Matlab Toolbox *MatTuGames* Version 1.9.2

Contents:

1. Introduction

The game theoretical *Matlab* toolbox *MatTuGames* provides more than 700 functions for modeling, and calculating some solutions as well as properties of cooperative games with transferable utilities. In contrast to existing Matlab toolboxes to investigate TU-games, which are written in a C/C++ programming style with the consequence that these functions are executed relatively slowly, we heavily relied on vectorized constructs in order to write more efficient Matlab functions. In particular, the toolbox provides functions to compute the (pre-)kernel, (pre-)nucleolus, anti (pre-)kernel, and modiclus as well as game values like the Banzhaf, Myerson, Owen, position, Shapley, solidarity, and coalition solidarity value and much more. In addition, we will discuss how one can use *Matlab's Parallel Computing Toolbox* in connection with this toolbox to benefit from a gain in performance by launching supplementary Matlab workers. Some information are provided how to call our *Mathematica* package *TuGames* within a running Matlab session.

2. Getting Started

In order to get some insight how to analyze a cooperative game, a so-called transferable utility game, with the Game Theory Toolbox *MatTuGames*, we discuss a small example to demonstrate of how one can compute some game properties or solution concepts, like convexity, the Shapley value, the (pre-)nucleolus or a pre-kernel element.

For this purpose, consider a situation where an estate is insufficient to meet simultaneously all the debts/claims of a set of claimants, such a situation is known in game theory as a bankruptcy problem. The problem is now to find a fair/stable distribution in the sense that no claimant/creditor can find an argument to obstruct the proposed division to satisfy at least partly the mutual inconsistent claims of the creditors. In a first step, we define a bankruptcy situation while specifying the debts vector and the estate that can be distributed to the creditors. We restrict our example to a six-person bankruptcy problem with a debts vector given by

```
>> d = [40.0000 32.0000 11.0000 73.3000 54.9500 81.1000];
```

and an estate value which is equal to

```
>> E = 176;
```

We immediately observe that the estate `E` is insufficient to meet all the claims simultaneously. It should be obvious that with these values we do not have defined a cooperative game, however, this information was enough to compute a proposal how to divide the estate between the creditors. A fair division rule which is proposed by the Babylonian Talmud, is given by

```
>> tlm_rl=Talmudic_Rule(E,d)
>>
tlm_rl =

20.0000 16.0000 5.5000 48.3500 30.0000 56.1500
```

However, this distribution rule does not incorporate the coalition formation process. Thus, we might get a different outcome when we consider the possibility that agents can form coalitions to better enforce their claims. This means, we have to study the corresponding cooperative game. This can be constructed while calling the following function

```
>> bv=bankruptcy_game(E,d);
```

Having generated a game, we can check some game properties like convexity

```
>> cvQ=convex_gameQ(bv)

cvQ =

  logical

   1
```

The returned logical value indicates that this game is indeed convex. This must be the case for bankruptcy games. In addition, we can also verify if the core of the game is non-empty or empty. To see this one needs just to invoke

```
>> crQ=coreQ(bv)

crQ =

  logical

   1
```

which is answered by affirmation. This result confirms our expectation, since each convex game has a non-empty core.

After this short introduction of game properties, we turn our attention now to some well known solution concepts from game theory. We start with the Shapley value, which can be computed by

```
>> sh_v=ShapleyValue(bv)

sh_v =

   23.5175   18.7483    6.4950   44.3008   33.3317   49.6067
```

A pre-kernel element can be computed with the function

```
>> prk_v=PreKernel(bv)

prk_v =

   20.0000   16.0000    5.5000   48.3500   30.0000   56.1500
```

which must be identical to the distributional law of justice proposed by the Talmudic rule. Moreover, it must also coincide with the nucleolus due to the convexity of the game. To see this, let us compute first the nucleolus and in the next step the pre-nucleolus

```
>> nc_bv=nucl(bv)

nc_bv =

   20.0000   16.0000    5.5000   48.3500   30.0000   56.1500

>> pn_bv=PreNucl(bv)

pn_bv =

   20.0000   16.0000    5.5000   48.3500   30.0000   56.1500
```

We observe that both solutions coincide, which must be the case for zero-monotonic games. To check that these solutions are indeed the pre-nucleolus can be verified by Kohlberg’s criterion

```
>> balancedCollectionQ(bv,pn_bv)

ans =

  logical
```

```
   1

>> balancedCollectionQ(bv,nc_bv)

ans =

  logical

   1
```

In order to verify that the solution found is really a pre-kernel element can be done while typing

```
>> prkQ=PrekernelQ(bv,prk_v)

prkQ =

  logical

   1
```

Furthermore, with the toolbox we can also compute the modiclus of the game, which takes apart from the primal power also the preventive power of coalitions into account.

```
>> mnc_bv=Modiclus(bv)

mnc_bv =

   22.5067   17.7567    7.4533   41.8600   37.1100   49.3133
```

Checking this solution can be established while invoking a modified Kohlberg criterion.

```
>> modiclusQ(bv,mnc_bv)

ans =

  logical

   1
```

The return value is a logical one, hence the solution is the modiclus. For bankruptcy game we can rely on the computation of the anti pre-nucleolus as a simple cross-check to figure out that the solution is correct (cf. Meinhardt 2018c).

```
>> apn_bv=Anti_PreNucl(bv)

apn_bv =

   22.5067   17.7567    7.4533   41.8600   37.1100   49.3133
```

We observe that both solutions coincide, hence this gives additional evidence that the computation of the modiclus was correct. Moreover, for the class of convex games the modiclus must belong to the core, which can be examined through

```
>> belongToCoreQ(bv,mnc_bv)

ans =

  logical

   1
```

However, if this should still not be enough evidence, then we can refer to the axiomatization of the modiclus, which is characterized by SIVA, COV, EC, LEDCONS, and DCP, whereas DCP can also be replaced by DRP (cf. Meinhardt 2018c).

Apart from SIVA (Single Valuedness), the toolbox can examine COV

```
>> COV_mnc_bv=COV_propertyQ(bv,mnc_bv,'','','MODIC')

COV_mnc_bv =

  struct with fields:

      covQ: 1
    sol_v2: [23.5067 18.7567 8.4533 42.8600 38.1100 50.3133]
       sgm: [23.5067 18.7567 8.4533 42.8600 38.1100 50.3133]
        v2: [1x63 double]
         x: [22.5067 17.7567 7.4533 41.8600 37.1100 49.3133]
```

as well as EC

```
>> ECQ_mnc_bv=EC_propertyQ(bv,mnc_bv,'MODIC')

ECQ_mnc_bv =

  struct with fields:
```

```
    propQ: 1
        y: [22.5067 17.7567 7.4533 41.8600 37.1100 49.3133]
        x: [22.5067 17.7567 7.4533 41.8600 37.1100 49.3133]
```

and LEDCONS

```
>> [LEDC_mnc_bv, LEDCGPQ_mnc_bv]=Ledcons_propertyQ(bv,mnc_bv,'MODIC')

LEDC_mnc_bv =

  struct with fields:

    ledconsQ: 1
        rgpq: [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
    ledpropQ: [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]


LEDCGPQ_mnc_bv =

  1x4 cell array

    {'vS'}    {2x62 cell}    {'impVec'}    {1x63 cell}
```

to finally check DCP

```
>> DCP_mnc_bv=DCP_propertyQ(bv,mnc_bv,'MODIC')

DCP_mnc_bv =

  struct with fields:

    propQ: 1
       xQ: 1
        y: [22.5067 17.7567 7.4533 41.8600 37.1100 49.3133 22.5067 17.7567 7.4533 41.8600 37.1100 49.3133]
        x: [22.5067 17.7567 7.4533 41.8600 37.1100 49.3133]
```

and DRP

```
>> DRP_mnc_bv=DRP_propertyQ(bv,mnc_bv,'MODIC')

DRP_mnc_bv =
```

```
  struct with fields:

    propQ: 1
       xQ: 1
        y: [22.5067 17.7567 7.4533 41.8600 37.1100 49.3133 22.5067 17.7567 7.4533 41.8600 37.1100 49.3133]
        x: [22.5067 17.7567 7.4533 41.8600 37.1100 49.3133]
```

By this example, we observed that the axiomatization of the modiclus was satisfied, from we which can conclude that the modiclus of the game was found by this evaluation. Of course, the toolbox offers in addition routines to examine the axiomatization of the pre-nucleolus, pre-kernel, anti pre-nucleolus, anti pre-kernel, modified as well as proper modified pre-kernel, and Shapley value.

Moreover, the toolbox offers to the user the possibility to create several game class objects to perform several computations for retrieving and modifying game data with the intention to ensure a consistent computation environment. Hence, these classes should avoid that some data from a different game are used or that game data are unintentionally changed, which allow the user to concentrate on the crucial aspects of analyzing the game instead of dealing with the issue of supplying the correct game data. Such a class is, for instance `TuSol`, which executes several computations in serial for retrieving and storing game solutions. A class object, let us call it `sclv`, is created by calling `TuSol` with at least one argument, that is, the values of the characteristic function. The other two input arguments can be left out. However, if they are supplied, then the second specifies the game type, for instance `cv` for the class of convex games. Whereas the last argument specifies the game format, which is for the discussed example `mattug` to indicate that the coalitions are ordered in accordance with their unique integer representation to carry out some computation under MatTuGames.

```
>> scl_bv=TuSol(bv,'cv','mattug');
```

Having created the class object `scl_bv`, one can invoke a computation of getting results for some selected solution concepts while executing

```
>> scl_bv.setAllSolutions

ans =

  TuSol with properties:

        tu_prk: [20.0000 16.0000 5.5000 48.3500 30.0000 56.1500]
        tu_prn: [20.0000 16.0000 5.5000 48.3500 30.0000 56.1500]
       tu_prk2: []
         tu_sh: [23.5175 18.7483 6.4950 44.3008 33.3317 49.6067]
       tu_tauv: [24.0807 19.2646 6.6222 44.1279 33.0809 48.8237]
        tu_bzf: []
       tu_aprk: [22.6550 17.9050 7.3050 41.8600 37.1100 49.1650]
     prk_valid: 1
     prn_valid: 1
```

```
    prk2_valid: 0
    aprk_valid: 1
      tuvalues: [1x63 double]
        tusize: 63
     tuplayers: 6
        tutype: 'cv'
        tuessQ: 1
      tuformat: 'mattug'
          tumv: 176.0000
         tumnQ: 0
          tuSi: [62 61 59 55 47 31]
          tuvi: [0 0 0 0 0 0]
        tustpt: []
```

which stores apart of the solution concepts also some important data of the game. This class object can then be used, for instance, to determine the modified pre-kernel of the underlying game

```
>> mpk_bv=scl_bv.ModPreKernel

mpk_bv =

   22.6550   17.9050    7.3050   41.8600   37.1100   49.1650

>> mpkQ_bv=scl_bv.ModPrekernelQ(mpk_bv)

mpkQ_bv =

  logical

   1
```

or the proper modified pre-kernel through

```
>> pmpk_bv=scl_bv.PModPreKernel

pmpk_bv =

   22.2100   17.4600    7.7500   41.8600   37.1100   49.6100

>> pmpkQ_bv=scl_bv.PModPrekernelQ(pmpk_bv)

pmpkQ_bv =
```

```
logical

 1
```

or much more. For a deeper discussion of the function set provided by the toolbox consult the Manual or type `help mat_tug` to get a short overview.

3. Custom Installation

To install the toolbox, we recommend a custom installation. Having downloaded the .mltbx file, navigate to it within the Matlab file explorer, double-click on the mltbx file `mat_tugV1d9d2.mltbx` and click “install”. Alternatively, right-click on the .mltbx, and click “Install.”

Additional instructions can be found at the URL:

- mltbx

The mltbx file `mat_tugV1d9d2.mltbx` is provided at

- mltbx-file

4. Requirements

This release of *MatTuGames* was developed and tested using *Matlab R2024b* and earlier releases. A set of functions use the *Optimization Toolbox* and the *cdd-library* by *Komei Fukuda*, which can be found at the URL:

- CDD

as well as the Matlab interface to the cdd solver *CDDMEX*:

- CDDMEX

Alternatively, in order to get even full scope of operation of the graphical features, one can also install the *MPT3* toolbox that can be downloaded from

- MPT3

which ships with *CDDMEX*. We strongly recommend the user to apply the *MPT3 toolbox*, in particular of using the graphical features of our toolbox.

For the computation of the pre-kernel and related solutions the *SuiteSparse* for *Matlab* is recommended that can be got from the URL

- SuiteSparse

If you do not want to use *SuiteSparse*, then replace the function `qr_dec` by `pinv` in all functions for the pre-kernel and related solutions. The same argument applies for the function `qrginv`. It should be noted that this may cause accuracy issues with the consequence that the result is incorrect.

To run the toolbox even in parallel mode, *Matlab's Parallel Computing Toolbox* is needed.

For connecting the *Mathematica* Package *TuGames*, the *Mathematica Symbolic Toolbox* is required, which can be found under the URL:

- Mathematica Symbolic Toolbox

whereas *TuGames* Version 3.1.4 can be downloaded from the URL:

- TuGames

We recommend a custom installation with paclet, which can be found at

- Paclet

The *MatTuGames* toolbox should work with all platforms.

Moreover, the toolbox works also with the game theory toolbox written by *Jean Derks*, which can be requested from:

- Derks

We added some adjusted files that fix a problem with closed loops under certain game classes.

This toolbox can be used to compute the pre-nucleolus up to 10-persons, if one has no license of Matlab's optimization toolbox.

Finally, the toolbox `MatTuGames` offers interfaces to access the solvers of CVX, CPLEX, GLPK, GUROBI, HSL, IPOPT, MOSEK, and OASES. The CPLEX interfaces are compatible with version 12.10.

To summarize, apart of the mentioned software, the toolbox requires the following *MATLAB* toolboxes:

MATLAB Parallel Server, *Optimization Toolbox*, *Parallel Computing Toolbox*, *Signal Processing Toolbox*, *Statistics and Machine Learning Toolbox*, *Symbolic Math Toolbox*

to get full functionality in serial as well as in parallel.

5. Acknowledgment

The author acknowledges support by the state of Baden-Württemberg through bwHPC.

Of course, the usual disclaimer applies.

6. License

7. Citation

For citation consult the URL:

- Cite As:

8. MATLAB File Exchange

For additional comments and information of the current version consult the URL:

File Exchange

Author

Holger I. Meinhardt Institute of Operations Research University of Karlsruhe (KIT) E-mail: holger.meinhardt ät partner.kit.edu