University of Padua

DEPARTMENT OF MATHEMATICS “TULLIO LEVI-CIVITA”
Faculty of Mathematical, Physical and Natural Sciences

Master's degree thesis

Extrapolation Methods for Quasi-Variational Inequalities

Candidate:
Chiara Todesco
Official No: 1155035

Advisor:
Prof. Francesco Rinaldi

Co-Advisor:
Dr. Stefano Cipolla

18 October 2019

Contents

List of Algorithms

List of Figures

List of MATLAB Codes

Introduction

Equilibrium problems, i.e., problems having as solution a condition or a state of the system where all the competing influences are balanced, have been widely used to model phenomena coming from different areas of science.

A further generalization of this kind of problems is represented by quasi-equilibrium problems: those ones represent a specific class of equilibrium problems whose feasible regions are subject to changes according to the point considered as a candidate solution. Variable feasible regions are well suited to model situations in which the agents share resources or, more generally, when their supposed behavior may influence the behaviors of other agents. As one can easily imagine, even though such a class of problems allows to model a broader variety of phenomena, the fact that the feasible regions are variable, represents a quite challenging complication from the theoretical/modeling point of view.

Quasi-Variational Inequalities (QVIs) represent a very important tool to model different classes of quasi-equilibrium problems. This is the reason why many reasearchers in different fields focus their studies on this subject.

Just to give an idea of the relevance that QVIs have in applications, we mention the fact that the generalized Nash Equilibrium problem, which is used to model plenty of different applications in engineering, economics and so on, (see, e.g., [1, 22, 21]), is strictly related to the solution of a QVI.

Several algorithmic approaches have been devised for QVIs: fixed points and projections methods [30, 14], penalization of coupling constraints method [6, 20], KKT based methods [7] and Newton type methods [18, 19]. Since QVIs can be reformulated as a fixed point problem, it seems quite natural and straightforward to solve those problems using fixed point methods.

In this thesis we hence focus on fixed point methods and, more specifically, on projection methods. The main reasons why we choose to analyze those iterative methods are the following:

1. They can be used in different scenarios, without having a deep knowledge of the considered problem;

2. They are easy to implement (especially if we consider problems with simple bounds or linear constraints);

3. They have limited storage requirements;

4. They can easily exploit any sparsity or separable structure of the corresponding constrained sets.

Despite the great amount of research that has been devoted, projection-based approaches for QVIs, in some cases, do not seem to guarantee good practical performance. This is the reason why, in this thesis, we propose new strategies to improve effectiveness and robustness of those methods.

More specifically, we will focus on "*Algorithm 1b*" in [26] and on"*Algorithm QVI*" in [15], which we will call Generalized Solodov and Nguyen-Strodiot respectively. These two methods belong to the class of hybrid extragradient methods. This class computes first a single projection onto the feasible set to get a trial point and, afterwards, performs a line search procedure between the current approximation and the trial point to obtain the prediction step. Once this step has been calculated, the correction step is obtained thanks to a search direction and a step length.

These methods could be affected by an extremely low rate of convergence, and, for this reason we propose to couple these methods with some suitable extrapolation techniques (see, e.g., [3]). Extrapolation is a technique commonly used to accelerate the convergence of a sequence in a vector space: it is able to transform a slowly convergent sequence into a new one which converges faster. In recent times, the use of these techniques has been applied quite successfully to different computational frameworks and it is a research topic currently in full development.

We will consider two type of extrapolation techniques, that is the regularized nonlinear acceleration developed in [24] and the regularized topological Shanks type acceleration developed in [4]. These techniques compute estimates of the optimum from a nonlinear average of the iterates produced by a given iterative method. The weights in this average are computed via a simple linear system. It is important to note that acceleration schemes run in parallel to the base algorithm, providing improved estimates of the solution on the fly, while the original method is running.

Finally some numerical results are displayed to show the behavior of the two hybrid extragradient algorithms combined with the two types of acceleration to solve generalized Nash equilibrium problems.

The thesis is organized as follows. In Chapter 1 we formally state the QVI problem and summarize some definitions and results; in particular we reformulate a QVI problem in a fixed-point fashion. In Chapter 2 we present the Generalized Solodov method and Nguyen-Strodiot method and report some convergence results. In Chapter 3 we introduce the regularized nonlinear acceleration and the regularized topological Shanks acceleration and describe the way we embedded them in the two hybrid extragradient methods. In Chapter 4 we present some numerical results and some concluding remarks, while, in Chapter 5 we display the MATLAB codes used for the numerical experiments.

Chapter 1

Preliminaries

1.1 Preliminaries and problem statement

In this section we will give some preliminaries needed to understand the abstract concept of quasi-equilibrium problem and the theory around it. We then focus on quasi variational inequality problems, that are a particular case of a quasi equilibrium problems.

Definition 1.1. Let $\emptyset \neq X \subseteq \mathbb{R}^n$ be a closed convex set, $K : X \rightrightarrows X$ be a multivalued mapping such that $\forall x \in X, \quad \emptyset \neq K(x) \subseteq X$ is closed convex. Let $f : X \times X \to \mathbb{R}$ be an equilibrium bi-function , i.e., it satisfies $f(x, x) = 0 \quad \forall x \in X$ and $f(x, \cdot)$ be a convex function on X. The *quasi − equilibrium problem*, denoted with $QE(K; f)$, consists in

$$\text{find } x^* \in K(x^*) \text{ s.t. } f(x^*, y) \geq 0 \quad \forall y \in K(x^*).$$

Definition 1.2. Let $\emptyset \neq X \subseteq \mathbb{R}^n$ be a closed convex set, $K : X \rightrightarrows X$ be a multivalued mapping such that $\forall x \in X, \quad \emptyset \neq K(x) \subseteq X$ is closed convex. Let $F : X \to \mathbb{R}^n$ be a monotone operator. The *quasi − variational inequality problem*, denoted with $QVI(K; F)$, consists in

$$\text{find } x^* \in K(x^*) \text{ s.t. } \langle F(x^*), y - x^* \rangle \geq 0 \quad \forall y \in K(x^*).$$

Remark 1.1. The problem $QVI(K; F)$ is a $QE(K; f)$ where $f(x, y) = \langle F(x), y - x \rangle$ with $F : X \to \mathbb{R}^n$.

Definition 1.3. Let $\emptyset \neq K \subseteq \mathbb{R}^n$ be a closed convex set. Let $F : K \to \mathbb{R}^n$ be a continuous operator. The *variational inequality problem*, denoted with $VI(K; F)$, consists in

$$\text{find } x^* \in K \text{ s.t. } \langle F(x^*), y - x^* \rangle \geq 0 \quad \forall y \in K.$$

Remark 1.2. The problem $VI(K; F)$ is a $QVI(K; f)$ where $K(x)$ is a fixed constraint set, say, $K(x) \equiv K \ \ \forall x \in X$.

Througout the thesis the following definitions will be used:

Definition 1.4. Given $\mu \in \mathbb{R}$, a map $F : \mathbb{R}^n \to \mathbb{R}^n$ is called

- $\mu -$ *monotone* on K if the inequality

$$\langle F(x) - F(y),\ x - y\rangle \geq \mu\|x - y\|^2$$

  holds $\forall\, x, y \in K$.

- $\mu -$ *pseudomonotone* on K if the implication

$$\langle F(y),\ x - y\rangle \geq 0 \implies \langle F(x),\ x - y\rangle \geq \mu\|x - y\|^2$$

  holds $\forall\, x, y \in K$.

If $\mu > 0$, F is also called *strongly* $($*pseudo*$)$*monotone*, if $\mu < 0$, F is also called *weakly* $($*pseudo*$)$*monotone* and if $\mu = 0$, F is also called $($*pseudo*$)$*monotone*.

Definition 1.5. Given $\mu \in \mathbb{R}$, a bi-function $f : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}^n$ is called

- $\mu -$ *monotone* on K if the inequality

$$f(x, y) + f(y, x) \leq -\mu\|x - y\|^2$$

  holds $\forall\, x, y \in K$.

- $\mu -$ *pseudomonotone* on K if the implication

$$f(x, y) \geq 0 \implies f(y, x) \leq -\mu\|x - y\|^2$$

  holds $\forall\, x, y \in K$.

If $\mu > 0$, f is also called *strongly* $($*pseudo*$)$*monotone*, if $\mu < 0$, f is also called *weakly* $($*pseudo*$)$*monotone*, and if $\mu = 0$, f is also called $($*pseudo*$)$*monotone*.

Remark 1.3. f is *strictly monotone* at $x \in K$ if $\forall\, y \in K$, $y \neq x$, we have

$$f(x, y) + f(y, x) < 0.$$

Definition 1.6. Let $\emptyset \neq X \subseteq \mathbb{R}^n$ be a closed convex set. A multi-value map $K : X \rightrightarrows X$ is said to be

- *upper semicontinuous* (u.s.c.) at $\bar{x} \in X$ if

$$\left.\begin{aligned} x^k \subset X \text{ and } x^k &\xrightarrow{k\to\infty} \bar{x} \\ y^k &\in K(x^k) \\ y^k &\xrightarrow{k\to\infty} \bar{y} \end{aligned}\right\} \implies \bar{y} \in K(\bar{x}).$$

- *lower semicountinuos* (l.s.c.) at $\bar{x} \in X$ if $\bar{x} \in X$ and $x^k \xrightarrow{k\to\infty} \bar{x}$, then $\forall\, \bar{y} \in K(\bar{x})\ \exists\ \{y^k\}$ with $y^k \in K(x^k)$, s.t. $y^k \xrightarrow{k\to\infty} \bar{y}$.

- *continuos* on X if K is u.s.c. and l.s.c. at every point of X.

1.2 Equivalent Reformulation

This section is devoted to the reformulation of the $QE(K;f)$ as another problem with the same set of solution. In particular we will show that $QE(K;f)$ can be reformulated as a fixed point problem. With this aim, consider the multi-value map $Y : \mathbb{R}^n \rightrightarrows \mathbb{R}^n$ given by

$$Y(x) = \arg\min\{f(x,y) \,:\, y \in K(x)\}$$

which could be possibly empty. The fixed point of Y coincide with the solution of $QE(K;f)$.

Theorem 1.2.1 ([2]). *The point $x^* \in K(x^*)$ solves $QE(K;f)$ if and only if $x^* \in Y(x^*)$.*

The next equivalent QE play a key role in many solution methods.

Corollary 1.2.2 ([2]). *Suppose $f(x,\cdot)$ is τ-convex $\forall x \in K(\bar{x})$ with $\tau \geq 0$ and let*

$$f_\alpha(x,y) = f(x,y) + \alpha\|x-y\|^2/2$$

with $\alpha \geq -\tau$. Then $QE(K;f)$ and $QE(K;f_\alpha)$ have the same set of solutions.

The equivalence between $QE(K;f)$ and $QE(K;f_\alpha)$ allows deducing some alternative formulation of Theorem 1.2.1 when $\alpha > -\tau$.
First consider the multi-value map $Y_\alpha : \mathbb{R}^n \rightrightarrows \mathbb{R}^n$ given by

$$Y_\alpha(x) = \arg\min\{f_\alpha(x,y) \,:\, y \in K(x)\}$$

Due $f_\alpha(x,\cdot)$ being $(\tau+\alpha)$-convex with $\tau+\alpha > 0$, guarantees that $Y_\alpha(x) = \{y_\alpha(x)\}$ is a singleton for any $x \in \mathbb{R}^n$.

Theorem 1.2.3 ([2], pp.76). *Suppose fix $K = \{x \in \mathbb{R}^n \,:\, x \in K(x)\}$ is nonempty and $f(x,\cdot)$ is τ-convex $\forall\, x \in$ fix K. Given any $\alpha > -\tau$, the following statements are equivalent:*

a) $\bar{x}$ solves $QE(K;f)$,

b) $y_\alpha(\bar{x}) = \bar{x}$.

Theorem 1.2.3 shows that $QE(K;f)$ can be turn into a fixed point problem. The following theorem gives us existence results for $QE(K;f)$:

Theorem 1.2.4 ([2], pp.77). *Suppose K be a lower semi-continuous with nonempty convex values, fix K is closed and there exists a compact convex set X, such that $K(x) \subseteq X$ $\forall x \in \mathbb{R}^n$. If $f(\cdot,y)$ is upper semi-continuous $\forall y \in \mathbb{R}^n$ and $f(x,\cdot)$ is quasi-convex $\forall x \in X$ and upper semi-continuous $\forall x \in \partial_X fix\, K$, then $QE(K;f)$ has at least one solution.*

1.3 General Algorithm

In this section our aim is to generalize a class of double-projection methods for solving problems $QE(K;f)$. The strategy is to reduce at each step the distance from the solution set. We will give conditions on the data to force the convergence of this very general algorithm.
From now on the following assumption is supposed to be satisfied for problem $QE(K;f)$:

Assumption (A)

(a) $f : X \times \Lambda \to \mathbb{R}$ bi-function finite on $X \times \Lambda$ where $\Lambda \subseteq \mathbb{R}^n$ is an open set containing X, $f(x,\cdot)$ convex on Λ $\forall\, x \in X$, continuous on $X \times \Lambda$ and $f(x,x) = 0\;\forall\, x \in X$.

(b) K is continuous on X and $K(x)$ is a nonempty closed convex subset of X $\forall x \in X$.

(c) $x \in K(x)$ $\forall x \in X$.

(d) $S^* = \{x \in S \mid f(x,y) \geq 0,\ \forall y \in T\}$ is nonempty, where $S = \cap_{x\in X} K(x)$ and $T = \cup_{x \in X} K(x)$.

(e) f pseudo-monotone on X with respect to S^*, i.e.,

$$f(y,\bar{x}) \leq 0 \quad \forall \bar{x} \in S^*,\ \forall y \in X$$

Our general algorithm can be expressed as follows:

Algorithm 1: General Algorithm, [26]

Data: $x^0 \in X,\ \mu \in (0,1),\ \gamma \in (0,2)$.

for $k = 0, 1, \dots$ **do**
- Compute $y^k = \arg\min_{y \in K(x^k)}\{f(x^k,y) + 1/2\|y - x^k\|^2\}$;
- **if** $y^k = x^k$ **then**
  - Stop.
- **else**
  - Find d^k such that $\langle d^k, x^k - x^*\rangle \geq \mu\|x^k - y^k\|^2 > 0 \quad \forall x \in S^*$;
  - Compute $x^k(\beta_k) = P_{K(x^k)}(x^k - \beta_k d^k)$ where β_k is such that

$$\|x^k(\beta_k) - x^*\|^2 \leq \|x^k - x^*\|^2 - \gamma(2-\gamma)\mu^2 \frac{\|x^k - y^k\|^4}{\|d^k\|^2} \quad (1.1)$$

  $\forall x^* \in S^*$;
- **end**
- Set $x^{k+1} = x^k(\beta_k)$.

end

Remark 1.4. (1) When the vector $-d^k$ is a descent direction at x^k for the function $\frac{1}{2}\|x - x^*\|^2 \ \forall \ x^* \in S^*$. In particular $d^k \neq 0 \ \ \forall \, k$.

(2) An example of β_k satisfying (1.1) is given by

$$\beta_k = \gamma\mu \frac{\|x^k - y^k\|^2}{\|d^k\|^2}.$$

Indeed $\forall \ x^* \in S^*$, we have $x^* \in K(x^k)$ and consequently, using the definition of orthogonal projection,$P_{K(x^k)}(u) = \arg\min_{y \in K(x^k)} \|y - u\|^2$, and the propriety of d^k, $\langle d^k, x^k - x^* \rangle \geq \mu \|x^k - y^k\|^2$, we obtain

$$\begin{aligned}
\|x^k(\beta_k) - x^*\|^2 &\overset{(C.S.)}{\leq} \|x^k - x^*\|^2 - 2\beta_k \langle d^k, \ x^k - x^* \rangle + \beta_k^2 \|d^k\|^2 \\
&\leq \|x^k - x^*\|^2 - 2\beta_k \mu \|x^k - y^k\|^2 + \beta_k^2 \|d^k\|^2 \\
&= \|x^k - x^*\|^2 - \gamma(2 - \gamma)\mu^2 \frac{\|x^k - y^k\|^4}{\|d^k\|^2}.
\end{aligned}$$

(3) When $f(x, y) = \langle F(x), \ y - x \rangle \quad \forall \, x, y \in X$, step 1 becomes: compute $y^k = P_{K(x^k)}(x^k - F(x^k))$.

1.3.1 Properties

First we give a characterization of y^k computed from x^k at step 1 of the General Algorithm.

Proposition 1.3.1. *For every $y \in K(x^k)$, we have*

$$f(x^k, y) \geq f(x^k, y^k) + \langle x^k - y^k, \ y - y^k \rangle.$$

In particular $f(x^k, y^k) + \|x^k - y^k\|^2 \leq 0$.

Proof. The vector y^k being a solution of a convex minimization problem, the optimality conditions imply that $\exists \, s^k \in \partial f(x^k, y^k)$ such that

$$0 \in s^k + y^k - x^k + \mathcal{N}_{K(x^k)}(y^k)$$

where $\mathcal{N}_{K(x^k)}(y^k) \equiv \{d \in \mathbb{R}^n \ : \ \langle d, \, y - y^k \rangle \leq 0, \quad \forall \, y \in K(x^k)\}$ is the normal cone to $K(x^k)$ at y^k. Hence, by definition of this cone, we obtain that

$$\langle x^k - y^k - s^k, \, y - y^k \rangle \leq 0, \ \ \forall \, y \in K(x^k). \tag{1.2}$$

On the other hand, since $s^k \in \partial f(x^k, y^k)$, we can write

$$f(x^k, y) \geq f(x^k, y^k) + \langle s^k, \, y - y^k \rangle \quad \forall \, y \in K(x^k). \tag{1.3}$$

Combining (1.2) and (1.3) and taking $y = x^k$, we obtain the desired result because $x^k \in K(x^k)$ by assumption (A)(c). $\square$

Now we justify the stopping criterion: $y^k = x^k$.

Proposition 1.3.2. *If $y^k = x^k$, then x^k is a solution of the problem $QE(K; f)$.*

Proof. Since $y^k = x^k$ and $x^k \in K(x^k)$, it follows from Proposition 1.3.2 that

$$f(x^k, y) \geq f(x^k, x^k) + \langle x^k - x^k,\, y - x^k \rangle = 0 \ \ \forall\, y \in K(x^k),$$

, i.e., that x^k is a solution of $QE(K; f)$. □

Next we assume that $x^k \neq y^k \quad \forall\, k$ and we prove that the sequence $\{x^k\}$ generated by the General Algorithm is bounded.

Proposition 1.3.3. *The sequence $\{x^k\}$ is bounded.*

Proof. Since by construction (step 7 of the General Algorithm), $\{\|x^k - x^*\|\}$ is a decreasing sequence, we have

$$\|x^k\| \leq \|x^k - x^*\| + \|x^*\| \leq \|x^0 - x^*\| + \|x^*\| \quad \forall\, k,$$

and thus $\{x^k\}$ is bounded. □

To prove the boundedness of the sequence $\{y^k\}$, we need the next lemma.

Lemma 1.3.4. $\|x^k - y^k\| \leq \|g\| \quad \forall\, g \in \partial f(x^k, x^k)$

Proof. Let $g \in \partial f(x^k, x^k)$, then

$$f(x^k, y^k) \geq f(x^k, x^k) + \langle g,\, y^k - x^k \rangle = \langle g,\, y^k - x^k \rangle.$$

Using progressively Proposition 1.3.1, the previous inequality and the Cauchy-Schwarz inequality, we obtain

$$\|x^k - y^k\|^2 \leq -f(x^k, y^k) \leq -\langle g,\, y^k - x^k \rangle \leq \|g\| \, \|x^k - y^k\|,$$

and thus $\|x^k - y^k\| \leq \|g\|$. □

The sequence $\{x^k\}$ being bounded, let $\bar{x}$ be one of its limit points. Then there exists a subsequence x^{k_j} converging to $\bar{x}$. Thanks to Lemma 1.3.4 we can prove that the corresponding sequence $\{y^{k_j}\}$ is also bounded.

Proposition 1.3.5. *The sequence $\{y^{k_j}\}$ is bounded.*

Proof. By Lemma 1.3.4 it is sufficient to prove that $\exists\, M > 0$ such that

$$\|g\| \leq M \quad \forall\, g \in \partial f(x^{k_j}, x^{k_j}) \text{ and } \forall\, j.$$

Since $\bar{x} \in \Lambda$, $\{x^{k_j}\} \subset \Lambda$, $f(\bar{x}, \cdot)$ is finite on Λ and since the sequence of convex functions $\{f(x^{k_j}, \cdot)\}$ converges point-wise on Λ to the convex function $f(\bar{x}, \cdot)$, it follows form [23] Theorem 24.5 that $\exists\, j_0$ such that

$$\partial f(x^{k_j}, x^{k_j}) \subset \partial f(\bar{x}, \bar{x}) + B \quad \forall\, j \geq j_0$$

where B denotes the close Euclidean unit ball of $\mathbb{R}^n$. Since B and $\partial f(\bar{x}, \bar{x})$ are bounded, $\exists\, M > 0$ such that

$$\|g\| \leq M \quad \forall\, g \in \partial f(x^{k_j}, x^{k_j}) \text{ and } \forall\, j \geq j_0.$$

Hence the sequence $\{y^{k_j}\}$ is bounded. □

Proposition 1.3.6. *Let $\bar{x}$ be a limit point of $\{x^k\}$. Assume that $x^{k_j} \longrightarrow \bar{x}$ and that $\|x^{k_j} - y^{k_j}\| \xrightarrow{j\longrightarrow\infty} 0$. Then $\bar{x}$ is a solution of the problem $QE(K; f)$.*

Proof. By assumption $y^{k_j} = y^{k_j} - x^{k_j} + x^{k_j} \longrightarrow \bar{x}$. Since $y^{k_j} \in K(x^{k_j})\ \ \forall\, j$ and since K is u.s.c. on X, we obtain that $\bar{x} \in K(\bar{x})$.
Now let $\bar{y} \in K(\bar{x})$. We have to prove that $f(\bar{x}, \bar{y}) \geq 0$. Since K is l.s.c. on X, $\exists\, \{\bar{y}^{k_j}\}$ sequence such that

$$\bar{y}^{k_j} \in K(x^{k_j})\ \ \forall\, j \text{ and } \bar{y}^{k_j} \longrightarrow \bar{y}.$$

So, $\forall\, j$, we have, by definition of y^{k_j}, that

$$f(x^{k_j}, y^{k_j}) + \frac{1}{2}\|x^{k_j} - y^{k_j}\|^2 \leq f(x^{k_j}, \bar{y}^{k_j}) + \frac{1}{2}\|x^{k_j} - \bar{y}^{k_j}\|^2.$$

Taking the limit as $j \longrightarrow \infty$ and remembering that f is continuous, we obtain

$$0 = f(\bar{x}, \bar{x}) + \frac{1}{2}\|\bar{x} - \bar{x}\|^2 \leq f(\bar{x}, \bar{y}) + \frac{1}{2}\|\bar{x} - \bar{y}\|^2. \tag{1.4}$$

But this implies that $f(\bar{x}, \bar{y}) \geq 0$. Indeed, the inequality (1.4) means that $\bar{x}$ is a solution of the convex minimization problem

$$\min_{y\in K(\bar{x})} \left[f(\bar{x}, y) + \frac{1}{2}\|\bar{x} - \bar{y}\|^2 \right].$$

Hence $0 \in \partial f(\bar{x}, \bar{x}) + \mathcal{N}_{K(\bar{x})}(\bar{x})$, that is, $\bar{x}$ is a solution of $\min f(\bar{x}, y)$ subject to $y \in K(\bar{x})$. Consequently $f(\bar{x}, \bar{y}) \geq 0$. $\square$

Finally we obtain the convergence of the whole sequence $\{x^k\}$ to a solution of the problem $QE(K; f)$ when the function f is strictly monotone.

Proposition 1.3.7. *If, in addition to the assumption of Proposition* 1.3.6 *the function f is strictly monotone at $\bar{x}$, then the whole sequence $\{x^k\}$ converges to $\bar{x}$ as $k \longrightarrow \infty$. Furthermore $\bar{x}$ is a solution of problem $QE(K; f)$.*

Proof. Let $x^{k_j} \longrightarrow \bar{x}$. By Proposition 1.3.6, $\bar{x}$ is a solution of the problem $QE(K; f)$.

(1) First we prove that $\bar{x} \in S^*$. Let $x^* \in S^*$. Then $x^* \in \cap_{x\in X} K(x)$ and $f(x^*, y) \geq 0\ \ \forall\, y \in K(x)$ and $\forall\, x \in X$. Since $\bar{x} \in K(\bar{x})$, we have $f(x^*, \bar{x}) \geq 0$.
On the other hand $f(\bar{x}, x^*) = 0$. Indeed, by Assumption (A)(e), we have that $f(\bar{x}, x^*) \leq 0$ and, since $\bar{x}$ belongs to the solution set of $QE(K; f)$ and $x^* \in K(\bar{x})$, we have that $f(\bar{x}, x^*) \geq 0$. Consequently $x^* = \bar{x}$. Indeed, if $x^* \neq \bar{x}$, we deduce from the strict monotonicity of f at $\bar{x}$ that

$$f(x^*, \bar{x}) = f(x^*, \bar{x}) + f(\bar{x}, x^*) < 0,$$

which contradicts $f(x^*, \bar{x}) \geq 0$. Hence $\bar{x} = x^* \in S^*$.

(2) Next we prove that $x^k \longrightarrow \bar{x}$. Since $\bar{x} = x^* \in S^*$, it follows from step 7 of the General Algorithm that the sequence $\{\|x^k - \bar{x}\|\}$ is non-increasing and consequently converges to some $a \geq 0$. Since $x^{k_j} \longrightarrow \bar{x}$, we deduce that the whole sequence $\|x^k - \bar{x}\| \longrightarrow 0$, that is $x^k \xrightarrow{k\longrightarrow\infty} \bar{x}$.

□

The convergence of the General Algorithm is obtained under the assumption that $\|x^k - y^k\| \xrightarrow{k \longrightarrow \infty} 0$. Thanks to the inequality (1.1) the sequence $\{\|x^k - x^*\|\}$ is non-increasing and converges to some $a \geq 0$, which implies that

$$\frac{\|x^k - y^k\|^4}{\|d^k\|^2} \xrightarrow{k \longrightarrow \infty} 0.$$

Consequently $\|x^k - y^k\| \longrightarrow 0$ when the sequence $\{\|d^k\|\}$ is unbounded.

1.3.2 Line-search

In this subsection we give an example of direction d^k such that for some $\mu \in (0,1)$

$$\langle d^k, x^k - x^* \rangle \geq \mu \|x^k - y^k\|^2 > 0 \quad \forall x \in S^* \text{ and } \forall\, k.$$

This line-search has the property that when the step-lengths tend to zero, then the sequence $\{d^k\}$ is unbounded and $\|x^k - y^k\| \longrightarrow 0$. More precisely, step 6 of the General Algorithm is replaced by the following line-search procedure:

Linesearch: Let $x^k,\ y^k$ be defined as in the General Algorithm; $\alpha,\ c \in (0,1)$. Find the smallest $m \in \mathbb{N}$ such that

$$\begin{cases} f(z^{k,m}, x^k) - f(z^{k,m}, y^k) \geq c\|x^k - y^k\|^2 \\ z^{k,m} := (1-\alpha^m)x^k + \alpha^m y^k. \end{cases}$$

Set $\alpha_k = \alpha^m$, $z^k = z^{k,m}$ and set $d^k = \frac{g^k}{\alpha_k}$ where $g^k \in \partial f(z^k, x^k)$.

Remark 1.5. When $f(x,y) = \langle F(x), y - x \rangle \ \ \forall x,\ y \in X$, the inequality satisfied by the line-search coincides with $\langle F(x^k - \alpha^m(x^k - y^k)), x^k - y^k \rangle \geq c\|x^k - y^k\|^2$ and the direction d^k becomes equal to $\frac{F(z^k)}{\alpha_k}$.

First we prove that the line-search is finite when $y^k \neq x^k$.

Proposition 1.3.8. *Assume $y^k \neq x^k$. Then the line-search gives α_k and z^k after finitely many iterations.*

Proof. Suppose that the line-search is not finite, then $\forall\, m \in \mathbb{N}$ we have

$$f(z^{k,m}, x^k) - f(z^{k,m}, y^k) < c\|x^k - y^k\|^2.$$

Since $f(\cdot, x)$ is continuous $\forall\, x \in \Lambda$, and $z^{k,m} \xrightarrow{m \longrightarrow \infty} x^k$, we obtain

$$-f(x^k, y^k) = f(x^k, x^k) - f(x^k, y^k) \leq c\|x^k - y^k\|^2.$$

On the other hand, by Proposition 1.3.1, we have that

$$f(x^k, y^k) + \|x^k - y^k\|^2 \leq 0.$$

Combining these two inequalities yields

$$\|x^k - y^k\|^2 \leq c\|x^k - y^k\|^2.$$

Since $c \in (0,1)$, we deduce that $y^k = x^k$, which contradicts the assumptions $y^k \neq x^k$. □

Now our aim is to prove that the direction d^k obtained from the line-search satisfies the property: $\langle d^k, x^k - x^* \rangle \geq \mu \|x^k - y^k\|^2 \;\; \forall\, x^* \in S^*$ and some $\mu \in (0, 1)$.

Proposition 1.3.9. *For the line search, $\forall\, g \in \partial f(z^k, x^k)$ and $x^* \in S^*$, we have*

$$\langle \frac{g^k}{\alpha_k}, x^k - x^* \rangle \geq f(z^k, x^k) - f(z^k, y^k).$$

Proof. Let $g^k \in \partial f(z^k, x^k)$ and $x^* \in S^*$. Then

$$f(z^k, x^*) \geq f(z^k, x^k) + \langle g^k, x^* - x^k \rangle.$$

Since f is pseudo-monotone on X with respect to S^*, we have $f(z^k, x^*) \leq 0$, so

$$\langle g^k, x^k - x^* \rangle \geq f(z^k, x^k). \tag{1.5}$$

Since $f(z^k, \cdot)$ is convex, we can write, using the definition of z^k,

$$f(z^k, z^k) \leq (1 - \alpha_k) f(z^k, x^k) + \alpha_k f(z^k, x^k),$$

that is

$$f(z^k, x^k) \geq \alpha_k [f(z^k, x^k) - f(z^k, y^k)]. \tag{1.6}$$

Combing (1.5) and (1.6) yields the announced result. $\square$

It follows immediately from Proposition 1.3.9 that if Line-search is used, then the required property on $d^k = \frac{g^k}{\alpha_k}$:

$$\langle d^k, x^k - x^* \rangle \geq \mu \|x^k - y^k\|^2$$

is satisfied for $\mu = c$. In particular, when Line-search is used, that is when $y^k \neq x^k$, the direction $d^k \neq 0$ whatever $g^k \in \partial f(z^k, x^k)$.
From now on we denote by General Modified Algorithm the General Algorithm with step 6 replaced by Line-search.

Algorithm 2: General Modified Algorithm, Alg.1 of [26]

Data: $x^0 \in X,\ c \in (0,1),\ \alpha \in (0,1),\ \gamma \in (0,2)$.

for $k = 0, 1, \ldots$ **do**
  Compute $y^k = \arg\min_{y \in K(x^k)} \{f(x^k, y) + 1/2\|y - x^k\|^2\}$;
  if $y^k = x^k$ **then**
    Stop.
  else
    Find the smallest $m \in \mathbb{N}$ such that

$$\begin{cases} f(z^{k,m}, x^k) - f(z^{k,m}, y^k) \geq c\|x^k - y^k\|^2 \\ z^{k,m} := (1 - \alpha^m) x^k + \alpha^m y^k. \end{cases}$$

    Set $\alpha_k = \alpha^m$, $z^k = z^{k,m}$;
    Compute $g^k \in \partial f(z^k, x^k)$ and $x^{k+1} = P_{K(x^k)}(x^k - \beta_k d^k)$
    where $d^k = \frac{g^k}{\alpha_k}$ and $\beta_k = \gamma c \dfrac{\|x^k - y^k\|^2}{\|d^k\|^2}$.
  end
end

Now for the General Modified Algorithm we have the following boundedness properties:

Proposition 1.3.10. *Let $\bar{x}$ be a limit point $\{x^k\}$ and assume that $x^{k_j} \longrightarrow \bar{x}$. Then the sequences $\{y^{k_j}\}$, $\{z^{k_j}\}$ and $\{g^{k_j}\}$ are bounded.*

Proof. Since the sequence $\{x^{k_j}\}$ and $\{y^{k_j}\}$ are bounded, see Proposition 1.3.3 and 1.3.5, it follows that the sequence $\{z^{k_j}\}$ is also bounded because z^{k_j} belongs to the segment $[x^{k_j}; y^{k_j}]$ $\forall j$. So a subsequence of $\{z^{k_j}\}$, again doted $\{z^{k_j}\}$, converges to some $\bar{z} \in X$. Since $\bar{x} \in X \subseteq \Lambda$, $\{x^{k_j}\} \subseteq X \subseteq \Lambda$, $x^{k_j} \longrightarrow \bar{x}$, and the sequence of convex functions $\{f(z^{k_j}, \cdot)\}$ converges pointwise to the convex function $f(\bar{z}, \cdot)$. It follows from Theorem 24.5 of [23] that $\exists\, j_0$ such that $\partial f(z^{k_j}, x^{k_j}) \subseteq \partial f(\bar{z}, \bar{x}) + B \quad \forall j \geq j_0$, where B denotes the closed Euclidean unit ball of $\mathbb{R}^n$. Since B and $\partial f(\bar{z}, \bar{x})$ are bounded, the sequence $\{g^{k_j}\}$ is also bounded. □

In order to apply Proposition 1.3.6, we need to prove the next result.

Proposition 1.3.11. *Let $x^{k_j} \longrightarrow \bar{x}$. Then $\|x^{k_j} - y^{k_j}\| \xrightarrow{j \longrightarrow \infty} 0$.*

Proof. We examine two cases:

1. $\underline{\inf_j \alpha_{k_j} > 0}$: the sequence $\{d^{k_j}\}$ is bounded because the sequence $\{x^{k_j}\}$, $\{y^{k_j}\}$ and $\{g^{k_j}\}$ are bounded, for Proposition 1.3.10, and $d^{k_j} = \frac{g^{k_j}}{\alpha^{k_j}}$. Since, from (1.1), $\dfrac{\|x^k - y^k\|^4}{\|d^k\|^2} \longrightarrow 0$, we deduce that $\|x^{k_j} - y^{k_j}\| \longrightarrow 0$.

2. $\underline{\inf_j \alpha_{k_j} = 0}$: then $\alpha_{k_j} \longrightarrow 0$ for a subsequence. But this implies that $\alpha_{k_j} < 1$ for j large enough and that the line-search conditions are not satisfied for $\frac{\alpha_{k_j}}{\alpha}$. Let us denote
$$\bar{z}^{k_j} = \left(1 - \frac{\alpha_{k_j}}{\alpha}\right) x^{k_j} + \frac{\alpha_{k_j}}{\alpha} y^{k_j}.$$

It is immediate that $\bar{z}^{k_j} \longrightarrow \bar{x}$. Now if the line-search is used we have
$$f(\bar{z}^{k_j}, x^{k_j}) - f(\bar{z}^{k_j}, y^{k_j}) < c\|x^{k_j} - y^{k_j}\|^2.$$

By definition of y^{k_j} we also have
$$\|x^{k_j} - y^{k_j}\|^2 \leq -f(\bar{x}^{k_j}, y^{k_j}).$$

Let $\bar{y}$ be a limit point of $\{y^{k_j}\}$. Then combining the two inequalities and taking the limit as $j \longrightarrow \infty$, we obtain
$$f(\bar{x}, \bar{x}) - f(\bar{x}, \bar{y}) \leq -cf(\bar{x}, \bar{y}),$$

which implies that $f(\bar{x}, \bar{y}) \geq 0$. So $-f(\bar{x}^{k_j}, y^{k_j}) \longrightarrow -f(\bar{x}, \bar{y}) \leq 0$ and $\|x^{k_j} - y^{k_j}\| \longrightarrow 0$. □

Finally using successively Proposition 1.3.6, 1.3.7 and 1.3.11, we obtain the following convergence result for General Algorithm Modified.

Proposition 1.3.12. *Any limit point of the sequence* $\{x^k\}$ *generated by General Algorithm Modified is a solution of the problem* $QE(K; f)$*. If* f *is stricly monotone at a limit point* $\bar{x}$ *of* $\{x^k\}$*, then* $x^k \xrightarrow{k\to\infty} \bar{x}$*.*

Chapter 2

Hybrid Extragradient Methods

While there exists many different methods for solving $VI(K;F)$ (like, e.g., Solodov-Svaiter method [25, 9]), the number of algorithms for handling $QVI(K;F)$ is quite small. In this chapter, our goal is to extend (following the basic idea of the General Modified Algorithm) a well-known class of double-projection methods for solving problem $VI(K;F)$ to the case of solving problem $QVI(K;F)$.

2.1 Generalized Solodov Method

2.1.1 $VI(K;F)$ case

Let us consider for a moment the case of a variational inequality with a fixed constraint set $K(x) = K \ \ \forall x \in X$. It can be shown that this problem can be reformulated as a fixed-point equation:

$$x - P_K(x - \lambda F(x)) = 0 \tag{2.1}$$

where P_K denotes the orthogonal projection from $\mathbb{R}^n$ onto K and $\lambda > 0$ is a constant. The corresponding fixed point algorithm: $x^{k+1} = P_K(x^k - \lambda F(x^k))$ is convergent to a solution of problem $VI(K;F)$ under a strong assumption: F is Lipschitz and strongly monotone. To avoid that, the following modified fixed point equation has been introduced:

$$x - P_K(x - \lambda F(\bar{x})) = 0 \quad \text{where} \quad \bar{x} = P_K(x - \lambda F(x)). \tag{2.2}$$

When F is Lipschitz continuous, it can be proven (see [13, 29]) that if x satisfies (2.2), than x satisfies (2.1), thus is a solution of problem $VI(K;F)$, provided that the Lipschitz constant L is such that $\lambda < \frac{1}{L}$. The equation of (2.2) gives rise to the classical extragradient method [13] and its variants [11, 12]: given $x^k \in K$, x^{k+1} is obtained after two projections as follows:

$$\begin{cases} y^k = P_K(x^k - \lambda F(x^k)) \\ x^{k+1} = P_K(x^k - \lambda F(y^k)). \end{cases}$$

This method generates sequences converging to a solution of problem $VI(K;F)$ under the assumption that F is pseudo-monotone and Lipschitz continuous with a condition on the Lipschitz constant.
A well-known strategy [27, 28] to avoid the use of the Lipschitz constant is first to define $y^k = P_K(x^k - \lambda F(x^k))$ and then to find the direction d^k such that the inequality

$$\langle d^k,\, x^k - x^* \rangle \geq \mu \|x^k - y^k\|^2 \quad \text{with } \mu > 0 \tag{2.3}$$

holds for any solution x^* of problem $VI(K;F)$. When $y^k \neq x^k$, the direction $-d^k$ is a descent direction at x^k for the distance function to K^*, the solution set of $VI(K;F)$: $\frac{1}{2}\|x - x^*\|^2$ with $x^* \in K^*$. For the classical extragradient method an example of such a direction is given by $d^k = \frac{F(z^k)}{\alpha_k}$ where $z^k = (1-\alpha_k)x^k + \alpha_k y^k$ and $\alpha_k = \alpha^{m_k}$ with m_k is the smallest $m \in \mathbb{N}$ satisfying the inequality

$$\langle F(x^k - \alpha_k(x^k - y^k)),\, x^k - y^k \rangle \geq c\|x^k - y^k\|^2 \tag{2.4}$$

and $\alpha,\, c \in (0,1)$. In fact, this vector z^k gives rise to the hyperplane

$$H^k = \{x \in \mathbb{R}^n |\, \langle F(z^k),\, x - z^k \rangle = 0\} \tag{2.5}$$

which separates x^k from the solution set K^* of problem $VI(K;F)$. The direction d^k satisfies (2.3) with $\mu = c$ and the next iterate x^{k+1} is given by

$$x^{k+1} = P_K(x^k - \beta_k d^k)$$

where $\beta_k > 0$ is chosen such that $\|x^{k+1} - x^*\|^2 < \|x^k - x^*\|^2 \ \ \forall\, x^* \in K^*$ (see [11, 27] for more details). For example, the step-length β_k can be chosen in such a way that $x^k - \beta_k d^k$ be the orthogonal projection of x^k onto H^k. It is easy to see that this step is given by

$$\beta_k = \alpha_k \frac{\langle F(z^k),\, x^k - z^k \rangle}{\|F(z^k)\|^2}.$$

2.1.2 $QVI(K;F)$ case

In this subsection we will reformulate the General Modified Algorithm for $QE(K;f)$ in terms to solve problems $QVI(K;F)$.
In the General Algorithm the next iterate was defined as $x^{k+1} = x^k(\beta_k)$ where $\forall\, \beta > 0,\ x^k(\beta) = P_{K(x^k)}(x^k - \beta d^k)$. The direction was equal to $d^k = \frac{g^k}{\alpha_k}$ with $g^k \in \partial f(z^k, x^k)$ and α_k obtained by using the Line-search. Furthermore the step β_k was chosen such that the following inequality holds:

$$\|x^k(\beta_k) - x^*\|^2 \leq \|x^k - x^*\|^2 - \gamma(2-\gamma)c^2 \frac{\|x^k - y^k\|^4}{\|d^k\|^2}. \tag{2.6}$$

An example of such a step is $\beta_k = \gamma c \dfrac{\|x^k - y^k\|^2}{\|d^k\|^2}$. In this subsection we show that it is possible to choose other steps β_k while keeping true the inequality (2.6). These steps will give rise to better decreases on the distance between the iterates and the set S^*.

By definition of $g^k \in \partial f(z^k, x^k)$, we have

$$\langle g^k,\, x^k - x^* \rangle \geq f(z^k, x^k) - f(z^k, x^*)$$

where x^* is any element in S^*. Since f is pseudo-monotone on X with respect to S^*, we obtain that $f(z^k, x^*) \leq 0$ and thus that

$$\langle d^k,\, x^k - x^* \rangle = \langle \frac{g^k}{\alpha_k},\, x^k - x^* \rangle \geq \frac{f(z^k, x^k)}{\alpha_k}. \tag{2.7}$$

Using this inequality $\|x^k(\beta_k) - x^*\|^2 \leq \|x^k - x^*\|^2 - 2\beta_k \langle d^k,\, x^k - x^* \rangle + \beta_k^2 \|d^k\|^2$ and taking

$$\beta_k = \gamma \frac{f(z^k, x^k)}{\|g^k\|^2} \alpha_k, \tag{2.8}$$

we deduce that

$$\begin{aligned} \|x^{k+1} - x^*\|^2 &\leq \|x^k - x^*\|^2 - 2\frac{\beta_k}{\alpha_k} f(z^k, x^k) + \frac{\beta_k^2}{\alpha_k^2} \|g^k\|^2 \\ &\leq \|x^k - x^*\|^2 - \gamma(2-\gamma) \frac{f(z^k, x^k)^2}{\|g^k\|^2}. \end{aligned}$$

Now from the convexity of $f(z^k, \cdot)$ and the Line-search, we obtain that

$$f(z^k, x^k) \geq \alpha_k (f(z^k, x^k) - f(z^k, y^k)) \geq \alpha_k c \|x^k - y^k\|^2. \tag{2.9}$$

So if we use the new step β_k given in (2.7), we can conclude that

$$\|x^{k+1} - x^*\|^2 \leq \|x^k - x^*\|^2 - \gamma(2-\gamma)\alpha_k^2 c^2 \frac{\|x^k - y^k\|^4}{\|g^k\|^2},$$

and thus that inequality (2.6) holds.
Replacing β_k by its new value in step 7 of the General Modified Algorithm, we obtain

$$x^{k+1} = P_{K(x^k)}\left[x^k - \gamma \frac{f(z^k, x^k)}{\|g^k\|^2} \alpha_k d^k\right] = P_{K(x^k)}[x^k - \gamma \sigma_k g^k]$$

where $\sigma_k = \dfrac{f(z^k, x^k)}{\|g^k\|^2}$.
Consider the hyperplane H^k defined by

$$H^k = \{x \in \mathbb{R}^n | \langle g^k,\, x^k - x \rangle = f(z^k, x^k)\}$$

Now this hyperplane H^k separates x^k from S^*. Indeed, from (2.9), it follows that $f(z^k, x^k) > 0 = \langle g^k,\, x^k - x^k \rangle$ and from (2.7), that $\langle g^k,\, x^k - x^* \rangle \geq f(z^k, x^k) \quad \forall x^* \in S^*$. Furthermore g^k is the normal vector to H^k and since $x^k - \sigma_k g^k \in H^k$, we can say that $x^k - \sigma_k g^k$ is the orthogonal projection of x^k onto H^k. Since the set $S^* \subseteq K^*$, where K^* is the solution set of the problem $QVI(K; F)$, is contained in $K(x^k) \cap H_+^k$ where

$$H_+^k = \{x \in \mathbb{R}^n | \langle g^k,\, x^k - x \rangle \geq f(z^k, x^k)\},$$

a variant of the Generalized Modified Algorithm consist in replacing in step 7 the iterate $x^{k+1} = P_{K(x^k)}(x^k - \gamma\sigma_k g^k)$ by $x^{k+1} = P_{K(x^k)\cap H_+^k}(x^k - \gamma\sigma_k g^k)$. Using the non-expansiveness of $P_{K(x^k)\cap H_+^k}$ instead of the one of $P_{K(x^k)}$, we immediately obtain that the inequality (2.6) holds. So the convergence of the sequence $\{x^k\}$ is preserved for this variant. So we obtain the following change:

step 7 Compute $g^k \in \partial f(z^k, x^k)$ and $x^{k+1} = P_{K(x^k)\cap H_+^k}(x^k - \gamma\sigma_k g^k)$ where $\sigma_k = \dfrac{f(z^k, x^k)}{\|g^k\|^2}$ and $H_+^k = \{x \in \mathbb{R}^n | \langle g^k,\ x^k - x\rangle \geq f(z^k, x^k)\}$

When $\gamma = 1$ we have that $x^k - \gamma\sigma_k g^k = P_{H_+^k}(x^k)$ and we can use the proof of Lemma 2.2 in [25] to show that

$$x^{k+1} = P_{K(x^k)\cap H_+^k}(x^k)$$

When $\gamma = 1$, it is also possible to give a geometric interpretation of step 7 in the Generalized Modified Algorithm. In that purpose we recall the following property of the orthogonal projection onto a convex set.

Proposition 2.1.1 ([27]). *Let $\emptyset \neq C \subseteq \mathbb{R}^n$ be a closed convex set. Then*

$$\|P_C(x) - z\|^2 \leq \|x - z\|^2 - \|P_C(x) - x\|^2 \quad \forall\, x \in \mathbb{R}^n \text{ and } z \in C.$$

Using Lemma 2.1.1 with $C = K(x^k)$, $x = x^k - \beta d^k$ and $z = x^*$, we can write

$$\|x^k(\beta) - x^*\|^2 \leq \|x^k - \beta d^k - x^*\|^2 - \|x^k(\beta) - x^k + \beta d^k\|$$

Developing the first term of the right-hand side of this inequality and using successively $\|x^k(\beta_k) - x^*\|^2 \leq \|x^k - x^*\|^2 - 2\beta_k\langle d^k,\ x^k - x^*\rangle + \beta_k^2\|d^k\|^2$, (2.7) and (2.8), we obtain

$$\|x^k(\beta) - x^*\|^2 \leq \|x^k - x^*\|^2 + \bar{\varphi}_k(\beta) \tag{2.10}$$

where

$$\bar{\varphi}_k(\beta) = \varphi_k(\beta) - \|x^k(\beta) - x^k + \frac{\beta}{\alpha_k} g^k\|^2$$

with

$$\varphi_k(\beta) = -2\frac{\beta}{\alpha_k} f(z^k, x^k) + \frac{\beta^2}{\alpha_k^2}\|g^k\|^2.$$

Since $\bar{\varphi}_k(\beta) \leq \varphi_k(\beta)$, we have, in particular, that

$$\|x^k(\beta) - x^*\|^2 \leq \|x^k - x^*\|^2 + \varphi_k(\beta) \tag{2.11}$$

It easy to check that $\beta_1 = \frac{f(z^k, x^k)}{\|g^k\|^2}\alpha_k = \sigma_k\alpha_k$ minimizes the right-hand side of (2.11). Since $x^k(\beta_1) = P_{K(x^k)}(x^k - \sigma_k\alpha_k d^k) = P_{K(x^k)}(x^k - \sigma_k g^k)$, it follows that the new iterate x^{k+1} in Generalized Modified Algorithm is given by $x^{k+1} = x^k(\beta_1)$. Now if we minimize the right-hand side of (2.10), it can be shown exactly as in [27] that the function $\bar{\varphi}_k(\beta)$ is convex and admits a minimum for a step-length $\beta_2 \geq \beta_1$. Computing an explicit value for β_2 seems difficult but it is possible, using a proof similar to the one of Lemma 3.2 in [27], to show that

$$x^k(\beta_2) = P_{K(x^k)\cap H_+^k}(x^k - \beta_1 d^k) = P_{K(x^k)\cap H_+^k}(x^k - \sigma_k g^k).$$

Hence the new iterate x^{k+1} in Generalized Modified Algorithm is given by $x^{k+1} = x^k(\beta_2)$.

Now let's focus on problem $QVI(K;F)$, that is $f(x,y) = \langle F(x),\, y - x\rangle$ $\forall\, x, y \in X$.
Remember that a convex function g is differentiable at $x \Leftrightarrow \partial g(x) = \{\nabla g(x)\}$, since $f(x,y) = \langle F(x), y - x\rangle$ $\forall\, x, y \in X$ is linear and $f(x,\cdot)$ is convex $\forall\, x \in X$ for Assumption (A), then f is differentiable and $\partial_y f = \nabla_y f$. then we deduce that:

- $g^k \in \partial_y f(z^k, x^k) = \nabla_y f(z^k, x^k) = F(z^k)$
- $\sigma_k = \frac{f(z^k,x^k)}{\|g^k\|^2} = \frac{\langle F(z^k),\, x^k - z^k\rangle}{\|F(z^k)\|^2}$
- $f(z^k, x^k) - f(z^k, y^k) = \langle F(z^k), x^k - z^k\rangle - \langle F(z^k), y^k - z^k\rangle = \langle F(z^k), x^k - y^k\rangle$
- $\langle g^k, x^k - x\rangle \geq f(z^k, x^k) \Leftrightarrow \langle F(z^k), x^k - x\rangle \geq \langle F(z^k), x^k - z^k\rangle$
  $\Leftrightarrow \langle F(z^k), x - z^k\rangle \leq 0$. Then the set H_+^k becomes
  $$H_+^k = \{x \in \mathbb{R}^n |\, \langle F(z^k),\, x - z^k\rangle \leq 0\}.$$
  Notice that the hyperplane H^k coincides with the one defined in (2.5)

In conclusion with these changes the General Modified Algorithm becomes the Generalized Solodov, in fact when $K(x^k) = K \;\; \forall\, x \in X$ and $\gamma = 1$, we find again the projection method introduced by Solodov and algorithm for solving variational inequality problems [25].

Algorithm 3: Generalized Solodov, Alg.1b of [26]

Data: $x^0 \in X,\ c \in (0,1),\ \alpha \in (0,1)$.

for $k = 0, 1, \ldots$ **do**

  Compute
  $$\begin{aligned} y^k &= \arg\min_{y \in K(x^k)} \{\langle F(x^k),\, y - x^k\rangle + 1/2\|y - x^k\|^2\} \\ &= P_{K(x^k)}(x^k - F(x^k)) \end{aligned}$$

  if $y^k = x^k$ **then**

    Stop.

  else

    Find the smallest $m \in \mathbb{N}$ such that
    $$\begin{cases} \langle F(z^{k,m}),\, x^k - y^k\rangle \geq c\|x^k - y^k\|^2 \\ z^{k,m} := (1 - \alpha^m)x^k + \alpha^m y^k \end{cases}$$

    Set $\alpha_k = \alpha^m$, $z^k = z^{k,m}$;

    Compute $x^{k+1} = P_{K(x^k)\cap H_+^k}(x^k - \gamma\frac{\langle F(z^k),\, x^k - z^k\rangle}{\|F(z^k)\|^2}F(z^k))$ where $H_+^k = \{x \in \mathbb{R}^n |\, \langle F(z^k),\, x - z^k\rangle \leq 0\}$.

  end

end

2.2 Nguyen-Strodiot Method

In this subsection we present an efficient method for solving a quasi-variational inequality problem. The strategy is to combine the well-known search directions in the correction step from literature with the direction defined by the current iterate and the trial point obtained in the prediction step. This new combined search direction allows us to improve the convergence of the sequence of iterates to the solution of the $QVI(K;F)$ but under a slightly stronger assumption, namely the co-coercivity of the problem operator. The new algorithm is devised to solve problems where the projections onto the moving feasible set are not easy to obtain.

2.2.1 Basic Idea

Let $x^k \in X$; two procedures can be used to get the next iterate x^{k+1}, depending on the numerical difficulty to compute the projection onto the moving feasible set $K(x^k)$. When the projection onto $K(x^k)$ is easy to compute, the prediction step can be defined by

$$\bar{x}^k = P_{K(x^k)}(x^k - \beta_k F(x^k)) \tag{2.12}$$

where $\beta_k = \gamma l^{m_k}$, $\gamma \in (0,1)$, $l \in (0,1)$ and m_k is the smallest nonnegative integer m such that

$$\beta_k \langle F(x^k) - F(\bar{x}^k),\, x^k - \bar{x}^k \rangle \leq \|x^k - \bar{x}^k\|^2$$

with $c \in (0,1)$. In this procedure, a new projection onto $K(x^k)$ must be computed each time the parameter m_k is updated.
When the projection on $K(x^k)$ is numerically more expensive, it is preferable to use only one projection on $K(x^k)$ per line-search. So, in that situation, we first calculate

$$z^k = P_{K(x^k)}(x^k - F(x^k))$$

and after we compute

$$y^k = (1 - \beta_k)x^k + \beta_k z^k \tag{2.13}$$

where $\beta_k = l^{m_k}$, $l \in (0,1)$ and m_k is the smallest nonnegative integer m such as

$$\langle F(x^k) - F(y^k),\, x^k - z^k \rangle \leq c\|x^k - z^k\|^2$$

where $c \in (0,1)$.
Once $\bar{x}^k$ or y^k is obtained, a correction step is done by calculating

$$x^{k+1} = P_{K(x^k)}(x^k - \alpha_k d^k)$$

where d^k is a search direction and α_k is a step-length.
When $\bar{x}^k$ is used, Zhang et al. [30] propose to take, with $\sigma \in (0,2)$,

$$d^k = x^k - \bar{x}^k + \beta_k F(\bar{x}^k) := d_k^{Z1} \quad \text{and} \quad \alpha_k = \sigma(1-c)\frac{\|x^k - \bar{x}^k\|^2}{\|d^k\|^2}$$

for the search direction and the step-length along this direction respectively. On the other hand, when it is y^k that is used, it is suggested to take, with $\sigma \in (0, 2)$,

$$d^k = x^k - z^k + \frac{1}{\beta_k} F(y^k) := d_k^Z \quad \text{and} \quad \alpha_k = \sigma(1-c)\frac{\|x^k - z^k\|^2}{\|d^k\|^2}.$$

With this choice, it was proved [[30], Lemma 5.2, inequality (29)] that

$$\langle d_k^Z,\, x^k - x^* \rangle \geq \|x^k - z^k\|^2 - \langle F(x^k) - F(y^k),\, x^k - z^k \rangle. \tag{2.14}$$

On the other hand, Han et al. [10] recently revisited the prediction set in the case when $\bar{x}^k$ is used, and proposed, in the correction step, to combine the direction $d^k := d_k^{Z1} - \beta_k F(x^k)$ with the direction $x^k - \bar{x}^k$ as follows:

$$\bar{d}^k = \rho d^k + (1-\rho)(x^k - \bar{x}^k)$$

where $\bar{x}^k$ is given by (2.12) and $\rho \in (0, 1)$. With this strategy, the numerical behavior of Han et al.'s algorithm [10] is better than the one of Zhang et al. [30]. However, its convergence is obtained under the assumption that F is co-coercive, while Zhang and al.'s algorithm requires the monotonicity of F to ensure the convergence.

Our aim in this section is to modify Han and al.'s [10] algorithm as follows: instead of computing the prediction step $\bar{x}^k$ given by (2.12), [15] proposes to use the prediction step y^k given by (2.13); doing so the projection step z^k is computed only once. Furthermore, to obtain a very general algorithm, [15] considers a class of search directions which will be used in the correction step.

2.2.2 Algorithm Description and Convergence Analysis

In order to prove the convergence of the resulting algorithm, we use the following assumption and result:

- Assumption (A) with $f(x, y) = \langle F(x),\, y - x \rangle$
- F is $\mu - co - coercive$ on T

Definition 2.1. Let us say that F is *co-coercive with modulus* $\mu > 0$ (or $\mu -$ *co-coercive*) on X if, $\forall\, x, y \in X$

$$\langle F(y) - F(x),\, y - x \rangle \geq \mu \|F(y) - F(x)\|^2.$$

Lemma 2.2.1 ([10], Lemma 4.2)**.** *Let $x^* \in S^*$ and suppose that F is co-coercive on X with modulus $\mu > \frac{1}{4}$. If $z^k = P_{K(x^k)}(x^k - F(x^k))$, then*

$$\langle x^k - z^k,\, x^k - x^* \rangle \geq \left(1 - \frac{1}{4\mu}\right)\|x^k - z^k\|^2 \quad \forall\, x^k \in X.$$

Algorithm 4: Nguyen-Strodiot prototype, Algorithm QVI of [15]

Data: $x^0 \in X,\ l \in (0,1),\ c \in (0,1),\ \mu > \frac{1}{4},\ \rho \geq 0,\ \gamma \in (0,1)$.

for $k = 0, 1, \dots$ **do**

  Compute

$$\begin{aligned} z^k &= \arg \min_{z \in K(x^k)} \{\langle F(x^k),\, z - x^k \rangle + 1/2 \|z - x^k\|^2\} \\ &= P_{K(x^k)}(x^k - F(x^k)); \end{aligned}$$

  if $z^k = x^k$ **then**

    Stop.

  else

    Find m_k the smallest $m \in \mathbb{N}$ such that

$$\langle F(x^k) - F((1 - \gamma l^m)x^k + \gamma l^m z^k),\, x^k - z^k \rangle \leq c\|x^k - z^k\|^2 \tag{2.15}$$

    Set $y^k := (1 - \beta_k)x^k + \beta_k z^k$ where $\beta_k = \gamma\, l^{m_k}$;

    Choose a direction d^k satisfying $\forall\, x^* \in S^*$ the inequality

$$\langle \beta_k d^k,\, x^k - x^* \rangle \geq \|x^k - y^k\|^2 - \beta_k \langle F(x^k) - F(y^k),\, x^k - y^k \rangle; \tag{2.16}$$

    Compute

$$\begin{aligned} \bar{d}^k &= \frac{\rho}{1+\rho}(x^k - y^k) + \frac{1}{1+\rho} d^k \\ x^{k+1} &= P_{K(x^k)}(x^k - \alpha_k \beta_k \bar{d}^k) \end{aligned}$$

    where $\alpha_k > 0$.

  end

end

Before proving the convergence of Nguyen-Strodiot prototype Algorithm, it remains to define the step-size α_k and to give some examples of directions d^k satisfying (2.6). It is the aim of the next propositions.

Proposition 2.2.2. *Let $x^* \in S^*$ and assume that $y^k \neq x^k$ at iteration k, let also be $\rho_1 = \frac{1}{1+\rho_1}$. Then $-\bar{d}^k$ is a descent direction at x^k for the merit function $\frac{1}{2}\|x - x^*\|^2$ when*

$$1 - \frac{\rho_1 \rho}{4\mu} - \rho_1 c > 0. \tag{2.17}$$

In particular this inequality is satisfied when $c < 1 + \rho$ and $\mu > \frac{\rho\rho_1}{4(1-\rho_1 c)}$.

Proof. Using successively the definition of $\bar{d}^k$, Lemma 2.2.1, (2.15), (2.16), we obtain

$$\begin{aligned}\langle \beta_k \bar{d}^k, x^k - x^* \rangle &= \beta_k \langle \rho_1 \rho(x^k - y^k) + \rho_1 d^k, x^k - x^* \rangle \\ &= \rho_1 \rho \beta_k \langle x^k - y^k, x^k - x^* \rangle + \rho_1 \beta_k \langle d^k, x^k - x^* \rangle \\ &\geq \left(1 - \frac{\rho_1 \rho}{4\mu}\right) \|x^k - y^k\|^2 - \rho_1 \beta_k \langle F(x^k) - F(y^k), x^k - y^k \rangle \end{aligned} \tag{2.18}$$

$$\geq \left(1 - \frac{\rho_1 \rho}{4\mu} - \rho_1 c\right) \|x^k - y^k\|^2 > 0. \tag{2.19}$$

But this implies that $\bar{d}^k$ is a descent direction at x^k for the merit function $\frac{1}{2}\|x - x^*\|^2$ when (2.17) is satisfied. □

Now we can determine the value of α_k in step 6 of Nguyen-Strodiot prototype Algorithm. Indeed, since $\langle \beta_k \bar{d}^k, x^k - x^k \rangle = 0$, it follows from (2.18) that for

$$\alpha_k = \frac{\left(1 - \frac{\rho_1 \rho}{4\mu}\right) \|x^k - y^k\|^2 - \rho_1 \beta_k \langle F(x^k) - F(y^k), x^k - y^k \rangle}{\beta_k^2 \|\bar{d}^k\|^2} \tag{2.20}$$

the hyperplane $H^k = \{x \in \mathbb{R}^n | \langle \bar{d}^k, x^k - x \rangle = \alpha_k \beta_k \|\bar{d}^k\|^2\}$ strictly separates x^k from the set S^*. Using the definition of α_k and observing that $\bar{d}^k$ is orthogonal to the hyperplane H^k, we obtain that $x^k - \alpha_k \beta_k \bar{d}^k = P_{H^k}(x^k)$. So x^{k+1} is computed thanks to two successive projections: first x^k is projected onto H^k and afterwards, the resulting vector is projected onto $K(x^k)$.

Now we can give three examples of directions d^k satisfying (2.16). (Note that in the next Proposition 2.2.3 and Proposition 2.2.4 the mapping F needs only to be pseudo-monotone).

Proposition 2.2.3. *If d_k^Z is a direction satisfying* (2.14) *at iteration k, then the direction $d^k = \beta_k d_k^Z$ satisfies* (2.16)*. In particular, the direction $d_k^1 = x^k - y^k + F(y^k)$ satisfies* (2.16)

Proof. Since $x^k - y^k = \beta_k(x^k - z^k)$, we have successively

$$\begin{aligned}\langle \beta_k d^k, x^k - x^* \rangle &= \beta_k^2 \langle d_k^Z, x^k - x^* \rangle \\ &\geq \beta_k^2 \|x^k - z^k\|^2 - \beta_k^2 \langle F(x^k) - F(y^k), x^k - z^k \rangle \\ &= \|x^k - y^k\|^2 - \beta_k \langle F(x^k) - F(y^k), x^k - y^k \rangle.\end{aligned}$$

So the direction d^k satisfies (2.16). On the other hand, it was proven in [30] that the direction $d_k^Z := x^k - z^k + \frac{1}{\beta_k} F(y^k)$ satisfies (2.14). Consequently, the direction d_k^1, being equal to $\beta_k d_k^Z$, satisfies (2.16). □

Proposition 2.2.4. *At iteration k, the two directions*

$$d_k^2 := x^k - y^k + F(x^k) + F(y^k)$$

$$d_k^3 := x^k - y^k - \beta_k \left(F(x^k) - \frac{F(y^k)}{\beta_k}\right)$$

introduced in Noor et al.[16] and [17], respectively, satisfy (2.16).

Proof. First we observe that

$$d_k^2 = d_k^1 + F(x^k) \quad \text{and} \quad d_k^3 = d_k^1 - \beta_k F(x^k).$$

Since the direction d_k^1 satisfies (2.16), it suffices to see that $\langle F(x^k), x^k - x^*\rangle \geq 0$ (because F is pseudo-monotone) to obtain the direction d_k^2 satisfies (2.16).

On the other hand, since $x^k - y^k = \beta_k(x^k - z^k)$, $z^k = P_{K(x^k)}(x^k - F(x^k))$, $x^* \in K(x^k)$ and F is pseudo-monotone, we have

$$\begin{aligned}
\langle d_k^3, x^k - x^*\rangle &= \langle x^k - y^k + F(y^k) - \beta_k F(x^k), x^k - x^*\rangle \\
&= \beta_k\langle x^k - z^k - F(x^k) + \frac{F(y^k)}{\beta_k}, x^k - x^*\rangle \\
&= \beta_k\langle x^k - F(x^k) - z^k, x^k - x^*\rangle + \langle F(y^k), x^k - x^*\rangle \\
&= \beta_k\langle x^k - F(x^k) - z^k, x^k - z^k\rangle + \beta_k\langle x^k - F(x^k) - z^k, z^k - x^*\rangle \\
&\quad + \langle F(y^k), x^k - y^k\rangle + \langle F(y^k), y^k - x^*\rangle \\
&\geq \beta_k\langle x^k - F(x^k) - z^k, x^k - z^k\rangle + \langle F(y^k), x^k - y^k\rangle \\
&= \beta_k\langle x^k - F(x^k) - z^k, x^k - z^k\rangle + \beta_k\langle F(y^k), x^k - z^k\rangle \\
&= \beta_k\|x^k - z^k\|^2 - \beta_k\langle F(x^k) - F(y^k), x^k - z^k\rangle.
\end{aligned}$$

This implies that

$$\begin{aligned}
\langle \beta_k d_k^3, x^k - x^*\rangle &\geq \beta_k^2\|x^k - z^k\|^2 - \beta_k^2\langle F(x^k) - F(y^k), x^k - z^k\rangle \\
&= \|x^k - y^k\|^2 - \beta_k\langle F(x^k) - F(y^k), x^k - y^k\rangle.
\end{aligned}$$

So, the direction d_k^3 satisfies (2.16). □

Remark 2.1. When $\rho = 0$, we have that $\rho_1 = 1$ and $\bar{d}^k = d^k$. So, if (2.19) is used instead of (2.18), we obtain that for

$$\alpha_k = \frac{(1-c)\|x^k - y^k\|^2}{\beta_k^2\|d^k\|^2}$$

the hyperplane $H^k := \{x \in \mathbb{R}^n | \langle d^k, x^k - x\rangle = \alpha_k\beta_k\|d^k\|^2\}$ also strictly separates x^k from S^*. With this choice for α_k and with $d^k = d_k^1$ Nguyen-Strodiot prototype Algorithm coincides with Algorithm 2 in [30]. In that case, it is not necessary to assume that F is co-coercive on X. The monotonicity of F is sufficient to ensure the convergence of the proposed algorithm.

The following lemma shows that Nguyen-Strodiot prototype Algorithm is well defined.

Lemma 2.2.5. *Suppose that F is μ-co-coercive on X. At the current iteration k, if $z^k = x^k$, then x^k is a solution to $QVI(K; F)$. Otherwise the line-search condition* (2.15) *holds after finitely many inner iterations.*

Proof. If $z^k = x^k$, then $x^k = P_{K(x^k)}(x^k - F(x^k))$, and thus x^k is a solution of $QVI(K;F)$. Next we suppose, to get a contradiction, that the line-search condition (2.15) is never satisfied. Then the following inequality is satisfied $\forall\ m \in \mathbb{N}$

$$\langle F(x^k) - F((1-\gamma l^m)x^k + \gamma l^m z^k),\, x^k - z^k\rangle > c\|x^k - z^k\|^2.$$

Using the Cauchy-Schwarz inequality on the left hand side of the last inequality and dividing both sides of the resulting inequality by $\|x^k - z^k\|$, we obtain that

$$\|F(x^k) - F((1-\gamma l^m)x^k + \gamma l^m z^k)\| > c\|x^k - z^k\| \tag{2.21}$$

On the other hand, since F is μ-co-coercive and thus $\frac{1}{\mu}$-Lipschitz continuous, we have that

$$\mu\|F(x^k) - F((1-\gamma l^m)x^k + \gamma l^m z^k)\| \le \gamma l^m\|x^k - z^k\|.$$

Combining this inequality with (2.21), we obtain

$$\frac{\gamma l^m}{c\mu} > 1.$$

Taking the limit of this inequality as $m \to \infty$, we deduce that $0 \ge 1$, which is impossible. So, the line-search condition (2.15) holds after finitely many iterations. $\square$

The main result that [15] wants to prove, is the convergence of Nguyen-Strodiot prototype Algorithm, which is stated in the following theorems.

Theorem 2.2.6. *Let $\{x^k\}$ be the sequence generated by Nguyen-Strodiot prototype Algorithm. Suppose that Assumption (A) is satisfied and that the parameters ρ, ρ_1, μ and c satisfy* (2.17)*. Suppose also that $y^k \ne x^k$ $\forall k$ and the sequence $\{d^k\}$ is bounded. Then the sequence $\{x^k\}$ generated by Nguyen-Strodiot prototype Algorithm is bounded, and any limit point of the sequence $\{x^k\}$ is a solution to $QVI(K;F)$.*

Precisely because of this result, after the proof of convergence, we aim to identify some concrete search directions d^k for which we can guarantee the boundedness of the sequence d^k (see Proposition 2.2.8 below)

Proof. Let $x^* \in S^*$ and let $k \in \mathbb{N}$. Then we have that $x^* \in K(x^k)$ and we obtain, using successively the non-expansiveness of the projection and (2.18)

$$\begin{aligned}
\|x^{k+1} - x^*\|^2 &= \|P_{K(x^k)}(x^k - \alpha_k\beta_k\bar{d}^k) - x^*\|^2 \\
&\le \|x^k - \alpha_k\beta_k\bar{d}^k - x^*\|^2 \\
&= \|x^k - x^*\|^2 - 2\alpha_k\langle\beta_k\bar{d}^k,\, x^k - x^*\rangle + \alpha_k^2\beta_k^2\|\bar{d}^k\|^2 \\
&\le \|x^k - x^*\|^2 - 2\alpha_k\Bigg[\left(1 - \frac{\rho_1\rho}{4\mu}\right)\|x^k - y^k\|^2 \\
&\quad - \rho_1\beta_k\langle F(x^k) - F(y^k),\, x^k - y^k\rangle\Bigg] + \alpha_k^2\beta_k^2\|\bar{d}^k\|^2.
\end{aligned}$$

Consequently, from the definition of α_k, we immediately deduce that

$$\|x^{k+1}-x^*\|^2 \leq \|x^k-x^*\|^2 - \frac{\left((1-\frac{\rho_1\rho}{4\mu})\|x^k-y^k\|^2 - \rho_1\beta_k\langle F(x^k)-F(y^k),\, x^k-y^k\rangle\right)^2}{\beta_k^2\|\bar{d}^k\|^2} \tag{2.22}$$

But (2.22) implies that $\|x^{k+1}-x^*\| \leq \|x^k-x^*\|$. So, the sequence $\{\|x^k-x^*\|\}$ is convergent and the sequence $\{x^k\}$ is bounded. Moreover thanks to (2.22), we have

$$\lim_{k\to\infty} \frac{(1-\frac{\rho_1\rho}{4\mu})\|x^k-y^k\|^2 - \rho_1\beta_k\langle F(x^k)-F(y^k),\, x^k-y^k\rangle}{\beta_k\|\bar{d}^k\|} = 0. \tag{2.23}$$

From (2.19), we obtain easily that

$$\frac{(1-\frac{\rho_1\rho}{4\mu})\|x^k-y^k\|^2 - \rho_1\beta_k\langle F(x^k)-F(y^k),\, x^k-y^k\rangle}{\beta_k\|\bar{d}^k\|} \geq \frac{\rho_1\bar{\rho}\|x^k-y^k\|^2}{\beta_k\|\bar{d}^k\|}$$

where $\bar{\rho} = 1-c+\rho(1-\frac{1}{4\mu})$. Therefore, we have from (2.23) and the definition of y^k that

$$\lim_{k\to\infty} \frac{\rho_1\bar{\rho}\beta_k\|x^k-z^k\|^2}{\|\bar{d}^k\|} = \lim_{k\to\infty} \frac{\rho_1\bar{\rho}\|x^k-y^k\|^2}{\beta_k\|\bar{d}^k\|} = 0. \tag{2.24}$$

Furthermore, it is easy to verify that $\{z^k\}$ is bounded. Indeed, since $x^* \in K(x^k)$, we have successively

$$\begin{aligned}
\|z^k\| &= \|P_{K(x^k)}(x^k - F(x^k))\| \\
&= \|P_{K(x^k)}(x^k - F(x^k)) + x^* - P_{K(x^k)}(x^*)\| \\
&\leq \|x^*\| + \|P_{K(x^k)}(x^k - F(x^k)) - P_{K(x^k)}(x^*)\| \\
&\leq \|x^*\| + \|x^k - x^*\| + \|F(x^k)\|.
\end{aligned}$$

Since F is continuous and the sequence $\{x^k\}$ is bounded, we can conclude that the sequence $\{z^k\}$ and $\{y^k\}$ are also bounded. In addition, since the sequence $\{d^k\}$ is bounded by assumption, we have also that the sequence $\{\bar{d}^k\}$ is bounded. Therefore it follows from (2.24) that

$$\lim_{k\to\infty} \beta_k\|x^k - z^k\|^2 = 0. \tag{2.25}$$

Let $\bar{x}$ be a limit point of $\{x^k\}$. Then there exists a subsequence $\{x^{k_j}\}$ of $\{x^k\}$ converging to $\bar{x}$ when $j \to \infty$. Two cases may occur:

Case 1: $\inf_j \beta_{k_j} = \beta_{\min} > 0$. Then by (2.25), we get $\lim_{j\to\infty}\|x^{k_j} - z^{k_j}\| = 0$.

Case 2: $\inf_j \beta_{k_j} = \beta_{\min} = 0$. Then there exists a subsequence of $\{\beta_{k_j}\}$ denoted again $\{\beta_{k_j}\}$ that converges to 0 as $j \to \infty$. So, for j large enough, $\beta_{k_j} = \gamma l^{m_j}$ with $m_j > 1$. Then $\gamma l^{m_j-1} \to 0$ and, for j large enough, we can write

$$\langle F(x_{k_j}) - F((1-\gamma l^{m_j-1})x^{k_j} + \gamma l^{m_j-1}z^{k_j}),\, x^{k_j} - z^{k_j}\rangle > c\|x^{k_j} - z^{k_j}\|^2.$$

Since F is co-coercive, F is also continuous, and using the Cauchy-Schwarz inequality, we obtain that $\lim_{j\to\infty}\|x^{k_j} - z^{k_j}\| = 0$.

Therefore, since $\|z^{k_j} - \bar{x}\| \leq \|z^{k_j} - x^{k_j}\| + \|x^{k_j} - \bar{x}\|$, we obtain in both cases that $z^{k_j} \xrightarrow{j\to\infty} \bar{x}$.

Moreover, by construction of z^k, we have that $z^k \in K(x^k) \ \ \forall k$. Hence K being upper semi-continuous on X, we deduce that $\bar{x} \in K(\bar{x})$.

On the other hand, since K is lower semi-continuous on X, $\forall w \in K(\bar{x})$, there exists a sequence $\{w^{k_j}\}$ with $w^{k_j} \in K(x^{k_j})$, such that $w^{k_j} \to w$. Since $z^{k_j} = P_{K(x^{k_j})}(x^{k_j} - F(x^{k_j}))$, we obtain that

$$\langle z^{k_j} - x^{k_j} + F(x^{k_j}),\, w^{k_j} - z^{k_j}\rangle \geq 0,$$

, i.e.,

$$\langle F(x^{k_j}),\, w^{k_j} - z^{k_j}\rangle + \langle z^{k_j} - x^{k_j},\, w^{k_j} - z^{k_j}\rangle \geq 0.$$

Taking the limit as $j \to \infty$ gives $\langle F(\bar{x}),\, w - \bar{x}\rangle \geq 0 \ \ \forall w \in K(\bar{x})$. But this means that $\bar{x}$ is a solution to $QVI(K;F)$. □

Remark 2.2. One way to obtain that the whole sequence $\{x^k\}$ generated by Nguyen-Strodiot prototype Algorithm converges to a solution $QVI(K;F)$ is to impose that every limit point of $\{x^k\}$ belongs to S^*. Indeed, let $\bar{x}$ be such a limit point. Using (2.22) with $x^* = \bar{x}$, we immediately deduce that the sequence $\{\|x^k - \bar{x}\|\}$ is convergent and thus that the sequence $\{x^k\}$ converges to a solution of $QVI(K;F)$.

In the next theorem, we give a condition to assure that every limit point of $\{x^k\}$ belongs to S^*.

Theorem 2.2.7. *If, in addition to the assumption of Theorem* 2.2.6*, the operator F is strictly monotone on X, then the sequence $\{x^k\}$ generated by Nguyen-Strodiot prototype Algorithm is convergent to a solution of $QVI(K;F)$.*

Proof. Let $\bar{x}$ be a limit point of the sequence $\{x^k\}$. By Theorem 2.2.6, $\bar{x}$ is a solution to $QVI(K;F)$ and by Remark 2.2, we have only to prove that $\bar{x} \in S^*$ to obtain that $\{x^k\}$ converges to $\bar{x}$. In that purpose, let $\{x^{k_j}\}$ be a subsequence of $\{x^k\}$ converging to $\bar{x}$ and let $x^* \in S^*$. Then $x^* \in K(x^{k_j}) \ \ \forall j$ and by the upper semi-continuity of K, $x^* \in K(\bar{x})$. Hence, $\bar{x}$ being a solution of $QVI(K;F)$, we can write that

$$\langle F(\bar{x}),\, x^* - \bar{x}\rangle \geq 0. \tag{2.26}$$

On the other hand, since $x^{k_j} \in K(x^{k_j}) \ \ \forall j$, we have, by definition of S^*, that

$$\langle F(x^*),\, x^{k_j} - x^*\rangle \geq 0 \quad \forall j.$$

So, taking the limit as $j \to \infty$, we obtain that

$$\langle F(x^*),\, \bar{x} - x^*\rangle \geq 0. \tag{2.27}$$

Consequently, from the monotonicity of F, (2.26) and (2.27), we deduce that

$$\langle F(x^*) - F(\bar{x}),\, \bar{x} - x^*\rangle = 0,$$

which implies that $\bar{x} = x^* \in S^*$ because F is strictly monotone on X □

Proposition 2.2.8. *The directions d_k^1, d_k^2 and d_k^3 introduced in Proposition* 2.2.2 *and* 2.2.3 *are bounded. So Nguyen-Strodiot prototype Algorithm is convergent when the sequence of directions $\{d^k\}$ is one of the sequences $\{d_k^1\}$, $\{d_k^2\}$ and $\{d_k^3\}$.*

Proof. From Theorem 2.2.7, it is sufficient to prove that each of the sequences of directions $\{d_k^1\}$, $\{d_k^2\}$ and $\{d_k^3\}$ is bounded. In this purpose, first we observe that $x^k \in K(x^k)\ \ \forall k$ and that, by the non-expansiveness of the projection,

$$\begin{aligned}\|z^k - x^k\| &= \|P_{K(x^k)}(x^k - F(x^k)) - P_{K(x^k)}(x^k)\| \\ &\leq \|F(x^k)\|.\end{aligned}$$

This implies that $\|y^k - x^k\| = \beta_k \|z^k - x^k\| \leq \beta_k \|F(x^k)\|$. Therefore, we have, for all k, that

$$\begin{aligned}\|d_k^1\| &= \|x^k - y^k + F(y^k)\| \\ &\leq \|x^k - y^k\| + \|F(y^k)\| \\ &\leq \beta_k \|F(x^k)\| + \|F(y^k)\|.\end{aligned}$$

Since F is continuous and the sequences $\{x^k\}$ and $\{y^k\}$ are bounded, we easily deduce that the sequence $\{d_k^1\}$ is bounded. On the other hand, the sequences $\{F(x^k)\}$ and $\{\beta_k\}$ being bounded, we also obtain the sequences $\{d_k^2\}$ and $\{d_k^3\}$ are bounded. □

In conclusion the algorithm that we will use to solve QVI is

Algorithm 5: Nguyen-Strodiot

Data: $x^0 \in X,\ l \in (0,1),\ c \in (0,1),\ \mu > \max\{\frac{1}{4}, \frac{\rho\,\rho_1}{4(1-\rho_1 c)}\}$,
$\rho \geq 0,\ \rho_1 = \frac{1}{1+\rho},\ \gamma \in (0,1)$.

for $k = 0, 1, \ldots$ **do**

Compute

$$\begin{aligned} z^k &= \arg\min_{z \in K(x^k)} \{\langle F(x^k),\, z - x^k\rangle + 1/2\|z - x^k\|^2\} \\ &= P_{K(x^k)}(x^k - F(x^k)); \end{aligned}$$

if $z^k = x^k$ **then**

Stop.

else

Find m_k the smallest $m \in \mathbb{N}$ such that

$$\langle F(x^k) - F((1-\gamma l^m)x^k + \gamma l^m z^k),\, x^k - z^k\rangle \leq c\|x^k - z^k\|^2$$

Set $y^k := (1-\beta_k)x^k + \beta_k z^k$ where $\beta_k = \gamma\, l^{m_k}$;

Choose a direction d^k among

- $d_k^1 = x^k - y^k + F(y^k)$
- $d_k^2 = x^k - y^k + F(x^k) + F(y^k)$
- $d_k^3 = x^k - y^k - \beta_k(F(x^k) + \frac{F(y^k)}{\beta_k})$

Compute

$$\begin{aligned} \bar{d}^k &= \frac{\rho}{1+\rho}(x^k - y^k) + \frac{1}{1+\rho}d^k \\ x^{k+1} &= P_{K(x^k)}(x^k - \alpha_k \beta_k \bar{d}^k) \end{aligned}$$

where

$$\alpha_k = \frac{(1 - \frac{\rho\rho_1}{4\mu})\|x^k - y^k\|^2 - \rho_1\beta_k\langle F(x^k) - F(y^k),\, x^k - y^k\rangle}{\beta_k^2\|\bar{d}^k\|^2}\ .$$

end

end

Chapter 3

Acceleration Method

The objective of this chapter is to present a current and very active research topic, namely some methods for accelerating the convergence of sequences in a vector space. It is well known that many methods used in numerical analysis and applied mathematics are iterative, for example fixed point methods as those presented in previous sections. It is well known, moreover, that iterative methods could be slowly convergent and many approaches have been devised to overcome this issue [3, 5]. In some cases, it is possible to modify the construction of the sequence itself. But, if the sequence is produced by a "*black box*", i.e., the user has no access to its computation, it is possible to use extrapolation techniques to transform this sequence into a new sequence which, under some assumptions, convergences faster.

We will consider two acceleration methods:

- Regularized nonlinear acceleration [24];
- Regularized Topological-Shanks-type acceleration [4].

Our aim will be to show the idea of the aforementioned acceleration techniques and how they combine with the hybrid extragradient methods that we have presented.

3.1 Regularized Nonlinear Acceleration (RNA)

3.1.1 The Idea

Assume we are using the *fixed-point iteration*

$$\tilde{\mathbf{x}}^{i+1} = g(\tilde{\mathbf{x}}^i), \quad \text{for } i = 0, \dots, k, \tag{3.1}$$

where $\tilde{x}^i \in \mathbb{R}^n$ and k is is a fixed integer.

The core idea behind this class of methods is to use a Taylor expansion of the function g in (3.1) to approximate the fixed point iterations by a vector autoregressive model, then compute a weighted mean of the iterates $\tilde{\mathbf{x}}^i$ to produce a better estimate of the limit $\mathbf{x}^*$. We assume $\mathbf{x}^*$ is unique.

Suppose $g(\mathbf{x})$ is differentiable and let G be the Jacobian of g evaluated at $\mathbf{x}^*$. We will assume that G is symmetric, positive semi-definite and $G \preceq \sigma I$, with $\sigma < 1$. Equation (3.1) becomes

$$\tilde{\mathbf{x}}^{i+1} = g(\mathbf{x}^*) + G(\tilde{\mathbf{x}}^i - \mathbf{x}^*) + O(\|\tilde{\mathbf{x}}^i - \mathbf{x}^*\|^2), \quad \text{for } i = 1, \ldots, k.$$

By neglecting the second order term, and because $g(x^*) = x^*$, we obtain the *linear fixed-point iteration*

$$\mathbf{x}^{i+1} - \mathbf{x}^* = G(\mathbf{x}^i - \mathbf{x}^*), \tag{3.2}$$

where $\mathbf{x}^0 = \tilde{\mathbf{x}}^0$. We can hence recognize in 3.2 a vector autoregressive process. Because $\|G\|_2 \leq \sigma < 1$, the iterates $\mathbf{x}^k$ converge to $\mathbf{x}^*$ at a linear rate, with

$$\|\mathbf{x}^i - \mathbf{x}^*\| \leq \sigma \|\mathbf{x}^{i-1} - \mathbf{x}^*\| \leq \sigma^i \|\mathbf{x}^0 - \mathbf{x}^*\|.$$

Suppose we run k iterations of (3.2), a linear combinations of iterates $\mathbf{x}^i$ with coefficients c_i reads

$$\begin{aligned}
\sum_{i=0}^{k} c_i \mathbf{x}^i &= \sum_{i=1}^{k} c_i \mathbf{x}^* + \sum_{i=1}^{k} c_i G(\mathbf{x}^i - \mathbf{x}^*) \\
&= \left(\sum_{i=0}^{k} c_i\right)\mathbf{x}^* + \left(\sum_{i=0}^{k} c_i G^i\right)(\mathbf{x}^0 - \mathbf{x}^*).
\end{aligned} \tag{3.3}$$

Defining the polynomial

$$p(z) := \sum_{i=0}^{k} c_i z^i, \tag{3.4}$$

we can write (3.3) more concisely in terms of the matrix polynomial $p(G)$, setting $p(1) = \sum_{i=0}^{k} c_i = 1$ without loss of generality, to get

$$\sum_{i=0}^{k} c_i \mathbf{x}^i = \mathbf{x}^* + \underbrace{p(G)(\mathbf{x}^0 - \mathbf{x}^*)}_{\text{Error term}}.$$

Ideally, we need to find c (or equivalently p) which minimizes the error term $p(G)(\mathbf{x}^0 - \mathbf{x}^*)$. Using [24], we know that the optimal solution satisfies

$$\begin{aligned}
\left\|\sum_{i=0}^{k} c_i^* \mathbf{x}^i - \mathbf{x}^*\right\| &= \min_{\{c \in \mathbb{R}^{k+1} :\, c^T \mathbf{1} = 1\}} \left\|\sum_{i=0}^{k} c_i G^i (\mathbf{x}^0 - \mathbf{x}^*)\right\| \\
&= \min_{\{p \in \mathbb{R}_k[x] :\, p(1) = 1\}} \left\| p(G)(\mathbf{x}^0 - \mathbf{x}^*)\right\|
\end{aligned}$$

where $\mathbb{R}_k[x]$ is the subspace of polynomials of degree at most k, i.e.,

$$c^* = \arg \min_{\{c \in \mathbb{R}^{k+1} :\, c^T \mathbf{1} = 1\}} \left\|\sum_{i=0}^{k} c_i G^i (\mathbf{x}^0 - \mathbf{x}^*)\right\|.$$

Now we focus on a method which will approximately minimize the error $\|p(G)(\mathbf{x}^0 - \mathbf{x}^*)\|$. Since we do not observe G and $\mathbf{x}^*$ we will work with the residuals

$$\tilde{\mathbf{r}}^i = \tilde{\mathbf{x}}^{i+1} - \tilde{\mathbf{x}}^i = g(\tilde{\mathbf{x}}^i) - \tilde{\mathbf{x}}^i. \tag{3.5}$$

Observe that, when g is a linear function (3.2) this becomes

$$\mathbf{r}^i = \mathbf{x}^{i+1} - \mathbf{x}^i = (G - I)(\mathbf{x}^i - \mathbf{x}^*) \tag{3.6}$$

A linear combination of residuals $\mathbf{r}^i$ with coefficients c_i is written

$$\sum_{i=0}^{k} c_i \mathbf{r}^i = (G - I) \sum_{i=0}^{k} c_i (\mathbf{x}^i - \mathbf{x}^*) = (G - I) p(G)(\mathbf{x}^0 - \mathbf{x}^*).$$

We recognize the error term we wanted to minimize, multiplied by the matrix $G - I$. Using the coefficients which minimize this alternative quantity will approximately minimize the error as stated in the following proposition.

Proposition 3.1.1 ([24]). *Let $p^*(x)$ be the polynomial solving*

$$p^*(x) = \min_{\{p \in \mathbb{R}_k[x] \colon p(1)=1\}} \left\| (G - I) p(G)(\boldsymbol{x}^0 - \boldsymbol{x}^*) \right\|.$$

Then its coefficients, denoted by $\mathbf{c}^*$, satisfy

$$\boldsymbol{c}^* = \arg \min_{\{\boldsymbol{c} \in \mathbb{R}^{k+1} \colon \boldsymbol{c}^T \boldsymbol{1} = 1\}} \left\| \sum_{i=0}^{k} c_i \boldsymbol{r}^i \right\|. \tag{3.7}$$

The iterates $\boldsymbol{x}^i$ defined in (3.2) *averaged with coefficients $\mathbf{c}^*$ satisfy*

$$\left\| \sum_{i=0}^{k} c_i^* \boldsymbol{x}^i - \boldsymbol{x}^* \right\| \leq \frac{1}{1-\sigma} \min_{\{\boldsymbol{c} \in \mathbb{R}^{k+1} \colon \boldsymbol{c}^T \boldsymbol{1} = 1\}} \left\| \sum_{i=0}^{k} c_i G^i (\boldsymbol{x}^0 - \boldsymbol{x}^*) \right\|, \tag{3.8}$$

where we have assumed $0 \preceq G \preceq \sigma I$, with $\sigma < 1$.

This leads to the following acceleration algorithm.

Algorithm 6: Nonlinear Acceleration of Convergence, [24]

Input: Iterates $\tilde{\mathbf{x}}^0, \tilde{\mathbf{x}}^1, \ldots, \tilde{\mathbf{x}}^{k+1} \in \mathbb{R}^d$.

1 Compute $\tilde{R} = [\tilde{\mathbf{r}}^0, \ldots, \tilde{\mathbf{r}}^k]$;

2 Solve

$$\mathbf{c}^* = \arg \min_{\{\mathbf{c} \in \mathbb{R}^{k+1} \colon \mathbf{c}^T \mathbf{1} = 1\}} \|\tilde{R}\mathbf{c}\|$$

Output: Approximation of $\mathbf{x}^*$ ensuring (3.8), computed as $\sum_{i=0}^{k} c_i^* \tilde{\mathbf{x}}^i$

The next proposition gives us an explicit solution, involving involving the solution of $k \times k$ linear system.

Proposition 3.1.2 ([24])**.** *The explicit solution of the problem*

$$\boldsymbol{c}^* = \arg \min_{\boldsymbol{c}^T \boldsymbol{1}=1} \|\tilde{R}\boldsymbol{c}\| \tag{3.9}$$

in the variable $\boldsymbol{c} \in \mathbb{R}^k$, where $\tilde{R}$ is a $d \times k$ matrix assumed to be of rank k is given by

$$\boldsymbol{c}^* = \frac{(\tilde{R}^T\tilde{R})^{-1}\boldsymbol{1}}{\boldsymbol{1}^T(\tilde{R}^T\tilde{R})^{-1}\boldsymbol{1}}. \tag{3.10}$$

Remark 3.1. In practice, instead of computing the inverse of the matrix $\tilde{R}^T\tilde{R}$, we solve the linear system

$$\tilde{R}^T\tilde{R}\mathbf{z} = \mathbf{1},$$

and then set

$$\mathbf{c}^* = \frac{\mathbf{z}}{\mathbf{1}^T\mathbf{z}}.$$

So far, we have only considered linear function G in (3.2), when computing the iterates $\mathbf{x}^i$. In general, the fixed point iteration (3.1) is usually generated by a nonlinear function g, thus inducing a second order error term in $O(\|\mathbf{x}^i - \mathbf{x}^*\|^2)$ compared to the dynamics in (3.2). In fact even in practical cases where k is small, $\tilde{R}^T\tilde{R}$ is usually a singular or nearly singular matrix, that means that even if the perturbations are small, their impact on the solution can be arbitrarily large. This particular issue means that the linear system $(\tilde{R}^T\tilde{R})^{-1}\mathbf{1}$ in (3.10) needs to be regularized. This brought to derive a regularized version of Algorithm 6, which better controls the impact of perturbations, using Tikhonov regularization in order to solve the linear system (3.10). This leads to the *Regularized Nonlinear Acceleration*:

Algorithm 7: Regularized Nonlinear Acceleration (RNA), [24]

Input: Iterates $\tilde{\mathbf{x}}^0, \tilde{\mathbf{x}}^1, \ldots, \tilde{\mathbf{x}}^{k+1} \in \mathbb{R}^d$ produced by (3.1), $\lambda > 0$ regularization parameter.

1 Compute $R = [\tilde{\mathbf{r}}^0, \ldots, \tilde{\mathbf{r}}^k]$ where $\tilde{\mathbf{r}}^i = \tilde{\mathbf{x}}^{i+1} - \tilde{\mathbf{x}}^i$;

2 Solve

$$\tilde{\mathbf{c}}^*_\lambda = \arg \min_{\mathbf{c}^T\mathbf{1}=1} \|\tilde{R}\mathbf{c}\|^2 + \lambda\|\mathbf{c}\|^2$$

or equivalently solve $(\tilde{R}^T\tilde{R} + \lambda I)\mathbf{z} = \mathbf{1}$ then set $\tilde{\mathbf{c}}^*_\lambda = \dfrac{\mathbf{z}}{\mathbf{1}^T\mathbf{z}}$;

Output: Approximation of $\mathbf{x}^*$ computed as $\mathbf{x}_{extr}(\lambda) = \sum_{i=0}^k (\tilde{\mathbf{c}}^*_\lambda)_i \mathbf{x}^i$

Notice that regularization allows a better control of the impact of perturbations, but also changes the solution $\mathbf{c}^*$ into $\mathbf{c}^*_\lambda$ in Algorithm 6.

3.1.2 RNA for QVI

In this subsection we present how we incorporate the Regularized Nonlinear Acceleration (RNA) in order to accelerate our fixed-point methods (Solodov, Nguyen-Strodiot). Let us suppose we want to solve the fixed point problem $f(\mathbf{x}) = \mathbf{x}$, then the main structure of our Algorithm will be

Algorithm 8: Prototype RNA

Data: Choose $Nmax \in \mathbb{N}$ (outer cycles), $Kmax \in \mathbb{N}$ (inner cycles), λ regularization parameter, $\mathbf{x}^0 \in X$.

```
for i = 0, 1, ..., Nmax do
    Set u^0 = x^i;
    for n = 1, ..., 2*Kmax + 1 do
        Compute u^n = f(u^{n-1});
    end
    Apply the RNA to u^0, ..., u^{2*Kmax+1} using λ;
    Set x^i = x*_extr(λ);
end
```

The major problem of the Regularized Nonlinear Acceleration, Algorithm 7, is the presence of the parameter λ, unknown in advance. To avoid this problem we use an adaptive strategy to find λ, based on grid search.

Since we restart our algorithm with $\mathbf{u}^0 = \mathbf{x}^i = \mathbf{x}^*_{extr}(\lambda)$ for $i = 1, \dots, Nmax$, obviously we are interested in finding the best λ that minimizes the residuals. In fact we are using fixed-point methods whose stopping criteria is the coincidence of the prediction step with the previous iteration, i.e.,

$$\mathbf{x}^k - P_{K(\mathbf{x}^k)}(\mathbf{x}^k - F(\mathbf{x}^k)) = 0.$$

For the above reason, in practice, we choose λ such that

$$\min_{\lambda} \|\mathbf{x}_{extr}(\lambda) - P_{K(\mathbf{x}_{extr}(\lambda))}(\mathbf{x}_{extr}(\lambda) - F(\mathbf{x}_{extr}(\lambda)))\|^2.$$

We use a grid of dimension $Kmax$ in order to find a good λ, i.e., we solve

$$\min_{j=1,\dots,Kmax} \|\mathbf{x}_{extr}(\lambda_j) - P_{K(\mathbf{x}_{extr}(\lambda_j))}(\mathbf{x}_{extr}(\lambda_j) - F(\mathbf{x}_{extr}(\lambda_j)))\|^2.$$

In conclusion our algorithm becomes

Algorithm 9: Regularized Fixed-Point Method

Data: Choose $Nmax \in \mathbb{N}$ (outer cycles), $Kmax \in \mathbb{N}$ (inner cycles), $\mathbf{x}^0 \in X$, $c \in (0,1)$, $\alpha \in (0,1)$. Set bounds $[\lambda_{min}, \lambda_{max}]$.

1 Divide the segment $[\lambda_{min}, \lambda_{max}]$ into $Kmax$ points $\{\lambda_j\}$ using a logarithmic scale;
2 **for** $i = 0, 1, \ldots, Nmax$ **do**
3   Set $\mathbf{u}^0 = \mathbf{x}^i$;
4   **for** $n = 1, \ldots, 2^* Kmax + 1$ **do**
5   Compute $\mathbf{u}^n = f(\mathbf{u}^{n-1})$;
6   **end**
7   Compute the residual matrix $\tilde{R}$ such that $\tilde{R}_i = \mathbf{u}^{i+1} - \mathbf{u}^i$;
8   Build the matrix $M = \tilde{R}^T \tilde{R} / \|\tilde{R}^T \tilde{R}\|$;
9   **for** $j = 1, \ldots, Kmax$ **do**
10   Solve in $\mathbf{z}$ the linear system $(M + \lambda_j I)\mathbf{z} = \mathbf{1}$;
11   Normalize the solution $\tilde{\mathbf{c}}^*_{\lambda_j} = \mathbf{z}/(\mathbf{1}^T \mathbf{z})$;
12   Compute $\mathbf{x}_{extr}(\lambda_j) = \sum_{h=0}^{Kmax} (\tilde{c}^*_{\lambda_j})_h \mathbf{u}^h$;
13   **end**
14   Pick

$$\lambda^* = \arg \min_{j=1,\ldots,Kmax} \|\mathbf{x}_{extr}(\lambda_j) - P_{K(\mathbf{x}_{extr}(\lambda_j))}(\mathbf{x}_{extr}(\lambda_j) - F(\mathbf{x}_{extr}(\lambda_j)))\|^2;$$

$$\text{Set} \quad \mathbf{x}^*_{extr} = \mathbf{x}_{extr}(\lambda^*) \quad \text{and} \quad \mathbf{x}^i = \mathbf{x}^*_{extr};$$

15 **end**

Remark 3.2. Notice that the function f in Algorithm 9 can be substitute with Generalized Solodov (Algorithm 3) or with Nguyen-Strodiot (Algorithm 5) and we will call Algorithm 9 *Regularized Solodov* or *Regularized Nguyen-Strodiot* respectively.

Remark 3.3. Observe that step 6 to step 12, except step 9, in Algorithm 9 the Regularizd Nonlinear Acceleration (Algorithm 7).

Remark 3.4. Last but not least observation, we explain the choice of the number of the inner cycles. We choose $2^* Kmax + 1$ inner iteration so that we can easily compare it with the Restarted Topological-Shanks-type acceleration in the numerical experiences.

3.2 Regularized Topological Shanks Acceleration (RTSA)

3.2.1 Topological Shanks Transformations

Definition 3.1. A sequence $\{\mathbf{x}^i\}$ is in the *Shanks Kernel* if there exists $\mathbf{x}^* \in \mathbb{R}^d$, $\ell_0, \ldots, \ell_\nu \in \mathbb{R}$ with $\ell_0 + \ldots + \ell_\nu \neq 0$ such that, for all $i \geq 0$

$$\ell_0(\mathbf{x}^i - \mathbf{x}^*) + \ldots + \ell_\nu(\mathbf{x}^{i+\nu} - \mathbf{x}^*) = 0. \tag{3.11}$$

We suppose that ν is the minimal integer for which (3.11) holds

Definition 3.2. Let us define the *minimal polinomial* of A with respect to $\mathbf{v} \in \mathbb{R}^d$ as the monic polynomial of minimal degree such that $p(A)\mathbf{v} = 0$. If such polynomial of has degree ν we write $p_\nu(x)$.

It can be shown the following result:

Theorem 3.2.1 ([5])**.** *Suppose that there exists $\boldsymbol{x}^*$ such that $\boldsymbol{x}^* = A\boldsymbol{x}^*$. Let us consider a Picard iteration of the form $\boldsymbol{x}^i = A\boldsymbol{x}^{i-1}$. If $\boldsymbol{x}^0$ is such that $\boldsymbol{x}^0 - \boldsymbol{x}^*$ has a minimal polynomial $q_\nu(t) = \sum_{j=0}^{\nu} \ell_j t^j$ for which $\sum_{j=0}^{\nu} \ell_j \neq 0$, then $\{\boldsymbol{x}^i\}$ is in the Shanks Kernel.*

Consider a sequence $\{\mathbf{x}^i\}$ belonging to the Shanks Kernel, then for every $i \geq 0$, using the normalization condition on the coefficients c_j for $j = 0, \ldots, \nu$, we have an explicit expression of the limit in terms of the element of the computed sequence:

$$\sum_{j=0}^{\nu} c_j \mathbf{x}^{i+j} = \mathbf{x}^* \tag{3.12}$$

Observe that (3.12) holds for every $i \geq 0$, so we can just write

$$\sum_{j=0}^{\nu} c_j \mathbf{x}^{i+j+1} - \sum_{j=0}^{\nu} c_j \mathbf{x}^{i+j} = 0. \tag{3.13}$$

Let us define $R := [\mathbf{r}^i, \ldots, \mathbf{r}^{i+\nu}]$ with $\mathbf{r}^{i+j} := \mathbf{x}^{i+j+1} - \mathbf{x}^{i+j}$ for $j = 0, \ldots, \nu$. From (3.13), it is clear that it must be $Rank(R) < \nu + 1$, actually it can be shown that $Rank(R) = \nu$.

Let us select an element $\mathbf{y} \in \mathbb{R}^d$ and use it to multiply (3.12), obtaining

$$\sum_{j=0}^{\nu} c_j \mathbf{y}^T \mathbf{x}^{i+j} = \mathbf{y}^T \mathbf{x}^*. \tag{3.14}$$

Of course, we need to obtain $\nu + 1$ equations of this type to be able to recover the coefficients, i.e., we need to produce $2\nu + 2$ element of the sequence and consider

$$\sum_{j=0}^{\nu} c_j \mathbf{y}^T \mathbf{r}^{i+j+h} = \sum_{j=0}^{\nu} c_j \mathbf{y}^T (\mathbf{x}^{i+j+1+h} - \mathbf{s}^{i+j+h}) = 0 \quad \text{for } h = 0, \ldots, \nu.$$

Defining $b_h := \mathbf{y}^T\mathbf{r}^{i+h}$ for $h = 0,\dots,2\nu$ and defining the Hankel matrix,

$$T^{(n,\nu)} := \begin{bmatrix} b_0 & \dots & b_\nu \\ \vdots & & \vdots \\ b_\nu & \dots & b_{2\nu} \end{bmatrix},$$

we can determine the coefficients c_j solving the following problem

$$\mathbf{c} = \arg \min_{\mathbf{t}\in\mathbb{R}^{\nu+1}:\sum_{j=0}^{\nu} t_j=1} \|T^{(i,\nu)}\mathbf{t}\|.$$

The limit can be *extrapolated* just looking at $\nu+1$ elements of the sequence and using one of the two relations

$$\mathbf{x}^* = \sum_{j=0}^{\nu} c_j \mathbf{x}^{i+j} \quad \text{or} \quad \mathbf{x}^* = \sum_{j=0}^{\nu} c_j \mathbf{x}^{i+j+1}.$$

Suppose now we have produced a certain number of iterations, say the $2k+2$ iterations $\mathbf{x}^0,\dots,\mathbf{x}^{2k+1}$. It is possible to produce an extrapolated approximation solving the problem

$$\mathbf{c} = \arg \min_{\mathbf{t}\in\mathbb{R}^{k+1}:\sum_{j=0}^{k} t_j=1} \|T^{(0,k)}\mathbf{t}\|.$$

The following algorithm formalizes this heuristic:

Algorithm 10: Restarted Topological method, [4]

Data: Choose $Nmax \in \mathbb{N}$ (outer cycles), $k \in \mathbb{N}$ (inner cycles), $\mathbf{x}^0, \mathbf{y} \in \mathbb{R}^d$.

for $i = 0, 1, \dots, Nmax$ **do**
- Set $\mathbf{s}^0 = \mathbf{x}^i$;
- **for** $n = 1, \dots, 2^*k + 1$ **do**
  - Compute $\mathbf{s}^n = A\mathbf{s}^{n-1}$;
- **end**
- Compute $T^{(0,k)}$;
- Solve $\mathbf{c} = \arg\min_{\mathbf{t}\in\mathbb{R}^{k+1}:\sum_{j=0}^{k} t_j=1} \|T^{(0,k)}\mathbf{t}\|$;
- Set $\mathbf{x}^i = \sum_{j=0}^{k} c_j \mathbf{s}^{k+1+j}$;
- Select $\mathbf{y} \in \mathbb{R}^d$;

end

if $Rank(T^{(0,k)}) = k+1$, the solution is

$$\mathbf{c} = \frac{({T^{(0,k)}}^T T^{(0,k)})^{-1}\mathbf{1}}{\mathbf{1}^T({T^{(0,k)}}^T T^{(0,k)})^{-1}\mathbf{1}}.$$

Moreover, observe that $T^{(0,k)}$ could be a ill conditioned matrix (or better, we aspect this matrix to be singular), and hence we propose to solve

$$\mathbf{c} = \arg \min_{\mathbf{t}\in\mathbb{R}^{k+1}:\sum_{j=0}^{k} t_j=1} \|T^{(0,k)}\mathbf{t}\| + \lambda\|\mathbf{t}\|$$

with solution

$$\mathbf{c} = \frac{({T^{(0,k)}}^T T^{(0,k)} + \lambda I)^{-1}\mathbf{1}}{\mathbf{1}^T({T^{(0,k)}}^T T^{(0,k)} + \lambda I)^{-1}\mathbf{1}}.$$

The following algorithm represents a general computational scheme for fixed point problems where the problem concerning the choice of the regularization parameter is addressed:

Algorithm 11: Regularized Topological Shanks type Acceleration, [4]

Data: Choose $Nmax \in \mathbb{N}$ (outer cycles), $k \in \mathbb{N}$ (inner cycles), $\mathbf{x}^0, \mathbf{y} \in \mathbb{R}^d$.

for $i = 0, 1, \dots, Nmax$ **do**
    Set $\mathbf{s}^0 = \mathbf{x}^i$;
    for $n = 1, \dots, 2^*k + 1$ **do**
        Compute $\mathbf{s}^n = A\mathbf{s}^{n-1}$;
    end
    Compute $b_i = (\mathbf{y}^T R)_i$ and $T^{(0,k)}$;
    for $\lambda \in [\lambda_{min}, \lambda_{max}]$ **do**
        Solve $\mathbf{c}^\lambda = \arg\min_{\mathbf{t} \in \mathbb{R}^{k+1} : \mathbf{t}^T \mathbf{1} = 1} \|T^{(0,k)}\mathbf{t}\| + \lambda\|\mathbf{t}\|$;
        Set $\mathbf{x}_\lambda = \sum_{j=0}^{k} c_j^\lambda \mathbf{s}^{k+1+j}$;
    end
    $\mathbf{x}^i = \arg\min_{\lambda \in [\lambda_{min}, \lambda_{max}]} \|A\mathbf{x}_\lambda - \mathbf{x}_\lambda\|$;
    Choose $\mathbf{y} = \mathbf{x}^i \in \mathbb{R}^d$;
end

3.2.2 RTSA for QVI

Like we did before with the regularized nonlinear acceleration, we want to present how we incorporate the Restarted Topological Shanks Acceleration (RTSA) in order to accelerate our fixed-point methods (Solodov, Nguyen-Strodiot). Again we have the same problem: the choice of the regularization parameter λ that is unknown. We applay the same argument that we did before and in conclusion our algorithm becomes

Algorithm 12: Regularized Topological Fixed-Point Method

Data: Choose $Nmax \in \mathbb{N}$ (outer cycles), $Kmax \in \mathbb{N}$ (inner cycles), $x^0 \in X$. Set bounds $[\lambda_{min}, \lambda_{max}]$.

Divide the segment $[\lambda_{min}, \lambda_{max}]$ into $Kmax$ points $\{\lambda_j\}$ using a logarithmic scale;
for $i = 0, 1, \ldots, Nmax$ **do**
  Set $\mathbf{s}^0 = \mathbf{x}^i$;
  for $n = 1, \ldots, 2^*Kmax + 1$ **do**
    Compute $\mathbf{s}^n = f(\mathbf{s}^{n-1})$;
  end
  Set $\mathbf{y} = \mathbf{s}^{2*Kmax+1}$;
  Compute $b_i = (\mathbf{y}^T \tilde{R})_i$ and $T^{(0,Kmax)}$;
  for $\lambda \in [\lambda_{min}, \lambda_{max}]$ **do**
    Solve $\mathbf{c}^\lambda = \arg\min_{\mathbf{t} \in \mathbb{R}^{Kmax+1} : \mathbf{t}^T \mathbf{1} = 1} \|T^{(0,Kmax)}\mathbf{t}\| + \lambda \|\mathbf{t}\|$;
    Set $\mathbf{x}_\lambda = \sum_{j=0}^{Kmax} c_j^\lambda \mathbf{s}^{Kmax+1+j}$;
  end
  Set
$$\mathbf{x}^i = \arg \min_{\lambda \in [\lambda_{min}, \lambda_{max}]} \|x_\lambda - P_{K(x_\lambda)}(x_\lambda - F(x_\lambda))\|^2;$$
end

Remark 3.5. Notice that the function f in Algorithm 12 can be substitute with Generalized Solodov (Algorithm 3) or with Nguyen-Strodiot (Algorithm 5) and we will call Algorithm 12 *Regularized Topological Solodov* or *Regularized Topological Nguyen-Strodiot* respectively.

Chapter 4

Numerical Results

In this chapter our aim is to give some insight into the performance of our accelerated fixed-point methods. We have implemented these algorithms in MATLAB version R2018b to solve various quasi-variational inequality problem. Some of the test problems are the numerical experiments examined in [26] and [15], the others come from [8].

Each QVI is defined by the function F and the point-to-set mapping $K(x)$. We assume that $K(x)$ is defined as the intersection of a fixed set $\bar{K}$ and a set $\tilde{K}(x)$ that depends on the point x: $K(x) = \bar{K} \cap \tilde{K}(x)$. The sets $\bar{K}$ and $\tilde{K}(x)$ are described by inequalities and equalities:

$$\bar{K} := \{y \in \mathbb{R}^n \,|\, g^I(y) \leq 0,\ M^I y + v^I = 0\}$$
$$\tilde{K}(x) := \{y \in \mathbb{R}^n \,|\, g^P(y,x) \leq 0,\ M^P(x)y + v^P(x) = 0\}.$$

The constraints defining the set $\bar{K}$ are individual constraints that are independent of x, we use the superscript "I" in our notation (for individual/independent of x). On the other hand, the constraints defining $\tilde{K}(x)$ are parametric due to their dependence on x, therefore, we use the superscript "P" (for parametric). We assume that $g^I(\cdot)$ is a vector of convex functions and that each component function of $g^P(\cdot, x)$ is convex for all x. When we refer to the whole set of inequality or (linear) equality constraints, we use the notation

$$g(y,x) := \begin{pmatrix} g^I(y) \\ g^P(y,x) \end{pmatrix}, \quad M(x)y + v(x) := \begin{pmatrix} M^I \\ M^P(x) \end{pmatrix} y + \begin{pmatrix} v^I \\ v^P(x) \end{pmatrix}.$$

The type of constraints we focus on are the linear and bound ones, that means that g^I is linear and defines bounds on the variables, while g^P has the form $a^T y + b(x) - c \leq 0$ and $ay^i + b(x) - c \leq 0$. This characteristic choice is due to the fact that these linear and bound constraints make the QVI problem efficiently solvable. We use the quadratic-program solver `quadprog` from MATLAB optimization toolbox to perform the projection. Since `quadprog` can only works on linear constraints (independent of x or parametric), we had to rewrite the QVI test problems in an acceptable form in [8] to make `quadprog` work.

In literature, the CPU time is usually chosen as measure of efficiency. Nevertheless, we have decided to not taking into account this measure because our goal is to show the acceleration performance of the extrapolation algorithms, so we have run a fixed number of iterations.

Note that when performing the grid search the λ_j are independent of each other, so it would be possible to calculate them in parallel instead of using a loop. This choice would lead to a big gain of time.

For our comparison we have implemented in MATLAB four algorithms corresponding to Regularized Solodov, Regularized Nguyen-Strodiot for RNA and Regularized topological Solodov, Regularized topological Nguyen-Strodiot for RTSA. For the algorithms linked to Nguyen-Strodiot method we have also studied them changing the choice of the direction d^k with d_k^1, d_k^2 and d_k^3. In our experiments we have chosen the following parameters:

- Solodov: $c = 0.5$, $\alpha = 0.5$ and $\gamma = 1.99$ like in [26];
- Nguyen-Strodiot: $c = 0.5$, $l = 0.5$, $\gamma = 0.99$, $\rho = 1$ and $\mu = 0.5$ like in [15].

Let us point out that the parameter $Kmax$ is connected with the extrapolation routine and affects the acceleration performance. We propose here the $Kmax$ for which we obtain the best acceleration performance for RTSA and RNA.

4.1 OutZ

4.1.1 OutZ40

Source : [8]

Description :

$$
\begin{aligned}
F(x) &:= \begin{pmatrix} 2 & 8/3 \\ 5/4 & 2 \end{pmatrix} x - \begin{pmatrix} 34 \\ 24.25 \end{pmatrix}, \\
g^I(y) &:= \begin{pmatrix} -1 & 0 \\ 1 & 0 \\ 0 & -1 \\ 0 & 1 \end{pmatrix} y - \begin{pmatrix} 0 \\ 11 \\ 0 \\ 11 \end{pmatrix}, \\
g^P(y,x) &:= \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} y - \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} x - \begin{pmatrix} 15 \\ 15 \end{pmatrix}.
\end{aligned}
$$

Known solution : $x^* = (5, 9)^T$.

For both the extrapolated methods we took (0,0) as starting point, OptimalityTolerance $= 1e - 20$ and MaxIterations $= 500$.

Best parametric choice of Kmax for RTSA

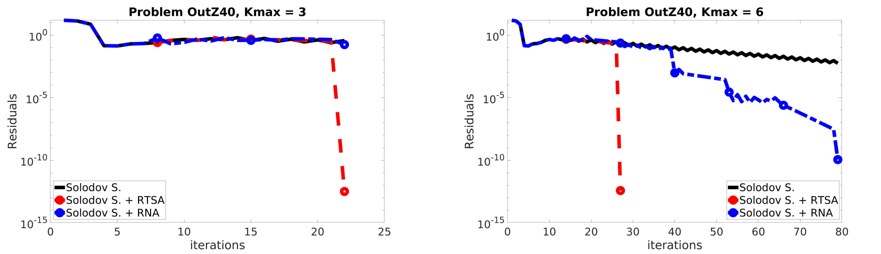
We took for

- Solodov: $Nmax = 3$, $Kmax = 3$;
- Nguyen-Strodiot:
  - d_k^1: $Nmax = 5$, $Kmax = 12$;
  - d_k^2: $Nmax = 5$, $Kmax = 17$;
  - d_k^3: $Nmax = 5$, $Kmax = 12$;

Best parametric choice of Kmax for RNA

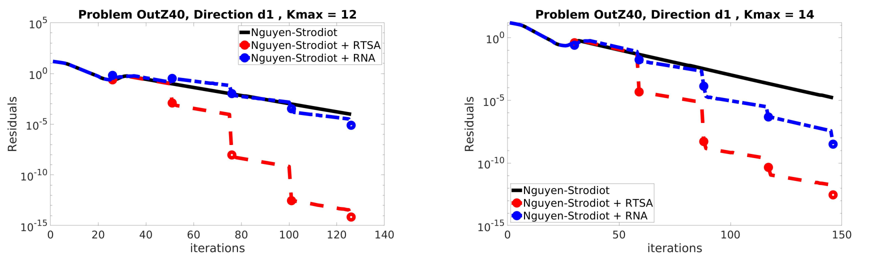
We took for

- Solodov: $Nmax = 6$, $Kmax = 6$;
- Nguyen-Strodiot:
  - d_k^1: $Nmax = 5$, $Kmax = 14$;
  - d_k^2: $Nmax = 4$, $Kmax = 23$;
  - d_k^3: $Nmax = 5$, $Kmax = 24$;



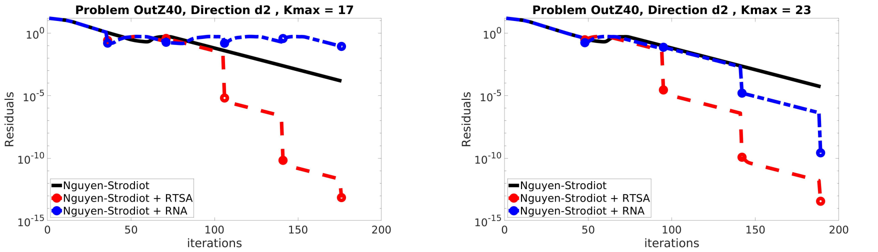
(a) *Solodov with best Kmax choice for RTSA* (b) *Solodov with best Kmax choice for RNA*

Figure 4.1: OutZ40 solved with Solodov



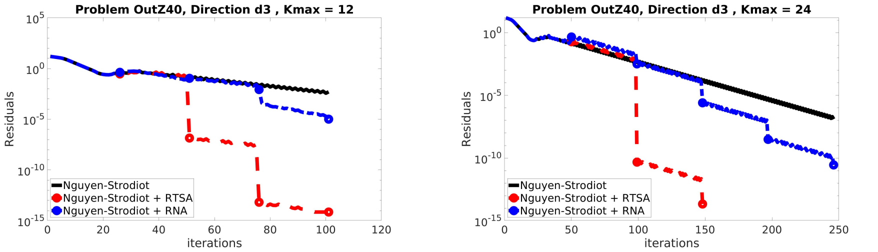
(a) *Nguyen-Strodiot with best Kmax choice for RTSA* (b) *Nguyen-Strodiot with best Kmax choice for RNA*

Figure 4.2: OutZ40 solved with Nguyen-Strodiot with d_k^1



(a) *Nguyen-Strodiot with best Kmax choice for RTSA* (b) *Nguyen-Strodiot with best Kmax choice for RNA*

Figure 4.3: OutZ40 solved with Nguyen-Strodiot with d_k^2



(a) *Nguyen-Strodiot with best Kmax choice for RTSA* (b) *Nguyen-Strodiot with best Kmax choice for RNA*

Figure 4.4: OutZ40 solved with Nguyen-Strodiot with d_k^3

4.1.2 OutZ41

Source : [8]
Description :

$$F(x) := \begin{pmatrix} 2 & 8/3 \\ 5/4 & 2 \end{pmatrix} x - \begin{pmatrix} 100/3 \\ 22.5 \end{pmatrix},$$

$$g^I(y) := \begin{pmatrix} -1 & 0 \\ 1 & 0 \\ 0 & -1 \\ 0 & 1 \end{pmatrix} y - \begin{pmatrix} 0 \\ 11 \\ 0 \\ 11 \end{pmatrix},$$

$$g^P(y,x) := \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} y - \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} x - \begin{pmatrix} 15 \\ 20 \end{pmatrix}.$$

Known solution : $x^* = (10, 5)^T$.

For both the extrapolated methods we took (0,0) as starting point, OptimalityTolerance $= 1e - 20$ and MaxIterations $= 500$.

Best parametric choice of Kmax for RTSA

We took for

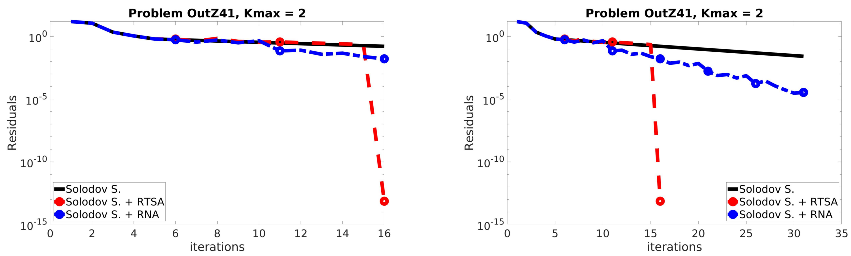
- Solodov: $Nmax = 3$, $Kmax = 2$;

- Nguyen-Strodiot:
  - d_k^1: $Nmax = 5$, $Kmax = 7$;
  - d_k^2: $Nmax = 5$, $Kmax = 15$;
  - d_k^3: $Nmax = 5$, $Kmax = 7$;

Best parametric choice of Kmax for RNA

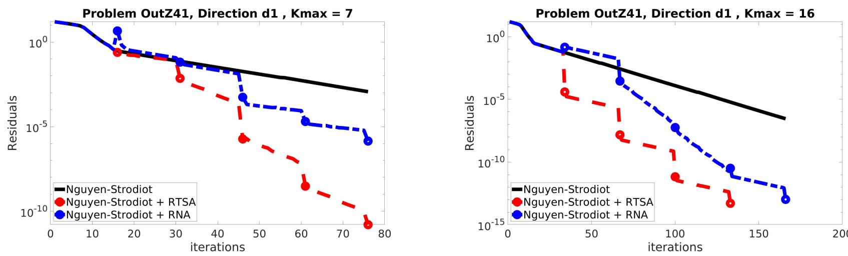
We took for

- Solodov: $Nmax = 6$, $Kmax = 2$;
- Nguyen-Strodiot:
  - d_k^1: $Nmax = 5$, $Kmax = 16$;
  - d_k^2: $Nmax = 5$, $Kmax = 19$;
  - d_k^3: $Nmax = 5$, $Kmax = 11$;



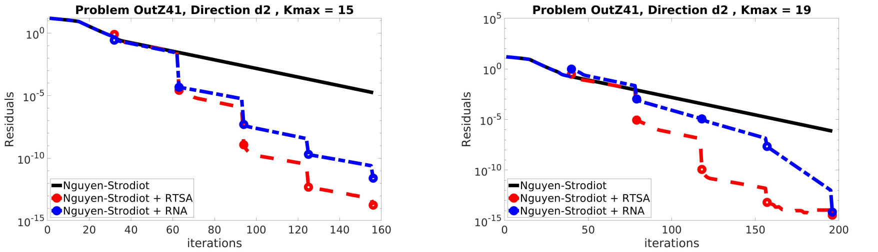
(a) *Solodov with best Kmax choice for RTSA* (b) *Solodov with best Kmax choice for RNA*

Figure 4.5: OutZ41 solved with Solodov



(a) *Nguyen-Strodiot with best Kmax choice for RTSA* (b) *Nguyen-Strodiot with best Kmax choice for RNA*

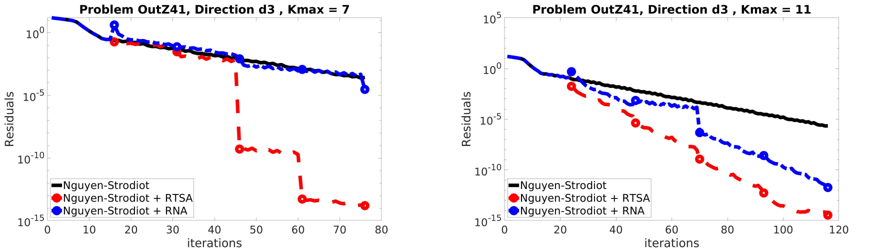
Figure 4.6: OutZ41 solved with Nguyen-Strodiot with d_k^1



(a) *Nguyen-Strodiot with best Kmax choice for RTSA*

(b) *Nguyen-Strodiot with best Kmax choice for RNA*

Figure 4.7: OutZ41 solved with Nguyen-Strodiot with d_k^2



(a) *Nguyen-Strodiot with best Kmax choice for RTSA*

(b) *Nguyen-Strodiot with best Kmax choice for RNA*

Figure 4.8: OutZ41 solved with Nguyen-Strodiot with d_k^3

4.1.3 OutZ45

Source : [26], [15]

Description :

$$F(x) := \begin{pmatrix} 2 & 8/3 \\ 5/4 & 2 \end{pmatrix} x - \begin{pmatrix} 34 \\ 24.25 \end{pmatrix},$$
$$g^I(y) := \begin{pmatrix} -1 & 0 \\ 1 & 0 \\ 0 & -1 \\ 0 & 1 \end{pmatrix} y - \begin{pmatrix} 0 \\ 10 \\ 0 \\ 10 \end{pmatrix},$$
$$g^P(y,x) := \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} y - \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} x - \begin{pmatrix} 15 \\ 15 \end{pmatrix}.$$

Known solution : $x^* = (5,9)^T$.

For both the extrapolated methods we took (0,0) as starting point, OptimalityTolerance $= 1e-20$ and MaxIterations $= 500$.

Best parametric choice of Kmax for RTSA

We took for

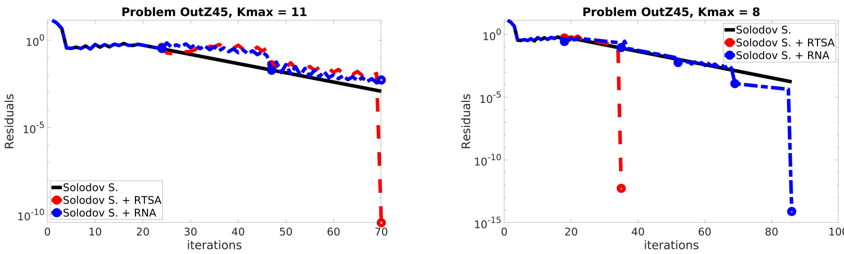
- Solodov: $Nmax = 3$, $Kmax = 11$;

- Nguyen-Strodiot:
  - d_k^1: $Nmax = 5$, $Kmax = 11$;
  - d_k^2: $Nmax = 5$, $Kmax = 19$;
  - d_k^3: $Nmax = 5$, $Kmax = 11$;

Best parametric choice of Kmax for RNA

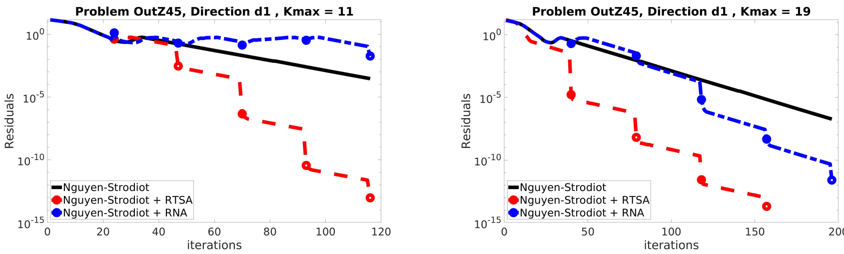
We took for

- Solodov: $Nmax = 5$, $Kmax = 8$;
- Nguyen-Strodiot:
  - d_k^1: $Nmax = 5$, $Kmax = 19$;
  - d_k^2: $Nmax = 5$, $Kmax = 22$;
  - d_k^3: $Nmax = 5$, $Kmax = 17$;



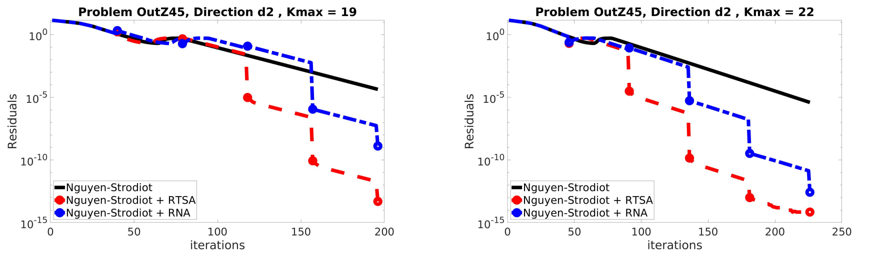
(a) *Solodov with best Kmax choice for RTSA* (b) *Solodov with best Kmax choice for RNA*

Figure 4.9: OutZ45 solved with Solodov



(a) *Nguyen-Strodiot with best Kmax choice for RTSA* (b) *Nguyen-Strodiot with best Kmax choice for RNA*

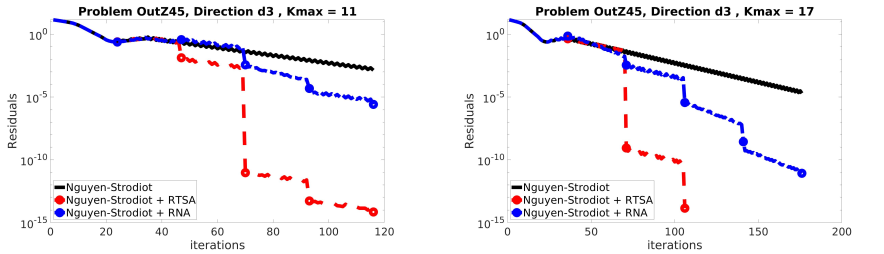
Figure 4.10: OutZ45 solved with Nguyen-Strodiot with d_k^1



(a) *Nguyen-Strodiot with best Kmax choice for RTSA*

(b) *Nguyen-Strodiot with best Kmax choice for RNA*

Figure 4.11: OutZ45 solved with Nguyen-Strodiot with d_k^2



(a) *Nguyen-Strodiot with best Kmax choice for RTSA*

(b) *Nguyen-Strodiot with best Kmax choice for RNA*

Figure 4.12: OutZ45 solved with Nguyen-Strodiot with d_k^3

4.1.4 OutZ46

Source : [26], [15]

Description :

$$\begin{aligned}
F(x) &:= \begin{pmatrix} 2 & 8/3 \\ 5/4 & 2 \end{pmatrix} x - \begin{pmatrix} 34 \\ 24.25 \end{pmatrix}, \\
g^I(y) &:= \begin{pmatrix} -1 & 0 \\ 1 & 0 \\ 0 & -1 \\ 0 & 1 \end{pmatrix} y - \begin{pmatrix} 0 \\ -10 \\ 2 \\ -10 \end{pmatrix}, \\
g^P(y,x) &:= \begin{pmatrix} 1 & 0 \end{pmatrix} y - \begin{pmatrix} 0 & 1 \end{pmatrix} x - \begin{pmatrix} 15 \end{pmatrix}.
\end{aligned}$$

Known solution : $x^* = (5,9)^T$.

For both the extrapolated methods we took (0,0) as starting point, OptimalityTolerance $= 1e-20$ and MaxIterations $= 500$.

Best parametric choice of Kmax for RTSA

We took for

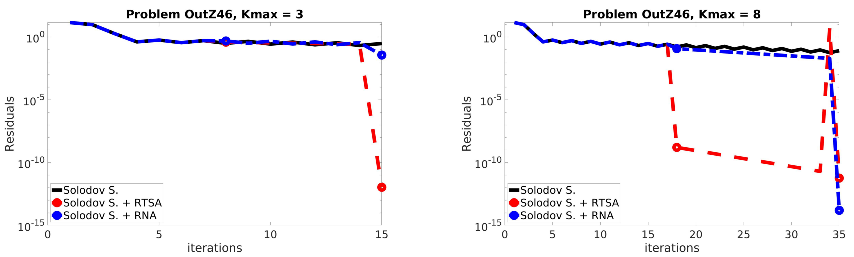
- Solodov: $Nmax = 2$, $Kmax = 3$;
- Nguyen-Strodiot:

– d_k^1: $Nmax = 5$, $Kmax = 9$;
– d_k^2: $Nmax = 4$, $Kmax = 22$;
– d_k^3: $Nmax = 3$, $Kmax = 14$;

Best parametric choice of Kmax for RNA

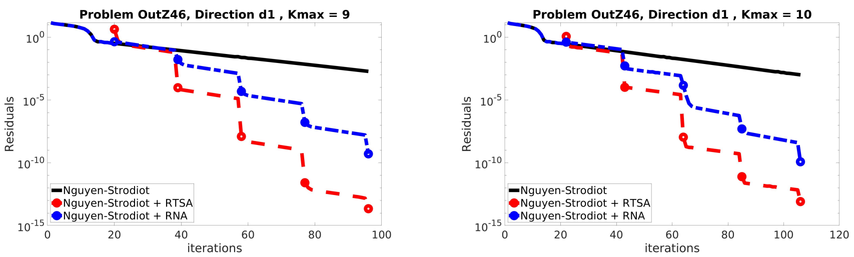
We took for

- Solodov: $Nmax = 2$, $Kmax = 8$;
- Nguyen-Strodiot:
  – d_k^1: $Nmax = 5$, $Kmax = 10$;
  – d_k^2: $Nmax = 4$, $Kmax = 21$;
  – d_k^3: $Nmax = 5$, $Kmax = 13$;



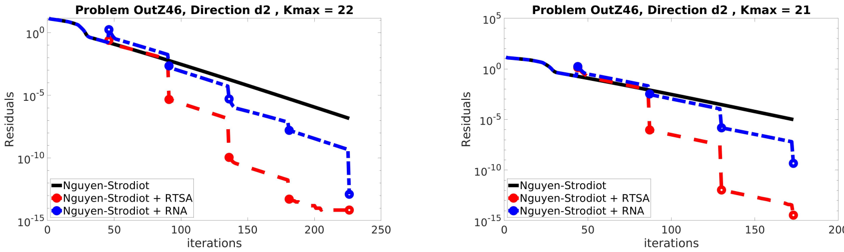
(a) *Solodov with best Kmax choice for RTSA* (b) *Solodov with best Kmax choice for RNA*

Figure 4.13: OutZ46 solved with Solodov



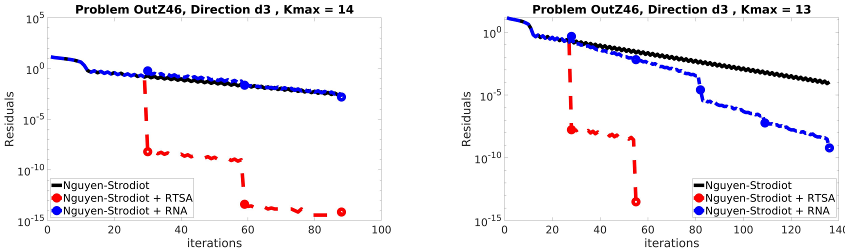
(a) *Nguyen-Strodiot with best Kmax choice for RTSA* (b) *Nguyen-Strodiot with best Kmax choice for RNA*

Figure 4.14: OutZ46 solved with Nguyen-Strodiot with d_k^1



(a) *Nguyen-Strodiot with best Kmax choice for RTSA* (b) *Nguyen-Strodiot with best Kmax choice for RNA*

Figure 4.15: OutZ46 solved with Nguyen-Strodiot with d_k^2



(a) *Nguyen-Strodiot with best Kmax choice for RTSA* (b) *Nguyen-Strodiot with best Kmax choice for RNA*

Figure 4.16: OutZ46 solved with Nguyen-Strodiot with d_k^3

4.2 RHS

In this class of problem, the feasible set $\tilde{K}(x)$ is defined by

$$g^P(y, x) := Ey - d + c(x)$$

where $E \in \mathbb{R}^{m \times n}$ is a given matrix, $c : \mathbb{R}^n \to \mathbb{R}^{m_P}$ and $d \in \mathbb{R}^{m_P}$. In this class of QVIs, the feasible set is defined by linear inequalities in which the right-hand side depends on x.

4.2.1 RHS1A1

Source : [8]

Description :

$$\begin{aligned} F(x) :=& Ax + b, \\ g^P(y) :=& Ey - d + C(\sin(x^i))_{i=1}^n \end{aligned}$$

where A, b, E, d and C are available in the corresponding Matlab functions (RHS1A1 differs from RHS1B1 only in the matrix C).

For both the extrapolated methods we took $\texttt{zeros}(200, 1)$ as starting point, $\text{OptimalityTolerance} = 1e - 20$ and $\text{MaxIterations} = 500$.

Best parametric choice of Kmax for RTSA

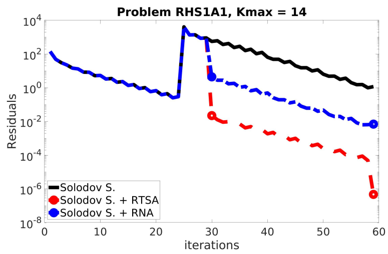
We took for

- Solodov: $Nmax = 2$, $Kmax = 14$;
- Nguyen-Strodiot:
  - d_k^1: $Nmax = 5$, $Kmax = 8$;
  - d_k^2: $Nmax = 5$, $Kmax = 7$;
  - d_k^3: $Nmax = 5$, $Kmax = 7$;

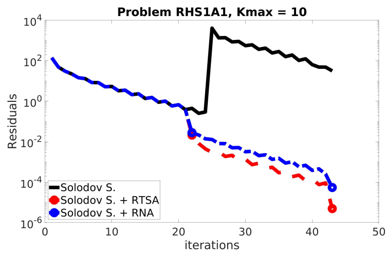
Best parametric choice of Kmax for RNA

We took for

- Solodov: $Nmax = 2$, $Kmax = 10$;
- Nguyen-Strodiot:
  - d_k^1: $Nmax = 5$, $Kmax = 12$;
  - d_k^2: $Nmax = 5$, $Kmax = 6$;
  - d_k^3: $Nmax = 5$, $Kmax = 6$;

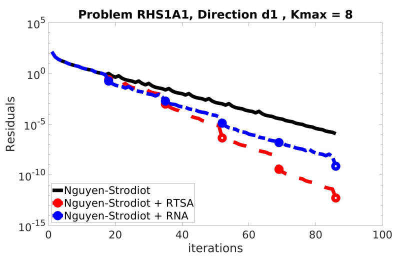


(a) *Solodov with best Kmax choice for RTSA*

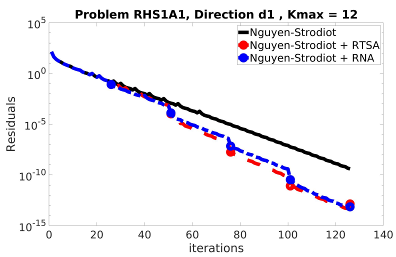


(b) *Solodov with best Kmax choice for RNA*

Figure 4.17: RHS1A1 solved with Solodov

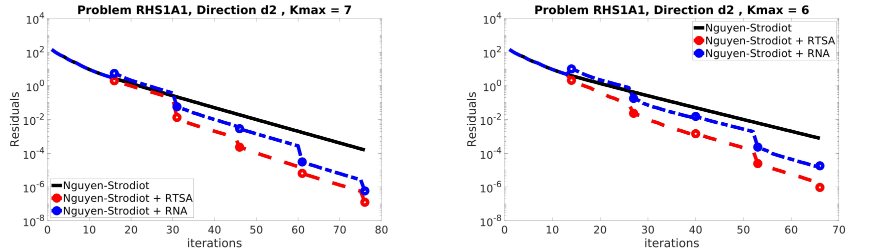


(a) *Nguyen-Strodiot with best Kmax choice for RTSA*



(b) *Nguyen-Strodiot with best Kmax choice for RNA*

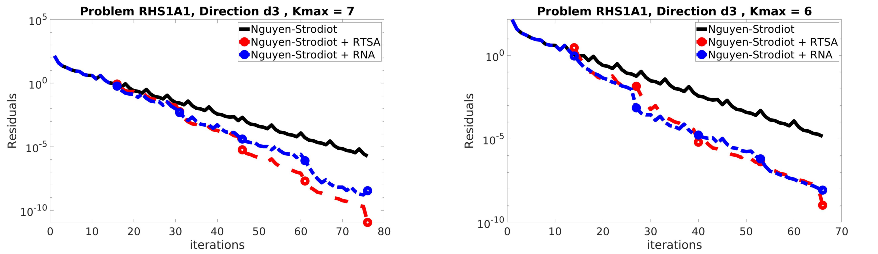
Figure 4.18: RHS1A1 solved with Nguyen-Strodiot with d_k^1



(a) *Nguyen-Strodiot with best Kmax choice for RTSA*

(b) *Nguyen-Strodiot with best Kmax choice for RNA*

Figure 4.19: RHS1A1 solved with Nguyen-Strodiot with d_k^2



(a) *Nguyen-Strodiot with best Kmax choice for RTSA*

(b) *Nguyen-Strodiot with best Kmax choice for RNA*

Figure 4.20: RHS1A1 solved with Nguyen-Strodiot with d_k^3

4.2.2 RHS1B1

Source : [8]
Description :

$$
\begin{aligned}
F(x) &:= Ax + b, \\
g^P(y) &:= Ey - d + C(\sin(x^i))_{i=1}^n
\end{aligned}
$$

where $A,\ b,\ E,\ d$ and C are available in the corresponding Matlab functions.

For both the extrapolated methods we took $\texttt{zeros}(200, 1)$ as starting point, OptimalityTolerance $= 1e - 20$ and MaxIterations $= 500$.

Best parametric choice of Kmax for RTSA

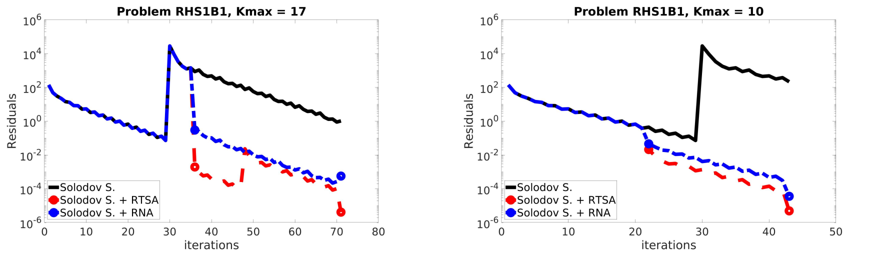
We took for

- Solodov: $Nmax = 2$, $Kmax = 17$;
- Nguyen-Strodiot:
  - d_k^1: $Nmax = 5$, $Kmax = 8$;
  - d_k^2: $Nmax = 5$, $Kmax = 7$;
  - d_k^3: $Nmax = 5$, $Kmax = 7$;

Best parametric choice of Kmax for RNA

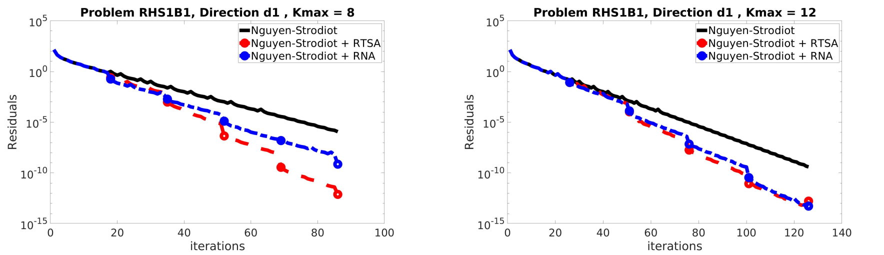
We took for

- Solodov: $Nmax = 2$, $Kmax = 10$;
- Nguyen-Strodiot:
  - d_k^1: $Nmax = 5$, $Kmax = 12$;
  - d_k^2: $Nmax = 5$, $Kmax = 6$;
  - d_k^3: $Nmax = 5$, $Kmax = 6$;



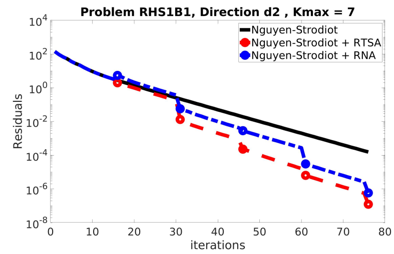
(a) *Solodov with best Kmax choice for RTSA* (b) *Solodov with best Kmax choice for RNA*

Figure 4.21: RHS1B1 solved with Solodov

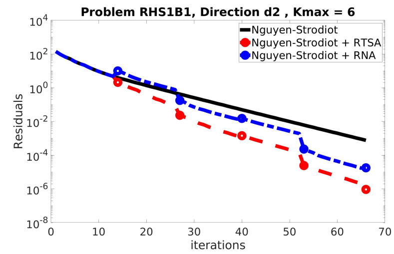


(a) *Nguyen-Strodiot with best Kmax choice for RTSA* (b) *Nguyen-Strodiot with best Kmax choice for RNA*

Figure 4.22: RHS1B1 solved with Nguyen-Strodiot with d_k^1

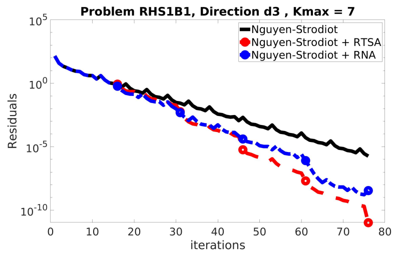


(a) *Nguyen-Strodiot with best Kmax choice for RTSA*

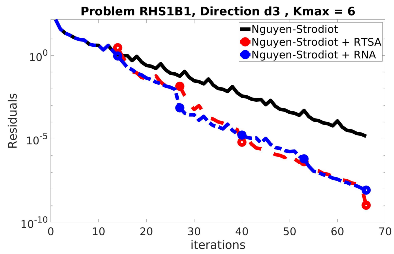


(b) *Nguyen-Strodiot with best Kmax choice for RNA*

Figure 4.23: RHS1B1 solved with Nguyen-Strodiot with d_k^2



(a) *Nguyen-Strodiot with best Kmax choice for RTSA*



(b) *Nguyen-Strodiot with best Kmax choice for RNA*

Figure 4.24: RHS1B1 solved with Nguyen-Strodiot with d_k^3

4.2.3 RHS2A1

Source : [8]
Description :

$$F(x) := Ax + b,$$
$$g^P(y) := Ey - d + Cx$$

where A, b, E, d and C are available in the corresponding Matlab functions (RHS2A1 differs from RHS2B1 only in the matrix C).

For both the extrapolated methods we took `zeros`$(200, 1)$ as starting point, OptimalityTolerance $= 1e - 20$ and MaxIterations $= 500$.

Best parametric choice of Kmax for RTSA

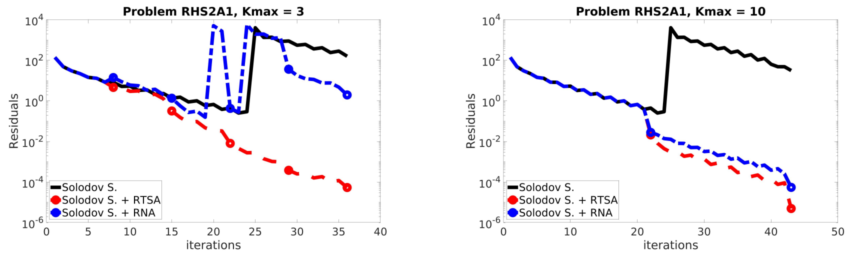
We took for

- Solodov: $Nmax = 5$, $Kmax = 3$;
- Nguyen-Strodiot:
  - d_k^1: $Nmax = 5$, $Kmax = 8$;
  - d_k^2: $Nmax = 5$, $Kmax = 7$;
  - d_k^3: $Nmax = 5$, $Kmax = 7$;

Best parametric choice of Kmax for RNA

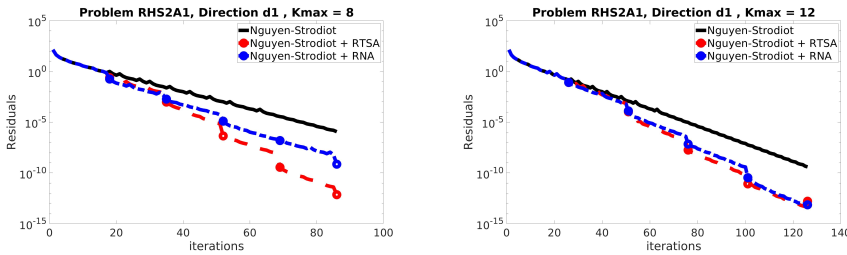
We took for

- Solodov: $Nmax = 2$, $Kmax = 10$;
- Nguyen-Strodiot:
  - d_k^1: $Nmax = 6$, $Kmax = 12$;
  - d_k^2: $Nmax = 5$, $Kmax = 6$;
  - d_k^3: $Nmax = 5$, $Kmax = 6$;



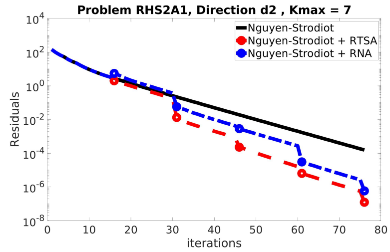
(a) *Solodov with best Kmax choice for RTSA* (b) *Solodov with best Kmax choice for RNA*

Figure 4.25: RHS2A1 solved with Solodov

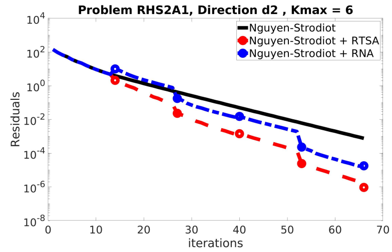


(a) *Nguyen-Strodiot with best Kmax choice for RTSA* (b) *Nguyen-Strodiot with best Kmax choice for RNA*

Figure 4.26: RHS2A1 solved with Nguyen-Strodiot with d_k^1

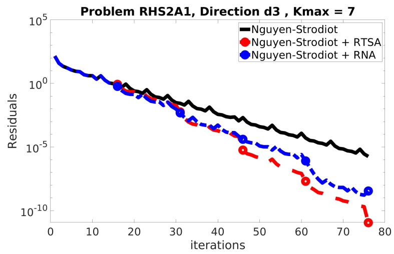


(a) *Nguyen-Strodiot with best Kmax choice for RTSA*

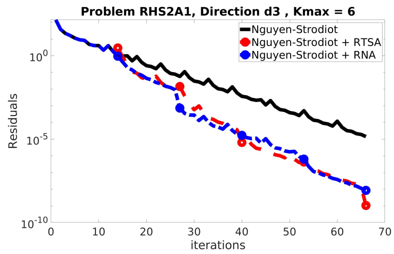


(b) *Nguyen-Strodiot with best Kmax choice for RNA*

Figure 4.27: RHS2A1 solved with Nguyen-Strodiot with d_k^2



(a) *Nguyen-Strodiot with best Kmax choice for RTSA*



(b) *Nguyen-Strodiot with best Kmax choice for RNA*

Figure 4.28: RHS2A1 solved with Nguyen-Strodiot with d_k^3

4.2.4 RHS2B1

Source : [8]
Description :

$$F(x) := Ax + b,$$
$$g^P(y) := Ey - d + Cx$$

where $A,\ b,\ E,\ d$ and C are available in the corresponding Matlab functions.

For both the extrapolated methods we took `zeros`$(200, 1)$ as starting point, OptimalityTolerance $= 1e-20$ and MaxIterations $= 500$.

Best parametric choice of Kmax for RTSA

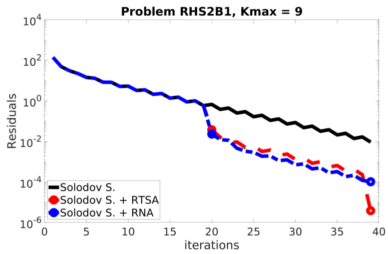
We took for

- Solodov: $Nmax = 2$, $Kmax = 9$;
- Nguyen-Strodiot:
  - d_k^1: $Nmax = 5$, $Kmax = 8$;
  - d_k^2: $Nmax = 5$, $Kmax = 7$;
  - d_k^3: $Nmax = 5$, $Kmax = 7$;

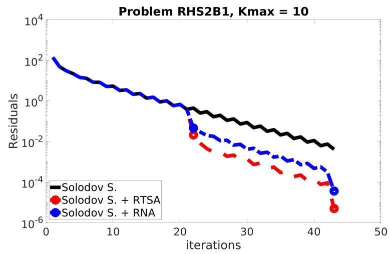
Best parametric choice of Kmax for RNA

We took for

- Solodov: $Nmax = 2$, $Kmax = 10$;
- Nguyen-Strodiot:
  - d_k^1: $Nmax = 5$, $Kmax = 12$;
  - d_k^2: $Nmax = 5$, $Kmax = 6$;
  - d_k^3: $Nmax = 5$, $Kmax = 6$;

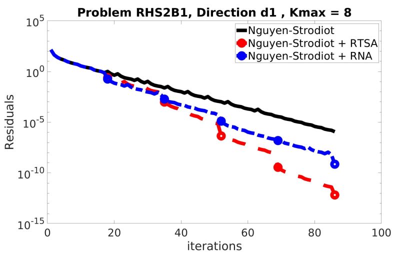


(a) *Solodov with best Kmax choice for RTSA*

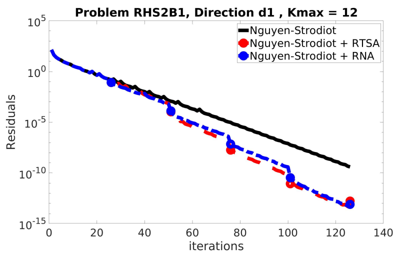


(b) *Solodov with best Kmax choice for RNA*

Figure 4.29: RHS2B1 solved with Solodov

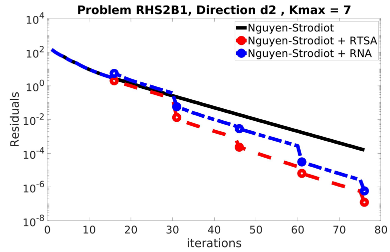


(a) *Nguyen-Strodiot with best Kmax choice for RTSA*

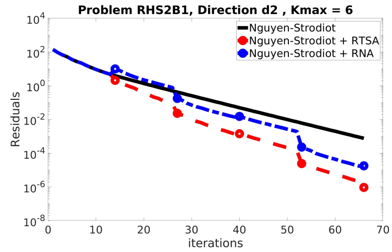


(b) *Nguyen-Strodiot with best Kmax choice for RNA*

Figure 4.30: RHS2B1 solved with Nguyen-Strodiot with d_k^1

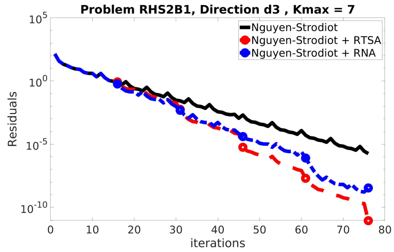


(a) *Nguyen-Strodiot with best Kmax choice for RTSA*

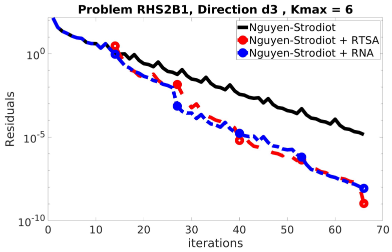


(b) *Nguyen-Strodiot with best Kmax choice for RNA*

Figure 4.31: RHS2B1 solved with Nguyen-Strodiot with d_k^2



(a) *Nguyen-Strodiot with best Kmax choice for RTSA*



(b) *Nguyen-Strodiot with best Kmax choice for RNA*

Figure 4.32: RHS2B1 solved with Nguyen-Strodiot with d_k^3

4.3 Example

Description :

$$
\begin{aligned}
F_i(x) &:= c_i + \left(\frac{x_i}{\tau}\right)^{1/\beta_i} + \left(\frac{5000}{Q}\right)^{1/\eta}\left(\frac{x_i}{\eta Q} - 1\right) \quad i = 1, \dots, nVar, \\
g^I(y) &:= [-\texttt{eye}(nVar); \texttt{eye}(nVar)]y + \texttt{zeros}(2 * nVar, nVar)x - \\
&\quad [-\texttt{ones}(nVar, 1); 150 * \texttt{ones}(nVar, 1)], \\
g^P(y, x) &:= \texttt{eye}(nVar)y + [\texttt{ones}(nVar) - \texttt{eye}(nVar)]x - 700 * \texttt{ones}(nVar, 1).
\end{aligned}
$$

where $nVar$ is the number of variables, $Q = \sum_{1 \le i \le nVar} x^i$, the coefficients $c(j) := 12 - 2 * j$ and $b(j) := 1.3 - j * 0.1$ for $j = 1, \dots, nVar$, $\tau = 5$ and $\eta = 1.1$.

4.3.1 Number of variables: 5

Source : [26], [15]

Known solution :

$$x^* \approx (36.9325; 41.8181; 43.7066; 42.6592; 39.1790)$$

.

For both the extrapolated methods we took $(10; 10; 10; 10; 10)$ as starting point, OptimalityTolerance $= 1e - 10$ and MaxIterations $= 500$.

Best parametric choice of Kmax for RTSA

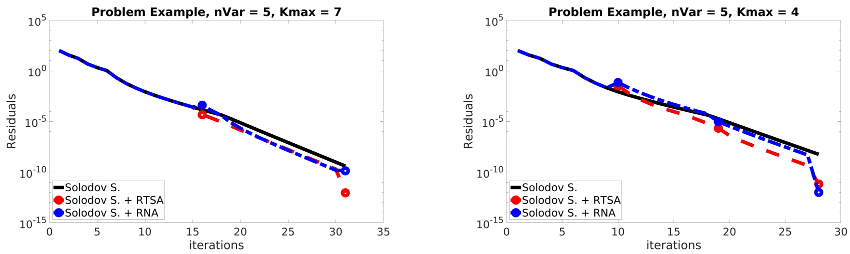
We took for

- Solodov: $Nmax = 2$, $Kmax = 7$;
- Nguyen-Strodiot:
  - d_k^1: $Nmax = 5$, $Kmax = 5$;
  - d_k^2: $Nmax = 5$, $Kmax = 6$;
  - d_k^3: $Nmax = 5$, $Kmax = 4$;

Best parametric choice of Kmax for RNA

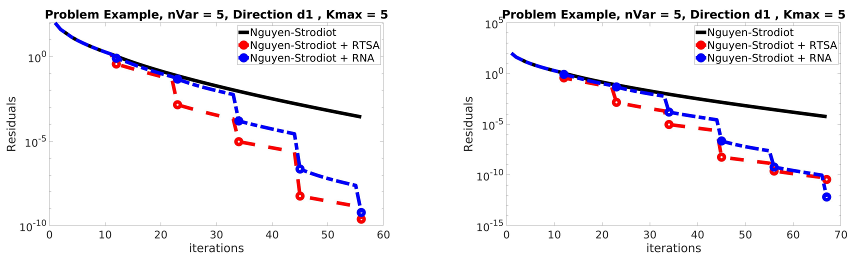
We took for

- Solodov: $Nmax = 3$, $Kmax = 4$;
- Nguyen-Strodiot:
  - d_k^1: $Nmax = 6$, $Kmax = 5$;
  - d_k^2: $Nmax = 5$, $Kmax = 7$;
  - d_k^3: $Nmax = 5$, $Kmax = 5$;



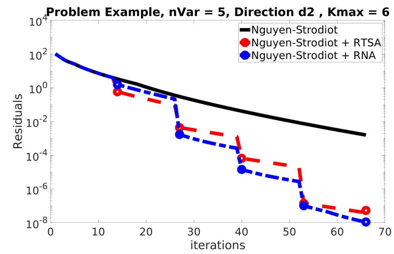
(a) *Solodov with best Kmax choice for RTSA* (b) *Solodov with best Kmax choice for RNA*

Figure 4.33: Example of dimension 5 solved with Solodov

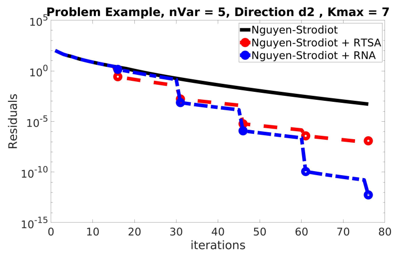


(a) *Nguyen-Strodiot with best Kmax choice for RTSA* (b) *Nguyen-Strodiot with best Kmax choice for RNA*

Figure 4.34: Example of dimension 5 solved with Nguyen-Strodiot with d_k^1

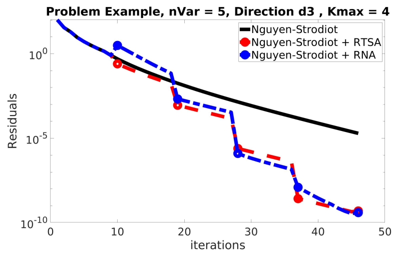


(a) *Nguyen-Strodiot with best Kmax choice for RTSA*

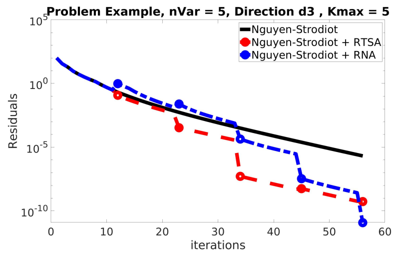


(b) *Nguyen-Strodiot with best Kmax choice for RNA*

Figure 4.35: Example of dimension 5 solved with Nguyen-Strodiot with d_k^2



(a) *Nguyen-Strodiot with best Kmax choice for RTSA*



(b) *Nguyen-Strodiot with best Kmax choice for RNA*

Figure 4.36: Example of dimension 5 solved with Nguyen-Strodiot with d_k^3

4.3.2 Number of variables: 6

Known solution :

$$x^* \approx (32.3187; 38.0902; 40.7454; 40.3477; 37.4245; 32.8182).$$

For both the extrapolated methods we took (10; 10; 10; 10; 10; 10) as starting point, OptimalityTolerance $= 1e - 10$ and MaxIterations $= 500$.

Best parametric choice of Kmax for RTSA

We took for

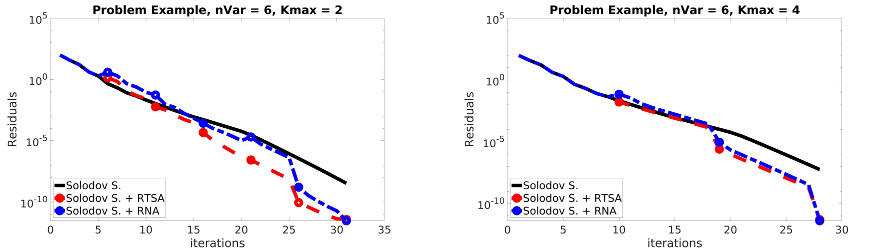
- Solodov: $Nmax = 6$, $Kmax = 2$;
- Nguyen-Strodiot:
  - d_k^1: $Nmax = 5$, $Kmax = 5$;
  - d_k^2: $Nmax = 5$, $Kmax = 10$;
  - d_k^3: $Nmax = 5$, $Kmax = 4$;

Best parametric choice of Kmax for RNA

We took for

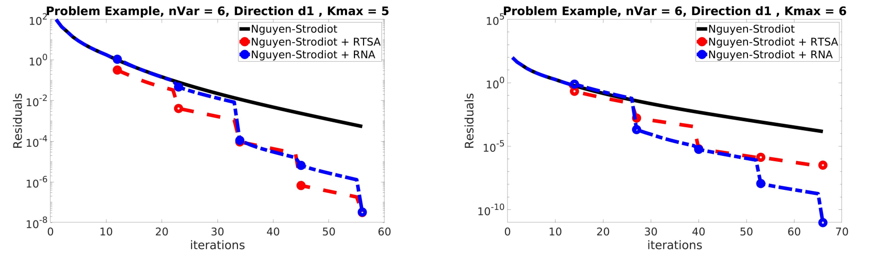
- Solodov: $Nmax = 3$, $Kmax = 4$;

- Nguyen-Strodiot:
  - d_k^1: $Nmax = 5$, $Kmax = 6$;
  - d_k^2: $Nmax = 5$, $Kmax = 7$;
  - d_k^3: $Nmax = 5$, $Kmax = 6$;



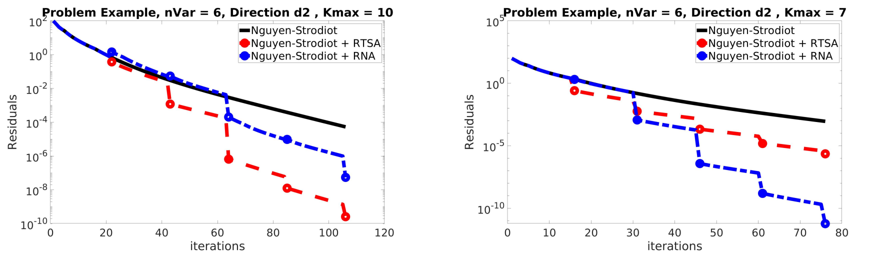
(a) *Solodov with best Kmax choice for RTSA* (b) *Solodov with best Kmax choice for RNA*

Figure 4.37: Example of dimension 6 solved with Solodov



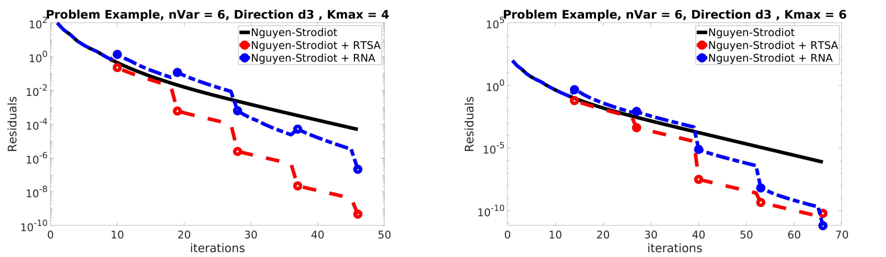
(a) *Nguyen-Strodiot with best Kmax choice for RTSA* (b) *Nguyen-Strodiot with best Kmax choice for RNA*

Figure 4.38: Example of dimension 6 solved with Nguyen-Strodiot with d_k^1



(a) *Nguyen-Strodiot with best Kmax choice for RTSA* (b) *Nguyen-Strodiot with best Kmax choice for RNA*

Figure 4.39: Example of dimension 6 solved with Nguyen-Strodiot with d_k^2



(a) *Nguyen-Strodiot with best Kmax choice for RTSA*

(b) *Nguyen-Strodiot with best Kmax choice for RNA*

Figure 4.40: Example of dimension 6 solved with Nguyen-Strodiot with d_k^3

4.3.3 Number of variables: 7

Known solution :

$$x^* \approx (28.7158; 35.1727; 38.4430; 38.5727; 36.0974; 31.8672; 26.7946).$$

For both the extrapolated methods we took $(10; 10; 10; 10; 10; 10; 10)$ as starting point, OptimalityTolerance $= 1e - 10$ and MaxIterations $= 500$.

Best parametric choice of Kmax for RTSA

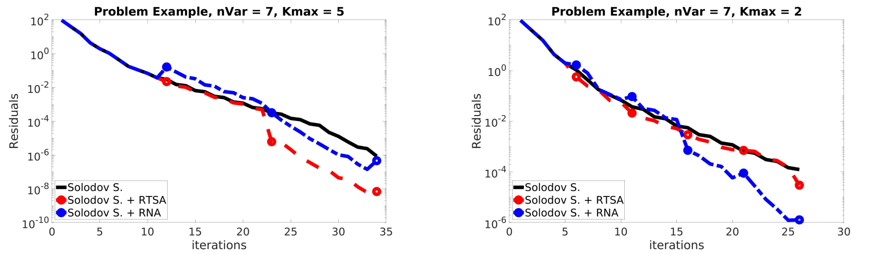
We took for

- Solodov: $Nmax = 3$, $Kmax = 5$;
- Nguyen-Strodiot:
  - d_k^1: $Nmax = 5$, $Kmax = 4$;
  - d_k^2: $Nmax = 5$, $Kmax = 11$;
  - d_k^3: $Nmax = 5$, $Kmax = 5$;

Best parametric choice of Kmax for RNA

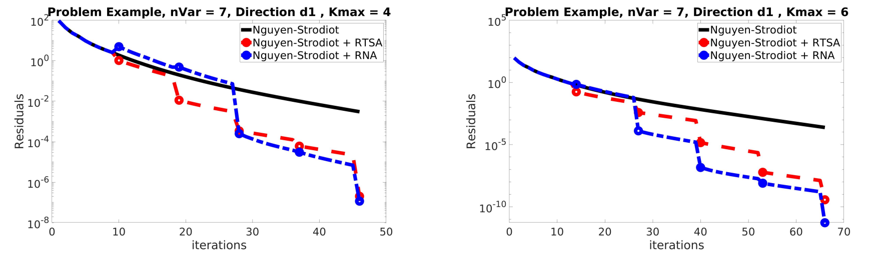
We took for

- Solodov: $Nmax = 5$, $Kmax = 2$;
- Nguyen-Strodiot:
  - d_k^1: $Nmax = 5$, $Kmax = 6$;
  - d_k^2: $Nmax = 5$, $Kmax = 12$;
  - d_k^3: $Nmax = 5$, $Kmax = 6$;



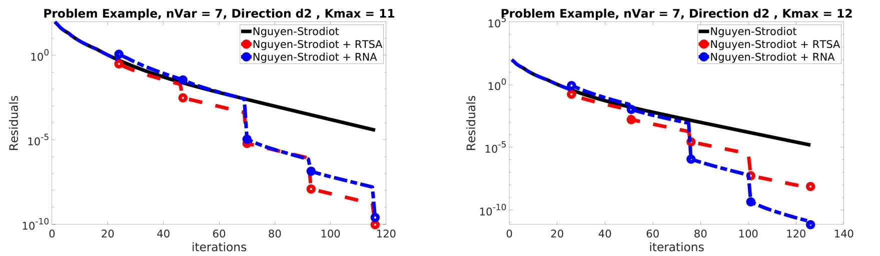
(a) *Solodov with best Kmax choice for RTSA* (b) *Solodov with best Kmax choice for RNA*

Figure 4.41: Example of dimension 7 solved with Solodov



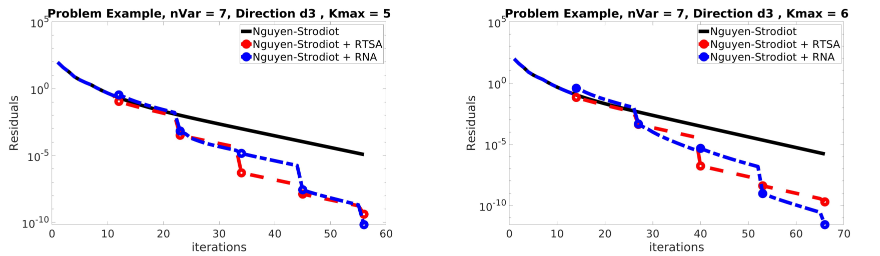
(a) *Nguyen-Strodiot with best Kmax choice for RTSA* (b) *Nguyen-Strodiot with best Kmax choice for RNA*

Figure 4.42: Example of dimension 7 solved with Nguyen-Strodiot with d_k^1



(a) *Nguyen-Strodiot with best Kmax choice for RTSA* (b) *Nguyen-Strodiot with best Kmax choice for RNA*

Figure 4.43: Example of dimension 7 solved with Nguyen-Strodiot with d_k^2



(a) *Nguyen-Strodiot with best Kmax choice for RTSA* (b) *Nguyen-Strodiot with best Kmax choice for RNA*

Figure 4.44: Example of dimension 7 solved with Nguyen-Strodiot with d_k^3

4.3.4 Number of variables: 8

Known solution :

$$x^* \approx (25.9498; 32.9243; 36.6743; 37.2193; 31.1551; 26.3144; 21.3346)$$

For both the extrapolated methods we took $(10; 10; 10; 10; 10; 10; 10; 10)$ as starting point, OptimalityTolerance $= 1e - 10$ and MaxIterations $= 500$.

Best parametric choice of Kmax for RTSA

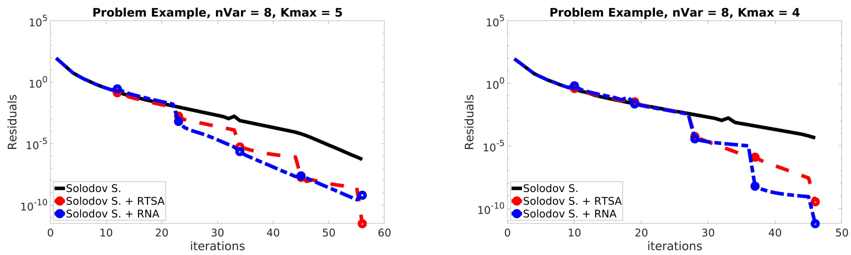
We took for

- Solodov: $Nmax = 5$, $Kmax = 5$;
- Nguyen-Strodiot:
  - d_k^1: $Nmax = 5$, $Kmax = 4$;
  - d_k^2: $Nmax = 5$, $Kmax = 6$;
  - d_k^3: $Nmax = 5$, $Kmax = 4$;

Best parametric choice of Kmax for RNA

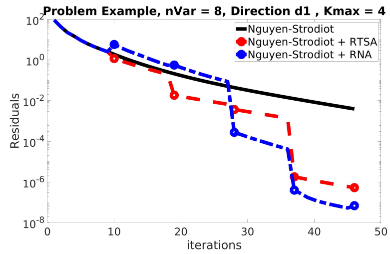
We took for

- Solodov: $Nmax = 5$, $Kmax = 4$;
- Nguyen-Strodiot:
  - d_k^1: $Nmax = 5$, $Kmax = 6$;
  - d_k^2: $Nmax = 5$, $Kmax = 11$;
  - d_k^3: $Nmax = 5$, $Kmax = 3$;

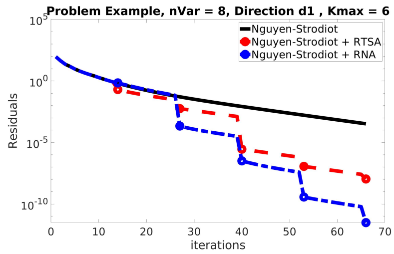


(a) *Solodov with best Kmax choice for RTSA* (b) *Solodov with best Kmax choice for RNA*

Figure 4.45: Example of dimension 8 solved with Solodov

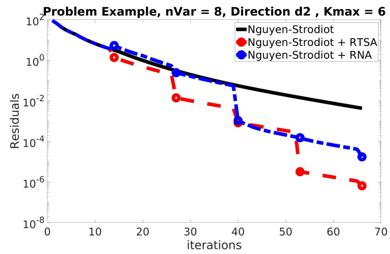


(a) *Nguyen-Strodiot with best Kmax choice for RTSA*

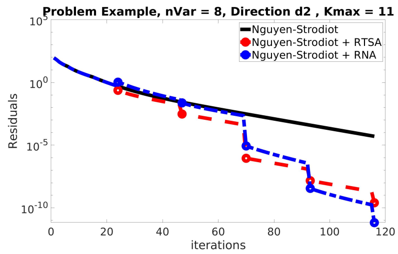


(b) *Nguyen-Strodiot with best Kmax choice for RNA*

Figure 4.46: Example of dimension 8 solved with Nguyen-Strodiot with d_k^1

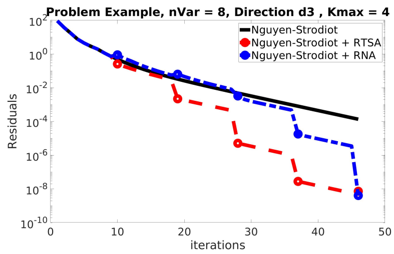


(a) *Nguyen-Strodiot with best Kmax choice for RTSA*

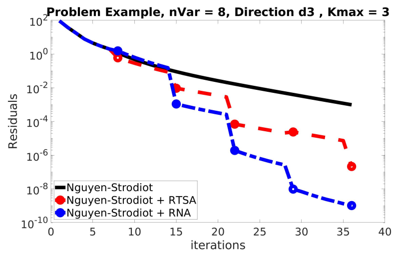


(b) *Nguyen-Strodiot with best Kmax choice for RNA*

Figure 4.47: Example of dimension 8 solved with Nguyen-Strodiot with d_k^2



(a) *Nguyen-Strodiot with best Kmax choice for RTSA*



(b) *Nguyen-Strodiot with best Kmax choice for RNA*

Figure 4.48: Example of dimension 8 solved with Nguyen-Strodiot with d_k^3

4.3.5 Number of variables: 9

Known solution :

$$x^* \approx (23.8581; 31.2167; 35.3330; 36.1976; 34.3426; 30.6240; 25.9580; \ldots$$
$$21.1092; 16.5936).$$

For both the extrapolated methods we took $10 * \texttt{ones}(9,1)$ as starting point, OptimalityTolerance $= 1e - 10$ and MaxIterations $= 500$.

Best parametric choice of Kmax for RTSA

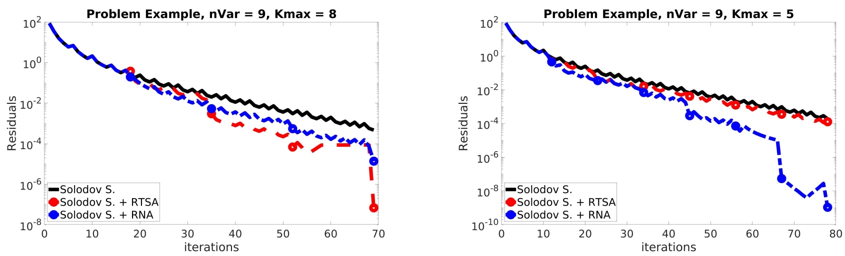
We took for

- Solodov: $Nmax = 4$, $Kmax = 8$;
- Nguyen-Strodiot:
  - d_k^1: $Nmax = 5$, $Kmax = 8$;
  - d_k^2: $Nmax = 5$, $Kmax = 7$;
  - d_k^3: $Nmax = 5$, $Kmax = 10$;

Best parametric choice of Kmax for RNA

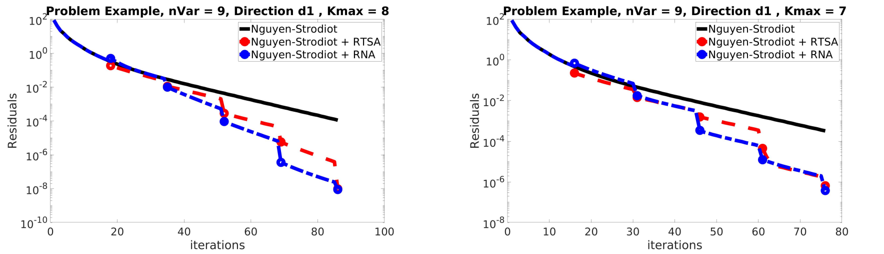
We took for

- Solodov: $Nmax = 7$, $Kmax = 5$;
- Nguyen-Strodiot:
  - d_k^1: $Nmax = 5$, $Kmax = 7$;
  - d_k^2: $Nmax = 5$, $Kmax = 9$;
  - d_k^3: $Nmax = 5$, $Kmax = 12$;



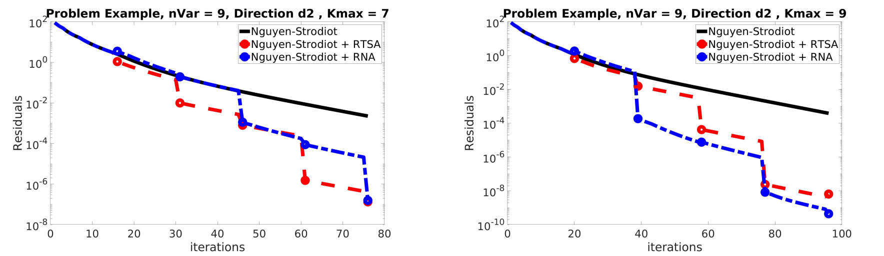
(a) *Solodov with best Kmax choice for RTSA* (b) *Solodov with best Kmax choice for RNA*

Figure 4.49: Example of dimension 9 solved with Solodov



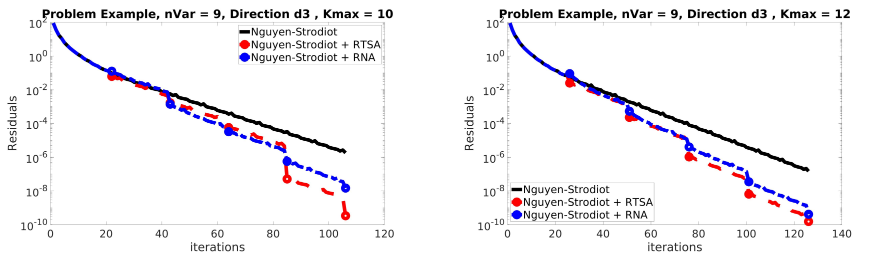
(a) *Nguyen-Strodiot with best Kmax choice for RTSA*
(b) *Nguyen-Strodiot with best Kmax choice for RNA*

Figure 4.50: Example of dimension 9 solved with Nguyen-Strodiot with d_k^1



(a) *Nguyen-Strodiot with best Kmax choice for RTSA*
(b) *Nguyen-Strodiot with best Kmax choice for RNA*

Figure 4.51: Example of dimension 9 solved with Nguyen-Strodiot with d_k^2



(a) *Nguyen-Strodiot with best Kmax choice for RTSA*
(b) *Nguyen-Strodiot with best Kmax choice for RNA*

Figure 4.52: Example of dimension 9 solved with Nguyen-Strodiot with d_k^3

4.3.6 Number of variables: 10

Known solution :

$$x^* \approx (22.2991; 29.9385; 34.3294; 35.4355; 33.7836; 30.2309; 25.6951; \ldots$$
$$20.9433; 16.4959; 12.6327).$$

For both the extrapolated methods we took $10 * \texttt{ones}(10, 1)$ as starting point, OptimalityTolerance $= 1e - 10$ and MaxIterations $= 500$.

Best parametric choice of Kmax for RTSA

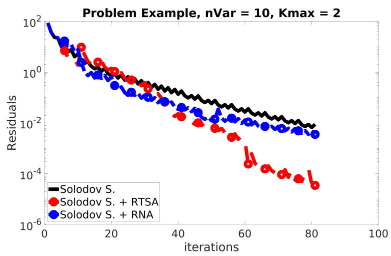
We took for

- Solodov: $Nmax = 16$, $Kmax = 2$;
- Nguyen-Strodiot:
  - d_k^1: $Nmax = 5$, $Kmax = 9$;
  - d_k^2: $Nmax = 5$, $Kmax = 13$;
  - d_k^3: $Nmax = 5$, $Kmax = 11$;

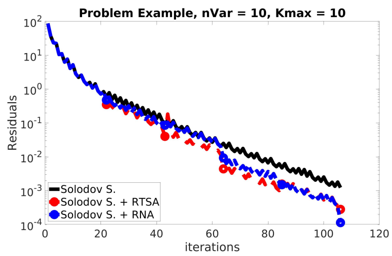
Best parametric choice of Kmax for RNA

We took for

- Solodov: $Nmax = 5$, $Kmax = 10$;
- Nguyen-Strodiot:
  - d_k^1: $Nmax = 5$, $Kmax = 6$;
  - d_k^2: $Nmax = 5$, $Kmax = 12$;
  - d_k^3: $Nmax = 5$, $Kmax = 23$;

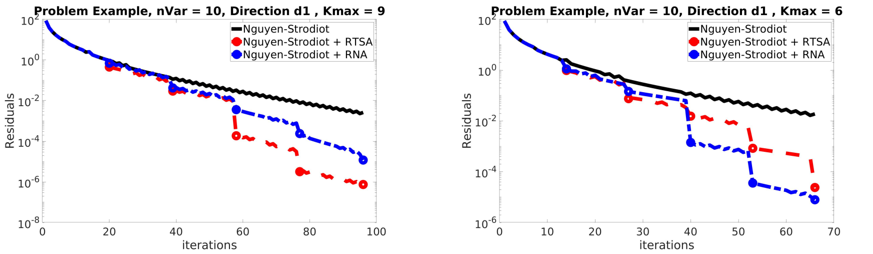


(a) *Solodov with best Kmax choice for RTSA*



(b) *Solodov with best Kmax choice for RNA*

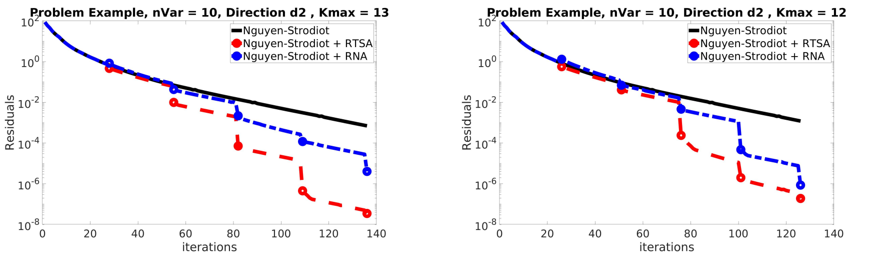
Figure 4.53: Example of dimension 10 solve with Solodov



(a) *Nguyen-Strodiot with best Kmax choice for RTSA*

(b) *Nguyen-Strodiot with best Kmax choice for RNA*

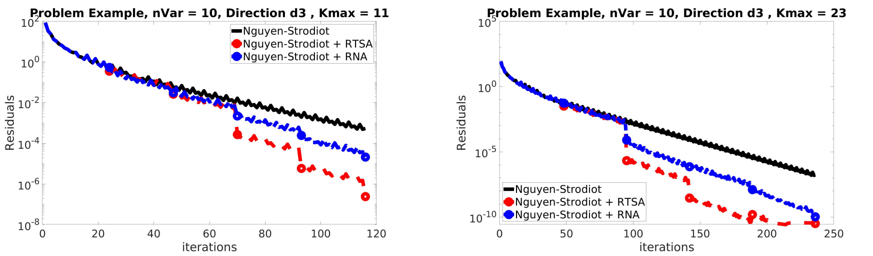
Figure 4.54: Example of dimension 10 solve with Nguyen-Strodiot with d_k^1



(a) *Nguyen-Strodiot with best Kmax choice for RTSA*

(b) *Nguyen-Strodiot with best Kmax choice for RNA*

Figure 4.55: Example of dimension 10 solve with Nguyen-Strodiot with d_k^2



(a) *Nguyen-Strodiot with best Kmax choice for RTSA*

(b) *Nguyen-Strodiot with best Kmax choice for RNA*

Figure 4.56: Example of dimension 10 solve with Nguyen-Strodiot with d_k^3

4.3.7 Number of variables: 11

Known solution :

$$x^* \approx (21.1564; 28.9931; 33.5888; 34.8733; 33.3734; 29.9428; 25.5029; \ldots$$
$$20.8223; 16.4248; 12.5945; 9.4331)$$

For both the extrapolated methods we took $10*\texttt{ones}(11, 1)$ as starting point, OptimalityTolerance $= 1e - 10$ and MaxIterations $= 500$.

Best parametric choice of Kmax for RTSA

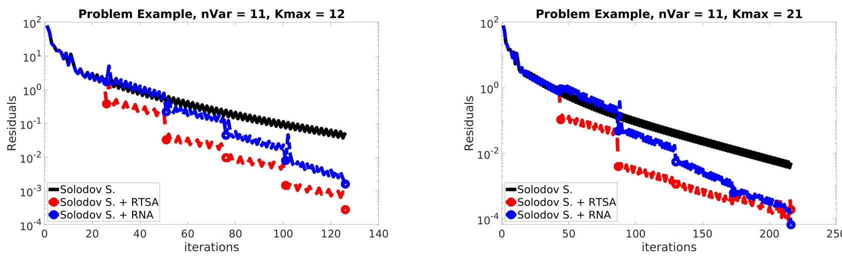
We took for

- Solodov: $Nmax = 5$, $Kmax = 12$;
- Nguyen-Strodiot:
  - d_k^1: $Nmax = 5$, $Kmax = 17$;
  - d_k^2: $Nmax = 5$, $Kmax = 14$;
  - d_k^3: $Nmax = 5$, $Kmax = 11$;

Best parametric choice of Kmax for RNA

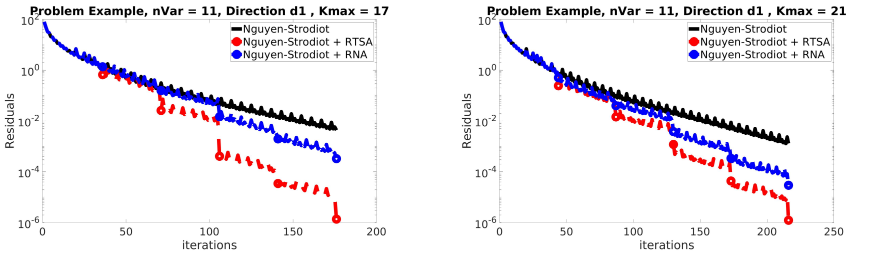
We took for

- Solodov: $Nmax = 5$, $Kmax = 21$;
- Nguyen-Strodiot:
  - d_k^1: $Nmax = 5$, $Kmax = 21$;
  - d_k^2: $Nmax = 5$, $Kmax = 31$;
  - d_k^3: $Nmax = 5$, $Kmax = 33$;



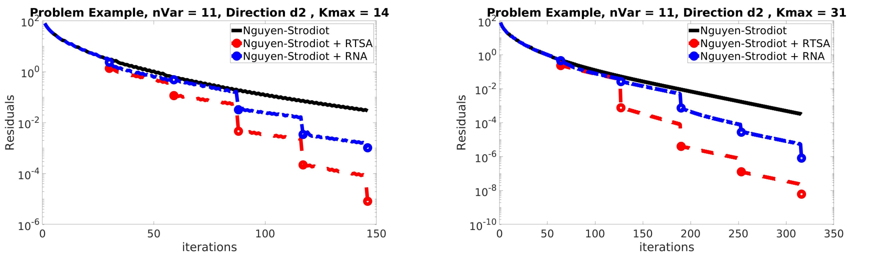
(a) *Solodov with best Kmax choice for RTSA* (b) *Solodov with best Kmax choice for RNA*

Figure 4.57: Example of dimension 11 solve with Solodov



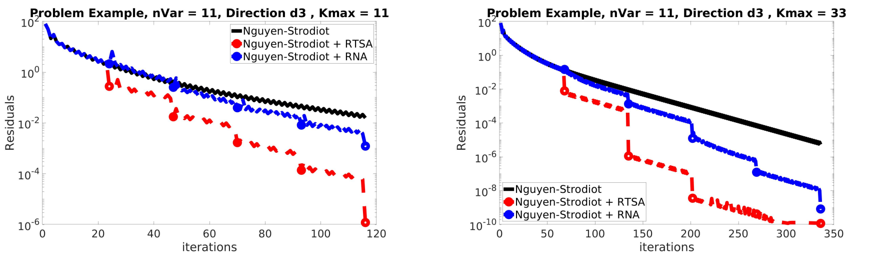
(a) *Nguyen-Strodiot with best Kmax choice for RTSA* (b) *Nguyen-Strodiot with best Kmax choice for RNA*

Figure 4.58: Example of dimension 11 solve with Nguyen-Strodiot with d_k^1



(a) *Nguyen-Strodiot with best Kmax choice for RTSA* (b) *Nguyen-Strodiot with best Kmax choice for RNA*

Figure 4.59: Example of dimension 11 solve with Nguyen-Strodiot with d_k^2



(a) *Nguyen-Strodiot with best Kmax choice for RTSA* (b) *Nguyen-Strodiot with best Kmax choice for RNA*

Figure 4.60: Example of dimension 11 solve with Nguyen-Strodiot with d_k^3

4.4 Concluding Remarks

In this thesis we presented a numerical study for the behavior of two hybrid extragradient methods, namely Generalized Solodov and Nguyen-Strodiot, and of two different type of regularized accelerations, namely the Regularized Nonlinear Acceleration (RNA) and the Regularized Topological Shanks Acceleration (RTSA).

It can be observed that on the one side, Nguyen-Strodiot has a low rate of convergence if compered to Solodov method but, on the other side it is more robust, i.e., it works on a larger number of test problems. An example is the family of RHS test problems where the Nguyen-Strodiot method converges while the Solodov one does not. Therefore Nguyen-Strodiot can be considered a better choice in practice.

Regarding the performance between RTSA and RNA we can observe that RTSA provides robust accelerations performance with respect to the choice of the $Kmax$ parameter, while RNA needs specific $Kmax$ values in order to have good results. Furthermore, we observed that, usually, RNA requires a greater value of $Kmax$ to exhibit a good acceleration performance.

We can conclude that on the problem set we considered, RTSA behaves better than RNA. Moreover, to conclude, we found that the coupling on Nguyen-Strodiot with RTSA strongly improves the robustness and effectiveness of the latter method.

Chapter 5

Matlab Codes

In this chapter we show the MATLAB codes that we have written in order to do the numerical experiments.

5.1 QVILIB quadprog

In this section there is the library of MATLAB codes for the test QVI problems used for the numerical examples.

5.1.1 OutZ

Listing 5.1: OutZ40new.m

```
function out = OutZ40_new(i,x)
% QVILIB test problem OutZ40 [LBB/A/2-4-0-2-0]
% From: QVILIB: A LIBRARY OF QUASI-VARIATIONAL INEQUALITY
%                TEST PROBLEMS
% Authors: Facchinei F., Kanzow C., Sagratella S.
%
% Input arguments:
%     i: function flag;
%        it must be an integer between 0 and 7
%     x: input vector of dimension (nVar,1)
%
% Description: <QVI name> = OutZ40_new
%
%   <QVI name>(0) initializes nVar (= number of
%                 variables), nIneq (= number of
%                 inequality constraints), nEq (= number
%                 of equality constraints), nIneqInd
%                 (= number of inequality constraints
%                 that do not depend on x), nEqInd
%                 (= number of equality constraints that
%                 do not depend on x), and the data
```

```
%                    defining the problem which are used
%                    when invoking <QVI name> with other
%                    flags; it must be the first <QVI name>
%                    function call and should be called
%                    only one time
%
%      out = <QVI name>(1,x) returns a vector of dimension
%                           (nVar,1) containing F(x)
%
%      out = <QVI name>(2) returns a vector of dimensions
%                          (nVar,1) containing the lower
%                          bounds for the variable x
%
%      out = <QVI name>(3) returns a vector of dimensions
%                          (nVar,1) containing the upper
%                          bounds for the variable x
%
%      out = <QVI name>(4) returns a sparse matrix of
%                          dimensions (nIneq-nIneqInd,nVar)
%                          containing A of Ay <= b(x);
%
%      out = <QVI name>(5,x)  returns a vector of
%                            dimensions (nIneq-nIneqInd,nVar)
%                            containing b(x)of Ay <= b(x);
%
%      out = <QVI name>(6) returns a sparse matrix of
%                          dimensions (nEq,nVar)
%                          containing M of My = v(x);
%                          the first nEqInd rows refer to the
%                          those constraints that do not
%                          depend on x
%
%      out = <QVI name>(7,x) returns a vector of dimension
%                           (nEq,1) containing v(x) of
%                           My = v(x);
%                           the first nEqInd components refer
%                           to the those constraints that do
%                           not depend on x
%
%
% Problem definition

global nVar nIneq nEq nIneqInd nEqInd QVItestA QVItestb;

switch i

case 0
    nVar = 2;
    nIneq = 6;
    nEq = 0;
```

```
    nIneqInd = 4;
    nEqInd = 0;

    QVItestA = sparse([2 8/3; 5/4 2]);
    QVItestb = [-34; -24.25];

case 1
    % Function F
    out = QVItestA*x + QVItestb;

case 2
    % Bound constrains [l,u]: lower bound l
    out = zeros(nVar,1);

case 3
    % Bound constrains [l,u]: upper bound u
    out = 11*ones(nVar,1);

case 4
    % Linear constraints Ay<=b: matrix A
    out = eye(nVar);

case 5
    % Linear constraints Ay<=b: known term b
    out = 15*ones(nVar,1)-(ones(nVar)-eye(nVar))*x;

case 6
    % Equalities constraints My=v: matrix M
    out = [];

case 7
    % Equalities constraints My=v: known term v
    out = [];

end


return
```

Listing 5.2: OutZ41new.m

```
function out = OutZ41_new(i,x)
% QVILIB test problem OutZ41 [LBB-A-2-4-0-2-0]
% From: QVILIB: A LIBRARY OF QUASI-VARIATIONAL INEQUALITY
%                  TEST PROBLEMS
% Authors: Facchinei F., Kanzow C., Sagratella S.
%
% Input arguments:
%      i: function flag;
```

```
%        it must be an integer between 0 and 7
%     x: input vector of dimension (nVar,1)
%
% Description: <QVI name> = OutZ41_new
%
%    <QVI name>(0) initializes nVar (= number of
%                  variables), nIneq (= number of
%                  inequality constraints), nEq (= number
%                  of equality constraints), nIneqInd
%                  (= number of inequality constraints
%                  that do not depend on x), nEqInd
%                  (= number of equality constraints that
%                  do not depend on x), and the data
%                  defining the problem which are used
%                  when invoking <QVI name> with other
%                  flags; it must be the first <QVI name>
%                  function call and should be called
%                  only one time
%
%     out = <QVI name>(1,x) returns a vector of dimension
%                           (nVar,1) containing F(x)
%
%     out = <QVI name>(2) returns a vector of dimensions
%                         (nVar,1) containing the lower
%                         bounds for the variable x
%
%     out = <QVI name>(3) returns a vector of dimensions
%                         (nVar,1) containing the upper
%                         bounds for the variable x
%
%     out = <QVI name>(4) returns a sparse matrix of
%                         dimensions (nIneq-nIneqInd,nVar)
%                         containing A of Ay <= b(x);
%
%     out = <QVI name>(5,x)  returns a vector of
%                            dimensions (nIneq-nIneqInd,nVar)
%                            containing b(x)of Ay <= b(x);
%
%     out = <QVI name>(6) returns a sparse matrix of
%                         dimensions (nEq,nVar)
%                         containing M of My = v(x);
%                         the first nEqInd rows refer to the
%                         those constraints that do not
%                         depend on x
%
%     out = <QVI name>(7,x) returns a vector of dimension
%                           (nEq,1) containing v(x) of
%                           My = v(x);
%                           the first nEqInd components refer
%                           to the those constraints that do
```

```
%                             not depend on x
%
%
% Problem definition

global nVar nIneq nEq nIneqInd nEqInd QVItestA QVItestb;

switch i

case 0
    nVar = 2;
    nIneq = 6;
    nEq = 0;
    nIneqInd = 4;
    nEqInd = 0;

    QVItestA = sparse([2 8/3; 5/4 2]);
    QVItestb = [-100/3; -22.5];

case 1
    % Function F
    out = QVItestA*x + QVItestb;

case 2
    % Bound constrains [l,u]: lower bound l
    out = zeros(nVar,1);

case 3
    % Bound constrains [l,u]: upper bound u
    out = 11*ones(nVar,1);

case 4
    % Linear constraints Ay<=b: matrix A
    out = eye(nVar);

case 5
    % Linear constraints Ay<=b: known term b
    out = [15; 20]-(ones(nVar)-eye(nVar))*x;

case 6
    % Equalities constraints My=v: matrix M
    out = [];

case 7
    % Equalities constraints My=v: known term v
    out = [];

end
```

```
return
```

Listing 5.3: OutZ45new.m

```
function out = OutZ45_new(i,x)
% QVILIB test problem OutZ45 [LBB/A/2-4-0-2-0]
% From: QVILIB: A LIBRARY OF QUASI-VARIATIONAL INEQUALITY
%                 TEST PROBLEMS
% Authors: Facchinei F., Kanzow C., Sagratella S.
%
% Input arguments:
%      i: function flag;
%         it must be an integer between 0 and 7
%      x: input vector of dimension (nVar,1)
%
% Description: <QVI name> = OutZ45_new
%
%    <QVI name>(0) initializes nVar (= number of
%                  variables), nIneq (= number of
%                  inequality constraints), nEq (= number
%                  of equality constraints), nIneqInd
%                  (= number of inequality constraints
%                  that do not depend on x), nEqInd
%                  (= number of equality constraints that
%                  do not depend on x), and the data
%                  defining the problem which are used
%                  when invoking <QVI name> with other
%                  flags; it must be the first <QVI name>
%                  function call and should be called
%                  only one time
%
%      out = <QVI name>(1,x) returns a vector of dimension
%                             (nVar,1) containing F(x)
%
%      out = <QVI name>(2) returns a vector of dimensions
%                          (nVar,1) containing the lower
%                          bounds for the variable x
%
%      out = <QVI name>(3) returns a vector of dimensions
%                          (nVar,1) containing the upper
%                          bounds for the variable x
%
%      out = <QVI name>(4) returns a sparse matrix of
%                          dimensions (nIneq-nIneqInd,nVar)
%                          containing A of Ay <= b(x);
%
%      out = <QVI name>(5,x)  returns a vector of
%                             dimensions (nIneq-nIneqInd,nVar)
%                             containing b(x)of Ay <= b(x);
```

```
%
%      out = <QVI name>(6) returns a sparse matrix of
%                          dimensions (nEq,nVar)
%                          containing M of My = v(x);
%                          the first nEqInd rows refer to the
%                          those constraints that do not
%                          depend on x
%
%      out = <QVI name>(7,x) returns a vector of dimension
%                            (nEq,1) containing v(x) of
%                            My = v(x);
%                            the first nEqInd components refer
%                            to the those constraints that do
%                            not depend on x
%
%
% Problem definition

global nVar nIneq nEq nIneqInd nEqInd QVItestA QVItestb;

switch i

case 0
    nVar = 2;
    nIneq = 6;
    nEq = 0;
    nIneqInd = 4;
    nEqInd = 0;

    QVItestA = sparse([2 8/3; 5/4 2]);
    QVItestb = [-34; -24.25];

case 1
    % Function F
    out = QVItestA*x + QVItestb;

case 2
    % Bound constrains [l,u]: lower bound l
    out = zeros(nVar,1);

case 3
    % Bound constrains [l,u]: upper bound u
    out = 10*ones(nVar,1);

case 4
    % Linear constraints Ay<=b: matrix A
    out = eye(nVar);

case 5
    % Linear constraints Ay<=b: known term b
```

```
    out = 15*ones(nVar,1)-(ones(nVar)-eye(nVar))*x;

case 6
    % Equalities constraints My=v: matrix M
    out = [];

case 7
    % Equalities constraints My=v: known term v
    out = [];

end


return
```

Listing 5.4: OutZ46new.m

```
function out = OutZ46_new(i,x)
% QVILIB test problem OutZ46 [LBB/A/2-4-0-2-0]
% From: QVILIB: A LIBRARY OF QUASI-VARIATIONAL INEQUALITY
%                  TEST PROBLEMS
% Authors: Facchinei F., Kanzow C., Sagratella S.
%
% Input arguments:
%     i: function flag;
%        it must be an integer between 0 and 7
%     x: input vector of dimension (nVar,1)
%
% Description: <QVI name> = OutZ45_new
%
%    <QVI name>(0) initializes nVar (= number of
%                  variables), nIneq (= number of
%                  inequality constraints), nEq (= number
%                  of equality constraints), nIneqInd
%                  (= number of inequality constraints
%                  that do not depend on x), nEqInd
%                  (= number of equality constraints that
%                  do not depend on x), and the data
%                  defining the problem which are used
%                  when invoking <QVI name> with other
%                  flags; it must be the first <QVI name>
%                  function call and should be called
%                  only one time
%
%     out = <QVI name>(1,x) returns a vector of dimension
%                           (nVar,1) containing F(x)
%
%     out = <QVI name>(2) returns a vector of dimensions
%                         (nVar,1) containing the lower
```

```
%                           bounds for the variable x
%
%      out = <QVI name>(3) returns a vector of dimensions
%                           (nVar,1) containing the upper
%                           bounds for the variable x
%
%      out = <QVI name>(4) returns a sparse matrix of
%                           dimensions (nIneq-nIneqInd,nVar)
%                           containing A of Ay <= b(x);
%
%      out = <QVI name>(5,x)  returns a vector of
%                             dimensions (nIneq-nIneqInd,nVar)
%                             containing b(x)of Ay <= b(x);
%
%      out = <QVI name>(6) returns a sparse matrix of
%                           dimensions (nEq,nVar)
%                           containing M of My = v(x);
%                           the first nEqInd rows refer to the
%                           those constraints that do not
%                           depend on x
%
%      out = <QVI name>(7,x) returns a vector of dimension
%                             (nEq,1) containing v(x) of
%                            My = v(x);
%                             the first nEqInd components refer
%                             to the those constraints that do
%                             not depend on x
%
%
% Problem definition

global nVar nIneq nEq nIneqInd nEqInd QVItestA QVItestb;

switch i

case 0
    nVar = 2;
    nIneq = 5;
    nEq = 0;
    nIneqInd = 4;
    nEqInd = 0;

    QVItestA = sparse([2 8/3; 5/4 2]);
    QVItestb = [-34; -24.25];

case 1
    % Function F
    out = QVItestA*x + QVItestb;

case 2
```

```
    % Bound constrains [l,u]: lower bound l
    out = [0; 2];

case 3
    % Bound constrains [l,u]: upper bound u
    out = 10*ones(nVar,1);

case 4
    % Linear constraints Ay<=b: matrix A
    out = [1 0];

case 5
    % Linear constraints Ay<=b: known term b
    out = 15-[0 1]*x;

case 6
    % Equalities constraints My=v: matrix M
    out = [];

case 7
    % Equalities constraints My=v: known term v
    out = [];

end


return
```

5.1.2 RHS

Listing 5.5: RHS1A1new.m

```
function out = RHS1A1_new(i,x)
% QVILIB test problem RHS1A1 [LAL-A-200-0-0-199-0]
% From: QVILIB: A LIBRARY OF QUASI-VARIATIONAL INEQUALITY
%                   TEST PROBLEMS
% Authors: Facchinei F., Kanzow C., Sagratella S.
%
% Input arguments:
%     i: function flag;
%        it must be an integer between 0 and 7
%     x: input vector of dimension (nVar,1)
%
% Description: <QVI name> = RHS1A1_new
%
%    <QVI name>(0) initializes nVar (= number of
%                  variables), nIneq (= number of
%                  inequality constraints), nEq (= number
```

```
%                   of equality constraints), nIneqInd
%                   (= number of inequality constraints
%                   that do not depend on x), nEqInd
%                   (= number of equality constraints that
%                   do not depend on x), and the data
%                   defining the problem which are used
%                   when invoking <QVI name> with other
%                   flags; it must be the first <QVI name>
%                   function call and should be called
%                   only one time
%
%      out = <QVI name>(1,x) returns a vector of dimension
%                          (nVar,1) containing F(x)
%
%      out = <QVI name>(2) returns a vector of dimensions
%                       (nVar,1) containing the lower
%                       bounds for the variable x
%
%      out = <QVI name>(3) returns a vector of dimensions
%                       (nVar,1) containing the upper
%                       bounds for the variable x
%
%      out = <QVI name>(4) returns a sparse matrix of
%                       dimensions (nIneq-nIneqInd,nVar)
%                       containing A of Ay <= b(x);
%
%      out = <QVI name>(5,x)  returns a vector of
%                         dimensions (nIneq-nIneqInd,nVar)
%                         containing b(x)of Ay <= b(x);
%
%      out = <QVI name>(6) returns a sparse matrix of
%                       dimensions (nEq,nVar)
%                       containing M of My = v(x);
%                       the first nEqInd rows refer to the
%                       those constraints that do not
%                       depend on x
%
%      out = <QVI name>(7,x) returns a vector of dimension
%                        (nEq,1) containing v(x) of
%                        My = v(x);
%                        the first nEqInd components refer
%                        to the those constraints that do
%                        not depend on x
%
%
% Problem definition

global nVar nIneq nEq nIneqInd nEqInd;
global QVItestA QVItestb QVItestE QVItestC QVItestd;

```

```
switch i

case 0
    nVar = 200;
    nIneq = nVar-1;
    nEq = 0;
    nIneqInd = 0;
    nEqInd = 0;

    load RHS1A1.dat -mat
    QVItestb = 10*ones(nVar,1);
    QVItestd = 10*ones(nIneq,1);
    a = 0.5;
    QVItestC = a*QVItestC;

case 1
    % Function F
    out = QVItestA*x + QVItestb;

case 2
    % Bound constrains [l,u]: lower bound l
    out = [];

case 3
    % Bound constrains [l,u]: upper bound u
    out = [];

case 4
    % Linear constraints Ay<=b: matrix A
    out = QVItestE;

case 5
    % Linear constraints Ay<=b: known term b
    out = QVItestd-QVItestC*sin(x);

case 6
    % Equalities constraints My=v: matrix M
    out = [];

case 7
    % Equalities constraints My=v: known term v
    out = [];

end


return
```

Listing 5.6: RHS1B1new.m

```
function out = RHS1B1_new(i,x)
% QVILIB test problem RHS1B1 [LAL-A-200-0-0-199-0]
% From: QVILIB: A LIBRARY OF QUASI-VARIATIONAL INEQUALITY
%               TEST PROBLEMS
% Authors: Facchinei F., Kanzow C., Sagratella S.
%
% Input arguments:
%     i: function flag;
%        it must be an integer between 0 and 7
%     x: input vector of dimension (nVar,1)
%
% Description: <QVI name> = RHS1B1_new
%
%   <QVI name>(0) initializes nVar (= number of
%                 variables), nIneq (= number of
%                 inequality constraints), nEq (= number
%                 of equality constraints), nIneqInd
%                 (= number of inequality constraints
%                 that do not depend on x), nEqInd
%                 (= number of equality constraints that
%                 do not depend on x), and the data
%                 defining the problem which are used
%                 when invoking <QVI name> with other
%                 flags; it must be the first <QVI name>
%                 function call and should be called
%                 only one time
%
%     out = <QVI name>(1,x) returns a vector of dimension
%                           (nVar,1) containing F(x)
%
%     out = <QVI name>(2) returns a vector of dimensions
%                         (nVar,1) containing the lower
%                         bounds for the variable x
%
%     out = <QVI name>(3) returns a vector of dimensions
%                         (nVar,1) containing the upper
%                         bounds for the variable x
%
%     out = <QVI name>(4) returns a sparse matrix of
%                         dimensions (nIneq-nIneqInd,nVar)
%                         containing A of Ay <= b(x);
%
%     out = <QVI name>(5,x)  returns a vector of
%                            dimensions (nIneq-nIneqInd,nVar)
%                            containing b(x)of Ay <= b(x);
%
%     out = <QVI name>(6) returns a sparse matrix of
%                         dimensions (nEq,nVar)
```

```
%                             containing M of My = v(x);
%                             the first nEqInd rows refer to the
%                             those constraints that do not
%                             depend on x
%
%      out = <QVI name>(7,x) returns a vector of dimension
%                              (nEq,1) containing v(x) of
%                              My = v(x);
%                              the first nEqInd components refer
%                              to the those constraints that do
%                              not depend on x
%
%
% Problem definition

global nVar nIneq nEq nIneqInd nEqInd;
global QVItestA QVItestb QVItestE QVItestC QVItestd;

switch i

case 0
    nVar = 200;
    nIneq = nVar-1;
    nEq = 0;
    nIneqInd = 0;
    nEqInd = 0;

    load RHS1B1.dat -mat
    QVItestb = 10*ones(nVar,1);
    QVItestd = 10*ones(nIneq,1);

case 1
    % Function F
    out = QVItestA*x + QVItestb;

case 2
    % Bound constrains [l,u]: lower bound l
    out = [];

case 3
    % Bound constrains [l,u]: upper bound u
    out = [];

case 4
    % Linear constraints Ay<=b: matrix A
    out = QVItestE;

case 5
    % Linear constraints Ay<=b: known term b
    out = QVItestd-QVItestC*sin(x);
```

```

case 6
    % Equalities constraints My=v: matrix M
    out = [];

case 7
    % Equalities constraints My=v: known term v
    out = [];

end


return
```

Listing 5.7: RHS2A1new.m

```
function out = RHS2A1_new(i,x)
% QVILIB test problem RHS2A1 [LAL-A-200-0-0-199-0]
% From: QVILIB: A LIBRARY OF QUASI-VARIATIONAL INEQUALITY
%                  TEST PROBLEMS
% Authors: Facchinei F., Kanzow C., Sagratella S.
%
% Input arguments:
%      i: function flag;
%         it must be an integer between 0 and 7
%      x: input vector of dimension (nVar,1)
%
% Description: <QVI name> = RHS2A1_new
%
%    <QVI name>(0) initializes nVar (= number of
%                  variables), nIneq (= number of
%                  inequality constraints), nEq (= number
%                  of equality constraints), nIneqInd
%                  (= number of inequality constraints
%                  that do not depend on x), nEqInd
%                  (= number of equality constraints that
%                  do not depend on x), and the data
%                  defining the problem which are used
%                  when invoking <QVI name> with other
%                  flags; it must be the first <QVI name>
%                  function call and should be called
%                  only one time
%
%      out = <QVI name>(1,x) returns a vector of dimension
%                           (nVar,1) containing F(x)
%
%      out = <QVI name>(2) returns a vector of dimensions
%                         (nVar,1) containing the lower
%                         bounds for the variable x
```

```
%
%      out = <QVI name>(3) returns a vector of dimensions
%                          (nVar,1) containing the upper
%                          bounds for the variable x
%
%      out = <QVI name>(4) returns a sparse matrix of
%                          dimensions (nIneq-nIneqInd,nVar)
%                          containing A of Ay <= b(x);
%
%      out = <QVI name>(5,x)  returns a vector of
%                            dimensions (nIneq-nIneqInd,nVar)
%                            containing b(x)of Ay <= b(x);
%
%      out = <QVI name>(6) returns a sparse matrix of
%                          dimensions (nEq,nVar)
%                          containing M of My = v(x);
%                          the first nEqInd rows refer to the
%                          those constraints that do not
%                          depend on x
%
%      out = <QVI name>(7,x) returns a vector of dimension
%                           (nEq,1) containing v(x) of
%                           My = v(x);
%                           the first nEqInd components refer
%                           to the those constraints that do
%                           not depend on x
%
%
% Problem definition

global nVar nIneq nEq nIneqInd nEqInd;
global QVItestA QVItestb QVItestE QVItestC QVItestd;

switch i

case 0
    nVar = 200;
    nIneq = nVar-1;
    nEq = 0;
    nIneqInd = 0;
    nEqInd = 0;

    load RHS2A1.dat -mat
    QVItestb = 10*ones(nVar,1);
    QVItestd = 10*ones(nIneq,1);
    a = 0.5;
    QVItestC = a*QVItestC;

case 1
    % Function F
```

```
        out = QVItestA*x + QVItestb;

    case 2
        % Bound constrains [l,u]: lower bound l
        out = [];

    case 3
        % Bound constrains [l,u]: upper bound u
        out = [];

    case 4
        % Linear constraints Ay<=b: matrix A
        out = QVItestE;

    case 5
        % Linear constraints Ay<=b: known term b
        out = QVItestd-QVItestC*x;

    case 6
        % Equalities constraints My=v: matrix M
        out = [];

    case 7
        % Equalities constraints My=v: known term v
        out = [];

    end


    return
```

Listing 5.8: RHS2B1new.m

```
function out = RHS2B1_new(i,x)
% QVILIB test problem RHS2A1 [LAL-A-200-0-0-199-0]
% From: QVILIB: A LIBRARY OF QUASI-VARIATIONAL INEQUALITY
%                  TEST PROBLEMS
% Authors: Facchinei F., Kanzow C., Sagratella S.
%
% Input arguments:
%      i: function flag;
%          it must be an integer between 0 and 7
%      x: input vector of dimension (nVar,1)
%
% Description: <QVI name> = RHS2B1_new
%
%    <QVI name>(0) initializes nVar (= number of
%                  variables), nIneq (= number of
%                  inequality constraints), nEq (= number
```

```
%                     of equality constraints), nIneqInd
%                     (= number of inequality constraints
%                     that do not depend on x), nEqInd
%                     (= number of equality constraints that
%                     do not depend on x), and the data
%                     defining the problem which are used
%                     when invoking <QVI name> with other
%                     flags; it must be the first <QVI name>
%                     function call and should be called
%                     only one time
%
%      out = <QVI name>(1,x) returns a vector of dimension
%                            (nVar,1) containing F(x)
%
%      out = <QVI name>(2) returns a vector of dimensions
%                         (nVar,1) containing the lower
%                         bounds for the variable x
%
%      out = <QVI name>(3) returns a vector of dimensions
%                         (nVar,1) containing the upper
%                         bounds for the variable x
%
%      out = <QVI name>(4) returns a sparse matrix of
%                         dimensions (nIneq-nIneqInd,nVar)
%                         containing A of Ay <= b(x);
%
%      out = <QVI name>(5,x)  returns a vector of
%                           dimensions (nIneq-nIneqInd,nVar)
%                           containing b(x)of Ay <= b(x);
%
%      out = <QVI name>(6) returns a sparse matrix of
%                         dimensions (nEq,nVar)
%                         containing M of My = v(x);
%                         the first nEqInd rows refer to the
%                         those constraints that do not
%                         depend on x
%
%      out = <QVI name>(7,x) returns a vector of dimension
%                          (nEq,1) containing v(x) of
%                          My = v(x);
%                          the first nEqInd components refer
%                          to the those constraints that do
%                          not depend on x
%
%
% Problem definition

global nVar nIneq nEq nIneqInd nEqInd;
global QVItestA QVItestb QVItestE QVItestC QVItestd;

```

```
switch i

case 0
    nVar = 200;
    nIneq = nVar-1;
    nEq = 0;
    nIneqInd = 0;
    nEqInd = 0;

    load RHS2B1.dat -mat
    QVItestb = 10*ones(nVar,1);
    QVItestd = 10*ones(nIneq,1);

case 1
    % Function F
    out = QVItestA*x + QVItestb;

case 2
    % Bound constrains [l,u]: lower bound l
    out = [];

case 3
    % Bound constrains [l,u]: upper bound u
    out = [];

case 4
    % Linear constraints Ay<=b: matrix A
    out = QVItestE;

case 5
    % Linear constraints Ay<=b: known term b
    out = QVItestd-QVItestC*x;

case 6
    % Equalities constraints My=v: matrix M
    out = [];

case 7
    % Equalities constraints My=v: known term v
    out = [];

end


return
```

5.1.3 Example

Listing 5.9: Examplegenerator.m

```
function out = Example_generator(i,nVar,x)
% Test problem Example 2
% From: Some projection-like methods for the generalized
%       Nash equilibria
% Authors: Jianzhong Zhang, Biao Qu, Naihua Xiu
%
% Input arguments:
%     i: function flag;
%        it must be an integer between 0 and 7
%     x: input vector of dimension (nVar,1)
%     nVar: number of variables
%
% Description: <QVI name> = Example_generator
%
%    <QVI name>(0,nVar) initializes nIneq (= number of
%                        inequality constraints), nEq
%                       (= number of equality constraints),
%                        nIneqInd (= number of inequality
%                        constraints that do not depend on
%                        x), nEqInd (= number of equality
%                        constraints that do not depend on
%                        x), and the data defining the
%                        problem which are used when
%                        invoking <QVI name> with other
%                        flags; it must be the first
%                        <QVI name> function call
%                        and should be called only one time
%
%     out = <QVI name>(1,nVar,x) returns a vector of
%                        dimension (nVar,1) containing F(x)
%
%     out = <QVI name>(2,nVar) returns a vector of
%                        dimensions (nVar,1)
%                        containing the lower bounds
%                        for the variable x
%
%     out = <QVI name>(3,nVar) returns a vector of
%                        dimensions (nVar,1)
%                        containing the upper bounds
%                        for the variable x
%
%     out = <QVI name>(4,nVar) returns a sparse matrix of
%                        dimensions (nIneq-nIneqInd,nVar)
%                        containing A of Ay <= b(x);
%
%     out = <QVI name>(5,nVar,x)  returns a vector of
```

```
%                               dimensions (nIneq-nIneqInd,nVar)
%                               containing b(x) of Ay <= b(x);
%
%       out = <QVI name>(6,nVar) returns a sparse matrix of
%                               dimensions (nEq,nVar) containing
%                               M of My = v(x); the first nEqInd
%                               rows refer to the those
%                               constraints that do not depend
%                               on x
%
%       out = <QVI name>(7,nVar,x) returns a vector of
%                               dimension (nEq,1) containing
%                               v(x) of My = v(x); the first
%                               nEqInd components refer to the
%                               those constraints that do not
%                               depend on x
%
%
% Problem definition
global nIneq nEq nIneqInd nEqInd c b;

switch i

case 0
     nIneq = 3*nVar;
     nEq = 0;
     nIneqInd = 2*nVar;
     nEqInd = 0;

     for j=1:nVar
          c(j) = 12-2*j;
          b(j) = 1.3-j*0.1;
     end

case 1
     % Function F
     Q = sum(x);
     out = zeros(nVar,1);
     for j=1:nVar
          out(j) = c(j)+(x(j)/5)^(1/b(j))+(5000/Q)^(1/1.1)
               *(x(j)/(1.1*Q)-1);
     end

case 2
     % Bound constrains [l,u]: lower bound l
     out = ones(nVar,1);

case 3
     % Bound constrains [l,u]: upper bound u
     out = 150*ones(nVar,1);
```

```
case 4
    % Linear constraints Ay<=b: matrix A
    out = eye(nVar);

case 5
    % Linear constraints Ay<=b: known term b
    out = 700*ones(nVar,1)-(ones(nVar)-eye(nVar))*x;

case 6
    % Equalities constraints My=v: matrix M
    out = [];

case 7
    % Equalities constraints My=v: known term v
    out = [];

end


return
```

5.2 Hybrid Extragradient Methods

In this section there are the MATLAB codes of Generalized Solodov and Nguyen-Strodiot.

Listing 5.10: Solodov.m

```
function [xn,residuals] = Solodov_quadprog(x0,F,A,b,Aeq,
    beq,lb,ub,nVar,gamma,theta,c,options)
% Algorithm 1b from
% *A new class of hybrid extragradient algorithms for
% solving quasi-equilibrium problems*
% by J.J. Strodiot, T.T.V. Nguyen, V.H. Nguyen
%
% Solodov_quadprog returns the next terms of the sequence
% genereted from algorithm 1b x_{n}=F(x_{n-1})
% and its residuals
%
% Input arguments:
%       x0 = previus terms of the sequence
%       F = function of QVI, that is F(x)^T(y-x)\geq 0
%       A = inequality constraints of QVI's domain,
%           , i.e., Ax <= b(x)
%       b = known term of the inequality constrains,
%           that is b(x)
%       Aeq = equality constraints of QVI's domain,
```

```
%               , i.e., Aeq*x = beq(x)
%       beq = known term of the equality constrains,
%              that is beq(x)
%       lb = lower bound for the variable x
%       ub = upper bound for the variable x
%       nVar = number of variables
%       theta,sigma,c = parameters of Alg 1b
%       options = optioptions(...) in order to make work
%                    quadprog
%
% Output arguments:
%       xn = next term of the sequence
%       residuals = ||x0-P_{K(x0)}(x0-F(x0))||

k = -1;

% Prediction step [coicides with P_K(x_k)(x_k-F(x_k))]
xx0 = quadprog(eye(nVar),F(x0)-x0,A,b(x0),Aeq,beq(x0),lb,
    ub,x0,options);
rx = x0-xx0;
residuals = norm(rx,2);

% Line search
while (k == -1 || sx < dx) && (k<30)
    k = k+1;
    z0 = (1-theta^k)*x0+(theta^k)*xx0;
    Fz0 = F(z0);
    sx = Fz0'*rx;
    dx = c*residuals^2;
end

disp(['k = ',num2str(k,'%d')])

% Computing x_(k+1): projection on K(x_k) intersected
% with hyperplane Hk = {x: <F(z_k),x-z_k> <=0}
Aa = [A;Fz0'];
Bb = [b(x0);Fz0'*z0];
sigma = (Fz0'*(x0-z0))/norm(Fz0,2)^2;
xn = quadprog(eye(nVar),gamma*sigma*Fz0-x0,Aa,Bb,Aeq,
        beq(x0),lb,ub,x0,options);

end
```

Listing 5.11: NguyenStrodiot.m

```
function [xn,residuals] = NguyenStrodiot_quadprog(x0,F,A,b,Aeq,beq,lb,ub,nVar,gamma,elle,ro,ro1,mu,c,options)
% Algorithm QVI from
% *A class of hybrid methods for quasi-variational inequalities*
% by J.J. Strodiot, T.T.V. Nguyen, V.H. Nguyen,
% T.P.D. Nguyen
%
% Hybrid_methods_quadprog returns the next terms of the
% sequence genereted from algorithm x_{n}=F(x_{n-1})
% and its residuals
%
% Input arguments:
%       x0 = previus terms of the sequence
%       F = function of QVI, that is F(x)^T(y-x)\geq 0
%       A = inequality constraints of QVI's domain,
%           , i.e., Ax <= b(x)
%       b = known term of the inequality constrains,
%           that is b(x)
%       Aeq = equality constraints of QVI's domain,
%             , i.e., Aeq*x = beq(x)
%       beq = known term of the equality constrains,
%             that is beq(x)
%       lb = lower bound for the variable x
%       ub = upper bound for the variable x
%       nVar = number of variables
%       gamma,elle,ro,ro1,mu,c = parameters of Alg QVI
%       options = optioptions(...) in order to make work
%                 quadprog
%
% Output arguments:
%       xn = next term of the sequence
%       residuals = ||x0-P_{K(x0)}(x0-F(x0))||

k = -1; sx = 0; dx = -1;

% Prediction step z_k = P_K(x_k)(x_k-F(x_k))
z0 = quadprog(eye(nVar),F(x0)-x0,A,b(x0),Aeq,beq(x0),lb,ub,x0,options);
rx = x0-z0;
residuals = norm(rx,2);

% Line search
while (sx > dx) && (k<30)
    k = k+1;
    y0 = (1-gamma*elle^k)*x0+(gamma*elle^k)*z0;
    sx = (F(x0)-F(y0))'*rx;
    dx = c*residuals^2;
```

```
end

disp(['k = ',num2str(k,'%d')])
beta = gamma*elle^k;

% Computing descent direction
d_k = x0-y0+F(y0);     %d1
%d_k = x0-y0+F(x0)+F(y0);    %d2
%d_k = x0-y0-beta*(F(x0)-F(y0)/beta);   %d3
d_bar = ro/(1+ro)*(x0-y0)+1/(1+ro)*d_k;

% Computing hyperplane
% Hk = {x: <d_bar,x_k-x> = alpha*beta*||d_bar||^2}
alpha = ((1-ro1*(ro/4*mu))*norm(x0-y0,2)^2-ro1*beta*(F(x0
    )-F(y0))'*(x0-y0))/(beta^2*norm(d_bar,2)^2);

% Computing x_(k+1): projection on K(x_k)
xn = quadprog(eye(nVar),-(x0-alpha*beta*d_bar),A,b(x0),
    Aeq,beq(x0),lb,ub,x0,options);

end
```

5.3 Accelerated Methods

In this sections there are the MATLAB codes for Solodov and Nguyen-Strodiot methods accelerated with regularized nonlinear acceleration (RNA) and regularized topological Shanks acceleration (RTSA).

5.3.1 RNA

Listing 5.12: RNA.m

```
function [x_extr, c] = RNA(X, lambda)
% Regularized Nonlinear Acceleration (RNA) Alg2 from
% *Regularized Nonlinear Acceleration*
% by Damien Scieur, Alexandre d'Aspremont, Francis Bach
%
% RNA returns the approximation
% sum_{i=0}^{k}(c*_{lambda})_{i}x_{i} of the
% sequence {x_{i}} generated by x_{i}= G(x_{i-1}) where
% G is a fixed-point method
%
% Input arguments:
%       X = [x0,x1,...,x_{2*Kmax+1}]
%       lambda = regularization parameter, it must be >0
%
% Output arguments:
```

```
%        x_extr = approximation
%                 sum_{i=0}^{k}(c*_{lambda})_{i}x_{i}
%        c = vector of coefficient (c*_{lambda})_{i}


% Computing R = [r_0,r_1,...,r_k] where
% r_i:= R(:,i) = X(:,i+1)- X(:,i)
R = diff(X,1,2);
R = R'*R;
R = R/norm(R); % normalized

k = size(R,2);

matrix = (R + eye(k)*lambda);

% Coefficient
c = matrix\ones(k,1);
c = c/sum(c);

% Approximation
x_extr = X(:,2:end)*c;

end
```

Listing 5.13: RegularizedSolodovRNA.m

```
%% Solodov implemented with regularized nonlinear acceleration
%
% Regularized Nonlinear Acceleration (RNA) Alg2 from
% *Regularized Nonlinear Acceleration*
% by Damien Scieur, Alexandre d'Aspremont, Francis Bach
%
% QVI formulation (Latex notation used):
%     find x such that: g(x,x) \leq 0,
%                       M(x)x+v(x) = 0 and
%                       F(x)^T (y-x)\geq 0,
%                       for all y such that
%                       g(y,x)\leq 0 and M(x)y+v(x) = 0
%                       where
%                       F(x):\Re^{nVar}\to\Re^{nVar}
%                       g(y,x):\Re^{nVar}\times
%                              \Re^{nVar}\to\Re^{nIneq}
%                       M(x):\Re^{nVar}\to
%                            \Re^{nEq\times nVar}
%                       v(x):\Re^{nVar}\to\Re^{nEq}
%
%                       Note that some of the constraints
%                       g(y,x)\leq 0 and M(x)y+v(x) = 0
```

```
%                          may actually be independent of x.
%                          The constraints are always
%                          ordered so that these constraints
%                          independent of x are the first
%                          ones. For example
%                              g(y,x) = [ g1(y); g2(y,x) ]

%
%% Problem Definition
clear all;
close all;
clc
addpath('.../QVILIB_quadprog')
Method_name = '_Solodov';
QVIname = 'RHS2A1_new';
QVIproblem = @RHS2A1_new;

% Generating data files for some large scale problems of
% QVILIB
% N.B. necessary only for RHS QVI type problems
QVILibGenData(QVIname)


% Initialization of the data defining the problem
QVIproblem(0)

% Starting point
number = 1;
x0 = startingPoints(QVIname,number);

% Function
F = @(x)QVIproblem(1,x);
nVar = size(F(x0),1);

% Equality constraints
Aeq = QVIproblem(6);
beq = @(x)QVIproblem(7,x);

% Inequality constraints
A = QVIproblem(4);
b = @(x)QVIproblem(5,x);

% Bound constraints
lb = QVIproblem(2);
ub = QVIproblem(3);

% Residuals
residuals_RNA = [];      % residuals from the Solodov_RNA
                            alg
residuals = [];          % residuals from the standard
```

Solodov

```
%% QVI_algorithm accelerated
% Algorithm Initialization
% Parameters
gamma = 1.99;
theta = 0.5;
c = 0.5;
iter = 0;
options = optimoptions('quadprog','Algorithm','interior-point-convex',
'OptimalityTolerance',1e-20,'MaxIterations',500);

% Set number of outer loops
Nmax = 5;

% Set number of inner loops
Kmax = 3;

% Set the total number of cycles for the original sequence
TOT = Nmax*(2*Kmax+1);

% Set regularization parameter
info.lambdaRange=[1, 1e-14];
lambda_min = min(info.lambdaRange);
lambda_max = max(info.lambdaRange);

% Computing grid
lambdavec = [0, logspace(log10(lambda_min),log10(lambda_max),Kmax)];

% Main part
% Start the outer loop of the RNA method
for i = 1:Nmax
    X(:,1) = x0;
    disp(['Inner iteration 1 of cycle ',num2str(i,'%d'),' completed']);

    % Start of the inner loop of the modified RNA method
    for n = 1:2*Kmax+1
        iter = iter+1;

        % Performing 2*Kmax+1 Solodov steps
        [x0,residuals_RNA(iter)] = Solodov_quadprog(x0,F,A,b,Aeq,beq,lb,ub,nVar,gamma,theta,c,options);
        X(:,n+1) = x0;
        disp(['Inner iteration ', num2str(n+1,'%d'),' of cycle ', num2str(i,'%d'),' completed']);
```

```
    end

    % End of the inner loop

    warning off
    normvec = zeros(size(lambdavec));   % for the grid search
    memory_extrapolation_1 = zeros(size(x0,1),size(lambdavec,2));
    memory_extrapolation_2 = zeros(size(x0,1),size(lambdavec,2));

    % Grid search on lambda
    for h = 1:length(lambdavec)
        % extrapolation using differents values of lambda
        [x_l,~] = RNA(X,lambdavec(h));
        memory_extrapolation_1(:,h) = x_l;
        memory_extrapolation_2(:,h) = quadprog(eye(nVar),F(x_l)-x_l,A,b(x_l),Aeq,beq(x_l),lb,ub,x_l,options);
        normvec(h) = norm(memory_extrapolation_1(:,h)-memory_extrapolation_2(:,h))^2;
        disp(['Inner iteration for lambda ', num2str(h,'%d'),' of cycle ', num2str(i,'%d'),' completed']);
    end

    % Choosing best lambda
    warning on
    [~, idx_min] = min(normvec);
    lambdamin = lambdavec(idx_min);
    info.lambdaUsed(i) = lambdamin;

    % Approximation of x*
    x0 = memory_extrapolation_1(:,idx_min);

    disp(['* Outer iteration ', num2str(i,'%d'), ' completed']);
    disp(' ');
end
% End of the outer loop

[x0,residuals_RNA(TOT+1)] = Solodov_quadprog(x0,F,A,b,Aeq,beq,lb,ub,nVar,gamma,theta,c,options);

%
%% QVI_alg basic method
% Reinitialize the starting point
X0 = startingPoints(QVIname,number);

```

```
% Solodov_generalizazion (basic method)
for j = 1:TOT+1
    % In output, X0 is the new element x_j
    [X0,residuals(j)] = Solodov_quadprog(X0,F,A,b,Aeq,beq,lb,ub,nVar,gamma,theta,c,options);
end


%
%% Graph of residuals
graphic_star = 2*Kmax+2 :2*Kmax+1: TOT+1;

fig1 = figure('Position', get(0, 'Screensize'));
semilogy(residuals,'k','Linewidth',6);
hold on
semilogy(residuals_RNA,'r--o','MarkerIndices',graphic_star,'Linewidth',6,'MarkerSize',10);
hold on
title(['Residuals, problem: ', QVIname],'Fontsize',22);
xlabel('iterations');
legend({'Solodov S.','Solodov S. + RNA'},'LOCATION', 'SouthWest','Fontsize',22);

%
%% Saving data
savefig(fig1,['Graph/',QVIname, Method_name,'_Nmax_',num2str(Nmax),'_Kmax_',num2str(Kmax),'_bestlambda_RNA','.fig'])
saveas(fig1, ['Graph/',QVIname, Method_name,'_Nmax_',num2str(Nmax),'_Kmax_', num2str(Kmax),'_bestlambda_RNA','.jpg'])
save(['Results_number_RNA_alg/',QVIname, Method_name, '_Nmax_',num2str(Nmax), '_Kmax_', num2str(Kmax),'_bestlambda_RNA','.mat'])
```

Listing 5.14: RegularizedNguyenStrodiotRNA.m

```
%% Nguyen-Strodiot implemented with regularized nonlinear
    acceleration
%
% Regularized Nonlinear Acceleration (RNA) Alg2 from
% *Regularized Nonlinear Acceleration*
% by Damien Scieur, Alexandre d'Aspremont, Francis Bach
%
% QVI formulation (Latex notation used):
%     find x such that: g(x,x) \leq 0,
%                        M(x)x+v(x) = 0 and
%                        F(x)^T (y-x)\geq 0,
%                        for all y such that
%                        g(y,x)\leq 0 and M(x)y+v(x) = 0
%                        where
%                        F(x):\Re^{nVar}\to\Re^{nVar}
%                        g(y,x):\Re^{nVar}\times
%                               \Re^{nVar}\to\Re^{nIneq}
%                        M(x):\Re^{nVar}\to
%                             \Re^{nEq\times nVar}
%                        v(x):\Re^{nVar}\to\Re^{nEq}
%
%                        Note that some of the constraints
%                        g(y,x)\leq 0 and M(x)y+v(x) = 0
%                        may actually be independent of x.
%                        The constraints are always
%                        ordered so that these constraints
%                        independent of x are the first
%                        ones. For example
%                            g(y,x) = [ g1(y); g2(y,x) ]

%
%% Problem Definition
clear all;
close all;
clc
addpath('.../QVILIB_quadprog')
Method_name = '_Nguyen-Strodiot';
QVIname = 'RHS2A1_new';
QVIproblem = @RHS2A1_new;

% Generating data files for some large scale problems of
   QVILIB
% N.B. necessary only for RHS QVI type problems
QVILibGenData(QVIname)

% Initialization of the data defining the problem
QVIproblem(0)

```

```
% Starting point
number = 1;
x0 = startingPoints(QVIname,number);

% Function
F = @(x)QVIproblem(1,x);
nVar = size(F(x0),1);

% Equality constraints
Aeq = QVIproblem(6);
beq = @(x)QVIproblem(7,x);

% Inequality constraints
A = QVIproblem(4);
b = @(x)QVIproblem(5,x);

% Bound constraints
lb = QVIproblem(2);
ub = QVIproblem(3);

% Residuals
residuals_RNA = [];      % residuals from the
                           Nguyen-Strodiot_RNA alg
residuals = [];          % residuals from the standard
                           Nguyen-Strodiot


%% QVI_algorithm accelerated
% Algorithm Initialization
% Parameters
c = 0.5;
elle = 0.5;
gamma = 0.99;
ro = 1;            % ro >= 0
ro1 = 1/(1+ro);
mu = 0.5;              % mu > max(1/4,ro/(4*(1+ro-c)))
iter = 0;
options = optimoptions('quadprog','Algorithm','interior-
    point-convex','OptimalityTolerance',1e-20,'
    MaxIterations',500);

% Set number of outer loops
Nmax = 5;

% Set number of inner loops
Kmax = 7;

% Set the total number of cycles for the original
    sequence
TOT = Nmax*(2*Kmax+1);
```

```

% Set regularization parameter
info.lambdaRange=[1, 1e-14];
lambda_min = min(info.lambdaRange);
lambda_max = max(info.lambdaRange);

% Computing grid
lambdavec = [0, logspace(log10(lambda_min),log10(lambda_max),Kmax)];

% Main part
% Start the outer loop of the RNA method
for i = 1:Nmax
    X(:,1) = x0;
    disp(['Inner iteration 1 of cycle ',num2str(i,'%d'),' completed']);

    % Start of the inner loop of the modified RNA method
    for n = 1:2*Kmax+1
        iter = iter+1;
        % Performing 2*Kmax+1 Nguyen-Strodiot steps
        [x0,residuals_RNA(iter)] = NguyenStrodiot_quadprog(x0,F,A,b,Aeq,beq,lb,ub,nVar,gamma,elle,ro,ro1,mu,c,options);
        X(:,n+1) = x0;
        disp(['Inner iteration ', num2str(n+1,'%d'), ' of cycle ', num2str(i,'%d'),' completed']);
    end
    % End of the inner loop

    warning off
    normvec = zeros(size(lambdavec));   % for the grid search
    memory_extrapolation_1 = zeros(size(x0,1),size(lambdavec,2));
    memory_extrapolation_2 = zeros(size(x0,1),size(lambdavec,2));

    % Grid search on lambda
    for h = 1:length(lambdavec)
        % extrapolation using differents values of lambda
        [x_l,~] = RNA(X,lambdavec(h));
        memory_extrapolation_1(:,h) = x_l;
        memory_extrapolation_2(:,h) = quadprog(eye(nVar),F(x_l)-x_l,A,b(x_l),Aeq,beq(x_l),lb,ub,x_l,options);
        normvec(h) = norm(memory_extrapolation_1(:,h)-memory_extrapolation_2(:,h))^2;
        disp(['Inner iteration for lambda ', num2str(h,'%d'), ' of cycle ', num2str(i,'%d'),' completed
```

```
                ']);
    end

    % Choosing best lambda
    warning on
    [~, idx_min] = min(normvec);
    lambdamin = lambdavec(idx_min);
    info.lambdaUsed(i) = lambdamin;

    % Approximation of x*
    x0 = memory_extrapolation_1(:,idx_min);

    disp(['* Outer iteration ', num2str(i,'%d'), '
        completed']);
    disp(' ');
end
% End of the outer loop

[x0,residuals_RNA(TOT+1)] = NguyenStrodiot_quadprog(x0,F,
    A,b,Aeq,beq,lb,ub,nVar,gamma,elle,ro,ro1,mu,c,options)
    ;

%
%% QVI_alg basic method
% Reinitialize the starting point
X0 = startingPoints(QVIname,number);

% Nguyen-Strodiot (basic method)
for j = 1:TOT+1
    % In output, X0 is the new element x_j
    [X0,residuals(j)] = NguyenStrodiot_quadprog(X0,F,A,b,
        Aeq,beq,lb,ub,nVar,gamma,elle,ro,ro1,mu,c,options)
        ;
end


%
%% Graph of residuals
graphic_star = 2*Kmax+2:2*Kmax+1:TOT+1;

fig1 = figure('Position', get(0, 'Screensize'));
semilogy(residuals,'k','Linewidth',6);
hold on
semilogy(residuals_RNA,'r--o','MarkerIndices',
    graphic_star,'Linewidth',6,'MarkerSize',10);
hold on
title(['Residuals with d3, problem: ', QVIname],'Fontsize
    ',22);
xlabel('iterations');
legend({'Nguyen-Strodiot','Nguyen-Strodiot + RNA'},'
```

```
    LOCATION','SouthWest','Fontsize',22);


%
%% Saving data
savefig(fig1,['Graph/',QVIname, Method_name,'_d3', '_Nmax_', num2str(Nmax),'_Kmax_', num2str(Kmax),'_bestlambda_RNA','.fig'])
saveas(fig1, ['Graph/',QVIname, Method_name,'_d3', '_Nmax_', num2str(Nmax),'_Kmax_', num2str(Kmax),'_bestlambda_RNA','.jpg'])
save(['Results_number_RNA_alg/',QVIname, Method_name,'_d3', '_Nmax_', num2str(Nmax), '_Kmax_', num2str(Kmax),'_bestlambda_RNA','.mat'])
```

5.3.2 RTSA

Listing 5.15: TopologicalShanksTransformation.m

```
function [ x_extr, c] = topologicalShanksTransformation_new( x,y,lambda,S )
% TopologicalShanksTransformation
%    [x_extr] = topologicalShanksTransformation_new(x,y,lambda,S)
%    extrapolate the limit
%    The output x_extr is equal to sum_i=1^k c^*_i x_i.
%
% Author: Stefano Cipolla
%
if (nargin < 4)
     k = (size(x,2))/2;
     Delta = diff(x,1,2);
     b=y'*Delta;
     S=hankel(b(1:k)',b(k:end)');
     S=S'*S;
end
%%% This part MUST be further optimized
%S=S'*S;
S=S+spdiags(lambda*ones(size(S,1),1),0,size(S,1),size(S,1));
c=S\ones(size(S,1),1);
c=c./sum(c);
%%%
x_extr=x(:,size(S,1)+1:end)*c;
end
```

Listing 5.16: RegularizedTopologicalSolodov.m

```
%% Solodov implemented with restarted topological Shanks
    acceleration
%
% Restarted topological Shanks acceleration (RTSA)
% Alg from
% *Anderson type trasformations for systems of nonlinear
    equations*
% by Claude Branzinski, Stefano Cipolla, Michela Redivo-
    Zoglia, Yousef Saad
%
% QVI formulation (Latex notation used):
%     find x such that: g(x,x) \leq 0,
%                       M(x)x+v(x) = 0 and
%                       F(x)^T (y-x)\geq 0,
%                       for all y such that
%                       g(y,x)\leq 0 and M(x)y+v(x) = 0
%                       where
%                       F(x):\Re^{nVar}\to\Re^{nVar}
%                       g(y,x):\Re^{nVar}\times
%                              \Re^{nVar}\to\Re^{nIneq}
%                       M(x):\Re^{nVar}\to
%                            \Re^{nEq\times nVar}
%                       v(x):\Re^{nVar}\to\Re^{nEq}
%
%                       Note that some of the constraints
%                       g(y,x)\leq 0 and M(x)y+v(x) = 0
%                       may actually be independent of x.
%                       The constraints are always
%                       ordered so that these constraints
%                       independent of x are the first
%                       ones. For example
%                           g(y,x) = [ g1(y); g2(y,x) ]


%% Problem Definition
clear all;
close all;
clc
addpath('.../QVILIB_quadprog')
Method_name = '_Solodov';
QVIname = 'RHS2A1_new';
QVIproblem = @RHS2A1_new;

% Generating data files for some large scale problems of
% QVILIB
% N.B. necessary only for RHS QVI type problems
QVILibGenData(QVIname)

```

```
% Initialization of the data defining the problem
QVIproblem(0)

% Starting point
number = 1;
x0 = startingPoints(QVIname,number);

% Function;
F = @(x)QVIproblem(1,x);
nVar = size(F(x0),1);

% Equality constraints
Aeq = QVIproblem(6);
beq = @(x)QVIproblem(7,x);

% Inequality constraints
A = QVIproblem(4);
b = @(x)QVIproblem(5,x);

% Bound constraints
lb = QVIproblem(2);
ub = QVIproblem(3);

% Residuals
residuals_RTSA = [];     % residuals from the Solodov_RNA
    alg
residuals = [];          % residuals from the standard
    Solodov


%% QVI_algorithm accelerated
% Algorithm Initialization
% Parameters
gamma = 1.99;
theta = 0.5;
c = 0.5;
iter = 0;
options = optimoptions('quadprog','Algorithm','interior-
    point-convex', 'OptimalityTolerance',1e-20,'
    MaxIterations',500);

% Set number of outer loops
Nmax = 5;

% Set number of inner loops
Kmax = 3;

% Set the total number of cycles for the original
    sequence;
```

```
TOT = Nmax*(2*Kmax+1);

% Set regularization parameter
info.lambdaRange=[1, 1e-14];
lambda_min = min(info.lambdaRange);
lambda_max = max(info.lambdaRange);

% Computing grid
lambdavec = [0, logspace(log10(lambda_min),log10(lambda_max),Kmax)];

% Main part
% Start the outer loop of the RTSA method
for i = 1:Nmax
    X(:,1) = x0;
    disp(['Inner iteration 1 of cycle ',num2str(i,'%d'),' completed']);

    % Start of the inner loop of the modified RTSA method
    for n = 1:2*Kmax+1
        iter = iter+1;

        % Performing 2*Kmax+1 Solodov steps
        [x0,residuals_RTSA(iter)] = Solodov_quadprog(x0,F,A,b,Aeq,beq,lb,ub,nVar,gamma,theta,c,options);
        X(:,n+1) = x0;
        disp(['Inner iteration ', num2str(n+1,'%d'), ' of cycle ', num2str(i,'%d'),' completed']);
    end

    % End of the inner loop

    warning off
    normvec = zeros(size(lambdavec));   % for the grid search
    memory_extrapolation_1 = zeros(size(x0,1),size(lambdavec,2));
    memory_extrapolation_2 = zeros(size(x0,1),size(lambdavec,2));
    param.y = X(:,end);

    % Grid search on lambda
    for h = 1:length(lambdavec)
        % extrapolation using differents values of lambda
        x_l = topologicalShanksTransformation_new(X,param.y,lambdavec(h));
        memory_extrapolation_1(:,h) = x_l;
        memory_extrapolation_2(:,h) = quadprog(eye(nVar),F(x_l)-x_l,A,b(x_l),Aeq,beq(x_l),lb,ub,x_l,
```

```
            options);
        normvec(h) = norm(memory_extrapolation_1(:,h)-memory_extrapolation_2(:,h))^2;
        disp(['Inner iteration for lambda ', num2str(h,'%d'), ' of cycle ', num2str(i,'%d'),' completed']);
    end

    % Choosing best lambda
    warning on
    [~, idx_min] = min(normvec);
    lambdamin = lambdavec(idx_min);
    info.lambdaUsed(i) = lambdamin;

    % Approximation of x*
    x0 = memory_extrapolation_1(:,idx_min);

    disp(['* Outer iteration ', num2str(i,'%d'), ' completed']);
    disp(' ');
end
% End of the outer loop

[x0,residuals_RTSA(TOT+1)] = Solodov_quadprog(x0,F,A,b,Aeq,beq,lb,ub,nVar,gamma,theta,c,options);

%
%% QVI_alg basic method
% Reinitialize the starting point
X0 = startingPoints(QVIname,number);

% Solodov_generalizazion (basic method)
for j = 1:TOT+1
    % In output, X0 is the new element x_j
    [X0,residuals(j)] = Solodov_quadprog(X0,F,A,b,Aeq,beq,lb,ub,nVar,gamma,theta,c,options);
end


%
%% Graph of residuals
graphic_star = 2*Kmax+2:2*Kmax+1:TOT+1;

fig1 = figure('Position', get(0, 'Screensize'));
semilogy(residuals,'k','Linewidth',6);
hold on
semilogy(residuals_RTSA,'r--o','MarkerIndices',graphic_star,'Linewidth',6,'MarkerSize',10);
hold on
title(['Residuals, problem: ', QVIname],'Fontsize',22);
```

```
xlabel('iterations');
legend({'Solodov S.','Solodov S. + RTSA'},'LOCATION','
    SouthWest','Fontsize',22);

%
%% Saving data
savefig(fig1,['grafici/',QVIname, Method_name,'_Nmax_',
    num2str(Nmax),'_Kmax_', num2str(Kmax),'
    _bestlambda_RTSA','.fig'])
saveas(fig1, ['grafici/',QVIname, Method_name,'_Nmax_',
    num2str(Nmax),'_Kmax_', num2str(Kmax),'
    _bestlambda_RTSA','.jpg'])
save(['results_number_RTSA_alg/',QVIname, Method_name,'
    _Nmax_', num2str(Nmax), '_Kmax_', num2str(Kmax),'
    _bestlambda_RTSA','.mat'])
```

Listing 5.17: RegularizedTopologicalNguyenStrodiot.m

```
%% Nguyen-Strodiot implemented with restarted topological
     Shanks acceleration
%
% Restarted topological Shanks acceleration (RTSA)
% Alg from
% *Anderson type trasformations for systems of nonlinear
    equations*
% by Claude Branzinski, Stefano Cipolla, Michela Redivo-
    Zoglia, Yousef Saad
%
% QVI formulation (Latex notation used):
%     find x such that: g(x,x) \leq 0,
%                       M(x)x+v(x) = 0 and
%                       F(x)^T (y-x)\geq 0,
%                       for all y such that
%                       g(y,x)\leq 0 and M(x)y+v(x) = 0
%                       where
%                       F(x):\Re^{nVar}\to\Re^{nVar}
%                       g(y,x):\Re^{nVar}\times
%                              \Re^{nVar}\to\Re^{nIneq}
%                       M(x):\Re^{nVar}\to
%                            \Re^{nEq\times nVar}
%                       v(x):\Re^{nVar}\to\Re^{nEq}
%
%                       Note that some of the constraints
%                       g(y,x)\leq 0 and M(x)y+v(x) = 0
%                       may actually be independent of x.
%                       The constraints are always
%                       ordered so that these constraints
%                       independent of x are the first
%                       ones. For example
```

```
%                              g(y,x) = [ g1(y); g2(y,x) ]


%% Problem Definition
clear all;
close all;
clc
addpath('.../QVILIB_quadprog')
Method_name = '_Nguyen-Strodiot';
QVIname = 'RHS2A1_new';
QVIproblem = @RHS2A1_new;

% Generating data files for some large scale problems of
% QVILIB
% N.B. necessary only for RHS QVI type
% problems
QVILibGenData(QVIname)

% Initialization of the data defining the problem
QVIproblem(0)

% Starting point
number = 1;
x0 = startingPoints(QVIname,number);

% Function
F = @(x)QVIproblem(1,x);
nVar = size(F(x0),1);

% Equality constraints
Aeq = QVIproblem(6);
beq = @(x)QVIproblem(7,x);

% Inequality constraints
A = QVIproblem(4);
b = @(x)QVIproblem(5,x);

% Bound constraints
lb = QVIproblem(2);
ub = QVIproblem(3);

% Residuals
residuals_RTSA = [];      % residuals from the Nguyen-Strodiot_RTSA alg
residuals = [];           % residuals from the standard Nguyen-Strodiot


%% QVI_algorithm accelerated
% Algorithm Initialization
```

```
% Parameters
c = 0.5;
elle = 0.5;
gamma = 0.99;
ro = 1;            % ro >= 0
ro1 = 1/(1+ro);
mu = 0.5;              % mu > max(1/4,ro/(4*(1+ro-c)))
iter = 0;
options = optimoptions('quadprog','Algorithm','interior-point-convex', 'OptimalityTolerance',1e-20,'MaxIterations',500);

% Set number of outer loops
Nmax = 5;

% Set number of inner loops
Kmax = 7;

% Set the total number of cycles for the original sequence
TOT = Nmax*(2*Kmax+1);

% Set regularization parameter
info.lambdaRange=[1, 1e-14];
lambda_min = min(info.lambdaRange);
lambda_max = max(info.lambdaRange);

% Computing grid
lambdavec = [0, logspace(log10(lambda_min),log10(lambda_max),Kmax)];

% Main part
% Start the outer loop of the RTSA method
for i = 1:Nmax
    X(:,1) = x0;
    disp(['Inner iteration 1 of cycle ',num2str(i,'%d'),' completed']);

    % Start of the inner loop of the modified RTSA method
    for n = 1:2*Kmax+1
        iter = iter+1;

        % Performing 2*Kmax+1 Nguyen-Strodiot steps
        [x0,residuals_RTSA(iter)] = NguyenStrodiot_quadprog(x0,F,A,b,Aeq,beq,lb,ub,nVar,gamma,elle,ro,ro1,mu,c,options);
        X(:,n+1) = x0;
        disp(['Inner iteration ', num2str(n+1,'%d'), ' of cycle ', num2str(i,'%d'),' completed']);
    end
```

```
    % End of the inner loop

    warning off
    normvec = zeros(size(lambdavec));   % for the grid search
    memory_extrapolation_1 = zeros(size(x0,1),size(lambdavec,2));
    memory_extrapolation_2 = zeros(size(x0,1),size(lambdavec,2));
    param.y = X(:,end);

    % Grid search on lambda
    for h = 1:length(lambdavec)
        % extrapolation using differents values of lambda
        x_l = topologicalShanksTransformation_new(X,param.y,lambdavec(h));
        memory_extrapolation_1(:,h) = x_l;
        memory_extrapolation_2(:,h) = quadprog(eye(nVar),F(x_l)-x_l,A,b(x_l),Aeq,beq(x_l),lb,ub,x_l,options);
        normvec(h) = norm(memory_extrapolation_1(:,h)-memory_extrapolation_2(:,h))^2;
        disp(['Inner iteration for lambda ', num2str(h,'%d'), ' of cycle ', num2str(i,'%d'),' completed']);
    end

    % Choosing best lambda
    warning on
    [~, idx_min] = min(normvec);
    lambdamin = lambdavec(idx_min);
    info.lambdaUsed(i) = lambdamin;

    % Approximation of x*
    x0 = memory_extrapolation_1(:,idx_min);

    disp(['* Outer iteration ', num2str(i,'%d'), ' completed']);
    disp(' ');
end
% End of the outer loop

[x0,residuals_RTSA(TOT+1)] = NguyenStrodiot_quadprog(x0,F,A,b,Aeq,beq,lb,ub,nVar,gamma,elle,ro,ro1,mu,c,options);

%
%% QVI_alg basic method
% Reinitialize the starting point
X0 = startingPoints(QVIname,number);
```

```
% Nguyen-Strodiot (basic method)
for j = 1:TOT+1
    % In output, X0 is the new element x_j
    [X0,residuals(j)] = NguyenStrodiot_quadprog(X0,F,A,b,Aeq,beq,lb,ub,nVar,gamma,elle,ro,ro1,mu,c,options);
end


%
%% Graph of residuals
graphic_star = 2*Kmax+2:2*Kmax+1:TOT+1;

fig1 = figure('Position', get(0, 'Screensize'));
semilogy(residuals,'k','Linewidth',6);
hold on
semilogy(residuals_RTSA,'r--o','MarkerIndices',graphic_star,'Linewidth',6,'MarkerSize',10);
hold on
title(['Residuals with d3, problem: ', QVIname],'Fontsize',22);
xlabel('iterations');
legend({'Nguyen-Strodiot','Nguyen-Strodiot + RTSA'},'LOCATION','SouthWest','Fontsize',22);

%
%% Saving data
savefig(fig1,['grafici/',QVIname, Method_name,'_d3','_Nmax_', num2str(Nmax), '_Kmax_', num2str(Kmax),'_bestlambda_RTSA','.fig'])
saveas(fig1, ['grafici/',QVIname, Method_name,'_d3','_Nmax_', num2str(Nmax), '_Kmax_', num2str(Kmax),'_bestlambda_RTSA','.jpg'])
save(['results_number_RTSA_alg/',QVIname, Method_name,'_d3','_Nmax_', num2str(Nmax), '_Kmax_', num2str(Kmax),'_bestlambda_RTSA','.mat'])
```

Bibliography

[1] E. Allevi, R. Riccardia, M. Rocco, *Evaluating the carbon leakage effect on cement sector under different climate policies*, J. Cleaner Prod. 2016.

[2] G. Biagi, M. Castellani, M. Pappalardo, M. Passacantando, *Nonlinear Programming Techniques for Equilibria*, Springer, 2019.

[3] C. Brenzinski, M. Redivo-Zaglia, *Shanks Transformations and their Applications.*

[4] C. Brenzinski, S. Cipolla, M. Redivo-Zaglia, Y. Saad, *Anderson type transformation for systems of non-linear equations*, in preparation.

[5] C. Brenzinski, M. Redivo-Zaglia, Y. Saad, *Shanks sequence transformations and anderson acceleration*, SIAM Review, 60 (2018), pp. 646-669.

[6] F. Facchinei, C. Kanzow, *Generalized Nash equilibrium problems*, Ann. Oper. Res. 175, pp. 177-211 (2010).

[7] F. Facchinei, C. Kanzow, S. Sagratella, *Solving quasi-variational inequalities via their KKT conditions*, Springer, Math. Program. Ser. A (2014), pp. 369-412.

[8] F. Facchinei, C. Kanzow, S. Sagratella, *QVILIB: a library of quasi-variational inequality test problems*, Pac. J. Optim. 9 (2013), pp. 225-250.

[9] F. Giannessi, A. Maugeri, *Variational Analysis and Applications*, Springer, (2005) pp. 1101-1128.

[10] D. Hang, H. Zhang, G. Qian, L. Xu, *An improvement two-step method for solving generalized Nash equilibrium problems*, Eur. J. Oper. Res., 216 (2012), pp. 613-623.

[11] A.N. Iusem, B.F. Svaiter, *A variant of Korpelevich's method for variant inequalities with a new search strategy*, Optimization, 42 (1997), pp. 309-321.

[12] E.N. Khobotov, *Modification of the extragradient method for solving variational inequalities and certain optimization problems*, USSR Comput.Math.Phys., 27 (1987), pp. 120-127.

[13] G.M. Korpelevich, *The extragradient method for finding saddle points and other problems*, Matekon, 12 (1976), pp.747-756.

[14] T.T.V. Nguyen, J. J. Strodiot, V.H. Nguyen, P. T. Vuong, *An extragradient-type method for solving non-monotone quasi-equilibium problems*, Operations Research and Control System, Research Unit ORCOS.

[15] T.T.V. Nguyen, T.P.D. Nguyen, J. J. Strodiot, V.H. Nguyen, *A class of hybrid methods for quasi-variational inequalities*, Springer, Optimization Letters, 4 (2014), pp. 2211-2226.

[16] M.A. Noor, Y. Whang, N. Xiu, *Some new projection methods for variational inequalities*, Appl. Math. Comput., 137 (2003), pp. 423-435.

[17] M.A. Noor, Y.J. Whang, N. Xiu, *Projection iterative schemes for general variational inequalities*, J. Inequal Pure Appl. Math, 3(3) (2002).

[18] J. Outrata, M. Kocvara, *On a class of quasi-variational inequalities*, Optim. Methods Softw. 5, pp. 275-295 (1995).

[19] J. Outrata, J. Zowe, *A Newton method for a class of quasi-variational inequalities*, Comput. Optim. Appl. 4, 5-21 (1995).

[20] J.S. Pang, M. Fukushima, *Quasi-variational inequalities, generalized Nash equilibria and multileader-follower games*, Comput. Manag. Sci. 2, pp. 21-56 (2005).

[21] M. Passacantando, D. Ardagna, A. Salvi, *Service provisioning problem in cloud and multi-cloud systems*, INFORMS J. Comput. 2016, pp. 265-277.

[22] R. Riccardi, F. Bonenti, C. Avanzi, A. Gnudi, *The steel industry: A mathematical model under enviromental regulations*, Eur. J. Oper. Res. 2015, pp. 1017-1027.

[23] R.T. Rockafellar, *Convex Analysis*, Princeton University Press, Priceton (1970).

[24] D. Scieur, A. d'Aspremont, F. Bach, *Regularized nonlinear acceleration*, Mathematical Programming (2018).

[25] M.V. Solodov, B.F. Svaiter, *A new projection method for variational inequality problems*, SIAM J. Control Optim., 37 (1999), pp. 765-776.

[26] J. J. Strodiot, T. T. V. Nguyen, V. H. Nguyen *A new class of hybrid extragradient algorithms for solving quasi-equilibrium problems*, Springer Science+Business Media, 2011.

[27] Y. Wang, N. Xiu, C. Wang, *Unified framework of extragradient-type methods for pseudo-monotone variational inequalities*, J. Optim. Theory Appl., 111 (2001), pp. 641-656.

[28] Y. Wang, N. Xiu, C. Wang, *A new version of extragradient method for variational inequalities problems*, Comput. Math. Appl., 42 (2001), pp. 969-979.

[29] N. Xiu, Y. Wang, X. Zhang, *Modified fixed-point equations and related iterative methods for variational inequalities*, Comput. Math. Appl., 47 (2004), pp. 913-920.

[30] Z. Zhang, B. Qu, N. Xiu, *Some projection-like method for the generalized Nash equilibria*, Comput. Optim. Appl., 45 (2010), pp. 89-109.

[31] J. Zhang, B. O'Donoghue, S. Boyd, *Globally convergent type-i anderson acceleration for non-smooth fixed-point iterations*, arXiv preprint arXiv:1808.03971, (2018).