IIASA

International Institute for Applied Systems Analysis • A-2361 Laxenburg • Austria
Tel: +43 2236 807 • Fax: +43 2236 71313 • E-mail: info@iiasa.ac.at • Web: www.iiasa.ac.at

INTERIM REPORT IR-98-090/October

Technical Description of the Financial "Toy-Room"

Mariele Bertè (mberte@sda.uni-bocconi.it)

Approved by
Giovanni Dosi (dosi@iiasa.ac.at)
Leader, *TED Project*

Contents

Abstract

This paper contains the description of the code developed for the implementation of Financial "Toy-Room" (a micro-founded simulation model of decentralized trade in a homogeneous financial asset). The model has been created using the LSD platform. The paper describes how the various elements of the model have been organized and the main functions governing their evolution. After the illustration of the generic version, a specification of the of the behavioral repertoires is considered.

Acknowledgments

The author wishes to thank Yuri Kaniovski and Francesca Chiaromonte for their valuable comments.

About the Author

M. Bertè is from the University of Bocconi in Milano, Italy.

Technical Description of theFinancial "Toy-Room"

Mariele Bertè

Introduction

This paper describes the code developed to implement **Financial "Toy-Room"**, which is a micro-founded simulation model of decentralized trade in a homogeneous financial asset. A detailed description of the overall model is given in F.Chiaromonte, G.Dosi (1998), and therefore will not be repeated here. Some preliminary experiments have been realized with **Financial "Toy-Room"**, specifying trader's behavioral repertoires. The code used for this specification is described here, while details on experiments and results can be found in F.Chiaromonte, M.Bertè (1998)[1].

Financial "Toy-Room" has been implemented using the LSD platform. This choice has been made due to the facilitation in creating simulation models with LSD. Moreover, we tested LSD's potential using it to code a model quite different form the ones for which it was used in the past[2].

1 Model Implementation

Following the philosophy of **Financial "Toy-Room"**, the implementation of the model has been organized to be self-contained and modular, in a way that makes it easy to change certain features without influencing the rest of the model, or to run "sub-models". The structure of the model, which in LSD is defined in terms of objects and relations among them with a tree hierarchical structure, is represented in the following figure:

---

[1] A model in a similar spirit can be found in M. Bertè (1997).

[2] A complete description of the LSD platform is given in M. Valente, *Laboratory for Simulation Development User Manual,* IIASA Interim Report IR-97-020/May.

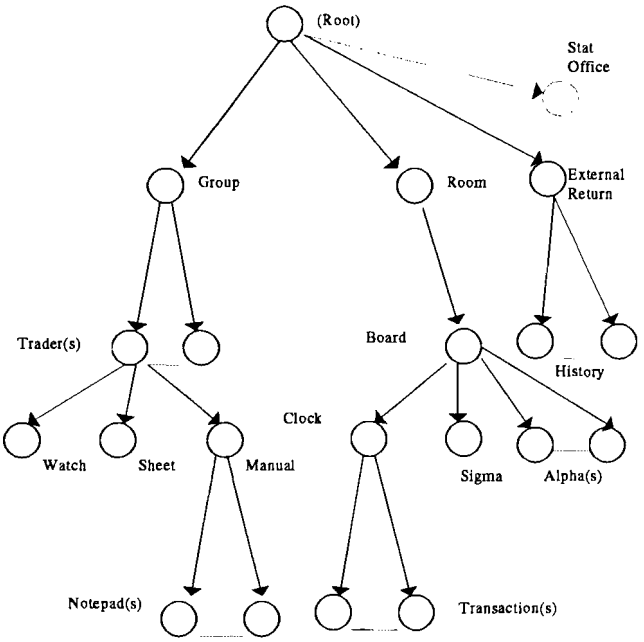

Figure 1

There are some differences between the conceptual model and the LSD implementation, due to the great exchange and repetition of information embodied in the former. The exchange of information among traders was implemented in LSD through the dynamic construction of objects containing it (objects NOTEPAD). The repetition of information regarding transactions[3] has been avoided by dynamically creating a sort of database for transactions (objects TRANSACTIONS), to which each trader has limited access. A statistical office has been conceptually included in the model, even though, loosely, statistical variables have been positioned in the objects hosting variables used in their calculation. Statistical variables concern demographics and attitudes of the traders , and transaction prices.

The model embodies three different timings that govern different features; the transaction round, the transaction (N) and the minute (H). The LSD time step t has been associated with the round; this was judged the most convenient choice (being the round the most basic timing, no information is lost). Every step of the simulation consists therefore of:

(i) Updating of demographic statistical variables and of trader's decision functions (when necessary),

(ii) New transaction round (with exchange of notepads),

(iii) Conclusion of a new transaction (if any),

(iv) Updating of statistical variables related to prices (when necessary), updating of the board and completion of transactions.

[3] In the conceptual model, each trader keeps a record of all the transactions he has been involved in, and the board has a tape with all the past prices.

Variables depending on other timings are updated only when necessary (the timing is shown on the tables: N = updated after every new transaction, H = updated after every minute). Since the values of these variables are of interest only upon updating, the user has the possibility of choosing the timing of the variables to be saved by fixing a parameter contained in the object ROOT (see appendix B). When ambiguous, the optimal timing for saving will be given for statistical variables, together with their description.

The following sections detail how the model was translated in LSD objects and their contents. The material has been divided in four parts, which correspond to the main blocks of the model; *root, market structure, traders and external returns.* The role of each LSD object is briefly described, and a table is given illustrating its contents in terms of variables and parameters.

Parameters, indicated with "(P)", keep the constant value assigned by the user (or automatically) at the beginning of the simulation throughout the run, unless their value is changed by the equation for a variable. *Variables*, indicated with their lag "(lag)"[4], are listed with a formal illustration of the *equations* used to calculate them at each time step. The most important equations are illustrated using a clear pseudo-code. This matches the code implemented in LSD, as can be found in the fun_*.cpp file (where equations are grouped by objects), except for a more "colloquial" style employed to facilitate the reading.

Some parameters, and the equations relative to some variables, are *experimental*: they have to be specified depending on the simulation experiment the user intends to perform (in the figures, experimental entities are marked by a parenthetical "exp"). Some of the experimental equations have been left as "random updatings" in the generic implementation, but for most we introduced a list of "reasonable possibilities" from which the user can choose through a parameter. There also are some purely technical parameters and equations; these govern the mechanics of the model, and are not subject to experimental specification (in the figures, technical entities are given in square brackets). Last, some variables have a lag for technical reasons (not because their initial value is relevant for experimental specification)[5]. The following tables emphasize what variables and parameters *must* be specified by the user before running a simulation.

1.1 Root

This object contains parameters that are not directly related to the conceptual model, for fixing saving options and termination conditions.

ROOT

sv_opt(P)	Parameter which enables to change timing in saving variables: 1=saving occurs every time step t (trading round)

[4] Regarding the lag system, x[0] hosts the current value of x, x[1] the one in the last time step, x[2] the one two time steps back, etc. x(lag) on the left hand side of a table means that the current value and all the lagged ones up to "lag" are available under those "names". The equation described on the right hand side of a table always refers to x[0].

[5] A variable specified with a lag equal to n has to be assigned n initial values by the user, corresponding to its values in the n past time steps. Even if the initial values are not relevant for experimental specification a lag might be needed to keep track of past values during the simulation.

	2=saving occurs every time a transaction is concluded (N[0]=N[1]+1) 3=saving occurs every time the minute changes (H[0]=H[0]+1) 4=saving occurs during the first step of a new minute (H[1]≠H[2])
end_sim(P)	Parameter used to fix the number of unsuccessful[6] rounds after which a simulation is ended (experimental: the simulation is ended if the number of unsuccessful rounds is greater than end_sim·(num_tr[0]/k[P])).

1.2 Market Structure

By market structure we mean the architectural characteristics of the room in which traders interact. As in the conceptual model, the room contains a board which conveys public information that is made up of the objects SIGMA (caller), ALPHA (caller) and CLOCK (which contains the transaction and minute counters). The object CLOCK nests objects that keep track of the concluded transactions (objects TRANSACTION), through which the price tape can be retrieved[7].The disclosure sheet, which is also an element of the board in the conceptual model, is embodied in the objects TRADERS (as will be described later).

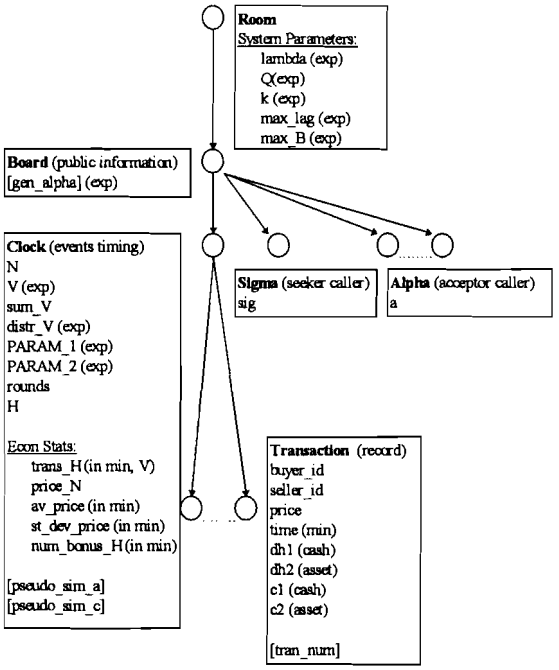

Figure 2: Market Structure

[6] An unsuccessful trading round is one in which no transaction is concluded.

[7] This has a purely technical reason, as the minute H on the clock advances following transaction conclusions (registered by the transaction counter N). This is also why statistics on transaction prices are contained in the clock. See later.

The object ROOM contains a series of experimental system parameters, which are fixed by the user at the beginning of the simulation. Here and in the following, italic symbols refer to the notation used for the conceptual model in F.Chiaromonte, G.Dosi (1998).

ROOM

lambda(P)	The acceptor's pricing power $\in[0,1]$, used to calculate the transaction price as p[o]=p[T]+(1-)p[T'].
Q(P)	Threshold of asset level over which a trader's identity, and his current asset level, are posted in the disclosure sheet on the display.
k(P)	Number of acceptors involved in each round.
max_lag(P)	Maximum permitted lag between conclusion of transactions and completion flows (deliveries and payments).
max_B(P)	Maximum permitted number of "bonus minutes" for traders[8].

The object BOARD nests objects defining elements of the board (CLOCK, SIGMA and ALPHA 1 and a technical variable corresponding to the equation that selects available acceptors for the current round[9].

BOARD

gen_alpha(0)	Lights up the caller for the acceptors.	The selection of the k[P] acceptors is presently implemented at random, as if any seeker, because of time constraints and/or distance, was only capable of screening at random k[P] among the available acceptors. Other criteria for choosing the acceptors could eventually be introduced. During the first round, the equation creates the k[P] objects ALPHA. In each following round, it assigns new values to the parameters a[P] contained in them, with the following procedure: 1.The number of available acceptors n=avail[0][10] is registered (the equation avail[0] has also sorted the traders putting the available acceptors first). 2. For each object ALPHA:

[8] A global value has to be fixed in order to eliminate old enough transactions (see later in description of objects).

[9] This is a technical expedient to concentrate the whole procedure in one location, and thus improve speed and modularization: Instead of having the identity of the acceptor as a variable within each object ALPHA (with an equation associated to it), we set these identities as parameters (a[P] in ALPHA), which are over-written one by one when the variable gen_alpha[0] in BOARD is updated.

[10] See description of the object GROUP later on.

		• if na random integer $x \in \{0,1...,n_\alpha - 1\}$ is generated. The object TRADER with position x in the tree is selected as an acceptor if his id[P]≠sig[0], otherwise a new random integer is generated and a new TRADER is selected. The selected trader's identity number is written in the a[P] of ALPHA. Else (na[P] is set to -1 • the selected trader's acceptor[0] is set to 0 • the objects TRADERS are sorted again, putting the remaining acceptors (traders with acceptor[0]=1) first • n n and the procedure is restarted from point 2 for the following ALPHA. (traders are sorted by increasing identity number)

The object CLOCK contains three different groups of variables and parameters:

1. Variables and parameters which control events timing
2. Variables of the statistical office concerning the economic statistics
3. Variables which control the pseudo-simultaneous transaction completion procedure.

The object CLOCK also nests an object TRANSACTION for each transaction that has been concluded.

CLOCK, Events timing:

N(1)	Transaction counter.	The equation completes a trading round which may (N[0]=N[1]+1) or may not (N[0]=N[1]) produce conclusion of a transaction. Conclusion is attempted between the trader with the identity equal to sig[0] (current seeker) and the trader with identity equal to the seeker's select[0] (the current acceptor selected by the current seeker): 1. Flags and reference prices of the seeker are checked. 2. If f_seek_b[0] =1 (the seeker is willing to buy) and b_or_s[0]=0 (the seeker has chosen to buy): • the lag intervals of seeker and acceptor are checked to find an intersection • if f_acc_s[0]=1 (the acceptor is willing to sell), p_seek_b[0]≥p_acc_s[0] (the reference prices of seeker and acceptor are compatible), and there is an intersection between the two lag intervals, a transaction is concluded with price_N[0] =

		lambda·p_seek_b[0] + (1-lambda) p_acc_s[0] • lags are generated randomly within the intersection • a new object TRANSACTION is created and all the information concerning the transaction is written in it • cash and asset levels of the two traders involved are updated if the transaction is spot on one or both sides • if a transaction was concluded, N[0]=N[1]+1. 3. If f_seek_s[0]=1 (the seeker is willing to sell), and b_or_s[0]=0 (the seeker has chosen to sell), the same steps of point 2 are performed.
V(1)	(experimental) Value of the current element in the system-level converting sequence, ν.	This variable is updated by the equation for H[0].
sum_V(1)	Sum of the elements in the system-level converting sequence, up to the current one.	This variable is updated by the equation for H[0].
rounds(1)	Unsuccessful round counter (counts rounds occurred since the last update of the transaction counter).	• if N[0]=N[1], then rounds[0]=rounds[1]+1 • if rounds[0]>end_sim·(num_tr[0]/k]), then end the simulation This equation was constructed to give the user the possibility of ending a simulation after a large enough number of unsuccessful rounds. The threshold can be seen as an experimental entity. The one specified here is justified in F.Chiaromonte, M.Bertè (1998), where failure in transaction conclusion occurs when the traders' reference prices are too close to each other to permit trade.
H(2)	Current minute. The lag of 2 is due to the dynamics of the model. When changing, the current minute is updated at the end of the last step/round which is considered part of the preceding minute. Thus, everything taking place during the round occurs during minute H[1], and H[2] represents the timing of the preceding round.	The corresponding equation updates the current minute if $\sum_{j=1}^{H[1]} \nu_j \le N$. If sum_V[0]≤N[0]: • if distr_V[P]=1, then V[0]=draw from a Poisson distribution with mean PARAM_1[P] + 1 (to avoid positive probability on 0) • if distr_V[P]=2, then V[0]=nearest integer of a draw from a lognormal distribution with mean PARAM_1[P] and standard deviation

		PARAM_2[P][11]. • if distr_V[P]=3, then V[0]=draw from a binomial distribution with parameters PARAM_1[P] (probability for each trial) and PARAM_2[P] (number of trials). • if distr_V[P]=4 then V[0] is always equal to the value fixed by the user at the beginning of the simulation (PARAM_1[P]) • sum_V[0]=sum_V[1]+V[0] • H[0]=H[1]+1.

distr_V(P)	Experimental parameter which enables to specify the distribution generating the elements of the system-level converting sequence: 1 = Poisson distribution 2 = Lognormal distribution 3 = Binomial distribution 4 = Constant value (V[0]=PARAM_1[P]).
PARAM_1(P)	First parameter of the distribution for V (experimental).
PARAM_2(P)	Second parameter of the distribution for V (experimental, if any[12]).

CLOCK, Economic statistics variables:

trans_H(1)	($=V_{t-1}$) Number of transactions within a minute (sv_opt =3)	The updating is given by • if H[1]≠H[2], restart the count (a new minute has started) • if N[0]≠N[1] (a transaction has occurred), then trans_H[0]=trans_H[1]+1 (or 1 if H[1] ≠H[2]).
price_N(1)	Price of the last concluded transaction.	This variable is updated by the equation for N[0].
av_price(1)	Average price of transactions within the last minute (sv_opt=3).	The average (and standard deviation) are calculated iteratively: • if H[1]≠H[2], then restart the count • if H[0]≠H[1] (the minute has finished), then

[11] The draw from a lognormal distribution is calculated by taking the exponential of a draw from a normal distribution with standard deviation $\sigma = \sqrt{\ln\left(\frac{\text{PARAM\_2[P]}^2}{e^{2\ln(\text{PARAM\_1[P]})}} + 1\right)}$ and mean $\text{PARAM\_1[P]} - 0.5\sigma^2$.

[12] All the distributions in the pre-defined collection above are identified by one, or at most two, parameters.

		$av\_price[0] = \frac{av\_price[1] + price\_N[0]}{trans\_H[0]}$ $st\_dev\_price[0] = \sqrt{\frac{(st\_dev\_price[1] + price\_N[0]^2)}{trans\_H[0]} - av\_price[0]^2}$ • else (H[0]=H[1]), if N[0]>N[1] (a transaction has occurred), then av_price[0]=av_price[1]+price_N[0] st_dev_price[0]=st_dev_price[1]+price_N[0]^2 • else (N[0]=N[1], meaning no transaction has occurred and thus H[0]=H[1]) av_price[0]=av_price[1] st_dev_price[0]=st_dev_price[1] (during each minute, the two variables store, respectively, the cumulated sums of prices and squared prices; at the end of each minute, they are transformed into the actual average and standard deviation by the formulas above).
st_dev_price(1)	Standard deviation of the price within the last minute (sv_opt=3)	This variable is updated by the equation for av_price[0].
num_bonus_H(1)	Total number of used bonus-minutes within the last minute, calculated at the end of each minute (sv_opt=3).	This variable is updated by the equations for pseudo_sim_a[0] and pseudo_sim_c[0].

CLOCK, Pseudo-simultaneous completion procedure:

pseudo_sim_a(0)	Activates completion procedure on the asset side.	See Appendix A.
pseudo_sim_c(0)	Activates completion procedure on the cash side.	Idem

The objects TRANSACTION act as a record of all the information concerning concluded transactions. Each object refers to one single transaction and is created dynamically when the transaction is concluded. Traders have only partial access to this data base: A trader has complete access to the objects TRANSACTION in which he is/has been a party as buyer or seller (in the conceptual model, this information is replicated in both the buyer's and the seller's notepads). Moreover, each trader has access to the price in each object TRANSACTION (in the conceptual model, this is public information and is found in the price tape on the board).

This formulation has a two-fold advantage in terms of running time. First, it reduces the size of the model by avoiding replication of information; this is important as the model is quite large. Second, from a technical point of view, it puts the objects TRANSACTION in the optimal position: Since they are generated by the equation for N, they are easy to include as descendants of the object containing N (CLOCK). Moreover, the completion procedure, itself located in CLOCK, needs information contained in the object TRANSACTION; retrieval is facilitated if they are nested in CLOCK.

In the current version of the model, all the transactions occurred since the beginning of the simulation are kept in the database. Even if all the elements of the objects TRANSACTION are parameters and therefore need no updating, the number of these objects may become very large and cause running time problems (especially due to the completion procedure, which needs to search all the objects). A solution to this could be to eliminate all the transactions with time[P]≤H[0]-max_lag-max_B[P]. This has not been included in the current version to leave is as general as possible.

TRANSACTION

buyer_id(P)	Buyer's identity b[o].
seller_id(P)	Seller's identity s[o].
price(P)	Price of the transaction p[o].
time(P)	Minute in which the transaction was concluded h[o].
dh1(P)	Lag for cash flow (payment) in minutes $dh_1[o]$.
dh2(P)	Lag for asset flow (delivery) in minutes $dh_2[o]$.
tran_num(P)	Technical parameter used for the completion procedure. It represents the position of the transaction in the matrix used in the algorithm.
c1(P)	Completion flag $c_1[o]$, indicating whether the transaction has been completed on the cash side (i.e. the payment has occurred): 0=not completed, 1=completed.
c2(P)	Completion flag $c_2[o]$, indicating whether the transaction has been completed on the asset side (i.e. the delivery has occurred).

The object SIGMA is a static object that represents the seeker caller.

SIGMA

sig(0)	Identity of the current seeker,	The equation selects a seeker for the round: • the objects TRADER are sorted putting available seekers first (traders with seeker[0] =1). • the number of available seekers n is registered • if na random integer $x \in \{0,1\ldots,n_\sigma - 1\}$ is generated, and the object TRADER with position

		x in the tree is selected as seeker for the round • sig[0] = identity of this trader • the traders are sorted again by increasing identity number .

The objects ALPHA are k (k[P]), and are created dynamically during the first time step of a simulation.

ALPHA

a(P)	Identity of one of the current acceptors, $_i$.

1.3 Traders

The second block of the model represents all the elements concerning the traders, following the description in F. Chiaromonte, G.Dosi (1998), with the exception of some information already included elsewhere in the model.

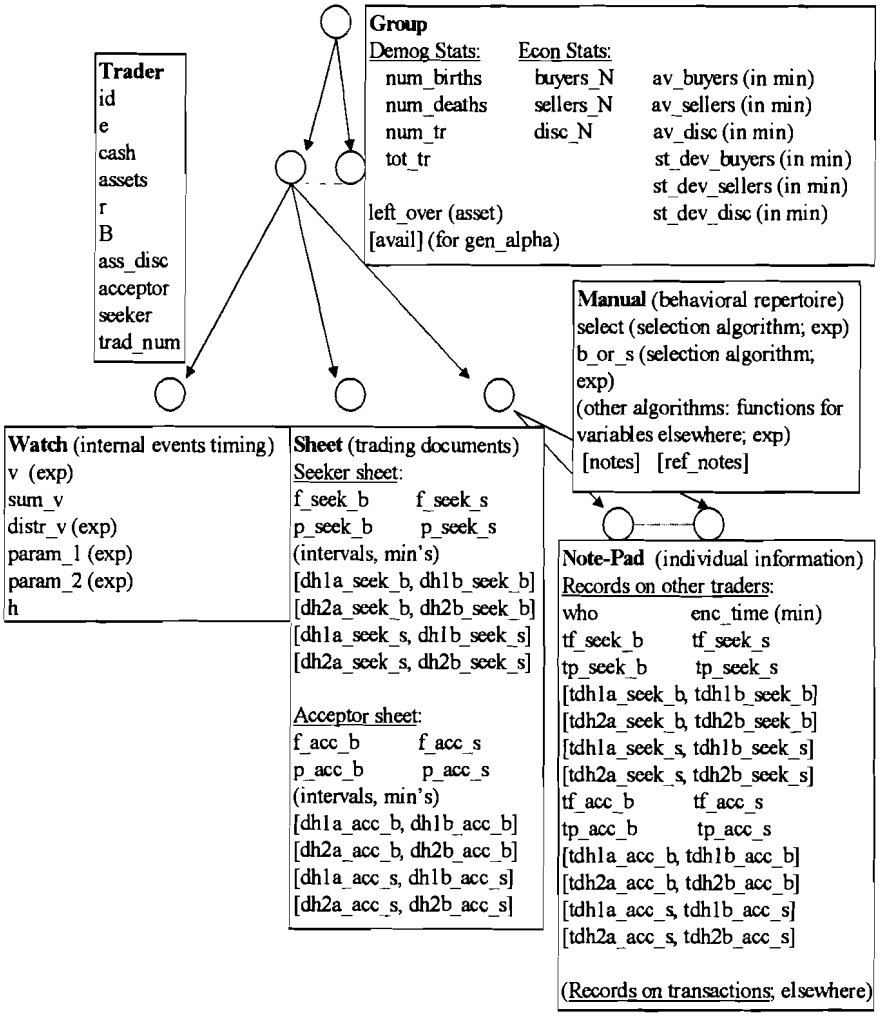

Figure 3: Traders

The objects TRADER are nested in an object GROUP, which contains mostly statistical variables, for both demographic and economic analysis, concerning the traders.

GROUP, Demographic statistics:

num_births(0)	Number of births at the beginning of a new minute (sv_opt=4).	This equation is divided in two parts: 1. During the first step of the simulation, the objects TRADER are created. In the generic version all the traders are identical to the one initialized by the user[13]. The user can modify this part to assign parameters differently. An instance in which traders are divided in subgroups is described in Section

[13] One object TRADER has been allocated when specifying the structure of the model. Most of the variables contained in this object must be initialized by the user and are used as a reference when, during the first time step, the other objects TRADER are generated.

		2. 2. New objects TRADER are created during the simulation, using an experimental equation that can be modified by the user[14]. The generic version goes as follows: • one or two traders enter the market every ten minutes • a new trader enters with a cash level equal to that of an existing trader selected at random, and an asset level equal to 0, unless there are some left over assets (left_over≠0) from some expelled traders. In this case these assets are evenly redistributed among the new entrants[15].
num_deaths(0)	Number of deaths at the end of the last minute (sv_opt=4)	• all traders are checked to see if there is at least one with e[0]=1 • if so, all trader's notepads are refreshed (all the information on the expelled traders is deleted with the equation for ref_notes[0]) • expelled traders are removed[16] .
num_tr(1)	Number of traders currently in the market. In the generic version, the initial number of traders has to be assigned by the user. If the traders are divided in subgroups (see Section 2), the user is required to define only the number of members in each subgroup (and not the total).	num_tr[0]=num_tr[1]+num_births[0]-num_deaths[0]
tot_tr(1)	Total number of traders who have been in the market (this is a technical variable used to assign id numbers to new traders entering the market).	tot_tr[0]=tot_tr[1]+num_births[0]

GROUP, Economic statistics:

[14] See discussion in F. Chiaromonte, G. Dosi (1998).

[15] The user will also have to initialize new traders' behavioral repertoires (the manuals in the conceptual model). See the discussion in F. Chiaromonte, G. Dosi (1998) for further details. We do not specify this initialization here, as the conceptual model's manuals are "empty". In terms of the behavioral framework we will describe in Section 2, initialization would correspond to assigning the behavioral parameters in the objects MANUAL.

[16] To perform the last two operations, a new function of the LSD class "object" (clean()) has been defined at the end of the fun_*.cpp file. This has been done because of the difficulty in handling operations involving the deletion of objects.

buyers_N(0)	Number of traders ready to buy in the current round	Iterated on all traders: if f_seek_b[0]=1 or f_acc_b[0]=1, then buyers_N[0]=buyers_N[0]+1
av_buyers(1)	Average number of traders ready to buy within the last minute (sv_opt=3).	See av_price[0] in CLOCK.
st_dev_buyers(1)	Standard deviation of the number of traders ready to buy within the last minute (sv_opt=3)	This variable is updated by the equation for av_buyers[0].
sellers_N(0)	Number of traders ready to sell in the current round.	Idem on f_seek_s[0]=1, or f_acc_s[0]=1.
av_sellers(1)	Average, as above.	Idem.
st_dev_sellers(1)	Standard deviation, as above.	Idem.
disc_N(1)	Number of traders listed on the disclosure sheet.	Idem, on ass_disc[0]>0.
av_disc(1)	Average, as above	Idem.
st_dev_disc(1)	Standard deviation, as above	Idem.

GROUP, technical:

avail(0)	Number of agents available as acceptors for each round.	• the objects TRADER are sorted putting the available acceptors first (traders with acceptor[0]=1). This is necessary for the functioning of the acceptor caller, which is managed by the equation for gen_alpha[0] • the available acceptors are counted • if there are no available acceptors the traders are sorted again by increasing identity number

left_over(P)	Assets left over by exiting traders (to be assigned to new entrants). This parameter is updated by the equation associated with e[1] (in objects TRADER; it handles the bankruptcy procedure) and by the equation associated with num_births[0] (when the leftover assets are assigned to the new traders).

The objects TRADER are created dynamically during the first time step by the equation associated with num_births(0). Each represents a single trader, and has descendants representing the trader's watch (WATCH), the trader's sheets (SHEET) and the trader's manual (MANUAL).

TRADER

e(1)	Termination flag: 0 = the trader remains in the room 1 = the trader is irreversibly removed from the room, i.e. the object is removed with all its descendants, after the bankruptcy procedure has been completed.	This equation handles the bankruptcy procedure, described in detail in F.Chiaromonte, G.Dosi (1998).
cash(2)	Current level of cash m[T].	Updated by the equations for num_births[0] (at the beginning of the simulation), N[0] (when a concluded transaction is spot on the cash side), pseudo_sim_c[0] (when a transaction is completed on the cash side) and e[0] (when a bankruptcy procedure is started, if the exiting trader still has some cash and some cash debits).
assets(2)	Current level of assets q[T].	Updated by the equations for num_births[0] (at the beginning of the simulation), N[0] (when a concluded transaction is spot on the asset side), pseudo_sim_c[0] (when a transaction is completed on the asset side) and e[0] (when a bankruptcy procedure is started, if the exiting trader still has some assets and some asset debits).
B(1)	Number of bonus-minutes still available (B[T]). This value is initialized automatically at the beginning of the simulation, and does not have to be specified by the user.	Updated by equations for num_births[0] (at the beginning of the simulation), pseudo_sim_a[0] and pseudo_sim_c[0] (when traders fails to complete transactions).
ass_disc(1)	Variable for the asset disclosure sheet: 0 = the trader is not on the disclosure sheet. >0 = the trader is on the disclosure sheet with assets[0]=asset_disc[0]. The asset disclosure sheet is not explicitly on the board. The trader has access to the variable ass_disc[0] of every other trader: when such variable is strictly positive, it represents the size in assets of the latter.	• if assets[0]>Q[P], then ass_disc[0]=assets[0] • else, ass_disc[0]=0
acceptor(0)	Technical flag used by the acceptors caller: 0 = the trader is not available as acceptor, or has already been chosen by the	• if f_acc_s[0]=1 or f_acc_b[0]=1, then acceptor[0]=1 • else, acceptor[0]=0

	acceptors caller. 1 = the trader is available as acceptor.	Moreover, in the equation for gen_alpha[0], when a trader is chosen by the acceptor caller, his acceptor[0] is set to 0.
seeker(0)	Technical flag used by the seeker caller. 0 = the trader is not available as see +ker 1 = the trader is available as seeker.	• if f_seek_s[0]=1 or f_seek_b[0]=1, then seeker[0]=1 • else, seeker[0]=0

id(P)	Trader's identity (technical parameter; an integer).
r(P)	Behavioral state. This parameter is experimental and is not used in the generic version. For an instance of its use, see Section 2. One could also imagine r[ ] as a variable, when modeling traders who change their behavioral state during the simulation[17].
trad_num(P)	Technical parameter used for the completion procedure. It represents the trader's position in the matrix used in the algorithm.

The object WATCH contains all the elements relative to the internal events timing, which are very similar in structure to the elements of CLOCK and are therefore not described in detail.

WATCH

v(1)	(experimental) Value of the current element in the trader's converting sequence, v[T].
sum_v(1)	Sum of the sequence up to the current element.
h(2)	Minute on the trader's watch H[T].
distr_v(P)	Experimental parameter which enables to specify the distribution generating the $v_i[T]$: 1 = Poisson distribution 2 = Lognormal distribution 3 = Binomial distribution 4 = The trader's watch works with the market's clock (v[0]=V[0]) 5 = Constant value (v[0]=param_1[P]).
param_1(P)	First parameter of the distribution for v (experimental).
param_2(P)	Second parameter of the distribution for v (experimental, if any).

The object SHEET contains the trader's trading documents, i.e. the information about the trader's current position and targets. This is also the information that is exchanged between two agents when they meet as seeker and acceptor, and then stored in the notepads. All the elements

[17] For example, when trading is non-spot one could specify a *regular* and a *red-alert* state for traders, the latter being connected with transaction completion. See F. Chiaromonte, G.Dosi (1998).

of this object are variables: the equations to generate their values, conceptually belonging to the first chapter of the manual (algorithms constituting the behavioral repertoire of a trader), have been included here for simplicity. All these equations are experimental, and are left as random updatings in the generic version of the model. The user must define them according to his or her specific needs. An instance is given in Section 2. It is important to remark that this representation is sufficiently flexible and reasonable to permit a wide variety of experiments.

SHEET, Seeker sheet:

f_seek_b(1)	Flag $f_b^\sigma[T]$: 1 = the trader is seeking transactions as a buyer
f_seek_s(1)	Flag $f_s^\sigma[T]$: 1 = the trader is seeking transactions as a seller
p_seek_b(1)	Reference price $p_b^\sigma[T]$: highest price at which the trader is willing to buy as a seeker.
p_seek_s(1)	Reference price $p_s^\sigma[T]$: lowest price at which the trader is willing to sell as a seeker.
[dh1a_seek_b(1), dh1b_seek_b(1)]	Completion scheduling options $D_b^\sigma[T]$: range of lags (in minutes) with which the trader is willing to give cash as a seeker on the buying side.
[dh2a_seek_b(1), dh2b_seek_b(1)]	Completion scheduling options $D_b^\sigma[T]$: range of lags (in minutes) with which the trader is willing to receive asset as a seeker on the buying side.
[dh1a_seek_s(1), dh1b_seek_s(1)]	Completion scheduling options $D_s^\sigma[T]$: range of lags (in minutes) with which the trader is willing to receive cash as a seeker on the selling side.
[dh2a_seek_s(1), dh2b_seek_s(1)]	Completion scheduling options $D_s^\sigma[T]$: range of lags (in minutes) with which the trader is willing to give asset as a seeker on the selling side.

SHEET, Acceptor sheet:

f_acc_b(1)	Flag $f_b^a[T]$: 1 = the trader is accepting transactions as a buyer
f_acc_s(1)	Flag $f_s^a[T]$: 1 = the trader is accepting transactions as a seller
p_acc_b(1)	Reference price $p_b^a[T]$: as above (as acceptor).
p_acc_s(1)	Reference price $p_s^a[T]$: as above (as acceptor).
[dh1a_acc_b(1), dh1b_acc_b(1)]	Completion scheduling options $D_b^a[T]$: as above (as acceptor).
[dh2a_acc_b(1), dh2b_acc_b(1)]	Completion scheduling options $D_b^a[T]$: as above (as acceptor).
[dh1a_acc_s(1), dh1b_acc_s(1)]	Completion scheduling option $D_s^a[T]$: as above (as acceptor).
[dh2a_acc_s(1),	Completion scheduling option $D_s^a[T]$: as above (as acceptor).

dh2b_acc_s(1)]	

The last object descending from the object TRADER is the object MANUAL. Conceptually, the manual contains the whole behavioral repertoire, but in the implementation it handles only the second chapter, i.e. the selection algorithm (the equations are experimental and once again left as random updatings). This object can also be a container for behavioral parameters, as we will see in Section 2. The trader's NOTEPAD has been considered a descendant of MANUAL because the individual information contained in it is used only by the behavioral algorithms.

MANUAL

select(0)	Acceptor to transact with in case the trader is the current seeker.	This equation is experimental and has to be defined by the user.
b_or_s(0)	Selects the "side" to take with the chosen acceptor (if both the acceptor's flags are set to 1): 0 = buy , 1 = sell	This equation is experimental and has to be defined by the user.
notes(0)	Technical variable. Its equation updates the trader's notepad every time his identity number appears on the callers. The value of the variable corresponds to the identity of the last entry in the notepad.	Iterated on all objects ALPHA: if the trader's id[P] = a[P] (of the current object ALPHA)or id=sig[0] • collect the other trader's information • update the object NOTEPAD containing data concerning this trader, if it exists • create a new object NOTEPAD to contain this information if this trader has never been encountered before.
ref_notes(0)	Technical variable. Refreshes the notepads when a trader is expelled from the room: ref_notes=-1 if the notepads have not been refreshed ref_notes = identity of the last trader eliminated from the notepads otherwise.	If the equation is called from num_deaths[0] (at least one trader has been expelled), it scans all traders: • when one with e[0]=1 is found, the object NOTEPAD concerning him is removed (with clean(id)) • ref_notes[0]=expelled agent's id[P] Else (the equation is not called from num_deaths[0]) ref_notes[0]=-1.

Each object NOTEPAD contains a record on another trader that has been encountered. Every time a trader meets another he goes through his records; if there already is an object referring to the agent he just met, the information is updated. If not, a new object is created and the agent's current seeker and acceptor sheets are registered. Records for each concluded transaction are implicitly contained in the transaction database (in the conceptual model they belonged to the notepad).

NOTEPAD, General

enc_time(P)	Minute in which the trader was last encountered.
who(P)	Trader's identity.

NOTEPAD, Trader's seeker sheet (upon meeting):

tf_seek_b(P)	Trader's flag $f_b^{\sigma}[T]$.
tf_seek_s(P)	Trader's flag $f_s^{\sigma}[T]$.
tp_seek_b(P)	Trader's reference price $p_b^{\sigma}[T]$.
tp_seek_s(P)	Trader's reference price $p_s^{\sigma}[T]$.
[tdh1a_seek_b(P), tdh1b_seek_b(P)]	Trader's completion scheduling options $D_b^{\sigma}[T]$ for cash.
[tdh2a_seek_b(P), tdh2b_seek_b(P)]	Trader's completion scheduling options $D_b^{\sigma}[T]$ for asset.
[tdh1a_seek_s(P), tdh1b_seek_s(P)]	Trader's completion scheduling options $D_s^{\sigma}[T]$ for cash.
[tdh2a_seek_s(P), tdh2b_seek_s(P)]	Trader's completion scheduling options $D_s^{\sigma}[T]$ for asset.

NOTEPAD, Trader's acceptor sheet (upon meeting):

tf_acc_b(P)	Trader's flag $f_b^{a}[T]$.
tf_acc_s(P)	Trader's flag $f_s^{a}[T]$.
tp_acc_b(P)	Trader's reference price $p_b^{a}[T]$.
tp_acc_s(P)	Trader's reference price $p_s^{a}[T]$.
[tdh1a_acc_b(P), tdh1b_acc_b(P)]	Trader's completion scheduling options $D_b^{a}[T]$ for cash.
[tdh2a_acc_b(P), tdh2b_acc_b(P)]	Trader's completion scheduling options $D_b^{a}[T]$ for asset.
[tdh1a_acc_s(P), tdh1b_acc_s(P)]	Trader's completion scheduling options $D_s^{a}[T]$ for cash.
[tdh2a_acc_s(P), tdh2b_acc_s(P)]	Trader's completion scheduling options $D_s^{a}[T]$ for asset.

1.4 External Returns

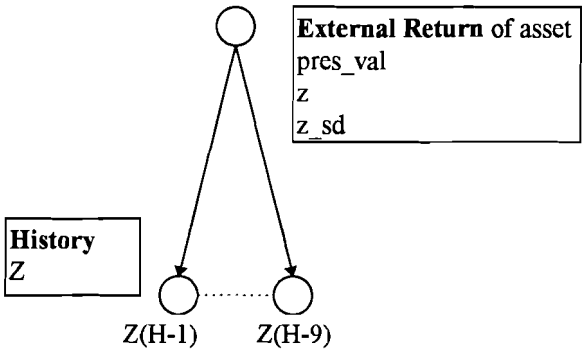

Figure 4: External Returns

This part of the model deals with the value of the asset, as would be determined independently of the trading process (e.g. dividends); that is, outside the room. It is necessary if fundamentalist traders are present in the room (see Section 2). The object EXTERNAL_RET contains the current value of the asset; the descendants HISTORY (which in this version are 9 but can be easily increased by the user[18]) contain the values the asset had in the preceding minutes (starting from the most recent). At the beginning of a new minute, the current value is updated, the old value is assigned to the parameter Z in a new object HISTORY placed in front of the others, and the last object HISTORY is deleted.

The equation for the current value is experimental. In the generic version, we have implemented it as the sum between a systematic component (or underlying value) and a noise which is drawn from a normal distribution with mean 0 and a given standard deviation[19].

EXTERNAL_RET

pres_val(1)	Current asset value (experimental): pres_val[0] = z[P]+N(0,z_sd[P]) (updated at the beginning of every new minute).
z(P)	Experimental parameter expressing the systematic component.
z_sd(P)	Experimental parameter expressing the noise standard deviation.

[18] Notice that there is an issue concerning the relative time scales of trading process and evolution of the external value: if a trader cannot access the present value, the series relative to the last "n" minutes is what he will be able to use to extrapolate it.

[19] Obviously the noise can be eliminated by setting the standard deviation equal to zero. In the following we always parameterize normal distributions through mean and standard deviation (not variance).

HISTORY

Z(P)	Old asset value.

Obviously the parameters in the object EXTERNAL_RET could be turned into variables. In particular, it would be of high experimental interest to specify a dynamics for the systematic component.

2 A Behavioral Framework

In the generic version of the model, all variables (and parameters) concerning traders' behavior have been left constant, or updated at random. This section presents an example of the use of traders' behavior-related equations. The example is meant as a benchmark to test the possibilities of the model. The setting was created to try to include in a common, simple framework, coherent with the characteristics and implementation of the model, a range of different "behavioral types". The equations taken into consideration are those for traders' flags and reference prices. Transactions are restricted to be spot on both sides, taking all completion scheduling options to be constant and equal to {0}. In most cases, there is no distinction between acceptor and seeker flags and reference prices.

Let us consider a price assessment, in the form

$$x_0 + g(x - x_0) = x_0 + \theta_1 \cdot (x - x_0) \cdot Ind(|x - x_0| \geq \theta_2)$$

where θ_1 and θ_2 are parameters embodying, respectively, "sign and size" of the reaction, and a "non-reactivity" radius around 0. This can be used to determine a trader's flags and reference prices, diversifying traders in three respects:

- the values of the parameters θ_1 and θ_2[20], on which one could introduce some form of parametric learning,
- the variables entering the computation, i.e. x, x_0
- the way in which the values of $x_0, g(x - x_0)$ are employed to determine flags and reference prices.

[20] As well as any other parameters used for the determination of flags and reference prices.

The first type of diversification is straightforward. We elaborated on the second and the third. Traders might be diversified by means of the argument variables x, x_0. We consider three cases:

1. Current External Return. "Fundamentalist" traders with (noisy) access to the current value $Z_{(H)}$ of the external return, could simply take $Z_{(H)}$ + e as price assessment. This corresponds to setting $x_0 = x = Z_{(H)}$ + e. Clearly $g(x - x_0) = 0$, and the parameters θ_1 and θ_2 are irrelevant.

2. History of External Returns. "Fundamentalist" traders who can observe only the series of past values $Z_{(H-1)}, Z_{(H-2)}, \ldots Z_{(H-n)}$, could take:

$$x_0 = Z_{(H-1)} \quad \text{and} \quad x = \overline{Z} = \sum_{j=0,1,\ldots} \chi_j Z_{(H-1-j)}.$$

   Here the moving average serves as an extrapolation for the current return value.

3. Prices. "Adaptive" or "Chartist" traders could use the time series of prices (which is public knowledge, up to the very last transaction), and take

$$x_0 = p_{(N)} \quad \text{and} \quad x = p = \sum_{j=0,1,\ldots} \varphi_j p_{(N-j)}.$$

   Here the moving average serves as an extrapolation for the next transaction price. "Noise" traders are a sub-case, taking $p_{(N)}$ as their price assessment, very much like the fundamentalists in (1) with $Z_{(H)}$ + e (all the weights are equal to 0 except $\varphi_0 = 1$). This corresponds to setting $x_0 = x = p_{(N)}$. Again, $g(x - x_0) = 0$ and the parameters θ_1 and θ_2 are irrelevant.

Other examples could be imitators who target another trader, and form their price assessment using what they know about his reference price, say p_{target} as it appears in the notepad (i.e. the target trader's reference price as it was the last time the imitator met him). This corresponds to setting $x_0 = p_{(N)}$ and $x = p_{\text{target}}$. Last, some traders could follow "Sun-Spots"; that is, use a random variable completely unrelated to the system and to the external return, taking for example $x_0 = p_{(N)}$ and $x = W \approx N(x_0, \sigma^2)$.

Once the values of $x_0, g(x - x_0)$ are computed, traders can be further diversified in terms of the way they use them to determine flags and reference prices. Again we considered three cases:

1. **"Take Action"**. The trader's seeker flags are formed on the basis of the value of $g(x-x_0)$ as:

$$g(x-x_0)>0 \Rightarrow f_b^{\sigma}=1, f_s^{\sigma}=0 \text{ (seek to buy)}$$
$$g(x-x_0)=0 \Rightarrow f_b^{\sigma}=f_s^{\sigma}=0 \text{ (hold)}$$
$$g(x-x_0)<0 \Rightarrow f_b^{\sigma}=0, f_s^{\sigma}=1 \text{ if } q\geq 1 \text{ (seek to sell)}$$

Reference prices, on the other hand, do not depend on the price assessment; the trader will pursue the action he selected to the limit of his current cash and asset endowments[21]. If $f_b^{\sigma}=1$, the trader is willing to buy at any price he can afford: $p_b^{\sigma}=m$ (cash endowment); if $f_s^{\sigma}=1$, the trader is willing to sell at any price, no matter how low: $p_s^{\sigma}=0$. "Take Action" embodies a very extreme strategy. Thus, we confine it to seeker flags and reference prices, and we assign it to traders that seek only: $f_b^{\alpha}=f_b^{\alpha}=0$, and the acceptor reference prices are irrelevant. Given the way the selection algorithm is specified (see later), when called upon participating in a trading round, these agents, although ready to buy/sell at very extreme prices, will at least select the most convenient to themselves among the ones proposed by acceptors.

2. **"Form a Spread".** The trader is in principle always available to exchange, within the limits of his asset endowment and of a share of his cash endowment. Moreover, he makes no distinction between seeking and accepting (here and in the following, f 's and p's without superscript refer to both). Flags are not based on the price assessment: $f_b=1$ and $f_s=1$ if $q\geq 1$ (and 0 if the trader has no asset to sell). The values of $x_0, g(x-x_0)$ are used to determine reference prices as to define a typical spread. The calculation also involves a spread parameter ε, and a "caution" parameter $\gamma\in[0,1]$:

$$g(x-x_0)>0 \Rightarrow p_b=\min(x_0,\gamma\cdot m), p_s=x_0+2\varepsilon$$
$$g(x-x_0)=0 \Rightarrow p_b=\min(x_0-\varepsilon,\gamma\cdot m), p_s=x_0+\varepsilon$$
$$g(x-x_0)<0 \Rightarrow p_b=\min(x_0-2\varepsilon,\gamma\cdot m), p_s=x_0$$

In this strategy, the price assessment $x_0+g(x-x_0)$ is considered as relative to some "generic future", or maybe to the close future, but not completely reliable as a point evaluation. The trader does not use $x_0+g(x-x_0)$ in forming the reference prices for the very next transaction; instead, he uses the sign of $g(x-x_0)$ (which he trusts to be reliable) to "orient" upwards or downwards a spread anchored to the center x_0.

---

[21] Since transactions are spot on both sides, he can not postpone payments. The same holds for deliveries, which is why $f_s^{\sigma}=1$ is conditional on $q\geq 1$.

1. **"Form a Divide".** Premises and flags are the same as in (2) above, but the trader forms a unique reference price acting as a divide between buying and selling:

$$p_s = p^* = x_0 + g(x - x_0)$$
$$p_b = \min(p^*, \gamma \cdot m)$$

This can be interpreted as using the price assessment $x_0 + g(x - x_0)$ in the very next transaction. The trader considers it a reliable point evaluation relative to the close future. Thus, any price below p* is seen as a buying opportunity (within the limits of a share of his current cash endowment), and any price above it as a selling opportunity.

On the basis of the above classifications, one can create a whole range of metaphorical traders' types. For the time being, we concentrate on the following subset of the taxonomy described in F. Chiaromonte, G. Dosi (1998):

	"Form a Divide"	**"Form a Spread"**	**"Take Action"**
Current Ext. R.	Strong Fund. (1): uses information on the current external return, is always available to buy and sell (equivalently as seeker and acceptor), and is confident in his assessments.	Strong Fund. (2): uses information on the current external return, is always available to buy and sell (equivalently as seeker and acceptor), and is cautious about his assessments.	
History of Ext. R.'s, (not the current)		Weak Fundamentalist: uses information on the history of the external return, is always available to buy and sell (equivalently as seeker and acceptor), and is cautious about his assessments.	
Prices		Weak (Chartist or) Noise Trader: uses public information on prices, is always available to buy and sell (equivalently as seeker and acceptor), and is cautious about his assessments.	Strong (Chartist or) Noise Trader: uses public information on prices, and seeks only, with a very extreme price strategy.

2.1 Implementation of the Framework

To complete the set of preliminary experiments illustrated in F.Chiaromonte, M.Bertè (1998) a specific version was created, implementing the above sub-taxonomy and deleting a number of unnecessary modules (such as the management of the completion phase; all transaction are spot on both sides).

The following tables list parameters that were added to the model to implement the taxonomy. Within the object GROUP, some parameters were introduced to control how many traders of each type are present in the room. Given the values attributed to these parameters, the objects TRADER are automatically generated in the first time step of the simulation. Leaving generation as part of the simulation initialization by the user would have been impractical because of a technical problem with the user-interface for initialization in LSD. This problem has been solved in the forthcoming version of LSD[22].

GROUP, add:

strong_fund(P)	Number of strong fundamentalists.
weak_fund(P)	Number of weak fundamentalists.
weak_noise(P)	Number of weak (chartists or) noise traders.
strong_noise(P)	Number of strong (chartists or) noise traders.

Another set of parameters was introduced to control the spread-widths used by trader-types, and the generation of trader-types converting sequences (experiments have been performed for various values, see F.Chiaromonte, M.Bertè, 1998). The values attributed to these parameters are automatically assigned to traders of the various types during the first time step.

GROUP, add:

x_par_i(P)	Spread ε_I used by all "informed" traders forming a spread (i.e. strong and weak fundamentalists).
x_par_u(P)	Spread ε_U used by "uninformed" traders forming a spread (i.e. strong and weak - chartists or- noise traders).
param1_i(P)	Mean of the converting sequence for informed traders.
param1_u(P)	Mean of the converting sequence for uninformed traders.[23]

[22] A new paper by M.Valente is forthcoming, see http://www.business.auc.dk/~mv/homelsd.html.

[23] These parameters are assigned to param_1[P] in traders' objects WATCH, see later.

The parameter r[P] in the object TRADER is now employed to distinguish the trader's argument variables x, x_0:

TRADER

r(P)	1 = current external return 2 = history of external return 3 = prices

The parameters involved in the calculation of $g(.)$, as well as other parameters, have been located into the object MANUAL. In particular, type[P] is employed to distinguish among the various ways in which the values of $x_0, g(x - x_0)$ can be used.

MANUAL, add:

gamma(P)	Agent-specific caution parameter γ for traders forming a divide or a spread.
teta1(P)	θ_1.[24]
teta2(P)	θ_2.
type(P)	0 = "Take Action" 1 = "Form a Divide" 2 = "Form a Spread".
x_par(P)	Agent-specific parameter which is assigned the value of x_par_i[P] or x_par_u[P] (depending on the trader).

Additional options have been included in the object EXTERNAL_RET because in the experimental setting strong fundamentalists see the current value of the external return with an error. We distinguish between two cases:

- All strong fundamentalists updating within the same board-clock minute read a common "noisy" value of the external return, i.e. pres_val[0] plus the same draw from a N(0,σ) (σ is assigned internally, through the code).
- Strong fundamentalists updating within the same minute read different "noisy" values, i.e. pres_val[0] plus i.i.d. draws from a N(0,σ) (this is realized by adding the noise independently when the single trader reads the value).

[24] As we shall see when describing the equations for the variables in SHEET, θ_1 and θ_2 have been fixed to zero when r[P]=1. This simplifies the implementation of the rules. Moreover, it guarantees an equal spread on both sides when type[P]=2, which is needed in our experiments (see F. Chiaromonte, M. Bertè, 1998).

In our experiments all we are going to consider is the underlying value z[P] plus these "reading" noises on the side of the traders. Thus, we set z_sd[P]=0, so that pres_val[0]=z[0][25].

EXTERNAL_RET, add:

observed_Z(0)	• if one_draw[P] = 1, then observed_Z[0] = pres_val[0]+ N(0,σ) • else observed_Z[0] = pres_val[0] -the error occurs in the reading of the single trader- (updated at the beginning of every new minute).
one_draw(P)	0 = different "noisy" values 1 = common "noisy" value

2.2 Equations

The equations embodying the behavioral repertoire refer to variables in SHEET (equations generating flags and reference prices) and in MANUAL (selection algorithm). The trader's parameters r[P] and type[P], spanning our taxonomy, are used to distinguish among different updating forms.

SHEET:

f_acc_b[1]	• if type[P]=0, then f_acc_b[0]=0 (take action; only seeker) • else (type[P]=1 or 2), f_acc_b[0]=1 (form a divide or a spread; always available to buy as acceptor
f_acc_s[1]	if h[1]≠h[2] (updating time) or asset[1]≠asset[2] (endowment has changed in the last time step)[26], then: • if type[P]=0 , then f_acc_s[0]=0 (take action; only seeker) • else (type[P]=1 or 2): if asset[0]≥1 then f_acc_s[0]=1 else, f_acc_s[0]=0 (form a divide or spread; always available to sell as acceptor, if he has at least one unit of asset) else f_acc_s[0]=f_acc_s[1] (keep old value of the flag)

[25] One could, though, superimpose the "reading noises" to an intrinsic noise in the mechanism generating pres_val[0], i.e. set z_sd[P]>0.

f_seek_b[1]	if h[1]≠h[2] (updating time) or cash[1]≠cash[2] (endowment has changed in the last time step)[27], then: • if type[P]=0, then: if p_seek_b[0]>0 then f_seek_b[0]=1 else f_seek_b[0]=0 (take action; see equation for p_seek_b[0]) • else (type[P]=1 or 2), then f_seek_b[0]=1 (form a divide or spread; always available to buy as seeker) else f_seek_b[0]=f_seek_b[1] (keep old value of the flag)
f_seek_s[1]	if h[1]≠h[2] (updating time) or asset[1]≠asset[2], then: • if type[P]=0, then: if p_seek_s[0]=0 then f_seek_s[0]=1 else f_seek_s[0]=0 (take action; see equation for p_seek_s[0]) • else (type[P]=1 or 2), then: if asset[0]≥1 then f_seek_s[0]=1 else f_seek_s[0]=0 (form a divide or spread; always available to sell as seeker, if he has at least one unit of asset) else f_seek_s[0]=f_seek_s[1] (keep old value of the flag)
p_acc_b[1]	if h[1]≠h[2] (updating time) or cash[1]≠cash[2], then: • if type[P]=0, then p_acc_b[0]=0 (take action; only seeker --irrelevant) • if type[P]=1, then p_acc_b[0]=min(p_acc_s[0],gamma[P]·cash[0]) (form a divide) • if type[P]=2, then p_acc_b[0]=min(p_acc_s[0] - 2·x_par[P], gamma[P]·cash[0]) (form a spread) else p_acc_b[0]=p_acc_b[1] (keep old value of the reference price)
p_acc_s[1]	if h[1]≠h[2] (updating time) or asset[1]≠asset[2], then: • if type[P]=0 (take action) then p_acc_s[0]=0

[26] Notice that since transactions are spot on both sides, this is equivalent to the trader having been a party of the last transaction; his asset level has increased or decreased, depending on whether he was the buyer or the seller.

[27] Also this is equivalent to having been party to the last transaction, unless it occurred at price 0. We assume that if this is the case the trader will not update.

(take action; only seeker –irrelevant)

- else

A. An intermediate variable y, depending on r[P][28], is calculated:

- if r[P]=1 (current external return), then

  if one_draw=0 (independent draws) then y=pres_val[0]+N(0,)

  else (single draw), y=observed_Z[0] (which already contains the common reading error)

- if r[P]=2 (history of external returns), then

$$y=\frac{1}{9}\sum_{j=0}^{8} Z_{(H-1-j)}[P]-Z_{(H-1)}[P]$$ [29]

  (the subscripts refer to the objects HISTORY, in order)

- if r[P]=3 (prices), then

$$y=\frac{1}{9}\sum_{j=0}^{8} price_{(N-j)}[P]-price\_N[0]$$ [30]

  (the subscripts refer to the objects TRANSACTION, in order;

  price_N[0]=$price_{(N)}$).

B. g(y) = teta1[P]·y·Ind(|y|≥teta2[P]) is calculated

(for r[P]=1, teta1[P]=teta2[P]=0 so that the calculation actually results in g(y)=0)[31].

C. x_0 is calculated, depending on r[P]:

- if r[P]=1, then $x_0=y$
- if r[P]=2, then $x_0=Z_{(H-1)}[P]$
- if r[P]=3, then x_0=price_N[0].

D. The reference price is calculated, depending on type[P]:

- if type[P]=1 (form a divide), then p_acc_s[0]= $x_0+g(y)$
- if type[P]=2 (form a spread), then

[28] This has been implemented as a "case" control structure in the code.

[29] Here the return moving average is based on all retained values (not including the current; thus the 9 historic values we kept) with equal weights.

[30] We have imposed the price moving average to be based on 9 equally weighted elements (including the last transaction price) for simplicity; this could be easily changed. In particular, one might want to use more than 9 price values, as they refer to transactions, while the return values refer to minutes (the two scales might be very different).

[31] Notice that the easiest way to implement traders who use $p_{(M)}$ as their price assessment is, analogously, to set teta1[P]=teta2[P]=0 for r[P]=3 (instead of explicitly introducing weights for the moving average and then set them all to 0, except for j=0). This is what we have done for the experiments described in F. Chiaromonte, M. Bertè (1998).

	if g(y)<0, then p_acc_s[0]= x_0 else if g(y)=0, then p_acc_s[0]= x_0+x_par[P] else if g(y)>0 then p_acc_s[0]= x_0+2·x_par[P]. else p_acc_s[0]=p_acc_s[1] (keep old value of the reference price)
p_seek_b[1]	if h[1]≠h[2] (updating time) or cash[1]≠cash[2], then: • if type[P]=0, then: if p_seek_s[0]=-10 then p_seek_b[0]=cash[0] else p_seek_s[0]=0 (take action; the trader buys at any price he can afford) • else (if type[P]=1 or 2), p_seek_b[0]=p_acc_b[0] (form a divide or spread; same prices are used for seeking and accepting) else p_seek_b[0]=p_seek_b[1] (keep old value of the reference price)
p_seek_s[1]	• if type[P]=1 or 2, then p_seek_s[0]=p_acc_s[0][32] • else (type[P]=0; take action)[33] • y, g(y) and x_0 are computed as in the equation for p_acc_s[0], according to r[P], and the reference price is given by: • if g(y)<0 and assets[0]≥1, then p_seek_s[0]=0 (the trader sells at any price) • if g(y)=0 then p_seek_s[0]=-1 (through p_seek_b[0] and flags, he holds) • if g(y)>o, then p_seek_s[0]=-10 (through p_seek_b[0] and flags, he buys at any price he can afford)

MANUAL:

select(0)	The seeker chooses the acceptor whose reference price is most convenient: if id[P]=sig[P] (the trader is the seeker) iterated on all objects ALPHA (for all possible acceptors): • (we use intermediate variables y_1, y_2; side, identity. y* indicates the current y's maximum through the iteration and is initialized at -1) • the notepad is checked for the reference prices of the trader with identity a[P] (the traders have already exchanged information)

[32] Even though the code repeats some calculations; this was used as a check.

[33] For type[P]=0, p_acc_s[0] has not been really calculated, but only set as an irrelevant default.

	• if p_seek_b[0]-tp_acc_s[P]>0 and tf_acc_s[P]=1 (the transaction is feasible on buying side), then y_1=|p_seek_b[0]-(lambda[P]·p_seek_b[0] +(1-lambda[P])·tp_acc_s[0])|, else y_1=-2 • if p_seek_s[0]-tp_acc_b[P]<0 and tf_acc_b[P]=1 (the transaction is feasible on selling side), then y_2= |p_seek_s[0]-(lambda[P]·p_seek_s[0] +(1-lambda[P])·tp_acc_b[0])|, else y_2=-2 • if max{y_1,y_2}>y*, then y*= max{y_1,y_2}, side=the flag determining the side, identity=a[P] select[0]=identity b_or_s[0]=side
b_or_s(0)	Calculated in the equation for select[0].

Another equation becomes extremely important in this specific implementation of the model; namely, that of num_births() (in object GROUP). The reason for this is that the initialization of the various types of agents has to be fulfilled automatically assigning all the parameters, prices and initial data.

num_births(1)	the total number of traders is calculated (adding the different types): sum=strong_fund[P]+weak_fund[P]+strong_noise[P]+weak_noise[P] for i=1,...,sum (all traders in the market): • an object TRADER is created[34] • id[P] is assigned (id[P]1=i) • assets[0]=assets[0] of first trader (as initialized by the user) • cash[0]=cash[0] of first trader (as initialized by the user)[35] • B[0]=max_B[P] • r [P] is assigned: if i<strong_fund[P], then r[P]=1 else if i<strong_fund[P]+weak_fund[P], then r[P]=2 else r[P]=3 • watch parameters and variables are initialized: if i<strong_fund[P]+weak_fund[P] if one_draw[P]=1[36] (single draw)

[34] The one object TRADER allocated when specifying the structure hosts the first trader.

[35] All traders thus start with equal levels of cash and asset.

[36] The distribution of the converting sequence will be a Poisson if the traders read the same "noisy" values of the external return and constant otherwise (one_draw[P]=0). If traders were synchronous and read the

	distr_v[P]=1 (Poisson) v[0]=draw from a Poisson(param1_i[P]) else (iid draws) distr_v[P]=5 (Constant value) v[0]=param1_i[P] else (uninformed traders) distr_v[P]=1 (Poisson) v[0]=draw from a Poisson(param1_u[P]) h[0]=1 sum_v[0]=v[0] param2[P]=-1 (not needed, both for Poisson and constant value case) if i<strong_fund[P]+weak_fund[P], then param1[P]=param1_i[P] else param1[P]=param1_u[P] (uninformed traders) (these will be used for all draws after the first) • reference prices are initialized: this is experimental; in the experiments completed the agents have been separated into two groups with different centers • type[P], teta1[P] teta2[P] and x_par[P] are assigned, with the same method as for r[P]

same noisy value, their price assessments would converge right away (see F. Chiaromonte, M. Bertè, 1998).

APPENDIX A: Pseudo-Simultaneous Procedure for Completing Transactions

The completion of transactions is a very important issue because a failure in completing a transaction within the scheduled time results in loosing one or more bonus minutes and eventually in being expelled from the room. For this reason the model was conceived in a way that prevents failures dependent from the order in which transactions which are due simultaneously are completed. A transaction is not completed only if the trader involved actually does not possess the endowments necessary to honor it.

This view of the completion phase implies the introduction of a pseudo-simultaneous procedure, which virtually enables all the transactions involved to be completed at the same time. The procedure implemented has been chosen evaluating both effectiveness and computational efficiency, which becomes a critical point when solving a problem that is by definition computationally heavy. The procedure is described thoroughly in the next sections, giving both analytical justifications and implementation details.

1. Description of the pseudo-simultaneous procedure

The symbols we will introduce in this section are completely unrelated to those used elsewhere in the paper.

Define:

i=1...k	the active transactions (transactions in completion phase) on the cash side, right after a new transaction conclusion,
a_i	the quantity of cash involved in transaction i,
$_i$	the trader who has to deliver the cash in transaction i,
$_i$	the trader who has to receive the cash in the transaction i,
j=1...n	all the traders present in the market,
b_j	endowment of agent j.

Consider the algorithm:

l=0

for j=1...n (all traders)

for i=1...k (all active transactions)

$$\alpha_{ji} = \begin{cases} 0 \text{ if } j \neq \theta_i \text{ and } j \neq \psi_i \\ a_i \text{ if } j = \theta_i \\ -a_i \text{ if and } j \neq \psi_i \end{cases}$$

if $\alpha_{ji} = 0 \ \forall i = 1...k$, then j= j+1

else

l=l+1

Id_l=j

$\overline{\alpha_l} = \left(\alpha_{j1}...\alpha_{jk}\right)^T$

$\beta_l = b_j$

j=j+1

(1)

This algorithm produces:

l=number of traders involved,

a matrix $A = \left(\overline{\alpha_1}..\overline{\alpha_l}\right)$ of dimension (k×l),

a vector $\overline{\beta} = (\beta_1...\beta_l)^T$ of dimension (l×1),

an "identities" vector $\overline{Id} = (Id_1...Id_l)^T$.

Consider now the optimization problem:

$$\max \overline{x}^T \overline{1}$$

$$\text{with } A^T \overline{x} \leq \overline{\beta} \quad i.e. \quad \begin{pmatrix} \overline{\alpha}_1^{\mathrm{T}} \overline{x} \leq \beta_1 \\ \overline{\alpha}_2^{\mathrm{T}} \overline{x} \leq \beta_2 \\ \vdots \\ \overline{\alpha}_l^{\mathrm{T}} \overline{x} \leq \beta_l \end{pmatrix} \qquad (2)$$

$$\text{and } \overline{x} \in \{0,1\}^k.$$

This corresponds to maximizing the number of active transactions that are completed under the constraints given by agents' endowments.[37] If there is more than one solution $\bar{x}^*$, clearly any one can be chosen randomly, since no other criteria is considered.

A transaction i with $x_i^* = 0$ is a transaction that has failed. The number of failures of agent j can be calculated by finding l so that Id_l=j and counting the number of transactions i with $_{li}$>0 and $x_i^* = 0$ (j is the failing agent in them).

The same procedure can be applied to the asset side, where a_i will always be equal to 1.

Claim:

using $\bar{x}^*$, no "solvent" agent is failing (no agent is failing a transaction only due to the order in which "active" transactions are completed).

Proof:

$j \leftrightarrow l$ (via $Id_l = j$) is solvent *iff* $\bar{\alpha}_l^T \bar{1} \le \beta_l$ so we need to show that if $\bar{\alpha}_l^T \bar{1} \le \beta_l$, there is no i s.t. $\alpha_{li} > 0$ and $x_i^* = 0$.

By contradiction, suppose $\exists i$ st $a_{li} > 0$ and $x_i^* = 0$; if we define x' as $x^* + (0...1...0)^T$ where the 1 is in position i (this is equal to $\bar{x}^*$ with position i switched to 1). Clearly $x'^T \bar{1} > x^{*T} \bar{1}$ $\left(x'^T \bar{1} = x^{*T} \bar{1} + 1\right)$, and $x' \in \{0,1\}^k$.

Moreover, $\alpha_t^T x' = \alpha_t^T x^* + \alpha_{ti} \le \alpha_t^T x^* \le \beta_t$ whenever t≠l (because in these cases $\alpha_{ti} = 0$ or $\alpha_{ti} = -a_i$), and $\alpha_t^T x' = \alpha_l^T x' \le \alpha_l^T \bar{1} \le \beta_l$ (the constraint is guaranteed by the solvency of j↔l).

But then $\bar{x}^*$ would not be a solution of (2) because x' would.

The optimization problem described above can clearly be solved with linear integer programming. Since the most known integer programming methods, such as "branch and bound" or "Gomory cuts" might give efficiency and convergence problems in some instances, the particularly simple objective function has been exploited to devise an ad hoc method, which is summarized below.

A simplex method is applied to the problem:

---

[37] Note that no priority is given to transactions involving higher amounts or any other criterion.

$$\begin{aligned} &\max y^T \bar{1} \\ &A^T y \le \beta, \\ &y \le \bar{1}, \\ &y \ge \bar{0}. \end{aligned} \tag{3}$$

which is equal to the original problem without the integer constraints. Obviously if the solution y^* is such that $y^* \in \{0,1\}^k$ then the optimum of the original problem has been found; if not, one certainly has $x^{*T} \bar{1} \le y^{*T} \bar{1}$.

At this point, since the objective function is merely a maximization of the number of ones in a binary vector, the set of possible optimums is $[0, y^{*T} \bar{1}] \cap N$ i.e. all the integer numbers ranging from the optimum given by the simplex method rounded to the lowest integer, down to zero. Through a method which has resulted quite efficient, for each integer number t contained in this set (starting from the highest) solutions with all the possible combinations of t ones and k-t zeroes are substituted in the constraints. As soon as one constraint is not respected the solution is discarded.

2. Implementation

A new class, optim, has been added to the LSD files, for the management of the pseudo-simultaneous procedure described above. The class is defined in file decl.h and its functions are described in the new file simplex.cpp. The class contains the method for performing the simplex algorithm, the procedure to find the final integer solution (ps()), functions related to memory management[38] and variables for the matrix on which the simplex is performed and its dimensions m and n.

The new class is used by equations pseudo_sim_a[0] and pseudo_sim_c[0], which complete the transactions, the first on the asset side and the second on the cash side. These eqations are described in the next paragraph.

2.1 Equations pseudo_sim_a[0] and pseudo_sim_c[0]

The two equations work exactly in the same way and therefore the description is left general.

Before constructing the matrix to solve problem (3) its dimensions have to be defined for technical reasons. This is what the first part of the equation is devoted to.

---

[38] Taken from W. H. Press et al., *Numerical Recipes in C Second Edition*, Cambridge University Press, Cambridge, 1992.

First of all the equation goes through all the objects TRANSACTION to detect if there are any active ones on the appropriate side. If there are they are assigned their position in the matrix which will be constructed:

```
v[1]=p->cal("N",0);
v[2]=p->cal("N",1);
v[7]=p->cal("H",0);
v[8]=0;

for (cur=p->son;cur!=NULL;cur=cur->next) //checks if there are transactions
{                                        //in completion phase
  v[3]=cur->cal("c2",0);
  v[4]=cur->cal("time",0);
  v[5]=cur->cal("dh2",0);
  v[6]=v[5]+v[4];
  if ((v[3]==0) && ((v[6])<=v[7]))
  {
    cur->write("tran_num",v[8]+1,t);     //if there are their position in the matrix is
    v[8]++;                              //assigned
  }
}
```

If there are transactions in completion and a new negotiation occurred, the transactions are sorted according to their value of tran_num[P]. Each trader is then checked, to see if he is involved in any of the transactions in completion phase. If he is, his position in the matrix is assigned:

```
if ((v[8]!=0) && (v[1]!=v[2]))   //if there are transactions in completion
{                                //and a new negotiation occurred
  (p->son)->sort_asc(p,"tran_num");
  cur_v=p->search_var(p,"num_tr");
  cur1=cur_v->up;                //points to object GROUP
   v[13]=0;
   for (cur=cur1->son;cur!=NULL;cur=cur->next) //for each agent
   {
     v[9]=1;
     v[12]=cur->cal("id",0); //agent's identity
```

```
for(cur2=p->son;cur2!=NULL && v[9]<=v[8];cur2=cur2->next) //for each active transaction
 {
     v[10]=cur2->cal("buy",0);
     v[11]=cur2->cal("sell",0);
     v[14]=cur->cal("trad_num",0);
     if (((v[10]==v[12])||(v[11]==v[12])) && (v[14]==0)) //if agent is involved in the transaction
     {                                                  //(and it is the first one in which he is involved)
      cur->write("trad_num",v[13]+1,t);          //he is assigned a position in the matrix
      v[13]++;
     }
     v[9]++;
   }
 }
```

The variables of an instance of the object optim (integer_sol) are now initialized. The matrix for solving the maximization problem (3) is defined in restricted normal form as follows:

integer_sol.n+l+k

0	1	⋯	1	1	0	⋯	⋯	⋯	⋯	⋯	⋯	0	← objective function
b_1	$-a_{11}$	$-a_{12}$	⋯	$-a_{1k}$	-1	0	⋯	⋯	⋯	⋯	0	0	
b_2	$-a_{21}$	$-a_{22}$	⋯	$-a_{2k}$	0	−1	⋯	⋯	⋯	⋯	0	0	← $b_i, -A_i$, slack variable
⋮	⋮		⋱	⋮	⋮	⋮	⋱				⋮	⋮	
b_l	$-a_{l1}$	$-a_{l2}$	⋯	$-a_{lk}$	⋮	⋮		⋱			⋮	⋮	
1	−1	0	⋯	0	⋮	⋮			⋱		⋮	⋮	
1	0	−1	⋯	0	⋮	⋮				⋱	⋮	⋮	
⋮	⋮		⋱	⋮	0	0	⋯	⋯	⋯	⋯	−1	0	← constraints $x_j \le 1$, slack variable
1	0	⋯	0	−1	0	0	⋯	⋯	⋯	⋯	0	−1	
0	0	⋯	⋯	⋯	⋯	⋯	⋯	⋯	⋯	⋯	⋯	0	← used in simplex algorithm

Since all the constraints are inequalities, one slack variable for each has to be considered. This is done automatically by the simplex algorithm; therefore the matrix initialized has dimensions (l+k+2)×(k+1) which are assigned to integer.sol.m (number of constraints for the simplex+2) and integer.sol.n (variables in the simplex, which correspond to the number of transactions to be completed, and one column for). The values a_{ij} are assigned as described in (1).

```
for (cur=cur1->son;cur!=NULL;cur=cur->next)
```

```
{
 v[17]=2;
 v[9]=cur->cal("trad_num",0);
 if (v[9]!=0)
 {
  v[12]=cur->cal("id",0);
  v[14]=cur->cal("assets",0);
  for(cur2=p->son;cur2!=NULL && v[17]<=v[8]+1;cur2=cur2->next)
  {
    v[10]=cur2->cal("buy",0);
    v[11]=cur2->cal("sell",0);
    if (v[10]==v[12])
       integer_sol.a[(int)(v[9]+1)][(int)(v[17])]=1; //receives asset
    else if (v[11]==v[12])
       integer_sol.a[(int)(v[9]+1)][(int)(v[17])]=-1;//gives asset
     v[17]++;
   }
   integer_sol.a[(int)(v[9]+1)][1]=v[14];//budget constraint (b)
  }
 }
```

The integer solution is then calculated (and assigned to vector integer_sol.ans) by the function integer_sol.ps() which first solves the problem (3) with a simplex method and then solves problem (2) as described in the previous section. At this point all the possible negotiations are completed and any bonus points needed are assigned.

```
for (cur=p->son;(cur!=NULL && v[25]!=100);cur=cur->next)
  {
   if (integer_sol.ans[(int)v[25]]==1) //completes all possible negotiations
   {
    cur->write("c2",1,t);
    v[26]=cur->cal("buy",0);
    cur1=p->search_var_cond("id",v[26],0);
    v[27]=cur1->cal("assets",0);
    v[27]++;
    cur1->write("assets",v[27],t);
    v[28]=cur->cal("sell",0);
```

```
    cur1=p->search_var_cond("id",v[28],0);
    v[27]=cur1->cal("assets",0);
    v[27]--;
    cur1->write("assets",v[27],t);
  }
  else if (v[7]!=v[0])   //if the minute is finished, assign bonus points to who needs them
  {
    v[28]=cur->cal("sell",0);
    cur1=p->search_var_cond("id",v[28],0);
    v[12]=cur1->cal("B",0);
    cur1->write("B",v[12]-1,t);
    if (v[12]>0)
    {
        cur_v=p->search_var(p,"num_bonus"); //stastistical variable num_bonus is updated
        cur2=cur_v->up;
        v[15]=cur2->cal("num_bonus",0);
        cur2->write("num_bonus",v[15]+1,t);
     }
  }
  if ((cur->next)!=NULL)
    v[25]=(cur->next)->cal("tran_num",0);
 }
```

The function returns the value of integer_sol.n.

Appendix B: Saving Options

The implementation of Financial "Toy-Room" associates the LSD time step to a round of the model. Since, though, some features of the model are governed by different timings, the implementation offers the possibility of using different saving options depending on what the user is studying. This is done by the functions special_sv() and sp_sv() which are included in LSD class OBJECT and are defined in the fun_*.cpp file. The implementation also uses a lot of indirect updating, i.e. variables and parameters are frequently updated from other variables, often even more than once during the same time step; as for example for the variables cash and assets. This is the reason why instead of saving variables when these are calculated, it was found useful to modify LSD in order to calculate all the variables first and then save them; obviously to the expense of time.

1. Function special_sv()

This function is called by main_ux or main_win (in the LSD src directory), associated with the first object of the model. The function reads parameter sv_opt[P], fixed by the user, and goes through every variable of the object. Depending on the value of sv_opt[P], different criteria (apart from checking if the user has set the saving flag on) are used to determine if the variable needs to be saved; if it does its value is written on the appropriate file. This process is repeated, with an iterative process, for all the other objects in the model.

2. Function sp_sv()

This function manages the carriage return in the result file. Every time the function special_sv() has completed its search sp_sv() checks to see if to insert a carriage return. Depending on the value of sv_opt[P] the carriage return is inserted at every time step, at every N or at every H.

References

Bertè Mariele, Un Mercato Artificiale per la Valutazione di Strumenti Finanziari Derivati, Università degli Studi di Milano, Milan, 1997.

Chiaromonte Francesca, Giovanni Dosi, Modeling a Decentralized Asset Market: Financial "Toy-Room", IIASA Interim Report,???, Laxenburg, 1998.

Chiaromonte Francesca, Mariele Bertè, Some Preliminary Experiments with the Financial "Toy-Room", IIASA Interim Report,???, Laxenburg, 1998.

Kernighan Brian W., Dennis M. Ritchie, The C Programming Language, Second Edition, Prentice-Hall Inc., New Jersey, 1989.

Press William P. et al., Numerical Recipes in C Second Edition, Cambridge University Press, Cambridge, 1992.

Schildt Herbert, Guida al Linguaggio C++, McGraw-Hill, Milan, 1996.

Valente Marco, Laboratory for Simulation Development User Manual, IIASA Interim Report IR-97-020/May, Laxenburg, 1997.

(also http://www.business.auc.dk/~mv/homelsd.html for Programmer Manual and new version of LSD)