Game Theory Programming Functions for the TI-Nspire CX CAS Handheld and Associated Emulators.

Language: English

Version: 1.0

Date: 1/3/2019

Author: James O. Thompson

Email: jodaddy101@hotmail.com

Table of Contents

1. Introduction

The functions presented here are not intended to teach game theory. Instead they should complement a good introductory text on the subject. Such a text can be found in “Finite Mathematics” by Lial, Greenwell and Ritchey. This book is frequently used in a college course for students majoring in business, management, economics, etc.

Any student attempting to learn game theory without, at least, help with a modern calculator such as the TI-84 is sorely disadvantaged. Even with the help of the matrix features of the TI-84, the flood of trivial calculations obscures the beauty of game theory. Burdened with repetitive, redundant obscuring calculations such a disadvantaged student would be able to solve few of the many rich exercises at the end of each section of the book. With the help of the functions presented here the student should be able to solve many of the interesting problems posed at the end of each book section. That … and have time for other concurrent college courses as well.

The functions presented here are for the student equipped with a TI-Nspire CX CAS handheld or an emulator on a PC, iPad, etc. The “CAS” (Computer Algebra System) part is essential. The functions cannot be installed on the TI-Nspire without the CAS part.

The functions should be introduced piece meal through the course as aids to automate methods already taught from the book. The student should know the process being automated and use the automation only to avoid repetitive calculations not contributing to the learning process.

2. Platforms

The functions presented here will execute on the TI-Nspire CX CAS handheld, the TI-Nspire CX CAS Student Software and associated emulators executing on the iPad.

3. Installation

All of these functions are bundled into a single file, “game.tns”. This file is to be installed in the “mylib” folder where it will join the other files (numtheory.tns and linalgcas.tns) from the default installation. The file can be transferred to the handheld using the process described for file transfers in the handheld owner’s manual. On a PC the “mylib” subdirectory is in the User’s Documents subdirectory which, unfortunately, Microsoft likes to hide. See this web reference on how to locate the User’s Documents subdirectory.

After installing this file, you must “Refresh Libraries”. On the handheld, press Doc, then choose the “Refresh Libraries” option. On the Student Software, choose the Tools menu, then the “Refresh Libraries” option. Following this the functions may be accessed as the file

name (games) followed by the backslash (\) followed by the function name. e.g., “games\about()”.

Following installation, the function source code may be inspected and modified as desired. Furthermore, it may be distributed freely and for free.

4. Game Theory Template

Textbooks commonly illustrate game theory functions where the matrix columns and rows are unlabeled. This can be extremely confusing, especially when the column and row meanings are changed. The functions in this set work with labeled rows and columns and will shuffle the labels when appropriate. This and its value are seen in examples below.

a. Example illustrating labeled rows and columns of game theory payoff matrix.

_	p_b	0.6	0.4
p_a	_	*rain*	*norain*
1	*stadium*	-1800	2400
0	*gym*	1500	1500
0	*both*	1200	2100

5. Game Theory Functions

This section contains the public game theory functions that should be used directly.

a. About()

The function gives some development data for the library.

games\about () ▸

```
"Game Theory Functions "
"Language: English "
"Version: 1.0 "
"Date: 1/3/2019 "
"Author: James O. Thompson "
"Email: jodaddy101@hotmail.com "
```

b. NewGame()

The functions in this library are limited to zero-sum, two-person games. The payoffs for the game are represented by a two-dimensional matrix where the rows indicate the strategies for the first person (person A) and the columns the strategies for the second person (person B). A positive entry in the matrix indicates a payoff from person B to person A while a negative entry indicates a payoff from person A to person B.

The NewGame() function generates a payoff matrix skeleton for such a new game. The first argument gives the number of rows and the second argument the number of columns. Two additional features surrounding the payoff matrix track the strategy names and the probabilities for each strategy. Default strategy names are subscripted a's and b's. They may be edited into any legal variable name. They should never represent actual variables. The probabilities are initially all question marks but may be set manually or by using the GetOptimal() or DoSimplex() functions.

You get a skeleton via the NewGame() function which you can past into an assignment statement and edit.

$$\text{games\newgame}(2,2) \rightarrow \begin{bmatrix} \_ & p_b & ? & ? \\ p_a & \_ & b_1 & b_2 \\ ? & a_1 & ? & ? \\ ? & a_2 & ? & ? \end{bmatrix}$$

$$\text{variable} := \begin{bmatrix} \_ & p_b & ? & ? \\ p_a & \_ & better & notbetter \\ ? & market & 50000 & -25000 \\ ? & notmarket & -40000 & -10000 \end{bmatrix} \rightarrow \begin{bmatrix} \_ & p_b & ? & ? \\ p_a & \_ & better & notbetter \\ ? & market & 50000 & -25000 \\ ? & notmarket & -40000 & -10000 \end{bmatrix}$$

c. GetDominates()

This function accepts a payoff matrix and determines the dominating rows and/or columns.

$$\mathbf{g11} := \begin{bmatrix} \_ & p_b & ? & ? \\ p_a & \_ & b_1 & b_2 \\ ? & a_1 & 3 & 6 \\ ? & a_2 & 1 & 4 \\ ? & a_3 & 4 & -2 \\ ? & a_4 & -4 & 0 \end{bmatrix} \rightarrow \begin{bmatrix} \_ & p_b & ? & ? \\ p_a & \_ & b_1 & b_2 \\ ? & a_1 & 3 & 6 \\ ? & a_2 & 1 & 4 \\ ? & a_3 & 4 & -2 \\ ? & a_4 & -4 & 0 \end{bmatrix}$$

$$\mathbf{games\backslash getdominates}\ (\mathbf{g11}) \rightarrow \begin{bmatrix} \_ & \_ & \_ & \_ & \_ \\ row & a_1 & dominate & row & a_2 \\ row & a_1 & dominate & row & a_4 \end{bmatrix}$$

d. DoDominates()

This function determines and removes dominated rows and/or columns.

$$\mathbf{g11}:=\begin{bmatrix} \_ & p_b & ? & ? \\ p_a & \_ & b_1 & b_2 \\ ? & a_1 & 3 & 6 \\ ? & a_2 & 1 & 4 \\ ? & a_3 & 4 & -2 \\ ? & a_4 & -4 & 0 \end{bmatrix} \blacktriangleright \begin{bmatrix} \_ & p_b & ? & ? \\ p_a & \_ & b_1 & b_2 \\ ? & a_1 & 3 & 6 \\ ? & a_2 & 1 & 4 \\ ? & a_3 & 4 & -2 \\ ? & a_4 & -4 & 0 \end{bmatrix}$$

$$\mathbf{games\backslash getdominates}\left(\mathbf{g11}\right) \blacktriangleright \begin{bmatrix} \_ & \_ & \_ & \_ & \_ \\ row & a_1 & dominate & row & a_2 \\ row & a_1 & dominate & row & a_4 \end{bmatrix}$$

$$\mathbf{games\backslash dodominates}\left(\mathbf{g11}\right) \blacktriangleright \begin{bmatrix} \_ & p_b & ? & ? \\ p_a & \_ & b_1 & b_2 \\ ? & a_1 & 3 & 6 \\ ? & a_3 & 4 & -2 \end{bmatrix}$$

e. GetStrategies()

This function accepts a payoff matrix and returns the strategies/responses for both players as well as the payoff for the strategy pair. If the strategies yield a strictly determined game (has a saddle point) this is noted, else it is noted as requiring a mixed

strategy.

games\getstrategies

$$\begin{pmatrix} \_ & p_b & ? & ? & ? \\ p_a & \_ & b_1 & b_2 & b_3 \\ ? & a_1 & 3 & -4 & 1 \\ ? & a_2 & 5 & 3 & -2 \end{pmatrix} \rightarrow \begin{bmatrix} choice & a_1 & a_2 & \_ & b_3 & b_2 & b_1 \\ response & b_2 & b_3 & \_ & a_1 & a_2 & a_2 \\ payoff & -4 & -2 & mixed & 1 & 3 & 5 \end{bmatrix}$$

games\getstrategies

$$\begin{pmatrix} \_ & p_b & ? & ? \\ p_a & \_ & b_1 & b_2 \\ ? & a_1 & -6 & 2 \\ ? & a_2 & -1 & -10 \\ ? & a_3 & 3 & 5 \end{pmatrix} \rightarrow \begin{bmatrix} choice & a_2 & a_1 & a_3 & \_ & b_1 & b_2 \\ response & b_2 & b_1 & b_1 & \_ & a_3 & a_3 \\ payoff & -10 & -6 & 3 & saddle & 3 & 5 \end{bmatrix}$$

f. IsStrict()
This function accepts a payoff matrix and determines whether the outcome is strictly determined.

$$\mathbf{m23} := \begin{bmatrix} \_ & p_b & ? & ? & ? & ? \\ p_a & \_ & b_1 & b_2 & b_3 & b_4 \\ ? & a_1 & -6 & 1 & 4 & -2 \\ ? & a_2 & 9 & 3 & -8 & -7 \end{bmatrix} \rightarrow \begin{bmatrix} \_ & p_b & ? & ? & ? & ? \\ p_a & \_ & b_1 & b_2 & b_3 & b_4 \\ ? & a_1 & -6 & 1 & 4 & -2 \\ ? & a_2 & 9 & 3 & -8 & -7 \end{bmatrix}$$

$$\mathbf{games\backslash isstrict}\left(\mathbf{m23}\right) \rightarrow \text{false}$$

$$\mathbf{games\backslash getstrategies}\left(\mathbf{m23}\right) \rightarrow \begin{bmatrix} choice & a_2 & a_1 & \_ & b_4 & b_2 & b_3 & b_1 \\ response & b_3 & b_1 & \_ & a_1 & a_2 & a_1 & a_2 \\ payoff & -8 & -6 & mixed & -2 & 3 & 4 & 9 \end{bmatrix}$$

$$\mathbf{m21} := \begin{bmatrix} \_ & p_b & ? & ? & ? \\ p_a & \_ & b_1 & b_2 & b_3 \\ ? & a_1 & 2 & 3 & 1 \\ ? & a_2 & -1 & 4 & -7 \\ ? & a_3 & 5 & 2 & 0 \\ ? & a_4 & 8 & -4 & -1 \end{bmatrix} \rightarrow \begin{bmatrix} \_ & p_b & ? & ? & ? \\ p_a & \_ & b_1 & b_2 & b_3 \\ ? & a_1 & 2 & 3 & 1 \\ ? & a_2 & -1 & 4 & -7 \\ ? & a_3 & 5 & 2 & 0 \\ ? & a_4 & 8 & -4 & -1 \end{bmatrix}$$

$$\mathbf{games\backslash isstrict}\left(\mathbf{m21}\right) \rightarrow \text{true}$$

$$\mathbf{games\backslash getstrategies}\left(\mathbf{m21}\right) \rightarrow \begin{bmatrix} choice & a_2 & a_4 & a_3 & a_1 & \_ & b_3 & b_2 & b_1 \\ response & b_3 & b_2 & b_3 & b_3 & \_ & a_1 & a_2 & a_4 \\ payoff & -7 & -4 & 0 & 1 & saddle & 1 & 4 & 8 \end{bmatrix}$$

g. GetSimplex()

Games that are not strictly determined may be formulated as a linear programming problem. This function accepts a game definition and returns the corresponding linear programming definition. This may be processed using all the functions defined in the Simplex library published by James O. Thompson on the web site ticalc.org.

games\getsimplex

$$\begin{bmatrix} \_ & p_b & ? & ? & ? \\ p_a & \_ & b_1 & b_2 & b_3 \\ ? & a_1 & 5 & 10 & 10 \\ ? & a_2 & 8 & 4 & 8 \\ ? & a_3 & 6 & 6 & 3 \end{bmatrix} \rightarrow \begin{bmatrix} \_ & x_1 & x_2 & x_3 & \_ & \_ \\ maximize\_z & 1 & 1 & 1 & \_ & \_ \\ subject\_to & \_ & \_ & \_ & \_ & \_ \\ eqn1 & 5 & 10 & 10 & "\leq" & 1 \\ eqn2 & 8 & 4 & 8 & "\leq" & 1 \\ eqn3 & 6 & 6 & 3 & "\leq" & 1 \end{bmatrix}$$

After solving the linear programming problem, the answers would be manually retrieved and manually inserted into the game definition as associated probabilities. This method follows the textbook explanation. However, the DoSimplex() and GetOptimal() functions in the Games library do this automatically and would be the preferred method.

h. DoSimplex()

Games that are not strictly determined may be formulated as a linear programming problem and solved using the simplex algorithm. This function automatically does the formulation, invokes the simplex algorithm, retrieves the answers and deposits the corresponding probabilities into the game definition.

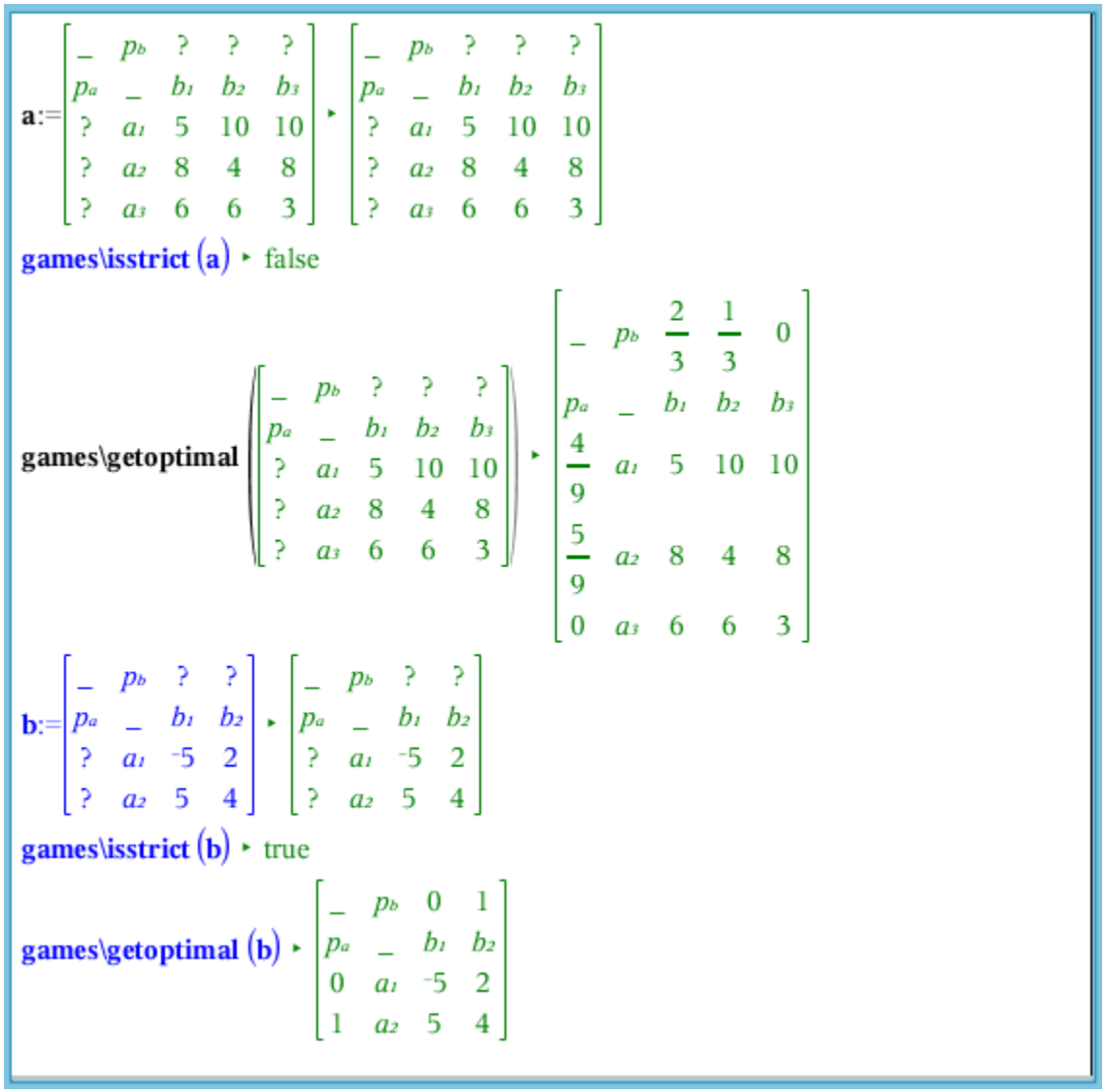
games\dosimplex

$$\begin{bmatrix} \_ & p_b & ? & ? & ? \\ p_a & \_ & b_1 & b_2 & b_3 \\ ? & a_1 & 5 & 10 & 10 \\ ? & a_2 & 8 & 4 & 8 \\ ? & a_3 & 6 & 6 & 3 \end{bmatrix} \rightarrow \begin{bmatrix} \_ & p_b & \frac{2}{3} & \frac{1}{3} & 0 \\ p_a & \_ & b_1 & b_2 & b_3 \\ \frac{4}{9} & a_1 & 5 & 10 & 10 \\ \frac{5}{9} & a_2 & 8 & 4 & 8 \\ 0 & a_3 & 6 & 6 & 3 \end{bmatrix}$$

i. GetOptimal()

This function accepts a game definition and computes the corresponding probabilities for the optimal solution. It automatically

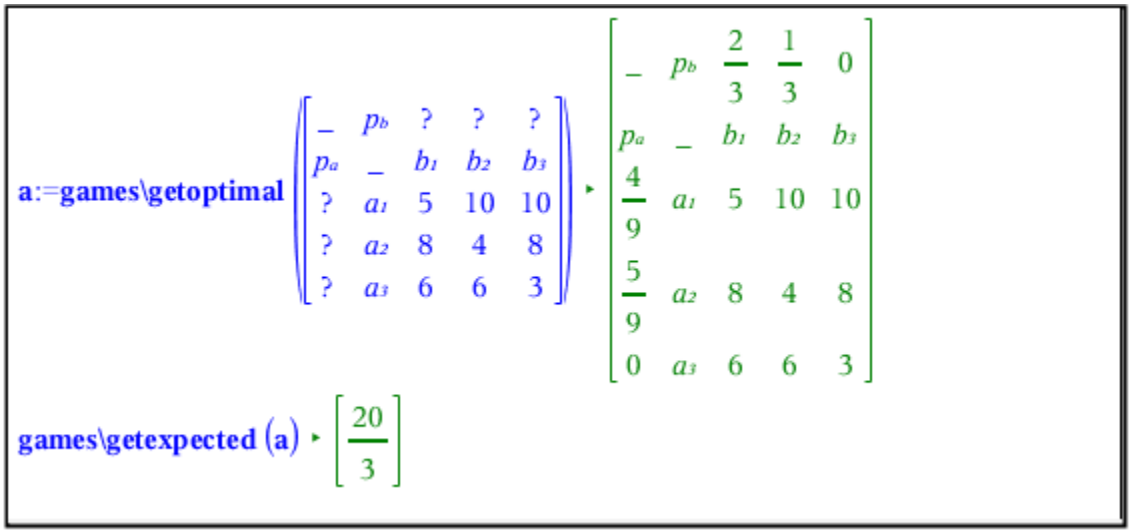
processes either a strictly determined game or a mixed game requiring the linear programming simplex algorithm.



j. GetExpected()

This function accepts a game with probabilities assigned and computes the expected payoff. Positive values indicate payoffs to

the first player and negative values a payoff to the second player.



6. Helper Functions

This section contains the private library functions specific to Game Theory. These are not explicitly documented here but the functions are liberally commented.

a. AddSubscript()
b. ColSwap()
c. Dominates()
d. GetMsg()
e. MatInsert()
f. SortMCols()

7. Simplex Functions

Games requiring a mixed strategy are solved using the linear programming simplex method. To avoid installing a library for this method, the functions are included within the Games library. These functions are listed below. Documentation for solving linear programming problems are included with the distribution for the Games library. It should be noted that the Simplex documentation assumes functions are located in a Simplex library. If used from the Games library, they are located within the Games library.

a. AutoPivot()
b. Dual()

c. Feasible()
d. GetPivot()
e. IsPblm()
f. IsTableau()
g. LetItRip()
h. NewFunction()
i. NewTab()
j. PblmMax()
k. PblmMin()
l. PblmNew()
m. Pivot()
n. PivotToFix()
o. PivotToImprove()
p. PivotToRow()
q. Shadows()
r. Solution()
s. TheWorks()
t. Unitize()