TE WHARE WĀNANGA O TE ŪPOKO O TE IKA A MĀUI
VICTORIA
UNIVERSITY OF WELLINGTON

SEF Working paper: 07/2014

NIRA-GUI

A MATLAB application which solves for couple-constraint Nash equilibria from a symbolic specification

Jacek B Krawczyk and Wilbur Townsend

The Working Paper series is published by the School of Economics and Finance to provide staff and research students the opportunity to expose their research to a wider audience. The opinions and views expressed in these papers are not necessarily reflective of views held by the school. Comments and feedback from readers would be welcomed by the author(s).

Further enquiries to:

The Administrator
School of Economics and Finance
Victoria University of Wellington
P O Box 600
Wellington 6140
New Zealand

Phone: +64 4 463 5353
Email: alice.fong@vuw.ac.nz

Working Paper 07/2014
ISSN 2230-259X (Print)
ISSN 2230-2603 (Online)

NIRA-GUI

A MATLAB APPLICATION WHICH SOLVES FOR COUPLED-CONSTRAINT NASH EQUILIBRIA FROM A SYMBOLIC SPECIFICATION

JACEK B. KRAWCZYK & WILBUR TOWNSEND

Abstract. A powerful method for computing Nash equilibria in constrained, multi-player games is created when the relaxation algorithm and the Nikaido-Isoda function are used within a MATLAB application. This paper describes that application, which is able to solve static and open-loop dynamic games specified symbolically.

Working Paper
NIRA-GUI

JEL Classification: C63 (Computational Techniques), C72 (Noncooperative Games), C87 (Economic Software).
Authors' Keywords: Nikaido-Isoda function; Coupled constraints.

Jacek B. Krawczyk. Commerce & Administration Faculty, Victoria University of Wellington, PO Box 600, Wellington, New Zealand; fax +64-4-4712200.
Email: Jacek.Krawczyk@vuw.ac.nz ; http://www.vuw.ac.nz/staff/jacek_ krawczyk

Wilbur Townsend. Email: wilbur.townsend@gmail.com

---

*Supported by the Victoria University Research Fund, Award Number 200757.

Contents

1. Introduction

Game theory problems are well known for being difficult to solve. The MATLAB application described in this document is able to solve games of varying complexity for their Nash equilibria and welfare-maximising solutions. Those games are specified through a graphical user interface which allows for symbolic input.

The application uses the Nikaido-Isoda function and the relaxation algorithm to find the unique Nash equilibrium. Problems with a unique equilibrium have been focused upon because of their importance in regulatory economics and management.

Also of importance in regulatory problems and particularly in environmental economics are games with coupled constraints Rosen (1965). Coupled constraints force players to restrict their collective actions and are a common part of many environmental problem specifications. This application is particularly suited to solving such problems.

In contrast to earlier versions of NIRA (*e.g.* Krawczyk and Zuccollo (2007)), this application allows for symbolic specification of games through a graphical user interface. These symbolic specifications are simplified by the application into a numerical form which can be efficiently computed. This version also contains features to produce output in accessible forms and repeat simulations for varying parametrisation.

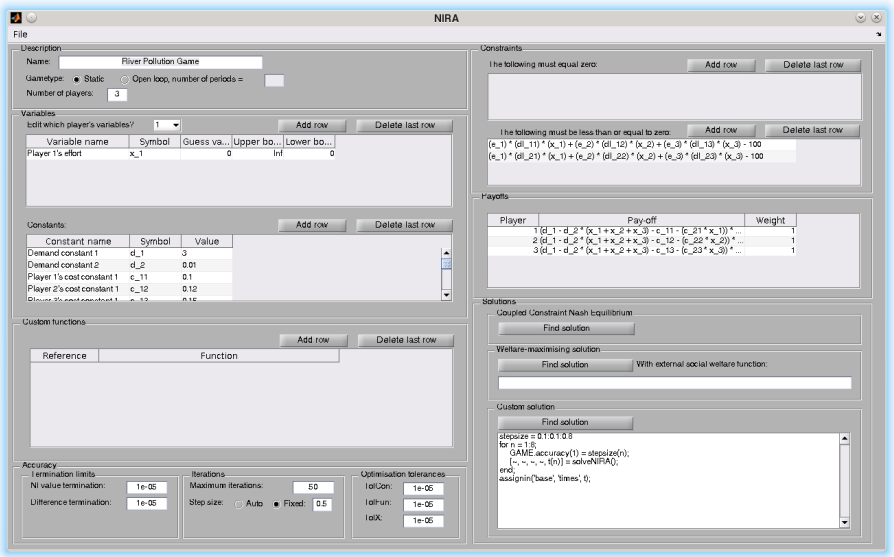
This paper comprises four sections. Section 2 introduces the program. Section 3 explains the programme design that is used to implement NIRA and compute the Nash equilibria. Section 4 gives a simple introduction to the theory underlying NIRA. (For a more detailed explanation refer to Krawczyk and Uryasev (2000) and Krawczyk (2005).)

2. Usage

2.1. **Getting started.** To initiate NIRA-GUI, type NIRA into the MATLAB console. The graphical interface (see Fig. 2.1) appears, by and large, self explanatory. (The code is available on Krawczyk and Townsend (2014a); also check Krawczyk and Townsend (2014b) for updates on this manual.)

NIRA supports both static and open loop games. While its core functionality is to solve for coupled-constraint Nash equilibria, it can also solve for welfare-maximising solutions.

Several example games are included with NIRA; these can be found in the 'Data' folder. These collectively demonstrate much of NIRA's functionality.



2.2. **Open loop periods.** When solving an open loop game, the dynamic action selected in each period can be referred to as the variable name followed by '_t', where t is the period. For example, if the variable is 'u_1', the action in period 2 is 'u_1_2'.

In order to allow the number of periods in an open loop game to be changed quickly, the term "(n)" can be used in the payoff and constraint functions. If we say that T = total periods, (n) will be substituted as:

- the current period (from 0 to T-1) in the stage payoffs,
- T in the scrap payoffs, and
- for each constraint that contains (n), that constraint will be duplicated with (n) = 0,...,T-1.

2.3. **Vector inputs.** If vectors are entered in the constants' 'Value' column, NIRA will solve for every possible combination of constants. This allows results from a range of parametrisation to be solved.

2.4. **Exporting results to the workspace.** From the results screen, the output can be exported to the workspace. It is stored in a structure array called Output. Output has the same number of dimensions as the number of constants that were entered as vectors, so each combination of constants (and therefore each solution) has a unique cell in Output.

2.5. **Troubleshooting.** NIRA-GUI automatically produces a log. This can be accessed through the global variable LOG or through the results GUI.

2.6. **Custom solution.** The custom solution functionality allows for greater flexibility in using NIRA-GUI. It allows the power of the GUI (and its simplification algorithms) to be combined with the flexibility of directly-coded solutions. For example, it could allow an openloop game to be run dynamically with different numbers of periods, and different specifications of variables in each length.

A game's specification is stored in a global structure GAME. GAME stores both the game as entered into the GUI and its simplified form that NIRA-GUI has produced. It does not store results. Accessing GAME through the MATLAB dashboard is possible through the command "global GAME" after NIRA has been initiated. Exploring this structure is recommended before using the custom solution functionality, as presumably the user will want to edit GAME directly.

The variables in the GUI will be loaded into GAME and simplified before the custom solution code is run. After editing GAME directly, calling simplifystruct() will update the simplified forms stored within GAME to match its unsimplified forms.

[NashEquilibrium, Payoffs, Constraints, Exitflag, Time] = solveNIRA() will solve the game for its coupled-constraint equilibrium using NIRA, with that vector of outputs having their obvious interpretations.

The following is an example of code that could be entered into the custom solution:

```
demand1 = 0:0.5:3
p = zeros (1,7)
for n = 1:7
      GAME.constants(1).value  = demand1(n)
       simplifystruct()
       [payoff, ~, ~, ~,~] = solveNIRA()
       p(n) = payoff(1)
end
assignin('base', 'Player1Payoffs', p)
```

3. MATLAB Programme Design

The following lists the files and folders of which NIRA-GUI is composed.

3.1. **NIRA GUI.** This folder contains the files which allow the primary GUI to function.

NIRA.fig: The GUIDE .fig file which provides the GUI design.
NIRA.m: The accompanying GUIDE .m file.
loadtostruct.m: Loads GUI data into GAME.

populate.m: Populates the GUI from GAME
resetvartable.m: Resets the table of player-controlled variables to display a certain player's variables after uploading currently entered data to GAME.
setplayernum.m: Is triggered when the number of players is changed.
simplifystruct.m: Simplifies GAME.

3.2. **NIRA solving.** This folder contains the files which solve for coupled-constraint equilibria and welfare-maximising solutions.

loadGAME.m: Loads GAME into the relaxation algorithm.
nik_iso.m: Defines the Nikaido-Isoda function.
op_alph3.m: Optimises step-sizes.
z_const.m: Optimises step-sizes.
relaxconstr.m: Constrains optimised step-sizes.
relax.m: Defines the relaxation algorithm.
solveNIRA.m: Solves the game.
loadconstraints.m: Loads the constraints.
loadpayofffunctions.m: Loads payoff functions.
utilitarian.m: Solves for welfare-maximising outcomes.

3.3. **Output GUIs.** This folder contains the files which display results.

logdisplay.fig: The GUIDE .fig file for displaying the log.
logdisplay.m: The GUIDE .m file for displaying the log.
plotpayoffs.m: Plots payoffs into results.fig.
plotstatic.m: Plots constraints and equilibrium into results.fig.
populateoutput.m: Populates the results GUI.
results.fig: The GUIDE .fig file for displaying the results.
results.m: The GUIDE .m file for displaying the results.
working.fig: The GUIDE .fig file for displaying the 'working' dialogue box.
working.m: The GUIDE .m file for displaying the 'working' dialogue box.
shapeoutput.m: Simplifies output exported to the desktop.

3.4. **Vector inputs.** This folder contains the files which process vector inputs

NIRAvector.m: Solves NIRA when constants are vectors.
checkarray.m: Checks cell arrays for arrays.
indicesmatrix.m: Creates an indices matrix for A.
permutationcellarray.m: Provides permutations of vectors.
utilitarianvector.m: Solves for the welfare-maximising solution when constants are vectors.

3.5. **Helpful functions.** This folder contains miscellaneous functions.

checkint.m: Checks if a number is an integer greater than zero.
closingbracketfinder.m: Finds the location of a closing bracket in a string.
evalinternal.m: Forces a symbolic formula to simplify.
load2log.m: Adds a new line of text to 'log'.
subsum.m: Sums a function over a variable, substituting subscripts.
tostr.m: Changes a num or a sym into a string.
underscorebracketfixer.m: Fixes syntax including underscores.
vect2str.m: Transforms a collective action into a readable string.

3.6. **Data.** This folder contains several working examples in .mat form. It contains no code.

4. Underlying theory

This section provides a brief explanation of the theory underlying the numerical solution of coupled-constraint Nash equilibria. Conventions and notations used in this paper are explained in section 4.1. Section 4.2 defines the structure of the games under consideration and defines a Nash equilibrium within this context. An explanation of coupled constraints is included in section 4.3 since this concept may be new to some readers. Section 4.6 concludes with a brief discussion of the conditions for the existence and uniqueness of Nash equilibria. The conditions are formulated as theorems whose proofs can be found in Rosen (1965) (and Haurie, Krawczyk, and Zaccour (2012) or Haurie and Krawczyk (2002)).

4.1. Conventions.

Convention 4.1. *There are two different types of vector: each player will have an* action*, and all players together will have a* collective action*. To avoid confusion, a player's action will be in normal type and have a subscript (*e.g. $x_i \in \mathbb{R}^{m_i}$*), and the collective action will be in boldface (*e.g. $\mathbf{x} \in \mathbb{R}^{m_1} \times \cdots \times \mathbb{R}^{m_n}$*). In this notation, $\mathbf{x} = (x_1, \ldots, x_n)$.*

Convention 4.2. *To distinguish between "their" collective action and "their" individual action we have attempted to avoid the use of collective pronouns to denote a gender inspecific person. Instead we adopt the convention that the words "he" and "she" have the common meaning "he or she" where appropriate.*

4.2. The structure of the game.

Definition 4.3. *A* game in normal form *is a three-tuple $\{\mathcal{N}, (X_i)_{i\in\mathcal{N}}, (\phi_i)_{i\in\mathcal{N}}\}$ where $\mathcal{N}$ is the set of players, $\mathcal{N} = \{1, 2, \ldots, n\}$, X_i is the action space of player i*

and ϕ_i is the payoff function of player i. These payoff functions are assumed to be continuous in $\mathbf{x} \in X$ and concave in x_i where X is the collective action space. A game of this kind is called concave.

Consider the solution to this game represented by the collective action $\mathbf{x}^*$. This is a Nash equilibrium and can be notated as

$$\mathbf{x}^* = \arg\operatorname{equil}_{\mathbf{x}\in X}\{\phi_1(\mathbf{x}), \ldots, \phi_n(\mathbf{x})\} \tag{1}$$

or alternatively written as

$$\phi_i(\mathbf{x}^*) = \max_{\mathbf{x}\in X} \phi_i(y_i|\mathbf{x}^*) \tag{2}$$

where $y_i|\mathbf{x}$ denotes a collection of actions where the ith agent "tries" y_i while the remaining agents continue to play x_j, $j = 1, 2, \ldots, i-1, i+1, \ldots, n$. At $\mathbf{x}^*$ no player can improve his own payoff through a unilateral change in his strategy hence a Nash equilibrium exists.

4.3. **Coupled constraint games.** This software is able to solve games of the type described above where the collective action set $X = X_1 \times \cdots \times X_n$ and each player's action is individually constrained. However, a more interesting group of games are those known as coupled constraint games. In these games one player's action affects how big the other players' actions can be.

In such games the collective action set X is assumed to be a closed convex subset of $\mathbb{R}_m$. Hence, $X \subseteq X_1 \times \cdots \times X_n$ where $X = X_1 \times \cdots \times X_n$ is the special case in which the constraints are described as being uncoupled.

4.4. **Existence and uniqueness of equilibrium points.** The obvious questions to ask before using any software to discover an equilibrium is whether an equilibrium exists at all and, if it does exist, whether it is unique. The conditions for existence and uniqueness for games with coupled constraints are more intricate than those for games with uncoupled constraints and so the two cases will be considered with separately.

Note that uniqueness is not a prerequisite for the software to find an equilibrium; however, if there are multiple equilibria the software will find the one closest to the starting point and there is no way of knowing how many others exist.[1]

4.4.1. *Uncoupled constraints.*

Theorem 4.4. *An equilibrium point exists for every concave n-person game.*

[1] Unless the technique of deflection, successful in the case of finite games (see McKelvey (1991)), could be extended to infinite games. This is a topic for independent research.

Therefore, if each player has a payoff function continuous in all players' actions and concave with respect to his own strategy, then the game must have at least one Nash equilibrium.

The uniqueness of the equilibrium relies on the concept of diagonal strict concavity (DSC) of the joint payoff function

$$f(\mathbf{x}, \mathbf{r}) \equiv \sum_{i=1}^{n} r_i \phi_1(\mathbf{x}). \tag{3}$$

The vector $\mathbf{r}$ is composed of weightings, r_i, of individual player's payoffs. They can have numerous interpretations depending upon the context.

Throughout this paper we will assume that $f(\mathbf{x}, \mathbf{r})$ is twice continuously differentiable. This is necessary to establish the conditions for uniqueness of the equilibrium.

Definition 4.5. *The joint payoff function is DSC if, for every* $\mathbf{x}^1, \mathbf{x}^2 \in X$*, we have*

$$(\mathbf{x}^2 - \mathbf{x}^1)' g(\mathbf{x}^1, \mathbf{r}) + (\mathbf{x}^1 - \mathbf{x}^2)' g(\mathbf{x}^2, \mathbf{r}) > 0 \tag{4}$$

where $g(\mathbf{x}^1, \mathbf{r})$ *is the pseudogradient of* f

$$g(\mathbf{x}^1, \mathbf{r}) = \begin{bmatrix} r_1 \nabla_1 \phi_1(\mathbf{x}) \\ \vdots \\ r_n \nabla_n \phi_n(\mathbf{x}) \end{bmatrix} \tag{5}$$

To check for DSC it is enough to test whether the Jacobian of $g(\mathbf{x}, \mathbf{r})$ is negative definite. In economic terms a game which is DSC is one in which each player has more control over his payoff than the other players have over it.

Theorem 4.6. *In a game with uncoupled constraints, if the joint payoff function* $f(\mathbf{x}, \mathbf{r})$ *is DSC for some* $\mathbf{r} > 0$*, then there exists a unique Nash equilibrium.*

4.4.2. *Coupled constraints.* Coupling the constraints complicates the equilibrium definition. A function $h : \mathbb{R}^m \to \mathbb{R}^L$ must be introduced to describe the constraint set X. Here m is the dimension of the collective strategy space and there are L constraints in that space. In the case of a concave game such as we have here each $h_\ell(\mathbf{x})$, $\ell = 1, \ldots, L$ is a concave function of $\mathbf{x}$ defined such that

$$X = \{\mathbf{x} : h(\mathbf{x}) \geq 0\} \tag{6}$$

Let the Lagrange multiplier vector for player i be denoted by λ_i^*. Here the Lagrange multiplier shows the shadow price of the constraints for player i. Given these definitions $\mathbf{x}^* \in X$ is a coupled constraint equilibrium point if and only if it satisfies

the following Kuhn-Tucker conditions:

$$h(\mathbf{x}^*) \geq 0 \tag{7}$$

$$(\lambda_i^*)^T h(\mathbf{x}^*) = 0 \tag{8}$$

$$\phi_i(\mathbf{x}^*) \geq \phi_i(y_i|\mathbf{x}^*) + (\lambda_i^*)^T h(y_i|\mathbf{x}^*) \tag{9}$$

Definition 4.7. *The equilibrium point* $\mathbf{x}^*$ *is a* Rosen-Nash normalised equilibrium *if, for some vectors* $\mathbf{r} > 0$ *and* $\lambda \geq 0$*, conditions* (7)-(9) *are satisfied where*

$$\lambda_i = \frac{\lambda}{r_i} \tag{10}$$

for each i*.*

Theorem 4.8. *There exists a normalised equilibrium point to a concave* n*-person game for every* $\mathbf{r} > 0$*.*

Theorem 4.9. *Let* Q *be a convex subset of* $\mathbb{R}_+^n$ *and the weighting of the joint payoff function be* $\mathbf{r} \in Q$*. Let* $f(\mathbf{x}, \mathbf{r})$ *be DSC on the convex set* X *and such that the Kuhn-Tucker conditions are satisfied. Then, for the weighting* $\mathbf{r}$*, there is a unique normalised equilibrium point.*

This means that if a game is DSC for some feasible distribution of weightings in the joint payoff function then the game possesses a unique coupled constraint equilibrium for each such distribution.

4.5. **A numerical solution to coupled constraint games.** The software implements an equilibrium search method based on the Nikaido-Isoda function and a relaxation algorithm, hence the acronym NIRA.

4.5.1. *The Nikaido-Isoda function.* This function transforms the difficult problem of finding an equilibrium into a far simpler optimisation problem.

Definition 4.10. *Let* ϕ_i *be the payoff function for player* i *and* X *a collective strategy space as before. The Nikaido-Isoda function* $\Psi : X \times X \to \mathbb{R}$ *is defined as*

$$\Psi(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{n} [\phi_i(y_i|\mathbf{x}) - \phi_i(\mathbf{x})] \tag{11}$$

Result 4.11.

$$\Psi(\mathbf{x}, \mathbf{x}) \equiv 0 \qquad \mathbf{x} \in X. \tag{12}$$

Each summand from the Nikaido-Isoda function can be thought of as the improvement in payoff a player will receive by changing his action from x_i to y_i while all other players continue to play according to $\mathbf{x}$. Therefore, the function represents the sum of these improvements in payoff. Note that the *maximum* value this function can take, for a given $\mathbf{x}$, is always nonnegative, owing to Result 4.11 above.

The function is everywhere non-positive when either $\mathbf{x}$ or $\mathbf{y}$ is a Nash equilibrium point, since in an equilibrium situation no player can make any more improvements to their payoff. Consequently, each summand in this case can be at most zero at the Nash equilibrium point Krawczyk and Uryasev (2000).

It can now be concluded that when the Nikaido-Isoda function cannot be made (significantly) positive for a given $\mathbf{y}$, we have (approximately) reached the Nash equilibrium point. This observation is used to construct a termination condition for the relaxation algorithm, that is, an ε is chosen such that, when $\max_{\mathbf{y}\in\mathbb{R}^m} \Psi(\mathbf{x}^s, \mathbf{y}) < \varepsilon$, the Nash equilibrium would be achieved to a sufficient degree of precision Krawczyk and Uryasev (2000).

Definition 4.12. *The* optimum response function *at point* $\mathbf{x}$ *is*

$$Z(\mathbf{x}) \in \arg\max_{\mathbf{y}\in X} \Psi(\mathbf{x}, \mathbf{y}). \tag{13}$$

In brief terms, this function returns the set of players' actions whereby they all attempt to unilaterally maximise their payoffs. The vector $Z(\mathbf{x})$ gives the 'best move' of each player when faced with the collective action $\mathbf{x}$.

4.5.2. *The Relaxation Algorithm.* The relaxation algorithm is used to find the Nash equilibrium of a game by starting with an initial estimate of the Nash equilibrium, like $\mathbf{x}_0$. The relaxation algorithm, when $Z(\mathbf{x})$ is single-valued, is

$$\mathbf{x}^{s+1} = (1-\alpha_s)\mathbf{x}^s + \alpha_s Z(\mathbf{x}^s) \qquad 0 < \alpha_s \leq 1 \qquad s = 0, 1, 2, \ldots \tag{14}$$

From the initial estimate, an iterate step $s+1$ is constructed by a weighted average of the players' improvement point $Z(\mathbf{x}^s)$ and the current action point $\mathbf{x}^s$. This averaging ensures convergence (see Krawczyk and Uryasev (2000)) to the Nash equilibrium by the algorithm. By taking a sufficient number of iterations of the algorithm, the Nash equilibrium $\mathbf{x}^*$ can be determined with a specified precision.

This can be interpreted that when only one player's payoff function and all players' past actions are computed, then at each stage in the real time process the optimum response for that player is chosen, assuming that the other players will play as they did in the previous period. In this way, convergence to the Nash normalised equilibrium will occur as $t \to \infty$.

4.5.3. *The relaxation algorithm with an optimised step-size.* In order for the algorithm to converge, any sequence of step-sizes (α_s) may be chosen between each iteration to converge to the Nash equilibrium. Suitable step-sizes may be obtained by trial and error when the step-sizes are $0 < \alpha < 1$. In many cases, it is found that using a constant step of $\alpha_s \equiv 0.5$ leads to a quick convergence.

Although step-sizes may be chosen by the user, the optimum step-size can be used also. A step-size may be optimised by the following definition:

Definition 4.13. *Suppose that we reach* $\mathbf{x}^s$ *at iteration* s*. Then* α_s^* *is* one-step optimal *if it minimises the optimum response function at* $\mathbf{x}^{s+1}$*. That is, if*

$$\alpha_s^* = \arg \min_{\alpha \in [0,1)} \left\{ \max_{\mathbf{y} \in X} \Psi(\mathbf{x}^{s+1}(\alpha), \mathbf{y}) \right\}. \tag{15}$$

(Recall that $\mathbf{x}^{s+1}$ *depends on* α*.)*

By optimising the step sizes, there are fewer iterations, but each step takes longer than when using constant step sizes.

4.6. **Definition Summary.** The Nikaido-Isoda function and the relaxation algorithm can be summarised as follows:

(1) The Nikaido-Isoda function is used as an indicator to tell when the Nash equilibrium has been (approximately) reached when the function's summands are (approximately) zero. This is where the players are at the Nash equilibrium and no other action can improve the players' situation.
(2) The relaxation algorithm is used to allow a convergence to the Nash equilibrium. Convergence is created by the algorithm taking steps (user specified or optimised). Each iterate step $s+1$ is found by a weighted average of the player's improvement point $Z(\mathbf{x}^s)$ and the current action point $\mathbf{x}^s$.
(3) As the Nash equilibrium points is approached the improvement point and the current action point become increasingly close. At the equilibrium point the difference between the two is zero.

References

Haurie A, Krawczyk JB (2002) An Introduction to Dynamic Games. Internet Textbook, URL http://www.vuw.ac.nz/staff/jacek_krawczyk/dyngam/e_textbook_HaurieKrawczyk.pdf

Haurie A, Krawczyk JB, Zaccour G (2012) Games and Dynamic Games, Business Series, vol 1. World Scientific – Now Publishers

Krawczyk JB (2005) Coupled constraint Nash equilibria in environmental games. Resource and Energy Economics 27:157–181

Krawczyk JB, Townsend W (2014a) A MATLAB application which solves for couple-constraint Nash equilbria – NIRA-GUI Code. URL https://code.google.com/p/nira-gui/

Krawczyk JB, Townsend W (2014b) NIRA-GUI: A Matlab application which solves for coupled-constraint Nash equilibria from a symbolic specification – NIRA Manual. SEF Working Paper 07/2014, Victoria University of Wellington, URL https://docs.google.com/file/d/0B39JLTTdR-V8cVVVT0M5QVU4Z0k/edit

Krawczyk JB, Uryasev S (2000) Relaxation algorithms to find Nash equilibria with economic applications. Environmental Modelling and Assessment 5:63–73

Krawczyk JB, Zuccollo J (2007) `NIRA-3:` An improved MATLAB package for finding Nash equilibria in infinite games. Munich Personal RePEc Archive, URL http://mpra.ub.uni-muenchen.de/1119/

McKelvey RD (1991) A Liapunov function for Nash equilibria. Tech. rep., California Institute of Technology

Rosen JB (1965) Existence and uniqueness of equilibrium points for concave n-person games. Econometrica 33(3):520–534