Requirements Document
A Computer Program for the Computation of Equilibria of finite Games and other Game Theoretical Computations

Group 20

Per Frost
Marcus Lång
Markus Thurlin
Christer Hedberg
Christopher Engelbrektsson

Contents

1. Preface

1.1. Of this document

This document enumerates the requirements on a system for the computation of the equilibria of finite games and other game theoretical computations and sketches the anticipated architecture of the system in sufficient detail to serve as a model for its implementation. The expected reader (from which the actual reader can of course deviate significantly) is (i) a software developer intending to use the document as a model for an implementation of the system the requirements on which this document describes (ii) a person intending to determine whether a

system satisfies the requirements on the same.

1.2. Version history

1.2.1. Version I

This is the first version of the requirements document, created as its authors* intend to use it as a model for the implementation of a system, the requierments of which this document describes, for which a clear statement of the requirements on it and an outline of its expected architechture is beneficial.

* The authors are listed on the first page beneath the title.

2. Introduction

2.1. The users of the system and the problem the system solves for them

Certainly it would be absurd to suggest that a person who is educated in mathematics and game theory would have any greater interest in a computer program to solve finite games, having long since lost interest in the analysis of particular finite games, and being able to transform any finite game into problems solvable by means of existing numerical analysis packages.
Game theory is however useful not only to specialists, and the lack of intuitive software usable by the non-specialist greatly hinder its application to more trivial matters where it despite the insignificance of the problems would be of great use.

For these reasons it is intended that the system is to be used by persons who know of game theory, engineers and accountants alike, but do not really know how to apply it, and who would make use of it in their working environment, for example software engineers and people with a need to analyse situations that can be modelled by game theory but who either lack the resources to hire specialists or have been placed in positions in which they are expected to produce analyses of this type. With access to appropriate documentation of the software such a person could reasonably easily learn, for example, how to construct game trees, assign values to coalitions, and make other simple representations of situations which he has need to analyse. A specialist would probably not use this software because he would choose more complex approaches to the problems, however there is nothing that keeps him from doing so for simple quick results. Despite not knowing how to transform these representations into linear programming problems and in particular not knowing how to solve these resulting systems, these persons would only need to be able to interpret the solutions that the program itself outputs and then realise that they are valid and nontrivial to find.

2.2. Briefly on game theory

Game theory is a branch of applied mathematics. It studies the interactions between players or agents in a certain situation and is hence often used for economic purposes. Depending on the type of game or situation the agents will act differently to achieve their goal, a common goal in the economic context would be to maximise profit. The concept of finite game means that there is a definite beginning and end of the game as opposed to infinite games where these are unknown and there is a constant goal to keep playing. Our software will focus only

on finite games. A solution concept to a game is a condition that determines an equilibrium of a game or the predicted outcome of the game, the strategy that will be used by the players. An equilibrium is when competing forces are balanced and in general is of course something one does not wish to change from. The Nash equilibrium is a solution concept of a game with multiple players in which the agents individually have chosen a strategy and have nothing to gain by changing it. Subgame perfect equilibrium further builds on this and refers to a part of a larger game and their behaviour for this part represents Nash equilibrium. Another solution concept is correlated equilibria which is the distribution in a game where strategies are chosen at random by this distribution and no player wants to change from this strategy.

2.3. The main uses of the system

The main uses of the system would be the construction and editing of representations of finite games, the calculation of subgame perfect equilibria, Nash equilibria, and the development of other computer programs that need to make use of optimal strategies in finite games. Provided that the documentation is sufficiently good and includes examples of modelling real world data, the system would probably not be (at least purposelessly) used to support bad decisions.

2.4. Usage narratives

2.4.1. Usage narrative I

Consider for a moment Bob, a software engineer who works in a small software company with a total of about 40 employees. In his line of work tacit collusion (a kind of untold agreement between companies concerning a certain market strategy) frequently occurs, but is not viable for the company to hire an professional analyst to calculate the potential profits.

When dealing with tacit collusion Nash equilibria are highly relevant. While he may sometimes be able to calculate the Nash equilibria he will often be faced with games where he simply won't be able to apply the theory himself.

Bob has a program that can solve these kind of problems for him in an instant. He starts up the program, and inputs the data for his specific problem. Given that the data and the game is correct, the program then presents a solution to the problem. His company then gets a competitive advantage over other companies who do not make use of these types of calculations.

2.4.2. Usage narrative II

A student named Mary is studying game theory wishes to find the solutions to a horrid but mathematically elementary problem and could by means of the software find equilibria, model the game; and even, due to the ease of finding equilibria use the system to explain empirical data. She could then, at a computer in school or at home, use the program to do exactly this by simply entering the relevant data of the problem. In this context the program is used in parallel with studying the related theory. She is now familiar with the program and can after the course use it for when needed.

2.4.3. Usage narrative III

Joe runs a small company is needs to evaluate a contract, but it is so complicated that his experience gives him no insight into what incentives other actors will have to fulfil the terms. The system can then be represented game theoretically and analysed, perhaps in preparation for a more detailed analysis by a specialist. If he is in more equal negotiations he can make use of Mechanism design to make the contract results in the incentive to the actors that the parties intend. (Mechanism design in general terms means to try to design and control the game to acquire the desired outcome.)

2.5. The context/environement in which the system is to be used

The program will normally be used on modern office computers, due to the target user group being people in small to mid-sized companies with limited technology. A larger company would most likely hire a specialist instead. However given a high performance environment it may be used to solve more extensive and complex problems.

The system will be used both, in the form of its visual frontend as a tool for analysing data, the editing of game trees, normal form games and as an aid to the interpretation of equilibria, but will be insufficient to perform a complete analysis of any more involved problems, and will therefore by tightly integrated with numerical analysis software such as Matlab.

Although it would be nice the program will most definitely not be platform independent, for we wish to leverage the platforms that exist and to have their rather extensive userbases simply install it almost as if it were a plugin. The system will be designed to be used on Windows.

2.6. The scope of the system

Topic	In	Out
Constructing/editing representations of finite games	x	
Calculating subgame perfect equilibria		x
Calculating Nash equilibria	x	
Full support for uncommon solution concepts		x
Graphical editing of game trees, normal form games etc.		x
Calculating correlated equilibria	x	
Auction theory		x
User defined solution concepts		x
Infinite games		x

2.7. The main factors that need to be taken in to account when designing and building the system

* Due to the mathematical nature of the program its correctness is of great importance and a lot of effort should be directed to assuring it.

* While some understanding of the theory is necessary to use the software, it should be a goal to try and make the system understandable for as many people as possible.

* It should be a design goal to integrate the program with Matlab.

This serves to insure that even specialists may find features to construct game trees and process the resulting representations useful with whatever proprietary software they use.

2.8. Technologies and risks

The system would be written in C++ and make minimal use of ad-hoc numerical algorithms and instead making use of mature libraries and software such as GNU Multi-Precision Library(GMP), GNU Scientific Library(GSL) and Multiprecision Polynomial Solver(MPSolve). Each subprogram being essentially a reduction between problems in game theory, the totality which is to be possible to string together either by means of pipe(a command line function to stream data between modules), by the planned frontend, or by a software developer using the modules as black-box solvers.

Possible risks could be using and, as already mentioned, problems with connecting the different modules due to unfamiliarity with required third party software and libraries. Another possible risk is the the need for groupmembers to learn the required mathematics and the need to study relevant algorithms, which may lead, not necessarily to setbacks, but to that features that are particularly difficult to implement may go unimplemented.

3. Glossary

Strategy
A strategy is a way of deciding which moves to choose in a game.

Solution concept
A solution concept is a class of combinations of strategies that can be considered reasonable.

Information set
An information set is a set of games states between the player to move in them cannot distinguish.

Transition between game states
A transition between game states is move by a player.

Player
A player is an agent who makes moves in the game.

Game tree
A game tree is essentially the same thing as an extensive form game in which players make moves and progress through game states until the reach a game state in which noone is to move.

Normal form game
A normal form game is a game in which each player chooses their moves simultaneously.

Mixed extension of game
The mixed extension of a game is game in which the players chooses the probabilities with which the moves in the game of which the mixed extension is an extension and in which the payoff is the expected payoff.

Nash equilibrium
A Nash equilibrium is a set of strategies from which it is not advantageous to deviate lest others also deviate.

Standard input
Standard input is a file from through which a program can read input to it. Terminals and terminal emulators typically write keyboard input to them to this file. It exists on POSIX compliant systems.

Standard output
Standard output is a file to which a program can write output. What is written to this file is commonly displayed by a terminal or terminal emulator. It exists on POSIX compliant systems.

4. User requirements definition

4.1. Functional requirements

Define information sets
The system shall provide a method for the user to define information sets.

Define player sets
The system shall provide a method for the user to define player sets.

Define transition between elements
The system shall provide a method for the user to define transitions between elements.

Associate players with transitions
The system shall provide a method for the user to associate players with transitions.

Define preferences for game states/payoffs for game states
The system shall provide a method for the user to define preferences/payoffs for game states.

Tree-editing tool
The system shall allow for taking graphical input and converting into a game tree.

Calculate pure Nash equilibrium
The system shall, given a valid input, calculate the pure Nash equilibria and return it in a requested
output form.

Calculate Nash equilibria in the mixed extension of a game (mixed Nash equilibria)
The system shall, given a valid input, calculate the mixed Nash equilibria and return it as requested output form.

Calculate correlated equilibria
The system shall, given a valid input, calculate the correlated equilibria and return it as requested output form.

Calculate sub game perfect equilibria
The system shall, given a valid input, calculate the sub game perfect equilibria and return it as

requested output form.

4.2. Non-functional requirements

Performance
It should not take more than 15 seconds to find Nash equilibria in the mixed extension of a
normal form game with eight alternatives for each player.

Compatibility
It should run on Windows XP Service Pack 2.

Reliability
The program should not crash or hang more than once per 1000 problems solved.

Licensing
All external libraries or executables used should as far as possible be licensed under LGPL,
X11/MIT, UoI/NCSA or similar open source licenses.
Each respective author retains all rights to his source code.

Support
Help on how the program is used should be included. It must enable any person with some knowledge in computers to install and run the program.

4.3. Use cases

Use case: Calculate a Nash-equilibrium
Primary actor: User
Minimal guarantee: The user is notified if the calculation was successful or not.
Success guarantee: The user is given an output from the calculation of the Nash-equilibrium.

Main Success Scenario:
1. The user requests the calculation of Nash-equilibria.
2. The user inputs a representation of a finite game.
3. The user chooses what output format to be used.
4. The system calculates the Nash equilibrium.
5. The user receives a solution output.

Extensions
3a. The user enters an invalid format.
5a. The user aborts the calculation of the Nash-equiibrium before it has been completed and
therefore the user can not receive a solution output.

Use case: Construct a Game-tree
Primary actor: User
Minimal guarantee: The user is notified if the construction of the game tree was successful or not.
Success guarantee: A representation of a game tree is constructed.

Main Success Scenario:
1. The user defines sets of players
2. The user defines game states
3. The user creates transitions between game states

4. The user assigns game states to unique players.
5. The user assigns game states to information sets.
6. A representation of a game tree is constructed.
7. The user defines payoffs for each player at the subset of the game states from which there are no transitions.
8. The representation of the game tree is presented to the user.

Extensions
5a.1. The user has entered incorrect data.
2. The user is notified that this does not produce a game tree.

Use case: Create a transition between game states
Primary actor: User
Minimal guarantee: A user-specified transition between game states is defined.
Success guarantee: A user-specified transition between game states is defined.

Main Success Scenario:
1. The user selects an ordered pair of game states.
2. The user selects a name for the transition.
3. A transition between the game states in the ordered pair going by the specified name is created.

Extensions
1a. 1. The user selects a pair of game states such that the first element and the second element
coincides.
2. The user is notified that this is not a transition.
3. The user chooses another pair of game states.
2a. 1. The user enters an invalid name.
2. The user is notified that the name may not be used
3. The user chooses another name for the transition.

5. System architecture

5.1. Overview of the anticipated system architecture

It is anticipated that the system shall be ordered into three modules (i) a module to edit, create and view
extensive games, normal form games and solution concepts for them, the solution concepts including for extensive
games, subgame prefect equilibria, pure nash equilibria in the to to the extensive game corresponding normal form game,
mixed n$
game corresponding normal form game and correlated equilibria of the corresponding normal form game, for normal form
games,
the correlated equilibria, the pure nash equilibria and the mixed nash equilibria, (ii) a module to transform
extensive games into corresponding normal form games and (iii) a module to given, a representation of a game in a
format supported by the game tree editor, the format in which the representation is, a solution concept format
supported by the game tree editor in which the output is desired, and a solution concept, find the solution on the
desired form.

It is anticipated that the module (ii) will be implemented in the form of a software library used as an application
program interface by the other modules when it is needed, therefore its input and output formats are internal to the
system.

It is anticipated that that module (iii) will be implemented as a collection of computer programs that read
representations of games from standard input and print representations of them to standard output, and that they will
use of a linear algebra submodule (iv).

It is anticipated that the linear algebra submodule will be an application programming interface developed by others
satisfying the properties detailed in the section of this document pertaining to licensing.

6. System requirements specification

6.1. Functional requirements

Functional requirements

Define information sets
The system shall provide a method for the user to define information sets.
This shall be tested by having any person define an information set.

Define player sets
The system shall provide a method for the user to define player sets.
This shall be tested by having some person define an information set.

Define transition between elements
The system shall provide a method for the user to define transitions between elements.
This shall be tested by having some person define a transition between some pair of two elements.

Associate game states with players
The system shall provide a method for the user to associate players with transitions.
This shall be tested by having some person associate a player with some game state.

Define preferences for game states/payoffs for game states
The system shall provide a method for the user to define preferences/payoffs for game states.
This shall be tested by having some person associate a player with a game state.

Tree-editing tool
The system shall allow for taking graphical input and converting into a game tree.
The tool will provide methods for creating the graphical input and use the required defined elements to create a game tree.
The tree shall constantly be displayed regardless of its completeness.
The game tree shall be stored as tttt and shall not be larger than tttt.
The user shall be able to write this tree to disk.

Calculate pure Nash equilibrium

The system shall, given a valid input, calculate the pure Nash equilibria and return it in a requested output form.
The time from when the functions receives the input to the time when it returns the result shall not exceed ten seconds if the game
has two players and they have no more than eight actions each.
The user shall have the option of storing the result to disk.

Calculate Nash equilibria in the mixed extension of a game (mixed Nash equilibria)
The system shall, given a valid input, calculate the mixed Nash equilibria and return it as requested output form.
The user shall have the option of storing the result to disk.

Calculate correlated equilibria
The system shall, given a valid input, calculate the correlated equilibria and return it as requested output form.
The time from when the functions receives the input to the time when it returns the result shall not exceed ten seconds if the game
has two players and they have no more than eight actions each.
The user shall have the option of storing the result to disk.

Calculate sub game perfect equilibria
The system shall, given a valid input, calculate the sub game perfect equilibria and return it as requested output form.
The time from when the functions receives the input to the time when it returns the result shall not exceed ten seconds if the game
has two players and they have no more than eight actions each.
The user shall have the option of storing the result to disk.

6.2. Non-functional requirements

Performance
It should not take more than 15 seconds to find Nash equilibria in the mixed extension of a
normal form game with eight alternatives for each player.

Compatibility
It should run on Windows XP Service Pack 2.

Reliability
The program should not crash or hang more than once per 1000 problems solved.

Licensing
All external libraries or executables used should as far as possible be licensed under LGPL,
X11/MIT, UoI/NCSA or similar open source licenses.
Each respective author retains all rights to his source code.

Support
Help on how the program is used should be included. It must enable any person with some
knowledge in computers to install and run the program.
The help should include basic descriptions off the functions available to the user.

7. System evolution

7.1. Fundamental assumptions about of the system

The system is based on the fundamental assumptions that Microsoft Windows will be installed on the computer on which it is to run, that Matlab will be installed and that the user has knowledge of game theory.

7.2. Anticipated future changes to the system

Due to the development of software to make use of graphics processing units and as the module, that the module
(iii) is computationally intensive and will use much vector arithmetic it is anticipated that the system could be
improved by replacing the linear algebra submodule of (iii) with one that makes use of the graphics processing unit.

As computing power increases in the future, the limit on how big systems that can be solved can be raised, and support for more resource demanding game theory problems can be added.

8. Appendix I, Some of the underlying mathematics

A 3-tuple (P, S, F) is a game if (and since this is a definition precisely if), P is a set of players, F is a set of
strategies^1 is an |S|-tuple of payoff functions, which are functions from the cartesian product of the strategy set
to the reals.

^1 Not necessarily a pure strategy, for the mixed extension of a game is again a game.