________
```

```
    eurostat_sold_quantity              =        3022.819609
    firm_total_sold_quantity_volume     =        3022.819612

  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

RULE 23 : abs (  eurostat_investment_value  −  igfirm_revenue  )  <  PRECISION


            ——— [ Iteration 80 :  OK ] ———

                          LHS                           RHS
   ——————————————————————————————————————————————————————————
                          0.0       <                   1.0
                      0.0000%                     100.0000%
   ——————————————————————————————————————————————————————————

    eurostat_investment_value           =           0.000000
    igfirm_revenue                      =           0.000000


            ——— [ Iteration 320 :  OK ] ———

                          LHS                           RHS
   ——————————————————————————————————————————————————————————
                          0.0       <                   1.0
                      0.0000%                     100.0000%
   ——————————————————————————————————————————————————————————

    eurostat_investment_value           =           0.000000
    igfirm_revenue                      =           0.000000


            ——— [ Iteration 560 :  OK ] ———

                          LHS                           RHS
   ——————————————————————————————————————————————————————————
                          0.0       <                   1.0
                      0.0000%                     100.0000%
   ——————————————————————————————————————————————————————————

    eurostat_investment_value           =           0.000000
    igfirm_revenue                      =           0.000000


            ——— [ Iteration 800 :  OK ] ———

                          LHS                           RHS
   ——————————————————————————————————————————————————————————
                          0.0       <                   1.0
                      0.0000%                     100.0000%
   ——————————————————————————————————————————————————————————

    eurostat_investment_value           =           0.000000
    igfirm_revenue                      =           0.000000

  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

RULE 24 : abs (  gov_tax_revenue  −  firm_tax_payment  −  igfirm_tax_payment
−  household_tax_payment  −  bank_tax_payment )  <  PRECISION
```

```
        ---- [ Iteration 80 :  OK ] ----

                LHS                          RHS
--------------------------------------------------------
   0.000214000000327       <                 1.0
              0.0214%                    99.9786%
--------------------------------------------------------

gov_tax_revenue                      =        73.853949
firm_tax_payment                     =        10.184216
igfirm_tax_payment                   =         0.000000
household_tax_payment                =        63.669947
bank_tax_payment                     =         0.000000

        ---- [ Iteration 320 :  OK ] ----

                LHS                          RHS
--------------------------------------------------------
    1.3999999856e-05       <                 1.0
              0.0014%                    99.9986%
--------------------------------------------------------

gov_tax_revenue                      =       108.813462
firm_tax_payment                     =        27.663922
igfirm_tax_payment                   =         0.000000
household_tax_payment                =        81.149552
bank_tax_payment                     =         0.000002

        ---- [ Iteration 560 :  OK ] ----

                LHS                          RHS
--------------------------------------------------------
   1.29999998575e-05       <                 1.0
              0.0013%                    99.9987%
--------------------------------------------------------

gov_tax_revenue                      =       133.665222
firm_tax_payment                     =        34.500250
igfirm_tax_payment                   =         0.000000
household_tax_payment                =        99.159135
bank_tax_payment                     =         0.005824

        ---- [ Iteration 800 :  OK ] ----

                LHS                          RHS
--------------------------------------------------------
   3.60000000855e-05       <                 1.0
              0.0036%                    99.9964%
--------------------------------------------------------

gov_tax_revenue                      =       129.342269
firm_tax_payment                     =        31.929109
igfirm_tax_payment                   =         0.000000
household_tax_payment                =        97.371891
```

```
    bank_tax_payment                        =           0.041305

  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

RULE 25 : abs ( gov_benefit_payment  −  gov_restitution_payment
−  household_unemployment_benefit  +  household_restitution_payment  )  <  PRECISION

            −−−− [ Iteration 80 : OK ] −−−−

                LHS                                  RHS
   ──────────────────────────────────────────────────────
      6.09999994623e−05        <                    1.0
               0.0061%                           99.9939%
   ──────────────────────────────────────────────────────

    gov_benefit_payment                     =         386.597037
    gov_restitution_payment                 =         119.756064
    household_unemployment_benefit          =         386.596974
    household_restitution_payment           =         119.756062

            −−−− [ Iteration 320 : OK ] −−−−

                LHS                                  RHS
   ──────────────────────────────────────────────────────
      6.99999964127e−06        <                    1.0
               0.0007%                           99.9993%
   ──────────────────────────────────────────────────────

    gov_benefit_payment                     =         552.052624
    gov_restitution_payment                 =         367.592169
    household_unemployment_benefit          =         552.052632
    household_restitution_payment           =         367.592170

            −−−− [ Iteration 560 : OK ] −−−−

                LHS                                  RHS
   ──────────────────────────────────────────────────────
      6.00000055329e−06        <                    1.0
               0.0006%                           99.9994%
   ──────────────────────────────────────────────────────

    gov_benefit_payment                     =         541.968845
    gov_restitution_payment                 =         441.651527
    household_unemployment_benefit          =         541.968851
    household_restitution_payment           =         441.651527

            −−−− [ Iteration 800 : OK ] −−−−

                LHS                                  RHS
   ──────────────────────────────────────────────────────
      7.99999963874e−06        <                    1.0
               0.0008%                           99.9992%
   ──────────────────────────────────────────────────────

    gov_benefit_payment                     =         572.877247
    gov_restitution_payment             =        430.146368
    household_unemployment_benefit      =        572.877250
    household_restitution_payment       =        430.146363

  ..........................................................................

RULE 26 : abs ( firm_outstanding_shares + igfirm_outstanding_shares
+ bank_outstanding_shares + gov_nr_bonds_oustanding - household_nr_assets -
ecb_nr_gov_bonds ) < PRECISION

              ---- [ Iteration 80 : OK ] ----

                   LHS                           RHS
    ----------------------------------------------------
                   0.0        <                  1.0
               0.0000%                      100.0000%
    ----------------------------------------------------

    firm_outstanding_shares             =    1280000.000000
    igfirm_outstanding_shares           =      16000.000000
    bank_outstanding_shares             =      32000.000000
    gov_nr_bonds_oustanding             =      35455.000000
    household_nr_assets                 =    1360000.000000
    ecb_nr_gov_bonds                    =       3455.000000

              ---- [ Iteration 320 : OK ] ----

                   LHS                           RHS
    ----------------------------------------------------
                   0.0        <                  1.0
               0.0000%                      100.0000%
    ----------------------------------------------------

    firm_outstanding_shares             =    1280000.000000
    igfirm_outstanding_shares           =      16000.000000
    bank_outstanding_shares             =      32000.000000
    gov_nr_bonds_oustanding             =     107733.000000
    household_nr_assets                 =    1360000.000000
    ecb_nr_gov_bonds                    =      75733.000000

              ---- [ Iteration 560 : OK ] ----

                   LHS                           RHS
    ----------------------------------------------------
                   0.0        <                  1.0
               0.0000%                      100.0000%
    ----------------------------------------------------

    firm_outstanding_shares             =    1280000.000000
    igfirm_outstanding_shares           =      16000.000000
    bank_outstanding_shares             =      32000.000000
    gov_nr_bonds_oustanding             =     124101.000000
    household_nr_assets                 =    1360000.000000
    ecb_nr_gov_bonds                    =      92101.000000
```

```
          ——— [ Iteration 800 : OK ] ———

                     LHS                          RHS
    ________________________________________________________
                     0.0        <                 1.0
                  0.0000%                     100.0000%
    ________________________________________________________

    firm_outstanding_shares            =     1280000.000000
    igfirm_outstanding_shares          =       16000.000000
    bank_outstanding_shares            =       32000.000000
    gov_nr_bonds_oustanding            =      124101.000000
    household_nr_assets                =     1360000.000000
    ecb_nr_gov_bonds                   =       92101.000000

 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

RULE 27 : abs ( bank_cash + gov_payment_account − ecb_payment_account_banks −
ecb_payment_account_govs ) < PRECISION

          ——— [ Iteration 80 : OK ] ———

                     LHS                          RHS
    ________________________________________________________
       3.12638803734e−13        <                 1.0
                  0.0000%                     100.0000%
    ________________________________________________________

    bank_cash                          =        7732.945957
    gov_payment_account                =        −212.079110
    ecb_payment_account_banks          =        7732.945957
    ecb_payment_account_govs           =        −212.079110

          ——— [ Iteration 320 : OK ] ———

                     LHS                          RHS
    ________________________________________________________
       9.99998192697e−07        <                 1.0
                  0.0001%                      99.9999%
    ________________________________________________________

    bank_cash                          =        9181.479966
    gov_payment_account                =         −63.143590
    ecb_payment_account_banks          =        9181.479967
    ecb_payment_account_govs           =         −63.143590

          ——— [ Iteration 560 : OK ] ———

                     LHS                          RHS
    ________________________________________________________
       7.95807864051e−13        <                 1.0
                  0.0000%                     100.0000%
    ________________________________________________________

    bank_cash                          =        8595.186580
    gov_payment_account                =         606.808140
    ecb_payment_account_banks          =       8595.186580
    ecb_payment_account_govs           =        606.808140

             ——— [ Iteration 800 :  OK ] ———

                    LHS                             RHS
   ————————————————————————————————————————————————————————
        6.8212102633e−13      <                     1.0
                 0.0000%                        100.0000%
   ————————————————————————————————————————————————————————

    bank_cash                          =       7811.629460
    gov_payment_account                =       1382.172985
    ecb_payment_account_banks          =       7811.629460
    ecb_payment_account_govs           =       1382.172985

 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

RULE 28 : abs (  ecb_fiat_money  −  ecb_fiat_money_banks  −  ecb_fiat_money_govs )
<  PRECISION

             ——— [ Iteration 80 :  OK ] ———

                    LHS                             RHS
   ————————————————————————————————————————————————————————
                     0.0      <                     1.0
                 0.0000%                        100.0000%
   ————————————————————————————————————————————————————————

    ecb_fiat_money                     =       7540.836535
    ecb_fiat_money_banks               =          0.000000
    ecb_fiat_money_govs                =       7540.836535

             ——— [ Iteration 320 :  OK ] ———

                    LHS                             RHS
   ————————————————————————————————————————————————————————
                     0.0      <                     1.0
                 0.0000%                        100.0000%
   ————————————————————————————————————————————————————————

    ecb_fiat_money                     =       9143.066199
    ecb_fiat_money_banks               =          0.000000
    ecb_fiat_money_govs                =       9143.066199

             ——— [ Iteration 560 :  OK ] ———

                    LHS                             RHS
   ————————————————————————————————————————————————————————
                     0.0      <                     1.0
                 0.0000%                        100.0000%
   ————————————————————————————————————————————————————————

    ecb_fiat_money                     =       9231.523971
    ecb_fiat_money_banks               =          0.000000
    ecb_fiat_money_govs              =       9231.523971


              —— [ Iteration 800 :  OK ] ——

                        LHS                          RHS
    ____________________________________________________
                        0.0       <                  1.0
                    0.0000%                    100.0000%
    ____________________________________________________

    ecb_fiat_money                   =       9231.523971
    ecb_fiat_money_banks             =          0.000000
    ecb_fiat_money_govs              =       9231.523971

 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

RULE 29 : abs (  ecb_fiat_money_banks  -  bank_ecb_debt )   <  PRECISION


              —— [ Iteration 80 :  OK ] ——

                        LHS                          RHS
    ____________________________________________________
                        0.0       <                  1.0
                    0.0000%                    100.0000%
    ____________________________________________________

    ecb_fiat_money_banks             =          0.000000
    bank_ecb_debt                    =          0.000000


              —— [ Iteration 320 :  OK ] ——

                        LHS                          RHS
    ____________________________________________________
                        0.0       <                  1.0
                    0.0000%                    100.0000%
    ____________________________________________________

    ecb_fiat_money_banks             =          0.000000
    bank_ecb_debt                    =          0.000000


              —— [ Iteration 560 :  OK ] ——

                        LHS                          RHS
    ____________________________________________________
                        0.0       <                  1.0
                    0.0000%                    100.0000%
    ____________________________________________________

    ecb_fiat_money_banks             =          0.000000
    bank_ecb_debt                    =          0.000000


              —— [ Iteration 800 :  OK ] ——

                        LHS                          RHS
    ------------------------------------------------------------
                      0.0        <                  1.0
                 0.0000%                      100.0000%
    ------------------------------------------------------------

    ecb_fiat_money_banks              =           0.000000
    bank_ecb_debt                     =           0.000000

  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

RULE 30 : abs (  bank_ecb_interest_payment  -  ecb_bank_interest  )  <  PRECISION

            ----- [ Iteration 80 :  OK ] -----

                      LHS                               RHS
    ------------------------------------------------------------
                      0.0        <                  1.0
                 0.0000%                      100.0000%
    ------------------------------------------------------------

    bank_ecb_interest_payment         =           0.000000
    ecb_bank_interest                 =           0.000000

            ----- [ Iteration 320 :  OK ] -----

                      LHS                               RHS
    ------------------------------------------------------------
                      0.0        <                  1.0
                 0.0000%                      100.0000%
    ------------------------------------------------------------

    bank_ecb_interest_payment         =           0.000000
    ecb_bank_interest                 =           0.000000

            ----- [ Iteration 560 :  OK ] -----

                      LHS                               RHS
    ------------------------------------------------------------
                      0.0        <                  1.0
                 0.0000%                      100.0000%
    ------------------------------------------------------------

    bank_ecb_interest_payment         =           0.000000
    ecb_bank_interest                 =           0.000000

            ----- [ Iteration 800 :  OK ] -----

                      LHS                               RHS
    ------------------------------------------------------------
                      0.0        <                  1.0
                 0.0000%                      100.0000%
    ------------------------------------------------------------

    bank_ecb_interest_payment         =           0.000000
    ecb_bank_interest                 =           0.000000
```

```
 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

RULE 31 : abs ( ecb_dividend_payment - gov_ecb_dividend ) < PRECISION

            ----- [ Iteration 80 : OK ] -----

                     LHS                            RHS
    ----------------------------------------------------
                     0.0      <                     1.0
                 0.0000%                      100.0000%
    ----------------------------------------------------

    ecb_dividend_payment           =          1.243000
    gov_ecb_dividend               =          1.243000

            ----- [ Iteration 320 : OK ] -----

                     LHS                            RHS
    ----------------------------------------------------
                     0.0      <                     1.0
                 0.0000%                      100.0000%
    ----------------------------------------------------

    ecb_dividend_payment           =         34.010625
    gov_ecb_dividend               =         34.010625

            ----- [ Iteration 560 : OK ] -----

                     LHS                            RHS
    ----------------------------------------------------
                     0.0      <                     1.0
                 0.0000%                      100.0000%
    ----------------------------------------------------

    ecb_dividend_payment           =         42.212958
    gov_ecb_dividend               =         42.212958

            ----- [ Iteration 800 : OK ] -----

                     LHS                            RHS
    ----------------------------------------------------
                     0.0      <                     1.0
                 0.0000%                      100.0000%
    ----------------------------------------------------

    ecb_dividend_payment           =         42.212958
    gov_ecb_dividend               =         42.212958

 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

RULE 32 : abs ( bank_deposits + bank_equity + gov_payment_account + ecb_cash
- bank_total_credit - ecb_fiat_money ) < PRECISION
```

```
        ——— [ Iteration 80 :  FAIL ] ———

                 LHS                              RHS
    ________________________________________________
             19.969689      >=                    1.0
              95.2312%                        4.7688%
    ________________________________________________

    bank_deposits                    =     18435.183686
    bank_equity                      =      2237.750197
    gov_payment_account              =      −212.079110
    ecb_cash                         =         0.000000
    bank_total_credit                =     12939.987927
    ecb_fiat_money                   =      7540.836535

        ——— [ Iteration 320 :  FAIL ] ———

                 LHS                              RHS
    ________________________________________________
             24.729823      >=                    1.0
              96.1135%                        3.8865%
    ________________________________________________

    bank_deposits                    =     19883.738862
    bank_equity                      =      2237.750253
    gov_payment_account              =       −63.143590
    ecb_cash                         =         0.000000
    bank_total_credit                =     12940.009149
    ecb_fiat_money                   =      9143.066199

        ——— [ Iteration 560 :  FAIL ] ———

                 LHS                              RHS
    ________________________________________________
             29.529251      >=                    1.0
              96.7245%                        3.2755%
    ________________________________________________

    bank_deposits                    =     19337.916998
    bank_equity                      =      2238.155808
    gov_payment_account              =       606.808140
    ecb_cash                         =         0.000000
    bank_total_credit                =     12980.886226
    ecb_fiat_money                   =      9231.523971

        ——— [ Iteration 800 :  FAIL ] ———

                 LHS                              RHS
    ________________________________________________
             37.721527      >=                    1.0
              97.4175%                        2.5825%
    ________________________________________________

    bank_deposits                    =     18762.661212
    bank_equity                      =      2243.575997
    gov_payment_account              =      1382.172985
    ecb_cash                               =           0.000000
    bank_total_credit                      =       13194.607750
    ecb_fiat_money                         =        9231.523971

  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

RULE 33 : abs (  bank_cash  +  gov_payment_account  −  ecb_fiat_money_govs  )  <  PRECISION

             ——— [ Iteration 80 :  FAIL ] ———

                         LHS                                RHS
   ——————————————————————————————————————————————————————————————
                   19.969688      >=                      1.0
                    95.2312%                          4.7688%
   ——————————————————————————————————————————————————————————————

    bank_cash                              =        7732.945957
    gov_payment_account                    =        −212.079110
    ecb_fiat_money_govs                    =        7540.836535

             ——— [ Iteration 320 :  FAIL ] ———

                         LHS                                RHS
   ——————————————————————————————————————————————————————————————
                   24.729823      >=                      1.0
                    96.1135%                          3.8865%
   ——————————————————————————————————————————————————————————————

    bank_cash                              =        9181.479966
    gov_payment_account                    =         −63.143590
    ecb_fiat_money_govs                    =        9143.066199

             ——— [ Iteration 560 :  FAIL ] ———

                         LHS                                RHS
   ——————————————————————————————————————————————————————————————
                   29.529251      >=                      1.0
                    96.7245%                          3.2755%
   ——————————————————————————————————————————————————————————————

    bank_cash                              =        8595.186580
    gov_payment_account                    =         606.808140
    ecb_fiat_money_govs                    =        9231.523971

             ——— [ Iteration 800 :  FAIL ] ———

                         LHS                                RHS
   ——————————————————————————————————————————————————————————————
                   37.721526      >=                      1.0
                    97.4175%                          2.5825%
   ——————————————————————————————————————————————————————————————

    bank_cash                              =        7811.629460
    gov_payment_account                    =        1382.172985
    ecb_fiat_money_govs                    =        9231.523971
```

```
SUMMARY: 33 rules tested, 2 failed
```

Bibliography

Macedo e Silva, A., Dos Santos, C. H., 2008. The Keynesian Roots of Stock-flow Consistent Macroeconomic Models. Levy Institute of Economics of Bard College, Working Paper no. 537, URL: http://www.levy.org/pub/wp_537.pdf.

Squazzoni, F., 2010. The impact of agent-based models in social sciences after 15 years of incursion. History of Economic Ideas 8 (2), 197 – 233.

Wolf, S., Bouchaud, J.-P., Cecconi, F., Cincotti, S., Dawid, H., Gintis, H., van der Hoog, S., Jaeger, C. C., Kovalevsky, D. V., Mandel, A., Paroussos, L., 2013. Describing economic agent-based models – Dahlem ABM documentation guidelines. Complexity Economics 2 (1).