Package ‘abSan’

July 5, 2015

Type Package

Title Computes the Abreu-Sannikov repeated game algorithm

Version 1.0

Date 2013-08-27

Author Philip Barrett

Maintainer Philip Barrett <pobarrett@uchicago.edu>

Description Code to solve repeated games using the method described in
Abreu & Sannikov's 2013 Theoretical Economics paper ``An algorithm for
two-player games with perfect monitoring''

License GPL-3

Depends R (>= 2.15.0), Rcpp (>= 0.10.4), parallel, RcppEigen, RUnit

LinkingTo Rcpp, RcppEigen

R topics documented:

abSan-package — *Computes the Abreu-Sannikov repeated game algorithm*

Description

Code to solve repeated games using the method described in Abreu & Sannikov's 2013 Theoretical Economics paper "An algorithm for two-player games with perfect monitoring".

Details

Package:	abSan
Type:	Package
Version:	1.0
Date:	2013-08-27
License:	GPL-3

Solves for the equilibrium set of values of a 2-played repeated game with monitoring. Games must be described as a list containing the actions, payoffs, and discount rate for the two players. See the documentation for `model.initiate` for details about how to create model files, or one the example model files (such as `examples.PD`). The equilibrium set can be found under the default options using only the `abSan.eqm(modelName)` command. Common examples available for immediate use are the Prisoner's Dilemma, Battle of the Sexes, and a Cournot duopoly. Model files for these are stored in `examples.PD`, `examples.sexes`, and `examples.cournot`. Abreu & Sannikov's arbitrary 9-action game on p.11 of their paper is also available as `examples.AS`. For these examples, `abSan` is accurate to a minimum of 7 decimal places (and in most cases, more than 10). A final example computes the solution to Fuchs & Lippi's independent monetary policy game in their 2006 ReStud paper. The accuracy of this last example has not been independently verified.

Comments and suggestions are gratefully received by the author.

Author(s)

Philip Barrett <pobarrett@uchicago.edu>

References

Abreu & Sannikov (2013) "An algorithm for two-player repeated games with perfect monitoring", Theoretical Economics. http://econtheory.org/ojs/index.php/te/article/viewForthcomingFile/1302/8114/1tion
Fuchs & Lippi (2006) "Monetary Union with Voluntary Participation", Review of Economic Studies

See Also

`abSan.eqm`, `examples.PD`, `examples.sexes`, `examples.cournot`, `examples.AS`, `examples.FL.union`

Richard Kratzwer's Java implementation (rgsolve): http://www.princeton.edu/~rkatzwer/rgsolve/index.html

Examples

```
## Compute Abreu-Sannikov's example in the paper
sol <- abSan.eqm( modelName=examples.AS )
sol$status
  # Should be 1 for success
sol$vStar$mZ
  # Print the vertices of the outcome
example <- examples.AS()
  # Load the example model
abs( example$ans$mZ - sol$vStar$mZ )
  # Compute differences oto (pre-loaded) exact solution

## Compute the Cournot duopoly game
sol <- abSan.eqm( modelName=examples.cournot, modelOpts=list( 'iActs' = 15 ), charts=TRUE )
  # Plots charts for the equilibrium set and convergence in the present working directory
sol$time
  # Time taken
```

---

abSan-addSet *Plot a set on an old set of axes*

Usage

```
abSan-plotSet( mZ )
```

Arguments

mZ	A nx2 matrix of vertices. The two columns are the x and y values of the vertices
...	Arguments to pass to the plot, e.g. color, line width

Value

A plot

Note

abSan.addSet

sol <- abSan.eqm(modelName = examples.cournot.CES, modelOpts = list(delta=.9)) abSan.plotSet(sol$vStar$mZ, lwd=2) sol.2 <- abSan.eqm(modelName = examples.cournot.CES, modelOpts = list(delta=.75)) abSan.addSet(sol.2$vStar$mZ, lwd=2, col='red')

Author(s)

Philip Barrett

abSan-plotSet — *Plot a set on a new set of axes*

Usage

```
abSan-plotSet( mZ )
```

Arguments

`mZ`	A nx2 matrix of vertices. The two columns are the x and y values of the vertices
`...`	Arguments to pass to the plot, e.g. color, line width

Value

A plot

Note

abSan.addSet

sol <- abSan.eqm(modelName = examples.cournot.CES, modelOpts = list(iAct=6, delta=.9)) abSan.plotSet(sol$vStar$mZ, lwd=2)

Author(s)

Philip Barrett

abSan.eqm — *Compute the set of equilibrium values*

Description

Calculates the set of values of all subgame-perfect equilibria for a repeated game.

Usage

```
abSan.eqm(modelName, model, set, pun, tol=1e-12, charts=FALSE, maxIt=150, detail=FALSE,
    print.output=TRUE, save.solution=FALSE, par=FALSE, cluster, modelOpts)
```

Arguments

`modelName`	the model definition function.
`model`	a post-processed model list. Usually created as the output of `model.initiate(modelName)` for some model. Useful when re-computing the same model with different solution options, as prevents re-defining the model each time.
`set`	an (m,2) matrix of vertices of the initial set of continuation values. Must wholly contain the equilibrium set. If missing or `NULL`, the set of stage payoffs is used.
`pun`	a length-2 vector of initial punishments. Must be less than the equilibrium punishment. If missing or `NULL`, the minmax payoff for the stage game is used.
`tol`	numeric convergence tolerance. The threshold Hausdorff difference between successive sets at which the algorithm terminates.
`charts`	Boolean flag for saving charts. Currently only equilibrium set and the sequence of convergent sets are created. These are saved as equilibrium.pdf, and convergence.pdf in the current working directory.
`maxIt`	The maximum number of iterations.
`detail`	Boolean flag for retaining information about the iterations
`par`	Boolean flag for using multicore execution.
`print.output`	Boolean flag for output display.
`save.solution`	Currently inactive.
`cluster`	number of nodes to use in cluster execution. Currently inactive.
`modelOpt`	options to pass to the model.

Value

Returns a list of solution components:

`status`	is `1` if solution converges to required tolerance. Otherwise `0`.
`vStar`	a list containing a description of the equilibrium set. `vStar$mZ` is a 2-column matrix of vertices of the equilibrium set. `vStar$mG` is a 2-column matrix of unit-magnitude normals to the boundary of the equilibrium set, and `vStar$vC` is a vector of intercepts to those normals, such that for every row g of `vStar$mG` and each entry c of `vStar$vC` the conditions $g.z \leq c$ are a necessary and sufficient condition for z to be in the equilibrium set.
`vBar`	the equilibrium punishment
`iterations`	The number of iterations to solution
`lSet`	returned only if `details=TRUE`. The list of sets at each iteration of the algorithm.
`lPun`	returned only if `details=TRUE`. The list of punishments at each iteration of the algorithm.
`time`	time to compute the equilibrium set.

See Also

`model.initiate`

Examples

```
## Compute the Cournot duopoly game
sol <- abSan.eqm( modelName=examples.cournot, modelOpts=list( 'iActs' = 15 ) )
  # Benchmark solution
sol$time
  # Time taken
sol <- abSan.eqm( modelName=examples.cournot, modelOpts=list( 'iActs' = 15 ), print.output=FALSE )
  # Turn off output
sol$time
  # Time taken
sol <- abSan.eqm( modelName=examples.cournot, modelOpts=list( 'iActs' = 40 ) )
  # Using model options to compute a finer discretization
sol$time
  # Time taken
sol <- abSan.eqm( modelName=examples.cournot, modelOpts=list( 'iActs' = 40 ), par=TRUE )
  # Using multicore execution to speed up larger example
sol$time
  # Time taken
```

---

examples.AS *Abreu & Sannikov's arbitrary game example*

Description

Example model definition function. Records an arbitrary game with three actions for each player and returns the a list of actions, payoffs, and a discount factor to define the game. These can then be used by `model.initiate` to generate a model description used in the body of the AS algorithm. This example also contains answers in the 'ans' list entry generated by Richard Kratzwer's rgsolve algorithm, when the default model options are passed. These are used in accuracy tests.

Usage

```
examples.AS(opts)
```

See Also

`model.initiate` `examples.PD`, `examples.cournot`, `examples.sexes`, `examples.FL.union`

Examples

```
model.defn <- examples.AS()
model.defn$delta
  # Print the discount factor
model <- model.initiate( examples.AS )
  # Pre-process the model
sol <- abSan.eqm( model=model )
  # Solve
```

examples.cournot *Cournot duopoly example*

Description

Example model definition function. Records a Cournot duopoly game and returns the a list of actions, payoffs, and a discount factor to define the game. These can then be used by `model.initiate` to generate a model description used in the body of the AS algorithm. This example also contains answers in the 'ans' list entry generated by Richard Kratzwer's rgsolve algorithm, when the default model options are passed. These are used in accuracy tests.

Usage

```
examples.cournot(opts)
```

Arguments

The list `opts` should be passed with the following entries:

iActs is the integer number of actions available to each player. Default is 15.

See Also

`model.initiate` `examples.PD`, `examples.sexes`, `examples.AS`, `examples.FL.union`

Examples

```
model.defn <- examples.cournot()
model.defn$delta
  # Print the discount factor
model <- model.initiate( examples.cournot )
  # Pre-process the model
sol <- abSan.eqm( model=model )
  # Solve
model <- model.initiate( examples.cournot, opts=list( 'iActs' = 80 ) )
sol <- abSan.eqm( model=model )
  # Alternative use
```

examples.FL.union *Fuchs & Lippi monetary union example*

Description

Example model definition function. Records a version of Fuchs & Lippi's independent monetary policy game. Here, monetary policy "spills over" from one country to another, so policymakers target not only their own inflation rate, but also it relative to their neightbor's. This function returns the a list of actions, payoffs, and a discount factor to define the game. These can then be used by `model.initiate` to generate a model description used in the body of the AS algorithm. This example also contains answers in the 'ans' list entry generated by Richard Kratzwer's rgsolve algorithm, when the default model options are passed. These are used in accuracy tests.

Usage

```
examples.FL.union(opts)
```

Arguments

The list `opts` should be passed with the following entries:

`iActs`	is the integer number of actions available to each player. Default is 40.
`piRange`	the range f inflation choices for each country. Default is (-5,5).

See Also

`model.initiate` `examples.PD`, `examples.sexes`, `examples.AS`, `examples.cournot`

Examples

```
model.defn <- examples.FL.union()
model.defn$delta
  # Print the discount factor
model <- model.initiate( examples.FL.union )
  # Pre-process the model
sol <- abSan.eqm( model=model )
  # Solve
model <- model.initiate( examples.FL.union, opts=list( 'iActs' = 80, piRange=c(-8,8) ) )
sol <- abSan.eqm( model=model )
  # Alternative use
```

`examples.PD` *Prisoner's dilemma example*

Description

Example model definition function. Records a Prisonner's dilemma game and returns the a list of actions, payoffs, and a discount factor to define the game. These can then be used by `model.initiate` to generate a model description used in the body of the AS algorithm. This example also contains answers in the 'ans' list entry generated by Richard Kratzwer's rgsolve algorithm. These are used in accuracy tests.

Usage

```
examples.PD(opts)
```

See Also

model.initiate examples.sexes, examples.cournot, examples.AS, examples.FL.union

Examples

```
model.defn <- examples.PD()
model.defn$delta
  # Print the discount factor
model <- model.initiate( examples.PD )
  # Pre-process the model
sol <- abSan.eqm( model=model )
  # Solve
```

---

examples.sexes *Battle of the sexes example*

Description

Example model definition function. Records a "Battle of the sexes" game and returns the a list of actions, payoffs, and a discount factor to define the game. These can then be used by `model.initiate` to generate a model description used in the body of the AS algorithm. This example also contains answers in the 'ans' list entry generated by Richard Kratzwer's rgsolve algorithm. These are used in accuracy tests.

Usage

```
examples.sexes(opts)
```

See Also

model.initiate examples.PD, examples.cournot, examples.AS, examples.FL.union

Examples

```
model.defn <- examples.sexes()
model.defn$delta
  # Print the discount factor
model <- model.initiate( examples.sexes )
  # Pre-process the model
sol <- abSan.eqm( model=model )
  # Solve
```

`model.initiate` *Processes a model for use in the algorithm*

Description

Processes a model definition function in two ways. First, it repackages the payoffs and actions for easier use in the rest of the code. Second, it computes the other properties of a game, such as the marginal gain from deviating, which the Abreu-Sannikov code uses. These are then returned as a list for use in other functions, such as, `abSan.eqm`.

Usage

```
model.initiate(defn.fn, opts)
```

Arguments

`defn.fn`	the model definition function. This must return a list including the following four entries: `iActions`, the 2-vector of the number actions for each player; `payoffs1`, the normal-form matrix of payoffs of the stage game for player 1; `payoffs2`, the normal-form matrix of payoffs of the stage game for player 1; and `delta` the discount factor. See, for instance, `examples.PD` in this package.
`opts`	a list of options to pass to the model definition function `defn.fn`.

Value

Returns a list of components used in the Abreu-Sannikov algorithm:

`iActions`	2-vector of the number actions for each player.
`mA`	a $(2, m)$-matrix, where m is the number of *joint* actions. Each row contains the action indices for the two players associated with that joint action.
`mF`	payoffs associated with the joint actions listed in mA.
`mH`	the gain from deviating from each joint action for the two players. Abreu-Sannikov's h(a) function.
`delta`	the discount factor.
`iJointActs`	the number of joint actions.
`minmax`	the minmax values of the stage game.

See Also

`examples.PD`, `examples.sexes`, `examples.cournot`, `examples.AS`, `examples.FL.union`

Examples

```
model <- model.initiate( examples.FL.union, opts=list( 'iActs' = 80, piRange=c(-8,8) ) )
  # Initiate a model with a large number of actions
```

Index