Computing equilibria of repeated games of complete information with the abSan package

Philip Barrett

07/05/2015

This document explains how to use the `abSan` package to compute equilibria of repeated games. This package uses the iterative approach outlined in Abreu & Sannikov's 2013 paper "An algorithm for two-player repeated games with perfect monitoring".

Computing the equilibrium is straightforward; we simply need create a function to define the game and then pass that funcction to the command that computes the equilibrium. The game definition comes in four parts: the number of possible actions, the payoffs from player 1; the payoffs for player 2; and the (common) rate of time preference.

For example, imagine that we want to compute the equilibrium of the following simple prisoner's dilemma:

	Cooperate	Defect
Cooperate	(4,4)	(0,6)
Defect	(6,0)	(2,2)

Where both players have common rate of time preference $\delta = 0.8$. Then, we can define this game using the following function:

The argument `opts` must be present in the model-definition function. In this simple example, it has no role, but in general can be used to pass options to the game definition (for example, varying the rate of time preference). The names of the return list are also essential. These are used by the function that computes the equilibrium, `abSan.eqm`. We are now ready to use this function to compute the solution.

```
require(abSan, quietly = TRUE )
sol <- abSan.eqm( modelName = PD.def, charts = TRUE )
```

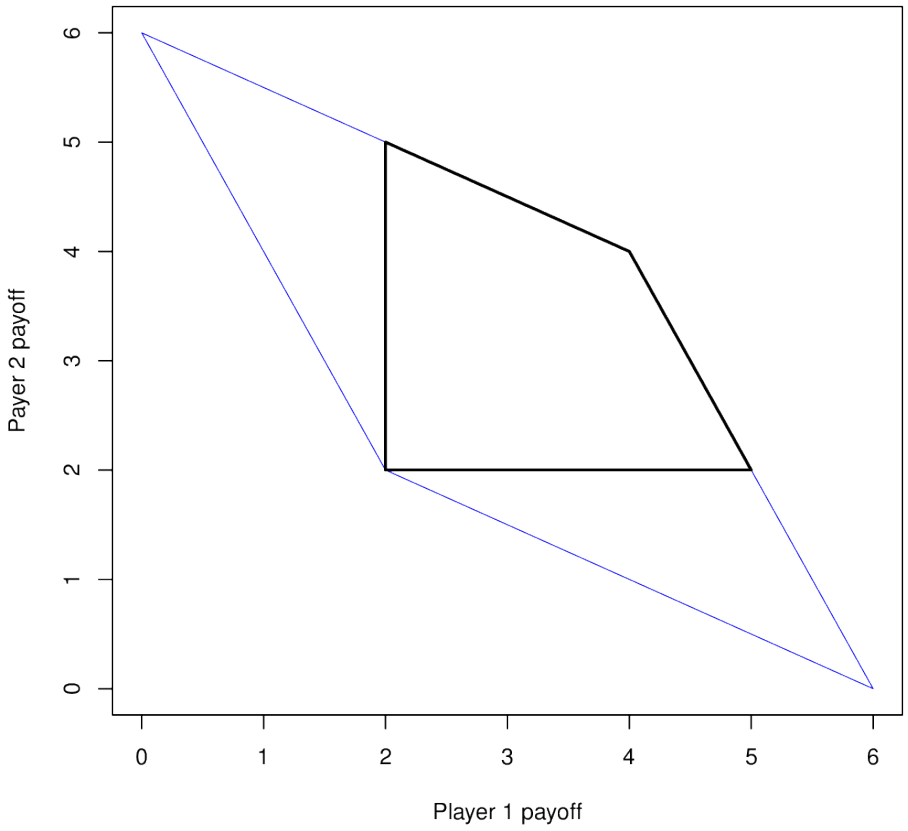
```
## Iteration 1: Hausdorff distance = 2.24
## Iteration 2: Hausdorff distance = 0
```

In this case, the code solves the problem exactly in just two iterations. The solution is a list, the most important of which is `$vStar$mZ`. This defines the vertices of the solution.

```
sol$vStar$mZ
```

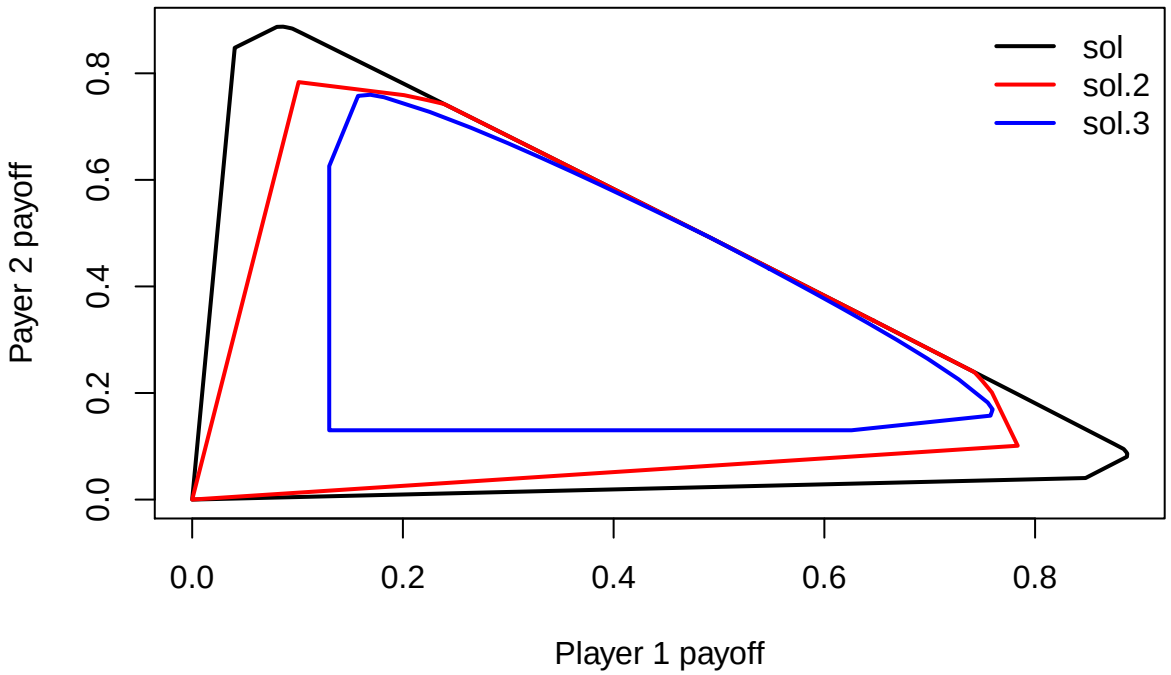
```
##      [,1] [,2]
## [1,]    5    2
## [2,]    2    2
## [3,]    2    5
## [4,]    4    4
```

With the `charts` flag turned on, the solution and the iterations are plotted and saved in `equilibrium.pdf` and `convergence.pdf`. The convergence plot clearly shows the initial guess in blue (the whole payoff space) and the solution in black.



A more interesting example is provided in the `examples.cournot.CES` function included in the library. This computes the payoffs in a two-player Cournot duopoly with a CES aggregate demand curve. The inputs of the function are a list with various elements (see the function definition for more details of possible elements). But in particular, the options `delta` and `elas` control the patience of the players and the elasticity of the demand curve.

```
sol <- abSan.eqm( modelName = examples.cournot.CES, modelOpts = list(delta=.9, elas=10),
                  par=TRUE, print.output = FALSE )
sol.2 <- abSan.eqm( modelName = examples.cournot.CES, modelOpts = list(delta=.75, elas=10),
                    par=TRUE, print.output = FALSE )
sol.3 <- abSan.eqm( modelName = examples.cournot.CES, modelOpts = list(delta=.9, elas=2),
                    par=TRUE, print.output = FALSE )
abSan.plotSet( sol$vStar$mZ, lwd=2 )
abSan.addSet( sol.2$vStar$mZ, lwd=2, col='red' )
abSan.addSet( sol.3$vStar$mZ, lwd=2, col='blue' )
legend( 'topright', c('sol', 'sol.2', 'sol.3' ), lwd=2, col=c('black', 'red', 'blue'), bty='n')
```



Note that in this last example, we not only pass different arguments to the function definition, but also use the `par` flag to exploit multicore processing (where available), , and utilize the plotting functions `abSan.plotSet` and `abSan.addSet`.

It is worth emphasizing that this is a fairly large game. There are by default 15 actions per player in this game (and so 225 joint actions in total). The underlying code is written in C++ so solves this to 10 decimal places in around 5 seconds per solution.

Finally, feel free to use this package in your own work, but please cite it appropriately.