# NIRA-3: An improved MATLAB package for finding Nash equilibria in infinite games

Krawczyk, Jacek and Zuccollo, James
Victoria University of Wellington

December 2006

NIRA-3

# AN IMPROVED MATLAB PACKAGE FOR FINDING NASH EQUILIBRIA IN INFINITE GAMES

JACEK B. KRAWCZYK & JAMES ZUCCOLLO

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 together in a suite of MATLAB routines. This paper updates the MATLAB suite described in [1] by adapting them to MATLAB 7. The suite is now capable of solving both static and open-loop dynamic games. An example solving a coupled constraints game using the suite is provided.

Working Paper
NIRA-3

*JEL* Classification: C63 (Computational Techniques), C72 (Noncooperative Games), C87 (Economic Software).

*AMS* Categories: 91A06, 91A35; 90-08.

*Authors'* Keywords: Nikaido-Isoda function; Coupled constraints.

This report documents 2005 - 2006 research into
Computational Economics Methods
directed by Jacek B. Krawczyk
supported by VUW FCA FRG-05 (24644).

Correspondence should be addressed to: Jacek B. Krawczyk

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

## Contents

## 1. Introduction

Game theory problems are well known for being difficult to solve. The MATLAB routines described in this document provide a numerical method for solving games of varying complexity for their Nash equilibria.

The software uses the Nikaido-Isoda function and the relaxation algorithm (both defined in section 2.5) to find the unique Nash equilibrium of $n$-player problems. Problems with a unique equilibrium have been focussed 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 [8]. Coupled constraints force players to restrict their collective actions and are a common part of many environmental problem specifications. The Nikaido-Isoda/Relaxation Algorithm (NIRA) software is particularly suited to solving such problems.

This paper comprises six sections. Section 2 gives a simple introduction to the concepts underlying NIRA. For a more detailed explanation refer to [6] and [5].

Section 3 explains the programme design that is used to implement NIRA and compute the Nash equilibria.

Section 4 explains the key MATLAB functions that are used in the programme. This is to allow more advanced MATLAB users to have finer control over the output of the programme.

An example using a coupled constraints environmental problem is presented in section 5. New users will find this section particularly helpful when using the software for the first time.

An Appendix demonstrates how the software can be used without much modification to solve open-loop dynamic games with coupled constraints.

## 2. Definitions and concepts

This section provides a brief explanation of the conventions used in this paper and of the concepts underlying the software. Conventions and notations used in this paper are explained in section 2.1. Section 2.2 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 2.3 since this concept may be new to some readers. Section 2 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 [8] and [4].

2.1. **Conventions.**

**Convention 2.1.** *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. $x_i \in I\!R^{m_i}$*), and the collective action will be in boldface (*e.g. $\mathbf{x} \in I\!R^{m_1} \times \cdots \times I\!R^{m_n}$*). In this notation,* $\mathbf{x} = (x_1, \ldots, x_n)$*.*

**Convention 2.2.** *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.*

2.2. **The structure of the game.**

**Definition 2.3.** *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* $\phi_i$ *is the payoff function of player* $i$*. These payoff functions are assumed to be continuous in* $\mathbf{x} \in X \subset I\!R^m$ *and concave in* $x_i \in I\!R^{m_i}$ *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

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

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 $i$th 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.

2.3. **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. These games are characterised by one player's action affecting 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.

2.4. **Existence and uniqueness of equilibrium points.** The obvious question to ask before using any software to discover the equilibrium of such a game is whether an equilibrium exists at all and, if it does exist, is it unique? The conditions for existence and uniqueness for games with coupled constraints are more intricate than those for games with uncoupled constraints and they will be dealt 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]

2.4.1. *Uncoupled constraints.*

**Theorem 2.4.** *An equilibrium point exists for every concave $n$-person game.*

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 equilibria.

**Definition 2.5.** *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 2.6.** *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.*

[1]Unless the technique of deflection, successful in the case of finite games (see [7]), could be extended to infinite games. This is a topic for independent research.

2.4.2. *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 $h_\ell(\mathbf{x})$, $\ell = 1, \dots, L$ 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 $\lambda_i^*$. 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 2.7.** *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 2.8.** *There exists a normalised equilibrium point to a concave* $n$*-person game for every* $\mathbf{r} > 0$.

**Theorem 2.9.** *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.

## 2.5. **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.

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

**Definition 2.10.** *Let $\phi_i$ be the payoff function for player $i$ and $X$ a collective strategy space as before. The Nikaido-Isoda function $\Psi : X \times X \to I\!R$ 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 2.11.**

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

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 2.11 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 [6].

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 $\varepsilon$ is chosen such that, when $\max_{\mathbf{y} \in I\!R^m} \Psi(\mathbf{x}^s, \mathbf{y}) < \varepsilon$, the Nash equilibrium would be achieved to a sufficient degree of precision [6].

**Definition 2.12.** *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 $\mathbf{x}$.

2.5.2. *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

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

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 [6]) 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 $t \to \infty$.

2.5.3. *The relaxation algorithm with an optimised step-size.* In order for the algorithm to converge, any sequence of step-sizes $(\alpha_s)$ 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 2.13.** *Suppose that we reach $\mathbf{x}^s$ at iteration $s$. Then $\alpha_s^*$ 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 $\alpha$.)*

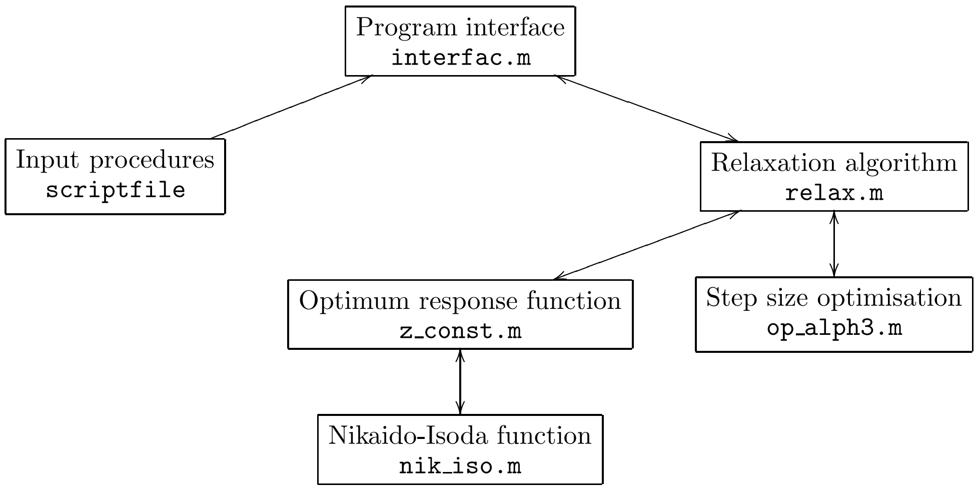
By optimising the step sizes, there are fewer iterations, but each step takes longer than when using constant step sizes.

2.6. **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.

## 3. MATLAB Programme

3.1. **MATLAB Programme Design.** The MATLAB programme features six main MATLAB m-file functions that are used to find the Nash equilibrium of a game. The programme design is shown in the flow diagram below.



3.2. **Description of the Procedures.** This section contains a description of each of the MATLAB programme files.

3.2.1. *Programme interface* `interfac.m`. A user interface is created by the file `interfac.m`. The file reads in the required parameters (entered by the user in `scriptfile`) which describes the game to be solved. The parameters are then fed to the relaxation algorithm and subsequently to the maximum response function `z_const.m`. When the relaxation algorithm has solved the problem, `interfac.m` then outputs the results to the user's screen.

3.2.2. *Parameter input* `scriptfile`. The parameters are entered into the programme through use of a MATLAB m-file script which takes no arguments and returns all required parameters to the programme. The user may choose to use the current example files explained in Section 5 and substitute in their own game parameters or create new parameter files.

The required parameters and their respective scriptfile abbreviations, are:

(1) the names of the payoff function and the constraint function.
   (a) The payoff function, `relaxfun` and the constraint function, `constrfun` are defined in separate files. The user needs to enter the payoff and constraint functions respectively in these files. See sections 3.2.3 and 3.2.4 for more details.

   (b) Where there are no constraints the user must leave an empty space in the scriptfile, that is,
   `constrfun = ‘’` or `constrfun = [ ];`

(2) `description`: a short description of the game;

(3) `gametype`: this string variable determines whether NIRA computes an open-loop dynamic equilibrium (see appendix A) or the equilibrium of the static game. The parameter takes the value `‘openloop’` or `‘static’` as appropriate;

(4) `dims`: a vector giving the dimensions of each player’s action space. This parameter is a vector since the dimensions of each player’s action set need not be identical;

(5) `lowerbound and upperbound`: the lower and upper bounds which the players’ actions cannot exceed. These are the constraints upon individual actions and are thus defined as vectors;

(6) `precision`: sets the termination conditions for the program. `precision` is a $1 \times 2$ vector: The first value sets Nikaido-Isoda function’s termination limits while the second sets the tolerance limit for the difference between the improvement point and the current value of $\mathtt{x}$;

(7) `maxits`: the maximum number of iterations allowed by the relaxation algorithm;

(8) `start`: a starting guess estimate of the Nash equilibrium for the relaxation algorithm. This is entered as a vector of actions for each player;

(9) `alphamethod`: the method for obtaining alphas. The value of alphamethod tells the relaxation algorithm the type of step-size to use.
   (a) $0 <$ `alphamethod` $< 1$ means $alpha =$ `alphamethod`, therefore the set steps used are not optimised;
   (b) `alphamethod` $= 3$ optimises the alphas to be used in the next step;
   (c) see Section 2.5.3 for an explanation of alpha;

(10) `TolCon`, `TolFun` and `TolX`: When minimising the function, these are the positive scalars that define the termination tolerance on the constraint violation, function value and $X$ respectively.

It is helpful to study the preceding definitions with reference to the example scriptfile provided with the software (`river_j.m`). This allows the user to better understand the syntax requirements of each parameter.

3.2.3. *The payoff function.* This takes the collective action and the player number and returns the corresponding payoff. The user will have to create a file containing the payoff function. The example file, `river_fj.m` may be used with the appropriate payoff function substituted.

The function should have a header of the form:

```
function f = relaxfun(x, i)
```

where `f` is the payoff for player `i` from the collective action vector `x`.

3.2.4. *The constraint function.* This function should return the value of inequality constraints as a vector `c` and the value of equality constraints as a vector `ceq`, where inequality constraints are written in the form $c(\mathbf{z}) \leq \mathbf{0}$ and equality constraints are written in the form $ceq(\mathbf{z}) = \mathbf{0}$.

The function should have a header of the form:

```
function [c, ceq] = constrfun(z, x, dims)
```

where `z` is the input vector that is taken by the constraint function. The vectors `x` and `dims` are required to be in the header for technical reasons but need not be explicitly used by the function. The function name `constrfun` may be replaced by the name of the user's constraint function.

The example file, `river_cj.m` may be used with the correct constraint function entered.

3.2.5. *The relaxation algorithm* `relax.m`. This is the main procedure containing the control loop for the relaxation algorithm. This function takes the required parameters from `interfac.m` and iterates the relaxation algorithm using the specified step-sizes ($\alpha$). The iterations are continued until the termination conditions have been reached by way of the Nikaido-Isoda function (i.e. Nikaido-Isoda function is less than `precision(1)` and the difference in $\mathbf{x}$ between iterations is less than `precision(2)`). The relaxation algorithm then returns the normalised Nash equilibrium to `interfac.m` together with the points traversed in achieving the solution, the individual payoffs at each step, the step sizes ($\alpha_s$) used and the time and number of iterations taken.

3.2.6. *Nikaido-Isoda function* `nik_iso.m`. This finds the Nikaido-Isoda function and returns the value of the Nikaido-Isoda function at the point $(\mathbf{y}, \mathbf{x})$. The order of the arguments is the reverse of that in the paper for technical reasons. The Nikaido-Isoda function is fed into the optimum response function `z_const.m` to be maximised.

3.2.7. *Optimum response function* `z_const.m`. This returns the $\mathbf{y}$ which maximises the Nikaido-Isoda function at the point $\mathbf{x}$. As mentioned above, this file contains the command for the function `fmincon` that maximises the Nikaido-Isoda function.

3.2.8. *Step-size optimisation general method* `op_alph3.m`. This general method of step-size optimisation for $\alpha$ works by trying to minimise the maximum of the Nikaido-Isoda function so that the next step used by the algorithm is optimised.

## 4. Important MATLAB Functions

The MATLAB programme uses the two functions, `fmincon` and `optimset` to help solve the problem. Examples of both functions can be found in the file `z_const.m`.

### 4.1. `fmincon`.

4.1.1. *Purpose.* `fmincon` is used to find the minimum of a constrained (nonlinear) function of several variables starting at an initial estimate. That is, it solves:

$$min_x = f(x) \tag{16}$$

subject to $AX \leq B$, $AeqX = Beq$, $C(x) \leq 0$, $Ceq(x) = 0$, and $LB \leq x \leq UB$;

where $X$, $B$, $Beq$, $LB$, $UB$ are vectors; $A$, $Aeq$ are matrices; $C(x)$, $Ceq(x)$ are functions that return vectors; $f(x)$ is a function that returns a scalar. Note that $f(x)$, $C(x)$, and $Ceq(x)$ can be nonlinear functions.

4.1.2. *Syntax.* The `fmincon` function requires input to be provided by input arguments defined by the user within the general parameter file.

```
[X] = fmincon ('fun', X0, A, B, Aeq, Beq, LB, UB, NONLCON, OPTIONS,
                                P1, P2,...);
```

Input Arguments. `'fun'`, $X_0$:
`'fun'` is the name of the function that is to be minimised. For solving the Nash equilibria, it is the Nikaido-Isoda function `nik_iso.m` being minimised. The starting estimate of the minimised point is given by, $X_0$.

`A` and `B`:
`A` and `B` are the coefficients of the function's linear inequality constraints, $AX \leq B$. By default they are not used by NIRA hence empty brackets (`[ ]`) can be used.

`Aeq` and `Beq`:
`Aeq` and `Beq` are the coefficients of the function's linear equality constraints, $AeqX = Beq$. In this implementation for solving Nash equilibria it does not apply, hence empty brackets (`[ ]`) can again be used.

`LB` and `UB`:
`LB` and `UB` defines a set of lower (`LB`) and upper (`UB`) bounds on the variable $X$, so that the solution is within these two bounds.

`NONLCON`:
`NONLCON` subjects the function `'fun'` to nonlinear inequalities $C(X)$ and equalities $Ceq(X)$. This is the constraint used by NIRA hence `NONLCON` is the constraint function. The reason for using `NONLCON` instead of `A`, `B`, `Aeq` and `Beq` is that `NONLCON` can call either linear or non linear functions and both types of constraint can be incorporated in to a single file.

`OPTIONS`:
`OPTIONS` allow the user to define the optimisation parameters by way of the utility function, `OPTIMSET` (see Section 4.2).

`P1` and `P2`:
Any other input argument placed after `OPTIONS` are problem-dependent parameters that apply directly to the functions `'fun'` (and `NONLCON`). To solve the present type of games the problem-dependent parameters are $X$ and the vector giving the dimensions of each player's action space, `dims`. Remember that in MATLAB you can also transfer parameters between functions through `global` (see section 5.2.1).

Where any of the input arguments other than `'fun'` and $X_0$ are not needed, empty matrices must be entered. For further information, users should refer to MATLAB's help section by typing `help fmincon` in the command window.

## 4.2. `optimset`.

4.2.1. *Purpose.* `optimset` is a utility function that allows the user to specify the `OPTIONS` under `fmincon`. This will allow `OPTIONS` to define the optimisation parameters while a function is being minimised by `fmincon`. This section will only cover parameters that are needed to solve games with coupled-constraints. Please refer to MATLAB's help section by typing `help optimset` for more details.

`optimset` is executed in `op_alph3`, `relax` and `z_const`.

4.2.2. *Syntax.* `optimset` should be defined before `fmincon`; however, in NIRA the user enters the following parameters in `scriptfile` and they are then passed to `optimset` as required.

```
options = optimset('parameter', parameter value, 'parameter',
                            parameter value);
```

Parameters. When entering parameters for optimset, word values need to be within apostrophes, `' '`. Parameter values within [ ] are the default values if the parameter is not specified. For solving games with coupled constraints, the medium-scale algorithm will suffice.

`‘LargeScale’, ‘[on]/off’`:
The nature of the present type of games only need a medium-scale algorithm. Therefore `LargeScale` would be `off`.

`‘MaxFunEvals’, positive integer`:
The parameter of `‘MaxFunEvals’` states the maximum number of function evaluations allowed.

`‘TolCon’, positive scalar`:
`‘TolCon’` is used to set a parameter on the termination tolerance on the constraint violation.

`‘TolFun’, positive scalar`:
`‘TolFun’` sets a parameter on the termination tolerance on the function value.

`‘TolX’, positive scalar`:
`‘TolX’` sets a parameter on the termination tolerance on $X$.

## 5. Problem Example - a Static Game with Coupled Constraints

This example covers a basic coupled constraints game which may be adapted by the user to solve other infinite games, for instance, infinite games such as those without any constraints. The example is the same as in [1] and [6] with the addition of an optimised step-size (i.e. $\alpha = 3$). Later, the example will briefly explore the differences in step-sizes by introducing $\alpha = 0.5$.

In a game of coupled constraints, the players' action set is a general convex set $X \subset \mathbb{R}^n$. Thus all players' actions are subjected to one or more common constraints.

### 5.1. **Formulation of a coupled constraints game.** *Problem: River Basin Pollution,* [3].

In this game, we consider three players $j = 1, 2, 3$ located along a river. Each agent is engaged in an economic activity at a chosen level $x_j$, but the players must meet environmental conditions set by a local authority.

Pollutants may be expelled into the river, where they disperse. Two monitoring stations $\ell = 1, 2$ are located along the river, at which the local authority has set maximum pollutant concentration levels.

The revenue for player $j$ is

$$\mathcal{R}_j(\mathbf{x}) = [d_1 - d_2(x_1 + x_2 + x_3)]x_j \tag{17}$$

with expenditure

$$\mathcal{F}_j(\mathbf{x}) = (c_{1j} + c_{2j}x_j)x_j. \tag{18}$$

| Player $j$ | $c_{1j}$ | $c_{2j}$ | $e_j$ | $\delta_{j1}$ | $\delta_{j2}$ | $a_1^j$ | $a_2^2$ | $d_1$ | $d_2$ |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 0.10 | 0.01 | 0.50 | 6.5 | 4.583 | 6.5 | 4.583 | 3.00 | 0.01 |
| 2 | 0.12 | 0.05 | 0.25 | 5.0 | 6.250 | 5.0 | 6.250 | - | - |
| 3 | 0.15 | 0.01 | 0.75 | 5.5 | 3.750 | 5.5 | 3.750 | - | - |

TABLE 1. User defined parameters `par`

Thus the net profit for player $j$ is

$$\begin{aligned} \phi_j(\mathbf{x}) &= \mathcal{R}_j(\mathbf{x}) - \mathcal{F}_j(\mathbf{x}) \\ &= [d_1 - d_2(x_1 + x_2 + x_3) - c_{1j} - c_{2j}x_j]x_j. \end{aligned} \tag{19}$$

The constraint imposed by the local authority at location $l$ is

$$q_\ell(\mathbf{x}) = \sum_{j=1}^{3} \delta_{j\ell} e_j x_j \leq 100, \ \ \ell = 1, 2. \tag{20}$$

The economic constants $d_1$ and $d_2$ determine the inverse demand law and are set to 3.0 and 0.01 respectively. The values for $c_{1j}$, $c_{2j}$ and the other constants are given in Table 1. The $\delta_{j\ell}$ are the decay-transportation coefficients from player $j$ to location $l$, and $e_j$ is the emission coefficient of player $j$.

5.2. **Solving the River Basin Pollution game.** As mentioned in Section 3.2, the user will have to create three parameter MATLAB m-files in locations visible for the path.

5.2.1. *The general parameter file* - `river_j.m`. A file containing the general parameters will need to be created. This example uses `river_j.m` which can be used with its parameter values changed according to the user's game problem. Alternatively users can create their own parameter file. Such a file should include the parameters described below while subsitituting the user's preferred file name for `river_j`.

The general header for `river_j.m` is:

```
function[relaxfun, constrfun, description, dims,...
lowerbound, upperbound, precision, maxits, start,...
alphamethod, toler] = river_j()
```

The parameters needed to be entered into this file are described in Section 3.2.2 with the exception of `par` which contains user defined parameters specific to the structure of this example. The parameter values of `river_j.m` are:

`global par`: This is added in to allow the files visible in the path to read the user defined parameters - the vector labelled `par`. If there are no extra user defined parameters, then there is no need for this to be added in.

`par`: This example features a matrix of user defined parameters which are used to define other parameters as found in Table 1. This is optional depending on the complexity of the problem that the user wishes to solve. The advantage of using a matrix of parameters in the scriptfile to define all of the parameters used in the constraint and payoff functions is that the user may easily modify his problem through changes to a single matrix.

`relaxfun`: Calls the payoff function m-file, for example, `'river_fj'`.

`constrfun`: Calls the constraint function m-file, for example, `'river_cj'`.

`description`: A short description helps to identify the game, for example, `'River Basin Pollution Game'`.

`dims`: In this game the dimensions of each player's action space will be $[1, 1, 1]$.

`gametype`: This problem is static so for this example `gametype` is set to `'static'`.

`lowerbound`: The lowerbound values for this example are $[0, 0, 0]$ as the players cannot produce at a negative level.

`upperbound`: The upperbound values for this example are $[inf, inf, inf]$.

`precision`: Sets the user's preference for how precise the solution needs to be.For this example it is set to $[1e-5, 1e-5]$.

`maxits`: Is the maximum number of iterations that the user wants the relaxation algorithm to perform. This example will set a limit of 100 iterations.

`start`: The starting estimate of the Nash equilibrium here is $[0, 0, 0]$.

`alphamethod`: This version of the example has the step-size $\alpha$ optimised, therefore the value of `alphamethod` is set to 3.

`TolCon`, `TolFun` and `TolX`: These are also set to the user's preference for the termination tolerance on the constraint violation, function value and $X$. In this case, all of them have been set to $[1e-10]$.

`toler = [TolCon, TolFun, TolX]`: Users do not have to change anything in this parameter. This is used to define the termination tolerance as `toler` so that it other parts of the programme may read these three values.

5.2.2. *The payoff function* - `river_fj.m`. Users will also have to create a m-file containing the payoff (profit) function. This example uses `river_fj.m` as the payoff function.

The header for `river_fj.m` is:

```
function f = river_fj(x, i)
```

Where the user substitutes `river_fj` for the name of their payoff function file (without the `.m` extension included).

Following the header, the user then defines the payoff function. This example has the payoff function represented by several variables and uses the parameters from `par`. Changes to the payoff function can later be made through modifications of the `par` matrix in `river_j.m`. In this case the user needs to define these variables before entering the payoff function.

```
global par;
c = par(1:3,1:2);
e = par(1:3,3);
δjℓ = par(1:3,4:5);
d = par(1,6:7);
```

`global par` is added as above to allow `river_fj` read the user defined parameters `par`.

Once any additional variables have been defined, the payoff function can be defined. So, the next and last line of `river_fj` is:

```
f = (( d(1) - (d(2) * (x(1) + x(2) + x(3))) - c(i,1) - (c(i,2)* x(i)))
*x(i));
```

which corresponds to expression (19).

5.2.3. *The constraint function* - `river_cj.m`. If there is a constraint function in the game, users will also have to create another m-file containing the constraint function. This example uses `river_cj.m` as the constraint function.

The header for `river_cj.m` is:

```
function [c, ceq] = river_cj(z, varargin)
```

Where the user substitutes `river_cj` for the name of their constraint function file (without the `.m` extension included).

Again if there are other variables used in the constraint function then it would have to be defined before the constraint function is entered. If variables have already been defined in another file, as in `river_j.m`, users can use the `global` command to allow the access to the variable along the same path. In this case, it is:

```
global par
```

The constraint function describes the inequalities $c(\mathbf{z}) \leq \mathbf{0}$ and can now be entered as:

```
c(1) = (δ_jℓ(1:3, 1) .* e)' * z'- 100;
c(2) = (δ_jℓ(1:3, 2) .* e)' * z'- 100;
```

which corresponds to equation (20). This problem has no equality constraints so `ceq = [ ]`.

5.3. **Nash equilibrium for the River Basin Pollution Game.** Given the user's creation of the general parameter, payoff and constraint files, the Nash equilibrium for the River Pollution Game can now be found.

The user only needs to enter `interfac` in MATLAB's command window to start the computations. Once activated, `interfac` will start the process described in Section 3.2.

Using the Nash equilibrium starting estimate of $\mathbf{x} = (0, 0, 0)$ and $\alpha \equiv 3$ to optimise the step-sizes, the results start to converge on the Nash equilibrium, as in Table 2. The final result of the Nash equilibrium is given on the bottom of the table. In this example with an optimised step-size, the Nash equilibrium is $[21.1445, 16.0279, 2.7262]$. The above process can also be used to find the Nash

| Iteration (s) | $x_1^s$ | $x_2^s$ | $x_3^s$ | $\alpha_s$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | - |
| 1 | 19.3494 | 17.1858 | 3.7896 | 1.000 |
| 2 | 20.1874 | 16.5216 | 3.3306 | 0.6144 |
| 3 | 20.5693 | 16.2804 | 3.1028 | 0.5257 |
| 4 | 20.7875 | 16.1636 | 2.9663 | 0.5000 |
| 5 | 20.9236 | 16.1007 | 2.8781 | 0.5028 |
| ⋮ | ⋮ | ⋮ | ⋮ | ⋮ |
| 10 | 21.1118 | 16.0343 | 2.7500 | 0.5000 |
| ⋮ | ⋮ | ⋮ | ⋮ | ⋮ |
| 20 | 21.1445 | 16.0279 | 2.7262 | 0.5000 |
| 21 | 21.1445 | 16.0279 | 2.7262 | 0.5000 |

TABLE 2. Convergence in the river basin pollution game using optimised step-sizes

equilibrium using a specified step-size (rather than an optimised one), like $\alpha \equiv 0.5$. To do this, `alphamethod` needs to be changed in the general parameter file so that *alphamethod* $\equiv 0.5$. Table 3 shows the convergence when the step-size is 0.5 and the Nash equilibrium as $[21.1448, 16.0279, 2.7260]$.

It can be seen that the use of an optimised step size tends to reduces the number of iterations required for convergence. However, the tradeoff for the increased efficiency in step size is the time spent calculating the optimal $\alpha$. It is ou experience that the run time for solving with an optimal step size is markedly higher than when using a fixed step size of 0.5. Given the above example used to find the Nash

| Iteration (s) | $x_1^s$ | $x_2^s$ | $x_3^s$ | $\alpha_s$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | - |
| 1 | 9.6747 | 8.5929 | 1.8948 | 0.5 |
| 2 | 14.8530 | 12.6191 | 2.6555 | 0.5 |
| 3 | 17.6535 | 14.4885 | 2.9129 | 0.5 |
| 4 | 19.1847 | 15.3467 | 2.9615 | 0.5 |
| 5 | 20.0316 | 15.7349 | 2.9342 | 0.5 |
| $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ |
| 10 | 21.0162 | 16.0242 | 2.7810 | 0.5 |
| $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ |
| 20 | 21.1439 | 16.0279 | 2.7266 | 0.5 |
| $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ |
| 30 | 21.1448 | 16.0279 | 2.7260 | 0.5 |
| 31 | 21.1448 | 16.0279 | 2.7260 | 0.5 |

TABLE 3. Convergence in the river basin pollution game using a specified step-size - 0.5

equilibrium, users can now adopt the same method for solving their own games of the same type.

## Appendix A. Open-loop coupled constraint equilibria

Dynamic games are becoming an increasingly important tool for economists to use. This section demonstrates how NIRA can be used to solve a particular class of dynamic equilibria where the 'state' variable is unobservable and thus the players' actions cannot depend upon the state of the world in a finite number of future periods. This class of equilibria are often termed 'open-loop' equilibria[2] since there is no feedback loop from the state variable to the player's action in future periods. To make economic sense of this it is necessary to justify restricting the players' information sets such that no player can observe other players' actions.

In this section the concept of open-loop equilibria is briefly canvassed before proceeding to solve an example problem.

A.1. **Open loop equilibria.** Since there is no feedback loop in this class of equilibria the players' optimal actions can be computed as functions of the current state and of the other players' optimal actions. In this way the optimal action path of each player can be computed from the current period to the final period.[3]

A.2. **Solving open loop equilibria with NIRA.** To compute the action path of each player for $T$-periods it is necessary to assign each player a $T$ dimensional action space. Having done that the three parameter files required by NIRA can be written in the normal fashion.

Sufficient mathematical knowledge to construct payoff and constraint functions suitable for dynamic games is assumed here. Note that multiple constraints — one for each period — must be used in order to constrain actions within each time period. The payoff function also needs to be re-expressed in terms of the initial state and all future actions.

A.3. **Solving a dynamic open-loop River Basin Pollution game.** This application of the software will be demonstrated with an example. The game of section 5 will be reformulated to be a game played over a finite number of periods $T$. The agents will have an option to invest in their production capacities, which can depreciate. For simplicity the values of all the game's parameters will remain the same as in the static case for all periods. This model can be found in greater detail in [5].

[2]To make economic sense an open loop game model usually needs to be completed by a finite state('scrap') value function.

[3]Some economists suggest that open-loop dynamic games are essentially static (e.g. [2]). We agree with this view but stress that there is an underlying investment process in this game that makes it relevant to 'dynamic' economics.

A.3.1. *The model.* Let $x_i^{(t)}$ denote the fully utilised production capacity that player $i$ has at time $t$, governed by the state equation

$$x_i^{(t+1)} = (1-\mu_i)x_i^{(t)} + u_i^{(t)} \tag{21}$$

where $t = 0, \dots, T$. The investment made by player $i$ at time $t$ is $u_i^{(t)}$ and $\mu_i \in [0, 1]$ is the depreciation coefficient. The initial condition $x_i^{(0)}$ is fixed for all players.

The collective action and state at time $t$ will be denoted by

$$\mathbf{u}^{(t)} = \left( \underbrace{u_1^{(0)}, \dots, u_1^{(t)}}_{\equiv u_1^{(t)}}, \underbrace{u_2^{(0)}, \dots, u_2^{(t)}}_{\equiv u_2^{(t)}}, \underbrace{u_3^{(0)}, \dots, u_3^{(t)}}_{\equiv u_3^{(t)}} \right) \qquad t = 0, \dots, T-1 \tag{22}$$

and $\mathbf{x}^{(t)} = \left(x_1^{(t)}, x_2^{(t)}, x_3^{(t)}\right) \quad t = 0, \dots, T-1$, respectively and the complete strategy vector $\mathbf{u} \equiv \mathbf{u}^{(T-1)}$.

We assume that the players cannot observe the other players' production capacities. After time $T$ the game can be continued under a different information structure. If the game continues as a feedback game then the utility to players for the period $[0, T]$ will have to be completed by a concave function $\kappa(x_i^{(T)})$ of the state at time $T$. If the game stops at time $T$ then the function $\kappa(x_i^{(T)})$ can be used to represent the capital's 'scrap value' utility. We thus model the present value payoff to player $i$ at $t = 0$ as

$$\phi_i(\mathbf{x}_0; \mathbf{u}) = \sum_{t=0}^{T-1} \left( \phi_i(\mathbf{x}^{(t)}) - c_{3i}(u_i^{(t)})^2 \right) + \rho^T \kappa(x_i^{(T)}) \tag{23}$$

where $\phi_i(\mathbf{x}^{(t)})$ is defined through equation (17) and $c_{3i}(u_i^{(t)})^2$ is the quadratic investment cost at time $t$; $0 < \rho < 1$ is the discount factor.

The constraints in equation (20) are to be satisfied for all periods. The current production capacity $x_i^{(t)})$ must satisfy the state equation and must also remain non-negative. These requirements define a convex constraint set $\mathbf{U}$ given explicitly by equation (26) below.

A.3.2. *Reformulating the problem.* To find the coupled constraint equilibrium

$$\mathbf{u}^* = \arg \operatorname{equil}_{\mathbf{u} \in \mathbf{U}} \{\phi_1, \phi_2, \phi_3\} \tag{24}$$

it is necessary to reformulate the payoff and constraint functions so that they contain only constants and actions.

Assume that $\kappa_i(\cdot) \equiv \kappa_i\sqrt{\cdot}$. The reformulated payoff functions for $i = 1, 2, 3$ are

(25)

$$\phi_i(\mathbf{x}_0; \mathbf{u}) = \sum_{t=0}^{T-1} \rho^t \left\{ \phi_i \left( \begin{bmatrix} (1-\mu_1)x_1^{(0)} + \sum_{s=1}^{t}(1-\mu_1)^{t-s}u_1^{(s)} \\ (1-\mu_2)x_2^{(0)} + \sum_{s=1}^{t}(1-\mu_2)^{t-s}u_2^{(s)} \\ (1-\mu_3)x_3^{(0)} + \sum_{s=1}^{t}(1-\mu_3)^{t-s}u_3^{(s)} \end{bmatrix} \right) - c_{3i}(u_i^{(t)})^2 \right\}$$

$$+ \rho^T \kappa_i \sqrt{(1-\mu_i)x_i^{(0)} + \sum_{s=1}^{T}(1-\mu_i)^{t-s}u_i^{(s)}}$$

where the sums over $s$ are interpreted as zero if $t$ or $T$ are zero.

The constraint functions are

$$q_\ell(\mathbf{u}^{(t)}) = \sum_{i=1}^{3} \delta_{j\ell} e_j \left( (1-\mu_i)x_i^{(0)} + \sum_{s=1}^{t}(1-\mu_i)^{t-s}u_i^{(s)} \right) \leq K_\ell^{(t+1)} \tag{26}$$

for $\ell = 1, 2$ and $t \in \{0, 1, \ldots, T-1\}$. The sum over $s$ is again treated as zero if $t = 0$. From section 5.1, $K_\ell^{(t+1)} = 100$.

A.3.3. *Constructing the NIRA files.* The three files required are the same as in section 5.2 with the appropriate functions substituted in.

**The general parameter file `rived_n`:** The parameter file contains all of the items described in section 5.2.1 with the following modifications:

- `gametype = ‘openloop’;`
- The parameter matrix `par` includes the number of periods $T$. $T$ is an important parameter to use in defining the payoff and constraint functions. In this example we have solved a four period game.

**The payoff function `rived_fn`:** This file follows the same format as in the static game (see section 5.2.2) with the payoff function in equation 25 substituted in for the static game's payoff function.

**The constraint function `rived_cn`:** Since there are four periods in this game( $T = 4$) and two constraints in each period ($\ell = 1, 2$) the constraint file defines eight separate constraints. The structure of the file is identical to the structure in the static game(see section 5.2.3).

A.3.4. *Open loop Nash equilibrium for the River Basin Pollution game.* Let $T = 4$, $\mu_i \equiv 0$, $c_{3i} \equiv 1$, $k_i = 1$, $\mathbf{x}^{(0)} = (21.149, 16.030, 2.722)$ and $\rho = 0.9$. Using a starting guess of $\mathbf{u} = ((0,0),(0,0),(0,0))$ gives the solution detailed in Table 4. The program's output also gives the constraint values and Lagrange multipliers associated with the constraints. Examining the multipliers for this problem shows $\lambda_1 = (0.5169, 0.4651, 0.4186, 0.0392)$ and $\lambda_2 = (0, 0, 0, 0)$. From this it can be deduced that the constraints at the second measuring point are not binding since $\lambda_2$ is zero. The decreasing value of $\lambda_1$ indicates that the polluters are investing

|  | State |  |  | Action |  |  |
|---|---|---|---|---|---|---|
| Period\Player | 1 | 2 | 3 | 1 | 2 | 3 |
| 1 | 21.1490 | 16.0300 | 2.7220 | -0.025759 | 0.012508 | 0.016505 |
| 2 | 21.1232 | 16.0425 | 2.7385 | -0.029026 | 0.014691 | 0.018418 |
| 3 | 21.0942 | 16.0572 | 2.7569 | -0.033173 | 0.017906 | 0.020711 |
| 4 | 21.0610 | 16.0751 | 2.7776 | -0.038377 | 0.022441 | 0.023436 |
| Final value | 21.0227 | 16.0975 | 2.8011 | N/A | N/A | N/A |

Table 4. Nash equilibrium in the dynamic River Basin Pollution game using $\alpha = 0.5$.

in such a way that they minimise their costs of compliance which is the expected result.

## References

1. S Berridge and J B Krawczyk, *Relaxation algorithms in finding Nash equilibria*, Research Papers in Economics, 1997, RePEc:wpa:wuwpco:9707002.
2. D Fudenberg and J Tirole, *Game theory*, MIT Press, Cambridge, Massachusetts, London, England, 1991.
3. A Haurie and J B Krawczyk, *Optimal charges on river effluent from lumped and distributed sources*, Environmental Modelling and Assessment **2** (1997), no. 3, 93–106.
4. A. Haurie and J. B. Krawczyk, *An introduction to dynamic games*, Internet textbook, 2002, http://ecolu-info.unige.ch/∼haurie/fame/textbook.pdf.
5. J B Krawczyk, *Coupled constraint Nash equilibria in environmental games*, Resource and Energy Economics **27** (2005), 157–181.
6. J B Krawczyk and S Uryasev, *Relaxation algorithms to find Nash equilibria with economic applications*, Environmental Modelling and Assessment **5** (1999), 63–73.
7. R D McKelvey, *A Liapunov function for Nash equilibria*, Tech. report, California Institute of Technology, 1991.
8. J B Rosen, *Existence and uniqueness of equilibrium points for concave n-person games*, Econometrica **33** (1965), no. 3, 520–534.