Project no.
035086
Project acronym
EURACE
Project title
An Agent-Based software platform for European economic policy design with heterogeneous interacting agents: new insights from a bottom up approach to economic modelling and simulation

Instrument STREP

Thematic Priority IST FET PROACTIVE INITIATIVE "SIMULATING EMERGENT PROPERTIES IN COMPLEX SYSTEMS"

Deliverable reference number and title

D3.1: Definition of formal, logical and statistical methods to identify the emergence of stable global regularities from the bottom up

Due date of deliverable:
31/08/2008
Actual submission date:
30/09/2008

Start date of project: September 1st 2006 Duration: 36 months

Organisation name of lead contractor for this deliverable
Université de la Mediterranée - GREQAM

Revision 1

Project co-funded by the European Commission within the Sixth Framework Programme (2002-2006)		
Dissemination Level		
PU	Public	X
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	

Contents

Executive summary

Workpackage 3 ‘Economy as a complex system: emergence of aggregate outcomes with heterogeneous and interacting agents’ originally had as objective to develop a class of models in which aggregate regularities emerge from the bottom-up. This class of models will be used to reproduce and explain the empirical evidence and to discover the micro-mechanisms that govern the interaction among economic agents.’ It foresaw the following deliverable: **”D3.1 Definition of formal, logical and statistical methods to identify the emergence of stable global regularities from the bottom-up. Month 24, UPM.”**

Very rapidly however, preliminary investigations conducted by GREQAM showed that the original plan was unrealistic and did not correspond to the most urgent project needs. The WP content was therefore revised, shortened, and its responsibility was attributed to GREQAM. The revised deliverable consists of three chapters:

Chapter 1 discusses the problem of data management. The discussion is written with specific reference to the problems encountered within EURACE, but can be applied to any large-scale agent based model that produces output in the form of time series of panel data sets.

Chapter 2 is a proposal on the inference methodology to be used, and describes parameter sampling techniques.

Chapter 3 discusses the problem of aggregation from a theoretical viewpoint and the related problem of the definition of macrovariables.

Chapter 1

Data management in EURACE

1.1 Summary

The topics addressed in this paper are:

- Data recording needs.
  What data should be recorded and stored for analysis **after** a simulation run, and what data should be available online **during** the simulation? What statistical information should be generated by the (central or local) Eurostat agent(s)? Should Eurostat produce *forecasts* of certain key economic indicators?

- What kind of macro-information should be made available to the agents for their decision making (prices, quality, skill levels, inflation rates, unemployment rates at the regional and national level e.g.)?
  The economist units should provide a list of the macro indicators relevant for their respective modules.

- What do we need to store and what not at each iteration (or at a given frequency)?

- How will the Eurostat agent collect data?

- How should the Simulation GUI be organized?

1.2 Data collection: time versus storage space

Already in running some very rudimentary EURACE simulations we encountered some limitations in storage capacities. A simulation with more than $10,000$ households in the Bielefeld model had to be aborted because the storage of all the memory variables quickly filled up harddisk space. Additional harddisk space is not a viable solution to this problem. The reduction in storage needs expected by moving from XML tags to database file formats will not be sufficient, by far, to solve this problem.

The economist units need to reflect on what to store and what not to store. We give here some opening considerations on that topic.

1.2.1 Online computation versus storage and post-processing of data

A central issue is the trade-off between online versus offline computation.

- With online computation all data processing occurs online. This costs CPU time but saves external storage capacity and I/O time.
- With offline computation all data must be stored and post-processed. This saves CPU time, but is costly in terms of I/O time.
- We should test which option is the more appropriate for our needs.
- Should the HPCs that are used for running the simulations also handle the data processing, or should this be done on smaller machines?

The answers may depend partly on the question: Will the processing of data from full-size simulations be done on the cluster itself, or on smaller machines? This leads to questions on data migration. All in all, this leads to the following set of questions:

- What data should be available for analysis **after** a simulation run?
- What data should be available online **during** a simulation?
- What statistical information should the (central or local) Eurostat agent(s) generate? Do we want Eurostat to produce *forecasts* of key economic indicators?

1.2.2 The historical record: data recording

In essence, the data we produce by running an agent-based simulation consists of panel data sets. Is it useful to store all these panel data sets, or is it more appropriate to summarize online the data into statistics and to store the statistics instead? The time-varying statistical distributions obtained with the latter approach may be sufficient for any desired comparison to the empirical stylized facts.

Currently, the entire memory content of each agent is stored in the iteration file. The resulting files are huge. To compute the distribution of a specific variable in a simulation, we need to post-process the data by perusing all these iteration files. This means that we have to use data-mining techniques to efficiently go through the files.

An alternative would be to record only what is necessary for the planned analyses. (These analyses can be different from simulation to simulation, so that the user will have to decide what data to record). Since the memory variables of the agents are overwritten at each iteration, this approach implies that we should compute the needed statistics online. Thus, the data reduction allowed by the approach comes at the cost of additional online data collection and computation. The potential bottleneck here is the need to send messages *across* the nodes of the cluster to the Eurostat agent, who will aggregate them.

The best model we have to organize this data aggregation efficiently is the real world, where the collection of statistics is organized in a hierarchical way. Data is first collected at the individual level by regional branches of the national statistical office. It is then sent to the national office that processes it. These national data sets are finally collected by Eurostat to produce the supranational data and the EU accounts.

Another possibility to restrict the historical record and to economize on storage space is to store a time trace of a complete panel data set for a random sample of agents drawn from the population. With that option, we still have to decide whether to store all memory variables or just a subset of particular interest. This brings us to the following considerations:

- Taking random samples means that we no longer have perfect data, that is, that we forego one of the advantages of agent-based simulations.
- Saving all or part of the memory variables for some or all agents at each iteration (or with a sampling frequency) will require tagging the memory variables as storable/non-storable.
- The option of restarting a simulation from a user-defined restore point requires a complete dump of the current memory (the memory tagged as non-storable, i.e., temporary data, must be saved nonetheless in order to be able to continue from the restore point).
- It might then also be a nice feature to run coarse-grained 'light' simulations where most agent memory is not saved, say only the memory of the Eurostat agents holding the statistics and indicators, and afterward to have the possibility to run more 'heavy' simulations to get more detailed information.
- Interesting uses of restore points can be:
  - the possibility to continue from a restore point with a finer level of detail.
  - the possibility to continue from a restore point with a coarser level of detail.
  - the possibility to continue from a restore point with more iterations.

The above discussion suggests that the Simulation GUI should provide an option for selecting for each agent type the memory variables that should be saved, with shortcuts for 'select all' or 'select none'.

1.2.3 Statistical information available online

This section is concerned with the computation of the macroeconomic variables taken into account by the agents.

Suppose we want to construct the unemployment rate at the regional and national levels. This involves in each iteration:

- Each worker entering/leaving the regions informs the regional Eurostat office of his move. The local office, that knows the initial population, computes the current one.
- Each firm agent sends a message with its current number of employees to the regional Eurostat agent.
- The regional Eurostat agent computes the regional unemployment rate.
- Once this computation is complete, the source data is deleted from memory.
- Each regional Eurostat agent sends a message to the centralized Eurostat agent with the regional number of employed and unemployed workers.
- The central Eurostat agent computes the national and supranational (EU-wide) unemployment rates. The source data is deleted afterwards.

To allow individual agents to retrieve the regional or national unemployment rates, we suggest the following:

- The central Eurostat agent sends a general message (a broadcast) to the local message boards with (all) national unemployment rates as content. Alternatively, the message can be focussed to each national level and contain only the national unemployment rate for each particular nation (i.e. multiple national messages).
- Each local Eurostat agent sends a general message (a broadcast) to the local message board with the local (regional) unemployment rate as content.
- According to its needs each agent can read the messages, unpack their content and use it.

The unemployment rate is a scalar statistic. Apart from scalars, Eurostat may also provide distributions. For example, the regional firm size distributions can be computed from the data on the number of employees of regional firms. These regional distributions can be combined by the central agent into national and EU-wide firm size distributions.

This means we should allow agents to send messages containing distributions (frequencies for each bin) as the message content. Note that this does not only apply to statistics that need to be available to the agents online, but also to the statistics that should be available offline after a simulation.

The economist units should provide a list of the macro indicators relevant for their respective modules (prices, quality levels, skill levels, inflation rates and unemployment rates at the regional and national level, etc.) that should be compiled by the Eurostat agents.

1.2.4 What to store and what not to store?

We propose to offer the user five options:

(1): Storing all memory variables for all agents - large storage requirement, offline data processing.

(2): Storing a user-determined subset of memory variables for all agents - medium storage requirement, offline data processing.

(3): Storing all memory variables for a user-determined sample of agents - medium storage requirement, offline data processing.

(4): Storing a user-determined subset of memory variables for a user-determined sample of agents - medium storage requirement, offline data processing.

(5): Storing only a pre-defined statistical summary of the data - small storage requirement, online data processing.

Option (1) reflects the *complete detail* option and option (5) is a *light mode* option. The economic statistics will always have to be computed online, but statistical distributions can be computed online or offline.

Another economizing feature could come from not saving the output at every iteration, but only at a certain frequency, i.e. to only store quarterly or yearly data.

1.2.5 Implementation of the Eurostat agent

The main task of the Eurostat agent is the collection and processing of microeconomic data and the redistribution of macroeconomic statistical information to the individual agents. For the implementation of the Eurostat agent we propose two methods:

a) There is only a single Eurostat agent and all data is collected centrally and then processed.
The disadvantage of this approach is that there is much communication across nodes when individual agents need to submit their data. Bottlenecks are likely to occur here.

b) There are multiple branches of the statistical office that produce statistics.
Local Eurostat agents collect data and produce statistics at the regional level. A central Eurostat agent receives statistical reports from the regional Eurostat agents and combines these into national and supranational statistics.
The advantage of this method is that it allows for efficient distributed data collection and data processing. The disadvantage is that there are multiple Eurostat agents.

Figure 1.1 illustrates the two designs for the Eurostat agent.

Another question we should answer is: Do we want the Eurostat agent(s) to produce forecasts of some key economic indicators (e.g. inflation and unemployment rate)? This could be done using simple backward-looking algorithms that take into account the values from previous periods (later on, this could be extended with more econometric techniques, to respond to a suggestion made by Sargent that we should build models with little econometricians inside).

Assume that we adopt the decentralized approach. That is, there are as many Eurostat agents as there are regions, plus one extra for the centralized, supranational level. Then:

We propose to store the indicators as memory variables in the memory of the Eurostat agent(s) as a structure:

```
struct indicator_list
{
   <indicator_name_1>
   ...
   <indicator_name_n>
}
```

Each regional Eurostat agent can have a memory variable to store the indicator list for its own region. The central Eurostat agent would need a dynamic array of indicator lists since the number of regions can vary from simulation to simulation:

```
dynamic_array_indicator_list regional_indicator_lists
```

An indicator with `<indicator_name>`, for a particular region `region_id` can then be accessed as follows:

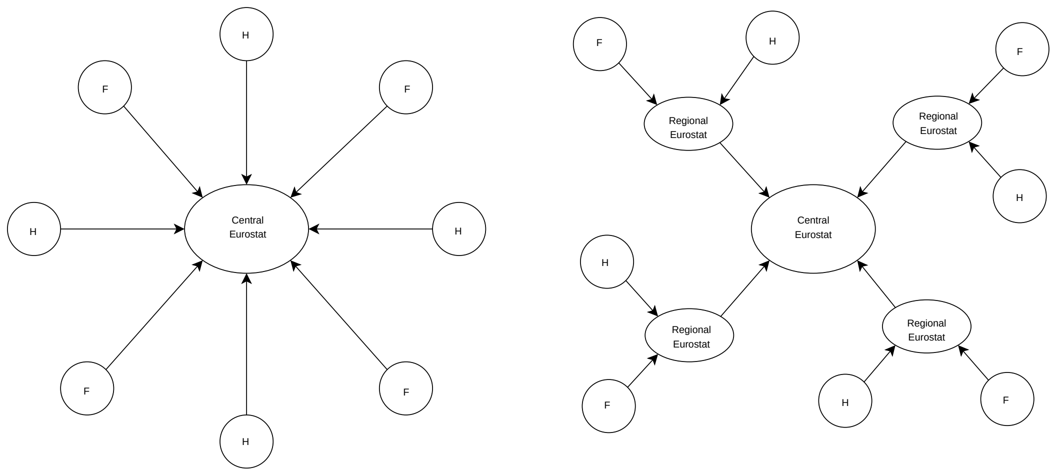


Figure 1.1: Design of the data collection by the Eurostat agent. Left: centralized design. Right: hierarchical design with regional statistical offices.

```
regional_indicator_lists -> array[region_id]-> <indicator_name>
```

To construct national and supranational indicators from the regional ones, the central Eurostat agent would have to perform some computations on its memory variables. The indicators at the national level are again to be collected into a dynamic array of indicator lists, while the supranational indicators are just a single indicator list:

```
dynamic_array_indicator_list national_indicator_lists
indicator_list supranational_indicator_list
```

1.3 Requirements for the GUI

In order to have reasonable confidence in the results from the model and in our policy recommendations, to display as much as possible the 'typical' average behavior of the model, and to possibly discover rare events, we will need to conduct, compare, and aggregate as many simulations as feasible. In this section, we introduce some definitions and suggest possible ways of intelligently conducting these simulations.

1.3.1 Running scenarios

Basic definition

A ***specification*** of the EURACE model is a model specification where all elements are fixed except possibly:

- the initial conditions
- the random seeds for the exogenous variables
- the number of agents
- the time span of the simulation
- the policy variables and policy functions (e.g., the tax rate, the parameters of the Taylor rule, or more generally any rule followed by the Central Bank)

In the above, we focus on changes in policy variables. However, all the suggestions below can be extended in a straightforward manner to changes in other elements of the model, i.e. in our terminology, to changes in the specification of the model.

For a given *specification* of the EURACE model:

- A ***scenario*** is defined by fixing the number of agents, and the policy variables and policy functions.
- Given a scenario, a ***run*** is defined by fixing the initial conditions, and the time span of the simulation.

- Given a scenario, a ***scenario batch*** is defined by running an ensemble of runs using different random seeds for the exogenous variables.

A ***scenario experiment*** is defined by running a set of scenario batches for different model specifications. We must decide whether to choose the initial conditions in a more or less arbitrary fashion or to use the values obtained at the end of a run that generated reasonable results.

Exploiting the synthetic data

A scenario batch will produce several time paths for each variable. How are we going to exploit them? A simple way would be to average the values across all runs. But averaging destroys much of the data and in particular the variability across time paths. Computing, in addition to the mean, the max and min values over all runs in each period is not necessarily meaningful, since one run may provide the max in one period, and the min in the next period. Likewise, a single time path (say, some medium path if one can be identified) may not be a good characterization of the ensemble of runs.

It would be nice if we were able to identify some invariant properties of the generated time paths, such as autocorrelation, instantaneous standard deviation, mean, etc. But we can not assume that we will find them in the data. Another possibility would be to build and estimate econometric models of EURACE using as well the data from single scenario simulations as aggregated data over several scenario simulations, and to compare the results.

Changing policy variables and functions at an intermediate time

Within a scenario simulation (run, batch, experiment), the policy variables and functions need not to be constant over the whole time horizon. A useful way to trace the impact of policy changes is as follows:

- Start with a particular setting of the policy parameters.
- Run the simulation for some predefined number of iterations.
- Keep the current values of the memory but change the value of a parameter.
- Resume the simulation.

The data storage requirement for this method leads to the following sequence:

- Fix the parameters and run the simulation up to time T.
- Save the memory values of all agents to harddisk, creating a restore point.
- Make a copy of the restore point on the harddisk.
- In the copied version, change the value of one policy parameter. Typically, this parameter will be a memory variable of a policy agent or a memory variable for all agents who use it for their decisions.

- Start a new simulation, using the stored memory values as initial conditions. This requires loading the memory from hard disk.
- Run the simulation run up to time, say, $2T$.
- Append the output of the second simulation to the first simulation.

This procedure of storing to/loading from harddisk requires much I/O. This is an expensive operation, in terms both of time and of storage space. We could consider whether changing just a single policy parameter cannot be done online *during* the simulation without the need to store everything to hard disk. This means that the parameter change should be automated such that all relevant variables remain in RAM instead of having to store them to hard disk. A change in the policy parameters could be implemented by setting a trigger for the iteration number after which the new policy value should be used. Since all model parameters in FLAME need to be modeled as memory values, it is necessary to be able to run scenario experiments without having to change the code and recompile the model between scenario runs. We therefore propose to include in the Simulation GUI an Experiment Manager allowing to define all parameter settings before actually running any simulations, with the Manager running all the needed scenario runs in batch mode.

1.3.2 GUI interface for running experiments: the Experiment Manager

In this section we consider the Simulation Design GUI (see the GUI scheme in the Vision Document).

This GUI should include a Population Creation Library. It should also include an 'Experiment Manager' for automating experiments, that is, to describe and manage the simulation runs.[1] The Experiment Manager Library will consist of multiple packages providing specific services. The manager should provide at least the following features (to be extended):

- *Memory selection package*: options for selecting which memory variables to save (detail versus speed).
- *Sampling package*: options for random sampling from the agent population (versus storing the memory of all agents).
- *Probing package*: options to explicitly select certain agents by agent id (used for tracing and probing a particular agent).
- *Batch run package*: options to create a batch run using the same initial conditions but different random seeds for the exogenous stochastic processes.

[1] Repast has such an experiment manager to run simulations in batch mode that is controlled via an input file giving the input parameters for each run.

- *Scenario experiment package*: options to systematically vary the policy parameters from batch to batch using in each batch the *same* set of random seeds. Typically, a scenario experiment involves only a small number of scenario cases (3-5). The GUI could provide an option 'Add scenario' to add a new parameter setting to a list.
- *Parameter sweep package*: options to perform a parameter sweep: to define the parameter space (a user-defined range and resolution) and to iterate over this space in an automated way. Typically, a parameter sweep involves testing many cases.
- *Space initialization package*: options to define spatial relationships between agents, providing tools to create grids (2D, torus) and networks (random, scale-free, small-world).
- *Source data package*: options to import source data. The Experiment Manager has multiple places where input files might be appropriate to import source data into a simulation's metadata:
  - inputting distributions (e.g., from empirical data sets)
  - inputting GIS data
  - inputting a parameter set (e.g., from a calibration exercise)
  - inputting a list of random seeds (to be used in the batch run package)
- *Data output package*: options to manage output data.
  - option to run in *fast/light* mode or *detailed/heavy* mode.
  - options to append data to an existing data set (restarting a pre-existing simulation).
  - an indication of the data storage required for the selected options.
  - an indication of the time required to run the selected options (based on previous runs).

1.3.3 Policy Design GUI

We suggest that the Experiment Manager and its library replace (and complement) the Policy Design GUI. Indeed, it is not entirely clear what should have been the content of the Policy Design library since a policy experiment basically consists of running multiple simulation experiments. So all we need is a GUI that deals with designing and running a set of simulations.

1.3.4 General GUI Design

Another point to be discussed is: Shouldn't we have a three-stage modelling process that starts from pre-simulation data (Agent Design GUI), followed by the simulation design (Simulation Design GUI), and then visualization of the post-simulation data (Visualization GUI)? This echoes the Model-View-Controller model for application design.

This consideration leads to the following proposal for a re-design of the GUIs:

1. Agent Design GUI: everything having to deal with pre-simulation data
    - X-Machine Modelling Library
2. Simulation Design GUI: everything having to deal with starting and running simulations
    - Population Creation Library
    - Memory Initialization Library
    - Parameter Initialization Library
    - Inference Library
    - Experiment Manager Library
3. Visualization GUI: everything having to deal with the post-simulation data
    - Visualization Library

1.3.5 Data visualization

We propose to have the Visualization Library integrated into the Visualization GUI. The selection of data visualization methods should be as general as possible, in order not to restrict the modeller. Therefore the Visualization Library should have at least the following features (to be extended):

- Graphics package: consisting of different graph types
    - Line graphs: views of single or multiple variables.
    - Histograms: useful because they can output full distributions of certain memory variables over the entire agent population, or of a sample of agents. The histogram view should also provide some summary statistics of the distribution being displayed, such as the number of observations, mean, median, minimum and maximum, skewness, kurtosis.
    - Scatter plots: data in (x_t, y_t)-space for two different variables
- Spatial package: consisting of different data viewers
    - Grid view: shows agent mobility across regions in spatial 2D grids.
    - Network view: shows abstract connections among agents.
    - Map view: shows geographical data on real-world maps of EU regions using color coded maps.

One should have the ability to use any of the graph types with any user-selected data source.

Feature requests

In order of development:

1. Multiple plots in one graph: a scroll-down menu in which all memory variables are listed, and you can select them by clicking. Selected variables are highlighted. Pressing a button ‘Plot’ plots all the selected variables in a single line graph.
2. Multi-plot with option to add an expression of variables, adding a name for the new variable, adding it to the list, and selecting and plotting it:
   First implementation:
   - Below the list there is an entry field allowing the user to enter a formula in terms of any variables in the list.
   - Next to this entry field there is a button ‘Add new variable’.
   - Pressing ‘Add new variable’ will ask in a dialog for the name of the new variable.
   - Pressing OK in the dialog adds the new variable to bottom of the list, and allows selection and plotting.

   Second implementation:
   - Below the list there is the button ‘Add new’.
   - Clicking ‘Add new’ opens an entry field allowing the user to enter a formula in terms of any variables in the list.
   - Next to this entry field there is a field for adding the name of the new variable.
   - Next to the name field there is a button ’Add’.
   - Pressing ’Add’ button adds the new variable to bottom of the list, and allows selection and plotting.

   Third implementation:
   - Below the list there is a button ‘Add new variable’.
   - Pressing ‘Add new variable’ will open a dialog with entry fields for the name of the new variable, and an entry field allowing the user to enter a formula in terms of any variables in the list.
   - Pressing OK in the dialog adds the new variable to the bottom of the list, and allows selection and plotting.
3. The plot window should have some standard GNUplot script options for plotting styles.
4. Option to use a user-created GNUplot script:
   - Button for ‘Add user script’.
   - Clicking on ‘Add user script’ opens an entry field to enter the path of the script.

1.3.6 Design of the Inference Library

We propose to have the Inference Library wrapped into the Simulation Design GUI. This GUI should also include a Parameter Initialization Library in which the user can set prior distributions on the parameters of the model. This information can then be used to generate random parameter draws.

Inference is not something static that only occurs after the simulation has finished and the results have been stored in the Iteration database. It is a dynamic procedure that occurs online without user intervention. It involves a feedback from the Iteration Database to an inference algorithm that is run by the Inference Library. The inference algorithm should steer the parameter settings that are being fed into the Simulation Controller (this will be a sequence of 0.xml's if the parameters are included in the 0.xml). This implies there should be a loop in the architecture of the GUI design between the Simulation controller and the Inference Library:

Parameters (0.xml) → Simulation controller → Iteration database → Inference Library → Parameters updated (0'.xml) → Simulation controller → Iteration database → ...

For the estimation of the model, the Inference Library runs some algorithm that adjusts the parameters in between simulation runs. The new parameter values are chosen conditional on the output in the Iteration database: the algorithm compares the moments of the synthetic distribution to the moments of the empirical data distribution. This inference algorithm is of course steered by the Inference Library, not the Simulation controller which only sees the 0.xml and runs it.

We could imagine that the Inference Library executes the following sequence:

1. send initial parameter draw to simulator engine
2. run simulations for the parameter draw (for an ensemble of n Monte Carlo replications, starting from 0_1.xml-0_n.xml that are created randomly)
3. store database (1_1.xml-T_1.xml to 1_n.xml-T_n.xml)
4. run inference algorithm on obtained database (e.g., compute moments of obtained data)
5. parameter draw is updated by the inference algorithm
6. send new parameter draw to simulator engine
7. run simulations for the new parameter draw (for a new ensemble of n Monte Carlo replications, starting again from the same 0_1.xml-0_n.xml, or from a new ensemble of random initial conditions $0'_1$.xml-$0'_n$.xml)
8. store database ($1'_1$.xml-T'_1.xml to $1'_n$.xml-T'_n.xml)
9. run inference algorithm on obtained database
10. repeat until algorithm stops by some convergence or stopping criterion

It would probably not be a good idea to just run the simulation for many different parameter settings, and then run the inference algorithm on the collected database, since in that case there would be no steering of the parameter settings and the algorithm can only use what data is already generated. It would imply that we have a collection of parameter settings that are drawn by the Inference Library at the start from the prior distributions that we set for each parameter in the Parameter Initialization Library. The disadvantage of this approach is that none of the drawn parameter settings may produce a distribution that is close to the empirical distribution, hence it cannot be considered an estimation technique in the proper sense.

Also, keep in mind that many Monte Carlo replications are needed for each parameter draw, to construct the probability density functions of the observed variables (this is in the order of $n = 1,000$ replications at least). Whenever there is a new draw of parameters, we can use exactly the same sequence of 0.xml files, or create an entirely new ensemble of 0.xml files. This choice depends on whether there is a strong dependence on initial conditions. In any case, there should always be a number of pre-iterations to limit the effect of a transient phase (there should be a setting in the GUI for the number of pre-iterations).

Using always the same set of initial conditions (0_1.xml-0_n.xml) could lead to degenerate results due to the fact that we always use the same ensemble for all parameter draws. That would imply that the estimation is very specific to the ensemble of initial conditions, which would make the estimation results non-generic. This implies that for *each* new parameter setting there is a new set of randomly drawn 0.xml files, one for each replication (there should be a setting in the GUI for using the same set of initial conditions each time, or generating a new set).

However, the set of seeds for the random number generators is a fixed set, consisting of n seeds $s_1, ..., s_n$ that are associated to the n Monte Carlo replications. For every new parameter draw, the exact same seed s_j should be used to generate the stochastic series in replication j (there should be a setting in the GUI for using the same set of random seeds).

The data that is stored in the database for each parameter draw (1_1.xml-T_1.xml to 1_n.xml-T_n.xml) are the values of a small number of observable variables (these will be macrovariables such as: real GDP, real consumption, real investment, nominal interest rate, inflation, and real wages).

Setting the number of observable variables for the estimation to $N = 10$, the number of parameter draws $P = 10,000$, the number of Monte Carlo replications $n = 1,000$, the sample size $T = 5400$, this would yield a total database (keeping all intermediate data):

$$10 \times 10,000 \times 1,000 \times 5400 = 5.4 \times 10^{11} \text{doubles} = 4.32 \times 10^{12} \text{ bytes} \approx 4 \text{ Tb}. \qquad (1.1)$$

If we would not store the intermediate results after each parameter draw, but simply overwrite the database each step of the inference algorithm, the database contains only the time series of the current parameter draw and would be 400 Mb in size. This is the minimal size of the database since this information is needed by the inference algorithm to construct the probability density functions of the observable variables and to compute the moments of these distributions. After comparing the moments of the artificial data to the moments in the empirical data, the parameter draw is updated and the database overwritten.

In the data flow chart of the GUI hierarchy, the Inference Library should be placed after the Population Generation Library, but before the Memory Initialization Library. This is so because the size of the populations is kept fixed throughout the estimation routine, but for each of the n Monte Carlo replications, a sequence of n random initial conditions 0_1.xml-0_n.xml should be created. The Memory Initialization Library can then be used to generate the entire bunch of 0.xml files as needed, and these are then stored by the Inference Library.

Alternatively, the Inference Library can also be placed after the Population Generation Library and the Memory Initialization Library, but the Inference Library nonetheless makes use of the Memory Initialization Library to generate the ensemble of 0.xml files.

1.3.7 Overview of the proposed GUI design

In the preceding sections we have described a hierarchical design in which each GUI consists of several libraries, and each library consists of several packages. The packages in turn consist of options that can be selected by the user.

1. Agent Design GUI: everything having to deal with pre-simulation data
    - X-Machine Modelling Library
2. Simulation Design GUI: everything having to deal with starting and running simulations
    - Population Creation Library
        - Population size per agent type
        - Regional distribution of agents
    - Memory Initialization Library: setting prior distributions on memory variables
    - Parameter Initialization Library: setting prior distributions on parameters
    - Inference Library
        - Setting: inference algorithm to use
        - Setting: convergence criterion of the algorithm
        - Setting: stopping criterion, max. number of parameter draws
        - Setting: number of Monte Carlo replications per draw
        - Option: use the same set of random seeds across all draws (one seed per replication)
        - Setting: sample size (number of iterations per replication, number of pre-iterations)
        - Selection: which parameters to use in the estimation procedure
        - Selection: which observable variables to use in the estimation procedure

- Experiment Manager Library
  - Memory selection package
  - Sampling package
  - Probing package
  - Batch run package
  - Policy experiment package
  - Parameter sweep package
  - Space initialization package
  - Source data package:
    - inputting distributions
    - inputting GIS data
    - inputting a parameter set
    - inputting a list of random seeds
  - Data output package:
    - option to run in *fast/light* mode or *detailed/heavy* mode
    - option to save data only at a certain frequency (quarterly, yearly)
    - options to append data to an existing data set (restarting a simulation)
    - indication of data storage required for selected options
    - indication of time required to run selected options

3. Visualization GUI: everything having to deal with the post-simulation data

   - Visualization Library
     - Graphics package:
       - Line graphs
       - Histograms
       - Scatter plots
     - Spatial package:
       - Grid view
       - Network view
       - Map view

Chapter 2

Inference in EURACE

The point of departure for this proposal is that any method of inference that will be used to estimate large-scale agent-based models has to be both feasible and acceptable. The criterion of feasibility refers to the fact that any method to be used has to be computationally feasible, and not require enormous amounts of simulation runs. This leads to a certain degree of pragmatism in the selection of the statistical methods, and also serves to discipline the modeller.

The proposal is to use a methodology akin to that being used to estimate DSGE models, but to use it in a more modest way. In DSGE models, the Data Generating Process (DGP) is derived from solving a log-linearized version of a system of Euler equations. The solution to that system at every iteration, after adding auxiliary stochastic shocks, is what drives the dynamics. In our case, we specify an explicit law of motion that provides us with the DGP. Since the inference method that is used to estimate DSGE models does not depend on any particular specification of the DGP, in essence we can adopt the same methodology to estimate the structural parameters of the agent-based EURACE model.

2.1 Estimation methods

To estimate the structural parameters of the model, we have a choice among several standard methods of statistical inference: Maximum Likelihood (ML), Simulated Method of Moments (SMM), and Generalized Method of Moments (GMM).

With respect to the informational requirements, Maximum Likelihood is the hardest computationally since it requires the numerical construction and evaluation of the likelihood function of the structural parameters given the data. It also requires the numerical computation of the Hessian matrix for the maximisation of this likelihood function. The Methods of Moments are less computationally demanding since they only rely on the derivatives of the unconditional moments with respect to changes in the parameters. The main difference between SMM and GMM is that GMM requires the analytical calculation of the unconditional moments in terms of the parameters, while SMM only requires the numerical computation of the derivatives.

Using the Maximum Likelihood method to estimate the EURACE model will probably be too heavy since it is already quite demanding for simple one sector RBC models. Therefore we would propose to use a method based on Simulated Method of Moments:

> In calibration, the researcher computes the unconditional moments of synthetic series simulated using given parameter values and then compares them with the unconditional moments of the data. The Simulated Method of Moments (SMM) estimator pursues this idea further by updating the parameter values in a manner that reduces the distance between these two sets of moments.' (Ruge-Murcia, 2007, p. 2608)

Recall that in Simulated Method of Moments the identification of the structural parameters requires that the number of observable variables has to be greater than (or at least as large as) the number of parameters we try to estimate.

2.2 Markov Chain Monte Carlo (MCMC)

The current state-of-the-art for estimating Dynamic Stochastic General Equilibrium (DSGE) models is to use the Markov chain Monte Carlo (MCMC) sampling technique. For technical details, see Fernández-Villaverde and Rubio-Ramírez (2004). For applications, see e.g., DeJong et al. (2000); Smets and Wouters (2003); Ruge-Murcia (2007).

A somewhat technical description of MCMC

The MCMC sampling method is illustrated in Fig.2.1. The estimation method starts by setting prior distributions on the parameters to construct a search space. A numerical algorithm is used to search through this space in a more or less systematic way. The objective of the algorithm is to numerically derive the posterior density of the parameters θ by minimizing the distance between the unconditional simulated moments in the artificial time series simulated for given parameter values, and the unconditional moments in the data. Usually this comes down to finding the mode of the posterior distribution, i.e. that value at which the density function attains it maximum. Theoretically, the mode of the posterior distribution converges in probability to the maximum likelihood estimator of the parameter almost surely.

To perform the simulations and sample from the prior parameter distributions, an algorithm can be used such as the Metropolis-Hastings algorithm, simulated annealing, or even a Genetic Algorithm. The algorithm starts from a random initial draw from the search space, i.e., a randomparameter draw, and then sequentially tries to improve on this draw until some convergence criterion is met, or until it reaches some stopping criterion (i.e., a maximum number of parameter draws). The sequence of draws shows a sequential dependence and thus forms a Markov chain. Globally speaking, the search may consist of just one Markov chain exploring the search space, or it may consist of multiple parallel chains, simultaneously exploring the space in parallel.

For each draw, the algorithm has to perform n Monte Carlo replications using the same (or different) random seeds (say at least $n = 1,000$ replications are needed to have a sufficient sampling of the artificial data). Each replication provides a sample by performing a simulation run. If the number of observations in the data is denoted by T, then the sample size should be τT, with $\tau = 5, 10, 20$. This is to ensure that there are a sufficient number of observations in the synthetic data.

After all Monte Carlo replications for the current parameter draw have been run, we continue to construct the distribution from the ensemble of n samples ($n\tau T$ observations in total). Let y_t be the empirical observations in the data, and let $y_i(\theta_i)$ be the synthetic observations from the artificial time series, given the parameter values θ_i. The probability density function over all n Monte Carlo replications is denoted by $f(y|\theta, i)$. This function is also called the pseudo-likelihood function of the observable data series y_i, conditional on the parameter vector θ_i (see Fernández-Villaverde and Rubio-Ramírez, 2004, p. 156, Smets and Wouters, 2003, p. 25).

There seems to be a theoretical reason why, for each new parameter draw, the same n random seeds should be used for the same n Monte Carlo replications, respectively. The probability density function $f(y|\theta)$ obtained from the Monte Carlo sampling is proportional to the marginal log-likelihood function of the parameters $L(\theta|y)$. Furthermore, it becomes a better estimate of this likelihood function as the sample size increases. Since the likelihood function is a continuous function of the parameters it can be maximized by an optimization routine that follows a gradient, which is what the Metropolis-Hastings algorithm actually does when it is drawing new parameters. If we would refrain from using the same set of n random seeds for each parameter draw, and instead use different random seeds for each replication, the numerically approximated marginal likelihood function would be discontinuous and therefore cannot be maximized by an algorithm that makes use of local derivatives.

The unconditional moments of $f(y|\theta, i)$ provide numerical estimates for the marginal log-likelihood functions $E_i(f(y|\theta, i))$ and for large enough sample size the moments converge asymptotically to their analytical counterparts.

The algorithm proceeds by comparing the simulated moments of the Monte Carlo simulations to the unconditional moments of the data, according to some measure of closeness (for example, using the Kullback-Leibler Information Criterion (KLIC), see Fernández-Villaverde and Rubio-Ramírez, 2004, p. 157).

Over the course of the Monte Carlo sampling, the algorithm numerically computes the derivatives of the simulated moments with respect to changes in the parameters.

$$\frac{\partial f(y|\theta, i)}{\partial \theta} \rightarrow_{prob.} \frac{\partial E_i(f(y|\theta, i))}{\partial \theta} \tag{2.1}$$

These derivatives can now be used as the gradient by the optimization routine: a new draw is made from the parameter search space, in the direction of the numerically computed gradient, in order to improve the closeness of the moments. The entire cycle of Monte Carlo replications is then repeated for the new draw, and the algorithm continues until a parameter value is found such that the simulated moments from the artificial time series have a minimal distance to the unconditional moments in the empirical data.

This parameter value is found at the mode of the posterior probability density function of the parameters, which theoretically corresponds to the parameter estimate that maximizes the log-likelihood function, since the posterior probability density function is precisely a numerical approximation to the log-likelihood function.

Once the algorithm converges, the set of parameter values that has been found is assumed to be the one that maximizes the log-likelihood function. The estimation can then be used to compute posterior functions of the parameters such as value functions, impulse-response functions, moments, etc.

Some remarks

Let us consider the computational complexity of the whole procedure.

Taking $T = 20$ years as the reference period for comparisons to the empirical data, and considering that for macroeconomic data we have usually four quarterly observations per year, this yields 80 observations of the observable variables. Considering that in the EURACE model one iteration refers to a day, we have to simulate $20 \times 240 = 4800$ iterations in order to match this number of observations at the correct frequency. We should also run a certain number of pre-iterates to limit the effect of transients, which we propose to limit to 2 months, i.e. 40 iterations. These initial observations are discarded in the estimation procedure. A single simulation run would then consists of 4840 iterations. However, it would always be advisable to produce simulated series that are larger than the sample size of 80 observations, by taking $\tau = 5, 10, 20$. This would lead to sample sizes of, respectively, $24,200$, $48,400$ and $242,000$ iterations.

It is not unlikely that convergence of the search algorithm will require approximately $N = 10^5$ parameter draws. With $n = 10^3$ replications per draw, this would amount to 10^8 simulation runs in total. Depending on the time it takes to perform a single simulation run, we may want to consider limiting the number of Monte Carlo replications and/or the maximum number of parameter draws such that the entire procedure can be run in one week of real simulation time.

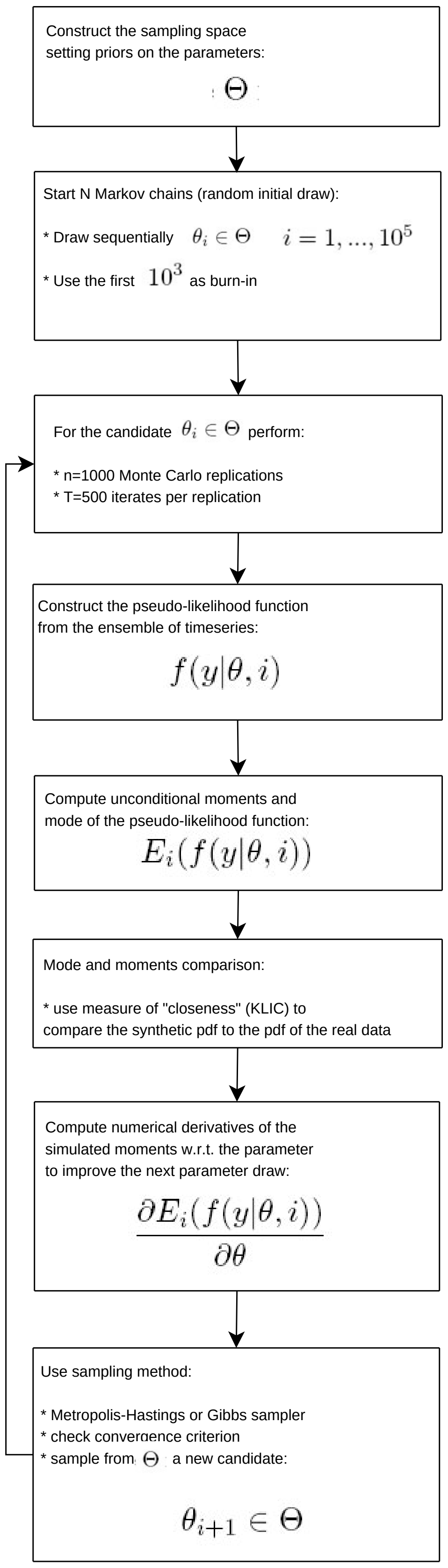
A final judgement as to whether the Markov chain Monte Carlo method will be computationally feasible for the EURACE model most likely will have to be postponed until we have performed a complexity analysis of the model, and an analysis of whether the results from simulations scale with the size of the population.

As a thought experiment, suppose that the results from simulating an economy with 10^6 households and $1,000$ firms and an economy with 10^4 households and 100 firms are qualitatively the same. This would imply that after we have tested this, we can employ the model with moderate population sizes to do the estimation exercise. Further suppose that we can bring down the number of replications to $n = 250$, and that a sample size of $T = 4840$ iterations is sufficient to have a sufficiently high degree of confidence in the estimation results. Another economizing feature would come from not saving the output at every iteration, but only at the actual frequency of the observations, i.e. at a quarterly or yearly frequency.

In other words, a number of concessions could be made that yield a pragmatic approach to estimating very large computational models.

Step-by-step instructions

- Set the prior distributions for parameter values. This generates the space Θ for parameter draws.
- Start N Markov chains to explore the prior distribution.
- For each Markov chain:
  - Starting from an initial parameter draw:
    - Perform n Monte Carlo replications (using the same n random seeds for each draw): Iterate the simulation for a sample size of τT periods.
    - Each Monte Carlo replication yields a distribution across the sample of τT observations (hence $n\tau T$ observations are obtained in total across all replications). The Monte Carlo distribution of the observed variable now follows from the strong law of large numbers (the sample converges in probability, and the sample uncertainty can be reduced by choosing a τ large enough).
    - Comparison of the moments of the Monte Carlo distribution to the moments of the empirical distribution yields information about the improvement in the parameter values. It can be used to compute the numerical derivatives.
  - Apply the algorithm to update the parameter draw in the direction of the gradient, selecting a new candidate parameter setting. The algorithm contains an acceptance or rejection rule to accept or reject the new draw, and to update the numerical approximation of the gradient.
  - Continue with new draws until some convergence criterion for the moments is met, or a stopping criterion (max. number of draws).
- Compare the final draws of each Markov chain. The sequence of draws in each Markov chain should converge to an ergodic probability density, which is the posterior probability density of the parameters.



Construct the sampling space setting priors on the parameters:
Θ
Start N Markov chains (random initial draw):
* Draw sequentially $\theta_i \in \Theta$ $i = 1, ..., 10^5$
* Use the first 10^3 as burn-in
For the candidate $\theta_i \in \Theta$ perform:
* n=1000 Monte Carlo replications
* T=500 iterates per replication
Construct the pseudo-likelihood function from the ensemble of timeseries:
$f(y|\theta, i)$
Compute unconditional moments and mode of the pseudo-likelihood function:
$E_i(f(y|\theta, i))$
Mode and moments comparison:
* use measure of "closeness" (KLIC) to compare the synthetic pdf to the pdf of the real data
Compute numerical derivatives of the simulated moments w.r.t. the parameter to improve the next parameter draw:
$\frac{\partial E_i(f(y|\theta, i))}{\partial \theta}$
Use sampling method:
* Metropolis-Hastings or Gibbs sampler
* check convergence criterion
* sample from Θ a new candidate:
$\theta_{i+1} \in \Theta$

Figure 2.1: Diagram of the MCMC sampling method.

Chapter 3

Aggregation

3.1 Summary

Ramsey (1996) offers a pragmatic approach to the problem of aggregation. He defines a macro variable as a measure of some statistical characteristic of a dynamical system as a whole. Further, he advocates using a statistical approach, taking ensemble averages (averaging over initial conditions) and time scale averages of time varying probability distributions of the "fast" micro variables. An operational definition of a macro variable is then to define it as a moment of the sequence of time varying probability distributions. This defines the macro variables as "slow" variables of the system. The principle notion is that the total (mesoscopic) fluctuations can be separated into "slow" and "fast" components using "random phase approximations" (à la Boltzmann).

3.2 The aggregation problem

[This is a summary of: Ramsey (1996)]

The main messages of the paper are:

- "Macro variables are not merely aggregates; they are measures of the properties of the system itself."
- "System variables [may] exhibit *fundamental* properties that are not held by the micro components."
- Any attempt to find a relationship between macro variables by aggregating micro relationships constitutes a "fallacy of composition".
- "Macro relationships, if they exist, are most likely to be discovered in terms of the parameters of time varying probability distributions of system-wide properties."

Ramsey (1996) defines macro variables as measurements of a system at the macro level. From a dynamical system point of view this implies that macro variables represent the statistical properties of a dynamical system. Micro and macro variables are thus measurements of the same system, but at different levels of analysis.

If macro variables are measurements of the overall system it does not necessarily make sense to define macro variables as sums of micro variables. They can be defined in their own right, as measurements of some characteristic properties of the system as a whole. One can think of the mean income or the mean consumption which are properties that exist at the macro level, but do not necessarily exist at the micro level, since they are statistical properties of probability distributions.

It is also important to consider the time-varying probability distributions of macro variables. The statistical properties of these distributions are also macro-variables in their own right. I.e. one could study the distributions of the time-varying mean and the time-varying variance:

> 'The ultimate task of the economist is to determine the time varying distributions, explore the relationships between the joint distributions of the variables involved, and **to provide explanations from economic theory for the observed time paths of the distributions and their distributional properties**. This view, of course, is merely a formal return to the formulation envisioned by the writers of the Cowles Commission monographs.' (Ramsey, 1996, p. 276, emphasis added)

Summarizing, a solution to the problem of defining macro variables is to focus on '*key system-wide*' variables that represent the properties of a system at an aggregated level. The problem is thus closely connected to emergent phenomena and to matching 'empirical stylized facts'. These stylized facts are also system-wide properties, only of a real-world system. Once the macro variables have been successfully defined, the next problem is to analyse the time paths of these variables. The aggregation problem can be succinctly summarized as follows:

- Define macro variables as statistical measures of some characteristic aspects of a system.

- Find stable relationships between the defined macro variables. These can be seen as the stylized facts of the system, since they relate to statistical properties.

- Having found these stable relationships, interpret the macro variables in terms of micro theory. This is the problem of *interpretability*: can one interpret the relationships found at the macro level in terms of a theory about the micro level?

- Relate the macro parameters to the micro parameters. This is the *recoverability* problem: is it possible to recover the micro level properties of the system from the macro level properties?

The basic recoverability problem is the following: Is it possible to find a connection between the relationships of macro variables and the relationships of micro variables? How do these two levels interact?

It also relates to the critique by Kirman on Representative Agents, stating that in principle the aggregate properties of a system cannot be derived from the properties of its constituent parts, and any attempt to do so would constitute a 'fallacy of composition'. The opposite of this is the 'fallacy of decomposition', which is the attempt to derive the properties of the constituent parts from the properties of the system as a whole. Hence the problem of recoverability.

Examples are macro level properties such as 'pressure' or 'temperature' that do not correspond to any property at the micro level. It is of course possible to define measures of entropy for the individual elements and then to relate these to the overall temperature, but the total entropy measure is a macro variable itself (since it is obtained by aggregating individual entropy measures, and it is assumed that we are dealing with a homogeneous particle system that is ergodic). So the relationship is therefore a macro relationship. Many 'economic laws' have a similar feature: they are stable relationships between macro variables that measure some statistical properties of the macro system.

Resolution

A potential resolution for the problem of aggregation is to use a purely statistical approach. One can examine the macro relationships in their own right, without making any appeal to the micro level:

> 'Macro relationships are best regarded as relationships in their own right and one should not necessarily attempt to relate the macro formulation to any specific micro formulation. Macro variables and their relationships deal with properties of the entire system and [...] these system-wide properties are often not mere aggregations of micro components.' (Ramsey, 1996, p. 294)

This recognizes the fact that the qualitative dynamical behavior of macro variables is usually very different from the qualitative dynamical behavior of micro variables. This approach implies that one assumes that the macro variables come from some underlying data generating process, but without any knowledge about the underlying mechanisms. The study of the macro relationships is performed in complete isolation from the study of the micro relationships.

The objective of this statistical approach is to define new macro variables that are explicitly designed to measure characteristic statistical properties of the system as a whole, without any reference to the underlying mechanisms:

> 'The dominating focus of the aggregation literature has been on the existence of stable relationships for indices of micro components; more simply, sums or averages of micro components. A broader view would allow for the existence of system variables that are not mere sums, or indices of micro components. Further, it might well be the case that no functional relationship can be found between micro components and an observed macro variable, but that the latter is an asymptotic limit of the convolution sum of the former.' (Ramsey, 1996, p. 281)

3.3 Some examples of definitions of macro variables

On entropy measures of an economic system:

> ‘In general, a sequence of entropy measures, for example, on consumption, investment, and savings, probably provide a useful set of measures on various aspects of the state of the economy, but without specifying the linkages among them. An obvious question involves the time evolution of the entropy measures and to relate those paths to economic and financial events.’ (Ramsey, 1996, p. 284)

As examples of possibly useful definitions of macro variables, Ramsey mentions the following:

1. The money supply M_1.
2. The entropy measure of income.
3. The velocity of money.
4. An index of the general price level.
5. Measures of the intensity of market participation (a measure of ”rate of trading”).
6. Concepts of thickness and thinness of the market.
7. Measures for the relative concentration of active agents in the market place.
8. The temperature of the market (as defined by Foley, 1994).

Ramsey issues the following warning on the aggregation of micro-components:

> ‘There is no a priori theorem in any discipline that the aggregation of micro-components will necessarily satisfy the stringent conditions required to yield a stable relationship among aggregations of components. There is, in fact, no theorem that any set of aggregates will necessarily satisfy a set of differential equations, no matter how well known the micro relationships. The existence of a useful dynamical macro relationship is an empirical matter.’ (Ramsey, 1996, p. 284)

On the need of ‘random phase approximations’ in economics, Ramsey notes that:

> ‘[In physics, Boltzmann coined the term] “stosszahlansatz,” or “random phase approximations” for the assumption that underlies the need for repeatedly averaging at successive time points over the irrelevant rapidly varying components to reveal the “slowly varying” ones. This averaging is over time. The principle notion is that the total fluctuation can be separated by time scale of variation into “slow” and “fast” components. We average out the “fast” components to reveal the “slow components”.’ (Ramsey, 1996, p. 285)

The macro variables of interest should then be defined as statistical measures of the probability distributions of ensemble averages over stochastic simulation paths. The macro variables themselves are "slow" variables, since they are measurements of the statistical properties of the "fast" variables in the system. According to Ramsey (1996, p. 286), the proper purview of macro economists is the macroscopic equations that relate the macro variables to each other dynamically:

> 'A fortiori, the need for such an assumption is even more pressing in economics, where one is dealing with very large numbers of agents and firms, each with a highly multidimensional array defining their state at each point, each with their own local equations that relate their behavior to their current state, and all acting within an open and non-isolated system.' (Ramsey, 1996, p. 285)

The problem of the possible non-existence of macro variables

> 'A priori, it is not clear that any such averaging procedure will work; that is, it is not at all obvious that any slowly varying variables can be revealed by this procedure.' (Ramsey, 1996, p. 286)

The problem is that for certain probability distributions the first and second moments may not exists.

Methodology

The methodological approach to properly define "macro" variables is to separate the "slowly" varying components from the "fast" changing components, by taking ensemble averages over time of the joint distribution of the system's state variables. The macro relationships can then be obtained as follows:

> By taking asymptotic limits of a convolution sum one can obtain a macro relationship without being able to aggregate the micro relationships in any meaningful functional way.' (Ramsey, 1996, p. 287)

3.4 Operationalizing the definition

To operationalize the definition of macro variables, Ramsey uses three levels of analysis: microscopic, macroscopic and mesoscopic.

> The mesoscopic fluctuations are the fluctuations of the system about the macroscopic states. In a precise sense, the mesoscopic fluctuations reflect the "averaged" variations of the micro components about the macrostate. We also need to recognize two types of averaging. Ensemble averaging over initial states and time averaging leads to a Markovian representation for the relatively slow components.' (Ramsey, 1996, p. 288)

The definition of a macro variable occurs in three steps (see Ramsey, 1996, p. 290):

1. Specify a measure of an observable and characteristic property of the dynamical system under consideration.

2. Take an ensemble-average over the dynamics, by averaging over the initial conditions. This produces the mesoscopic fluctuations that consist of a time-varying sequence of probability distributions of the chosen characteristic.

3. Extract from the mesoscopic fluctuations some property of the sequence of distributions. This property now constitutes an observable time path for a variable, that can now be called a macro variable.

Natural candidates for the macro variables are the time-varying moments from the sequence of distributions. The search for macro relationships then consists in finding relationships between these time-varying moments. The macro variables themselves are non-random since they are moments (parameters) of distributions. The macro relationships are deterministic, since they are relationships over time between the parameters of the time-varying probability distributions.

The macroscopic equations are deterministic in nature, while the mesoscopic equations show variations around the deterministic macro relations. These variations have been averaged out in the construction of the macro variables by taking the ensemble averages.

Bibliography

DeJong, D. N., Ingram, B. F., Whiteman, C. H., 2000. A bayesian approach to dynamic macroeconomics. Journal of Econometrics 98, 203 – 223.

Fernández-Villaverde, J., Rubio-Ramírez, J. F., 2004. Comparing dynamic equilibrium models to data: a bayesian approach. Journal of Econometrics 123, 153 187.

Ramsey, J. B., 1996. On the existence of macro variables and of macro relationships. Journal of Economic Behavior and Organization 30, 275–299.

Ruge-Murcia, F. J., 2007. Methods to estimate dynamic stochastic general equilibrium models. Journal of Economic Dynamics and Control 31 (8), 2599–2636.

Smets, F., Wouters, R., 2003. An estimated dynamic stochastic general equilibrium model of the euro area. Journal of the European Economic Association 1 (5), 1123–1175.