Technical University of Cluj-Napoca
North University Center at Baia Mare
Faculty of Sciences

PhD Thesis

New Implementations of Fixed Point Algorithms and Applications to Nonsmooth Optimization

Scientific Advisor:
Professor Dr. VASILE BERINDE

PhD Student:
ANDREI BOZANTAN

September 2013

Science is what we understand well enough to explain to a computer.
Art is everything else we do.

Donald E. Knuth

For Andra and Emma – I would have not achieved this without you.

CONTENTS

LIST OF FIGURES

MATHEMATICAL NOTATION

$\mathbb{R}^n$	is the real Euclidean n-dimensional vector space, with the usual operations of addition and scalar multiplication defined.
X, Y	denote generic metric or topological spaces, depending upon context.
v, x, y, z	denote vectors (points) from $\mathbb{R}^n$ or $\mathbb{R}^{n+1}$, depending upon context.
v_0, v_i, v_n	Are the elements of a set (family) of vectors; indexes of set elements start from 0, like the indexes of arrays in most of the modern programming languages.
$v_{[0]}, v_{[i]}, v_{[n]}$	denote the components of a vector v; $v_{[0]}, v_{[i]}, v_{[n]} \in \mathbb{R}$; indexes will start from 0.
$e_0, \ldots, e_n$	is the standard basis of the real Euclidean vector space; e_i is a column vector with the component $e_i = 1$ and the rest of components equal to 0.
$\|\|x\|\|$	represents the Euclidean norm of a vector $x \in \mathbb{R}^n, x = (x_{[0]}, \ldots, x_{[n-1]})$; $\|\|x\|\| = \sqrt{x_{[0]}^2 + \cdots + x_{[n-1]}^2}$.
$\langle x, y \rangle$	denotes the dot product (or inner product) $\langle x, y \rangle = \sum_{i=0}^{n-1} x_{[i]} y_{[i]}, \quad x, y \in \mathbb{R}^n$
$[v_0, \ldots, v_n]$	denotes a simplex with that specific vertices.
σ, τ	will usually denote some faces of a simplex S.
Δ^n	is the standard n-simplex with the vertices $e_i,\ i = 0, \ldots, n$.
$\mathcal{T}$	denotes a triangulation of a simplex.
$\overline{\text{co}}X$	denotes the convex hull of a set X, that is, the smallest convex set containing X.
$\mathcal{P}(X)$	denotes the power set of X, i.e., the set of all subsets of X.
F_T or $Fix(T)$	denotes the set of all fixed points of $T : X \to X$.

INTRODUCTION

Motivation and Objectives

Over the last 50 years or so, Fixed Point Theory has developed it self into a research domain and has been revealed as a very important, powerful and versatile tool in the study of nonlinear phenomena. Apart of the fundamental applications of Fixed Point Theory in nonlinear analysis, to problems in optimization, variational inequalities, complementarity problems and approximation theory, the fixed point techniques have received a lot of attention due to their extensive applications in many applied sciences to solve diverse problems in biology, chemistry, economics, engineering, game theory, physics, computer science, image recovery and signal processing, control theory, communications and geophysics.

If we refer to the consistent monograph Rus et al. (2008), Fixed Point Theory is one of the most dynamic area of research of the last 60 years, the dynamic of this topic is reflected, at least, by the following arguments:

- Over 120 books (monographs, lecture notes, proceedings) on fixed point theory and its applications;
- Over 12,000 papers on fixed point theory from 1940 until now;
- Almost 4,000 papers on fixed point theory only between 2000-2008;
- Except these theoretical books and papers, there are more than 2,000 books, monographs and proceedings and over 40,000 papers, which use the abstract theory of fixed point for various problems of pure, applied and computational mathematics.

The main reasons of this impressive development is the fact that the Fixed Point Theory approach offers - for any kind of problem we are tackling - not only information on the existence (and, sometimes, uniqueness) of the solution but also provides a constructive algorithm for approximating that solution.

Many algorithms have been developed in order to approximate the fixed points or the common fixed points for the great variety of mappings arising from the theoret-

ical and practical applications of fixed point theory we mentioned before. But, if we have a close look on this rich research work and try to classify the contributions into two classes, theoretical and applied, we shall notice that the great majority of contributions, even when dealing with concrete fixed point algorithms, are theoretically oriented.

Despite the extraordinary development of electronic computers in the last half of century, the implementations of many important algorithms in the field of fixed point theory are much behind the latest developments in the computer hardware, software and programming languages, and in some cases algorithms implementations date since the mainframe era, or are not accessible at all. It is also important to notice that numerical computing algorithms are no longer implemented only in FORTRAN and C programming languages, since a wide range of modern applications use numerical computing techniques in order to provide a better user experience.

Moreover, the process of using an existing old implementation of some algorithm is usually very tedious, if it is not already implemented in some popular computer algebra system. This process involves the following steps: try to obtain the source code from various sources (papers, books, online, e.g. Netlib (2013)), obtain a compiler and other programming tools, and maybe other necessary libraries, build the sources, read the documentation, if exists, try to figure out how to use the code in another program and understand the format for the input and output. This kind of work can be challenging even for professional computer programmers.

But, for most of the algorithms that were implemented in the past (mainly, in the seventies), the source code is not available at all and hence cannot be of help for interested users and, moreover, these specialised algorithms that have small area of applications in more complex problems are not supplied by the general routines offered by known computer algebra packages (Mathematica, Maple, Matlab etc.).

Starting from the above findings, the aim of this thesis is to partially fill these gaps, by achieving the following main objectives: to study some important fixed point algorithms and to create new and easy to use implementations of these fixed point algorithms; to test numerically the efficiency and robustness of the new implementations, on some relevant sample problems in nonlinear analysis and unconstrained optimisation theory. Another ambitious objective was proposed initially, to find possible improvements for some of these algorithms, in the sense of enlarging their area of applications, but unfortunately, this objective was only partially achieved.

Some initial implementations of the algorithms were realized using C++, which is one of the programming languages traditionally used for numerical computing algorithms (beside FORTRAN and C). But at some point I decided to also experiment

by using other programming languages. After a short experimentation with the modern C# language (Bozantan, 2010), I decided to implement the numerical computing algorithms in the very popular Javascript language. This decision allowed me to also create a much more accessible and easy to use graphical user interface for some of the implemented algorithms. A short report on this experiment is made in the conclusion chapter.

I also think that it is important to mention another more personal factor for my work on this subject. The origins of this PhD thesis can be traced back until my high-school studies, when I implemented a simple program to plot function graphs and different fractals. During my university studies I was interested in the compiler theory and I implemented a parser for mathematical expressions, which I later used for various projects, in the mathematics and computer science courses: function plotting, symbolic differentiation, computation and plotting of Lagrange interpolation polynomials, computation of solutions of ordinary differential equations using Runge-Kutta methods. During the studies for the master degree, I continued my work on numerical methods and building on the acquired experience I implemented a desktop application for computing fixed points using several iterative methods. As a conclusion, much of the work for this thesis comes as a natural continuation and improvement of my previous smaller projects.

Structure

The concepts of fixed point theory can be defined in various theoretical frameworks, and this thesis presents some new implementations of fixed algorithms having as theoretical background two famous fixed point theorems: the Banach fixed point theorem and the Brouwer fixed point theorem. As a result, the first two chapters of the thesis contain a mix theoretical notions and personal contributions related to the fixed point algorithms based on these two important fixed point theorems and the third chapter is dedicated to practical applications of the new implementations.

In the first chapter we present the new implementations of several fixed point iterative algorithms defined in the context of metrical fixed point theory and based on the Banach fixed point theorem. The first section (1.1) describes some basic theoretical notions: metric spaces (1.1.1), normed spaces (1.1.2), Hilbert spaces (1.1.3). Two important results are presented in this section: the Banach fixed point theorem, also known as the contraction mapping principle (1.1.4) and a similar result for weak contractions, due to Berinde (2004a) (1.1.5). The next section (1.2) presents first a review of the existing software for iterative fixed point methods (1.2.1) and then the details of the new Javascript implementation, together with personal contributions (1.2.2). This implementation is realized in a generic mode, so the same code is

reused for the implementation of several important iterative methods, as described in the following sections: the Picard iteration (1.3), the Krasnoselskii, Mann and Ishikawa iterations (1.4).

In the second chapter of the thesis we present the implementations of some fixed point algorithms defined in the Euclidean space $\mathbb{R}^n$, as particular case of the topological spaces. The first section (2.1) presents several basic notions in Euclidean geometry (2.1.1), and then describes the geometrical objects, simplices (2.1.2) and triangulations (2.1.3), and the elementary pivoting steps (2.1.3) used in the algorithms presented later in this chapter. As an example the Freudenthal triangulation is presented (2.1.4). The second section (2.2) presents the famous Banach fixed point theorem (2.2.1), its generalization for set-valued mappings, the Kakutani fixed point theorem (2.2.2), and several equivalent forms of the fixed point problem (2.2.3). The Sperner lemma (section 2.3) is presented, as an important connection between the Brouwer fixed point theorem and the simplicial fixed point algorithm initially developed by Scarf. The following section (2.4) of this chapter will describe the first fixed point algorithm for approximating a Brouwer fixed point, proposed by Scarf (1967) (2.4.1), then some details about the implementation of these simplicial fixed point algorithms (2.4.2) and the new Javascript implementation as a personal contribution (2.4.3). The last section (2.5) of this chapter presents an advanced version of the simplicial fixed point algorithms, the piecewise linear homotopy algorithms. The following main points, which are used in the implementation of the algorithm, are touched in this section: homotopy methods (2.5.1), piecewise linear approximations (2.5.2), a description of a generic PL algorithm (2.5.3) and some details about the J_3 refining triangulation of $\mathbb{R}^{n+1}$ (2.5.4). The section concludes with the description of the new implementation of the algorithm (2.5.5) proposed by Eaves and Saigal (1972).

The third chapter is dedicated to applications and is composed of two sections. The first section (3.1) presents practical applications of the new Javascript implementations of the Picard, Krasnoselskii, Mann and Ishikawa iterations: computing fixed points with the Picard iteration, comparing Krasnoselskii iterations, the usage and usefulness of the mixed error test, the detection of cyclic iterations. The second section (3.2) presents an empirical study of the efficiency and robustness of the new implementation of the PL homotopy algorithm, by solving several unconstrained optimization problems and comparing the results with some of the well known and widely used methods in optimization: the Newton's method, Broyden-Fletcher-Goldfarb-Shanno, conjugate gradient method, and nonlinear conjugate gradient method.

Finally, the last chapter of the thesis presents the conclusions and some possible directions of research for the future.

Personal contributions

The main personal contributions come in the form of the new Javascript implementations of the fixed point algorithms and the applications on some relevant sample problems, as detailed below.

In the first chapter the personal contributions are: the generic implementation of the iterative fixed point algorithm (also referred as FIXPOINT), based on the Banach fixed point theorem (section 1.2.2), which is used as a base for the other new implementations of fixed point iterations: Picard, Krasnoselskii, Mann and Ishikawa iterations (see sections 1.3.2, 1.4.2). The final section of the first chapter (3.1) which uses the new implementations in practical applications is another personal contribution. We also mention some new features of the FIXPOINT program, which at this time are not present in other implementations, like: the usage of a mixed error test for testing the convergence of the sequence of successive approximations, the additional check for cycle detection and the built in comparison function, for details see 1.2.2.

It is important to mention that the early versions of FIXPOINT were used for most of the results in Chapter 9 of Berinde (2007) - Error analysis of fixed point iteration procedures, and also for the numerical experiments in the comparative study of the rate of convergence of fixed point iteration procedures, that were reported in Berinde and Păcurar (2007), thus opening new research directions. By inspecting the empirical results obtained by means of FIXPOINT for several fixed point iterative methods, some theoretical results have been also inferred and proved in Berinde (2004b), Berinde and Berinde (2005) and continued by other authors: Babu and Prasad (2006, 2007) Duong (2012), Hussain et al. (2012, 2013, 2011), Kumar (2013), Olaleru (2007, 2009), Păcurar (2009a,b, 2010a,b, 2011, 2012), Phuengrattana and Suantai (2012), Popescu (2007), Xue (2008), Rhoades and Xue (2010).

Not last on this list is the complementary web-based application, which provides an easy way to use these implementations and additional features like the interactive visualisations, cobweb plots for the Picard iteration and a new intuitive visualisation for the Krasnoselskii iteration.

In the second chapter the personal contributions are the new implementation of the simplicial fixed point algorithm of Kuhn (1968), see 2.4.3, and the new implementation of the PL homotopy method described by Eaves (1972) and Eaves and Saigal (1972), see 2.5.5. The new implementation of the PL homotopy algorithm is then used in section 3.2 to solve several unconstrained non-smooth optimization problems in order to obtain new results about the robustness and efficiency of the algorithm and of the new implementation. As shown by these numerical experiments done on a set of classic test functions in optimization theory, the PL homo-

topy algorithm appears to be more reliable than the classical Newton's method and some other important methods (Broyden-Fletcher-Goldfarb-Shanno, conjugate gradient method, and nonlinear conjugate gradient method) for finding local minima. The main advantages of the PL homotopy method, advantages clearly illustrated by the numerical results, are that the method doesn't require smoothness of the underlying map and it can be also successfully applied when it is difficult to choose a suitable starting point for the iterative methods because there is no available a priori knowledge regarding the solutions of the system to be solved.

Acknowledgments

Firstly, I would like to express my deepest gratitude to my Ph. D. adviser, Professor Doctor Vasile Berinde for his excellent guidance and endless patience. He helped me continue my research when my morale was down and I would not have finished my thesis without his constant encouragement.

I would like to show my appreciation to everyone in the Departament of Mathematics and Computer Science at the North University of Baia Mare who guided me during the bachelor and master degree studies. Special thanks also to my doctoral studies colleagues for their support.

Finally, I would like to thank my wonderful daughter for accepting my long hours of work and my wife for her constant nagging, understanding and patience.

1. NEW IMPLEMENTATIONS OF FIXED POINT ALGORITHMS IN METRIC SPACES

1.1 BASIC NOTIONS IN METRICAL FIXED POINT THEORY

This section will present the basic notions in metrical fixed point theory, following mainly the presentations of Berinde (2007). For more details the following bibliography items can be consulted: Aliprantis and Border (2006), Rus et al. (2008), Istratescu (2002), Rus (1979).

1.1.1 Metric Spaces

(1.1.1) Definition Let X be a nonempty set and $T : X \to X$ be a selfmap. We say that $x \in X$ is a **fixed point** of T if

$$T(x) = x$$

and we denote by F_T or $Fix(T)$ **the set of all fixed points** of T.

(1.1.2) Example If $T : \mathbb{R} \to \mathbb{R}$, $T(x) = 4x^2 - 3x - 8$, then $F_T = \{-1, 2\}$

(1.1.3) Definition Let X be any set and $T : X \to X$ a selfmap. For any given $x \in X$ we define $T^n(x)$ inductively by $T^0(x) = x$ and $T^{n+1}(x) = T(T^n(x))$. We say that $T^n(x)$ is the n^{th} iterate of x under T. To simplify notation we will use Tx instead of $T(x)$. The mapping T^n, for $n \geq 1$ is called the n^{th} **iterate** of T.

(1.1.4) Definition A **metric** (or **distance**) on a set X is a function $d : X \times X \to \mathbb{R}_+$ satisfying the following properties:

1. $d(x, y) = 0 \Leftrightarrow x = y$ (discrimination or separation axiom).
2. $d(x, y) = d(y, x)$, $\forall x, y \in X$ (symmetry).
3. $d(x, y) \leq d(x, z) + d(y, z)$, $\forall x, y, z \in X$ (triangle inequality).

The pair (X, d) is called a **metric space**.

(1.1.5) Example Given any set X, the discrete metric is defined by:

$$d = \begin{cases} 0, & \text{if } x = y \\ 1, & \text{if } x \neq y \end{cases}$$

This is a simple but important example of a metric distance which can be applied to all non-empty sets.

(1.1.6) Definition Let (X, d) be a metric space and $\{x_n\}$ a sequence in X. The sequence $\{x_n\}$ is **convergent** to $a \in X$, if for any $\epsilon > 0$, there exists n_0 (depending on ϵ) such that $d(x_n, a) < \epsilon$, for any $n \in \mathbf{N}, n \geq n_0$.

(1.1.7) Definition Let (X, d) be a metric space and $\{x_n\}$ a sequence in X. The sequence $\{x_n\}$ is a **Cauchy sequence** (or **fundamental**), if for any $\epsilon > 0$, there exists n_0 (depending on ϵ) such that $d(x_n, x_{n+p}) < \epsilon$, for all $n \in \mathbf{N}, n \geq n_0$, and any $p \in \mathbf{N}^*$. In other words, given any small positive distance ϵ, all the elements of the sequence, excluding a finite number of them, are at a distance smaller than ϵ from each other.

(1.1.8) Remark In a metric space, any convergent sequence is also a Cauchy sequence.

(1.1.9) Definition A metric space (X, d) is called **complete metric space** if every Cauchy sequence is convergent.

1.1.2 Normed spaces

(1.1.10) Definition Let E be a real (or complex) linear vector space. A **norm** on E is a mapping $||\cdot|| : E \times E \to \mathbb{R}_+$ having the following properties:

1. $||x|| = 0 \Leftrightarrow x = 0$, the null element of E;
2. $||\lambda x|| = |\lambda| \cdot ||x||$, for any $x \in E$ and any scalar λ;
3. $||x + y|| \leq ||x|| + ||y||$, for all $x, y \in E$ (the triangle inequality).

The pair $(E, ||\cdot||)$ is called a **normed space** (or **linear space**).

(1.1.11) Remark If $||\cdot||$ is a norm on the (linear) vector space E, then $d : E \times E \to \mathbb{R}_+$, given by

$$d(x, y) = ||x - y||, \quad x, y \in E$$

is a distance on E. This shows that any normed space can be always regarded as a metric space with respect to the distance induced by the norm.

(1.1.12) Definition A normed space which is complete as metric space is called **Banach space**.

(1.1.13) Definition A Banch space $(E, ||\cdot||)$ is called **uniformly convex** if, given any $\epsilon > 0$, there exists $\delta > 0$ such that for all $x, y \in E$ satisfying $||x|| \leq 1$, $||y|| \leq 1$, and $||x - y|| \geq \epsilon$, we have

$$\frac{1}{2}||x + y|| < 1 - \delta.$$

1.1.3 Hilbert spaces

(1.1.14) Definition Let H be a real vector space. An **inner product** is a functional $\langle \cdot, \cdot \rangle : H \times H \to \mathbb{R}$ satisfying:

1. $\langle x, x \rangle \geq 0$, for all $x \in H$, and $\langle x, x \rangle = 0$ if and only if $x = 0$, the null vector in H;
2. $\langle x, y \rangle = \langle y, x \rangle$, for all $x, y \in H$;
3. $\langle ax + by, z \rangle = a \langle x, z \rangle + b \langle y, z \rangle$, for each $x, y, z \in H$ and all $a, b \in \mathbb{R}$.

The pair $(H, \langle \cdot, \cdot \rangle)$ is called **prehilbertian space**.

(1.1.15) Lemma If $(H, \langle \cdot, \cdot \rangle)$ is prehilbertian space, then the function $x \to \langle x, x \rangle^{1/2}$ is a norm on H, called the **norm induced by the inner product**.

(1.1.16) Definition A prehilbertian space is called **Hilbert space** if it is a Banach space (that is, a complete metric space) with respect to the metric corresponding to the norm induced by the inner product.

(1.1.17) Remark Any Hilbert space is a uniformly convex Banach space.

1.1.4 Contraction Mapping Principle

(1.1.18) Definition Let (X, d) be a metric space. A mapping $T : X \to X$ is called **Lipschitz continuous** if there exists a constant $L > 0$ such that

$$d(Tx, Ty) \leq L \cdot d(x, y), \quad \forall x, y \in X$$

Any such value L is called **Lipschitz constant**.

(1.1.19) Definition Let $T : X \to X$ be a Lipschitz continuous mapping with the constant L.

- If $0 \leq L < 1$ then T is called **contraction** or more precisely L-contraction.
- If $L = 1$ then T is called **nonexpansive**.
- If $d(Tx, Ty) < d(x, y), \forall x, y \in X, x \neq y$ then T is called **contractive**

(1.1.20) Example $f : \mathbb{R} \to \mathbb{R}$, $f(x) = \frac{x}{5} + 2$ is Lipschitz continuous and it is also a contraction.

(1.1.21) Example $f : \mathbb{R} \to \mathbb{R}$, $f(x) = x^2$ is not Lipschitz continuous.

(1.1.22) Theorem (Banach, 1922) Let (X, d) be a complete metric space and $T : X \to X$ be a given contraction. Then T has a unique fixed point x^*, and the sequence $\{T^n(x)\}_{n\in\mathbf{N}}$ converges to x^* for each $x \in X$.

Proof. (Palais, 2007) We will show first that the fixed point is unique. Since T is a contraction mapping we have

$$d(Tx, Ty) \leq Ld(x, y), \ \forall x, y \in X, \text{ with } 0 < L < 1.$$

By induction we also have

$$d(T^n x, T^n y) \leq L^n d(x, y).$$

By the triangle inequality

$$d(x, y) \leq d(x, Tx) + d(Tx, Ty) + d(Ty, y),$$

so

(1.1.23) $$(1 - L)d(x, y) \leq d(x, Tx) + d(Ty, y).$$

Since $L < 1$, we have

$$d(x, y) \leq \frac{1}{1 - L}(d(x, Tx) + d(y, Ty)), \forall x, y \in X$$

So, in particular, if x and y are fixed points of T we have $d(x, y) = 0$, that is a contraction mapping can have at most one fixed point.

We will show next that the sequence $\{T^n(x)\}_{n\in\mathbf{N}}$ of iterates of x under T is a Cauchy sequence. We consider $T^n x$ and $T^m x$, where $m = n + p$ and $p > 0$. From **1.1.23** we have

$$\begin{aligned} d(T^n x, T^m x) &\leq \frac{1}{1 - L}(d(T^n x, T^n(Tx)) + d(T^m x, T^m(Tx))) \\ &\leq \frac{L^n + L^m}{1 - L} d(x, Tx). \end{aligned}$$

Since $K < 1$, $K^n \to 0$, so $d(T^n x, T^m x) \to 0$ as $n \to \infty$, that is, the sequence $\{T^n(x)\}_{n\in\mathbf{N}}$ is a Cauchy sequence.

Since (X, d) is a complete metric space, the sequence $\{T^n(x)\}_{n\in\mathbf{N}}$ converges to some point $x^* \in X$, which clearly is a fixed point of T. □

(1.1.24) Definition Let E be an arbitrary real Banach space. A mapping T with domain $D(T)$ and range $R(T)$ in E is called

- **strong pseudocontraction** if there exists $k > 0$ such that for all $x, y \in D(T)$ there exists $j(x,y) \in J(x-y)$ such that

$$\langle (I-T)x - (I-T)y, j(x-y)\rangle \geq k \cdot ||x-y||^2$$

- **pseudocontractive** if for each $x, y \in D(T)$ there exists $j(x-y) \in J(x-y)$ such that

$$\langle (I-T)x - (I-T)y, j(x-y)\rangle \geq 0$$

where J is the normalized duality mapping.

1.1.5 Weak Contractions

(1.1.25) Definition (Berinde, 2007) Let (X,d) be a metric space. An operator $T : X \to X$ is called **weak contraction** if there exist two constants $L \in [0,1)$ and $\delta \geq 0$ such that:

(1.1.26) $$d(Tx,Ty) \leq Ld(x,y) + \delta d(y,Tx), \quad \forall x,y \in X$$

(1.1.27) Theorem (Berinde, 2004a) Let (X,d) be a complete metric space and $T : X \to X$ an weak contraction with constants $L \in [0,1)$ and $\delta \geq 0$. Then:

1. the Picard iteration $\{x_n\}_{n\geq 0}$ converges to some $x^*(x_0) \in F_T$ for any $x_0 \in X$;
2. for any $x \in X$ we have that:

(1.1.28) $$d(x,x^*(x)) \leq \frac{1}{1-L} d(x,Tx);$$

3. The following a priori and a posteriori error estimates hold:

(1.1.29) $$d(x_n,x^*(x_0)) \leq \frac{L^n}{1-L} d(x_0,x_1), \quad n = 1,2,\ldots$$

(1.1.30) $$d(x_n,x^*(x_0)) \leq \frac{L}{1-L} d(x_{n-1},x_n), \quad n = 1,2,\ldots$$

Proof. 1) Let $x_0 \in X$ be arbitrary and $\{x_n\}_{n\geq 0}$ be the Picard iteration of T starting from x_0. For $n \in \mathbf{N}$ we have that

$$d(x_n,x_{n+1}) = d(Tx_{n-1},Tx_n)$$

which by 1.1.26 implies

(1.1.31) $$d(x_n,x_{n+1}) < Ld(x_{n-1},x_n), n \geq 1.$$

By induction we get that

$$d(x_n,x_{n+1}) \leq L^n d(x_0,x_1), n \geq 0,$$

which for $p \geq 1$ implies:

(1.1.32) $$d(x_n, x_{n+p}) \leq L^n \frac{1-L^n}{1-L} d(x_0, x_1).$$

Since $L \in [0,1)$, by letting $n \to \infty$ in **1.1.32** we obtain that $\{x_n\}_{n\geq 0}$ is a Cauchy sequence. Since (X,d) is complete, $\{x_n\}_{n\geq 0}$ is also convergent. We denote:

$$x^*(x_0) = \lim_{n\to\infty} x_n.$$

Then:

$$\begin{aligned} d(x^*(x_0), Tx^*(x_0)) &\leq d(x^*(x_0), x_{n+1}) + d(x_{n+1}, Tx^*(x_0)) \\ &= d(x^*(x_0), x_{n+1}) + d(Tx_n, Tx^*(x_0)), \end{aligned}$$

which by **1.1.26** implies

$$d(x^*(x_0), Tx^*(x_0)) \leq (1+\delta) d(x^*(x_0), x_{n+1}) + L d(x_n, x^*(x_0)), n \geq 0.$$

Letting $n \to \infty$ in the previous relation we obtain

$$d(x^*(x_0), Tx^*(x_0)) = 0,$$

that is, $x^*(x_0)$ is a fixed point for T.

2) For any $x \in X$ we obtain by **1.1.26** that

$$d(x, x^*(x)) \leq d(x, Tx) + d(Tx, Tx^*(x)) \leq d(x, Tx) + L d(x, x^*(x))$$

which leads to **1.1.28**.

3) The a priori estimate **1.1.29** is obtained from **1.1.32** by letting $p \to \infty$. From **1.1.31** we obtain by induction that

$$d(x_{n+k}, x_{n+k+1}) \leq L^k d(x_{n-1}, x_n),\ k, n \in \mathbf{N}.$$

Then similarly to deriving **1.1.32** we get that

$$d(x_n, x_{n+p}) \leq L \frac{1-L^n}{1-L} d(x_{n-1}, x_n), n \geq 1, p \geq 1.$$

Letting $p \to \infty$ in the previous relation we obtain the a posteriori estimate **1.1.30**. □

1.2 A GENERIC FIXED POINT ITERATION ALGORITHM

1.2.1 Fixed Point Related Software

In this section we will analyse existing software alternatives for computing fixed points using iterative numerical methods and two different categories of software will be described: specialized software packages and general purpose computer algebra systems (CAS). Some parts and ideas of this section are included also in Bozantan and Berinde (2014a) and Bozantan (2014a). We start our description with the CAS category and we will analyse the features related to fixed point iterations, offered in some of the most used and well known general purpose CAS: Mathematica, Maple and Matlab.

From all the software analysed in this section, including the category of specialized software, **Maple** offers the most flexible and feature rich implementation for a fixed point iteration, in the form of the built in command **FixedPointIteration**, included in the Student[NumericalAnalysis] subpackage. The command will numerically approximate the roots of a real function f, by converting the problem to a fixed point problem and then using the Picard fixed point iteration with the supplied initial approximation for the root. The command can be invoked using one of the following forms:

- FixedPointIteration($f, x = a$, opts)
- FixedPointIteration(f, a, opts)

The arguments have the following significance:

- f - is an expression in the variable x representing a continuous function
- x - specifies the independent variable of f
- a - is the initial approximation of the root
- opts - optional arguments in the form keyword=value

Some of the most important other options for the command are:

- fixedpointiterator = expression - specifies directly the expression to be used in the fixed point iteration; if this option is present, the first argument, f, must be omitted.
- tolerance = value - specifies the error tolerance of the approximation
- stoppingcriterion = value - specifies the criterion that the approximation must meet before stopping the iteration, and can have one of the following values:
  - relative - $\frac{|x_n - x_{n-1}|}{|x_n|} < tolerance$
  - absolute - $|x_n - x_{n-1}| < tolerance$
  - function_value - $|f(x_n)| < tolerance$
- maxiterations = value - specifies the maximum iterations to perform, if the error tolerance is not achieved

- output = value - specifies the type of the return value of the command, and it can have on of the following values:
  - value - returns just the final numerical approximation of the root
  - sequence - returns the sequence of intermediate approximations produced by the fixed point iteration
  - plot - returns a plot of f with each iterative approximation shown
  - animation - returns an animation showing the iterations of the root approximation process
  - information - returns detailed information about the iterative approximations of the root of f

There are many other options available, used to control the plotting functionality of the command. More details about the Maple FixedPointIteration command are available on the Maple Help web page http://www.maplesoft.com/support/help/Maple/view.aspx?path=Student/NumericalAnalysis/FixedPointIteration. As an example we use the Maple commands to solve the problem $x^2 - x - 2 = 0$ and to produce a plot of the fixed point iteration which are displayed in figure 1.1.

In **Mathematica** there are available two built in functions related to fixed point iterations: **FixedPoint** and **FixedPointList**. Both functions use the Picard iteration in order to compute a fixed point and have the same parameters, with the difference that FixedPoint returns only the final value, while FixedPointList generates a list with all the intermediate values. The FixedPoint function can be called with using one of the following forms:

- FixedPoint[f, expr] - where expr is the initial approximation.
- FixedPoint[f, expr, n] - stops after at most n steps.
- FixedPoint[f, expr, ..., SameTest -> s] - a custom test, s, can be used to check if to consecutive results are equal

More details and examples are available online, on the Wolfram Mathematica Documentation website http://reference.wolfram.com/mathematica/ref/FixedPoint.html. In Mathematica there is no built in functionality for plotting the fixed point iteration, like in Maple, but this can be achieved using few lines of code, as shown below.

```
g = Function[x, 1 + 2/x];
fpl = FixedPointList[g, 1.0];
fpCoords = Flatten[Table[
    {{fpl[[k]], fpl[[k]]}, {fpl[[k]], fpl[[k + 1]]}},
    {k, 10}],
  1];
iterationPlot = ListPlot[fpCoords,
  AxesOrigin -> {0, 0},
  PlotRange -> {{0, 4}, {0, 4}},
  Joined -> True, PlotStyle -> {Blue, Dashed}];
```

```
11 functionPlot = Plot[{x, g[x]}, {x, 0, 4}, PlotStyle -> {Thick}];
12 Show[iterationPlot, functionPlot]
```

Listing 1.1 – Fixed point iteration and plot in Mathematica

with Student[*NumericalAnalysis*] :

$$\textit{FixedPointIteration}\left(\text{fixedpointiterator} = 1 + \frac{2}{x},\ \text{x} = 1.0,\ \text{maxiterations} = 16,\ \text{output} = \text{sequence}\right)$$

1.0, 3.000000000, 1.666666667, 2.200000000, 1.909090909, 2.047619048, 1.976744186, 2.011764706, 1.994152047, 2.002932551, 1.998535871, 2.000732601, 1.999633834, 2.000183117, 1.999908450, 2.000045777

$$\textit{FixedPointIteration}\left(\text{fixedpointiterator} = 1 + \frac{2}{x},\ \text{x} = 1.0,\ \text{output} = \text{plot},\ \text{maxiterations} = 16,\ \textit{view} = [0..5, -1..4]\right)$$

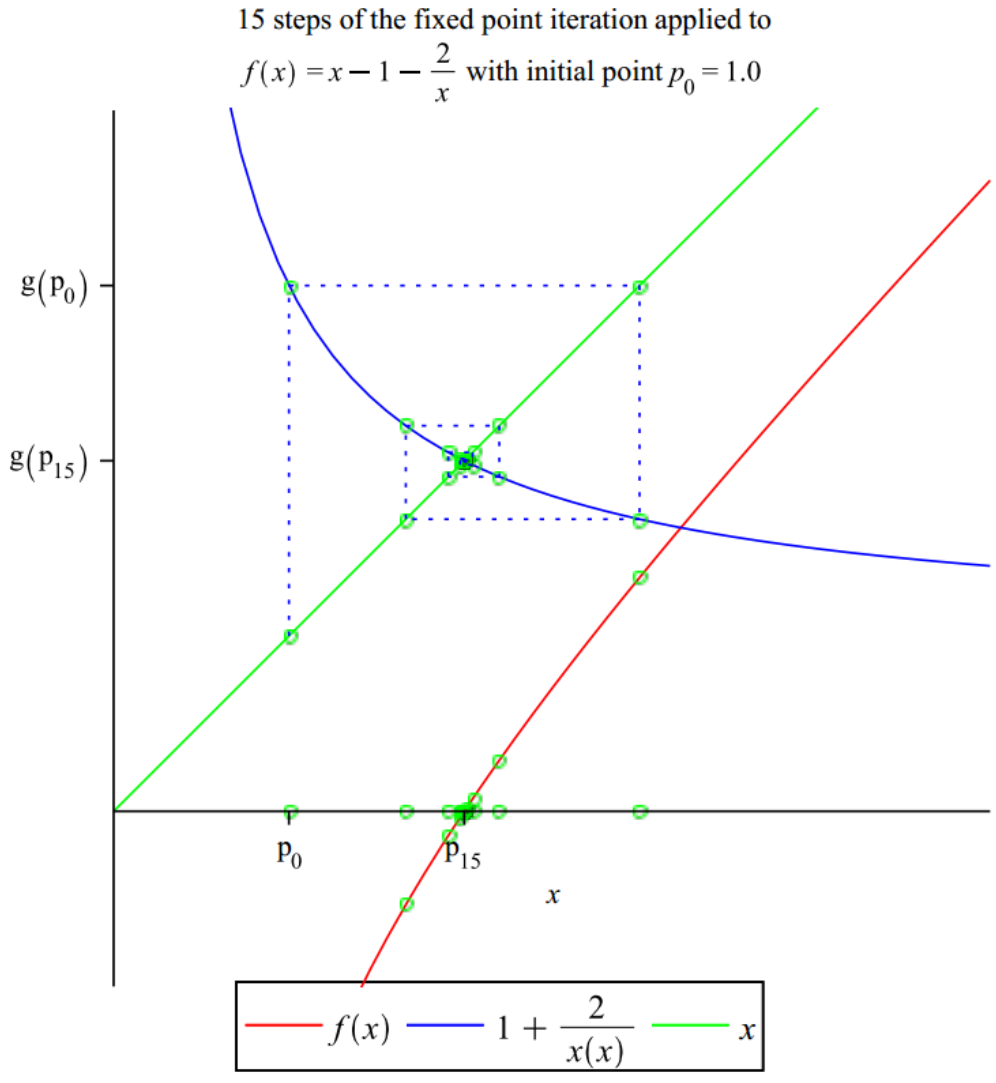


Figure 1.1 – Fixed point iteration commands in Maple

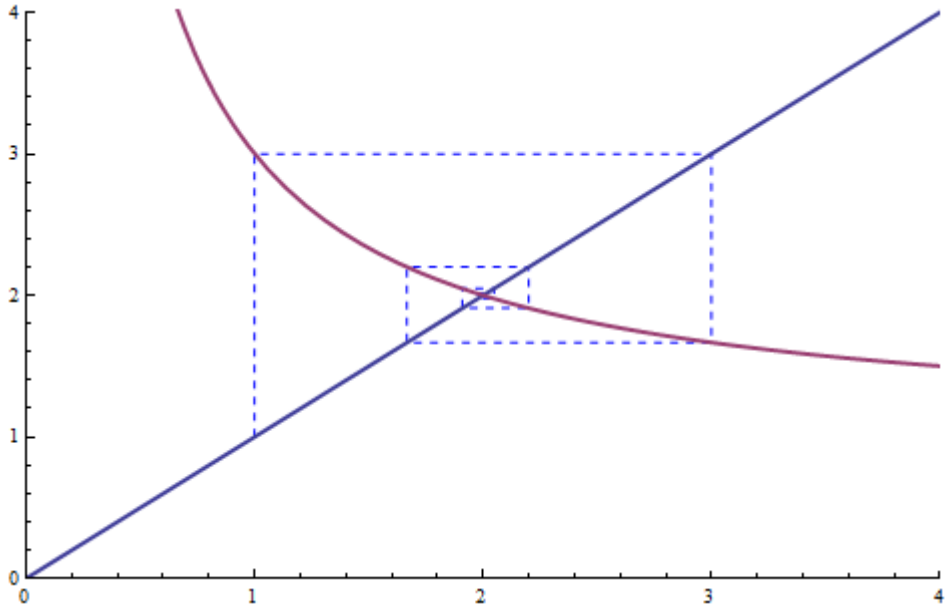


Figure 1.2 – Mathematica plot: fixed point iteration applied to $f(x) = 1 + \frac{2}{x}$ with initial point $x_0 = 1.0$

Matlab and MathCAD do not offer any built in functions related to fixed point iterations, but fixed point computation is possible using custom written functions, as demonstrated by the following Matlab listing.

```
function [p0,err,P] = fixpt(g, x0, tol, max1)
  x = x0; %initial guess
  P(0) = x0;
  xold = x;
  n = 1; %iteration counter
  while n < max1
    x = feval(g, x);
    P(n) = x;
    if abs(x-xold) < tol
      break;
  end
  xold = x;
  n = n + 1;
end
```

Listing 1.2 – Picard fixed point iteration in Matlab

As alternatives to the CAS software there are several other basic implementations of the Picard fixed point iteration which can be used, from which we mention the following:

- EasyNumerics is a desktop application which, beside other numerical algorithms, has an implementation of the Picard fixed point iteration together with some basic plotting of the iteration. It is available for download from

http://www.metu.edu.tr/~csert/EasyNumerics/.

- Fixed Point Iteration Java Applet is available online at http://www.csulb.edu/~wziemer/FixedPoint/FixedPoint.html and has another implementation of fixed point iteration plotting.
- other simple implementations of the Picard fixed point iterations are available online at:
  - http://maccery.com/maths/#fixed-point
  - http://planetcalc.com/2824/
  - http://cs.laurentian.ca/badams/numeric/javascript/fpoint.htm

The above research indicates that all existing implementations for fixed point iterations have rather basic functionality, excepting Maple, which also has a good cobweb diagram potting and animation. Another interesting point is that even the implementations in well known CAS like Maple and Mathematica are lacking advanced features like oscillation detection (e.g. the case of logistic map, $\lambda x(1-x)$), appropriate convergence tests for practical applications and comparison of different iterations. Also, there are no available implementations for other fixed point iterative methods like Krasnoselskii, Mann and Ishikawa. Such implementations are possible in CAS using custom written code, but, beside expensive licenses, this requires at least some familiarity with the corresponding CAS programming language and functions.

1.2.2 Details of the New Javascript Implementation

The new Javascript implementation contains implementations for several fixed point iterative methods for real valued functions: Picard, Krasnoselskii, Mann, Ishikawa. The implementation of the iteration algorithms for real valued functions are similar, and are based on the generic fixed point iteration method, with the only difference being the operator used in the iteration: the input function is wrapped in a special operator which is used by the generic implementation. The implementation is in the form of a Javascript function, which accepts the following parameters:

- f - the operator for which the approximate fixed point will be computed, specified as a Javascript function; in order to support the Mann and Ishikawa iterations using the same code, the operator is called at each iteration step with two arguments: the current approximation (x) and the iteration step (n, starting from 1).
- $x0$ - the real number which specifies the initial guess used to start the algorithm;
- *options* - a Javascript object which can contain additional optional arguments used to control the behaviour of the algorithm, as described below;
- *options.maxError* - specifies the maximum error to be accepted for the ap-

proximation of the fixed point; this value is used in the convergence test, as described below;

- *options.convergenceTest* - this parameter is a Javascript function which will be used by the algorithm to perform the convergence test for the sequence of successive approximations; using this function provides the same flexibility as for the Mathematica implementation; more details about how the convergence test is performed are provided below;
- *options.maxSteps* - the maximum number of steps to be executed before stopping the algorithm, in the case that desired precision was not achieved for the approximation;
- *options.checkCycles* - by default, the implementation of the algorithm is also checking for cycles of the iteration (oscillations), but this increases the complexity of the algorithm to $O(n^2)$; if greater performance is required, this option can be used to disable this additional check.

The function will return the following values (wrapped in a Javascript object):

- *xn* - the approximate fixed point obtained by the iteration;
- *values* - an array containing all the intermediate approximations (including the initial approximation x_0);
- *numSteps* - the number of iterations performed by the algorithm;
- *errorMessage* - it will contain a short description in case that the iteration is not successful (iteration is not convergent, the maximum number of iterations is reached or an undefined numerical operation is performed, e.g. $0/0$).

Next we will describe some details of the performed convergence test. For all the computations, the implementation of the algorithm is using IEEE 754 double precision floating point numbers, as most of the modern numerical computing software (some of the CAS, like Mathematica and Maple, also provide the possibility to use arbitrary precision arithmetic). Due to the precision limitations of the IEEE 754 floating point format, using a simple absolute error test for convergence is not enough for all the situations, so this implementation also provides the possibility to use different convergence tests, as appropriate to each problem. By default the implementation provides an absolute error test, a relative error test and a mixed error test. If another custom test is used, the *convergenceTest* function will be invoked after each iteration step and must return *true* when the sequence is considered convergent and $false$ otherwise. The arguments for the custom convergence test function are described below :

- *maxError* - the maximum accepted error, which is the value specified in *options.maxError* parameter;
- x - the approximation obtained after the last iteration step
- $xprev1, xprev2$ - the approximations obtained in the previous two iteration

steps (these are useful for a mixed error test);

- *values* - the complete sequence of approximations, which can be used for a more complex convergence test;

The new implementation also offers an additional helper function fixpoint.compare which can be used to generate a comparison table of several fixed point iterations. The result of the comparison function can be obtained as text format or as LaTeX format, for easy inclusion in scientific papers.

Due to the considerable size of the complete code, we present below just a simplified and incomplete version of the code, which shows the main loop corresponding to the iterative method. For further details about the algorithm implementation, the complete version of the code, which include the Picard, Krasnoselskii, Mann and Ishikawa iterations, the helper comparison function, the implementation of several error tests (absolute, relative and mixed error tests) and a suite of validation tests and examples is available online at http://algonum.appspot.com/fixpoint.

```
function iterate(f, x0, options)
{
  var res = {};
  // helper variables
  var values = [x0];
  var x = x0;
  var xprev1 = x0;
  var xprev2 = x0;
  var n = 0;
  var stop = false;

  // algorithm code
  while (!stop)
  {
    // perform an iteration step
    x = f(x, n);
    // store last result
    values.push(x);
    // increase iterations number
    n++;
    // perform the specified convergence test and stop if it is successful
    if (convergenceTest(options.maxError, x, xprev1, xprev2, values))
    {
      stop = true;
    }
    // check max number of iterations
    else if (n === options.maxSteps)
    {
      res.errorMessage = 'Maximum number of iterations was reached.';
      stop = true;
    }
```

```
32      // check for overflows and undefined operations
33      if (!Number.isFinite(x))
34      {
35        res.errorMessage = 'Iteration is divergent (numerical error).';
36        stop = true;
37      }
38      // check if the sequence already contains the current value
39      else if (findCycles())
40      {
41        res.errorMessage = 'Iteration is divergent (cycle detected).';
42        stop = true;
43      }
44      // save the values for the last two iterations
45      // these may be used for the convergence test
46      xprev2 = xprev1;
47      xprev1 = x;
48    }
49
50    // fill result
51    res.values = values;
52    res.numSteps = n;
53    res.x0 = x0;
54    if (res.errorMessage === undefined)
55    {
56      res.xn = x;
57    }
58
59    return res;
60  }
```

Listing 1.3 – Simplified (incomplete) implementation of the generic fixed point iterative method

1.3 PICARD ITERATION

This section will present the complete statement of the contraction mapping principle and some other theoretical details regarding the Picard iteration, following the presentations of Berinde (2007), and in the second part the new Javascript implementation of the Picard iteration.

1.3.1 Theoretical Background

We will present first the complete version of the contraction mapping principle, which was presented in a simplified version in section 1.1 (theorem **1.1.22**).

(1.3.1) Theorem (Banach Theorem or Contraction mapping principle) Let (X, d) be a complete metric space and $T : X \to X$ be a given L-contraction, that is, an operator satisfying:

(1.3.2) $$d(Tx, Ty) \leq Ld(x, y), \quad \forall x, y \in X$$

with $0 \leq L < 1$ fixed. Then

(i) T has an unique fixed point, that is, $F_T = \{x^*\}$;

(ii) The Picard iteration associated to T, i.e. the sequence $\{x_n\}_{n\in\mathbf{N}}$, defined by

(1.3.3) $$x_n = T(x_{n-1}) = T^n(x_0), \quad n = 1, 2, \ldots$$

converges to x^*, for any initial guess $x_0 \in X$.

(iii) The following a priori and a posteriori error estimates hold:

(1.3.4) $$d(x_n, x^*) \leq \frac{L^n}{1-L} d(x_0, x_1), \quad n = 1, 2, \ldots$$

(1.3.5) $$d(x_n, x^*) \leq \frac{L}{1-L} d(x_{n-1}, x_n), \quad n = 1, 2, \ldots$$

(iv) The rate of convergence is given by:

(1.3.6) $$d(x_n, x^*) \leq Ld(x_{n-1}, x^*) \leq L^n d(x_0, x^*), \quad n = 1, 2, \ldots$$

Proof. (Berinde, 2007) The proof for (i) and (ii) was given in the proof of the simplified Banach fixed point theorem (**1.1.22**).

Next we will prove (iii). Since T is a contraction, from equation **1.3.2** we have

$$d(x_2, x_1) = d(Tx_1, Tx_0) \leq Ld(x_1, x_0),$$

and by induction

$$d(x_{n+1}, x_n) \leq L^n d(x_1, x_0), \quad n = 0, 1, 2, \ldots$$

Thus, for any numbers $n, p \in \mathbf{N}$, $p > 0$, we have

$$d(x_{n+p}, x_n) \leq \sum_{k=n}^{n+p-1} d(x_{k+1}, x_k) \leq \sum_{k=n}^{n+p-1} L^k d(x_1, x_0) \leq \frac{L^n}{1-L} d(x_1, x_0).$$

By letting $p \to \infty$, we find

$$d(x_n, x^*) = d(x^*, x_n) = \lim_{p \to \infty} d(x_{n+p}, x_n) \leq \frac{L^n}{1-L} d(x_0, x_1), n \geq 0$$

which is the a priori error estimate.

To obtain the a posteriori error estimate (**1.3.5**), let us notice, that since T is a contraction we have (using **1.3.2**)

$$d(x_{n+1}, x_n) \leq L d(x_n, x_{n-1})$$

and, by induction,

$$d(x_{n+k}, x_{n+k-1}) \leq L^k d(x_n, x_{n-1}), \quad k \in \mathbf{N}^*,$$

so

$$d(x_{n+p}, x_n) \leq (L + L^2 + \cdots + L^p) d(x_n, x_{n-1}) \leq \frac{L}{1-L} d(x_n, x_{n-1}).$$

By letting $p \to \infty$ in the last inequality we get exactly the a posteriori error estimate. □

(1.3.7) Remark (Berinde, 2007)

1. The a priori error estimate shows that, when starting from an initial guess $x_0 \in X$, the approximation error of the n^{th} iterate is completely determined by the contraction coefficient L and the initial displacement $d(x_1, x_0)$;
2. Similarly, the a posteriori error estimate shows that, in order to obtain the desired error approximation of the fixed point by means of Picard iteration, that is, to have $d(x_n, x^*) \leq \epsilon$, we need to stop the iterative process at the first step n for which the displacement between two consecutive iterates is at most $(1 - L)\epsilon/L$. So, the a posteriori estimation offers a direct stopping criterion for the iterative approximation of fixed points by Picard iteration, while the a priori estimation indirectly gives a stopping criterion;
3. It is easy to see that the a posteriori estimation is better than the a priori one, in the sense that we can obtain the a priori estimation from the a posteriori estimation .

4. Each of the three estimations given in the previous theorem shows that the convergence of the Picard iteration is at least as quick as that of the geometric series $\sum L^n$. However, the convergence rate of Picard iteration for any contraction is linear;
5. In most of the cases, the contraction condition is not satisfied in the whole space X, but only locally. In this context, a local version of the contraction mapping principle is very useful for certain practical purposes.

(1.3.8) Corollary Let (X, d) be a complete metric space and

$$B(y_0, R) = \{x \in X \mid d(x, y_0) < R\}$$

be the open ball. Let $T : B(y_0, R) \to X$ be an L-contraction, such that

$$d(Ty_0, y_0) < (1 - L)R.$$

Then T has a fixed point that can be obtained using the Picard iterative scheme, starting from any $x_0 \in B(y_0, r)$.

Proof. (Berinde, 2007) We show that any closed ball $\overline{B} = \overline{B}(y_0, r), r < R$, is an invariant set with respect to T, that is $T(\overline{B}) \subset \overline{B}$. To prove this, let us consider $x \in \overline{B}$. Then $d(x, y_0) \leq R$, and from

$$d(Tx, y_0) \leq d(Tx, Ty_0) + d(Ty_0, y_0) \leq Ld(x, y_0) + (1 - L)R$$

we obtain

$$d(Tx, y_0) \leq LR + (1 - L)R = R,$$

which shows that $Tx \in \overline{B}$. Since $\overline{B}$ is complete, we can apply now Banach fixed point theorem (**1.1.22**) to get the conclusion. □

(1.3.9) Definition Let (X, d) be a complete metric space. A mapping $T : X \to X$ is called (strict) **Picard mapping** if there exists $x^* \in X$ such that $F_T = \{x^*\}$ and

$$T^n(x_0) \to x^* (\text{uniformly}) \forall x_0 \in X.$$

(1.3.10) Example If (X, d) is a complete metric space, then any contraction $T : X \to X$ is a Picard mapping.

1.3.2 A New Javascript Implementation

The new Javascript implementation of the Picard iteration computes an approximate fixed point of real valued functions $f : \mathbb{R} \to \mathbb{R}$. The implementation is in the form of a Javascript function fixpoint.picard, which accepts as a parameters the following input data for the algorithm:

- *f* - the function for which the approximate fixed point will be computed, specified as a Javascript function which must accept a single argument;
- *x*0 - the real number which specifies the initial approximation used to start the algorithm;
- *options* - a Javascript object which can contain additional optional arguments used to control the behaviour of the algorithm, as described below;
- *options.maxError* - specifies the maximum error to be accepted for the approximation of the fixed point;
- *options.convergenceTest* - this parameter is a Javascript function which will be used by the algorithm to perform the convergence test for the sequence of successive approximations;
- *options.maxSteps* - the maximum number of steps to be executed before stopping the algorithm, in the case that desired precision was not achieved for the approximation;
- *options.checkCycles* - this option can be used to disable the additional checks for oscillations.

The Picard iteration is based on the generic fixed point iteration described in 1.2.2, which contains all the necessary details about the implementation and usage. Below we present just the simple definition of the Javascript function corresponding to the Picard iterative method.

```
/**
* @public method which computes a fixed point using the Picard iteration;
    @see also #iterate.
* @param {Function} f is the function to be iterated.
* @param {Number} x0 is the initial approximation.
* @param {Object} options will contain additional optional parameters.
* @return {Object} iteration result.
*/
fixpoint.picard = function(f, x0, options)
{
  try
  {
    validatefx0(f, 1, x0);
    return iterate(f, x0, options);
  }
  catch (err)
  {
    return err;
  }
}
```

Complementary to the algorithm implementation a simple web-based user interface is available at http://algonum.appspot.com/#fixpoint.picard, which can be used very easily by persons without knowledge of Javascript programming for ex-

perimenting with the algorithm. The user interface allows the users to enter the input data for the algorithm in a very easy format. Some special attention was paid on editing the mathematical formulas, and as it is visible in figure 1.3, it is possible to edit complex formulas using a powerful editor.

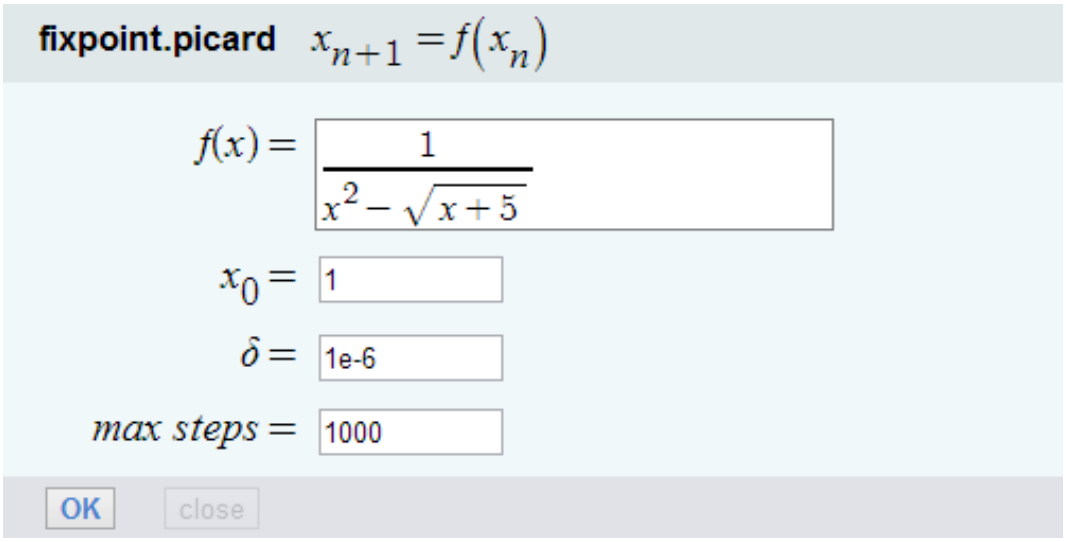


Figure 1.3 – Web-based user interface for Picard iteration

The web-based software package can be also used to obtain nice visualisations of the function and the associated Picard iteration (as a cobweb plot), as visible in the following figures.

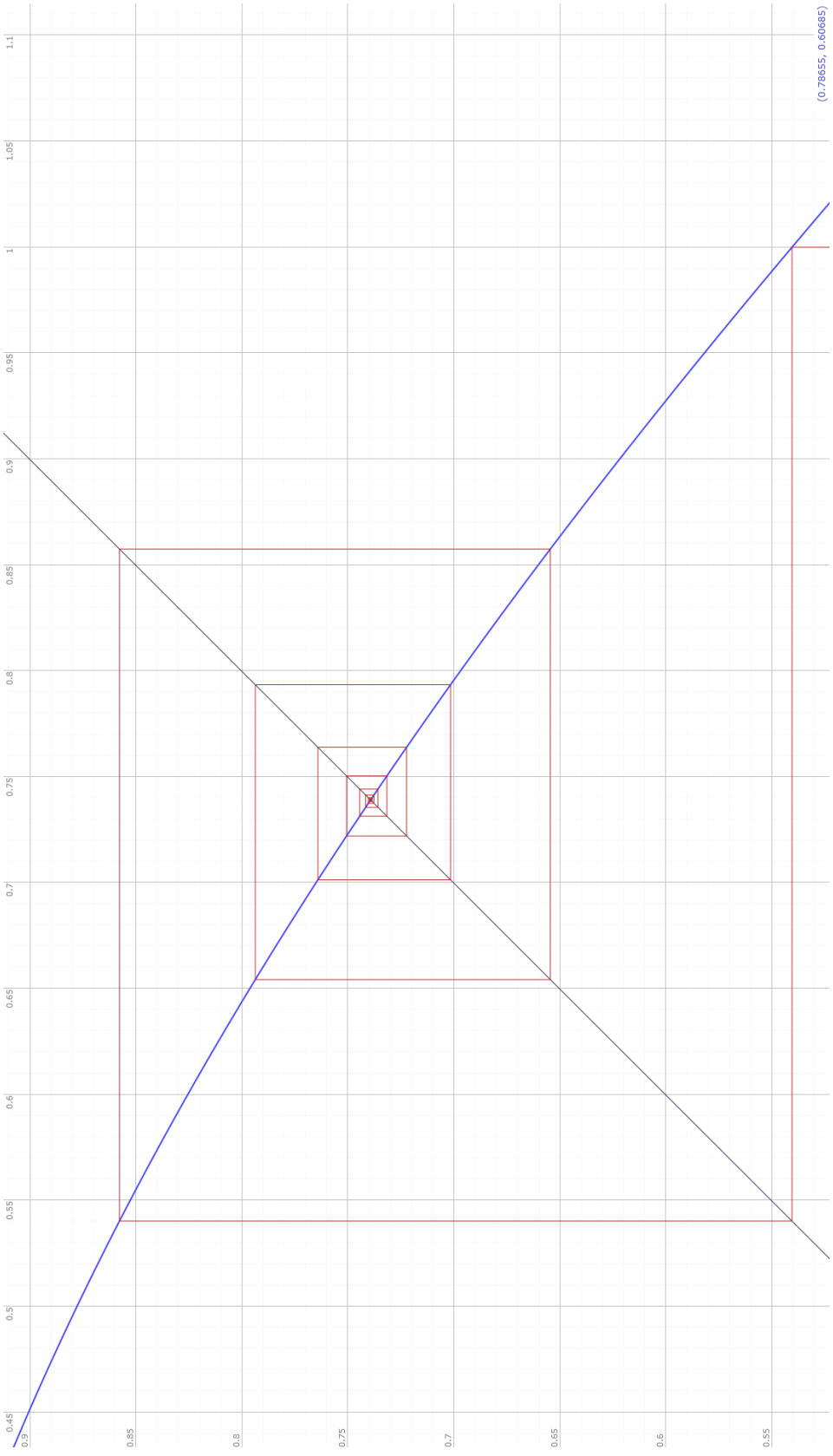


Figure 1.4 – Visualisation of the Picard iteration obtained with FIXPOINT

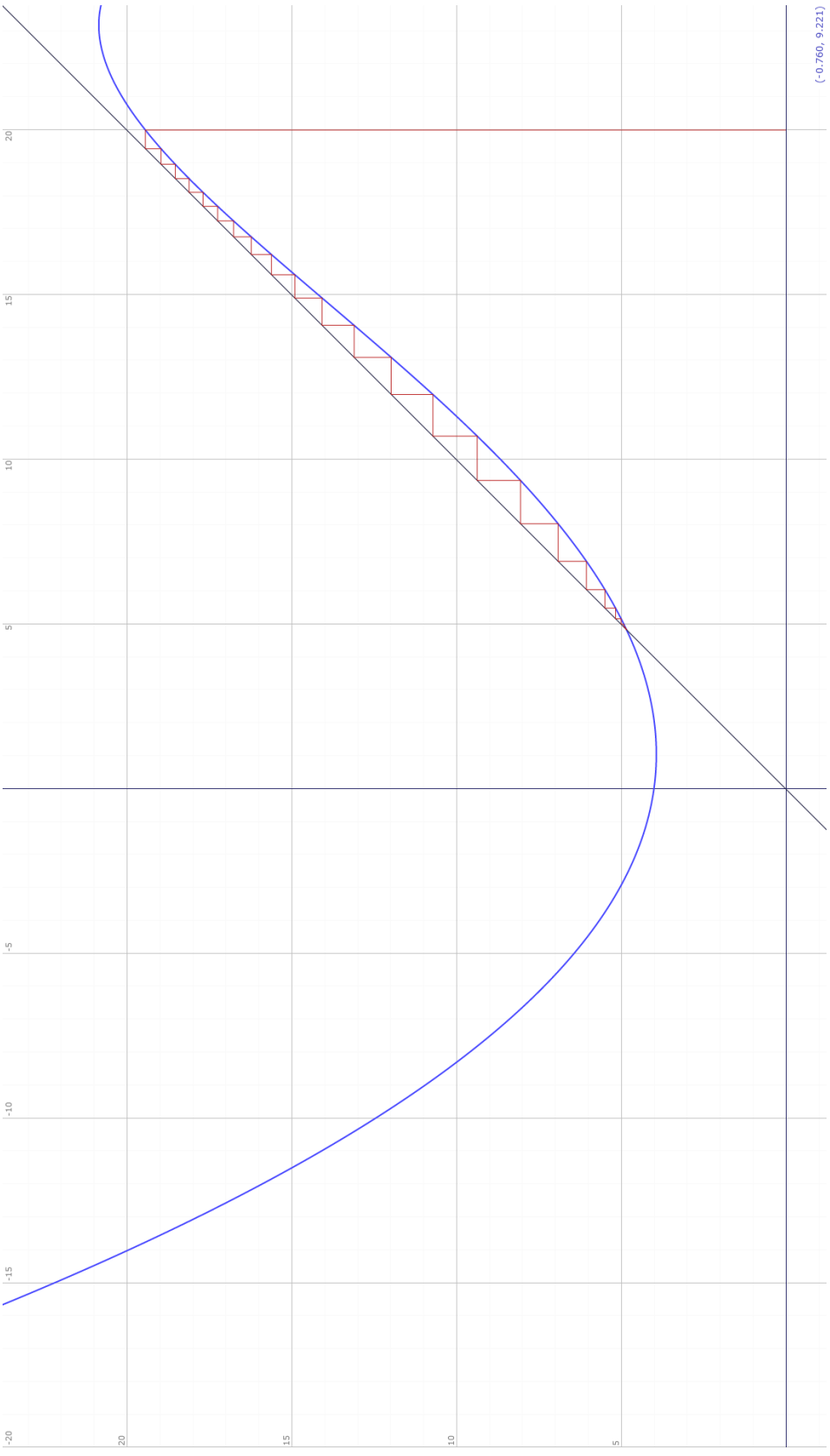


Figure 1.5 – Visualisation of the Picard iteration obtained with FIXPOINT

1.4 KRASNOSELSKII, MANN AND ISHIKAWA ITERATIONS

1.4.1 Theoretical Background

When considering the case of nonexpansive mappings, the Picard iteration is not necessarily always convergent. In this section we will describe the Krasnoselskii iteration, in the context of nonexpansive and pseudocontractive mappings, with the main theoretical notions presented as in Berinde (2007). The original form of the iteration formula and the corresponding convergence result was given by Krasnoselskii (1955), in the form of the averaged sequence involving two successive terms of the Picard iteration:

$$x_{n+1} = \frac{1}{2}(x_n + Tx_n)$$

The result obtained by Krasnoselskii was later extended by Schaefer (1957) to the case when the constant $1/2$ is replaced by a number $\lambda \in (0,1)$, obtaining in this way what it is known as the general Krasnoselskii iteration.

(1.4.1) Definition Let H be a Hilbert space and C a subset of H. A mapping $T : C \to H$ is called **demicompact** if it has the property that whenever $\{u_n\}$ is a bounded sequence in H and $\{Tu_n - u_n\}$ is strongly convergent, then there exists a subsequence $\{u_{n_k}\}$ of $\{u_n\}$ which is strongly convergent.

(1.4.2) Lemma Let C be a bounded closed convex subset of a Hilbert space H and $T : C \to C$ be a nonexpansive and demicompact operator. Then the set F_T of fixed points of T is a nonempty convex set.

(1.4.3) Theorem Let C be a bounded closed convex subset of a Hilbert space H and $T : C \to C$ be a nonexpansive and demicompact operator. Then the set F_T of fixed points of T is a nonempty convex set and for any given x_0 in C and any fixed number λ with $0 < \lambda < 1$, the Krasnoselskii iteration $\{x_n\}_{n=0}^{\infty}$ given by

$$x_{n+1} = (1-\lambda)x_n + \lambda Tx_n, \quad n = 0, 1, 2, \ldots$$

converges (strongly) to a fixed point of T.

(1.4.4) Theorem (Berinde, 2007) Let X be a Banach space and K a nonempty closed convex subset of X. If $T : X \to X$ is a Lipschitzian (with constant L) and strongly pseudo contractive operator (with constant k) such that the fixed point set of T, F_T, is nonempty, then the Krasnoselskii iteration $\{x_n\} \subset K$ generated by $x_0 \in K$, with $\lambda \in (0, a)$ and the number a given by

$$a = \frac{k}{(L+1)(L+2-k)}$$

converges strongly to the (unique) fixed point p of T. Moreover, amongst all Krasnoselskii iterations, there exists one which is the fastest one, obtained for

$$\lambda_0 = -1 + \sqrt{1+a}$$

Next we present the definitions of Man and Ishikawa iteration processes. Although, this iteration was introduced by Mann (1953), two years earlier than the Krasnoselskii iteration, the Mann iteration is formally a generalization of the later, in its normal form, is obtained by replacing the parameter λ in the Krasnoselskii iteration formula by a sequence of real numbers $\{\alpha_n\} \subset [0,]$. Originally, the Mann iteration was defined in a matrix formulation, see Berinde (2007).

(1.4.5) Definition Let C be a nonempty convex subset of a linear space X and $T : C \to C$ a mapping. Let $\{\alpha_n\}$ be a sequence of nonnegative numbers satisfying:

$$\sum_{n=0}^{\infty} \alpha_n = \infty, \;\; 0 \leq \alpha_n < 1, \;\; \forall n \in \mathbf{N}.$$

The sequence $\{x_n\} \in C$, defined by

$$x_{n+1} = M(x_n, \alpha_n, T), \; n > 0; \; x_0 \in C$$

where

$$M(x_n, \alpha_n, T) = (1 - \alpha_n) \cdot x_n + \alpha_n \cdot Tx_n$$

is called the (normal) **Mann iteration**.

If in the above definition we take the sequence $\alpha_n = \lambda \; (const)$, then the Man iterative process obviously reduces to the Krasnoselskii iteration.

(1.4.6) Theorem Let C be a nonempty closed convex subset of a Banach space X and $T : C \to C$ a continuous mapping. If the Mann iteration converges strongly to a point $x^* \in C$ then x^* is a fixed point of T.

The Ishikawa iteration was first used to establish the strong convergence to a fixed point for a Lipschitzian and pseudo-contractive selfmap of a convex compact subset of a Hilbert space. The Ishikawa iteration is given in the following definition.

(1.4.7) Theorem Let K be a convex compact subset of a Hilbert space H, $T : K \to K$ a Lipschitzian pseudocontractive map and let the Ishikawa iteration be the sequence $\{x_n\} \in K$ defined by

$$x_{n+1} = I(x_n, \alpha_n, \beta_n, T), \; n > 0; \; x_0 \in K$$

where

$$I(x_n, \alpha_n, \beta_n, T) = (1 - \alpha_n) \cdot x_n + \alpha_n \cdot T\left[(1 - \beta_n) \cdot x_n + \beta_n \cdot Tx_n\right]$$

with $\{\alpha_n\}, \{\beta_n\}$ sequences of positive numbers satisfying

$$(i)\ 0 \leq \alpha_n \leq \beta_n \leq 1,\ n \in \mathbf{N};\ \ (ii)\ \lim_{n\to\infty} \beta_n = 0;\ \ (iii)\ \sum_{n=0}^{\infty} \alpha_n \cdot \beta_n = \infty.$$

Then the Ishikawa iteration converges strongly to a fixed point of T.

(1.4.8) Remark The Ishikawa iteration can be rewritten as follows:

$$y_n = (1 - \beta_n) \cdot x_n + \beta_n \cdot Tx_n,\ n \in N$$
$$I(x_n, \alpha_n, \beta_n, T) = (1 - \alpha_n) \cdot x_n + \alpha_n \cdot Ty_n, n \in N$$

If we take $\beta_n = 0$ the Ishikawa iteration reduces to the Mann iteration, but there is no general dependence between results for Mann iteration and Ishikawa iteration. If we take $\alpha_n = 0$ the Ishikawa iteration reduces to the Picard iteration.

1.4.2 A New Javascript Implementation

The new Javascript implementations of the Krasnoselskii, Mann and Ishikawa iterations compute an approximate fixed point of real valued functions $f : \mathbb{R} \to \mathbb{R}$. The implementations are in the form of Javascript functions: fixpoint.krasnoselskii, fixpoint.mann, fixpoint.ishikawa, which accept as a parameters the following input data for the algorithm:

- f - the function for which the approximate fixed point will be computed, specified as a Javascript function which must accept a single argument;
- *lambda* - the value corresponding to the λ constant in the Krasnoselskii iteration formula;
- *alphan* - is a function which should return elements of the α_n sequence; the function should have parameter, n which is the index of the element in the sequence to be computed; this parameter is used in the Mann and Ishikawa iterations;
- *betan* - is similar to the above parameter *alphan*; this parameter is used in the Ishikawa iterations;
- $x0$ - the real number which specifies the initial approximation used to start the algorithm;
- *options* - a Javascript object which can contain additional optional arguments used to control the behaviour of the algorithm, as described below;
- *options.maxError* - specifies the maximum error to be accepted for the approximation of the fixed point;
- *options.convergenceTest* - this parameter is a Javascript function which will be used by the algorithm to perform the convergence test for the sequence of successive approximations;

- *options.maxSteps* - the maximum number of steps to be executed before stopping the algorithm, in the case that desired precision was not achieved for the approximation;
- *options.checkCycles* - this option can be used to disable the additional checks for oscillations.

The Krasnoselskii, Mann and Ishikawa iterations are based on the generic fixed point iteration described in 1.2.2, which contains all the necessary details about the implementation and usage. Below we present just the simple definition of the Javascript functions corresponding to the Krasnoselskii, Mann and Ishikawa iterative methods.

```
/**
* @public method which computes a fixed point using the Picard iteration;
    @see also #iterate.
* @param {Function} f is the function to be iterated.
* @param {Number} lambda is number between 0 and 1 (the constant in the
    Krasnoselskii formula)
* @param {Number} x0 is the initial approximation.
* @param {Object} options will contain additional optional parameters.
* @return {Object} iteration result.
*/
fixpoint.krasnoselskii = function(f, lambda, x0, options)
{
  function K(x)
  {
    return (1 - lambda) * x + lambda * f(x);
  }
  try
  {
    validatefx0(f, 1, x0);
    // validate lambda constant
    if (!Number.isFinite(lambda))
    {
      throw { errorMessage : 'A finite real number is expected.',
    errorArgument: 'lambda' };
    }
    if (lambda < 0 || lambda > 1)
    {
      throw { errorMessage : 'A real number between 0 and 1 is expected.',
     errorArgument: 'lambda' };
    }
    return iterate(f, lambda, options);
  }
  catch (err)
  {
    return err;
  }
}
```

```
34
35 /**
36 * @public method which computes a fixed point using the Mann iteration;
     @see also #iterate.
37 * @param {Function} f is the function to be iterated.
38 * @param {Function} alphan is the formula for the sequence in the Mann
     iteration.
39 * @param {Number} x0 is the initial approximation.
40 * @param {Object} options will contain additional optional parameters.
41 * @return {Object} iteration result.
42 */
43 fixpoint.mann = function(f, alphan, x0, options)
44 {
45   function M(x, n)
46   {
47     var a = alphan(n);
48     return (1 - a) * x + a * f(x);
49   }
50
51   try
52   {
53     validatefx0(f, 1, x0);
54     // validate alphan
55     if (typeof f !== 'function')
56     {
57       throw { errorMessage : 'A function is expected.', errorArgument: '
     alphan' };
58     }
59     else if (f.length !== 1)
60     {
61       throw { errorMessage : 'The function should have exactly 1 argument.
     ', errorArgument: 'alphan' };
62     }
63     return iterate(M, x0, options);
64   }
65   catch (err)
66   {
67     return err;
68   }
69 }
70
71 /**
72 * @public method which computes a fixed point using the Ishikawa iteration
     ; @see also #iterate.
73 * @param {Function} f is the function to be iterated.
74 * @param {Function} alphan is the formula for the sequence in the Ishikawa
      iteration.
75 * @param {Function} betan is the formula for the sequence in the Ishikawa
     iteration.
76 * @param {Number} x0 is the initial approximation.
```

```
77 * @param {Object} options will contain additional optional parameters.
78 * @return {Object} iteration result.
79 */
80 fixpoint.ishikawa = function(f, alphan, betan, x0, options)
81 {
82   function I(x, n)
83   {
84     var a = alphan(n);
85     var b = betan(n);
86     var y = (1 - b) * x + b * f(x);
87     var r = (1 - a) * x + a * f(y);
88     return r;
89   }
90
91   try
92   {
93     validatefx0(f, 1, x0);
94     // validate alphan
95     if (typeof f !== 'function')
96     {
97       throw { errorMessage : 'A function is expected.', errorArgument: '
     alphan' };
98     }
99     else if (f.length !== 1)
100    {
101      throw { errorMessage : 'The function should have exactly 1 argument.
     ', errorArgument: 'alphan' };
102    }
103    // validate betan
104    if (typeof f !== 'function')
105    {
106      throw { errorMessage : 'A function is expected.', errorArgument: '
     betan' };
107    }
108    else if (f.length !== 1)
109    {
110      throw { errorMessage : 'The function should have exactly 1 argument.
     ', errorArgument: 'betan' };
111    }
112    return iterate(M, x0, options);
113  }
114  catch (err)
115  {
116    return err;
117  }
118 }
```

Complementary to the algorithm implementation a simple web-based user interface is available at http://algonum.appspot.com/, which can be used very easily by

persons without knowledge of Javascript programming for experimenting with the algorithms. The user interface allows the users to enter the input data for the algorithm in a very easy format. Some special attention was paid on editing the mathematical formulas, and as it is visible in figure 1.6, it is possible to edit complex formulas using a powerful editor.

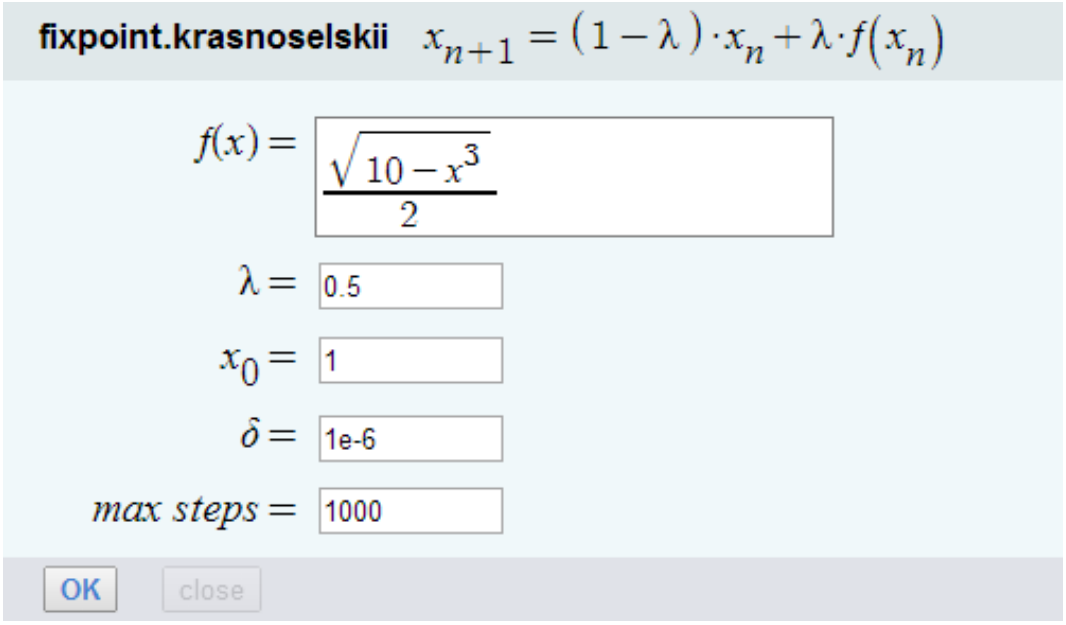


Figure 1.6 – Web-based user interface for Krasnoselskii iteration

The web-based software package can be also used to obtain visualisations for the Krasnoselskii, Mann and Ishikawa iterations. In particular, the visualisation of the Krasnoselskii iteration is realized in an innovative manner, that is, it displays the plot of the function and a plot of the associated Picard operator combined with a cobweb plot, as visible in the following figure.

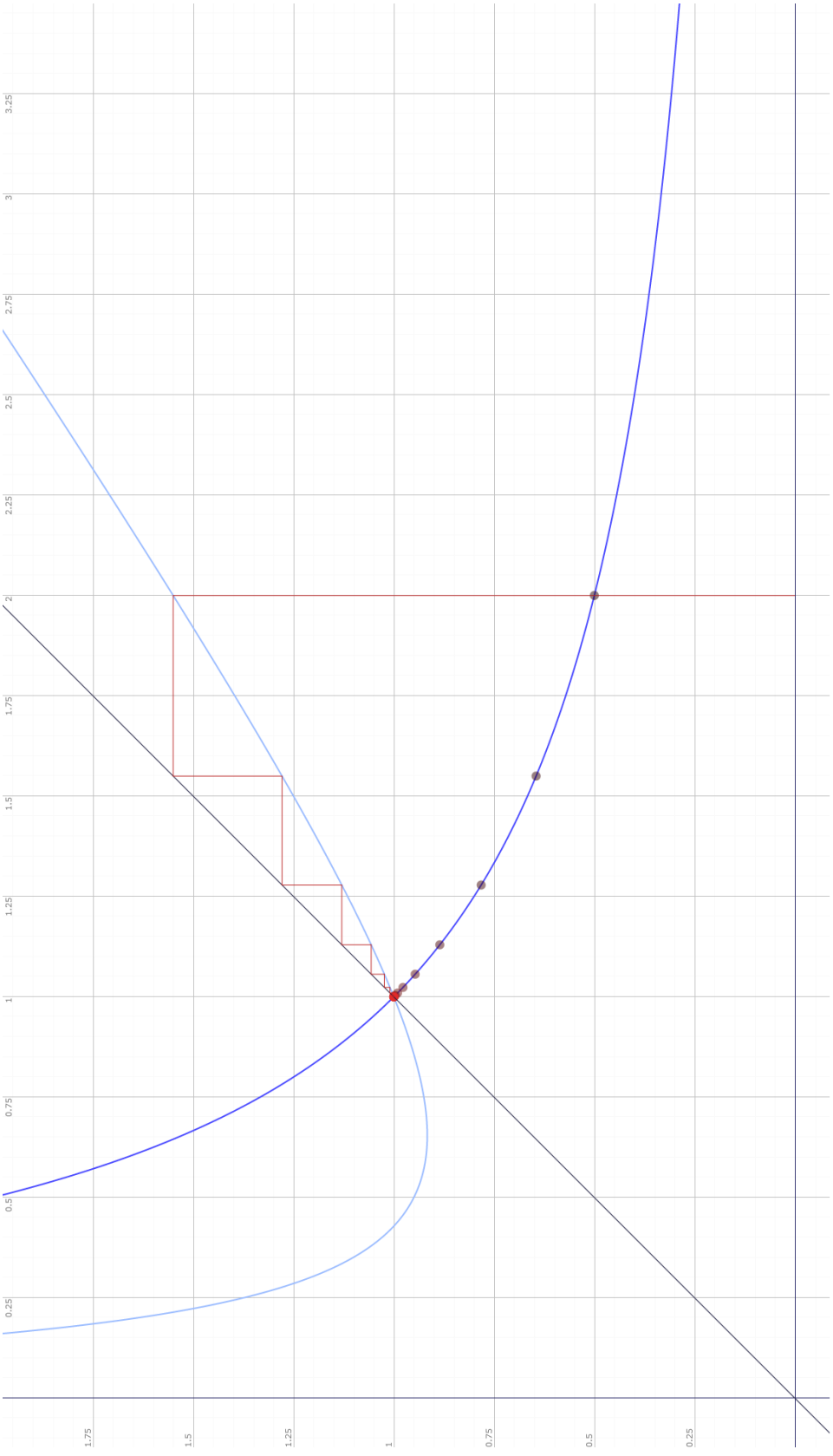


Figure 1.7 – Visualisation of the Krasnoselskii iteration obtained with FIXPOINT

2. NEW IMPLEMENTATIONS OF FIXED POINT ALGORITHMS IN $\mathbb{R}^n$

2.1 SIMPLICES AND TRIANGULATIONS

This section will describe the basic geometric objects, simplices and simplicial complexes, which will be used in the following sections for describing the implementations of several fixed point algorithms. This section also describes the pivoting steps, which are the basic operations used in the simplicial fixed point algorithms. The Freudenthal triangulation and the corresponding algorithm for pivoting in this triangulation is given as a simple example. For some more details, examples, other advanced notions and different approaches to this subject see Todd (1976a), Eaves (1984), Dang (1995), Yang (1996) and McLennan (2012).

2.1.1 Basic Notions in Euclidean Geometry

(2.1.1) Definition An **affine combination** of $v_0, \ldots, v_k \in \mathbb{R}^n$ is a point of the form $\lambda_{[0]}v_0 + \cdots + \lambda_{[k]}v_k$, where $\sum_{i=0}^{k} \lambda_{[i]} = 1$.

(2.1.2) Definition A set of points $v_0, v_1, \ldots, v_k \in \mathbb{R}^n$ is said to be **linearly independent** if the unique solution to $\sum_{i=0}^{k} \lambda_{[i]}v_i = 0$ is $\lambda_{[i]} = 0,\ \forall i = 0, 1, \ldots, k$, that is, none of the points is a linear combination of the others. Otherwise, the points are linearly dependent.

(2.1.3) Definition A set of points $v_0,\ v_1,\ \ldots\ ,\ v_k \in \mathbb{R}^n$ is said to be **affinely independent** (also called in general position), if the unique solution to

$$\sum_{i=0}^{k} \lambda_{[i]}v_i = 0, \quad \sum_{i=0}^{k} \lambda_{[i]} = 0 \tag{2.1.4}$$

is $\lambda_{[i]} = 0,\ \forall i = 0, 1, \ldots, k$, that is, none of the points is an affine combination of the others.

(2.1.5) Remark The following statements are equivalent:

- $v_0, v_1, \ldots, v_k \in \mathbb{R}^n$ are affinely independent.
- $v_1 - v_0, v_2 - v_0, \ldots, v_k - v_0$ are linearly independent.
- $(v_0, 1), (v_1, 1), \ldots, (v_k, 1) \in \mathbb{R}^{n+1}$ are linearly independent.

(2.1.6) Definition The **affine hull** $\mathrm{aff}(S)$ of a set $S \in \mathbb{R}^n$ is the set of all affine combinations of elements of S. The affine hull of S contains S as a subset, and we say that S is an **affine subspace** if the two sets are equal, that is, S is an affine subspace if it contains all affine combinations of its elements.

(2.1.7) Definition If $S \subset \mathbb{R}^n$ is an affine subspace and $v_0 \in S$, then $\{v - v_0 \mid v \in S\}$ is a linear subspace, and the **dimension** of S, $\dim S$, is the dimension of this linear subspace. The **codimension** of S is $n - \dim S$.

(2.1.8) Definition A **hyperplane** is an affine subspace of codimension one.

(2.1.9) Definition A (closed) **half-space** of a set S is a set of the form $H = \{x \in S \mid \langle p, x\rangle \leq \alpha\}$, where $p \in S, p \neq 0$ and $\alpha \in \mathbb{R}$.

(2.1.10) Definition A **convex combination** of $v_0, \ldots, v_k \in \mathbb{R}^n$ is a point of the form $\lambda_{[0]}v_0 + \cdots + \lambda_{[k]}v_k$, where $\sum_i^k \lambda_{[i]} = 1$ and $\lambda_{[0]}, \ldots, \lambda_{[k]} \geq 0$.

(2.1.11) Definition $C \in \mathbb{R}^n$ is a **convex set** if it contains all convex combinations of its elements, so that $(1-t)x_0 + tx_1 \in C$ for all $x_0, x_1 \in C$ and $0 \leq t \leq 1$.

(2.1.12) Definition The **convex hull** of a set $S \in \mathbb{R}^n$, $\overline{\mathrm{co}}S$, is the smallest convex set containing S. Equivalently, it is the set of all convex combinations of elements of S.

2.1.2 Simplices

(2.1.13) Definition For a set of $k+1$ affinely independent points $v_0, \ldots, v_k \in \mathbb{R}^n$ we define the k-dimensional **simplex** with the vertices $\{v_0, v_1, \ldots, v_k\}$ to be the convex hull

(2.1.14)
$$S = \left\{ x = \sum_{i=0}^{k} \lambda_{[i]} v_i \;\middle|\; \lambda_{[i]} \geq 0,\ \sum_{i=0}^{k} \lambda_{[i]} = 1 \right\}$$

The above coefficients, $\lambda_{[0]}, \ldots, \lambda_{[k]}$ are usually called the barycentric co-ordinates of x with respect to the affine basis $v_0, v_1, \ldots, v_k$.

(2.1.15) Remark A simplex is the generalization of a triangle or tetrahedron to the n-dimensional space. A 0-simplex is a point, a 1-simplex is a line segment, a 2-simplex is a triangle, a 3-simplex is a tetrahedron.

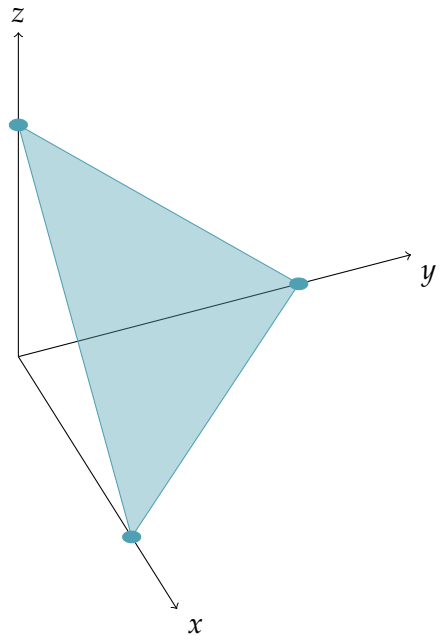


Figure 2.1 – The standard 2-simplex in $\mathbb{R}^3$

(2.1.16) Definition The **standard n-simplex** (or unit n-simplex) is the subset of $\mathbb{R}^{n+1}$ given by

(2.1.17)
$$\Delta^n = \left\{ (x_0, \ldots, x_n) \in \mathbb{R}^{n+1} \;\middle|\; \sum_{i=0}^{n} x_i = 1 \text{ and } x_i \geq 0, i = \overline{0, n} \right\}$$

The $n+1$ vertices of the standard n-simplex are the points $e_i \in \mathbb{R}^{n+1}$:

$$\begin{aligned} e_0 &= (1, 0, 0, \ldots, 0) \\ e_1 &= (0, 1, 0, \ldots, 0) \\ &\vdots \\ e_n &= (0, 0, 0, \ldots, 1) \end{aligned}$$

(2.1.18) Definition Let $S = [v_0, v_1, \ldots, v_k]$ be a k-simplex. The convex hull of any nonempty subset of the $k+1$ points that define an k-simplex is called a **face** of the simplex. The convex hull of a subset of size $j+1$ (of the $k+1$ defining points) is an j-simplex, called an j-face of the k-simplex. Of particular interest are:

- the 0-faces which are called the **vertices (vertex)** of S
- the 1-faces which are called the **edges** S
- the $(k-1)$-faces which are called the **facets** of S

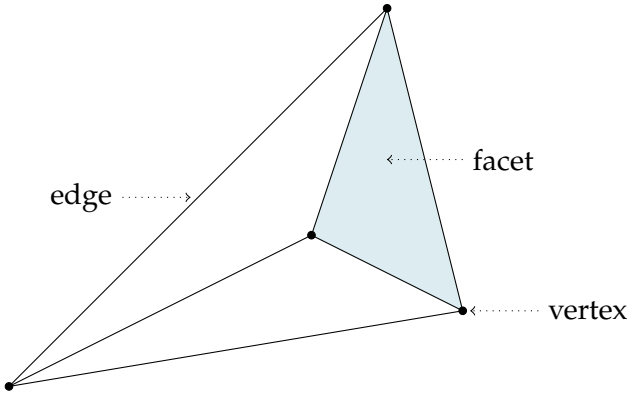


Figure 2.2 – Elements of 3-dimensional simplex

2.1.3 Triangulations

(2.1.19) Definition Let $C \subset \mathbb{R}^n$ be a k-dimensional convex set and let $\mathcal{T} = \{\tau_i\}$ be a collection of k-simplices. We call $\mathcal{T}$ a **triangulation** (also called **simplicial subdivision** or **simplicial complex**) of the convex set C if:

(a) $\bigcup_{\tau \in \mathcal{T}} \tau = C$

(b) for any two distinct simplices $\tau_1, \tau_2 \in \mathcal{T}$, the intersection $\tau_1 \cap \tau_2$ is either the empty set or a common face of both simplices.

(c) each element $x \in C$ has a neighborhood intersecting only a finite number of simplices in $\mathcal{T}$.

Also, we denote by $\mathcal{T}^i$ the collection of all i-faces of simplices of $\mathcal{T}$. It is trivial to see that $\mathcal{T}^k = \mathcal{T}$ and that the members of $\mathcal{T}^0$ are the vertices of simplices of $\mathcal{T}$, which are also called the **nodes** of the triangulation.

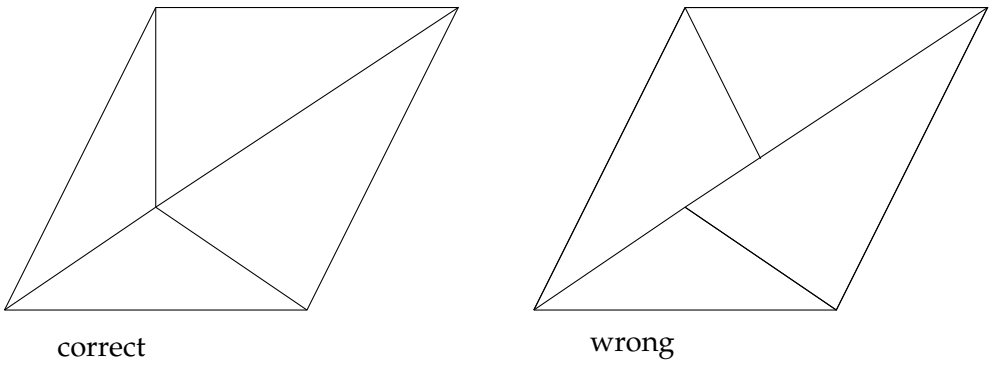


Figure 2.3 – Triangulation

(2.1.20) Definition Let $\mathcal{T}$ be a triangulation of a simplex S. The **diameter** of a simplex $\sigma \in \mathcal{T}$ is given by:

$$\operatorname{diam}(\sigma) = \max\{||x - y|| \mid x, y \in \sigma\}.$$

The **mesh size** of a triangulation $\mathcal{T}$ is the maximum diameter of the simplices in the triangulation:

$$\text{mesh}(\mathcal{T}) = \sup_{\tau \in \mathcal{T}} \text{diam}(\tau).$$

(2.1.21) Definition Two simplices σ_1, σ_2 from a triangulation $\mathcal{T}$ are called **adjacent** if they meet in a common facet of both simplices.

(2.1.22) Remark Usually, a triangulation is stored in the computer's memory using only a current simplex and a corresponding current facet of this simplex, together with information used to obtain adjacent simplices, as required in various algorithms.

(2.1.23) Lemma (Allgower and Georg, 2003) Consider $\mathcal{T}$ be a triangulation of a simplex S, $\sigma \in \mathcal{T}$ be a simplex from the triangulation and τ be a facet of σ. Then there is a unique simplex $\tilde{\sigma}$, such that: $\sigma \neq \tilde{\sigma}$ and τ is a facet of $\tilde{\sigma}$.

Proof. Let $\mathcal{H} \in \mathbb{R}^{n+1}$ denote the hyperplane which contains the facet τ of σ. Then σ must lie on one side of $\mathcal{H}$; let us call this side the "left" side of $\mathcal{H}$. Consider a straight line $s \in \mathbb{R} \to c(s) \in \mathbb{R}^{n+1}$ such that $c(0)$ is a point in the relative interior of τ and such that the tangent $\dot{c}(0)$ points into the "right" side of $\mathcal{H}$. By the properties of a triangulation, definition **2.1.19**, there must exist at least one simplex $\tilde{\sigma} \in \mathcal{T}$ which contains the interval $\{c(s) \mid 0 \leq s \leq \epsilon\}$ for some small $\epsilon > 0$. Also from the definition of the triangulation **2.1.19**, $\tilde{\sigma}$ must meet σ in the common facet τ. Of course, $\tilde{\sigma}$ lies on the "right" side of $\mathcal{H}$. Finally, we observe that two simplices in $\mathcal{T}$ which have the same facet τ and lie on the same side of $\mathcal{H}$ must have a common interior point and hence coincide. This shows the uniqueness of the simplex $\tilde{\sigma}$. □

(2.1.24) Definition (Allgower and Georg, 2003) Let $\sigma = [v_0, v_1, \ldots, v_n]$ be an n-simplex of a triangulation $\mathcal{T}$ and let $\tau = [v_0, \ldots, \hat{v}_i, \ldots, v_n]$ be the facet of σ lying opposite to the vertex v_i. By the preceding lemma (**2.1.23**), there must be a unique node $\tilde{v}_i$, which is different from v_i and such that $\tilde{\sigma} = [v_0, \ldots, \tilde{v}_i, \ldots, v_n] \in \mathcal{T}$. The passage from σ to $\tilde{\sigma}$ is called a **pivoting step**. We say that the vertex v_i of σ is **pivoted** into $\tilde{v}_i$ and that the simplex σ is pivoted into the simplex $\tilde{\sigma}$ across the facet τ.

2.1.4 Freudenthal Triangulation

(2.1.25) Example We describe the subdivision of a hypercube into $n!$ simplices along a diagonal, which is usually called **Freudenthal triangulation** or shorter, $K(h)$. This triangulation, already considered by Coxeter (1934) and Freudenthal

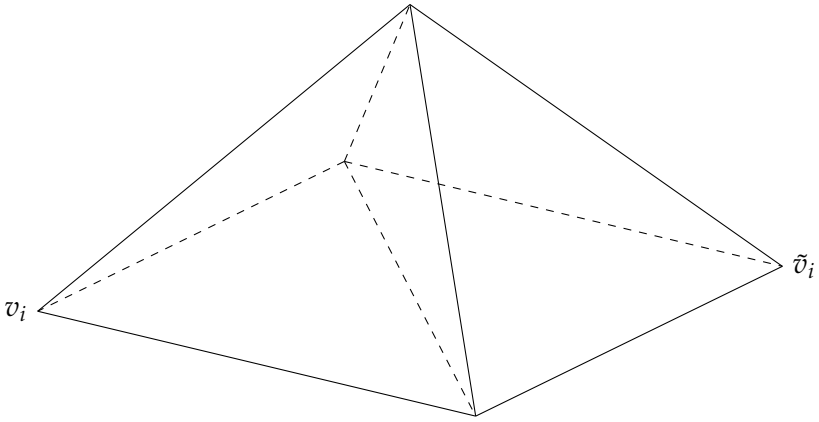


Figure 2.4 – Pivoting in a triangulation

(1942), was popularized by Kuhn in the context of fixed point calculation Kuhn (1960), and is frequently used for computational purposes, e.g. by Allgower and Georg (2003).

The nodes of this triangulation are the same as the vertices of the unit hypercube in $\mathbb{R}^n$. Let $v_0 = 0 \in \mathbb{R}^n$, $p = (p_{[1]}, \ldots, p_{[n]})$ be any permutation of $1, \ldots, n$ and denote by $I_{p_{[i]}}$ the $p_{[i]}$ column of the unit matrix of order n. Each of the $n!$ permutations of p leads to an n-dimensional simplex in this triangulation. The n-dimensional simplex associated with the permutation p, denoted by (v_0, p), is $[v_0, v_1, \ldots, v_n]$, where

$$v_i = v_{i-1} + I_{p_{[i]}}, \quad i = 1, \ldots, n.$$

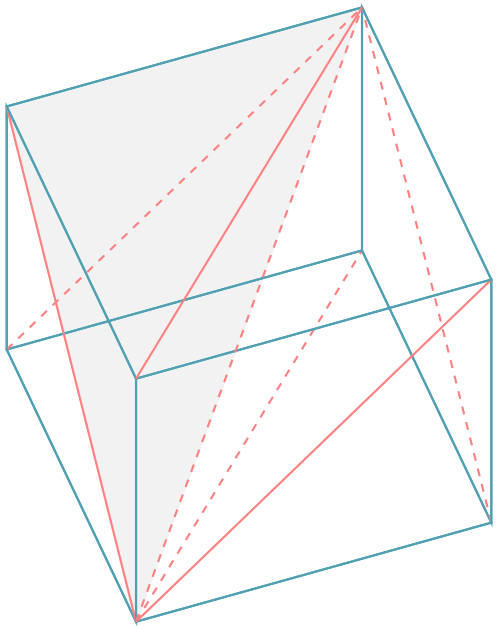


Figure 2.5 – Freudenthal triangulation of the unit cube in $\mathbb{R}^3$

An interesting property of $K(h)$ is that the subdivisions of its lower dimensional faces are also Freudenthal triangulations. Furthermore, opposite faces in this triangulation are compatibly decomposed, thus, a regularly sampled domain can be tiled using $K(h)$ subdivided hypercubes, see Kuhn (1960). This means that this triangulation can be extended, by translations, to provide a triangulation for the whole $\mathbb{R}^n$ space, which is called the K_1 triangulation. First $\mathbb{R}^n$ is partitioned in unit cubes using the integer points from $\mathbb{R}^n$ and then each unit cube is subdivided as above. The nodes of this triangulation are all the points with integer coordinates in $\mathbb{R}^n$. The mesh size of triangulation K_1 is $\sqrt{n}$.

Having a simplex σ of this triangulation and a vertex $v_i \in \sigma$, the following cyclic formulas describe how the vertex v_i is pivoted into the vertex $\tilde{v}_i$ of the simplex $\tilde{\sigma}$:

$$\textbf{(2.1.26)} \qquad \begin{array}{lllllll} \tilde{v}_i & = & v_{i-1} & - & v_i & + & v_{i+1}, \quad i = 1, \ldots, n-1 \\ \tilde{v}_0 & = & v_n & - & v_0 & + & v_1 \\ \tilde{v}_n & = & v_{n-1} & - & v_n & + & v_0 \end{array}$$

For the formal proof that this pivoting rule defines indeed a triangulation of $\mathbb{R}^n$ see Todd (1976a). This pivoting rule has been called pivoting by reflection (Allgower and Georg, 1978) and is easily to implement in a computer program, as exposed in the following algorithm.

(2.1.27) Algorithm Pivoting by reflection in Freudenthal's triangulation

Require: $[v_0, v_1, \ldots, v_n]$ is the input simplex
Require: $0 \leq i \leq n$ is the index of vertex to be pivoted next

procedure PIVOT($[v_0, v_1, \ldots, v_n], i$)
  $prev = i - 1$
  $next = i + 1$

  if $i = 0$ **then** *cyclic formulas*
    $prev = n$
  else if $i = n$ **then**
    $next = 0$
  end if

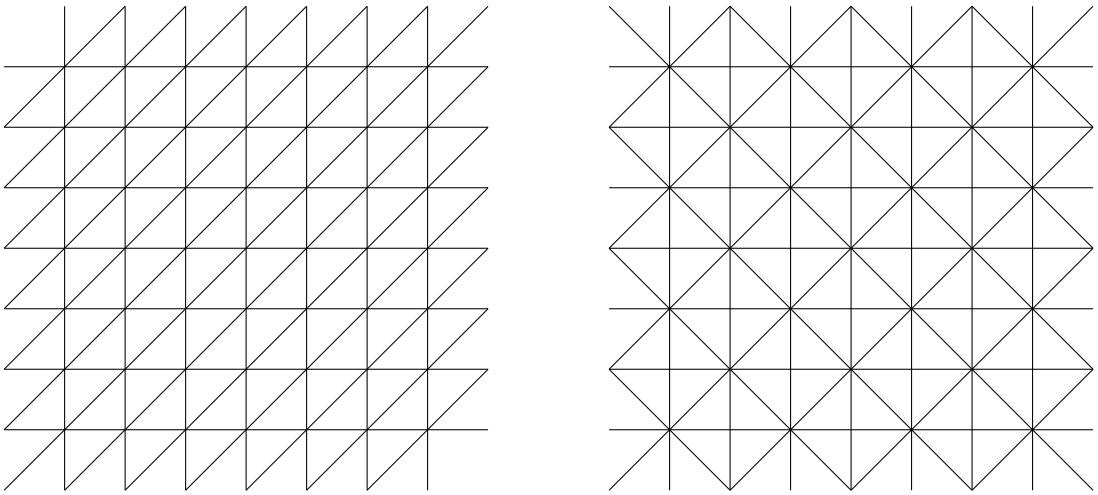
  $v_i := v_{prev} - v_i + v_{next}$ *reflection rule*
end procedure

The Freudenthal triangulation also has the advantage of being invariant under affine transformations, hence any triangulation which is obtained from Freuden-

thal's triangulation by some affine transformation obeys the above pivoting rule, as summarized in the following proposition.

(2.1.28) Proposition (Allgower and Georg, 1978) Let $\sigma = [v_0, v_1, \ldots, v_n] \in \mathbb{R}^n$ be an n-simplex, and denote by $\mathcal{T}$ the family of all simplices which are obtained from σ by a repeated use of the pivoting rule **2.1.26**. Then $\mathcal{T}$ is a triangulation of $\mathbb{R}^n$, and in fact $\mathcal{T}$ is some affine image of Freudenthal's triangulation.

A related example is the J_1 triangulation, which also has the advantage to carry less directional bias. The triangulation uses the same basic $K(h)$ subdivided hypercubes, and adjacent tiles are obtained by reflections instead of translations, as displayed in figure 2.6.



K_1 J_1

Figure 2.6 – K_1 and J_1 triangulations in R^2

Some other triangulations examples are: the J' triangulation, proposed by Todd (1984); the D_1-triangulation of $\mathbb{R}^n$, for simplicial algorithms for computing solutions of nonlinear equations, described by Dang and Talman (1990), Dang (1991); $D_1^{'}$, a new triangulation for simplicial algorithms, described by Todd and Tunçel (1993).

2.2 FIXED POINT PROBLEMS AND THEOREMS

This section presents some important results in the fixed point theory. The Brouwer fixed point theorem, one of the most important and elegant results in modern mathematics, provides a simple condition for the existence of solutions to the fixed point problem. In order to realize the importance of Brouwer fixed point theorem, see the impressive survey of Park (1999) regarding the areas of mathematics directly related to this theorem. This section also presents a generalization of Brouwer's result for set-valued mappings, the Kakutani fixed point theorem. In relation with the fixed point problem, this section will also describe some other equivalent problems. The bibliography on this subject is vast, with some excellent titles from Romanian authors, Istratescu (2002), Rus (1979), Rus et al. (2008) and with other important references like Smart (1974), Border (1989). The theoretical notions are presented mainly as in Rus (1979) and Rus et al. (2008).

2.2.1 Brouwer Fixed Point Theorem

(2.2.1) Definition A topological space X has the **fixed point property** if every continuous function $f : X \to X$ has at least one fixed point.

(2.2.2) Theorem Consider X, Y two topological spaces and $h : X \to Y$ a homeomorphism. If X has the fixed point property then Y also has the fixed point property.

Proof. (Rus, 1979) Let X to have fixed point property and $f : Y \to Y$, a continuous function. We will show that f has a fixed point.

If $h : X \to Y$ is a homeomorphism, then h is a bijection with h and h^{-1} being continuous. The mapping $h^{-1} \circ f \circ h : X \to X$ is continuous, so it has at least one fixed point, $x_0 \in X$. We have

$$h^{-1}(f(h(x_0))) = x_0,$$

so

$$f(h(x_0)) = h(x_0),$$

that is, $h(x_0)$ is a fixed point of f. □

(2.2.3) Lemma (Knaster et al., 1929) Let $S = [v_0, v_1, \ldots, v_n] \subset \mathbb{R}^n$ be a n-simplex and let $C_0, C_1, \ldots, C_n$ be a collection of closed subsets of S such that:

1. $S = \bigcup_{i=0}^{n} C_i$, that is, the simplex S is covered by the closed sets C_i.
2. $\overline{\text{co}}\{v_i \mid i \in I_k\} \subset \bigcup_{i \in I_k} C_i$, that is, the face spawned by the vertices v_i is covered by the closed sets $C_i, i \in I_k$, for all $I_k \subset \{0, 1, \ldots, n\}$.

Then $\bigcap_{i=0}^{n} C_i \neq \emptyset$.

(2.2.4) Theorem (Brouwer, 1911) Let $C \subset \mathbb{R}^n$ be a nonempty, compact and convex set, and $f : C \to C$ a continuous function. Then there exists at least one point $x^* \in C$, such that $f(x^*) = x^*$.

Proof. (Rus et al., 2008) If Brouwer theorem is true for a certain compact and convex set from $\mathbb{R}^n$, then it is true for any compact and convex set in $\mathbb{R}^n$. This due to the fact that any two compact and convex sets in $\mathbb{R}^n$ are homeomorphic and because the fixed point property is invariant under homeomorphisms.

Using the above remark we will prove that a simplex $\sigma = [v_0, \ldots, v_n] \subset \mathbb{R}^n$ has the fixed point property.

Let $f : \sigma \to \sigma$ be a continuous function and $v \in \sigma$. Then

$$v = \sum_{i=0}^{n} \lambda_{[i]} v_i, \quad \lambda_{[i]} \geq 0, \sum_{i=0}^{n} \lambda_{[i]} = 1.$$

Because $f(v) \in \sigma$, we have:

$$f(v) = \sum_{i=0}^{n} f_i(\lambda_{[0]}, \lambda_{[1]}, \ldots, \lambda_{[n]}) v_i, \quad \sum_{i=0}^{n} f_i(\lambda) = 1, \ f_i(\lambda) \geq 0.$$

We consider the sets:

$$C_k = \left\{ \sum_{i=0}^{n} \lambda_{[i]} v_i \mid \lambda_{[i]} \geq 0, \sum_{i=0}^{n} \lambda_{[i]} = 1, f_k(\lambda_{[0]}, \ldots, \lambda_{[n]}) \leq \lambda_{[k]} \right\},$$

which satisfy the conditions of KKM lemma, and let $x^* \in \bigcap_{k=0}^{n} C_k$.

If $x^* = \lambda^*_{[0]} v_0 + \cdots + \lambda^*_{[n]} v_{[n]}$, then, from the definition of the sets C_k, we have $f_i(\lambda^*_{[0]}, \ldots, \lambda^*_{[n]}) = \lambda^*_{[i]}$, so $f(x^*) = x^*$. □

2.2.2 Kakutani Fixed Point Theorem

(2.2.5) Definition Let X, Y be two sets. A mapping $f : X \to \mathcal{P}(Y)$ is named **point-to-set mapping** or **set-valued mapping**.

(2.2.6) Definition Let $f : X \to \mathcal{P}(Y)$ be a set-valued mapping. An element $x \in X$ is a **fixed point of the set-valued mapping** f, if $x \in f(x)$. The set of all fixed points of the set-valued mapping f is denoted by F_f.

(2.2.7) Definition A set-valued mapping $f : X \to \mathcal{P}(Y)$ is **upper hemicontinuous** (u.h.c.) at $x_0 \in X$, if for every open neighborhood V of $f(x_0)$, there exists an open neighborhood U of x_0 such that $f(x) \subset V, \forall x \in U$.

(2.2.8) Theorem Let X, Y be two compact metric spaces and $f : X \to \mathcal{P}(Y)$ a set-valued mapping such that $f(x)$ is closed for all $x \in X$. The following propositions are equivalent:

1. The set-valued mapping f is upper hemicontinuous.
2. The graph of f is a closed subset of $X \times Y$.
3. For any sequences $(x_n), (y_n)$, such that $x_n \to x, y_n \in f(x_n), y_n \to y$, we have $y \in f(x)$.

(2.2.9) Definition Let X, Y be two vector topological spaces. A set-valued mapping $f : X \to \mathcal{P}(Y)$ is named **Kakutani-type mapping** if f is upper hemicontinuous and $f(x)$ is nonempty, closed and convex for all $x \in X$.

(2.2.10) Definition Let X be a vector topological space and $Y \subset X$. Then Y has the **Kakutani fixed point property** if each Kakutani-type mapping $f : Y \to P(Y)$ has at leas a fixed point.

(2.2.11) Definition Let X be a topological space and Y a subspace of X. The subspace Y is called a **retract** of X if there exists a continuous map $r : X \to Y$, called **retraction**, such that the restriction of r to Y is the identity map, that is $r(y) = y, \forall y \in Y$.

(2.2.12) Theorem If the topological space X has the Kakutani fixed point property then any retract of X has the Kakutani fixed point property.

Proof. (Rus, 1979) Let Y be a retract of X and $r : X \to Y$ the corresponding retraction. Let $f : Y \to \mathcal{P}(Y)$ be a Kakutani-type mapping. The mapping $g : X \to \mathcal{P}(Y), g(x) = f(r(x))$ is a Kakutani-type mapping. If x^* be a fixed point of g then x^* is a fixed point of f. □

(2.2.13) Definition Let X be a liniar space and $Y \subset X$ a convex subset. A mapping $f : Y \to Y$ is named **affine mapping** if:

$$f(ax + (1-a)y) = af(x) + (1-a)f(y), \forall a \in (0,1), \forall x, y \in Y.$$

(2.2.14) Theorem Let X be a topological space having the Kakutani fixed point property. If Y is a topological space such that there exists an affine homeomorphism $h : X \to Y$, then Y has the Kakutani fixed point property.

Proof. (Rus, 1979) Let $f : Y \to \mathcal{P}(Y)$ be a Kakutani-type mapping. The mapping $h \circ f \circ h^{-1} : X \to \mathcal{P}(Y)$ is a Kakutani-type mapping so we have that $h^{-1}(x^*)$ is a fixed point of f. □

(2.2.15) Theorem (Kakutani, 1941) Let $C \subset \mathbb{R}^n$ be a nonempty, compact and convex set, and $f : C \to \mathcal{P}(C)$ an Kakutani-type mapping. Then there exists at least one point $x^* \in C$, such that x^* is a fixed point of the set-valued mapping f, that is $x^* \in f(x^*)$.

Proof. (Rus, 1979) From theorem **2.2.12** it follows that it is sufficient to prove the theorem for a closed simplex in $\mathbb{R}^n$.

Let $\sigma = [v_0, v_1, \ldots, v_n]$ be a closed n-simplex from $\mathbb{R}^n$. For any $\epsilon^{(m)} > 0$ we can construct a triangulation $\mathcal{T}$ of σ such that $\text{mesh}(\mathcal{T}) < \epsilon^{(m)}$. Let $f : \sigma \to \mathcal{P}(\sigma)$ be a Kakutani-type mapping. There exists a mapping $f^{(m)} : \sigma \to \sigma$ such that:

1. $f^{(m)}(y_i^{(m)}) \in f(y_i^{(m)})$ for any node $y_i^{(m)}$ of the triangulation $\mathcal{T}$.
2. $f^{(m)}$ is affine on any simplex in $\mathcal{T}$.

The mapping $f^{(m)}$ is continuous and from the Brouwer fixed point theorem it follows that it has at least a fixed point $x^{(m)^*}$:

$$x^{(m)^*} = \sum_{i=0}^{n} \lambda_{[i]}^{(m)} y_i^{(m)}, \quad \lambda_{[i]}^{(m)} \geq 0, \quad \sum_{i=0}^{n} \lambda_{[i]}^{(m)} = 1.$$

Because σ is a compact set it follows that there exists a convergent subsequence of $(x^{(m)^*})$ for which we will use the same notation $(x^{(m)^*})$. Let

$$x^* = \lim_{m \to \infty} x^{(m)^*}, \quad y_i = \lim_{m \to \infty} f^{(m)} y_i^{(m)}, \quad \lambda_{[i]} = \lim_{m \to \infty} \lambda_{[i]}^{(m)}.$$

It is evident that $\lambda_{[i]} \geq 0$ and $\sum_{i=0}^{n} \lambda_{[i]} = 1$. Because $(y_i^{(m)}, f^{(m)} y_i^{(m)}) \in G(f)$ and $G(f)$ is closed, it follows that $y_i \in f(x^*)$ and because $f(x^*)$ is convex it follows that $x^* \in f(x^*)$. □

2.2.3 Equivalent Forms of the Fixed Point Problem

In the next part of this section we will present some equivalent forms of the fixed point problem, following the presentation of Yang (1996).

(2.2.16) Definition Let C be a subset of $\mathbb{R}^n$, and let $f : C \to \mathbb{R}^n$ be a function. Then a point $x^* \in C$ is a **fixed point** of f if $f(x^*) = x^*$. The problem of finding a fixed point is called the **fixed point problem**.

As we saw before, the Brouwer fixed point theorem proves that if f is continuous it has a fixed point.

(2.2.17) Definition Let C be a subset of $\mathbb{R}^n$, and let $f : C \to \mathbb{R}^n$ be a function. Then a point $x^* \in C$ is a **stationary point** of f on C if, for all $x \in C$, we have:

$$(x^*)^\intercal f(x^*) \geq x^\intercal f(x^*)$$

The problem of finding a stationary point is called **stationary point problem**.

(2.2.18) Theorem Any continuous function $f : C \to \mathbb{R}^n$, where C is a nonempty, convex and compact subset $\mathbb{R}^n$, has at least one stationary point.

(2.2.19) Definition Let $h : \Delta^n \to \mathbb{R}^n$ be a continuous function satisfying $p^\intercal h(p) = 0$ for any $p \in \Delta^n$. Then a point $p^* \in \Delta^n$ is a **complementary point** of the function h if $h(p^*) \leq 0$. The problem of finding a complementary point is called **complementarity problem**.

(2.2.20) Theorem Let $h : \Delta^n \to \mathbb{R}^n$ be a continuous function satisfying $p^\intercal h(p) = 0$ for any $p \in \Delta^n$. Then h has a complementary point p^* in Δ^n.

(2.2.21) Definition Given a function $f : C \to \mathbb{R}^n$, a point $x^* \in C$ is called a **zero point** if $f(x^*) = 0$. The problem of finding a zero point is called **the zero point problem**.

The four problems defined in **2.2.16**, **2.2.17**, **2.2.19** and **2.2.21** are equivalent, for more details see Eaves (1971).

2.3 SPERNER LEMMA IN FIXED POINT THEORY

This section will present a simple combinatorial lemma, due to Sperner (1928). Although this lemma may look simple at a first sight, it has powerful implications, for example it is used in solving fair division problems, see Su (1999) and it can be used to prove the Fundamental Theorem of Algebra, see Huang (2004). Fifty years after the proving this result, Sperner himself listed the surprising applications of his lemma (Sperner, 1980).

In the context of fixed point theory Sperner lemma is important because it is a combinatorial analogue of Brouwer fixed point theorem. In fact, Sperner lemma is the base of the simplicial fixed point algorithms which are used to compute approximate Brouwer fixed points for continuous mappings.

2.3.1 Sperner Lemma

(2.3.1) Definition Let $\mathcal{T}$ be a triangulation of a simplex $S \in \mathbb{R}^n$ and $l : \mathcal{T}^0 \to \{0, 1, \ldots, n\}$ be a labeling function, which associates a **label** to each node of the triangulation $\mathcal{T}$. A simplex $\sigma = [v_0, \ldots, v_n] \in \mathcal{T}$ is **completely labeled** if

$$\{l(v_0), l(v_1) \ldots, l(v_n)\} = \{0, 1, \ldots, n\}$$

that is, all vertices of σ are labeled distinctly.

(2.3.2) Definition Let $\mathcal{T}$ be a triangulation of a simplex $S \in \mathbb{R}^n$. A labeling function $l : \mathcal{T}^0 \to \{0, 1, \ldots, n\}$ is called **Sperner labeling** if, for any $x \in \mathcal{T}^0$ with $x_{[i]} = 0$, we have $l(x) \neq i$, that is, the simplex S is labeled completely and the label of any node of the triangulation which is on a face of S matches the label of one of the vertices spanning that face.

(2.3.3) Remark It is evident that a Sperner labeling on the n-simplex S induces Sperner labelings on each facet as $(n-1)$-simplices.

(2.3.4) Lemma (Sperner, 1928) Any Sperner-labeled triangulation of an n-simplex contains an odd number of completely labeled n-simplices, and in particular, there is at least one.

Proof. (Cohen, 1967) The presented proof will use induction. For $n = 1$ the theorem is obvious. We assume the proposition to be true for dimensions less than n.

Let $S = [v_0, \ldots, v_n]$ be an n-simplex, $\mathcal{T}$ be a triangulation of S, $\mathcal{T}^0$ the set containing the nodes of this triangulation and $l : \mathcal{T}^0 \to \{0, 1, \ldots, n\}$ be a Sperner labeling.

We choose a face $[v_0, \ldots, v_{n-1}]$ of S. This face is triangulated and completely labeled, so it contains an odd number of complete $n-1$ simplices, with labels $\{0, 1, \ldots, n-$

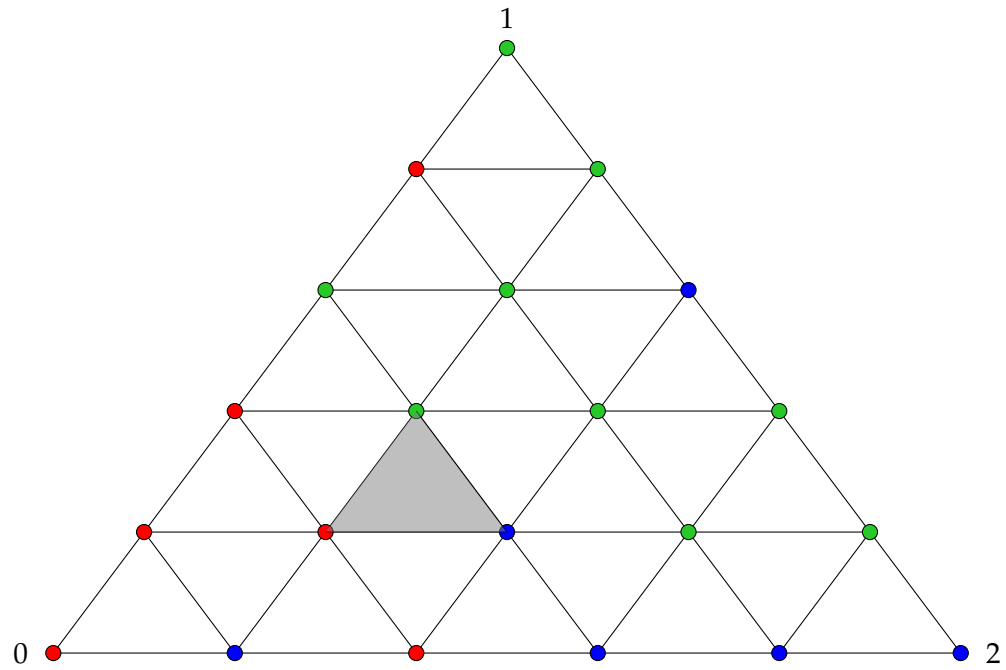


Figure 2.7 – Sperner labeling

1}. Let $[x_0, \ldots, x_{n-1}]$ be one such $n-1$ simplex, which will be the face of an n-simplex in $\mathcal{T}$ and let the other vertex of this n-simplex be x_n. If $l(x_n) = n$, then we have a completely labeled n-simplex and we stop the chain. If $l(x_n) = i, i < n$, then we have an n-simplex with two vertices with the same label. Let $l(x_0) = l(x_n)$. Then $[x_1, \ldots, x_n]$ is a complete $(n-1)$-simplex with no vertex labeled n, which, again, is the face of an n-simplex in $\mathcal{T}$ which is formed by adding the vertex $x_{n+1} \neq x_0$. Again we have either a completely labeled n-simplex or two vertices with the same label. Keeping x_{n+1} and dropping one of the previously acquired vertices, we again obtain a complete $(n-1)$-simplex.

We continue this process and establish a chain of n-simplices each with n distinct labels. The chain will not cover the same n-simplex twice; since both approaches to a given n-simplex in the chain are included, the first n-simplex covered twice would have to be the n-simplex we started with. This n-simplex, though, lies on the boundary of S and the only way it could be covered a second time by the chain is for the chain to recover the second n-simplex first. Therefore the chain ends after a finite number of n-simplices.

There are only two ways for the chain to end: one is by encountering a complete n-simplex and the other is by landing on a face of S. The only face of S that could contain a complete $(n-1)$-simplex that has no vertex labeled n is $[v_0, \ldots, v_{n-1}]$.

So we see that the completely labeled $(n-1)$-simplices of the face $[v_0, \ldots, v_{n-1}]$

pair off into pairs that are connected by type of chains described above. But an odd number of them start chains which end in completely labeled n-simplices. Similarly, a chain starting from a completely labeled n-simplex, starting with the face that has no face labeled n, will end either on the face $[v_0, \ldots, v_{n-1}]$ or at another complete n-simplex.

So the n-simplices pair off into pairs connected by chains, except for the ones connected to the face opposite v_n. We know that the number of simplices which are connected to this face is odd, therefore the total number of complete n-simplices in $\mathcal{T}$ is odd. □

2.3.2 Sperner Lemma and Brouwer Fixed Point Theorem

In the rest of this section we will describe the connections between Sperner lemma and Brouwer fixed point theorem.

(2.3.5) Theorem Sperner Lemma implies Brouwer fixed point theorem.

Proof. We consider a simplex $S = [x_0, \ldots, x_n]$ and a continuous mapping of the simplex into itself, $f : S \to S$, $f = (f_0, f_1, \ldots, f_n)$. We will show that there exists $x^* \in S$ such that $f(x^*) = x^*$.

We consider a sequence of triangulations $\mathcal{T}_\delta$ of the simplex S which we can choose in such a way that the maximum diameter of the simplices in the triangulations tends to zero, $\text{mesh}(\mathcal{T}_\delta) \to 0$.

For each triangulation $\mathcal{T}$ in the sequence $\mathcal{T}_\delta$ we also consider a corresponding labeling $l : \mathcal{T}^0 \to \{0, 1, \ldots, n\}$, with the following property: if $l(x) = i$ then $x_{[i]} > 0$ and $f_i(x) \leq x_{[i]}$. It is evident that there will be at least one such index for each node of the triangulation $\mathcal{T}$, and if there are several we make an arbitrary choice among them. It is easy to show that such a labeling is a Sperner labeling, for details see for example Sondjaja (2008).

From Sperner lemma it follows that there is at least on simplex in the triangulation $\mathcal{T}$ which has all of its vertices indexed differently. In other words, we can find a simplex in the triangulation such that at each of its vertices a different coordinate is not increased by means of the mapping f.

So each triangulation in the sequence $\mathcal{T}_\delta$ contains a completely labeled simplex, and we can find a subsequence of triangulations with the vertices converging to a single point x^*. Since the mapping f is continuous, $f(x^*) \leq x^*_{[i]}$, and therefore x^* is a fixed point of the mapping. □

(2.3.6) Theorem Brouwer fixed point theorem implies Sperner lemma.

Proof. (Sondjaja, 2008) We consider an n-simplex $S = [v_0, \ldots, v_n]$, a triangulation $\mathcal{T}$ of S and $l : \mathcal{T}^0 \to \{0, 1, \ldots, n\}$ a Sperner labeling of $\mathcal{T}$, such that $l(v_i) = i$. Using Brouwer fixed point theorem we will show that there exists a completely labeled simplex in $\mathcal{T}$.

We define $f : S \to S$, such that

$$f(x) = \begin{cases} v_{l(x)} & \text{if } x \in \mathcal{T}^0 \\ \sum_{i=0}^{k} x_{[i]} f(w_i) & \text{if } x \in [w_0, \ldots, w_k] \in \mathcal{T} \text{ and } x \notin \mathcal{T}^0 \end{cases}$$

It is a simple to see that f is continuous on the entire simplex S, so from Brouwer fixed point theorem, there is a point $x^* \in S$ such that $f(x^*) = x^*$. Without loss of generality, we assume that $x^* \in \text{int}(S)$, since when x^* is not in the interior of S, then there exists a face of S that contains x^* in its interior and we can make the following arguments for this face of S.

Let $\sigma = [w_0, \ldots, w_n] \in \mathcal{T}$ denote an n-simplex that contains x^*. We can write x^* as a convex combination of the vertices of σ:

$$x^* = \sum_{i=0}^{n} w_i x^*_{[i]}$$

First we show that x^* must lie in the interior of σ. Since f is linear within σ, then

$$f(x^*) = f(\sum_{i=0}^{n} w_i x^*_{[i]}) = \sum_{i=0}^{n} f(w_i) x^*_{[i]} \tag{2.3.7}$$

We suppose the contrary, that x^* lies in a proper face of σ. Then there is an index i for which $x^*_{[i]} = 0$. Then v_i is not a vertex of the carrier of $f(x^*)$, which means that x^* lies on a proper face of S, which contradicts the assumption that x^* is in the interior of S. So we know that x^* must be in the interior of σ, hence, $x^*_{[i]} > 0$ for all i.

Now we show that the images of $w_0, \ldots, w_n$ under f must be all the vertices of S, namely $\{v_0, \ldots, v_n\}$. We suppose the contrary, but from equation **2.3.7** we see that $f(x^*)$ lies on a proper face of S, which is again a contradiction. So the set of images of the vertices of σ is equal to the set containing all the vertices of S, which means that σ is completely labeled. □

2.4 SIMPLICIAL FIXED POINT ALGORITHMS

Motivated by the aim of finding algorithm for the computation of equilibrium prices for a general Walrasian model of competitive market exchange - see Debreu (1959), in 1967 Scarf described the first algorithm for computing a fixed point of a continuous mapping implied by Brouwer's theorem (Scarf, 1967). This section will present the main ideas used in this algorithm and the personal contributions for the implementation in the Javascript language.

The algorithm proposed by Scarf uses the idea of complementary pivoting described by Lemke and Howson (1964) and Lemke (1965). The papers of Lemke and Howson provided an algorithm for computing Nash equilibrium points for a general two-person non-zero sum game. This algorithm is also available for the more general linear complementarity problem, which as we saw in section 2.2 is related to the fixed point problem. Murty (1988) provides a clear and in-depth description of the linear complementarity problem and its applications in fixed point theory. An argument similar to the one used by Lemke and Howson, is used by Cohen (1967) to provide a proof for the Sperner lemma, which is also presented in section 2.3, but without considering an algorithm to find one of the completely labeled simplices.

For the first description of the algorithm, Scarf used the concept of primitive sets, but as he recognized in Scarf (1991), he didn't realize that the combinatorial lemma described in his original paper is the equivalent of Sperner's lemma. A few months later, Hansen proposed a great improvement for the original algorithm (Hansen, 1968), by providing an alternative to the concept of primitive sets. The improvement proposed by Hansen was also described by Kuhn (1968), where he also describes the equivalence of Scarf's algorithm with the Sperner lemma and the relation between primitive sets and triangulations.

We will now describe the original algorithm proposed by Scarf and then continue with the description of the improvements proposed by (Hansen, 1968) and Kuhn (1968), following the exposition made by Scarf (1982).

2.4.1 Scarf's Algorithm

(2.4.1) Definition A triangulation $\mathcal{T}$ of a simplex S is named **restricted triangulation** if none of the nodes $v \in \mathcal{T}^0$, other than the vertices of S, lie on the boundary of S.

Next we will describe the combinatorial lemma which is the basis of the simplicial

fixed point algorithms.

(2.4.2) Lemma (Scarf, 1982) Let $\mathcal{T}$ be a restricted triangulation of an n-simplex $S = [v_0, \ldots, v_n]$, with the nodes $v_0, \ldots, v_n, \ldots, v_k \in \mathcal{T}^0$ and let $l : \mathcal{T}^0 \to \{0, 1, \ldots, n\}$ be labeling function, such that $l(v_j) = j$, for $j = 0, 1, \ldots, n$. Then there exists at least one completely labeled simplex in the triangulation, that is, a simplex with all the vertices having distinct labels.

Proof. (Scarf, 1982) The existence of such a simplex will be proved by an explicit computational procedure based on an argument which was first described by Lemke and Howson (1964) and Lemke (1965).

We will consider the unique simplex in the triangulation whose vertices are v_1, $v_2, \ldots, v_n$ and a single other vertex, v_j. The vertices $v_1, v_2, \ldots, v_n$ will have the labels $1, 2, \ldots, n$. If the vertex v_j is labeled 0 the algorithm terminates immediately with a completely labeled simplex. Otherwise $l(v_j)$ will be one of the integers $1, 2, \ldots, n$. We proceed by eliminating the vertex whose label is the same with $l(v_j)$, arriving at a new simplex in the triangulation. Again we determine the label associated with the new vertex which has just been introduced. If this label is 0, we terminate; otherwise the process continues by eliminating that old vertex whose label agrees with that of the vertex just introduced.

At each stage of the algorithm we will be faced with a simplex of the triangulation whose vertices have the n labels $1, 2, \ldots, n$. A pair of the vertices, one of which has just been introduced, will have a common label, and we continue by removing the other member of this pair.

There are a finite number of simplices in the triangulation. We will demonstrate that the algorithm never returns to a simplex which it previously encountered. Assuming this to be correct, the algorithm must terminate after a finite number of iterations. But termination can only occur if we have reached a simplex all og whose labels are different, ot if we arrive at a simplex which has n vertices on the boundary of S, with the remaining vertex about to be removed. Such a boundary simplex, however, has to contain the vertices $v_1, \ldots, v_n$, since if the vertex v_0 appears in a simplex encountered before the label 0 would have been obtained and the algorithm would have terminated.

We see, therefore, that the we will have to demonstrate that the algorithm will never return to the same simplex. Consider the first simplex which is revisited. If this is not the initial simplex it can be revisited in through one of the two adjacent simplices with n distinct labels. But both of these adjacent simplices would have been encountered during the first visit and our simplex is therefore not the first simplex to be revisited. A similar argument demonstrates that the initial simplex is

not the first simplex to be revisited. □

Next we will give a new constructive proof of the Sperner lemma, based on the combinatorial lemma **2.4.2**.

(2.4.3) Lemma (Sperner, 1928) Let $\mathcal{T}$ be a triangulation of an n-simplex $S = [v_0, \ldots, v_n]$, and let $l : \mathcal{T}^0 \to \{0, 1, \ldots, n\}$ be labeling function, such that $l(v_{[i]}) = i$ only if $v_{[i]} > 0$, for $i = 0, 1, \ldots, n$. Then there exists at least one completely labeled simplex in the triangulation.

Proof. (Scarf, 1982) We will prove the Sperner lemma using the combinatorial lemma of Scarf (**2.4.2**) and the following simple idea. We embed the unit simplex in a larger simplex S' with the vertices $s_0, \ldots, s_n$. We then extend the triangulation $\mathcal{T}$ of S to a restricted triangulation of S', by introducing a number of simplices obtained in the following way: take an arbitrary subset $T \subset \{0, 1, \ldots, n\}$ with $t < n$ members. Consider a collection of $n + 1 - t$ vertices in the face of S defined by $x_i = 0$, for $i \in T$, and which lie in a single sub-simplex of $\mathcal{T}$. These $n + 1 - t$ vertices in S are combined with the t vertices s_i for $i \in T$ in order to define a simplex in the larger (restricted) triangulation.

In order to apply **2.4.2** to the triangulation of S' we must associate a distinct label with each of the new vertices $s_0, \ldots, s_n$, and we wish to do this in such a fashion that the completely labeled simplex obtained by applying lemma **2.4.2** will contain none of the new vertices.

We define $l(s_i) = (i + 1) \mod (n + 1)$, in other words $l(s_0) = 1$, $l(s_1) = 2$, $\ldots$, $l(s_n) = 0$. A completely labeled simplex may then contain the vertices s_i, and $n + 1 - t$ other vertices on the face $x_i = 0$, for $i \in T$. Since these remaining vertices will, by the assumption of Sperner lemma, have labels different from the members of T, it follows that the collection of vertices s_i in the completely labeled simplex must have all og the labels in T. This is contradiction to $l(s_i) = (i + 1) \mod (n + 1)$ unless T is the empty set and the completely labeled simplex is a member of the original triangulation of S. □

Considering the simplex S, the triangulation $\mathcal{T}$ and the labeling function l as in lemma **2.4.3**, we will present an alternative description of the algorithm following Saigal (1979), see also Kuhn (1968).

From Sperner lemma, we know that the triangulation $\mathcal{T}$ contains an odd number of completely labeled simplices. By considering all these completely labeled simplices from $\mathcal{T}$ together with all the facets of simplices in $\mathcal{T}$ having the set of labels $\{1, 2, \ldots, n\}$ we can create four types of paths

1. paths starting from a facet on the boundary of the simplex S, and ending in a

simplex of the triangulation (the path between A and B).
2. paths starting from a facet on the boundary of the simplex S, and ending at another facet on the boundary (the path between C and D).
3. paths starting from an simplex of the triangulation and ending at another simplex of the triangulation (the path between E and F).
4. looping paths (the path between G and G); the algorithm never reaches these paths.

The paths 1. and 2. above suggest the idea of the algorithm. The steps of the algorithm are the following:

Step 0 Find a facet τ_1 of a simplex in $\mathcal{T}$ which is on the boundary of S and has the set of labels $\{1, 2, \ldots, n\}$. Find the unique n-simplex $\sigma_1 \in \mathcal{T}$ which contains this facet. Find the vertex v of σ_1 which is not a vertex of τ_1.

Step 1 If the label of v is 0, we found a completely labeled simplex and we stop the algorithm.

Step 2 Since the label of v is one of the elements of $\{1, 2, \ldots, n\}$, there exists exactly one more facet τ_2 of σ_1 which has the set of labels $\{1, 2, \ldots, n\}$. Find the unique facet τ_2 and let $\tau_1 = \tau_2$.

Step 3 If τ_1 is on the boundary, stop. Otherwise, there exists another simplex $\sigma_2 \in \mathcal{T}$ which has τ_1 as a facet. Find the vertex $\tilde{v}$ of σ_2 which is not a vertex of τ_1 and let $v_{=}\tilde{v}$. Go to step 1.

The sequence of steps described above is finite since no simplex in the triangulation is considered more than once, and there are only a finite number of simplices in the triangulation. This is true since for a simplex to appear more than once in the algorithm, it must have more than to facets with the set of labels $\{1, 2, \ldots, n\}$, which is not possible (as seen in step 2). Thus, after a finite number of iterations, the algorithm will stop either at step 1 or at step 3. Stopping at step 1, leads us to a completely labeled simplex, while stopping at step 3 means that the algorithm failed to find a completely labeled simplex.

Many of the authors, see for example Murty (1988) or Border (1989), also give an informal description of the algorithm, using a little story similar to the following one. Consider a house with a finite number of rooms. There are three types of rooms in the house: rooms without doors, rooms with exactly two doors and rooms with just a single door. If we enter one of the outside doors of the house and keep going from room to room, entering through one of the doors and exiting through the other one, we either end up in a room with only one door or we end up back outside of the house. The house corresponds to the simplex S and the n-sub-simplices of the triangulation $\mathcal{T}$ are the rooms in this house. Each facet of an n-sub-simplex is corresponding to a wall of the room, with the completely labeled

facets corresponding to the walls with doors. The rooms with one door are the completely labeled simplices which we are searching.

By using the algorithm described above, combined with a well chosen labeling function, we may find approximate fixed points implied by the Brouwer theorem. Let us consider a continuous mapping of the unit simplex into itself, $f : \Delta^n \to \Delta^n$, $f = (f_0, f_1, \ldots, f_n)$, such that:

$$f_i(x) \geq 0,\ i = 0, 1, \ldots n$$

and

$$\sum_{i=0}^{n} f_i(x) = 1$$

We also consider a triangulation $\mathcal{T}$ of the unit simplex, and the following labeling function: $l : \mathcal{T}^0 \to 0, 1, \ldots, n$, such that:

$$l(x) = i \quad \text{if } f_i(x) \leq x_{[i]} \text{ and } x_{[i]} \neq 0$$

It is easy verified that the assumptions of the Sperner lemma are satisfied, and the previous algorithm on which the proof of Sperner lemma is based may be used to determine such a completely labeled simplex, for example $\sigma = [v_0, \ldots, v_n]$. Also, from the continuity of f we have that for any $\epsilon > 0$ there exists a $\delta > 0$ such that $||x - y|| < \delta \implies ||f(x) - f(y)|| < \epsilon$, for any $x, y \in \sigma$. So, if $\text{diam}(\sigma) \leq \delta$, then any point $x \in \sigma$ verifies $||f(x) - x|| \leq n(\epsilon + \delta)$, that is, any point in the completely labeled simplex will serve as an approximate fixed point of the mapping, with the above precision.

2.4.2 Simplicial Fixed Point Algorithms

The algorithm described above finds an approximate fixed point of a continuous mapping by moving systematically through a sequence of simplices in a triangulation. At each step, having a particular simplex of the triangulation, we remove one of its vertices with a duplicate label. Then, the algorithm moves to the unique adjacent simplex in the triangulation which has n vertices in common with the remaining vertices of the original simplex. In order to obtain an effective computer implementation of this algorithm, this pivoting step has to be as simple as possible. Also, the nodes of the triangulation must be positioned in a regular manner in the simplex, such that the approximation error will be independent of the region in which the fixed point is located. Having the nodes of the triangulation positioned regularly will also reduce the memory required by the algorithm, since in order to find the next simplex in the sequence, it is enough to store just the vertices of the current simplex, instead of storing all the nodes of the triangulation. The improvements proposed by Hansen (1968) and Kuhn (1968) solved these two issues,

present in the original algorithm described by Scarf using the concept of primitive sets, by using a particular triangulation which we will describe below, following the description given by Scarf (1982).

Let us consider and a triangulation of the unit simplex

$$\Delta^n = \left\{ (x_0, \ldots, x_n) \in \mathbb{R}^{n+1} \;\middle|\; \sum_{i=0}^{n} x_i = 1 \text{ and } x_i \geq 0, i = \overline{0,n} \right\}$$

with the nodes given by all the points in Δ^n of the form

$$\left(\frac{s_{[0]}}{D}, \frac{s_{[1]}}{D}, \ldots, \frac{s_{[n]}}{D} \right)$$

where D is a large positive integer and

$$\sum_{i=0}^{n} s_{[i]} = D.$$

Then, the vertices of any simplex of this triangulation are given using a node of the triangulation b, called the base point, and a permutation $(\pi_{[1]}, \ldots, \pi_{[n]})$ of the set $1, 2, \ldots, n$, using the following formulas:

$$\begin{aligned} v_0 &= b \\ v_1 &= v_0 + e_{\pi_{[1]}} \\ &\vdots \\ v_n &= v_{n-1} + e_{\pi_{[n]}} \end{aligned}$$

where

$$\begin{aligned} e_1 &= (1, -1, 0, \ldots, 0) \\ e_2 &= (0, 1, -1, \ldots, 0) \\ &\vdots \\ e_n &= (0, 0, \ldots, 1, -1) \end{aligned}$$

In order to describe the pivoting operation, let us consider a simplex of the triangulation, given by the vertices:

$$\begin{aligned} v_0 &= b \\ v_1 &= v_0 + e_{\pi_{[1]}} \\ &\vdots \\ v_j &= v_{j-1} + e_{\pi_{[j]}} \\ v_{j+1} &= v_j + e_{\pi_{[j+1]}} \\ &\vdots \\ v_n &= v_{n-1} + e_{\pi_{[n]}} \end{aligned}$$

and assume that we want to pivot the vertex v_j, with $0 < j < n$.

The new simplex will contain the vertices $v_0, v_1, \ldots, v_{j-1}$, so it will be defined by the same base point b and a new permutation $(\tilde{\pi}_{[1]}, \tilde{\pi}_{[2]}, \ldots, \tilde{\pi}_{[n]})$, which has the first $j-1$ members the same as π, that is, $\tilde{\pi}_{[1]} = \pi_{[1]}, \ldots, \tilde{\pi}_{[j-1]} = \pi_{[j-1]}$.

Because v_j will not be in the new simplex we must have $\tilde{\pi}_{[j]} \neq \pi_{[j]}$. But, since v_{j+1} must remain in the new simplex, and because $v_{j+1} = v_{j-1} + e_{\pi_{[j]}} + e_{\pi_{[j-1]}}$, we have that $(\tilde{\pi}_{[j]}, \tilde{\pi}_{[j+1]}) = (\pi_{[j+1]}, \pi_{[j]})$.

In other words, the adjacent simplex with the same vertices, except v_j, is found by taking the same base point b and the permutation in which $\pi_{[j]}$ and $\pi_{[j+1]}$ are switched. So, the pivoted vertex $\tilde{v}_j$ is given by:

$$\tilde{v}_j = v_{j-1} + e_{\pi_{[j+1]}}$$

or

$$\tilde{v}_j = v_{j-1} + v_{j+1} - v_j.$$

If the vertex v_0 is to be replaced, then the new base point will be v_1 and the new permutation is given by:

$$(\tilde{\pi}_{[1]}, \tilde{\pi}_{[2]}, \ldots, \tilde{\pi}_{[n]}) = (\pi_{[2]}, \ldots, \pi_{[n]}, \pi_{[1]}).$$

In this case, the vertices of the new simplex will be given by:

$$\begin{aligned}
&\tilde{v}_0 = v_1 \\
&\tilde{v}_1 = \tilde{v}_0 + e_{\tilde{\pi}_{[1]}} = v_1 + e_{\tilde{\pi}_{[2]}} = v_2 \\
&\vdots \\
&\tilde{v}_{n-1} = \tilde{v}_{n-2} + e_{\tilde{\pi}_{[n-1]}} = v_{n-1} + e_{\tilde{\pi}_{[n]}} = v_n \\
&\tilde{v}_n = \tilde{v}_{n-1} + e_{\tilde{\pi}_{[n]}} = v_n + e_{\pi_{[1]}}
\end{aligned}$$

So, the pivoted vertex $\tilde{v}_0$ is given by:

$$\tilde{v}_0 = v_n + e_{\pi_{[1]}} = v_n + v_1 - v_0.$$

Using a similar reasoning we find that the pivoted vertex $\tilde{v}_n$ is given by:

$$\tilde{v}_n = v_0 + v_{n-1} - v_n.$$

Now it is clear that this pivoting rules are the same given in **2.1.26**, so from proposition **2.1.28** it follows that the above pivoting rules define indeed a triangulation, which is in fact an affine image of the Freudenthal triangulation, with the mesh size $\sqrt{n}/D$.

In the implementation of the algorithm, only the numerators of the fractions which define the vertices of the current simplex are stored in a matrix, each vertex having

the components represented as a row of the matrix. For example, for $n = 4$ and $D = 30$ a sequence of pivoting steps will look like below:

$$\begin{bmatrix} 0 & 5 & 9 & 16 \\ 1 & 4 & 9 & 16 \\ 1 & 4 & 10 & 15 \\ 0 & 4 & 10 & 16 \end{bmatrix}$$

$$\downarrow$$

$$\begin{bmatrix} 0 & 5 & 9 & 16 \\ 1 & 4 & 9 & 16 \\ 1 & 4 & 10 & 15 \\ 1 & 5 & 9 & 15 \end{bmatrix}$$

$$\downarrow$$

$$\begin{bmatrix} 2 & 4 & 9 & 15 \\ 1 & 4 & 9 & 16 \\ 1 & 4 & 10 & 15 \\ 1 & 5 & 9 & 15 \end{bmatrix}$$

$$\downarrow$$

$$\begin{bmatrix} 2 & 4 & 9 & 15 \\ 2 & 4 & 10 & 14 \\ 1 & 4 & 10 & 15 \\ 1 & 5 & 9 & 15 \end{bmatrix}$$

$$\downarrow$$

$$\begin{bmatrix} 2 & 4 & 9 & 15 \\ 2 & 4 & 10 & 14 \\ 2 & 5 & 9 & 14 \\ 1 & 5 & 9 & 15 \end{bmatrix}$$

In order to simplify the pivoting steps the Freudenthal triangulation was chosen to be used in the algorithm, but this is not a restricted triangulation, so we can not directly apply lemma **2.4.2**. Next we will describe the method proposed by Kuhn (1968) for initiating the algorithm at any boundary point of the simplex.

The Freudenthal triangulation is extended with a set of nodes which have the first coordinate equal to -1, with the general form of these new nodes being given by: $(-1, s_{[1]}, \ldots, s_{[n]})$ with $s_{[i]} \geq 0$ for $i = 1, 2, \ldots, n$ and $\sum_{i=1}^{n} s_{[i]} = D + 1$.

In the Javascript implementation we chose D such that $D = nd$, where d is provided as user input and the starting simplex will be initialized with the vertices given by the rows of the following matrix:

$$\begin{bmatrix} -1 & d & \ldots & d & d+1 \\ -1 & d & \ldots & d+1 & d \\ \vdots & & & & \\ 0 & d & \ldots & d & d \end{bmatrix}$$

The new nodes added in the triangulation will be labeled with the first integer index i for which $s_{[i]} > d$.

2.4.3 A New Javascript Implementation

We complete this section by presenting the source code for the Javascript implementation of the simplicial fixed point algorithm described.

```
1  var fixpoint = fixpoint || {};
2
3  fixpoint.kuhn = function(f, D, maxSteps, fnLogPath)
4  {
5    var n = f.length;
6    var i;
7    var res = {};
8
9    if (isNaN(D) || D <= 0 || Math.floor(D) !== D)
10   {
11     throw new Exception('invalid value for parameter D; (a positive
       integer is expected)');
12   }
13
14   if (maxSteps === undefined)
15   {
16     maxSteps = n * D * Math.ceil(Math.sqrt(n * D));
17   }
18   else if (isNaN(maxSteps) || maxSteps <= 0 || Math.floor(maxSteps) !==
       maxSteps)
19   {
20     throw new Exception('invalid value for parameter maxSteps; (a positive
        integer is expected)');
21   }
22
```

```
23  // helper variables
24  var t = new TrSimplex(f, n, D);
25  var nSteps = 0;
26
27  // algorithm code
28  while (t.label(f) !== 0)
29  {
30    t.pivot();
31    nSteps++;
32    if (nSteps >= maxSteps) break;
33  }
34
35  // fill result
36  var res = {};
37  res.xn = t.getPoint();
38  res.numSteps = nSteps;
39
40  return res;
41
42
43  function TrSimplex(f, n, d)
44  {
45    var D = (n - 1) * d;
46    var vertices = new Array(n);
47    var labels = new Array(n);
48    var vtmp = new Array(n);
49    var cvrtx = n - 1;
50    var obj = {};
51
52    function initVertices()
53    {
54      var ivrtx, icrd, v;
55      for (ivrtx = 0; ivrtx < n; ivrtx++)
56      {
57        v = vertices[ivrtx] = new Array(n);
58        for (icrd = 1; icrd < n; icrd++)
59        {
60          v[icrd] = d;
61        }
62
63        v[0] = -1;
64        v[n - 1 - ivrtx] = (d + 1);
65
66        labels[ivrtx] = n - ivrtx - 1;
67      }
68      vertices[n - 1][0] = 0;
69
70      if (fnLogPath)
71      {
72        fnLogPath(vertices, 0);
```

```
73        }
74
75        obj.label(f);
76      }
77
78      obj.label = function(f)
79      {
80        var i;
81        var v = vertices[cvrtx];
82
83        if (v[0] === -1)
84        {
85          for (i = 1; i < n; i++)
86          {
87            if (v[i] > d)
88            {
89              labels[cvrtx] = i;
90              return i;
91            }
92          }
93
94          throw new Error("can't label vertex: " + v.toString());
95        }
96
97
98        for (i = 0; i < n; i++)
99        {
100         vtmp[i] = v[i] / D;
101       }
102
103       for (i = 0; i < n; i++)
104       {
105         if (vtmp[i] !== 0)
106         {
107           if (f[i].apply(null, vtmp) <= vtmp[i])
108           {
109             labels[cvrtx] = i;
110             return i;
111           }
112         }
113       }
114
115       return n - 1;
116       //throw new Error("can't label vertex: " + v.toString());
117     }
118
119     obj.pivot = function()
120     {
121       var i;
122       var pvrtx = -1;
```

```
123
124       for (i = 0; i < n; i++)
125       {
126         if (i !== cvrtx)
127         {
128           if (labels[i] === labels[cvrtx])
129           {
130             pvrtx = i;
131             break;
132           }
133         }
134       }
135
136       if (pvrtx === -1)
137       {
138         throw new Error("can't find pivot vertex");
139       }
140
141       if (fnLogPath)
142       {
143         //fnLogPath(vertices, pvrtx)
144       }
145
146       var i1 = pvrtx - 1;
147       var i2 = pvrtx + 1;
148
149       if (i1 === -1) i1 = n - 1;
150       if (i2 === n) i2 = 0;
151
152       var vp = vertices[pvrtx];
153       var v1 = vertices[i1];
154       var v2 = vertices[i2];
155
156       for (i = 0; i < n; i++)
157       {
158         vp[i] = v1[i] + v2[i] - vp[i];
159       }
160
161       cvrtx = pvrtx;
162     }
163
164     obj.getPoint = function()
165     {
166       var p = new Array(n);
167       var i, ivrtx;
168
169       for (i = 0; i < n; i++)
170       {
171         p[i] = 0;
172
```

```
173           for (ivrtx = 0; ivrtx < n; ivrtx++)
174           {
175             p[i] += vertices[ivrtx][i];
176           }
177
178           p[i] = p[i] / n / D;
179         }
180
181         return p;
182       }
183
184       initVertices();
185       return obj;
186     }
187 }
```

Listing 2.1 – Implementation of the simplicial fixed point algorithm

2.5 PIECEWISE-LINEAR HOMOTOPY ALGORITHMS

In 1967 Herbert Scarf proposed a method for approximating fixed points of continuous mappings Scarf (1967) which is also a numerically implementable constructive proof of the Brouwer fixed point theorem. Several improvements to the algorithm developed by Scarf were made by Terje Hansen in 1967, see Scarf and Hansen (1973) and by Harold W. Kuhn in 1968 Kuhn (1968). But the decisive advancements came in 1972, when Eaves Eaves (1972) and then Eaves and Saigal Eaves and Saigal (1972) described a piecewise-linear (PL) homotopy deformation algorithm as an improvement for the algorithm proposed by Scarf. Some other algorithms similar to piecewise linear algorithm of Eaves-Saigal, were proposed by Merrill (1972), van der Laan and Talman (1981), Wright (1981), Freund and Todd (1981), Doup et al. (1987).

The main practical advantage of the PL homotopy methods is that they don't require smoothness of the underlying map, and in fact they can be used to calculate fixed points of set-valued maps. Although PL methods can be viewed in the more general context of complementary pivoting algorithms usually are considered in the special class of homotopy or continuation methods (Allgower and Georg, 2003).

Some parts and ideas of this section are included also in Bozantan (2010) and Bozantan and Berinde (2013). Most of the definitions an theorems are presented as in Allgower and Georg (2003).

2.5.1 Homotopy Methods

The homotopy methods are useful alternatives and aides for the Newton methods in solving systems of n nonlinear equations in n variables:

$$F(x) = 0, \quad F : \mathbb{R}^n \to \mathbb{R}^n \tag{2.5.1}$$

mainly when very little a priori knowledge regarding the zero points of F is available and so, a poor starting value could cause a divergent Newton iteration sequence.

The idea of the homotopy methods is to consider a new function $G : \mathbb{R}^n \to \mathbb{R}^n$, related to F, with a known solution, and then to gradually deform this new function into the original function F. Typically one can define the convex homotopy:

$$H(x, \lambda) = \lambda G(x) + (1 - \lambda)F(x) \tag{2.5.2}$$

and can try to trace the implicitly defined curve

$$H^{-1}(0) = \{x \in \mathbb{R}^n \mid \exists\, \lambda \in [0, 1] \text{ such that } H(x, \lambda) = 0\} \tag{2.5.3}$$

from a starting point $(x_0, 1)$ to a solution point $(x^*, 0)$. The implicit function theorem ensures that the set $H^{-1}(0)$ is at least locally a curve under the assumption that $(x_0, 1)$ is a regular value of H i.e. the Jacobian $H'(x_0, 1)$ has full rank n.

However, because we don't require any smoothness conditions on F, a more complex approach involving piecewise-linear approximations is needed.

2.5.2 Piecewise Linear Approximations

(2.5.4) Definition The **piecewise linear approximation** of H (with respect to $\mathcal{T}$) is defined as the union

(2.5.5)
$$H_{\mathcal{T}} = \bigcup_{\sigma \in \mathcal{T}} H_\sigma$$

where $H_\sigma : \sigma \to \mathbb{R}^n$ is the uniquely defined affine map which coincides with H on the vertices of σ.

H_σ can be uniquely extended to an affine map on the affine space spanned by σ. The Jacobian H'_σ has the property $H'_\sigma(x - y) = H_\sigma(x) - H_\sigma(y)$ for x, y in this affine space.

(2.5.6) Definition A point $x \in \mathbb{R}^{n+1}$ is called a **regular point** of the PL map $H_{\mathcal{T}}$ if x is not contained in any face of dimension smaller than n, and if H'_τ has maximal rank n for all faces τ containing x. A value $y \in \mathbb{R}^n$ is a **regular value** of $H_{\mathcal{T}}$ if all points in $H^{-1}(y)$ are regular. If a point is not regular it is called **singular**. If a value is not regular it is called singular.

We introduce the following notation:

(2.5.7)
$$\overrightarrow{\epsilon} := \begin{pmatrix} \epsilon^1 \\ \vdots \\ \epsilon^N \end{pmatrix}$$

(2.5.8) Definition We call an n-simplex τ **completely labeled** if it contains solutions of the equation $H_\tau(v) = \vec{\epsilon}$ for all sufficiently small $\epsilon > 0$.

In other words, we define an n-simplex τ to be completely labeled if it contains a zero point of the PL approximation H_τ, and if this property of τ is stable under certain small perturbations in the above sense.

(2.5.9) Definition An $(n+1)$-simplex $\sigma \in \mathcal{T}$ is called **transverse** (with respect to H) if it contains a completely labeled n-face.

(2.5.10) Definition If $\tau = [v_1, \ldots, v_{n+1}] \in \mathcal{T}$ is an n-simplex, we call the matrix:

(2.5.11)
$$\mathcal{L}(\tau) := \begin{pmatrix} 1 & \ldots & 1 \\ H(v_1) & \ldots & H(v_{n+1}) \end{pmatrix}$$

the **labeling matrix** on τ induced by H.

(2.5.12) Proposition The n-simplex $\tau = [v_1, \ldots, v_{n+1}]$ is completely labeled if and only if: the labeling matrix $\mathcal{L}(\tau)$ is nonsingular and $\mathcal{L}(\tau)^{-1}$ is **lexicographically positive** i.e. the first non-zero entry in any row of $\mathcal{L}(\tau)^{-1}$ is positive.

2.5.3 Generic PL Continuation Algorithms

(2.5.13) Proposition Door-In-Door-Out Principle (Allgower and Georg, 2003) An $(n+1)$-simplex has either zero or two completely labeled n-faces.

Proof. We assume that σ is transverse and consider the equation $H_\sigma(v) = \vec{\epsilon}$ for $v \in \sigma$. An analogue version of Sard's theorem - the perturbation lemma - can be demonstrated for piecewise linear maps, see Allgower and Georg (2003). From this lemma we have that for any compact subset $C \subset \mathbb{R}^{n+1}$ there are at most finitely many $\epsilon > 0$ such that $C \cap H_{\mathcal{T}}^{-1}(\vec{\epsilon})$ contains a singular point of $H_{\mathcal{T}}$. As a consequence, $\vec{\epsilon}$ is a regular value of $H_{\mathcal{T}}$ for almost all $\epsilon > 0$. So for $\epsilon > 0$ being sufficiently small, the solutions v form a line which does not intersect lower-dimensional faces of σ. Hence, the line intersects exactly two n-faces of σ. These two n-faces cannot change as $\epsilon \to 0$, since otherwise a lower dimensional face would be traversed and the perturbation lemma would be contradicted. In other words, exactly two n-faces of σ contain solutions of the equation $H_\sigma(v) = \vec{\epsilon}$ for $\epsilon > 0$ being sufficiently small. □

The PL algorithm for tracing certain components of $H_{\mathcal{T}}^{-1}(0)$ can now be easily described via the Door-In-Door-Out-Principle. Heuristically, let us imagine that the $(n+1)$-simplexes $\sigma \in \mathcal{T}$ are "rooms" in an "infinite" building $\mathcal{T}$, and the "walls" of a room σ are its n-faces τ. A wall has a "door" if it is completely labeled. Hence a room has either no or exactly two doors. The algorithm consists of passing from one room to the next, and the following rule must be obeyed: if a room is entered through one door, it must be exited through the other door Allgower and Georg (2003).

In the general PL methods a suitable starting simplex has to be constructed, but for the special case of homotopy methods the choice of the starting simplex is straightforward.

The pivoting step finds the simplex which is adjacent to the current simplex on a given facet. The implementation of the pivoting step depends on the chosen

triangulation and typically is performed using only a few operations which define the pivoting rules of the triangulation.

The door-in-door-out step finds the second completely labeled n-face of a transverse simplex. This step is computationally more expensive than the pivoting step because it involves the solving of linear equations in a manner typical for linear programming methods and so it is also referred as linear programming (LP) step.

2.5.4 Refining Triangulations of $\mathbb{R}^n \times R$

In this section we will describe some special triangulations of $\mathbb{R}^n \times \mathbb{R}$, the refining triangulations, which are used in the context of PL homotopy methods. As an example of a refining triangulation we describe the triangulation J_3 of $\mathbb{R}^n \times (0,1]$ and the corresponding pivoting rules. The J_3 triangulation is used in the new Javascript implementation of the PL homotopy algorithm of Eaves and Saigal (1972) and is related to the triangulation K_3 used in the original implementation. Some more details and different approaches in the description of the J_3 triangulation are available in Allgower and Georg (2003), Todd (1976b) and Todd (1977).

We will give the definition of a refining triangulation of the space $\mathbb{R}^n \times (0,1]$.

(2.5.14) Definition We consider a triangulation $\mathcal{T}$ of $\mathbb{R}^n \times (0,1]$ into $(n+1)$-simplices, such that every simplex in the triangulation is contained in some slab of $\mathbb{R}^n \times (0,1]$, that is for every simplex $\sigma \in \mathcal{T}$ we have:

$$\sigma \subset \mathbb{R}^n \times \left[\frac{1}{2^k}, \frac{1}{2^{k+1}}\right], \quad k \in \mathbf{N}.$$

Let $\sigma = [v_0, v_1, \ldots, v_{n+1}] \in \mathcal{T}$ be an $(n+1)$-simplex and let $\pi : \mathbb{R}^n \times \mathbb{R} \to \mathbb{R}$ be the following canonical projection: $\pi(x_{[0]}, x_{[1]}, \ldots, x_{[n]}) = x_{[n]}$.

We define the **level** of σ as is the maximum of the last co-ordinates of all vertices of σ, that is:

$$\max \pi(v_i), \quad i = 0, 1, \ldots, n+1$$

We call $\mathcal{T}$ a **refining triangulation** if the diameter of σ tends to zero as the level of σ tends to zero.

For describing the J_3 triangulation and the corresponding pivoting rules we follow the exposition given by Saigal (1979). The vertices of the triangulation J_3 are given by the set of points:

(2.5.15)

$$J_3^0 = \left\{ v = (v_{[0]}, \ldots, v_{[n]}) \middle| v_{[n]} = \frac{1}{2^k},\ k \in \mathbf{N} \text{ and } \frac{v_{[i]}}{v_{[n]}} \in \mathbb{Z},\ i = 0, \ldots, n-1 \right\}$$

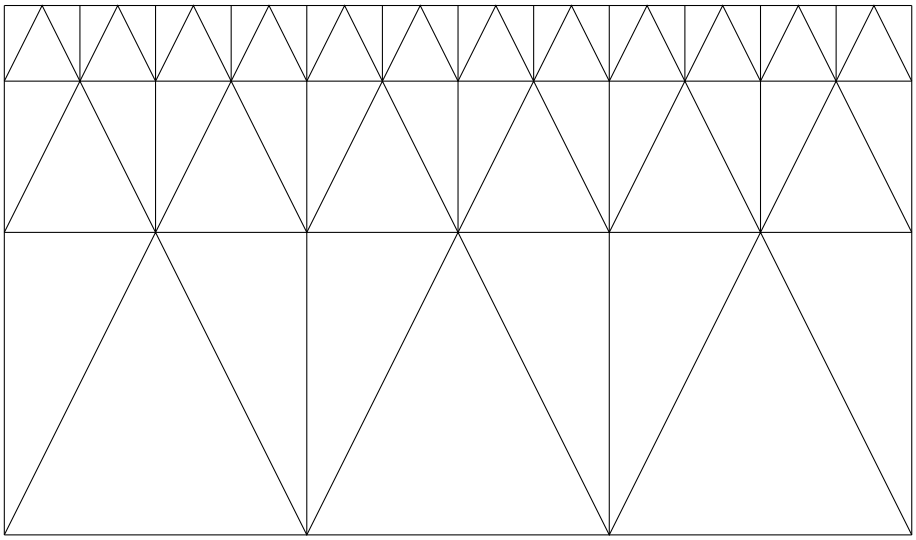


Figure 2.8 – J_3 triangulation in $\mathbb{R} \times \mathbb{R}$

In addition, the following subset of vertices in J_3^0 are called central vertices:

$$\textbf{(2.5.16)} \quad \overline{J}_3^0 = \left\{ v = (v_{[0]}, \ldots, v_{[n]}) \in J_3^0 \;\middle|\; \frac{v_{[i]}}{v_{[n]}} \text{ is an odd integer, } i = 0, \ldots, n-1 \right\}.$$

A vertex $v \in J_3^0$ has the level k if $v[n] = \frac{1}{2^k}$. Each central vertex of depth $k \geq 1$ has an unique nearest central vertex of depth $k-1$ obtained as:

$$\textbf{(2.5.17)} \quad y(v) = v - v_{[n]} \cdot \mu(v)$$

where

$$\mu(v) = (\mu_{[0]}(v), \mu_{[1]}(v), \ldots, \mu_{[n]}(v))$$

$$\textbf{(2.5.18)} \quad \mu_{[i]}(v) = \begin{cases} -1, & \text{if } \frac{v_{[i]}}{v_{[n]}} \bmod 4 = 1 \\ \\ +1, & \text{if } \frac{v_{[i]}}{v_{[n]}} \bmod 4 = 3 \end{cases}$$

Each simplex $\sigma \in J_3$ has an unique representation by a triplet (v, π, s) where $v = (v[0], \ldots, v[n])$ is a central vertex, π a permutation of $\{0, \ldots, n\}$ and $s = (s_{[0]}, \ldots, s_{[n]})$ with $s_{[i]} \in \{-1, +1\}$. The vertices of this simplex are obtained as follows:

$$\textbf{(2.5.19)} \quad \begin{aligned} &\gamma_0 = v \\ &\gamma_{i+1} = \gamma_i + v_{[n]} s_{[\pi_{[i]}]} e_{\pi_{[i]}} && i = 0, \ldots, j-1 \\ &\gamma_{j+1} = \gamma_j - v_{[n]} \sum_{k=j+1}^{n} \mu_{[\pi_{[k]}]} e_{\pi_{[k]}} + v_{[n]} e_n \\ &\gamma_{k+1} = \gamma_k + 2 v_{[n]} \mu_{[\pi_{[k]}]} e_{\pi_{[k]}} && j+1 \leq k \leq n \end{aligned}$$

where $\mu = \mu(v)$, $(e_0, \ldots, e_n)$ is the standard unit basis in $\mathbb{R}^{n+1}$ and j is the index such that $\pi_{[j]} = n$. A proof that J_3 is indeed a triangulation is available in Todd (1977).

Next we will present the pivoting rules for the J_3 triangulation. Let (v, π, s) be a simplex in J_3, and let $\pi_{[j]} = n$. Also, let $\gamma_0, \ldots, \gamma_{n+1}$ be the vertices of this simplex generated by the above relations.

In order to simplify the recursive relations, we define the n-dimensional vector b with components $b_{[i]} \in \{-1, +1\}$, $i = 0, 1, \ldots, n-1$, such that μ and s are combined in a single vector:

(2.5.20)

$$\begin{aligned} b_{[\pi_{[i]}]} &= s_{[\pi_{[i]}]} \quad 0 \le i \le j-1 \\ b_{[\pi_{[i]}]} &= \mu_{[\pi_{[i]}]} \quad j+1 \le i \le n \end{aligned}$$

We assume that γ_i is the vertex which has to be pivoted. The new simplex (v', π', s') is obtained by the following rules.

$i = 0$ $j = 0$	$v' = v - v_{[n]}(-1, b_{[0]}, \ldots, b_{[n-1]})$ $\pi' = (\pi_{[1]}, \ldots, \pi_{[n]}, \pi_{[0]})$ $b' = b$
$i = 0$ $j > 0$	$v' = v + 2v_{[n]} b_{[\pi_{[0]}]} e_{\pi_{[0]}}$ $\pi' = \pi$ $b' = b - 2b_{[\pi_{[0]}]} e_{\pi_{[0]}}$
$1 \le i \le j-1$	$v' = v$ $\pi' = (\pi_{[0]}, \ldots, \pi_{[i+1]}, \pi_{[i]}, \ldots, \pi_{[n]})$ $b' = b$
$i = j$ $b_{[\pi_{[j-1]}]} = \mu_{[\pi_{[j-1]}]}$	$v' = v$ $\pi' = (\pi_{[0]}, \ldots, \pi_{[j]}, \pi_{[j-1]}, \ldots, \pi_{[n]})$ $b' = b$
$i = j$ $b_{[\pi_{[j-1]}]} = -\mu_{[\pi_{[j-1]}]}$	$v' = v$ $\pi' = (\pi_{[0]}, \ldots, \pi_{[j-2]}, \pi_{[j]}, \ldots, \pi_{[n]}, \pi_{[j-1]})$ $b' = b - 2b_{[\pi_{[j-1]}]} e_{\pi_{[j-1]}}$
$j+1 \le i \le n$	$v' = v$ $\pi' = (\pi_{[0]}, \ldots, \pi_{[i+1]}, \pi_{[i]}, \ldots, \pi_{[n]})$ $b' = b$
$i = n+1$ $j < n$	$v' = v$ $\pi' = (\pi_{[0]}, \ldots, \pi_{[j-1]}, \pi_{[n]}, \pi_{[j]}, \ldots, \pi_{[n-1]})$ $b' = b - 2b_{[\pi_{[n]}]} e_{\pi_{[n]}}$
$i = n+1$ $j = n$	$v' = v + \frac{1}{2} v_{[n]}(-1, b_{[0]}, \ldots, b_{[n-1]})$ $\pi' = (\pi_{[n]}, \pi_{[0]}, \ldots, \pi_{[n-1]})$ $b' = b$

Because the performance of simplicial algorithms is very sensitive to the triangulation used, further study of this subject is required and some interesting papers are mentioned below. Rivara (1984) describes two general algorithms for refining triangular computational meshes based on the bisection of triangles by the longest side which can be applied globally or locally for selective refinement of any conforming triangulation and always generate a new conforming triangulation. Persiano et al. (1993) give general scheme for adaptive triangulation refinements, where the simplices of 3D triangulation are bisected as part of refinement and with an application defined refinement criterion. Weiss and Floriani (2011) describe approaches to hierarchical spatial decompositions that focus on a specific dimension and that apply to all dimensions.

2.5.5 Implementation Details of the PL Homotopy Algorithm

The idea of the piecewise linear continuation algorithm can be used to approximate a fixed point of a continuous bounded map $F : \mathbb{R}^n \to \mathbb{R}^n$. We consider the homotopy:

(2.5.21) $$H : \mathbb{R}^{n+1} \to \mathbb{R}^n, \quad H(x,\lambda) = x - \lambda x_0 - (1-\lambda)F(x)$$

and we try to follow the solution path from $(x_0, 1)$ to $(x^*, 0)$, where x_0 is the initial approximation of the solution and x^* is a fixed point of F which we try to approximate. Because there are no smoothness conditions on F, we use the piecewise linear continuation algorithm in conjunction with a refining triangulation $\mathcal{T}$ of $\mathbb{R}^n \times (0,1]$, see definition **2.5.14**.

We define the piecewise linear map $H_{\mathcal{T}}$ which interpolates H on the vertices of the given refining triangulation J_3.

- $H_{J_3}(x,1) = x_0$
- $H_{J_3}(x,0) = F(x)$
- $H_{J_3}(x,t) = \sum_{i=0}^{n+1} \lambda_i H(v_i,t)$, where:
  (v_i,t) are vertices of a simplex $\sigma \in J_3$
  $(x,t) = \sum_{i=0}^{n+1} \lambda_i (v_i,t), \quad \sum_{i=0}^{n+1} \lambda_i = 1,\ \lambda_i \geq 0.$

It can be shown that $\mathcal{T}$ induces a triangulation on the frontier of $\mathbb{R}^n \times (0,1]$ so we have an induced triangulation over $\mathbb{R}^n \times \{1\}$. If we assume that the starting point $u_0 := (x_0, 1)$ is in the interior of a facet τ_0, then it is clear that τ_0 is the only completely labeled facet on $\mathbb{R}^n \times \{1\}$. So the algorithm cannot terminate on the boundary $\mathbb{R}^n \times \{1\}$, so it generates a sequence $\tau_0, \tau_1, \ldots$ of completely labeled facets. It is possible to follow the polygonal solution path parametrized by arclength $0 \leq s < s_0 \leq \infty$, $c(s) = (x(s), \lambda(s))$ in $H^{-1}(0)$, starting at $c(0) = (x_0, 1)$. From the

boundedness of the map it follows that $\lambda(s)$ tends to zero as s tends to s_0 and

(2.5.22) $$\lim_{s \to s_0} ||x(s) - F(x(s))|| = 0$$

Since $x(s)$ remains bounded as s tends to s_0, this implies that every accumulation point of $x(s)$ is a fixed point of F.

The input data required by the algorithm is:

- n - the dimension of the problem
- $F : \mathbb{R}^n \to \mathbb{R}^n$ - the function for which we calculate a fixed point, specified as an expression depending on the parameters $x_1, x_2, \ldots, x_n$.
- $v_1, \ldots, v_{n+1} \in \mathbb{R}^n$ - the vertices of a starting simplex
- $x_0 \in \mathbb{R}^n$ - an initial approximation of the fixed point

There are also some parameters for flow controlling:

- the minimum required contraction factor for the Newton steps
- the maximal allowed bisection level of the triangulation
- the maximal number of steps to be performed
- the stopping tolerance for $||F||$

If an approximate solution is found the algorithm will output this solution and the number of steps executed to find it. During the execution, the algorithm will output the current point of the followed polygonal path.

In the linear programming step we try to find the second completely labeled n-face of a transverse simplex. This is usually a computational intensive step since involves solving of linear equations in a manner typical for linear programming methods, so it has to be implemented using efficient methods.

We denote the current transverse $(n+1)$-simplex by $\sigma_n = [v_1, \ldots, v_{n+2}]$ and its current completely labeled n-face by $\tau_{n-1} = [v_1, \ldots, v_{n+1}]$. Hence the vertex v_{n+2} was just pivoted in the preceding step. Our aim in the linear programming step is to find a completely labeled n-face τ_n of σ_n which is different from τ_{n-1}. So we have to find an index $i \in \{1, \ldots, n+1\}$ such that the right inverse of the matrix obtained by deleting the i-th column of $A = \mathcal{L}(\sigma_n)$ is lexicographically positive. One index which satisfies this is $i = N+2$, since the n-face opposite v_{n+2} is completely labeled. The other index which will be obtained is the new index we are seeking in the linear programming step.

For numerical purposes, the lexicographical operations in linear programming are usually performed only over the first coordinate of the rows. Theoretically, in cases of degeneracies, an algorithm based on this simplified test could cycle, but this is rarely observed in practice.

Using this simplification, the numerical linear algebra of the "door-in-door-out"

step consists of solving the equations

$$A\gamma = 0, \gamma[j] = -1$$

$$A\alpha = e_1$$

for $\gamma, \alpha \in \mathbb{R}^{n+2}$ and where the index j is given and corresponds to the known completely labeled n-face. Then a minimization

$$\min \left\{ \frac{\alpha[i]}{\gamma[i]} \;\middle|\; \gamma[i] > 0, i = 1, \ldots, n+2 \right\}$$

is performed to find the index i corresponding to the vertex to be pivoted next. The vertex v_i is pivoted into the vertex $\tilde{v}_i$. The corresponding label

$$y := \begin{pmatrix} 1 \\ H(\tilde{v}_i) \end{pmatrix}$$

is calculated, and the new labeling matrix $\tilde{A}$ is obtained by replacing the i-th column of A by y:

$$\tilde{A} = A + (y - Ae_i)e_i^{\intercal}.$$

At each step, a standard decomposition of $\tilde{A}$ is updated from a given decomposition of A, which enables us to solve the linear equations in each step. The cheapest such method directly updates the right inverse B of A such that $e_i^{\intercal}B = 0$ for the current pivot index i. However, this update method is not always stable, so we actually update a QR factorization of $\tilde{A}$ in each cycle.

In order to improve the convergence rate of the algorithm a technique of mixing PL steps with Newton steps can be used, see Saigal and Todd (1978). This follows from the fact that a modified Newton's method expressed in barycentric coordinates leads to a system of linear equations which is closely related to the linear equations obtained in the linear programming step and in fact it is possible to continue the updates of the labeling matrix during the Newton steps on the column corresponding to the vertex which has to be pivoted next. The Newton steps are performed when each new level $\mathbb{R}^n \times \{k\}$ is reached in the algorithm. If the contraction of the Newton steps is sufficiently strong the algorithm will continue to perform these steps until it stops with an approximate solution. If the Newton steps are not successful then the pivoting step is executed and the PL homotopy algorithm continues normally.

Due to the considerable size of the complete code, we present below just the main loop of the iterative method. For further details about the algorithm implementation, the complete version of the code and the suite of problems used for practical applications in this thesis is available online at `http://algonum.appspot.com/plh`.

```
function solve(labelFunc, x0, options)
{
  var res = {};
  var level = -1;
  var old_level;
  var new_index = 0;
  var n = 0;
  // ... other internal variables

  // initialize internal data structures
  init();
  // log information about the step in the result
  add_step_info();

  // perform iterations
  while (true)
  {
    n++;
    old_level = level;
    // compute triangulation level
    level = Math.round(-Math.log(Math.abs(depth[new_index])) / Math.log
    (2.0));

    // perform pivoting operation according to triangulation pivoting
    rules
    new_index = pivot(find_pivoting_index());
    // add information about the step in the result
    // (current simplex and approximation)
    add_step_info();

    if (check_solution())
    {
      // stop when solution is found
      break;
    }

    // for new encountered level attempt Newton steps
    if (level > old_level)
    {
      if (newton())
      {
        // stop if Newton steps are succesful
        break;
      }
    }

    if (level > options.max_bisections)
    {
      // stop if triangulation level is too high
      res.errorMessage = "Maximal triangulation level reached.";
```

```
49        break;
50      }
51      if (n > options.max_steps)
52      {
53        // stop after a maximum number of iterations
54        res.errorMessage = "Maximal number of iteration steps reached.";
55        break;
56      }
57
58      // compute the label for the new current vertex after pivoting
59      label();
60      // door-in-door-out step
61      // finds the second completely labeled facet
62      find_facet();
63    }
64
65    res.numsteps = n;
66    res.x0 = x0;
67    if (res.errorMessage === undefined)
68    {
69      res.xn = x;
70    }
71
72    return ret;
73  }
```

Listing 2.2 – Simplified (incomplete) implementation of the PL homotopy fixed point algorithm

3. APPLICATIONS

3.1 PRACTICAL APPLICATIONS OF FIXPOINT

In this section there are presented several practical applications of the FIXPOINT software package, demonstrating the usage of the implementations and the new features. Some parts of this section are also included in Bozantan and Berinde (2014b) and Bozantan (2014b).

3.1.1 Computing Fixed Points with the Picard Iteration

Let $f : [0,1] \to [0,1]$ be defined by:

$$f = \begin{cases} \frac{2}{3}x, & x \in \left[0, \frac{1}{2}\right) \\ \frac{2}{3}x + \frac{1}{3}, & x \in \left[\frac{1}{2}, 1\right) \end{cases}$$

As shown in Păcurar (2009b), f defined above is a weak contraction with $L = \frac{2}{3}$ and $\delta = 6$ and has two fixed points: $F_f = \{0,1\}$.

We compute the fixed points using the Javascript implementation of the Picard fixed point iteration:

```
function f(x)
{
  if (x >= 0 && x < 1/2)
    return 2/3 * x;
  if (x >= 1/2 && x < 1)
    return 2/3 * x + 1/3;
}
var ret = fixpoint.picard(f, 1/4);
console.log(ret);
```

The following tables will present the Picard iterations obtained using the new Javascript implementation previously described applied to the function f, for several starting initial approximations.

The tables are obtained using an additional helper function fixpoint.compare as illustrated in the corresponding listings below.

```
var ret05 = fixpoint.picard(f, 0.05, { maxError : 1e-4 });
var ret10 = fixpoint.picard(f, 0.10, { maxError : 1e-4 });
var ret15 = fixpoint.picard(f, 0.15, { maxError : 1e-4 });
var ret25 = fixpoint.picard(f, 0.25, { maxError : 1e-4 });
var ret35 = fixpoint.picard(f, 0.35, { maxError : 1e-4 });
var ret45 = fixpoint.picard(f, 0.45, { maxError : 1e-4 });
var s = fixpoint.compare([ret05, ret10, ret15, ret25, ret35, ret45], 5, '
    latex');
console.log(s);
```

0	0.05000	0.10000	0.15000	0.25000	0.35000	0.45000
1	0.03333	0.06667	0.10000	0.16667	0.23333	0.30000
2	0.02222	0.04444	0.06667	0.11111	0.15556	0.20000
3	0.01481	0.02963	0.04444	0.07407	0.10370	0.13333
4	0.00988	0.01975	0.02963	0.04938	0.06914	0.08889
5	0.00658	0.01317	0.01975	0.03292	0.04609	0.05926
6	0.00439	0.00878	0.01317	0.02195	0.03073	0.03951
7	0.00293	0.00585	0.00878	0.01463	0.02048	0.02634
8	0.00195	0.00390	0.00585	0.00975	0.01366	0.01756
9	0.00130	0.00260	0.00390	0.00650	0.00910	0.01171
10	0.00087	0.00173	0.00260	0.00434	0.00607	0.00780
11	0.00058	0.00116	0.00173	0.00289	0.00405	0.00520
12	0.00039	0.00077	0.00116	0.00193	0.00270	0.00347
13	0.00026	0.00051	0.00077	0.00128	0.00180	0.00231
14	0.00017	0.00034	0.00051	0.00086	0.00120	0.00154
15		0.00023	0.00034	0.00057	0.00080	0.00103
16		0.00015	0.00023	0.00038	0.00053	0.00069
17			0.00015	0.00025	0.00036	0.00046
18				0.00017	0.00024	0.00030
19					0.00016	0.00020
20						0.00014

```

var ret50 = fixpoint.picard(f, 0.50, { maxError : 1e-4 });
var ret60 = fixpoint.picard(f, 0.60, { maxError : 1e-4 });
var ret70 = fixpoint.picard(f, 0.70, { maxError : 1e-4 });
var ret75 = fixpoint.picard(f, 0.75, { maxError : 1e-4 });
```

```
var ret80 = fixpoint.picard(f, 0.80, { maxError : 1e-4 });
var ret90 = fixpoint.picard(f, 0.90, { maxError : 1e-4 });
var s = fixpoint.compare([ret50, ret60, ret70, ret75, ret80, ret90], 5, '
    latex');
console.log(s);
```

0	0.50000	0.60000	0.70000	0.75000	0.80000	0.90000
1	0.66667	0.73333	0.80000	0.83333	0.86667	0.93333
2	0.77778	0.82222	0.86667	0.88889	0.91111	0.95556
3	0.85185	0.88148	0.91111	0.92593	0.94074	0.97037
4	0.90123	0.92099	0.94074	0.95062	0.96049	0.98025
5	0.93416	0.94733	0.96049	0.96708	0.97366	0.98683
6	0.95610	0.96488	0.97366	0.97805	0.98244	0.99122
7	0.97074	0.97659	0.98244	0.98537	0.98829	0.99415
8	0.98049	0.98439	0.98829	0.99025	0.99220	0.99610
9	0.98699	0.98960	0.99220	0.99350	0.99480	0.99740
10	0.99133	0.99306	0.99480	0.99566	0.99653	0.99827
11	0.99422	0.99538	0.99653	0.99711	0.99769	0.99884
12	0.99615	0.99692	0.99769	0.99807	0.99846	0.99923
13	0.99743	0.99794	0.99846	0.99872	0.99897	0.99949
14	0.99829	0.99863	0.99897	0.99914	0.99931	0.99966
15	0.99886	0.99909	0.99931	0.99943	0.99954	
16	0.99924	0.99939	0.99954	0.99962	0.99970	
17	0.99949	0.99959	0.99970			
18	0.99966	0.99973				

3.1.2 Comparison of Krasonselskii Iterations

Let $f : \left[\frac{1}{2}, 2\right] \to \left[\frac{1}{2}, 2\right]$, $f(x) = \frac{1}{x}$. As shown in Berinde (2007), we have:

- f is Lipschitzian with constant $L = 4$;
- f is strongly pseudocontractive with any constant $k \in (0, 1)$;
- $F_f = 1$;
- The Picard iteration associated to f does not converge to the fixed point of f for any $x_0 \neq 1$;
- The Krasnoselskii iteration associated to f converges to the fixed point $p = 1$, for any x_0 and $\lambda \in (0, \frac{1}{16})$.

Now we use the new implementation of the Krasnoselskii iteration in order to compute the fixed point of f for several various values of λ, and then to compare the results.

```
var lambdas = [0.05, 0.10, 0.25, 0.33, 0.5, 0.66, 0.75, 0.9, 0.95]
var iterations = [];
var item;
for (var i = 0; i < lambdas.length; i++)
{
  iterations[i] = fixpoint.krasnoselskii(f, lambdas[i], 2,
    { maxError: 1e-3, convergenceTest: fixpoint.errtest.absolute });
}
fixpoint.compare(iterations, 4, 'latex');
```

λ	0.05	0.10	0.25	0.33	0.5	0.66	0.75	0.9	0.95
0	2.0000	2.0000	2.0000	2.0000	2.0000	2.0000	2.0000	2.0000	2.0000
1	1.9250	1.8500	1.6250	1.5050	1.2500	1.0100	0.8750	0.6500	0.5750
2	1.8547	1.7191	1.3726	1.2276	1.0250	0.9969	1.0759	1.4496	1.6809
3	1.7889	1.6053	1.2116	1.0913	1.0003	1.0010	0.9661	0.7658	0.6492
4	1.7274	1.5071	1.1150	1.0336	1.0000	0.9997	1.0179	1.2518	1.4958
5	1.6700	1.4227	1.0605	1.0118		1.0001	0.9913	0.8441	0.7099
6	1.6165	1.3507	1.0311	1.0040			1.0044	1.1506	1.3737
7	1.5666	1.2897	1.0158	1.0014			0.9978	0.8973	0.7603
8	1.5202	1.2383	1.0080	1.0005			1.0011	1.0928	1.2876
9	1.4770	1.1952	1.0040				0.9995	0.9329	0.8022
10	1.4370	1.1593	1.0020				1.0003	1.0580	1.2244
11	1.4000	1.1297	1.0010					0.9564	0.8371
12	⋮	⋮						⋮	⋮
20	1.1728	1.0186						1.0060	1.0715
⋮	⋮	⋮						⋮	⋮
27	1.0865	1.0039						0.9987	0.9677
⋮	⋮							⋮	⋮
32	1.0520							1.0004	1.0196
⋮	⋮								⋮
40	1.0227								1.0084
⋮	⋮								⋮
49	1.0089								0.9968
⋮									⋮
60									1.0010
⋮									⋮
68									1.0004

3.1.3 The Mixed Error Test

Let $f : [0,1] \to [0,1]$ be defined by:

$$f = \begin{cases} \frac{2}{3}x, & x \in \left[0, \frac{1}{2}\right) \\ \\ \frac{2}{3}x + \frac{1}{3}, & x \in \left[\frac{1}{2}, 1\right) \end{cases}$$

We compute the fixed points using the Javascript implementation of the Picard fixed point iteration and using the relative error test and the mixed error test:

```
function f(x)
{
  if (x >= 0 && x < 1/2)
    return 2/3 * x;
  if (x >= 1/2 && x < 1)
    return 2/3 * x + 1/3;
}

var retRelative = fixpoint.picard(f, 0.25,
  {
    convergenceTest: fixpoint.errtest.relative,
    maxError : 1e-3,
    maxSteps : 20
  });
console.log(retRelative);


var retMixed = fixpoint.picard(f, 0.25,
  {
    convergenceTest: fixpoint.errtest.mixed,
    maxError : 1e-3,
    maxSteps : 20
  });
console.log(retMixed);
```

The following tables will present the advantage of using the mixed error test instead of the relative error test. As it is shown by the data, by using the mixed error test with a maximum error of $1e-3$, the algorithm will stop after 15 iterations, but when using the relative error test, the desired error is never achieved (since the relative error is always constant $1/3$), so the algorithm will stop only when the maximum number of steps is reached.

n	x_n	relative error	mixed error
1	0.16666667	0.3333	0.0714
2	0.11111111	0.3333	0.1250
3	0.07407407	0.3333	0.0862
4	0.04938272	0.3333	0.0588
5	0.03292181	0.3333	0.0398
6	0.02194787	0.3333	0.0268
7	0.01463192	0.3333	0.0180
8	0.00975461	0.3333	0.0121
9	0.00650307	0.3333	0.0081
10	0.00433538	0.3333	0.0054
11	0.00289025	0.3333	0.0036
12	0.00192684	0.3333	0.0024
13	0.00128456	0.3333	0.0016
14	0.00085637	0.3333	0.0011
15	0.00057091	0.3333	0.0007
16	0.00038061	0.3333	
17	0.00025374	0.3333	
18	0.00016916	0.3333	
19	0.00011277	0.3333	
20	0.00007518	0.3333	

3.1.4 Detection of Cyclic Iterations

In order to demonstrate the cycle detection feature we use as an example a sample function similar to the logistic map, which is often cited as a basic example of chaotic behaviour. $f : \mathbb{R} \to \mathbb{R}, \qquad f(x) = 3.4 \cdot x \cdot (1 - x)$

We try to compute a fixed point using the Javascript implementation of the Picard fixed point iteration, using the following code:

```
function f(x)
{
  return 3.4 * x * (1 - x);
}
var ret = fixpoint.picard(f, 1/2);
console.log(ret);
```

The fixed point algorithm returns the following error message: "Iteration is divergent (cycle detected between iterations #227 and #231)." and we show the last few intermediate approximations obtained by the algorithm:

n	x_n
220	0.45196324762615453
221	0.84215439943267100
222	0.45196324762615200
223	0.84215439943267020
224	0.45196324762615375
225	0.84215439943267090
226	0.45196324762615220
227	0.84215439943267050
228	0.45196324762615320
229	0.84215439943267060
230	0.45196324762615300
231	0.84215439943267050

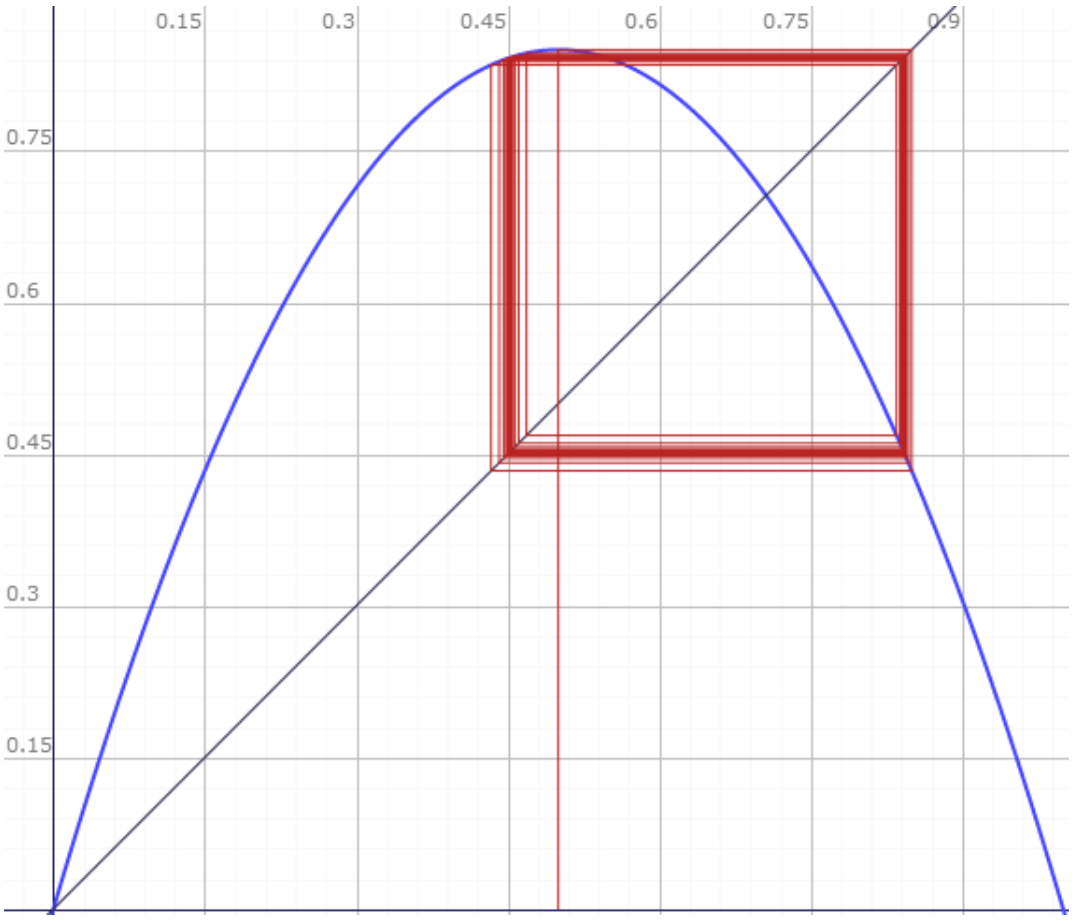


Figure 3.1 – Visualisation of Picard iteration applied to logistic map - obtained with FIXPOINT

3.1.5 Empirical study of the convergence rate

The early versions of FIXPOINT were used for the numerical experiments in the comparative study of the rate of convergence of fixed point iteration procedures, that were reported in Berinde and Păcurar (2007), and we quote here some extracts from from study.

Let $K = [0,1]$ and $T : K \to K$ be given by $Tx = (1-x)^6$.

Then T has $p_1 \approx 0.2219$ and $p_1 \approx 2.1347$ as fixed points. Both of them are repulsive fixed points with respect to the Picard iteration. However, p_1 is attractive with respect to Krasnoselskii, Mann and Ishikawa iterations, while p_2 stays repulsive, as indicated by the numerical results obtained by running the new version of the program FIXPOINT.

Krasnoselskii iteration: if we start from $x_0 = 2$ and the parameter that defines the iteration is $\lambda = 0.5$, then we obtain $x_1 = 1.5$, $x_2 = 0.757$, $x_3 = 0.379$, $x_4 = 0.2181$, $x_5 = 0.2232$ and $x_6 = 0.2214$;

Mann iteration: if we start from $x_0 = 2$ and the parameter sequence is $\alpha_n = 1/(n+1)$, then we obtain $x_1 = 1.0$, $x_2 = 0.5$, $x_3 = 0.338$, $x_4 = 0.2748$, $x_5 = 0.2489$ and $x_6 = 0.2378$;

Ishikawa iteration: if we start from $x_0 = 2$ and the parameter sequences are $\alpha_n = 1/(n+1)$ and $\beta_n = 1/(n+2)$, respectively, then we obtain $x_1 = 0.01$, $x_2 = 0.55$, $x_3 = 0.346$, $x_4 = 0.2851$, $x_5 = 0.2527$ and $x_6 = 0.2392$;

These empirical results suggest that Krasnoselskii iteration converges faster than both Mann and Ishikawa iterations. This fact is more clearer illustrated if we choose $x_0 = p_2$, the repulsive fixed point of T: after 20 iterations, Krasnoselskii method gives $x_{20} = 0.2219$, while Mann and Ishikawa iteration procedures give $x_{20} = 0.6346$ and $x_{20} = 0.6347$, respectively. The convergence of Mann and Ishikawa iteration procedures is indeed very slow in this case: after 500 iterations we get $x_{500} = 0.222$ for both methods.

Note that for $x_0 \in \{-2, 3, 4\}$ and the previous values of the parameters λ, α_n and β_n, all three iteration procedures: Krasnoselskii, Mann and Ishikawa, converge to 1, which is not a fixed point of T.

Starting from such kind of numerical results, it was tried to infer that, for certain classes of mappings, Picard iteration *always* converges faster than Mann or Ishikawa iterations. The first results of this kind have been reported in Berinde (2004a) and then continued in Berinde and Berinde (2005), Berinde and Păcurar (2007), Berinde (2007). This opened a fruitful direction of research that has been later considered by many other authors, see, for an incomplete list, Akbulut and

Ozdemir (2012), Alotaibi et al. (2013), Babu and Prasad (2006, 2007) Duong (2012), Hussain et al. (2012, 2013, 2011), Kang et al. (2013), Karahan and Ozdemir (2013), Khan et al. (2014), Kumar (2013), Olaleru (2007, 2009), Phuengrattana and Suantai (2012), Popescu (2007), Xue (2008), Rhoades and Xue (2010).

3.2 APPLICATIONS OF PL HOMOTOPY ALGORITHM TO NONSMOOTH OPTIMIZATION

3.2.1 Overview

The main aim of this section is to illustrate the relevance of the piecewise-linear homotopy algorithm described in the previous section, for solving unconstrained optimization problems and to compare its performances to other well known iterative methods. We consider the unconstrained optimization problem

$$\min f(x),\ x \in \mathbb{R}^n, \tag{3.2.1}$$

where $f : \mathbb{R}^n \to \mathbb{R}$ is continuously differentiable. There exists many (Newton type) iterative methods in literature for solving problem **(3.2.1)**, see for example Edgar and Himmelblau (1989), Güler (2010), Kelley (1999), and various studies and improvements of these methods, see for example Cătinaş (2001, 2002), Măruşter et al. (2012). Such an iterative method produces a monotone or non monotone sequence $x_0, x_1, x_2, \ldots$, where x_{k+1} is generated from x_k, the current direction d_k, and the stepsize α_k by the rule

$$x_{k+1} = x_k + \alpha_k d_k. \tag{3.2.2}$$

Amongst the most reliable and largely used iterative methods in unconstrained optimization, Newton's method and Newton type methods play an important role, due to the fact that they allow us to identify by a certain procedure the search directions.

In order to decide which one of these methods should be more convenient for a certain problem, it would be desirable to know *a priori*, if possible, a scale of the most efficient and robust techniques, for as many as possible classes of objective functions.

Efficiency is an important feature of any iterative procedure, since in concrete problems for more than three or four variables trial and error becomes impractical because, in some regions, the optimization algorithm may progress very slowly toward the optimum, requiring excessive computer time.

Robustness, i.e., the ability to achieve a solution, is equally or even more important because a general nonlinear function is unpredictable in its behaviour: there may be local maxima or minima, saddle points, regions of convexity, concavity, and so on. Therefore, it is of great theoretical and practical importance to draw an extensive experience in testing optimization algorithms for unconstrained functions to evaluate their efficiency and robustness.

This new approach to unconstrained optimization problems is based on the fact that algorithm **(3.2.2)** can be regarded as a particular case of a classic fixed point iterative method, that is, of the Picard iteration or successive approximations method associated to a certain nonlinear fixed point equation

(3.2.3) $$x = Tx,$$

where T is a given self operator of a space X. Suppose X and T are such that the equation **(3.2.3)** has at least one solution (usually called a *fixed point* of T). A typical situation of this kind is illustrated by the well known Brouwer's fixed point theorem, see **2.2.4**. Under the assumptions of Brouwer Theorem, the Picard iteration associated to **(3.2.3)**, defined by $x_0 \in X$ and

(3.2.4) $$x_{n+1} = Tx_n,\ n = 0, 1, 2, \ldots,$$

does not converge, in general, even though in many cases (e.g., for contractive type mappings, see Berinde (2007)) it is a useful method to solve nonlinear fixed point equations.

Any contraction mapping is continuous but the reverse is not true. This is the reason why several authors tried to find specific algorithms that could be successfully used to compute fixed points of continuous but not contractive mappings. This thesis describes two such algorithms: the simplicial fixed point algorithm for continuous mappings proposed by Scarf (1967) and the PL homotopy fixed point algorithm proposed by Eaves and Saigal (1972).

The PL homotopy fixed point algorithm discussed in section 2.5 is part of the class of homotopy continuation algorithms. This category of algorithms can be used to solve various problems, like:

- the economic equilibrium problems, see Herings et al. (1996), Scarf (1983), Hansen and Scarf (1969),
- game theory problems, see Herings and van den Elzen (2002), Herings and Peeters (2010)
- nonlinear constrained optimization problems, see Merrill (1972), Watson and Haftka (1989), Allgower and Georg (2000), Bozantan and Berinde (2014b)
- nonlinear boundary value problems, see Allgower and Georg (1993), Allgower and Jeppson (1973),
- finding roots of systems of nonlinear equations, see Saigal (1977),
- finding roots of complex polynomials, see Verschelde (1999), Li (1997), Kuhn (1974)
- nonlinear complementarity problem, see Murty (1988)

We also mention as other (more practical) possible applications of the PL homotopy algorithm: computing fixed points of non differentiable functions in elastic contact

problems with friction where the problem encountered is the minimization of a function with non differentiable terms (in particular the absolute value function) or in discrete contact problems with friction formulated as fixed point problems (the fixed point being the contact tension), see for example Pop (2008, 2009), Ligurskỳ (2012), Bozantan et al. (2014)

Following the descriptions given by Allgower and Georg (2003) and Saigal (1979) we describe the transformation of nonsmooth optimization problems as fixed point problems.

A function $\theta : \mathbb{R}^n \to \mathbb{R}$ is called **convex** if

$$\lambda_1\theta(x_1) + \lambda_2\theta(x_2) \geq \theta(\lambda_1 x_1 + \lambda_2 x_2)$$

holds for all convex combinations $\lambda_1, \lambda_2 \geq 0,\ \lambda_1 + \lambda_2 = 1,\ x_1, x_2 \in \mathbb{R}^n$.

For a convex function θ, a point $s \in \mathbb{R}^n$ is called a **subgradient** at $x \in \mathbb{R}^n$, if for any $z \in \mathbb{R}^n$ we have:

$$\theta(z) \geq \theta(x) + \langle s, z - x\rangle$$

The **subdifferential** of θ at x is the set $\partial\theta(x)$ of all subgradients of θ at x, that is:

$$\partial\theta(x) = \{s \in \mathbb{R}^n \mid \theta(z) \geq \theta(x) + \langle s, z - x\rangle,\ \ \forall z \in \mathbb{R}^n\}$$

The subdifferential of convex function has the property that it is nonempty, closed and convex at every point.

We consider the constrained optimization problem defined by

(3.2.5) $$\min_x\{\theta(x) \mid g_i(x) \leq 0,\ \ i = 0, 1, \ldots, m\}$$

where $\theta : \mathbb{R}^n \to R$ and $g_i : \mathbb{R}^n \to R$. In addition we also assume that each of these functions is convex, and that in the case the set $G = \{x \mid g_i(x) \leq 0,\ \ i = 0, \ldots, m\}$ is nonempty, there exists x, such that $g_i(x) < 0$ for each $i = 0, 1, \ldots, m$.

We define the mapping

$$s(x) = \max_{0 \leq i \leq m} g_i(x)$$

It is simple to show that $s(x)$ is convex. We also define

$$I(x) = \{i \mid s(x) = g_i(x)\}$$

Then, we have

$$\partial s(x) = \overline{\text{co}}\left\{\bigcup_{i \in I(x)} \partial g_i(x)\right\}$$

Now, we consider the following mapping from $\mathbb{R}^n$ into convex subsets of $\mathbb{R}^n$:

$$f(x) = \begin{cases} x - \partial\theta(x) & s(x) < 0 \\ x - \overline{\text{co}}\{\partial\theta(x) \cup \partial s(x)\} & s(x) = 0 \\ x - \partial s(x) & s(x) > 0 \end{cases}$$

The above mapping is upper hemicontinuous, and we will show that one of its fixed point x^* solves the optimization problem **3.2.5**. Depending on the value of $s(x^*)$, we have the following three cases:

1. $s(x^*) < 0$. Then $0 \in \partial\theta(x^*)$ and thus, x^* is a global minimizer of θ, and is also in G, and thus solves the problem.
2. $s(x^*) > 0$. Then $0 \in \partial s(x^*)$, and x^* is a global minimizer of s, and since $s(x^*) > 0$, the set G is empty.
3. $s(x^*) = 0$. Then $0 \in \overline{\text{co}}\{\partial\theta(x^*) \cup \partial s(x^*)\}$ and thus there exist the nonenegative numbers ρ_0, ρ_i for $i \in I(x^*)$, and the vectors $y_0 \in \partial\theta(x^*)$ and $y_i \in \partial g_i(x^*), i \in I(x^*)$, such that
$$\rho_0 y_0 + \sum_{i \in I(x^*)} \rho_i y_i = 0$$
If $\rho_0 = 0$, then since x^* is a global minimizer of s, it is impossible that $g_i(x) < 0$ for each i. Since we have assumed the contrary, $\rho_0 \neq 0$, and we have x^* and $\frac{\rho_i}{\rho_0}$ for $i \in I(x)$ satisfying the standard Karush-Kuhn-Tucker necessary and sufficient conditions for a solution to the problem.

Thus, solving the optimization problem **3.2.5** can be reduced to finding a fixed point of the mapping f.

Since the piecewise-linear homotopy method could be in particular applied to solve optimization problems, the following samples are used to test numerically the efficiency and robustness of this algorithm in the case of unconstrained optimization problems and to compare it with some of the well known and widely used methods in optimization: the Newton's method, Broyden-Fletcher-Goldfarb-Shanno, conjugate gradient method, and nonlinear conjugate gradient method. This empirical study is done on a small set of representative test functions taken from literature, see Floudas et al. (1999), Hedar (2014), Moré et al. (1981), Pohlheim (2006).

In order to find local or global minima of a nonlinear multivariable function $F : \mathbb{R}^n \to \mathbb{R}$, the following homotopy map is defined and used in the PL homotopy algorithm for all the considered examples:

(3.2.6) $$H : \mathbb{R}^n \times \mathbb{R} \to \mathbb{R}^n,\ H(x,t) = \begin{cases} x - x_0 & \text{for } t \leq 0, \\ \nabla F(x) & \text{for } t > 0. \end{cases}$$

We shall use the following symbols in the comparison tables that synthesise the numerical experiments we have done:

- BFGS = Broyden-Fletcher-Goldfarb-Shanno method (a quasi-Newton method)
- CG = conjugate gradient method
- NCG = nonlinear conjugate gradient method
- PLH = piecewise linear homotopy algorithm
- ∞ = method does not converge
- $\neq$ = method converges, but not to the correct global/local minimum

All the numerical results are computed to a maximal final error of $1e-4$.

We also include sample outputs produced by the PLH algorithm for the test functions. In the sample outputs x is the current intermediate approximation of the solution and S is the current simplex processed in the algorithm. Each column of the printed matrix represents a vertex of the simplex. The simplex S is printed in a compact form, that is, we print just n of the $n+1$ coordinates of a vertex. The last implicit coordinate which is not printed is determined by the other coordinates and by the current level of the triangulation (the last parameter of the homotopy used in the algorithm).

3.2.2 Bukin function N. 6

$$F(x,y) = 100 \cdot \sqrt{|y - 0.01 \cdot x^2|} + 0.01 \cdot |x+10|,\ (x,y) \in \mathbb{R}^2.$$

The global minimum value of F is $F(-10,1) = 0$. It is usually evaluated on the rectangle $(x,y) \in [-15,-5] \times [-3,3]$. The sixth Bukin function has many local minima, all of which lie in a ridge. Note also that F is not differentiable on the set $\{(-10,b) : b \in \mathbb{R}\} \cup \{(10\sqrt{|b|},b) : b \in \mathbb{R}\}$.

x_0, y_0	-15,0	-10,0	-11,3	-13, -3
Newton iterations	∞	∞	∞	∞
BFGS iterations	∞	∞	∞	∞
BFGS func evals				
BFGS gradient evals				
CG iterations	∞	∞	∞	∞
CG func evals				
CG gradient evals				
NCG iterations	∞	∞	∞	∞
NCG func evals				
NCG gradient evals				
NCG hessian evals				
PLH iterations	199	35	92	104
PLH Newton steps	109	29	54	54
PLH label evals	308	64	146	158

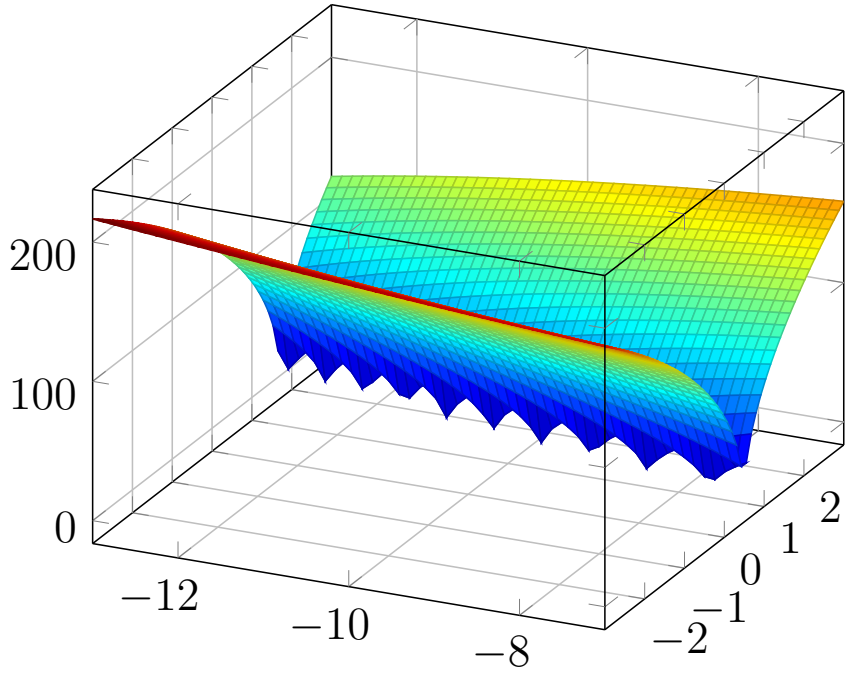


Figure 3.2 – Bukin Function N. 6

From the table above we note that all algorithms taken into consideration in our comparison study for the PL homotopy algorithm do not converge in the case of the starting points $(-15, 0), (-10, 0), (-11, 3), (-11, -3)$ (and, of course, for many other starting points). This is mainly due to the fact that the objective function F is not differentiable at some points.

The merit of the PL homotopy algorithm is that it converges in all those cases and reaches the minimal value.

Below are some fragments from the output produced by the algorithm for the initial point $(-13, -3)$.

```
1. PLH triangulation level =  0
S = [ -12.8333 -12.3333 -13.3333 -13.3333 ]
    [  -3.1667  -2.6667  -2.6667  -3.6667 ]
x = [ -13.3333  -3.6667 ]

2. PLH triangulation level =  1
S = [ -12.8333 -12.3333 -13.3333 -12.8333 ]
    [  -3.1667  -2.6667  -2.6667  -2.6667 ]
x = [ -12.8333  -2.6667 ]
Attempting Newton steps.
x = [ -12.9131  -2.6739 ]
x = [ -12.9594  -2.1550 ]
Newton steps failed.
```

```
3. PLH triangulation level =  1
S = [ -12.8333 -12.3333 -13.3333 -12.8333 ]
    [  -2.1667  -2.6667  -2.6667  -2.6667 ]
x = [ -12.8333  -2.1667 ]

4. PLH triangulation level =  1
S = [ -12.8333 -12.3333 -13.3333 -13.3333 ]
    [  -2.1667  -2.6667  -2.6667  -1.6667 ]
x = [ -13.3333  -1.6667 ]

5. PLH triangulation level =  1
S = [ -12.8333 -12.3333 -12.3333 -13.3333 ]
    [  -2.1667  -2.6667  -1.6667  -1.6667 ]
x = [ -12.3333  -1.6667 ]

...

50. PLH triangulation level =  13
S = [ -11.2702 -11.2701 -11.2703 -11.2701 ]
    [   1.2702   1.2701   1.2703   1.2703 ]
x = [ -11.2701   1.2703 ]
Attempting Newton steps.
x = [ -11.2702   1.2702 ]
x = [ -11.2702   1.2702 ]
Newton steps failed.

...

99. PLH triangulation level =  12
S = [ -10.0003 -10.0001 -10.0001  -9.9998 ]
    [   0.9998   0.9998   1.0001   0.9998 ]
x = [ -10.0001   0.9998 ]
Attempting Newton steps.
x = [ -10.0001   0.9999 ]
x = [  -9.9999   1.0001 ]
Newton steps failed.

100. PLH triangulation level =  12
S = [ -10.0000 -10.0001 -10.0001  -9.9998 ]
    [   1.0000   0.9998   1.0001   0.9998 ]
x = [ -10.0000   1.0000 ]

101. PLH triangulation level =  13
S = [ -10.0000 -10.0001 -10.0001 -10.0001 ]
    [   1.0000   0.9998   1.0001   1.0000 ]
```

```
x = [ -10.0001   1.0000 ]
Attempting Newton steps.
x = [ -10.0001   1.0000 ]
x = [ -10.0000   1.0003 ]
x = [ -10.0000   1.0003 ]
Newton steps failed.

102. PLH triangulation level =  13
S = [ -10.0000 -10.0000 -10.0001 -10.0001 ]
    [   1.0000   1.0000   1.0001   1.0000 ]
x = [ -10.0000   1.0000 ]

103. PLH triangulation level =  14
S = [ -10.0000 -10.0000 -10.0000 -10.0001 ]
    [   1.0000   1.0000   1.0000   1.0000 ]
x = [ -10.0000   1.0000 ]
Attempting Newton steps.
x = [ -10.0000   1.0000 ]
x = [ -10.0000   1.0000 ]
Newton steps failed.

104. PLH triangulation level =  14
S = [ -10.0000 -10.0000 -10.0000 -10.0000 ]
    [   1.0000   1.0000   1.0000   1.0000 ]
x = [ -10.0000   1.0000 ]

Approximate solution found.
x = [ -10.0000   1.0000 ]
f(x) =    0.2853
PLH iterations:  104
Newton steps:    54
Label evals:     158
```

3.2.3 Dixon-Price function

(3.2.7) $$f(x) = (x_1 - 1)^2 + \sum_{i=2}^{d} i \cdot (2 \cdot x_i^2 - x_{i-1})^2$$

Global minimum is: $f(x^*) = 0$, at $x_i = 2^{-\frac{2^i - 2}{2^i}}$, for $i = 1, \ldots, d$.

The table below compares the speed of convergence of the tested algorithms on the Dixon-Price function with $d = 4$.

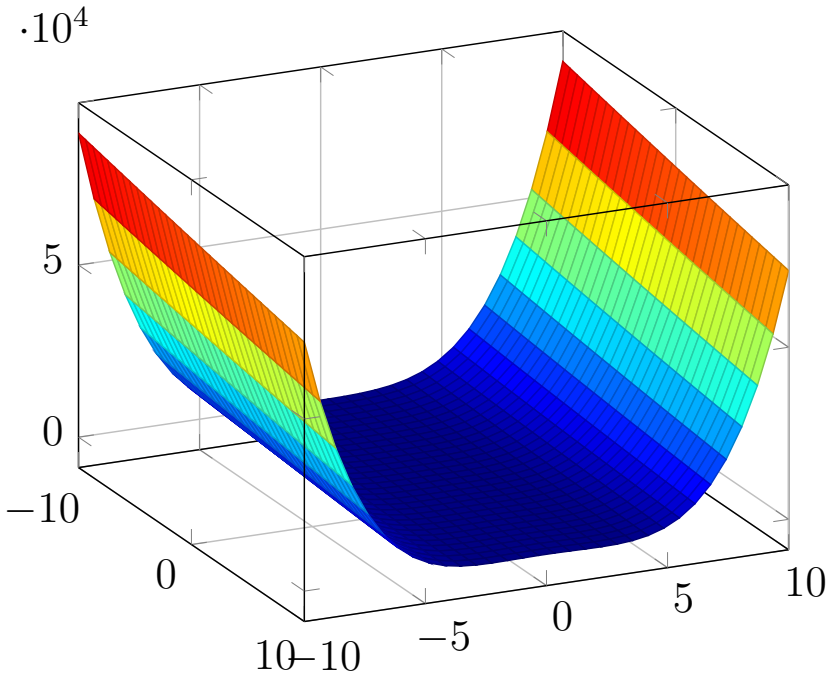


Figure 3.3 – Dixon-Price Function

x_0	0,0,0,0	1,1,1,1	-100,-100,-100,-100	2000,2000,2000,2000
Newton iterations	∞	6	∞	∞
BFGS iterations		11	111	∞
BFGS func evals	$\neq$	17	129	
BFGS gradient evals		17	129	
CG iterations		13		∞
CG func evals	$\neq$	29	$\neq$	
CG gradient evals		29		
NCG iterations		8		44
NCG func evals	$\neq$	9	$\neq$	56
NCG gradient evals		16		99
NCG hessian evals		8		44
PLH iterations	42	54	179	361
PLH Newton steps	14	14	34	35
PLH label evals	56	68	213	396

Below are some fragments from the output produced by the algorithm for the initial point $(0,0,0,0)$.

```
1. PLH triangulation level =  0
S = [   0.3000   0.8000  -0.2000  -0.2000  -0.2000  -0.2000 ]
    [   0.1000   0.6000   0.6000  -0.4000  -0.4000  -0.4000 ]
```

```
    [  -0.1000   0.4000   0.4000   0.4000  -0.6000  -0.6000 ]
    [  -0.3000   0.2000   0.2000   0.2000   0.2000  -0.8000 ]
x = [  -0.2000  -0.4000  -0.6000  -0.8000 ]

2. PLH triangulation level =  1
S = [   0.3000   0.8000  -0.2000  -0.2000  -0.2000   0.3000 ]
    [   0.1000   0.6000   0.6000  -0.4000  -0.4000   0.1000 ]
    [  -0.1000   0.4000   0.4000   0.4000  -0.6000  -0.1000 ]
    [  -0.3000   0.2000   0.2000   0.2000   0.2000   0.2000 ]
x = [   0.3000   0.1000  -0.1000   0.2000 ]
Attempting Newton steps.
x = [   0.0402   0.0047   0.1349   0.1654 ]
x = [   1.7141   0.1879  -0.8956   0.2389 ]
Newton steps failed.

3. PLH triangulation level =  1
S = [   0.3000   0.8000  -0.2000  -0.2000   0.3000   0.3000 ]
    [   0.1000   0.6000   0.6000  -0.4000   0.1000   0.1000 ]
    [  -0.1000   0.4000   0.4000   0.4000   0.4000  -0.1000 ]
    [  -0.3000   0.2000   0.2000   0.2000   0.2000   0.2000 ]
x = [   0.3000   0.1000   0.4000   0.2000 ]

4. PLH triangulation level =  1
S = [   0.3000   0.8000  -0.2000   0.3000   0.3000   0.3000 ]
    [   0.1000   0.6000   0.6000   0.6000   0.1000   0.1000 ]
    [  -0.1000   0.4000   0.4000   0.4000   0.4000  -0.1000 ]
    [  -0.3000   0.2000   0.2000   0.2000   0.2000   0.2000 ]
x = [   0.3000   0.6000   0.4000   0.2000 ]

5. PLH triangulation level =  1
S = [   0.3000   0.8000   0.8000   0.3000   0.3000   0.3000 ]
    [   0.1000   0.6000   0.6000   0.6000   0.1000   0.1000 ]
    [  -0.1000   0.4000   0.4000   0.4000   0.4000  -0.1000 ]
    [  -0.3000   0.2000   0.2000   0.2000   0.2000   0.2000 ]
x = [   0.8000   0.6000   0.4000   0.2000 ]

...

36. PLH triangulation level =  4
S = [   0.9875   1.0188   0.9875   1.0500   0.9875   0.9875 ]
    [   0.7250   0.6937   0.7250   0.7250   0.7250   0.6625 ]
    [   0.5875   0.6188   0.6500   0.6500   0.5875   0.5875 ]
    [   0.5125   0.5437   0.5750   0.5750   0.5750   0.5125 ]
x = [   1.0188   0.6937   0.6188   0.5437 ]

37. PLH triangulation level =  5
```

```
S = [   0.9875   1.0188   0.9875   0.9875   0.9875   0.9875 ]
    [   0.7250   0.6937   0.7250   0.6937   0.7250   0.6625 ]
    [   0.5875   0.6188   0.6500   0.6188   0.5875   0.5875 ]
    [   0.5125   0.5437   0.5750   0.5437   0.5750   0.5125 ]
x = [   0.9875   0.6937   0.6188   0.5437 ]
Attempting Newton steps.
x = [   0.9910   0.7017   0.5912   0.5413 ]
x = [   1.0051   0.7102   0.5964   0.5470 ]
Newton steps failed.

38. PLH triangulation level =  5
S = [   0.9875   1.0188   0.9875   0.9875   0.9875   0.9875 ]
    [   0.7250   0.6937   0.6937   0.6937   0.7250   0.6625 ]
    [   0.5875   0.6188   0.5875   0.6188   0.5875   0.5875 ]
    [   0.5125   0.5437   0.5437   0.5437   0.5750   0.5125 ]
x = [   0.9875   0.6937   0.5875   0.5437 ]

39. PLH triangulation level =  5
S = [   0.9875   1.0188   0.9875   1.0188   0.9875   0.9875 ]
    [   0.7250   0.6937   0.6937   0.6937   0.7250   0.6625 ]
    [   0.5875   0.6188   0.5875   0.5875   0.5875   0.5875 ]
    [   0.5125   0.5437   0.5437   0.5437   0.5750   0.5125 ]
x = [   1.0188   0.6937   0.5875   0.5437 ]

40. PLH triangulation level =  5
S = [   0.9875   1.0188   0.9875   1.0188   0.9875   0.9875 ]
    [   0.7250   0.6937   0.6937   0.6937   0.7250   0.7250 ]
    [   0.5875   0.6188   0.5875   0.5875   0.5875   0.5875 ]
    [   0.5125   0.5437   0.5437   0.5437   0.5750   0.5437 ]
x = [   0.9875   0.7250   0.5875   0.5437 ]

41. PLH triangulation level =  5
S = [   1.0031   1.0188   0.9875   1.0188   0.9875   0.9875 ]
    [   0.7094   0.6937   0.6937   0.6937   0.7250   0.7250 ]
    [   0.6031   0.6188   0.5875   0.5875   0.5875   0.5875 ]
    [   0.5594   0.5437   0.5437   0.5437   0.5750   0.5437 ]
x = [   1.0031   0.7094   0.6031   0.5594 ]

42. PLH triangulation level =  6
S = [   1.0031   1.0188   0.9875   1.0188   1.0031   0.9875 ]
    [   0.7094   0.6937   0.6937   0.6937   0.7094   0.7250 ]
    [   0.6031   0.6188   0.5875   0.5875   0.6031   0.5875 ]
    [   0.5594   0.5437   0.5437   0.5437   0.5437   0.5437 ]
x = [   1.0031   0.7094   0.6031   0.5437 ]
Attempting Newton steps.
x = [   0.9973   0.7055   0.5935   0.5447 ]
```

```
x = [   0.9998   0.7070   0.5945   0.5452 ]
x = [   1.0000   0.7071   0.5946   0.5452 ]
x = [   1.0000   0.7071   0.5946   0.5453 ]
Newton steps succedeed.

Approximate solution found.
x = [   1.0000   0.7071   0.5946   0.5453 ]
f(x) =    0.0000
PLH iterations:  42
Newton steps:    14
Label evals:     56
```

3.2.4 Easom function

$$F(x,y) = -\cos(x)\cdot\cos(y)\cdot e^{-(x-\pi)^2-(y-\pi)^2},\ (x,y)\in\mathbb{R}^2.$$

The global minimum value of F is $F(\pi,\pi) = -1$ within $-100 \le x,y \le 100$. It has many local minima.

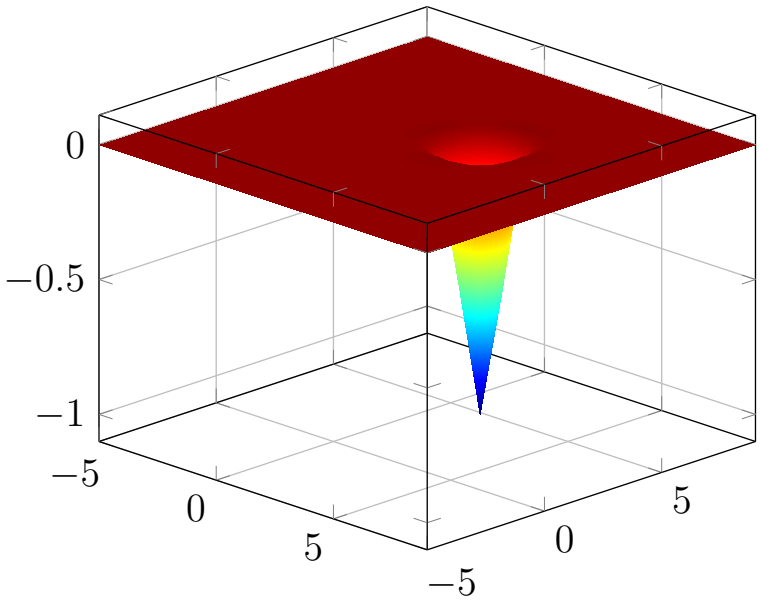


Figure 3.4 – Easom Function

x_0, y_0	0,0	1,1	2,2	3,3
Newton iterations	$\neq$	$\neq$	$\neq$	4
BFGS iterations	∞	$\neq$	3	3
BFGS func evals			9	5
BFGS gradient evals			9	5
CG iterations	∞	$\neq$	1	2
CG func evals			18	5
CG gradient evals			6	5
NCG iterations	2	$\neq$	3	4
NCG func evals	14		9	5
NCG gradient evals	15		11	8
NCG hessian evals	2		3	4
PLH iterations	$\neq$	$\neq$	11	5
PLH Newton steps			6	6
PLH label evals			17	11

Below is the full output produced by the algorithm for the initial point $(2, 2)$.

```
1. PLH triangulation level =  0
S = [   2.1667   2.6667   1.6667   1.6667 ]
    [   1.8333   2.3333   2.3333   1.3333 ]
x = [   1.6667   1.3333 ]

2. PLH triangulation level =  1
S = [   2.1667   2.6667   1.6667   2.1667 ]
    [   1.8333   2.3333   2.3333   2.3333 ]
x = [   2.1667   2.3333 ]
Attempting Newton steps.
x = [   2.1688   1.9143 ]
x = [   2.1812   2.0221 ]
Newton steps failed.

3. PLH triangulation level =  1
S = [   2.1667   2.6667   1.6667   2.1667 ]
    [   2.8333   2.3333   2.3333   2.3333 ]
x = [   2.1667   2.8333 ]

4. PLH triangulation level =  1
S = [   2.1667   2.6667   2.6667   2.1667 ]
    [   2.8333   2.3333   2.3333   2.3333 ]
x = [   2.6667   2.3333 ]

5. PLH triangulation level =  1
S = [   2.1667   2.6667   2.6667   2.6667 ]
```

```
    [   2.8333   2.3333   2.3333   2.8333 ]
x = [   2.6667   2.8333 ]

6. PLH triangulation level =  1
S = [   2.1667   2.6667   2.6667   2.6667 ]
    [   2.8333   2.3333   3.3333   2.8333 ]
x = [   2.6667   3.3333 ]

7. PLH triangulation level =  1
S = [   3.1667   2.6667   2.6667   2.6667 ]
    [   2.8333   2.3333   3.3333   2.8333 ]
x = [   3.1667   2.8333 ]

8. PLH triangulation level =  1
S = [   3.1667   2.6667   2.6667   2.6667 ]
    [   2.8333   3.3333   3.3333   2.8333 ]
x = [   2.6667   3.3333 ]

9. PLH triangulation level =  1
S = [   3.1667   2.6667   2.6667   3.1667 ]
    [   2.8333   3.3333   3.3333   3.3333 ]
x = [   3.1667   3.3333 ]

10. PLH triangulation level =  1
S = [   3.1667   2.6667   2.9167   3.1667 ]
    [   2.8333   3.3333   3.0833   3.3333 ]
x = [   2.9167   3.0833 ]

11. PLH triangulation level =  2
S = [   3.1667   3.1667   2.9167   3.1667 ]
    [   2.8333   3.0833   3.0833   3.3333 ]
x = [   3.1667   3.0833 ]
Attempting Newton steps.
x = [   3.1419   3.1330 ]
x = [   3.1415   3.1425 ]
x = [   3.1416   3.1415 ]
x = [   3.1416   3.1416 ]
Newton steps succedeed.

Approximate solution found.
x = [   3.1416   3.1416 ]
f(x) =   -1.0000
PLH iterations:  11
Newton steps:    6
Label evals:     17
```

3.2.5 Rosenbrock function

$$F(x,y) = (1-x)^2 + 100(y-x^2)^2,\ (x,y) \in \mathbb{R}^2.$$

The Rosenbrock function, also known as banana function, is unimodal, and the global minimum value, $F(1,1) = 0$, lies in a narrow, parabolic valley. Usually it is evaluated within the rectangle $-5 \leq x, y \leq 5$.

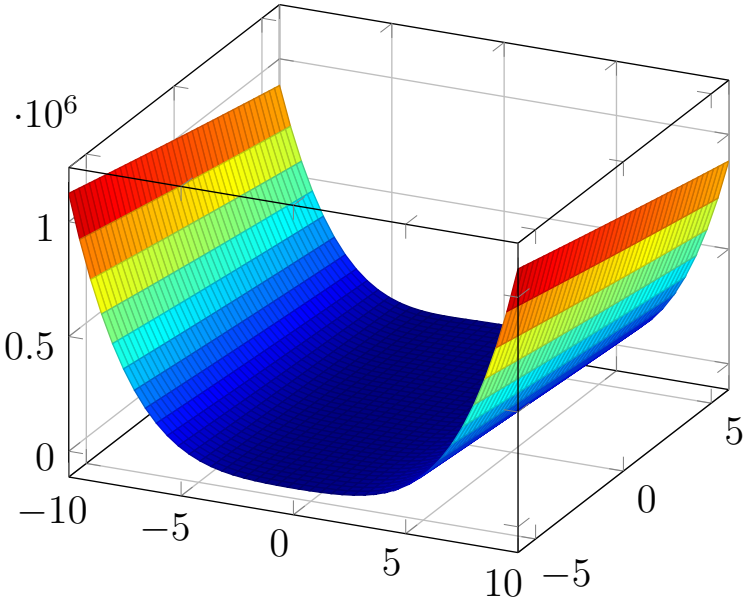


Figure 3.5 – Rosenbrock Function

x_1, x_2	0,0	2,2	3,3	10,10	10000,10000
Newton iterations	4	7	8	9	∞
BFGS iterations	21	20	38	49	$\neq$
BFGS func evals	26	30	57	69	
BFGS gradient evals	26	30	57	69	
CG iterations	17	$\neq$	31	$\neq$	$\neq$
CG func evals	33		78		
CG gradient evals	33		78		
NCG iterations	33	24	31	51	$\neq$
NCG func evals	53	37	44	83	
NCG gradient evals	85	60	74	133	
NCG hessian evals	33	24	31	51	
PLH iterations	115	91	100	169	308
PLH Newton steps	59	45	45	45	72
PLH label evals	174	136	145	214	380

Below are some fragments from the output produced by the algorithm for the initial point $(10000, 10000)$.

```
1. PLH triangulation level =  0
S = [ 10116.6667 10466.6667  9766.6667  9766.6667 ]
    [  9883.3333 10233.3333 10233.3333  9533.3333 ]
x = [  9766.6667  9533.3333 ]

2. PLH triangulation level =  1
S = [ 10116.6667  9766.6667  9766.6667  9766.6667 ]
    [  9883.3333  9883.3333 10233.3333  9533.3333 ]
x = [  9766.6667  9883.3333 ]
Attempting Newton steps.
x = [  9766.6667 10000.0115 ]
x = [  9451.7532 659955189.9653 ]
Newton steps failed.

3. PLH triangulation level =  1
S = [  9416.6667  9766.6667  9766.6667  9766.6667 ]
    [  9883.3333  9883.3333 10233.3333  9533.3333 ]
x = [  9416.6667  9883.3333 ]

4. PLH triangulation level =  1
S = [  9416.6667  9066.6667  9766.6667  9766.6667 ]
    [  9883.3333 10233.3333 10233.3333  9533.3333 ]
x = [  9066.6667 10233.3333 ]

5. PLH triangulation level =  1
S = [  9416.6667  9066.6667  9066.6667  9766.6667 ]
    [  9883.3333 10233.3333  9533.3333  9533.3333 ]
x = [  9066.6667  9533.3333 ]

...

124. PLH triangulation level =  1
S = [   316.6667   -33.3333   -33.3333   -33.3333 ]
    [   783.3333   783.3333  1133.3333  1133.3333 ]
x = [   316.6667   783.3333 ]

125. PLH triangulation level =  1
S = [   316.6667   -33.3333   -33.3333   141.6667 ]
    [   783.3333   783.3333  1133.3333   958.3333 ]
x = [   141.6667   958.3333 ]

126. PLH triangulation level =  2
```

```
S = [   -33.3333   -33.3333   -33.3333   141.6667 ]
    [   958.3333   783.3333  1133.3333   958.3333 ]
x = [   -33.3333   958.3333 ]
Attempting Newton steps.
x = [   -33.3333  1111.1121 ]
x = [   -33.3333  1111.1115 ]
Newton steps failed.

127. PLH triangulation level =  2
S = [   -33.3333    54.1667   -33.3333   141.6667 ]
    [   958.3333  1045.8333  1133.3333   958.3333 ]
x = [    54.1667  1045.8333 ]

128. PLH triangulation level =  3
S = [   -33.3333    54.1667   -33.3333   -33.3333 ]
    [   958.3333  1045.8333  1133.3333  1045.8333 ]
x = [   -33.3333  1045.8333 ]
Attempting Newton steps.
x = [   -33.3332  1111.1130 ]
x = [   -33.3332  1111.1069 ]
Newton steps failed.

129. PLH triangulation level =  3
S = [    10.4167    54.1667   -33.3333   -33.3333 ]
    [  1089.5833  1045.8333  1133.3333  1045.8333 ]
x = [    10.4167  1089.5833 ]

130. PLH triangulation level =  4
S = [    10.4167    54.1667    10.4167   -33.3333 ]
    [  1089.5833  1045.8333  1045.8333  1045.8333 ]
x = [    10.4167  1045.8333 ]
Attempting Newton steps.
x = [   -25.7563  1050.8195 ]
x = [   -65.4107  1023.7008 ]
Newton steps failed.

...

302. PLH triangulation level =  17
S = [     0.9959     0.9959     0.9852     0.9959 ]
    [     0.9923     0.9816     0.9923     1.0030 ]
x = [     0.9959     1.0030 ]

303. PLH triangulation level =  16
S = [     0.9745     0.9959     0.9852     0.9959 ]
    [     1.0030     0.9816     0.9923     1.0030 ]
```

```
x = [     0.9745     1.0030 ]

304. PLH triangulation level =  16
S = [     0.9745     0.9959     1.0173     0.9959 ]
    [     1.0030     0.9816     1.0030     1.0030 ]
x = [     1.0173     1.0030 ]

305. PLH triangulation level =  15
S = [     1.0066     0.9959     1.0173     0.9959 ]
    [     0.9923     0.9816     1.0030     1.0030 ]
x = [     1.0066     0.9923 ]

306. PLH triangulation level =  16
S = [     1.0066     0.9959     0.9959     0.9959 ]
    [     0.9923     0.9816     0.9923     1.0030 ]
x = [     0.9959     0.9923 ]
Attempting Newton steps.
x = [     0.9967     0.9934 ]
x = [     0.9973     0.9947 ]
Newton steps failed.

307. PLH triangulation level =  16
S = [     1.0066     1.0012     0.9959     0.9959 ]
    [     0.9923     0.9977     0.9923     1.0030 ]
x = [     1.0012     0.9977 ]

308. PLH triangulation level =  17
S = [     0.9959     1.0012     0.9959     0.9959 ]
    [     0.9977     0.9977     0.9923     1.0030 ]
x = [     0.9959     0.9977 ]
Attempting Newton steps.
x = [     0.9980     0.9960 ]
x = [     0.9990     0.9980 ]
x = [     0.9995     0.9990 ]
x = [     0.9998     0.9995 ]
x = [     0.9999     0.9998 ]
x = [     0.9999     0.9999 ]
Newton steps succedeed.

Approximate solution found.
x = [     0.9999     0.9999 ]
f(x) =      0.0000
PLH iterations:  308
Newton steps:    72
Label evals:     380
```

3.2.6 Schwefel function

$$F(x_1,\ldots,x_n)=\sum_{i=1}^{n}-x_i\cdot\sin\left(\sqrt{|x_i|}\right)$$

We will test the optimization algorithms for $n=10$, in the hypercube $x_i\in[-10,10]$, $i=1..n$. The function has a single minimum in this input domain which is:

$$F\left(\frac{5\pi}{3},\ldots,\frac{5\pi}{3}\right)=\frac{-7.89\cdot n}{2}.$$

Note also that the Schwefel function F is not differentiable on the set $\{(0,a_2,\ldots,a_n):a_2,\ldots,a_n\in\mathbb{R}\}\cup\{(a_1,0,a_3,\ldots,a_n):a_1,a_3,\cdots\in\mathbb{R}\}\cup\ldots$.

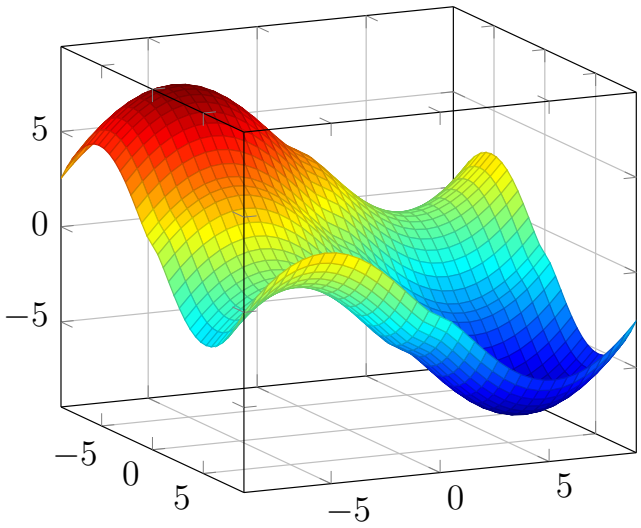


Figure 3.6 – Schwefel Function for $n=2$

We shall use the following starting points for comparing the algorithms:

$$c_1=(5,5,5,5,5,5,5,5,5,5),c_2=(5,5,5,5,5,5,5,5,5,1),c_3=(2,2,2,2,2,2,2,2,2,2),$$

$$c_4=(2,2,2,2,2,2,2,2,2,1),c_5=(1,1,1,1,1,1,1,1,1,1),c_6=(1,1,1,1,1,1,1,1,1,0),$$

$$c_7=(0,0,0,0,0,0,0,0,0,0).$$

x_0	c_1	c_2	c_3	c_4	c_5	c_6	c_7
Newton iterations	2	∞	∞	∞	4	∞	∞
BFGS iterations	3	$\neq$	5	$\neq$	4	$\neq$	∞
BFGS func evals	4		6		6		
BFGS gradient evals	4		6		6		
CG iterations	1	$\neq$	$\neq$	$\neq$	2	$\neq$	∞
CG func evals	15				6		
CG gradient evals	4				6		
NCG iterations	3	$\neq$	3	$\neq$	4	∞	∞
NCG func evals	4		5		5		
NCG gradient evals	6		7		8		
NCG hessian evals	3		3		4		
PLH iterations	14	20	43	44	45	50	45
PLH Newton steps	14	9	8	9	8	8	13
PLH label evals	28	29	51	53	53	58	58

Below are some fragments from the output produced by the algorithm for the initial point $(0,0,0,0,0,0,0,0,0,0)$. In order to be able to fit the data on the page the maximal accepted error used for the data below $1e-2$.

```
1. PLH triangulation level =  0
S =
[  4.09  9.09 -0.91 -0.91 -0.91 -0.91 -0.91 -0.91 -0.91 -0.91 -0.91 -0.91 ]
[  3.18  8.18  8.18 -1.82 -1.82 -1.82 -1.82 -1.82 -1.82 -1.82 -1.82 -1.82 ]
[  2.27  7.27  7.27  7.27 -2.73 -2.73 -2.73 -2.73 -2.73 -2.73 -2.73 -2.73 ]
[  1.36  6.36  6.36  6.36  6.36 -3.64 -3.64 -3.64 -3.64 -3.64 -3.64 -3.64 ]
[  0.45  5.45  5.45  5.45  5.45  5.45 -4.55 -4.55 -4.55 -4.55 -4.55 -4.55 ]
[ -0.45  4.55  4.55  4.55  4.55  4.55  4.55 -5.45 -5.45 -5.45 -5.45 -5.45 ]
[ -1.36  3.64  3.64  3.64  3.64  3.64  3.64  3.64 -6.36 -6.36 -6.36 -6.36 ]
[ -2.27  2.73  2.73  2.73  2.73  2.73  2.73  2.73  2.73 -7.27 -7.27 -7.27 ]
[ -3.18  1.82  1.82  1.82  1.82  1.82  1.82  1.82  1.82  1.82 -8.18 -8.18 ]
[ -4.09  0.91  0.91  0.91  0.91  0.91  0.91  0.91  0.91  0.91  0.91 -9.09 ]
x = [ -0.91 -1.82 -2.73 -3.64 -4.55 -5.45 -6.36 -7.27 -8.18 -9.09 ]

2. PLH triangulation level =  1
S =
[  4.09  9.09 -0.91 -0.91 -0.91 -0.91 -0.91 -0.91 -0.91 -0.91 -0.91  4.09 ]
[  3.18  8.18  8.18 -1.82 -1.82 -1.82 -1.82 -1.82 -1.82 -1.82 -1.82  3.18 ]
[  2.27  7.27  7.27  7.27 -2.73 -2.73 -2.73 -2.73 -2.73 -2.73 -2.73  2.27 ]
[  1.36  6.36  6.36  6.36  6.36 -3.64 -3.64 -3.64 -3.64 -3.64 -3.64  1.36 ]
[  0.45  5.45  5.45  5.45  5.45  5.45 -4.55 -4.55 -4.55 -4.55 -4.55  0.45 ]
[ -0.45  4.55  4.55  4.55  4.55  4.55  4.55 -5.45 -5.45 -5.45 -5.45 -0.45 ]
[ -1.36  3.64  3.64  3.64  3.64  3.64  3.64  3.64 -6.36 -6.36 -6.36 -1.36 ]
```

```
[ -2.27  2.73  2.73  2.73  2.73  2.73  2.73  2.73  2.73 -7.27 -7.27 -2.27 ]
[ -3.18  1.82  1.82  1.82  1.82  1.82  1.82  1.82  1.82  1.82 -8.18 -3.18 ]
[ -4.09  0.91  0.91  0.91  0.91  0.91  0.91  0.91  0.91  0.91  0.91  0.91 ]
x = [  4.09  3.18  2.27  1.36  0.45 -0.45 -1.36 -2.27 -3.18  0.91 ]
Attempting Newton steps.
x = [  3.02  2.64  2.21  1.67  0.89  0.29 -0.14 -0.82 -1.59 -2.42 ]
x = [  0.50  0.65  0.81  0.95  0.99  0.71  0.56  1.16  1.33  1.29 ]
Newton steps failed.

3. PLH triangulation level =  1
S =
[  4.09  9.09 -0.91 -0.91 -0.91 -0.91 -0.91 -0.91 -0.91 -0.91 -0.91  4.09 ]
[  3.18  8.18  8.18 -1.82 -1.82 -1.82 -1.82 -1.82 -1.82 -1.82 -1.82  3.18 ]
[  2.27  7.27  7.27  7.27 -2.73 -2.73 -2.73 -2.73 -2.73 -2.73 -2.73  2.27 ]
[  1.36  6.36  6.36  6.36  6.36 -3.64 -3.64 -3.64 -3.64 -3.64 -3.64  1.36 ]
[  0.45  5.45  5.45  5.45  5.45  5.45 -4.55 -4.55 -4.55 -4.55 -4.55  0.45 ]
[ -0.45  4.55  4.55  4.55  4.55  4.55  4.55 -5.45 -5.45 -5.45 -5.45 -0.45 ]
[ -1.36  3.64  3.64  3.64  3.64  3.64  3.64  3.64 -6.36 -6.36 -6.36 -1.36 ]
[ -2.27  2.73  2.73  2.73  2.73  2.73  2.73  2.73  2.73 -7.27 -7.27 -2.27 ]
[ -3.18  1.82  1.82  1.82  1.82  1.82  1.82  1.82  1.82  1.82 -8.18 -3.18 ]
[  5.91  0.91  0.91  0.91  0.91  0.91  0.91  0.91  0.91  0.91  0.91  0.91 ]
x = [  4.09  3.18  2.27  1.36  0.45 -0.45 -1.36 -2.27 -3.18  5.91 ]

...

44. PLH triangulation level =  1
S =
[  4.09  6.59  9.09  9.09  4.09  4.09  4.09  4.09  4.09  4.09  4.09  4.09 ]
[  3.18  5.68  8.18  8.18  8.18  8.18  3.18  3.18  3.18  3.18  3.18  3.18 ]
[  2.27  4.77  7.27  7.27  7.27  7.27  7.27  7.27  2.27  2.27  2.27  2.27 ]
[  1.36  3.86  6.36  6.36  6.36  6.36  6.36  6.36  6.36  6.36  1.36  1.36 ]
[  0.45  2.95  5.45  5.45  5.45  5.45  5.45  5.45  5.45  5.45  5.45  5.45 ]
[  9.55  7.05  4.55  4.55  4.55  4.55  4.55  4.55  4.55  4.55  4.55  9.55 ]
[  8.64  6.14  3.64  3.64  3.64  3.64  3.64  3.64  3.64  8.64  8.64  8.64 ]
[  7.73  5.23  2.73  2.73  2.73  2.73  2.73  7.73  7.73  7.73  7.73  7.73 ]
[  6.82  4.32  1.82  1.82  1.82  6.82  6.82  6.82  6.82  6.82  6.82  6.82 ]
[  5.91  3.41  0.91  5.91  5.91  5.91  5.91  5.91  5.91  5.91  5.91  5.91 ]
x = [  6.59  5.68  4.77  3.86  2.95  7.05  6.14  5.23  4.32  3.41 ]

45. PLH triangulation level =  2
S =
[  6.59  6.59  9.09  9.09  4.09  4.09  4.09  4.09  4.09  4.09  4.09  4.09 ]
[  5.68  5.68  8.18  8.18  8.18  8.18  3.18  3.18  3.18  3.18  3.18  3.18 ]
[  4.77  4.77  7.27  7.27  7.27  7.27  7.27  7.27  2.27  2.27  2.27  2.27 ]
[  3.86  3.86  6.36  6.36  6.36  6.36  6.36  6.36  6.36  6.36  1.36  1.36 ]
[  5.45  2.95  5.45  5.45  5.45  5.45  5.45  5.45  5.45  5.45  5.45  5.45 ]
```

```
[  7.05  7.05  4.55  4.55  4.55  4.55  4.55  4.55  4.55  4.55  4.55  9.55 ]
[  6.14  6.14  3.64  3.64  3.64  3.64  3.64  3.64  3.64  8.64  8.64  8.64 ]
[  5.23  5.23  2.73  2.73  2.73  2.73  2.73  7.73  7.73  7.73  7.73  7.73 ]
[  4.32  4.32  1.82  1.82  1.82  6.82  6.82  6.82  6.82  6.82  6.82  6.82 ]
[  3.41  3.41  0.91  5.91  5.91  5.91  5.91  5.91  5.91  5.91  5.91  5.91 ]
x = [  6.59  5.68  4.77  3.86  5.45  7.05  6.14  5.23  4.32  3.41 ]
Attempting Newton steps.
x = [  5.32  5.27  5.14  5.02  5.23  5.31  5.31  5.21  5.07  5.02 ]
x = [  5.23  5.24  5.25  5.29  5.24  5.23  5.23  5.24  5.26  5.34 ]
x = [  5.24  5.24  5.24  5.22  5.24  5.24  5.24  5.24  5.24  5.19 ]
x = [  5.24  5.24  5.24  5.24  5.24  5.24  5.24  5.24  5.24  5.26 ]
x = [  5.24  5.24  5.24  5.24  5.24  5.24  5.24  5.24  5.24  5.23 ]
Newton steps succedeed.

Approximate solution found.
x = [  5.24  5.24  5.24  5.24  5.24  5.24  5.24  5.24  5.24  5.23 ]
f(x) =  -39.45
PLH iterations:  45
Newton steps:    7
Label evals:     52
```

In the next table we show some of the results obtained for the bi-dimensional Schwefel function ($n = 2$).

x_1, x_2	5,5	3,3	1,1	0.1,0.1	0.0001,0.0001	0,0
Newton iterations	2	4	4	15	908	∞
BFGS iterations	3	4	4	2	3	$\neq$
BFGS func evals	4	5	6	8	19	
BFGS gradient evals	4	5	6	8	18	
CG iterations	1	1	2	1	1	$\neq$
CG func evals	15	15	6	18	28	
CG gradient evals	4	3	6	6	15	
NCG iterations	33	24	31	51	5	$\neq$
NCG func evals	53	37	44	83	22	
NCG gradient evals	85	60	74	133	25	
NCG hessian evals	33	24	31	51	5	
PLH iterations	5	20	35	40	42	42
PLH Newton steps	3	3	3	3	3	3
PLH label evals	8	23	38	43	45	45

Below is the full output produced by the algorithm for the initial point $(5, 5)$.

```
1. PLH triangulation level =  0
S = [ 5.1667 5.6667 4.6667 4.6667 ]
    [ 4.8333 5.3333 5.3333 4.3333 ]
x = [ 4.6667 4.3333 ]

2. PLH triangulation level =  1
S = [ 5.1667 5.6667 4.6667 5.1667 ]
    [ 4.8333 5.3333 5.3333 5.3333 ]
x = [ 5.1667 5.3333 ]
Attempting Newton steps.
x = [ 5.1314 4.9981 ]
x = [ 5.1366 5.0960 ]
Newton steps failed.

3. PLH triangulation level =  1
S = [ 5.1667 5.6667 5.6667 5.1667 ]
    [ 4.8333 5.3333 5.3333 5.3333 ]
x = [ 5.6667 5.3333 ]

4. PLH triangulation level =  1
S = [ 5.1667 5.4167 5.6667 5.1667 ]
    [ 4.8333 5.0833 5.3333 5.3333 ]
x = [ 5.4167 5.0833 ]

5. PLH triangulation level =  2
S = [ 5.1667 5.4167 5.1667 5.1667 ]
    [ 4.8333 5.0833 5.0833 5.3333 ]
x = [ 5.1667 5.0833 ]
Attempting Newton steps.
x = [ 5.2393 5.2392 ]
Newton steps succedeed.

Approximate solution found.
x = [ 5.2393 5.2392 ]
f(x) =  -7.8906
PLH iterations:  5
Newton steps:    3
Label evals:     8
```

CONCLUSIONS

The new implementation of the iterative fixed point algorithms based on the Banach fixed point theorem (the FIXPOINT software package), described in 1.2.2 proved itself as a very useful practical tool, due to the new features, which at this time are not present in other implementations, like: the usage of a mixed error test for testing the convergence of the sequence of successive approximations, the additional check for cycle detection and the built in comparison function. For example, some early versions of FIXPOINT were used for the most of the results reported in Chapter 9 of Berinde (2007) - Error analysis of fixed point iteration procedures, and also for the numerical experiments in the comparative study of the rate of convergence of fixed point iteration procedures, that were reported in Berinde and Păcurar (2007) and later used as a starting point for inferring and proving some theoretical results mentioned in the Introduction chapter. The complementary web-based application provides an easy way to use these FIXPOINT software package and some additional features like the interactive visualisations, cobweb plots for the Picard iteration and a new intuitive visualisation for the Krasnoselskii iteration, which may be useful also as a didactic tool. As possible future work directions related to the new implementations of the iterative methods described in the first chapter we mention: improving the interactive visualisations with animations and other features, implementing other fixed point algorithms see for example Berinde (2007), Agarwal et al. (2009) and implementing some acceleration techniques as described for example in Bumbariu (2013).

The main generic advantages of the PL homotopy methods described and in the second chapter is that they don't require smoothness of the underlying map. Also another important feature of these methods is that they can be applied when no a priori knowledge regarding the solutions of the system to be solved is available. When applying this algorithm to unconstrained optimization problems, represented here by some classic objective test functions (Bukin function, Easom function, Rosenbrock function, Schwefel function etc.), it appears the PL homotopy method is more robust than some of the most important and used iterative methods in optimization

(Newton's method, Broyden-Fletcher-Goldfarb-Shanno algorithm, conjugate gradient method, nonlinear conjugate gradient method), see the results from 3.2. Due to these results, some possible work directions for the future are to continue the study by to extend the numerical tests to even higher dimensions, considering other important optimization test functions, see Floudas et al. (1999), Hedar (2014), Moré et al. (1981), Pohlheim (2006) and considering other different types of problems see for example Herings and Peeters (2010) or Pop (2008, 2009). As other possible research directions we mention studying feasibility of parallelizing some parts of the implemented PL homotopy algorithm; using some acceleration techniques when attempting Newton steps; the implementation other versions of the PL homotopy algorithms like van der Laan and Talman (1981) or Wright (1981).

Considering that one of the initial objectives was to create easy to use implementations of some fixed point algorithms, the very popular Javascript language was finally used, after some experimentation with the C++ and C# languages. As a direct consequence of using the Javascript language, it was very natural to create a complementary web-based application, which has great advantages for users in the form of: easy accessibility and usability, seamless updates and even enhanced collaboration. Considering the latest advancements in the programming tools for Javascript language, these new implementations also offers the unique possibility to run the same implementation of an algorithm on the client machine, using a smart client application or on servers using for example Node.js (2013). Also the Javascript language has a straightforward syntax, it doesn't require a compiler and the support for developers is very good in the form of documentation, samples and tools, so it is expected that the new Javascript implementations could be used very easily, even by other persons, for different applications.

BIBLIOGRAPHY

Agarwal, R. P., O'Regan, D., and Sahu, D. (2009). *Fixed point theory for Lipschitzian-type mappings with applications*. Springer. (Cited on page 117.)

Akbulut, S. and Ozdemir, M. (2012). Picard iteration converges faster than Noor iteration for a class of quasi-contractive operators. *Chiang Mai Journal of Science*, 39(4):688–692. (Cited on page 92.)

Aliprantis, C. D. and Border, K. C. (2006). *Infinite dimensional analysis: a hitchhiker's guide*. Springer. (Cited on page 14.)

Allgower, E. L. and Georg, K. (1978). Triangulations by reflections with applications to approximation. *Numerische Methoden der Approximationstheorie*, 42:10–32. (Cited on pages 50 and 51.)

Allgower, E. L. and Georg, K. (1993). Continuation and path following. *Acta numerica*, 2(1):1–64. (Cited on page 95.)

Allgower, E. L. and Georg, K. (2000). Piecewise linear methods for nonlinear equations and optimization. *Journal of Computational and Applied Mathematics*, 124(1):245–261. (Cited on page 95.)

Allgower, E. L. and Georg, K. (2003). *Introduction to numerical continuation methods*, volume 45. SIAM. (Cited on pages 48, 49, 74, 76, 77, and 96.)

Allgower, E. L. and Jeppson, M. M. (1973). The approximation of solutions of nonlinear elliptic boundary value problems having several solutions. In *Numerische, insbesondere approximationstheoretische Behandlung von Funktionalgleichungen*, pages 1–20. Springer. (Cited on page 95.)

Alotaibi, A., Kumar, V., and Hussain, N. (2013). Convergence comparison and stability of Jungck-Kirk-type algorithms for common fixed point problems. *Fixed Point Theory and Applications*, 2013(1):1–30. (Cited on page 93.)

Babu, G. and Prasad, K. V. (2006). Mann iteration converges faster than Ishikawa iteration for the class of Zamfirescu operators. *Fixed Point Theory and Applications*, 2006(1):6. (Cited on pages 12 and 93.)

Babu, G. and Prasad, K. V. (2007). Comparison of fastness of the convergence among Krasnoselskij, Mann, and Ishikawa iterations in arbitrary real Banach spaces. *Fixed Point Theory and Applications*, 2006:12. (Cited on pages 12 and 93.)

Banach, S. (1922). Sur les opérations dans les ensembles abstraits et leur application aux équations intégrales. *Fund. Math*, 3(1):133–181. (Cited on page 17.)

Berinde, V. (2004a). Approximating fixed points of weak contractions using the Picard iteration. In *Nonlinear Analysis Forum*, volume 9, pages 43–54. (Cited on pages 10, 18, and 92.)

Berinde, V. (2004b). Picard iteration converges faster than Mann iteration for a class of quasi-contractive operators. *Fixed Point Theory and Applications*, 2004(2):97–105. (Cited on page 12.)

Berinde, V. (2007). *Iterative approximation of fixed points*. Springer Heidelberg, 2nd edition. (Cited on pages 12, 14, 18, 29, 30, 31, 36, 37, 87, 92, 95, and 117.)

Berinde, V. and Berinde, M. (2005). The fastest Krasnoselskij iteration for approximating fixed points of strictly pseudo-contractive mappings. *Carpathian J. Math*, 21(1-2):13–20. (Cited on pages 12 and 92.)

Berinde, V. and Păcurar, M. (2007). Empirical study of the rate of convergence of some fixed point iterative methods. *Proc. Appl. Math. Mech.*, 7(1):2030015–2030016. (Cited on pages 12, 92, and 117.)

Border, K. C. (1989). *Fixed point theorems with applications to economics and game theory*. Cambridge University Press. (Cited on pages 52 and 64.)

Bozantan, A. (2010). An implementation of the piecewise-linear homotopy algorithm for the computation of fixed points. *Creative Mathematics and Informatics*, 19(2):140–148. (Cited on pages 10 and 74.)

Bozantan, A. (2014a). Comparative study of fixed point software (poster). In *5th Minisymposium on Fixed Point Theory and Applications*. The Tenth International Conference on Applied Mathematics (ICAM 10). (Cited on page 20.)

Bozantan, A. (2014b). Usage of the FIXPOINT software package in practical applications (submitted). (Cited on page 85.)

Bozantan, A. and Berinde, V. (2013). Applications of the PL homotopy algorithm for the computation of fixed points to unconstrained optimization problems. *Creative Mathematics and Informatics*, 22(1):41–46. (Cited on page 74.)

Bozantan, A. and Berinde, V. (2014a). About the implementation and applications of the FIXPOINT software minipackage. *Creative Mathematics and Informatics*, 23(1):41–50. (Cited on page 20.)

Bozantan, A. and Berinde, V. (2014b). A numerical study on the robustness and efficiency of the PL homotopy algorithm for solving unconstrained optimization problems. *Miskolc Math. Notes (accepted)*. (Cited on pages 85 and 95.)

Bozantan, A., Pop, N., and Berinde, V. (2014). Using a new implementation of the PL homotopy algorithm for elastic contact problems with Coulomb friction (in preparation). (Cited on page 96.)

Brouwer, L. E. J. (1911). Über Abbildung von Mannigfaltigkeiten. *Mathematische Annalen*, 71(1):97–115. (Cited on page 53.)

Bumbariu, O. (2013). *Acceleration Techniques for Approximating Fixed Points*. PhD thesis, Technical University of Cluj Napoca, North University Center, Baia Mare. (Cited on page 117.)

Cătinaş, E. (2001). Inexact perturbed Newton methods and applications to a class of Krylov solvers. *Journal of Optimization Theory and Applications*, 108(3):543–570. (Cited on page 94.)

Cătinaş, E. (2002). On the superlinear convergence of the successive approximations method. *Journal of optimization theory and applications*, 113(3):473–485. (Cited on page 94.)

Cohen, D. I. (1967). On the Sperner lemma. *Journal of Combinatorial Theory*, 2(4):585–587. (Cited on pages 57 and 61.)

Coxeter, H. M. (1934). Discrete groups generated by reflections. *The Annals of Mathematics*, 35(3):588–621. (Cited on page 48.)

Dang, C. (1991). The D_1-triangulation of R^n for simplicial algorithms for computing solutions of nonlinear equations. *Mathematics of Operations Research*, 16(1):148–161. (Cited on page 51.)

Dang, C. (1995). Triangulations and simplicial methods. *Lecture Notes in Economics and Mathematical Systems*, 421. (Cited on page 44.)

Dang, C. and Talman, D. (1990). The D_1-triangulation in simplicial variable dimension algorithms on the unit simplex for computing fixed points. Technical report, Tilburg-Center for Economic Research. (Cited on page 51.)

Debreu, G. (1959). *Theory of value: An axiomatic analysis of economic equilibrium*. John Wiley and Sons, New York. (Cited on page 61.)

Doup, T. M., van der Laan, G., and Talman, A. J. J. (1987). The $2^{n+1} - 2$-ray algorithm: A new simplicial algorithm to compute economic equilibria. *Mathematical Programming*, 39(3):241–252. (Cited on page 74.)

Duong, V. (2012). The comparison of the convergence speed between Picard, Mann, Ishikawa and two-step iterations in Banach spaces. *Acta Math. Vietnam*, 37(2):243–249. (Cited on pages 12 and 93.)

Eaves, C. B. (1971). On the basic theory of complementarity. *Mathematical Programming*, 1:68–75. (Cited on page 56.)

Eaves, C. B. (1972). Homotopies for computation of fixed points. *Mathematical Programming*, 3(1):1–22. (Cited on pages 12 and 74.)

Eaves, C. B. (1984). A course in triangulations for solving equations with deformations. *Lecture Notes in Economics and Mathematical Systems*, 234. (Cited on page 44.)

Eaves, C. B. and Saigal, R. (1972). Homotopies for computation of fixed points on unbounded regions. *Mathematical Programming*, 3(1):225–237. (Cited on pages 11, 12, 74, 77, and 95.)

Edgar, T. F. and Himmelblau, D. M. (1989). *Optimization of chemical processes*. McGraw-Hill. (Cited on page 94.)

Floudas, C. A., Gümüş, Z. H., Adjiman, C. S., Schweiger, C. A., Esposito, W. R., Klepeis, J. L., Pardalos, P. M., Meyer, C. A., and Harding, S. T. (1999). *Handbook of test problems in local and global optimization*. Springer. (Cited on pages 97 and 118.)

Freudenthal, H. (1942). Simplizialzerlegungen von beschränkter Flachheit. *The Annals of Mathematics*, 43(3):580–582. (Cited on page 48.)

Freund, R. M. and Todd, M. J. (1981). A constructive proof of Tucker's combinatorial lemma. *Journal of Combinatorial Theory, Series A*, 30(3):321–325. (Cited on page 74.)

Güler, O. (2010). *Foundations of optimization*, volume 258. Springer. (Cited on page 94.)

Hansen, T. (1968). *On the approximation of a competitive equilibrium*. PhD thesis, Yale. (Cited on pages 61 and 65.)

Hansen, T. and Scarf, H. E. (1969). On the applications of a recent combinatorial algorithm. *Cowles Foundation Discussion Paper*, (272). (Cited on page 95.)

Hedar, A.-R. (2014). Test functions for unconstrained optimization. http://www-optima.amp.i.kyoto-u.ac.jp/member/student/hedar/Hedar_files/TestGO.htm. (Cited on pages 97 and 118.)

Herings, P. J.-J. and Peeters, R. (2010). Homotopy methods to compute equilibria in game theory. *Economic Theory*, 42(1):119–156. (Cited on pages 95 and 118.)

Herings, P. J.-J., Talman, D., and Yang, Z. (1996). The computation of a continuum of constrained equilibria. *Mathematics of Operations Research*, 21(3):675–696. (Cited on page 95.)

Herings, P. J.-J. and van den Elzen, A. (2002). Computation of the Nash equilibrium selected by the tracing procedure in n-person games. *Games and Economic Behavior*, 38(1):89–117. (Cited on page 95.)

Huang, J. (2004). On the Sperner lemma and its applications. (Cited on page 57.)

Hussain, N., Chugh, R., Kumar, V., and Rafiq, A. (2012). On the rate of convergence of Kirk-type iterative schemes. *Journal of Applied Mathematics*, 2012. (Cited on pages 12 and 93.)

Hussain, N., Kumar, V., and Kutbi, M. A. (2013). On rate of convergence of Jungck-type iterative schemes. In *Abstract and Applied Analysis*, volume 2013. Hindawi Publishing Corporation. (Cited on pages 12 and 93.)

Hussain, N., Rafiq, A., Damjanović, B., and Lazović, R. (2011). On rate of convergence of various iterative schemes. *Fixed Point Theory and Applications*, 2011(1):1–6. (Cited on pages 12 and 93.)

Istratescu, V. I. (2002). *Fixed point theory an introduction*, volume 7. Springer. (Cited on pages 14 and 52.)

Kakutani, S. (1941). A generalization of Brouwer's fixed point theorem. *Duke mathematical journal*, 8(3):457–459. (Cited on page 55.)

Kang, S. M., Ciric, L. B., Rafiq, A., Ali, F., and Kwun, Y. C. (2013). Faster multistep iterations for the approximation of fixed points applied to Zamfirescu operators. In *Abstract and Applied Analysis*, volume 2013. Hindawi Publishing Corporation. (Cited on page 93.)

Karahan, I. and Ozdemir, M. (2013). A general iterative method for approximation of fixed points and their applications. *Advances in Fixed Point Theory*, 3(3):510–526. (Cited on page 93.)

Kelley, C. T. (1999). *Iterative methods for optimization*, volume 18. Siam. (Cited on page 94.)

Khan, A. R., Kumar, V., and Hussain, N. (2014). Analytical and numerical treatment of Jungck-type iterative schemes. *Applied Mathematics and Computation*, 231:521–535. (Cited on page 93.)

Knaster, B., Kuratowski, C., and Mazurkiewicz, S. (1929). Ein Beweis des Fixpunktsatzes für n-dimensionale Simplexe. *Fundamenta Mathematicae*, 14(1):132–137. (Cited on page 52.)

Krasnoselskii, M. A. (1955). Two remarks on the method of successive approximations. *Uspekhi Matematicheskikh Nauk*, 10(1):123–127. (Cited on page 36.)

Kuhn, H. W. (1960). Some combinatorial lemmas in topology. *IBM Journal of research and development*, 4(5):518–524. (Cited on pages 49 and 50.)

Kuhn, H. W. (1968). Simplicial approximation of fixed points. *Proceedings of the National Academy of Sciences of the United States of America*, 61(4):1238. (Cited on pages 12, 61, 63, 65, 68, and 74.)

Kuhn, H. W. (1974). A new proof of the fundamental theorem of algebra. In *Pivoting and Extension*, pages 148–158. Springer. (Cited on page 95.)

Kumar, V. (2013). Comments on convergence rates of Mann and Ishikawa iterative schemes for generalized contractive operators. *International Journal of Mathematical Analysis (Ruse)*. (Cited on pages 12 and 93.)

Lemke, C. E. (1965). Bimatrix equilibrium points and mathematical programming. *Management science*, 11(7):681–689. (Cited on pages 61 and 62.)

Lemke, C. E. and Howson, J. T. (1964). Equilibrium points of bimatrix games. *Journal of the Society for Industrial & Applied Mathematics*, 12(2):413–423. (Cited on pages 61 and 62.)

Li, T.-Y. (1997). Numerical solution of multivariate polynomial systems by homotopy continuation methods. *Acta numerica*, 6(1):399–436. (Cited on page 95.)

Ligurskỳ, T. (2012). Theoretical analysis of discrete contact problems with Coulomb friction. *Applications of Mathematics*, 57(3):263–295. (Cited on page 96.)

Mann, R. W. (1953). Mean value methods in iteration. *Proceedings of the American Mathematical Society*, 4(3):506–510. (Cited on page 37.)

Măruşter, S., Negru, V., and Mafteiu-Scai, L. O. (2012). Experimental study on parallel methods for solving systems of equations. In *14th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*, pages 103–107. (Cited on page 94.)

McLennan, A. (2012). *Advanced Fixd Point Theory for Economics*. (Cited on page 44.)

Merrill, O. H. (1972). *Applications and extensions of an algorithm that computes fixed points of certain upper semi-continuous point to set mappings*. PhD thesis, University of Michigan., Ann Arbor, Michigan, 48106, USA. (Cited on pages 74 and 95.)

Moré, J. J., Garbow, B. S., and Hillstrom, K. E. (1981). Testing unconstrained optimization software. *ACM Transactions on Mathematical Software (TOMS)*, 7(1):17–41. (Cited on pages 97 and 118.)

Murty, K. G. (1988). *Linear complementarity, linear and nonlinear programming*. Heldermann, Berlin. (Cited on pages 61, 64, and 95.)

Netlib (2013). Netlib repository. http://netlib.org/. (Cited on page 9.)

Node.js (2013). Node.js. http://nodejs.org. (Cited on page 118.)

Olaleru, J. O. (2007). A comparison of Picard and Mann iterations for quasi-contraction maps. *Fixed Point Theory*, 8(1):87–95. (Cited on pages 12 and 93.)

Olaleru, J. O. (2009). On the convergence rates of Picard, Mann and Ishikawa iterations of generalized contractive operators. *Studia Universitatis Babes-Bolyai, Mathematica*, 54(4). (Cited on pages 12 and 93.)

Păcurar, M. (2009a). Approximating common fixed points of Presic-Kannan type operators by a multi-step iterative method. *An. St. Univ. Ovidius Constanta*, 17:153–168. (Cited on page 12.)

Păcurar, M. (2009b). *Iterative Methods for Fixed Point Approximation*. Risoprint, Cluj-Napoca. (Cited on pages 12 and 85.)

Păcurar, M. (2010a). A multi-step iterative method for approximating common fixed points of Prešić-Rus type operators on metric spaces. *Studia Universitatis Babes-Bolyai, Mathematica*, 55(1). (Cited on page 12.)

Păcurar, M. (2010b). A multi-step iterative method for approximating fixed points of Presic-Kannan operators. *Acta Math. Univ. Comenianae*, 79(1):77–88. (Cited on page 12.)

Păcurar, M. (2011). Fixed points of almost Prešić operators by a k-step iterative method. *An. Ştiint,. Univ. Al. I. Cuza Iaşi - Serie Noua-Matematica*, pages 199–210. (Cited on page 12.)

Păcurar, M. (2012). Common fixed points for almost Prešić type operators. *Carpathian Journal of Mathematics*, 28(1):117–126. (Cited on page 12.)

Palais, R. S. (2007). A simple proof of the Banach contraction principle. *Journal of Fixed Point Theory and Applications*, 2(2):221–223. (Cited on page 17.)

Park, S. (1999). Ninety years of the Brouwer fixed point theorem. *Vietnam Journal of Mathematics*, 27(3):187–222. (Cited on page 52.)

Persiano, R. M., Comba, J. L. D., and Barbalho, V. (1993). An adaptive triangulation refinement scheme and construction. In *Proceedings of the VI Brazilian Symposium on Computer Graphics and Image Processing (SIBGRAPI)*. (Cited on page 80.)

Phuengrattana, W. and Suantai, S. (2012). Strong convergence theorems and rate of convergence of multi-step iterative methods for continuous mappings on an arbitrary interval. *Fixed Point Theory and Applications*, 2012(1):1–14. (Cited on pages 12 and 93.)

Pohlheim, H. (2006). Examples of objective functions. http://www.geatbx.com. (Cited on pages 97 and 118.)

Pop, N. (2008). Preconditioned Uzawa algorithm for elastic contact problems. *Proceedings in Applied Mathematics and Mechanics*, pages 10985–10986. (Cited on pages 96 and 118.)

Pop, N. (2009). *Variational analysis and numerical methods for contact problems in elasticity*. North University of Baia Mare. (Cited on pages 96 and 118.)

Popescu, O. (2007). Picard iteration converges faster than Mann iteration for a class of quasi-contractive operators. *Mathematical Communications*, 12(2):195–202. (Cited on pages 12 and 93.)

Rhoades, B. and Xue, Z. (2010). Comparison of the rate of convergence among Picard, Mann, Ishikawa, and Noor iterations applied to quasicontractive maps. *Fixed Point Theory and Applications*, 2010. (Cited on pages 12 and 93.)

Rivara, C. M. (1984). Algorithms for refining triangular grids suitable for adaptive and multigrid techniques. *International journal for numerical methods in Engineering*, 20(4):745–756. (Cited on page 80.)

Rus, I. A. (1979). *Principii si aplicatii ale teoriei punctului fix*. Dacia, Cluj-Napoca. (Cited on pages 14, 52, 54, and 55.)

Rus, I. A., Petruşel, A., and Petruşel, G. (2008). *Fixed Point Theory*. Cluj University Press Cluj-Napoca. (Cited on pages 8, 14, 52, and 53.)

Saigal, R. (1977). On the convergence rate of algorithms for solving equations that are based on methods of complementary pivoting. *Mathematics of Operations Research*, 2(2):108–124. (Cited on page 95.)

Saigal, R. (1979). Fixed point computing methods. *Encyclopedia of computer science an technology*, 8:545–566. (Cited on pages 63, 77, and 96.)

Saigal, R. and Todd, M. J. (1978). Efficient acceleration techniques for fixed point algorithms. *SIAM Journal on Numerical Analysis*, 15(5):997–1007. (Cited on page 82.)

Scarf, H. E. (1967). The approximation of fixed points of a continuous mapping. *SIAM Journal on Applied Mathematics*, 15(5):1328–1343. (Cited on pages 11, 61, 74, and 95.)

Scarf, H. E. (1982). The computation of equilibrium prices: an exposition. *Handbook of mathematical economics*, 2:1007–1061. (Cited on pages 61, 62, 63, and 66.)

Scarf, H. E. (1983). Fixed-point theorems and economic analysis: Mathematical theorems can be used to predict the probable effects of changes in economic policy. *American Scientist*, 71(3):289–296. (Cited on page 95.)

Scarf, H. E. (1991). The origins of fixed point methods. *History of mathematical programming: A collection of personal reminiscences*, pages 126–134. (Cited on page 61.)

Scarf, H. E. and Hansen, T. (1973). *The computation of economic equilibria*. Yale University Press New Haven, Conneticut. (Cited on page 74.)

Schaefer, H. (1957). Über die Methode sukzessiver Approximationen. *Jahresbericht der Deutschen Mathematiker-Vereinigung*, 59:131–140. (Cited on page 36.)

Smart, D. R. (1974). *Fixed point theorems*. Cambridge University Press. (Cited on page 52.)

Sondjaja, M. (2008). Sperner's lemma implies Kakutani's fixed point theorem. (Cited on pages 59 and 60.)

Sperner, E. (1928). Neuer beweis für die Invarianz der Dimensionszahl und des Gebietes. In *Abhandlungen aus dem Mathematischen Seminar der Universität Hamburg*, volume 6, pages 265–272. Springer. (Cited on pages 57 and 63.)

Sperner, E. (1980). Fifty years of further development of a combinatorial lemma. *Numerical Solution of Highly Nonlinear Problems, North-Holland*, pages 183–218. (Cited on page 57.)

Su, F. E. (1999). Rental harmony: Sperner's lemma in fair division. *The American mathematical monthly*, 106(10):930–942. (Cited on page 57.)

Todd, M. J. (1976a). The computation of fixed points and applications. *Lecture Notes in Economics and Mathematical Systems*, 124. (Cited on pages 44 and 50.)

Todd, M. J. (1976b). On triangulations for computing fixed points. *Mathematical Programming*, 10(1):322–346. (Cited on page 77.)

Todd, M. J. (1977). Union Jack triangulations. *Fixed points: algorithms and applications*. (Cited on pages 77 and 79.)

Todd, M. J. (1984). J': A new triangulation of R^n. *SIAM Journal on Algebraic Discrete Methods*, 5(2):244–254. (Cited on page 51.)

Todd, M. J. and Tunçel, L. (1993). A new triangulation for simplicial algorithms. *SIAM Journal on Discrete Mathematics*, 6(1):167–180. (Cited on page 51.)

van der Laan, G. and Talman, A. (1981). A class of simplicial restart fixed point algorithms without an extra dimension. *Mathematical Programming*, 20(1):33–48. (Cited on pages 74 and 118.)

Verschelde, J. (1999). Algorithm 795: PHCpack: A general-purpose solver for polynomial systems by homotopy continuation. *ACM Transactions on Mathematical Software (TOMS)*, 25(2):251–276. (Cited on page 95.)

Watson, L. T. and Haftka, R. T. (1989). Modern homotopy methods in optimization. *Computer Methods in Applied Mechanics and Engineering*, 74(3):289–305. (Cited on page 95.)

Weiss, K. and Floriani, L. D. (2011). Simplex and diamond hierarchies: Models and applications. In *Computer Graphics Forum*, volume 30, pages 2127–2155. Wiley Online Library. (Cited on page 80.)

Wright, A. H. (1981). The octahedral algorithm, a new simplicial fixed point algorithm. *Mathematical Programming*, 21(1):47–69. (Cited on pages 74 and 118.)

Xue, Z. (2008). The comparison of the convergence speed between Picard, Mann, Krasnoselskij and Ishikawa iterations in Banach spaces. *Fixed Point Theory and Applications*, 2008. (Cited on pages 12 and 93.)

Yang, Z. (1996). Simplicial fixed point algorithms and applications. Technical report, Tilburg University. (Cited on pages 44 and 55.)