MSE in Mathematica v.1.0.0

Content

Introduction

This code was designed by Theodore Chronis in collaboration with Denisa Mindruta.
The code builds upon Jeremy Fox's theoretical work on the "pairwise maximum score estimator" (Fox 2010; Fox 2016) and the original Match Estimation toolkit (Santiago and Fox, 2009) which can be downloaded from http://fox.web.rice.edu/
To understand the present code the user needs to be familiar with the maximum score estimator and formal matching games. To ease the exposition, this documentation file and the code itself follow closely the terminology used by Jeremy Fox. Unless stated otherwise, please **refer back to the original sources** for definitions and technical details.

The main references are:

- Fox J. Forthcoming. "Estimating Matching Games with Transfers". Quantitative Economics. Last accessed from: http://fox.web.rice.edu/working-papers/fox-matching-maximum-score.pdf
- Fox J, (2010). "Identification in matching games". Quantitative Economics 1: 203–254
- Santiago D, and J Fox, (2009). "A Toolkit for Matching Maximum Score Estimation and Point and Set Identified Subsampling Inference". Last accessed from http://fox.web.rice.edu/computer-code/matchestimation-452-documen.pdf

Related work:

- For a comparison of the maximum score estimator with logit/probit please see:

Mindruta, D., Moeen, M. and Agarwal, R. (2016), A two-sided matching approach for partner selection and assessing complementarities in partners' attributes in inter-firm alliances. Strat. Mgmt. J., 37: 206–231.

- In another example, Chatain and Mindruta (2017) have used the estimator for an empirical application that builds on the value-based approach in Strategy:

Chatain, O. and Mindruta, D. (2017), Estimating value creation from revealed preferences: Application to value-based strategies. Strat. Mgmt. J. doi:10.1002/smj.2633

The current version of the code can solve three general type of problems: 1) it estimates the coefficients of the match payoff function in a many-to-many matching situation by maximizing the matching maximum score objective function as introduced by J Fox (2010, 2017) 2) it produces the "best" matches in a market when the payoff function and the agents' characteristics are known, via a linear programming approach which maximizes the sum of payoffs in a market 3) it also allows for manipulating data to generate counterfactual scenarios where some agents are removed from the market and new matches are generated, yielding various outcomes of interests to researchers (not yet released).

Below we present some relevant technical aspects and we highlight the most notable differences relative to the toolkit provided by J Fox.

Technical aspects & Improvements

1. Data structure was redesigned in order to accommodate more easily big data sets and to increase the speed of execution.
2. For empirical researchers, a notable difference lies in the flexibility allowed by the slightly different way of constructing the input data, which we call “precomputed data”.In this version, a data entry tuple takes the form {*m,i,j, match*}, where *m* is the market number, *i* and *j* are the indexes of the upstream and downstream agents, and *match* takes values of 1 or 0 depending on whether agents *i* and *j* are matched in the data or not. The tuples {*m,i,j,match*} consist of all possible pairwise combinations between the upstream and downstream agents in a given market *m*. That is, in each market, we list all upstream agents, their observed partners, and all hypothetical partners. Further, to each touple {*m,i,j, match*} of observed and counterfactual matches, we join the components of the payoff function, here named “distance attributes”. In the code these are the components of distanceMatrices. A distance attribute between any pair of upstream-downstream agents (matched or counterfactuals) can be of the following type: 1) multiplications (including those of higher order) of the upstream and downstream agents’ specific attributes as in a man i’s years of schooling *x* and a woman j’s years of schooling *y* (e.g. $x^n y^n$ where n>=1) and 2) a pair-specific characteristic, such as the geographic distance between man i and woman j, or the number of years they have known each other prior to the matching event under consideration, or whether they share the same religion, etc.

   We found that data entry in this format allows for more flexibility because researchers can generate all possible combinations (pairings) between the upstream and downstream agents and calculate any pair-specific “distance attributes” in the software of their choice and just upload the data in Mathematica to obtain the coefficient estimates of the payoff function.

   Uploading data in the precomputed format can be cumbersome for large data sets. Given the trade-offs between writing an efficient code for handling memory constraints (whose scope is to maximize the use of memory for handling big data in an efficient way) and increasing the execution speed (whose scope is to decrease the running time, particularly when it comes to obtaining the coefficient estimates and their confidence intervals), we have introduced two running modes, called Memory and Speed. The user can switch between them depending on the problem.

3. The code is designed to deal with the more general problem of many-to-many matching inequalities. For the moment, the code accommodates match payoffs that are additively separable across the individual one-to-one matches. Future extensions to other payoff functions are encouraged. As explained, payoff functions under the form of polynomials of higher order can be easily be implemented by including the higher order multiplications among the terms of the distanceMatrices. However, the current optimization approach will have to be modified for more complex payoff functions.

4. As an optional feature, we experimented with other optimization methods than the Differential Evolution, which is recommended and implemented by J Fox, and found

that Particle Swarm Optimization (PSO) behaves also well. The default remains the Differential Evolution method.

5. A lot of work has been done in the background to improve the speed of execution. Example: dataset of 25 Markets with 50 upstream and 50 downstream each producing 30625 inequalities in 1-1 relationships. pointIdentifiedCR for ssSize = 3 (sample size); numSubsamples = 50; alpha = 0.05; completed in 30' (tested in a i7 3600 series).

6. We mostly worked with the point identification, although Jeremy Fox's original routines include the possibility of generating set-identified confidence intervals. Future revisions of the code will include this possibility.

7. An inequality is true if it includes the ">=" condition. Equality is assumed in the Machine's precision sense. Typically if two quantities have a difference less than 10^(-15) they are assumed as equal. Using different precision may lead to slightly different results due to the randomized character of the maximization process.

8. Inequalities follow theoretical proofs proposed by J Fox, under the same assumptions (e.g. an agent's quota is inferred from the observed number of matches she participates in, and it remains unchanged when writing the the matching maximum score inequalities). The file inequalities.m shows how the inequalities of the objective function are formed. Before proceeding with the estimation, the user can first run this module of the code to observe how inequalities are formed in a specific empirical context (dataset).

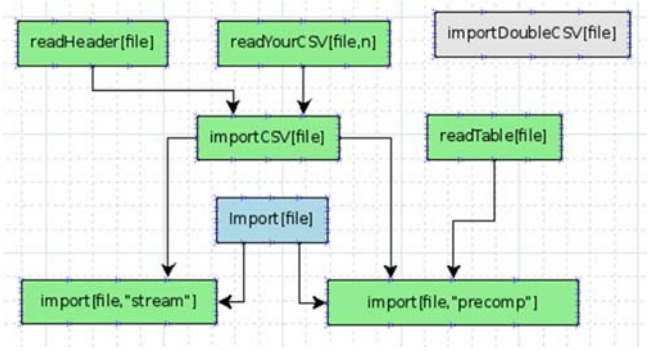
9. We have observed in some datasets that if we shuffle the ordering of the columns where the "distance attributes" are stored (i.e. {1,2,3}->{2,3,1}), then the maximize routine may return a different solution. This occurs because by switching the columns, in reality we rotate the search space and in that way we affect the searching path within the optimization method. Since the objective function has many local maxima it is plausible to end up with a different maximum. This could be a problem if two researchers are collaborating and for some reason they construct the datafile differently (not agreeing on the column order). For this reason, we have inserted an ad-hoc rule for selecting the column order. We calculate the standard deviation of each column attribute, and we sort the columns in a decreasing order of the standard deviation value. This is an ad-hoc rule which users can turn on and off. permuteinvariant = True enables the rule (The default value is false).
   When enabled, this rule should be used throughout the code, both for obtaining the solution to the maximization problem and the confidence intervals. Please remember this is an ad-hoc rule, and users should always do robustness checks to ensure their results are based on the global maximum. Increasing the number of inequalities (i.e. working with larger data sets) and using set identification confidence intervals are potential approaches when users suspect multiple maxima that are not near to each other in the specific data.

Modular Code

The code library contains multiple modules that each can be further improved and extended to other problems. The reason for not having a single file that contains all the code for it to be maintainable. See the corresponding diagram for a full description of the modules and relationships between modules.

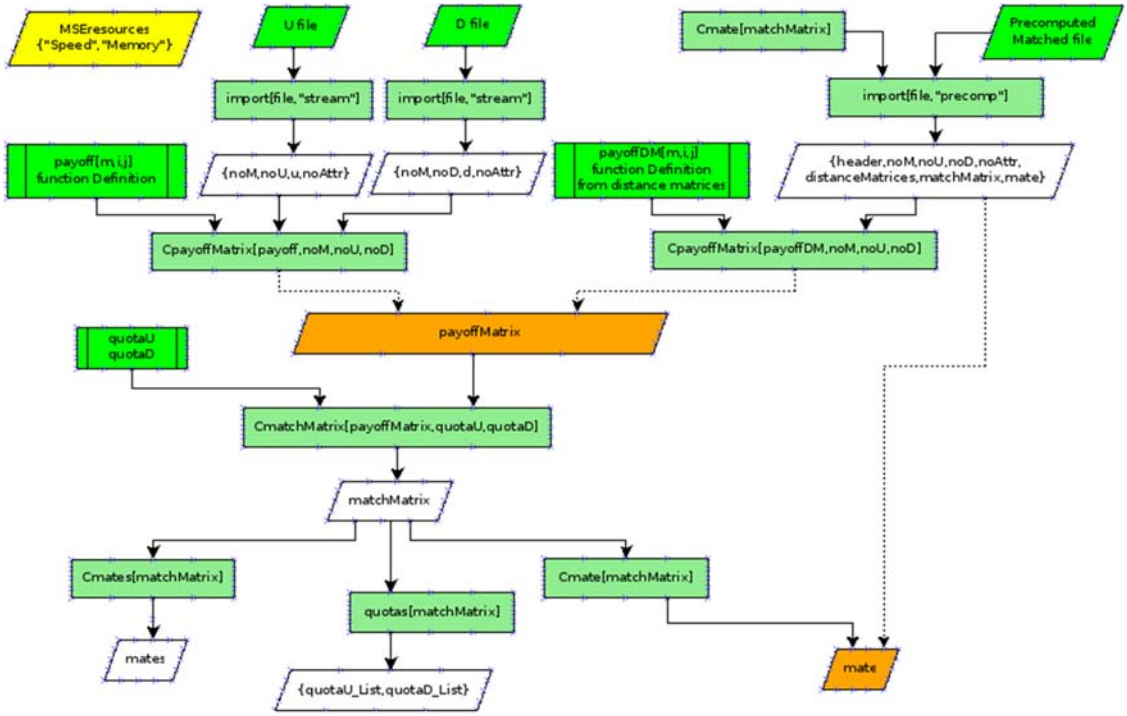
Use case Flows

Several methods of importing data have been created and are shown briefly in the diagram below. In almost all cases, the data files to import will contain header (there are exceptions but for simplicity we will not describe those inside this document).



Flow 1: Find the optimum match for all agents in a market, given a specific payoff function

In this flow we suppose that we have imported files that contain data (stream or precomputed). In the case of precomputed we already know the payoffMatrix.
We set the quotas per Market / Stream / Specific member that put constraints on how many matches each individual can create.

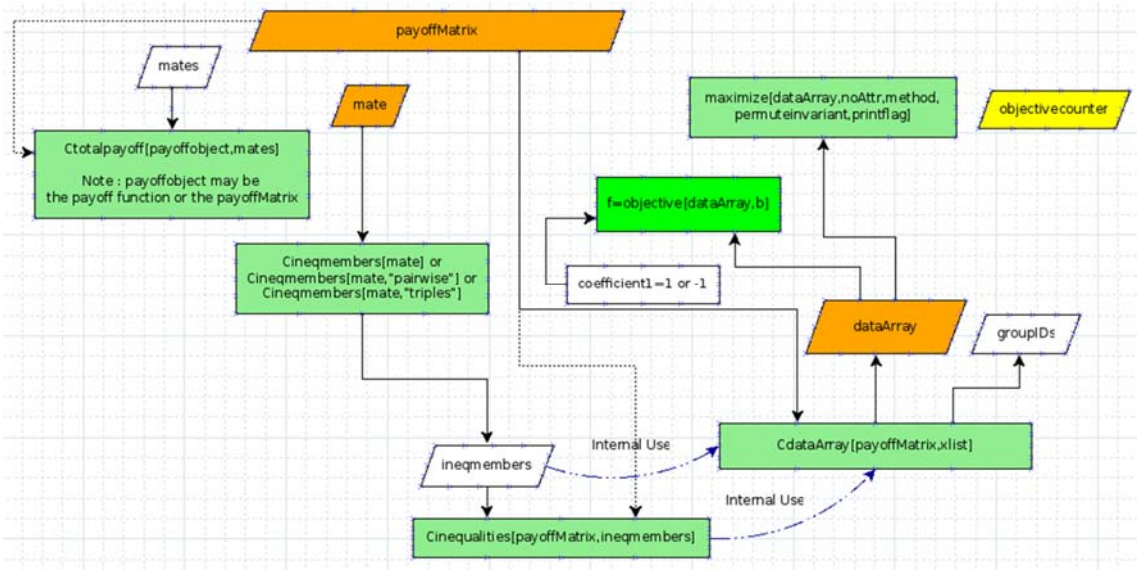


Flow 2: Maximize the parametric payoff function based on the observed matches

Description

In this flow we suppose that the dataArray is already created. The aim is to find the parameters of the payoff function (coefficients, or the b-list in the code) in order to maximize the number of satisfied inequalities.

Diagram



Examples

docs/precomputed_abstract.nb (ends on forming the dataArray - abstract values)
docs/precomputed_numeric.nb (full flow)

Flow 3: Calculate the confidence intervals around the estimated payoff function

Description

In this flow we suppose that we have calculated the parameters of the payoff function that maximizes the number of satisfied inequalities. We aim to test the stability of this solution through calculating the confidence intervals.
The process we are following is the known one, omitting markets - finding the new parameters that maximize the new dataArray and doing it several times in the way it is described in the code and in the original documentation by Santiago and Fox (2009). Notice that the original code allows for calculating confidence intervals by subsampling matches from one single market. We have not provided examples for these situations, but this code is flexible enough to accommodate these situations.

More Flows… The freedom to create your own flow.

Since the code is modular, that is, it consists of autonomous functions, one can combine them in a free way that makes sense. Researchers can define new functions that will allow for data manipulation at any point in the flow, in order to study for various patterns.

An example would be to remove some pairs of agents from the upstream and downstream sets and study how the Market would evolve without them. This is interesting especially in the many to many relationships (see the general problem 3 that we alluded to in the beginning of the document).

If you are interested in building your own flow, please send an e-mail to t.n.chronis@gmail.com to guide you through.

Appendix 2: Code style in Mathematica

We use functional programming to have Mathematical clarity and also assure performance. Packed Arrays are used wherever possible to save as much memory as possible (especially for big datasets) and to boost the performance further.

The default memory model is set for Speed but in extreme situations the model Memory is used. When setting Memory instead of Speed , the ineqmember, inequalities are compressed and decompressed to save memory while running across the Markets. When running on Market(i) , it decompresses it and then compresses it again when finished. Let's assume we have 20 markets. With this approach, only 5% is of the data is uncompressed,

leading to tradeoffs between memory saving and speed. More details on the functions that accomplish this can be found in the Functions-reference document found here:

https://github.com/tchronis/MSE-Mathematica/blob/master/doc/functions-reference.docx

Appendix 3: Testing all functions

In order to be certain that this software will always work as it is designed, on every Mathematica version 9,10,11 (including major and minor releases) testing is absolutely necessary.

All the testing is being done inside the folder **testing/**

Going through the tests inside each document is valuable to understand exactly what to expect as an output running for various inputs. For example:

Input : Cineqmembers[{{{{1}, {2}, {3}}, {{1, 2, 3}, {}, {1, 2, 3}}}}]

Output: {{{{{1, 1, 1}, {1, 1, 2}, {1, 1, 3}}, {{1, 1, 1}, {1, 1, 2}, {1, 1, 3}, {1, 3, 1}, {1, 3, 2}, {1, 3, 3}}, {{1, 3, 1}, {1, 3, 2}, {1, 3, 3}}}, {{{1, 2, 1}, {1, 2, 2}, {1, 2, 3}}, {{1, 1, 1}, {1, 1, 2}, {1, 1, 3}, {1, 3, 1}, {1, 3, 2}, {1, 3, 3}}, {{1, 2, 1}, {1, 2, 2}, {1, 2, 3}}}}}

Appendix 4: Optimization Methods

The first step to achieve an optimization is to define the objective function.

Then the Optimization methods take over and transverse the space to find the point where the function achieves a global maximum.

The methods that are being used are heuristic so the solution sometimes is sub-optimal, especially when working with data that do not obey the matching patterns assumed by the model (quota, payoff function, etc). Optimization (maximize) can be achieved using several Methods shown in the diagram below: