LIAM 2: a new open source development tool for the development of discrete-time dynamic microsimulation models.

Gaëtan De Menten\textsuperscript{a} Gijs Dekkers\textsuperscript{b}, Raphaël Desmet\textsuperscript{a}, Geert Bryon\textsuperscript{a}, Philippe Liègeois\textsuperscript{c}, Raymond Wagener\textsuperscript{d}, Cathal O’Donoghue\textsuperscript{e}

\textsuperscript{a}. Federal Planning Bureau, Belgium.
\textsuperscript{b}. Federal Planning Bureau, CESO University of Leuven, Belgium, and CEPS/INSTEAD, Luxembourg.
\textsuperscript{c}. ULB, Belgium, and CEPS/INSTEAD, Luxembourg.
\textsuperscript{d}. IGSS, Luxembourg.
\textsuperscript{e}. Teagasc and National University of Ireland, Ireland.

Paper presented at the séminaire scientifique international Direction des retraites et de la solidarité de la Caisse des Dépôts,

Bordeaux, France,
November 15th, 2012.
Keywords: microsimulation, software, LIAM2

JEL-codes: D31, I32, C53, C63, C88

Acknowledgement

This paper is the result of the MiDaL project (December 2009 - November 2011), “Towards the development of a dynamic Microsimulation toolbox « LIAM-II » and the complementary implementation of administrative Data needed for dynamic microsimulation of pensions in Luxembourg”. The authors gratefully acknowledge the support of the European Community Programme for Employment and Social Solidarity - PROGRESS (2007-2013), under the Grant VS/2009/0569, as well as the General Inspectorate for Social Security (IGSS) of Luxembourg.

Summary:

Most existing microsimulation models have been developed by separate (teams of) researchers. The drawback of each team working on its own is that they have to put a lot of time and effort in the customary development of fairly general simulation tools. Hence, economies of scale cannot be exploited, which makes microsimulation models even more expensive than strictly necessary. The objective of this paper is to introduce and present the simulation modelling package LIAM2. This is a free, open source, modelling framework, designed for the development of discrete-time dynamic models with dynamic ageing. The paper presents a template model and discusses the performance of LIAM2 in terms of data capacity and simulation speed.
1. Introduction

Dynamic microsimulation models are models that can be used to design and evaluate public policies that are affected by inter-temporal characteristics and choices as in the case of pensions or education policy. Given the need to model behaviour and policy over time, these models are significant undertakings requiring panel data, modelling and database management programmes and significant expertise.

The technology to run analyses like this in an operational capacity exist since the 1970’s (See Orcutt et al., 1976) and in many ways the structure of these models in use today is similar (see Li and O’Donoghue, 2012). Thus progress has been relatively slow. One of the reasons for this discussed in Li and O’Donoghue is the lack of availability of shared code or model frameworks. The proprietary ownership of source code while generating private returns to model builders may have inhibited the development of the field. The move to shared frameworks such as LIAM (O’Donoghue, et al., 2009), ModGen (Spielauer, 2006), UMDBS (Sauerbier, 2002) and JAMSIM (Mannion et al., 2012) can help to spread the technology and facilitate development, especially when they are open source.

The objective of this paper is to introduce and present the new dynamic microsimulation modelling package LIAM2, which although inspired by some of the capabilities of LIAM is substantially different in many respects. This is a free, open source, modelling framework, designed for the development of static models and discrete-time dynamic models with dynamic ageing.

Microsimulation models are usually large and rather complicated models consisting of many lines of code. With the static model EUROMOD (Lelkes and Sutherland, 2009), the spatial models SMILE (O’Donoghue et al., 2012) and the dynamic model MIDAS (Dekkers et al., 2010) as the most notable exceptions, most existing models have been developed by separate (teams of) researchers. Also, in many cases, the technical parts that are necessary for any model but that are not part of a model as such, are developed by each team separately in an ad hoc manner. This includes the routines for data-reading and writing, search and alignment routines, parsing, and so forth; these are all general tools, but as any team that wishes to develop a model needs to have them, the drawback of each team working on its own is that they have to put a lot of time and effort in the customary development of these tools. Hence, economies of scale cannot be exploited, which makes microsimulation models even more expensive than strictly necessary. Furthermore, in the development phase of many models, modellers were also responsible for the programming of the simulation tools and methods. Since these modellers often are not professional programmers, the result was not necessarily the most efficient in terms of simulation speed.

A solution to these problems lies in the use of simulation tools for modelling and development. Li and O’Donoghue (2012) group packages that are used in the development of social policy oriented microsimulation models in three categories. The first two are general purpose programming tools (such as C/C++/C#, Python, Fortran, ...) and statistical packages (Stata, SAS, R). The third category includes “simulation modelling packages”, such as
UMDBS (Sauerbier, 2002), LIAM (O'Donoghue et al., 2009), ModGen¹ (Spielauer, 2006), JAMSIM (Mannion et al., 2012), GENESIS (Strassburg and Tracey, 2011) and LIAM2.

In this paper we shall provide some context for the development of LIAM2 in relation to other existing modelling frameworks in section 2. Section 3 provides a discussion of the technical structure of the model. Section 4 concludes

2. Context: Model Structure and Comparisons

What this development tool is can be explained through the analogy with a statistical package such as SAS, or Stata in econometrics and empirical analysis. These packages allow researchers to use advanced statistical techniques and regression models without having to program the “nuts and bolts” themselves. So one can estimate a regression model without having to invert the data matrix or maximize a likelihood function.

The aim of LIAM2 is to take up the same role in microsimulation modelling. It is a framework that allows for comprehensible modelling and the straightforward use of various simulation techniques. It allows for simulations in dynamic discrete-time microsimulation models with dynamic ageing, as well as static microsimulation. It takes care of data-handling and parsing, while allowing the modeller freedom in applying the simulation procedures to its specific means.

The toolbox is made as generic as possible so that it can be used to develop a wide range of static or dynamic microsimulation models with cross-sectional ageing²

LIAM2 is a generic package, meaning that it allows for the simulation of whatever kind of objects the modeller chooses. To keep the discussion simple, however, the below examples pertain to demographic processes on individuals or households, and/or on the simulation of tax and transfer systems.

LIAM 2 is the result of intense and ongoing collaboration between individual researchers from various institutions. The software package has been developed at the Federal Planning Bureau. Testing was done by the Luxembourg Team (CEPS/INSTEAD and IGSS). Finally, Cathal O’Donoghue provided methodological and conceptual information and shared the source code of the first version of LIAM, which the FPB used in the AIM project (see Dekkers et al., 2010).

An important characteristic of LIAM2 is that it is freely available (http://liam2.plan.be/) and open source (the source code is also available and is licensed under the GNU Public License version 3).

The basic idea behind LIAM2 is to

¹ http://www.statcan.gc.ca/microsimulation/modgen/modgen-fra.htm
² all individuals are simulated at the same time for one period, then for the next period, etc.
Separate ‘modellers’ from ‘programmers’, where the former are responsible for the model and the latter take up the development of the methodological issues, such as data reading and writing, parsing and so forth.

Provide a development environment for the development of large discrete-time dynamic microsimulation models

Use state-of-the-art methods for data-handling and simulation, thereby allowing these models to simulate larger datasets at higher speed.

Stimulate collaboration and the exchange of tacit information between development teams through the use of a common development approach.

Be open source, so that developers worldwide can add to the toolbox and share current and future methodological developments.

**Comparative analysis**

What is the difference between LIAM2 and other simulation modelling packages? There unfortunately is not much literature on this comparison, with Li and O’Donoghue (forthcoming) the notable exception. This section compares LIAM2 with LIAM, ModGen, and GENESIS. UMDBS is not included in this discussion, because it is to our knowledge no longer in use.

LIAM2 has been developed based on experience gained through the use of LIAM in the development of MIDAS (see Dekkers et al., 2010; Dekkers and Desmet, 2011). Thus LIAM2 is the ‘spiritual successor’ of LIAM. The two packages therefore are substitutes and share that they are freely available. But LIAM2 is more general, considerably faster, and allows for much larger datasets to be simulated (cf. infra). Furthermore, where every part of a model (definition of entities, fields, their interactions and all processing rules) required separate ascii-files in LIAM, this has now been grouped in one file and is done using the YAML markup language. Finally, LIAM2 is broader than LIAM in terms of the tools and possibilities it offers the model developer.

ModGen is a freely available software tool developed and maintained by Statistics Canada. It is essentially a C++ library that allows modellers to include common microsimulation procedures and methodologies. So model development in ModGen still requires programming in C++ (and buying and using Visual Studio). JAMSIM is conceptually equivalent, other than that it is developed in Java. It is “less a framework and more a loose coupling of a set of open source packages to provide a base set of functionalities for microsimulation” (Mannion et al., 2012, 5.1).

LIAM2 is, by contrast, a development environment that has its own syntax. Even though LIAM2 itself has been developed in Python, this has no consequences for the end user. Another difference is that ModGen is more general than LIAM2. Although it is capable of handling discrete time models with cross-sectional ageing and alignment (see, for example,
ModGen is developed for continuous-time models, such as LIFEPATHS (Légaré and Décarie, 2011). JAMSIM seem to focus on the development of agent based models. LIAM2 is more specific than ModGen and possibly JAMSIM, focusing on large static and discrete-time dynamic models with cross-sectional ageing.

GENESIS is a generic dynamic simulation model developed and used within the UK Department of Work and Pensions. It can be used by analysts to simulate many different scenarios without requiring changes to the model code. The model is made up of two distinct areas, the SAS code that makes up the generic Model Engine and the parameter spreadsheets held in Excel. The Processing Rules spreadsheets are set up in a standard structure. A run takes the processing rules and parameters tables from the parameter spreadsheets and then uses this to convert the generic model into a specific model. This specific model is then simulated against the data tables. Data in GENESIS can come from a variety of sources that are brought into a standard data model. With the generic model thus comes a dataset and it is not possible, or—perhaps—very cumbersome to take a different starting dataset for a specific application.

GENESIS is not freely available: first of all, to date and to our knowledge, it is only used within the Ministry of Work and Pensions. Secondly, it requires SAS, which is not free. Furthermore, it is more specific than LIAM2, in that it is essentially a generic model and dataset. It is therefore not possible to go outside the ‘boundaries’ of this generic model/dataset. LIAM2, by contrast, is a development environment, and not a model; it is thus independent of the data available and does not require staying within the boundaries set by a generic model. Finally, preliminary tests have shown that LIAM2 seems faster in simulation than GENESIS.

Summarizing, LIAM2 is the spiritual successor of LIAM, but it is superior to its ‘ancestor’ in terms of modelling scope, speed of simulation, data handling capacity. It appears that LIAM2 is more general than GENESIS and more specific than ModGen and JAMSIM, it does not require programming in a full blown programming language (unlike ModGen and JAMSIM), nor does it imply modifying an existing generic model and dataset (unlike GENESIS). It is specifically designed for the development of large discrete-time dynamic microsimulation models with cross sectional ageing, and it can be used for static models more or less as a side-effect. Finally, LIAM2 is available without any direct or indirect costs, and it is open source.
3. Characteristics and properties of LIAM2

In this section, we report some of the Technical characteristics of the modelling framework. LIAM2 has been developed in Python, a powerful general programming language that is freely available under an open source licence. It has a large body of standard libraries, allows for C or C++ extension modules. Though development and testing has been done in a Windows environment, it runs on all major operating systems. Finally, there are 32-bit and 64-bit versions available.

LIAM2 internally reads and stores data in HDF5 format (HDF group). Hierarchical Data Format (HDF) is a set of libraries designed to store and organize large amounts of numerical data. It is freely available under an open-source license. Users will usually provide their data in the form of separate CSV files, and LIAM2 can convert it into HDF5, which can thereafter be used as a starting dataset for other simulations and model. Simulation results are also stored in HDF5 format for later processing. However, LIAM2 also has extensive reporting possibilities (cf. infra).

Defining a typical model requires describing objects (entities), the variables (fields) that describe them, the way they relate to each other (links) and how they behave over time (processes). Furthermore, the input-dataset and the simulation order have to be specified. All this is done in one file. LIAM uses the YAML-markup language (YAML), which stands for “YAML Ain't Markup Language”. It is a data serialization standard for all programming languages.

Like JAMSIM, LIAM2 can be used for both static microsimulation and dynamic microsimulation with dynamic ageing. In many models, static ageing techniques are being used in a starting phase to create or impute variables in the starting dataset in a way that is consistent with the model. These procedures are grouped in the ‘init-section’ of a LIAM2-model, and thus add to the starting dataset prior to the actual time-dependent simulation process. However, one can use this feature to develop static microsimulation models.

LIAM2 allows for mathematical functions, conditional functions, aggregate functions (including the mean, median, percentiles and Gini), temporal functions (lags, durations), stochastic changes (logits, Monte-Carlo simulations), life-cycle functions (birth, death, clone), and a matching function (a.k.a. ‘marriage market’), that can be applied to all object levels. Finally, LIAM2 is designed for extensive use of alignment by sorting on exogenous information of any number of dimensions (Li, 2011, chapter 4). A particularly interesting feature pertaining to this alignment is that the argument reflecting the risk can be one or more traditional logits pertaining to one alignment table, but it can also be a deterministic ranking number. This opens the possibility of implementing so-called “soft take’s and leave’s” in the model. By manipulating the score of individuals, the modeller can impose a priori have a high

---

3 To illustrate how large this can be, NASA uses HDF as the official format for all products derived by the Earth Observing System (EOS, see National Center for Atmospheric Research, 2012)
4 Note that this conversion needs to be done only once. After a HDF5 file is created, various models can use it for all sorts of simulations.
(“soft take”) of low (“soft leave”) probability of the event happening for a specific individual. This a priori order might not under all circumstances result in the simulated event to happen or not, since a) the modeller can choose to add a stochastic element to the score, and b) since some individuals with a low (high) risk may still (not) be selected, if the number of individuals meeting with the high a priori risk is lower (higher) than the number of transitions required by the alignment process.

By contrast to these “soft take and leave”, the alignment procedure itself also introduces an optional “hard take and leave” in the simulation of an event. These force inclusion or exclusion of objects with specified characteristics in the selection of the event. The individuals with variables specified in the “take” command will a priori be selected for the event. Suppose that the alignment specifies that 10 individuals should experience a certain event, and that there are 3 individuals who meet the conditions specified in the take. Then these 3 individuals will be selected a priori and the alignment process will select the remaining 7 candidates from the rest of the sample. The “leave” command works the other way around: those who match the condition in are a priori excluded from the event happening. Contrary to the “soft take and leave’s” which manipulate the a priori score but not the simulation of the events, the “hard take” and “hard leave” are absolute conditions, which means that the event will take place for all (take) or none (leave) of the individuals meeting these conditions.

Another important functionality of LIAM2 is that it respects the existing values of a variable in a certain period. Thus, if a value for an endogenous variable is available for one or more periods of time, LIAM2 will not overwrite this value but use it in the remainder of the model instead. This feature has been introduced to allow for retrospective modelling. Like many Western-European countries, Belgium has a Bismarckian-style public pension system where past labour market performance and earnings determine the pension benefit at retirement. When the simulation of pension benefits starts in the year 2001, then a number of retrospective variables are needed to introduce this information in the starting dataset. So, for example, retrospective simulations are started in 1947 and the labour market status of active individuals is available for these years. The retrospective part of the model MIDAS then simulates the pension claim that the individual builds up for every retrospective year, taking the required information as given. At the end of the retrospective simulation, so in 2001, the model reproduces the 2001-dataset (which is also in its input) including the endogenous variables that summarize the retrospective information. In short, the retrospective model produces the starting dataset for the prospective model.

Another application of this feature is that the output of one prospective run of the model can be used as input for another prospective run. For example, one could make one prospective run of the demographic module of the model MIDAS, and use it as input for a separate run without the demographic module. This not only introduces a considerable savings in runtime, but it also means that the result of the demographic module is identical for various simulation runs, which allows for a more straightforward identification of individual winners and losers of a specific policy measure.
3.1 A birds-eye view on a model

A typical model in LIAM2 discerns entities, fields, links, globals and macro’s.

- An ‘entity’ is an object with a unique identifier and is described by a set of attributes.
- Each of these attributes is called a ‘field’. There are three types of fields, booleans, integers and floats. The content of a field can be changed by a ‘process’. Note that a field may, but need not, be included in the starting dataset.
- A ‘link’ is a relation between an entity and one or more other entities. For example, mothers are linked to their children, spouses are linked together, and individuals are linked to households, and vice versa.
- A ‘global’ is a parameter that is not related to a specific entity. For example, the parameters that describe a social security system enter the model as globals.
- Finally, a ‘macro’ is a container that operates on the level of the fields. Macros may include a value, or a piece of code. The advantage of using macros is that they make the model code considerably easier to develop, interpret and maintain.

The text box presents a very rudimentary setup of a model (other than data importing functionality).

```plaintext
globals:
  periodic:
    - RETAGE: int

entities:
  person:
    fields:
      - age: int
      - y_in_retirement: {type: int, initialdata: false}
    processes:
      age: age + 1
      y_in_retirement: if(age >= RETAGE, age - RETAGE , -1)

simulation:
  processes:
    - person: [age, y_in_retirement]

input:
  file: base.h5

output:
  file: simulation.h5

start_period: 2002
periods: 10
```

This template describes a model that starts in 2002 and simulates 10 years, so up to 2012. The model and starting dataset contain only individuals as entity. The input dataset, named “base.h5” contains the identification number of the individual and his or her age. Output is
The model contains one parameter, RETAGE, which is the mandatory retirement age. Finally, the actual model consists of two processes only: simulating the age and ‘y_in_retirement’; the number of years that somebody has been in retirement, which equals the individuals’ age minus the mandatory retirement age.

A typical model consists of three main blocks: globals, entities and a simulation block. The first block lists the globals or parameters to be used in the model. This is where the parameter RETAGE is defined. The values of this parameter for all simulation years need to be included in the starting dataset base.h5. The second block is the entities block. It contains for every entity used in the model (individuals, households, firms, ...) the variables or fields, the links with other entities, the macros and processes. Our model contains only individuals as entities, so the fields ‘age’ and ‘y_in_retirement’ are defined in this block. Note that only ‘age’ is present in the dataset, and ‘y_in_retirement’ is not. This variable is thus void for any simulation period, until it is receives a value from a process. The main part of the entities block is that it lists the processes that pertain to the entity. In this case, there are two processes, of which the name coincides with the dependent variable. The first process increases age by 1 for every individual. The second process sets the field ‘y_in_retirement’ as the difference between the age of the individual and the mandatory retirement age. Or, it is the number of years in retirement, assuming that one cannot enter into retirement before the mandatory retirement age. This process also illustrates conditional statements, because ‘y_in_retirement’ has a value of -1 if the individual is younger than the mandatory retirement age RETAGE.

Note, finally, that the order in which processes are defined in this entities block is not relevant.

The third block is the simulation block. This contains the names and paths to the input- and output-datasets, the starting period and the number of simulation years. But the main part of this block is the processes-block, where the processes available in the model are listed in their order of simulation.

The above example is of course a gross simplification designed to illustrate the setup of a model in LIAM2. It contains no ‘init-section’ for static simulation, just one entity, no links, no macros and just two deterministic processes. More complex and illustrative examples can be found on the LIAM2 website (http://liam2.plan.be/).

3.2. Output and interaction

As said, LIAM2 writes the simulation results to a HDF5 output file. It can be analysed with the help of an HDF5 browser but can of course also be transformed to a format readable by the more traditional packages, such as Stata, SAS or R. However, output from a dynamic microsimulation model often is sizable and sometimes too big to handle. To overcome these problems and to allow for a rapid debugging and analysis of the simulation results, LIAM2 has several features. First of all, it can be run in interactive mode. All functions available in a model can also be used in the interactive console to inspect and analyze the simulation data.
This, combined with the possibility to put breakpoints anywhere in the model, opens extensive possibilities for debugging and data analysis. When reaching such a breakpoint or when reaching the end of the model, LIAM2 will go in interactive mode, thereby allowing the user to analyse the data by inputting any LIAM2 command.

Secondly, LIAM2 includes an extensive class of output routines, which allows any model to writing individual data as well as aggregates to CSV files. For example, one can create periodic CSV files containing a selection of variables describing certain individuals in any specific period. But one can also create pivot tables, and construct key output variables using means, percentiles and medians, but also more specific indicators such as at risk of poverty rates, Gini’s and Lerman-Yitzhaki decompositions) and have them append to a CSV file over all simulation years. All in all, this allows for a tailor-made range of output datasets that summarize the simulation data, and that can immediately be used to assess the simulation results, create tables and figures. It is therefore possible to develop all output variables and tables needed to assess the results of the model in that same model.

Third, LIAM2 is capable to use output from previous sessions as input. This means that separate applications can be developed that aim to assess and analyze the output of various other models developed in LIAM2.

4. Performance

The performance needs to cover two different aspects. The first dimension of performance is the size of the dataset the toolbox can handle. The second is the speed of simulation. The two are obviously connected, since it does not do much good to know that a simulation model can in theory simulate a large number of entities if the simulation speed is so low that it takes a decade to simulate one year. Inversely, an enormous speed of simulation is not much use if the toolbox has a very limited capacity.

Assessing the performance of LIAM2 is a tricky exercise because the performance entirely depends on the size of the model one simulates. But what is the size of a model? The number of lines is a flawed metric because some functions run orders of magnitude faster than others and the expressiveness of the language offered to modellers can vary considerably between tools. The only reliable method to compare LIAM2 performance to other similar tools would be to have a few different "representative" models run with the different tools (comparing using one model would not be enough since one model can run fastest using a particular tool and another model using another tool).

That said, given the extensive investment that would be required to create even one non trivial model and translate it to the different tools (assuming it can be translated at all), we have not done that exercise (yet) and can only discuss the performance we got for our own model with the current version. Also, one should bear in mind that LIAM2 is still being actively developed and there is still much room for performance improvements (both regarding memory usage and speed of simulation).
At this point, LIAM2 is being used on a regular basis in the development and use of the model MIDAS for Belgium (henceforth MIDAS_BE). This is a model of 3500 lines of code, 142 parameters, 132 (permanent) variables, 18 aligned processes, 14 CSV output files.

This model is being developed, tested and validated on a dataset of 300,000 individuals. The actual simulation runs are based on an expanded version of this dataset, being 2.2 million individuals or one-fifth of the Belgian population. The below Table 1 presents the performance of the most recent version of LIAM2 (64-bits executable, version 0.5 of October 29th, 2012).

### Table 1: performance of LIAM2 - simulation run from 2002 to 2060.

<table>
<thead>
<tr>
<th>Starting dataset of 300K individuals</th>
<th>Starting dataset of 2,200K individuals</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Total runtime:</strong></td>
<td><strong>Total runtime:</strong></td>
</tr>
<tr>
<td>31 minutes 18.58 seconds</td>
<td>3 hours 13 minutes 51.94 seconds</td>
</tr>
<tr>
<td>16792 individuals/s/period on average</td>
<td>17356 individuals/s/period on average</td>
</tr>
<tr>
<td><strong>top 10 processes:</strong></td>
<td><strong>top 10 processes:</strong></td>
</tr>
<tr>
<td>1- output_process: 9 m. 38.00 s. (31%)</td>
<td>1- output_process: 1 hr 14.65 s. (31%)</td>
</tr>
<tr>
<td>2- dead_procedure: 4 m. 23.51 s. (14%)</td>
<td>2- dead_procedure: 27 m. 50.41 s. (14%)</td>
</tr>
<tr>
<td>3- family_allowances: 2 m. 28.46 s. (8%)</td>
<td>3- family_allowances: 15 m. 32.62 s. (8%)</td>
</tr>
<tr>
<td>4- welfare: 1 m. 32.77 s. (5%)</td>
<td>4- matching_process: 9 m. 55.66 s. (5%)</td>
</tr>
<tr>
<td>5- matching_process: 1 m. 23.81 s. (4%)</td>
<td>5- welfare: 9 minutes 50.01 s. (5%)</td>
</tr>
<tr>
<td>6- ssc_tax_process: 55.47 s. (3%)</td>
<td>6- ssc_tax_process: 5 m. 19.80 s. (3%)</td>
</tr>
<tr>
<td>7- hours_process: 51.18 s. (3%)</td>
<td>7- hours_process: 5 m. 4.92 s. (3%)</td>
</tr>
<tr>
<td>8- inwork_process_working_t-1: 48.76 s. (3%)</td>
<td>8- inwork_process_working_t-1: 4 m. 42.53 s. (2%)</td>
</tr>
<tr>
<td>9- inwork_process_not_working_t-1: 42.92 s. (2%)</td>
<td>9- clean_empty: 4 m. 22.95 s. (2%)</td>
</tr>
<tr>
<td>10 - clean_empty: 37.41 s. (2%)</td>
<td>10 - inwork_process_not_working_t-1: 4 m. 4.12 s. (2%)</td>
</tr>
<tr>
<td>Total for top 10 processes: 23 minutes 22.30 seconds (75%)</td>
<td>Total for top 10 processes: 2 hours 26 minutes 57.66 seconds (76%)</td>
</tr>
</tbody>
</table>

This table shows that one simulation run from 2002 to 2060 on the 2.2 million individuals’ dataset currently takes 3 hours and 20 minutes. On the 300,000 individuals’ dataset, one run takes a bit over half an hour. The ‘individuals per second per period’ is however a better indicator of efficiency, since it corrects for the sample size, the number of periods and the increase of the sample size over time. This indicator shows that the simulation of the large dataset is a bit more efficient. This is because the time it takes to run some internal processes of LIAM2 are independent from the sample size, for example transforming the model into its
internal representation. Table 1 also presents the 10 processes in the model that takes up most of the time, and includes their specific runtimes, in time units but also proportionally to the overall run time. In this case, the differences between the runs on the large and ‘small’ dataset are minimal and mainly due to rounding differences. Thus, increasing the starting dataset from 300,000 to 2.2 million individuals of course increases overall simulation time, but increases the efficiency of the simulations a bit, and does not induce major changes to the proportional simulation times.

Not surprisingly, the output-generating process is the most CPU-consuming process; in both cases about 31% of overall run time. The output module creates CSV files and contains many routines needed to assess the simulation results of the model. A typical output dataset of MIDAS_BE has a size of roughly 200GB. This is too big to be conveniently processed by many other packages such as SAS or Stata. So post-simulation manipulation of the output dataset is currently avoided by developing all output variables and tables in LIAM2 itself. This integration of the output modules with the model however, makes the simulations (relatively) slow and cumbersome, because any new output variable or process requires that the model be simulated once more. Furthermore, this module creates a lot of output for all kinds of applications of the model; a majority of which that might not be needed for a specific application. LIAM2 therefore allows reading and manipulating the output dataset from previous simulations, thereby disentangling the actual simulation of the model with the analysis of the simulation results. Thus, various specific output modules could be simulated independently from the model and be tailor-made to specific needs and applications. There clearly is an important potential efficiency gain from this approach even though we have not used it in our own model yet.

Table 1 also shows that the ‘marriage market’ is not the most CPU-consuming procedure either (even though the matching function's runtime exhibit quadratic growth with the number of individuals to match for a period), but rather those parts of MIDAS that use considerable alignment and links between individuals or households.

Summarizing, the speed of simulation of LIAM2 is reasonably high and it scales well with the size of the starting dataset. Furthermore, the output module of MIDAS is by far the most time consuming part of the model and this suggests that there is a considerable potential efficiency gain from separating the output process from the actual model. However, even though models developed using LIAM2 are reasonably fast, they are currently probably several times slower than what could be achieved by "hard-coding" the model in the simulation software. But a model developed that way would take much longer to create, would be much harder to change and would not necessarily be faster. Also, as development on LIAM2 continues, the difference in runtime will considerably decrease.

Finally, pertaining to the data capacity, LIAM2 uses (very) approximately 20 bytes per variable-individual, so the 32-bits version (which can use up to 2 Gb of memory), can simulate approximately 100,000,000 variables-individuals. This means that, for example, if a model uses 100 variables (procedure local variables should not be counted), LIAM2 can
simulate up to a million individuals on the 32-bits version. Using the 64-bits version on a computer with, for example, 64 Gb of memory, one could use a dataset of approximately 32 million individuals using that same 100-variables model. This should also be improved in future versions of LIAM2.

Conclusions

This paper presents and discusses LIAM2, a new open source development tool for the development of static and discrete-time dynamic microsimulation models. It is a simulation framework that allows for comprehensible modelling and the straightforward use of various simulation techniques. LIAM2 is the spiritual successor of LIAM and is complementary to ModGen. The basic idea behind LIAM2 is to provide a development environment for the development of large discrete-time dynamic microsimulation models, and to stimulate collaboration and the exchange of tacit information between development teams through the use of a common development approach.

The paper presents the simulation properties of LIAM2. It shows the generic setup of a model through a simplified example. Finally, it discusses the performance of LIAM2 in terms of data capacity and simulation speed. The capacity of LIAM2 to handle large datasets is quite good, seeing that the current version of the model MIDAS is simulated in about 3 hours and 15 minutes starting from a dataset of no less than 2.2 million individuals.

References


HDF group, http://www.hdfgroup.org/ [05/114/2012]


YAML, See [http://yaml.org/](http://yaml.org/) [05/11/2012]