Report: GameTheorySim Class

Madis Ollikainen

September 8, 2015

This is a short overview of the GameTheorySim class I constructed in order to put all of the simulations under a united framework. The motivations for this was, that as the ideas for the simulations kept changing, the code started to branch out to much and it seemed that it would make more sense to put it under a unified class structure, such that further developments and changes could be made more easily. It also allows for a simple way of combining the existing code into new simulations. The class can be found in the *GameTheorySim* directory in our github repository. At the moment the class is just a class with a header and source file, it hasn't been made into a library. Maybe at some point I'll also do that (if it seems necessary). Also at the moment I have only implemented the *no-wealth-accumulation* (NWA) version of the code. But it shouldn't be to hard to add a wealth accumulating version.

1 Directories and files

1.1 *include* directory → headers

- *GameTheorySim.h* – The header file for the GameTheorySim class. Contains the class declarations. Also at some point in the debugging phase I put the multi-precision exponent definition into this header file. I still kept the multi-precision header in the include directory, but at the moment it isn't used.

- *my_mpEXP.h* – The multi-precision header file, which isn't in use at the moment as the multi-precision exponent function is defined as a class function of the GameTheorySim class and the the type definition of the multi-precision float *myMP_float* is done in the beginning of the class header. I've kept this header in the include directory just in case.

- *readcmd.h* – Two small functions *cmdOptionExists* and *getCmdOption* for easier reading of command line options. There is no associated *.cpp* file for this, the source code is in the *.h* file. These functions are used in *main.cpp* but not in the class definition.

- *timeEngine.h* – Header of a small class *TimeEngine*, that provides tools for timestamping. The GameTheorySim class has a field/variable (not sure about the terminology) of type TimeEngine, which is used for putting timestamps onto the output files.

1.2 *src* directory → source files

- *GameTheorySim.cpp* – The source file for the GameTheorySim class. It contains all of the code for the different class methods. At the moment the file is quite long. Maybe in future it might make sense to introduce sub-classes of the GameTheorySim class in order to make the code files shorter and better readable. As-well-as to make the code structure more logical.
- *main.cpp* – The main file for the current simulations. It take a command line input specifying, if we wish to conduct a Nash equilibrium simulation or the learning schema simulation. The specifications for the simulation parameter as-well-as whether the investment talents, investment caps and learning skills are distributed uniformly of by a Gaussian distribution, are given in the *config.conf* file. Optionally one can also give a command line argument, for naming the simulation.
- *timeEngine.cpp* – The source file for the TimeEngine class.

1.3 *obj* directory → object files

- If the code is compiled using the *makefile*, the class object are created into this directory.

1.4 Other files

- *makefile* – Used to compile the code in the *main.cpp* file.
- *config.conf* & *dummy.conf* – Configuration files used for simulation parameters as before. There just are a few more parameters now, then there were before.
- *gt4_sml.sh* – A script for running all four grouping schemas for the simple memory learning simulation. It is mostly a copy of the *doallfour.sh* script, with a few addition.

2 GameTheorySim class methods

Most of the class methods are quite obvious and just a class structured version/copy of the code we previously had existing to do the same thing. Thus just scanning trough the header file *GameTheorySim.h* and reading the short comments should explain what each of the methods do. Here I just go trough the larger methods, which combine the smaller one into larger pieces and thus allow for the simulations to be conducted.

- *initSociety_SML_NWA* & *initSociety_NashEQ_NWA* – Conducts the single society initialisation for either the simple memory learning (SML) or Nash equilibrium (NashEQ) schemas without wealth accumulation (NWA). Information for the initialisation is gotten from the configuration file.

- *runSociety_SML_NWA* & *runSociety_NashEQ_NWA* – Runs the time loop for the k-th society. The number of the society is given as an variable for the method, which is then used in the output files. In the case of the SML simulation a boolean indicating whether or not to print out the memories of the agents, is given.

- *runSimulation_NWA* – Runs the no-wealth-accumulation simulation. It takes a inputs a string which indicates the type of the simulation schema (either SML or NashEQ) and a boolean to specify the memory printing for the SML case.

3 config.conf

The configuration file now gives 14 parameters for the simulation. I also changed the order of the parameters a bit, to reflect the level of how fundamental the parameters are for the simulation (ie. number of ensembles and the agents are the two first parameters, while specifications for the learning skills are the two last). Just as before the comments in the *config.conf* file explain what each of the parameters does. Most of them are the same as before anyway. The only remarkable changes are the parameters specifying the distributions for the investment talent and cap, as well as the one for the learning skill.

- *muR* & *muW0* & *muXI* – Give the means for the distributions of the investment talent (R), investment cap (W0) and the learning skill (XI). If we wish any of these to be distributed uniformly, then we have to give the uniform value as a negative value.

- *sigmagR* & *sigmagW0* & *sigmagXI* – Gives the width of the Gaussian distributions of the investment talent (R), investment cap (W0) and the learning skill (XI).