Modeling the 8-Queens Problem and Sudoku using an Algorithm based on Projections onto Nonconvex Sets

by

Jason Schaad

B.Sc., The University of British Columbia, 2000
M.A., Gonzaga University, 2006

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

in

The College of Graduate Studies

(Mathematics)

THE UNIVERSITY OF BRITISH COLUMBIA

(Okanagan)

August 2010

Abstract

We begin with some basic definitions and concepts from convex analysis and projection onto convex sets (POCS). We next introduce various algorithms for converging to the intersection of convex sets and review various results (Elser's Difference Map is equivalent to the Douglas-Rachford and Fienup's Hybrid Input-Output algorithms which are both equivalent to the Hybrid Projection-Reflection algorithm). Our main contribution is two-fold. First, we show how to model the 8-queens problem and following Elser, we model Sudoku as well. In both cases we use several very nonconvex sets and while the theory for convex sets does not apply, so far the algorithm finds a solution. Second, we show that the operator governing the Douglas-Rachford iteration need not be a proximal map even when the two involved resolvents are; this refines an observation of Eckstein.

Table of Contents

List of Figures

Acknowledgements

First and foremost, I thank my supervisor Heinz Bauschke. His enthusiasm for mathematics is truly infectious. Without the support from my wife Yuki, the flexibility of my boss Jon Rever and the help of Liangjin Yao, I would not have been able to complete this thesis- thank you. I also thank Shawn Wang, Yves Lucet, Yong Gao, and Pulkit Bansal for careful reading of my thesis.

Chapter 1

Introduction

In mid August 2007, I attended the Second Mathematical Programming Society International Conference on Continuous Optimization [ICC07] held at McMaster University in Hamilton Ontario. I had the pleasure of attending a talk given by Dr. Veit Elser from Cornell University. Elser talked very passionately about a new algorithm he discovered while researching the hybrid input-output (HIO) algorithm of James R. Fienup, Ph.D., a University of Rochester optics researcher. I was intrigued by the variety of problems Elser was able to solve with his difference map.

A large number of problems can be formulated as follows: find an element of $A \cap B$ with the understanding that finding elements of A and B is easy [ERT07]. "Difficult problems can often be broken down into a collection of smaller, more tractable, subproblems." [GE08]. Elser gives a very easy to understand example: Suppose you want to unscramble the anagram "ilnoostu". We think about two sets, A and B. Set A is the 20,160 distinct permutations of the given 8 letters. Set B, is the collection of eight-letter English words; the cardinality of set B is approximately 30,000. The intersection of these two sets contains our answer, in fact $A \cap B = \{\text{solution}\}$.

The algorithm projects onto the sets A and B, this projection needs to be computationally quick in order for the difference map to be successful.

Convergence to a fixed point has been proved when our sets are convex [BCL02]. This of course is not the case with Sudoku, as we are working with positive integers from 1 to 9. But, the Sudoku solver has solved every Sudoku problem we have thrown at it. In fact, Science News published an article about the Sudoku Solver [Reh] and people from around the world tried to find a Sudoku puzzle that it could not solve, but to no avail. The server that hosted the PHP code to solve Sudoku was overloaded because of the interest. The Internet Service Provider emailed and was upset because the algorithm was using over 99 percent of virtual server's processor; we had to move the code to the University of British Columbia Servers. We were hoping to find a Sudoku puzzle that didn't work so that we can understand what makes that particular Sudoku special. There may be a theorem out there that shows our method always works on Sudoku. We leave that for

future work in this area.

Modeling Sudoku is complicated. Perhaps the greatest contribution of this thesis is providing the full details of modeling Sudoku in order for the difference map to solve it; it is very ingenious. I want to be very clear, we did not discover the Sudoku model, Dr. Veit Elser did. In fact, talking with him at the International Math Conference in Banff in 2009 [bir09], I asked him how he ever came up with such a clever model. Elser told me simply, "I was motivated". He knew his difference map could solve something as difficult as Sudoku puzzle, so he was determined to show the world.

We wanted to try to model something simpler in order to better explain how to model Sudoku. Once we explain how to model 8-queens, the modeling of Sudoku is much easier to understand. The modeling of the 8-queens problem is a new application, and as far as I know Elser has not attempted it.

As mathematicians we want to know why it works. The study of projection operators reveals a strong connection to monotone operators. We revisit work by Eckstein, correct a computational error, and provide a much simpler example.

The remainder of this thesis is organized as follows. Chapter 2 is a review of the well known definitions and concepts from the broad area of convex analysis. We build the theory so we can define a Hilbert space and projections.

In Chapter 3 we consider an algorithm for converging to the intersection of convex sets and review various related results. We show that Elser's Difference Map is equivalent to the Douglas-Rachford and Fienup's Hybrid Input-Output algorithms.

In Chapter 4 we show that the operator governing the Lions-Mercier iteration need not be a proximal map even when the two involved resolvents are; this refines an observation of Eckstein.

Chapter 5 focusses on modeling the 8-queens problem, and Chapter 6 concerns the Sudoku puzzle. Finally, Chapter 7 concludes this thesis.

Chapter 2

Background

This section is included so the results of this thesis can be understood in one self contained document. The readers can first familiarize themselves with the well known facts presented in this chapter.

2.1 Vector Space

Definition 2.1 (Vector space)**.** A vector space V over $\mathbb{R}$ is a nonempty set or collection of elements (called vectors) with two algebraic operations. These operations are called vector addition and multiplication of vectors by scalars. For all vectors x,y,$z \in V$ and α,$\beta \in \mathbb{R}$ the following conditions are required to hold:

1. $x + y$, called the sum of x and y, is a vector in V and is the unique result of addition applied to x and y.
2. αx, called the scalar multiple of x by α, is a vector in V and is the unique result of scalar multiplication applied to x by α.
3. Addition is commutative, $x + y = y + x$.
4. Addition is associative, $x + (y + z) = (x + y) + z$.
5. There is a zero vector 0 in V such that $x + 0 = x$.
6. There is an additive inverse $-x$ such that $x + (-x) = 0$.
7. Scalar multiplication is distributive, $\alpha(x+y) = \alpha x+\alpha y$, and $(\alpha+\beta)x = \alpha x + \beta x$.
8. Scalar multiplication is commutative, $\alpha(\beta x) = (\beta\alpha)x$.
9. Scalar multiplication by 1 is the identity operator, $1x = \mathrm{Id}\, x = x$.

2.2 Subspaces and Convex Sets

Definition 2.2 (subspace)**.** A subset W of a vector space V is a subspace of V if W is a vector space itself with the same operations of addition and scalar multiplication

We note that the zero vector, 0, is in every subspace. The smallest possible subspace is $\{0\}$. And the largest possible subspace is the vector space V. It is very tedious and time consuming to check if a subset of V is a subspace by applying the definition of a vector space. The following two facts are well known easier ways to verify if W is indeed a subspace. The proofs are omitted, but can be found in many Linear Algebra textbooks.

Fact 2.3. [Mey00, page 162] *Let V be a real vector space. Then W is a subspace of V if and only if $W \subseteq V$ and $\alpha x + y \in W \quad \forall x, y \in W, \quad \forall \alpha \in \mathbb{R}$.*

Proposition 2.4. *If W_1 and W_2 are subspaces of a real vector space V, then $W_1 \cap W_2$ is a subspace of the vector space V.*

Proof. By Fact 2.3 we need only show that $W_1 \cap W_2 \subseteq V$ and $\alpha x + y \in W_1 \cap W_2 \quad \forall x, y \in W \quad \forall \alpha \in \mathbb{R}$. Pick $x \in W_1 \cap W_2$, then $x \in W_1 \Rightarrow x \in V$ as W_1 is a subspace.
Now pick $x, y \in W_1 \cap W_2$ this implies that

$$x \in W_1 \text{ and } x \in W_2, \tag{2.1}$$
$$y \in W_1 \text{ and } y \in W_2. \tag{2.2}$$

Since W_1 and W_2 are both subspaces,

$$\alpha x + y \in W_1 \tag{2.3}$$
$$\alpha x + y \in W_2. \tag{2.4}$$

We conclude that $\alpha x + y \in W_1 \cap W_2$. □

Definition 2.5 (Affine subspace)**.** The set W is an affine subspace of V if there exists a vector $v \in V$ and a subspace W_0 of V such that $W = v + W_0 = \{v + w : w \in W_0\}$.

Definition 2.6 (Convex set)**.** Let X be a vector space and let $C \subseteq X$. We say C is convex if

$$\lambda C + (1 - \lambda)C \subseteq C, \quad \forall \lambda \in [0, 1]. \tag{2.5}$$

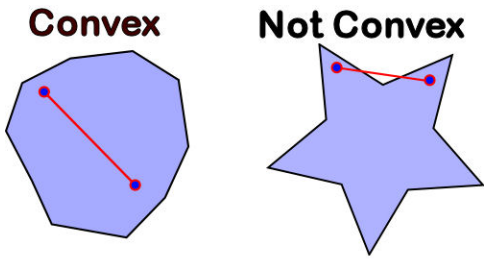


Figure 2.1: Example of Convexity

A geometric characterization of this definition is that a set is convex if it contains all line segments whose end points are in the set. See Figure 2.1

It is easy to see that subspaces are convex sets.

Definition 2.7 (Convex function)**.** Let $f : X \to]-\infty,+\infty]$. We say f is a convex function if

$$f((1-\lambda)x + \lambda y) \leq (1-\lambda)f(x) + \lambda f(y), \qquad \forall x, y \in X, \forall \lambda \in]0,1[\,. \tag{2.6}$$

Definition 2.8 (Domain)**.** Let $f : X \to]-\infty,+\infty]$. The domain of a function is

$$\operatorname{dom} f = \{x \,|\, f(x) < +\infty\}. \tag{2.7}$$

Definition 2.9 (Epigraph)**.** Let $f : X \to]-\infty,+\infty]$. Then the epigraph of a function is the set of points lying on or above the graph, i.e.,

$$\text{epi } f = \{(x,\mu) \,|\, x \in \operatorname{dom} f, \mu \in \mathbb{R}, \mu \geq f(x)\}.$$

A function is convex if and only if its epigraph is a convex set; this is called a geometric characterization of a convex function.

2.3 The Inner Product and Inner Product Spaces

Definition 2.10 (Inner Product)**.** An inner product on X is a mapping of $X \times X$ (where X is a vector space) to $\mathbb{R}$. For every pair of vectors x and y there is an associated scalar $\langle x, y\rangle$, called the inner product of x and y, such that for all $x, y, z \in X$ and for all $\alpha \in \mathbb{R}$ we have:

1. $\langle x + y, z\rangle = \langle x, z\rangle + \langle y, z\rangle$

2. $\langle \alpha x, y \rangle = \alpha \langle x, y \rangle$

3. $\langle x, y \rangle = \langle y, x \rangle$

4. $\langle x, x \rangle \geq 0$

5. $\langle x, x \rangle = 0 \iff x = 0$

Definition 2.11. *(Inner Product Space)* An inner product space (sometimes called a pre-Hilbert Space) is a vector space X with an inner product defined on X.

Definition 2.12. *(Orthogonality)* An element x of an inner product space X is said to be orthogonal to an element $y \in X$ if $\langle x, y \rangle = 0$.

Definition 2.13. *(Norm)* Let V be a vector space. The function $\|\cdot\| : V \to \mathbb{R}$ is called a norm if the following hold:

1. $\|x\| \geq 0$ and $\|x\| = 0$ if and only if $x = 0$

2. $\|x + y\| \leq \|x\| + \|y\|, \quad \forall x, y \in V$

3. $\|\alpha x\| = |\alpha| \|x\| \quad \forall \alpha \in \mathbb{R}, \forall x \in V$

An inner product on X defines a norm on X which is given by

$$\|x\| = \sqrt{\langle x, x \rangle} \geq 0. \tag{2.8}$$

Note that

$$\|x\|^2 = \langle x, x \rangle \geq 0. \tag{2.9}$$

We also have a metric on X which is given by

$$d(x, y) = \|x - y\| = \sqrt{\langle x - y, x - y \rangle} \geq 0. \tag{2.10}$$

2.4 Cauchy Sequences, Completeness and Hilbert spaces

Definition 2.14. *(Cauchy Sequence)* Let X be an inner product space. A sequence (x_n) in X is said to be a Cauchy sequence if for every $\epsilon > 0$ there is an $N = N(\epsilon)$ such that $\|x_m - x_n\| < \epsilon$ for every $m, n > N$.

Definition 2.15. *(Completeness)* The space X is said to be complete if every Cauchy sequence in X has a limit which is an element of X.

Definition 2.16. *(Hilbert Space)* A complete inner product space is said to be a Hilbert Space.

From now on we assume that X is a Hilbert Space.

Example 2.17 (Euclidean space $\mathbb{R}^n$)**.** The space $\mathbb{R}^n$ is a Hilbert Space. This is the primary Hilbert Space focussed on in this thesis. [Kre78, Example 1.5-1] shows us that $\mathbb{R}^n$ is complete. The inner product is defined by

$$\langle x, y\rangle = x_1y_1 + \cdots + x_ny_n, \tag{2.11}$$

where $x = (x_1, \cdots, x_n)$ and $y = (y_1, \cdots, y_n)$. Now from (2.11) we see that

$$\|x\| = \sqrt{\langle x, x\rangle} = \sqrt{x_1^2 + \cdots + x_n^2}. \tag{2.12}$$

If $n = 3$, (2.11) gives us the usual dot product

$$\langle x, y\rangle = x \cdot y = x_1y_1 + x_2y_2 + x_3y_3, \tag{2.13}$$

where $x = (x_1, x_2, x_3)$ and $y = (y_1, y_2, y_3)$. The orthogonality

$$\langle x, y\rangle = x \cdot y = 0 \tag{2.14}$$

agrees with the concept of perpendicularity.

Example 2.18 (ℓ_2)**.** *[Kre78, 1.2-3]* Pronounced "Little el 2" and sometimes written as $\ell_2(I)$ is the Hilbert space of sequences index by a countable set, usually $\mathbb{N}$ (but could be $\mathbb{Z}$). Each element x of the space ℓ_2 is a sequence $x = (x_n) = (x_1, x_2, \ldots, x_n, \ldots)$ with

$$\|(x_n)\| = \sqrt{\sum_{n=1}^{\infty} x_n^2} < +\infty. \tag{2.15}$$

For two elements (sequences) $x = (x_n) \in \ell_2$ and $y = (y_n) \in \ell_2$, the inner product is defined by

$$\langle x, y\rangle = \sum_{n=1}^{\infty} \langle x_n, y_n\rangle. \tag{2.16}$$

Definition 2.19 (Bounded)**.** Given a set A in X, A is said to be bounded if we have

$$\sup_{x\in A} \|x\| < +\infty$$

.

Theorem 2.20. *Every convergent sequence in a Hilbert space is a bounded Cauchy Sequence.*

Proof. Suppose $x_n \longrightarrow x$. We first show (x_n) is Cauchy. By the triangle inequality we have

$$\|x_n - x_m\| = \|x_n - x + x - x_m\| \leq \|x_n - x\| + \|x - x_m\|. \tag{2.17}$$

We now let $\varepsilon > 0$. Then $\exists N$ so for $n > N$ and $m > N$ we have

$$\|x_n - x\| < \frac{\varepsilon}{2}, \tag{2.18}$$

$$\|x - x_m\| < \frac{\varepsilon}{2}. \tag{2.19}$$

Hence $m, n > N$

$$\|x_n - x_m\| \leq \|x_n - x\| + \|x - x_m\| = \frac{\varepsilon}{2} + \frac{\varepsilon}{2} = \varepsilon, \tag{2.20}$$

which proves the sequence is Cauchy. Now we show (x_n) is bounded. Again we have $x_n \longrightarrow x$. For $\varepsilon = 1$, we get $N \in \mathbb{N}$ such that

$$n > N \Longrightarrow \|x_n - x\| < 1. \tag{2.21}$$

We claim

$$\|x_n\| - \|x\| \leq \|x_n - x\|. \tag{2.22}$$

By the triangle inequality

$$\|x_n\| = \| - x_n\| = \| - x_n + x - x\| = \|(x - x_n) + (-x)\| \leq \|x\| + \|x - x_n\|. \tag{2.23}$$

Thus, for $n > N$,

$$\|x_n\| < \|x\| + 1. \tag{2.24}$$

If we choose M to be

$$M = \max\{\|x\| + 1, \|x_1\|, \|x_2\|, ..., \|x_N\|\} \tag{2.25}$$

we get

$$\|x_n\| \leq M, \quad \forall n. \tag{2.26}$$

as desired. □

Definition 2.21 (Distance to a set)**.** The distance from a vector $x \in X$ to a nonempty set $A \subseteq X$ is defined as follows

$$d_A(x) := \inf_{y \in A} \|x - y\|.$$

Theorem 2.22 (Cauchy-Schwarz inequality)**.** *For any x and y in a Hilbert space X, we have that*

$$|\langle x, y \rangle| \leq \|x\| \|y\|.$$

Proof. We prove this with two cases.

Case 1: $x = 0$ or $y = 0$. Clear.

Case 2: $x \neq 0$ and $y \neq 0$. Then

$$0 \leq \left\langle \frac{x}{\|x\|} - \frac{y}{\|y\|}, \frac{x}{\|x\|} - \frac{y}{\|y\|} \right\rangle \tag{2.27}$$

$$= \left\| \frac{x}{\|x\|} - \frac{y}{\|y\|} \right\|^2 = \frac{\langle x, x \rangle}{\|x\|^2} - \frac{2\langle x, y \rangle}{\|x\|\|y\|} + \frac{\langle y, y \rangle}{\|y\|^2} \tag{2.28}$$

$$\iff \frac{\|x\|^2}{\|x\|^2} - \frac{2\langle x, y \rangle}{\|x\|\|y\|} + \frac{\|y\|^2}{\|y\|^2} \geq 0 \tag{2.29}$$

$$\iff 2 \geq \frac{2\langle x, y \rangle}{\|x\|\|y\|} \tag{2.30}$$

$$\iff \|x\|\|y\| \geq \langle x, y \rangle \tag{2.31}$$

$$\iff \langle x, y \rangle \leq \|x\|\|y\| \tag{2.32}$$

This is true for all x. Let x be replaced by $-x$. Now we have

$$\| - x\|\|y\| \geq \langle -x, y \rangle \tag{2.33}$$

$$\iff -\langle x, y \rangle \leq \|x\|\|y\|. \tag{2.34}$$

Now

$$\max\{\langle -x, y\rangle, -\langle x, y\rangle\} = |\langle x, y\rangle|. \tag{2.35}$$

Combine (2.32) and (2.34) to get

$$|\langle x, y\rangle| \leq \|x\|\|y\|, \quad \forall x, y \in X \tag{2.36}$$

as desired.

□

Theorem 2.23 (Triangle Inequality)**.** *Then for every $x, y \in X$, we have that*

$$\|x + y\| \leq \|x\| + \|y\|.$$

Proof. We have

$$\begin{aligned}
\|x + y\|^2 &= \langle x + y, x + y\rangle && (2.37)\\
&= \langle x, x + y\rangle + \langle y, x + y\rangle && (2.38)\\
&= \langle x, x\rangle + \langle x, y\rangle + \langle y, x\rangle + \langle y, y\rangle && (2.39)\\
&= \|x\|^2 + 2\langle x, y\rangle + \|y\|^2 && (2.40)\\
&\leq \|x\|^2 + 2|\langle x, y\rangle| + \|y\|^2. && (2.41)
\end{aligned}$$

By the Cauchy-Schwarz inequality we have

$$\begin{aligned}
\|x + y\|^2 &\leq \|x\|^2 + 2\|x\|\|y\| + \|y\|^2 && (2.42)\\
&= (\|x\| + \|y\|)^2. && (2.43)
\end{aligned}$$

We take the square root on both sides to obtain

$$\|x + y\| \leq \|x\| + \|y\| \tag{2.44}$$

as desired. □

Theorem 2.24 (Parallelogram law)**.** *For every $x, y \in X$*

$$\|x + y\|^2 + \|x - y\|^2 = 2(\|x\|^2 + \|y\|^2).$$

Proof. We have,

$$\begin{aligned}
\|x + y\|^2 + \|x - y\|^2 &= \langle x + y, x + y\rangle + \langle x - y, x - y\rangle && (2.45)\\
&= 2\langle x, x\rangle + 2\langle y, y\rangle + 2\langle x, y\rangle - 2\langle x, y\rangle && (2.46)\\
&= 2(\|x\|^2 + \|y\|^2). && (2.47)
\end{aligned}$$

□

2.5 Product Space

Definition 2.25 (Product Space)**.** Let $N \in \mathbb{N} = \{1, 2, 3, \ldots\}$, and X_i be Hilbert spaces, with inner products $\langle \cdot, \cdot \rangle_i$, for each $i \in \{1, \ldots, N\}$. The corresponding *product space* $H := X_1 \times X_2 \times \ldots \times X_N$ is defined by

$$H = \big\{(x_i) = (x_1, x_2, \ldots, x_N) \mid x_i \in X_i\big\}, \tag{2.48}$$

$$\langle x, y \rangle = \sum_{i=1}^{N} \langle x_i, y_i \rangle_i, \quad \forall x = (x_i), y = (y_i) \in H. \tag{2.49}$$

Then,

$$\|x\| = \sqrt{\sum_{i=1}^{N} \|x_i\|_i^2}, \quad \forall x = (x_i) \in H. \tag{2.50}$$

If $X_i = X, \forall i \in \{1, \cdots, N\}$, we denote H by X^N.

In fact, the well known space $\mathbb{R}^N$ is actually a product space. We can write $\mathbb{R}^N$ as $\mathbb{R} \times \cdots \times \mathbb{R}$.

Proposition 2.26. *The Product Space is a Hilbert space.*

Proof. We first prove that

$$\langle x, y \rangle = \sum_{i=1}^{N} \langle x_i, y_i \rangle_i, \quad \forall x = (x_i), y = (y_i) \in H. \tag{2.51}$$

is an inner product. Let $x = (x_i)$, $y = (y_i)$, $z = (z_i) \in H$, and $\alpha \in \mathbb{R}$. As described above, we need to show five properties in order to prove this is an inner product. Until the end of this proof, we abbreviate $\sum_{i=1}^{N}$ by $\sum$.
1.

$$\langle x + y, z \rangle = \langle (x_i) + (y_i), (z_i) \rangle = \langle (x_i + y_i), (z_i) \rangle = \sum \langle x_i + y_i, z_i \rangle_i \tag{2.52}$$

Since $x_i, y_i, z_i \in X_i$ and X_i is a Hilbert space, we can write

$$= \sum (\langle x_i, z_i \rangle_i + \langle y_i, z_i \rangle_i) = \sum \langle x_i, z_i \rangle_i + \sum \langle y_i, z_i \rangle_i = \langle x, z \rangle + \langle y, z \rangle. \tag{2.53}$$

2.

$$\langle \alpha x, y \rangle = \langle \alpha (x_i), (y_i) \rangle = \langle (\alpha x_i), (y_i) \rangle \tag{2.54}$$

Since $x_i, y_i, z_i \in X_i$ and X_i is a Hilbert space, we can write

$$\langle \alpha x, y\rangle = \sum \langle \alpha x_i, y_i\rangle_i = \sum \alpha \langle x_i, y_i\rangle_i = \alpha \sum \langle x_i, y_i\rangle_i = \alpha \langle x, y\rangle. \quad (2.55)$$

3.

$$\langle x, y\rangle = \langle (x_i), (y_i)\rangle = \sum \langle x_i, y_i\rangle_i = \sum \langle y_i, x_i\rangle_i = \langle (y_i), (x_i)\rangle = \langle y, x\rangle. \quad (2.56)$$

4.

$$\langle x, x\rangle = \langle (x_i), (x_i)\rangle = \sum \langle x_i, x_i\rangle_i = \sum \|x_i\|_i^2 \geq 0. \quad (2.57)$$

5. From above we know

$$\sum \|x_i\|_i^2 \geq 0. \quad (2.58)$$

Then $\langle x, x\rangle = 0 = \sum \|x_i\|_i^2 \Leftrightarrow x_i = 0, \forall i \Leftrightarrow x = (0, 0, \ldots) = 0$. Hence, $\langle \cdot, \cdot \rangle$ as defined above is an inner product.

It also induces the following norm,

$$\forall x = (x_i) \in H \quad (2.59)$$

$$\|x\| = \sqrt{\langle x, x\rangle} = \sqrt{\sum \|x_i\|_i^2}. \quad (2.60)$$

Finally, to show this is a Hilbert space, we need to show that it is complete. Let (x_n) be a Cauchy sequence in H. Write $x_n = (x_{n,i}) = (x_{n,1}, x_{n,2}, \ldots, x_{n,N})$. Since (x_n) is a Cauchy sequence we know that $\forall \varepsilon \exists N_1$ such that

$$\|x_n - x_m\| < \varepsilon, \quad \forall n, m > N_1 \quad (2.61)$$

If we square both sides we get

$$\|x_n - x_m\|^2 < \varepsilon^2. \quad (2.62)$$

Note that,

$$x_n - x_m \quad (2.63)$$
$$= (x_{n,1}, x_{n,2}, \ldots, x_{n,N}) - (x_{m,1}, x_{m,2}, \ldots, x_{m,N}) \quad (2.64)$$
$$= (x_{n,1} - x_{m,1}, x_{n,2} - x_{m,2}, \ldots, x_{n,N} - x_{m,N}). \quad (2.65)$$

Thus,

$$\|x_n - x_m\|^2 \tag{2.66}$$
$$= \|x_{n,1} - x_{m,1}\|_1^2 + \|x_{n,2} - x_{m,2}\|_2^2 + \ldots + \|x_{n,N} - x_{m,N}\|_N^2. \tag{2.67}$$

By (2.62)

$$\|x_{n,1} - x_{m,1}\|_1^2 + \|x_{n,2} - x_{m,2}\|_2^2 + \ldots + \|x_{n,N} - x_{m,N})\|_N^2 \leq \varepsilon^2. \tag{2.68}$$

So, this implies that

$$\|x_{n,i} - x_{m,i}\|_i^2 \leq \varepsilon^2, \quad \forall i. \tag{2.69}$$

Thus,

$$\|x_{n,i} - x_{m,i}\| \leq \varepsilon, \quad \forall i \quad \forall n, m > N_1, \tag{2.70}$$

i.e., $(x_{n,i})$ is a Cauchy sequence for every i, and $x_{n,i}$ is a sequence in X_i, which is complete. So $\exists x_{0,i} \in X_i$ such that $x_{n,i} \longrightarrow x_{0,i}$. Therefore,

$$x_n \longrightarrow (x_{0,1}, x_{0,2}, \ldots, x_{0,N}). \tag{2.71}$$

We have shown that our product space is indeed complete and thus a Hilbert space. □

Definition 2.27 (Diagonal space)**.** Let $N \in \mathbb{N}$, and let X be a Hilbert space. The *diagonal space* Δ in X^N is the subspace

$$\Delta = \{(x, \ldots, x) \in X^N \mid x \in X\}. \tag{2.72}$$

2.6 Lipschitz and Nonexpansivity

Definition 2.28 (Lipschitz)**.** Let $T : X \to X$. Then T is said to be α-Lipschitz continuous if

$$(\forall x \in X)(\forall y \in X) \quad \|Tx - Ty\| \leq \alpha \|x - y\|.$$

Example 2.29 (The Identity)**.** The Identity operator on X is 1-Lipschitz; this is very easy to show:

$$\|x - y\| = \|\operatorname{Id}(x) - \operatorname{Id}(y)\| \leq 1 \cdot \|x - y\|.$$

Theorem 2.30. *Let $F_1 : X \to X$ be L_1-Lipschitz and $F_2 : X \to X$ be L_2-Lipschitz, then $F_1 + F_2$ is $L_1 + L_2$ Lipschitz.*

Proof. We start with the definition of being Lipschitz continuous.

$$F_1 \quad \text{is} \quad L_1\text{−Lipschitz} \iff \forall x\, \forall y \quad \|F_1(x) - F_1(y)\| \leq L_1 \cdot \|x - y\| \tag{2.73}$$

$$F_2 \quad \text{is} \quad L_2\text{−Lipschitz} \iff \forall x\, \forall y \quad \|F_2(x) - F_2(y)\| \leq L_2 \cdot \|x - y\| \tag{2.74}$$

Hence,

$$\|(F_1 + F_2)(x) - (F_1 + F_2)(y)\| = \|(F_1(x) + F_2(x)) - (F_1(y) + F_2(y))\| \tag{2.75}$$

$$= \|(F_1(x) - F_1(y)) + (F_2(x) - F_2(y))\|, \tag{2.76}$$

which, by the triangle inequality, is

$$\leq \|F_1(x) - F_1(y)\| + \|F_2(x) - F_2(y)\|. \tag{2.77}$$

Now since F_1 is L_1-Lipschitz and F_2 is L_2-Lipschitz we get

$$\leq L_1 \cdot \|x-y\| + L_2 \cdot \|x-y\| = \|x-y\| \cdot (L_1+L_2) = (L_1+L_2) \cdot \|x-y\|, \tag{2.78}$$

as desired. □

Definition 2.31 (Nonexpansivity)**.** Let $T : X \to X$. We say T is nonexpansive if

$$(\forall x \in X)(\forall y \in X) \quad \|Tx - Ty\| \leq \|x - y\|.$$

Remark 2.32. We note that T being nonexpansive means it is 1-Lipschitz continuous. Therefore, the Identity is not only Lipschitz continuous but it is also nonexpansive.

Example 2.33 (The Right Shift Operator)**.** The circular right shift operator $R : X^n \to X^n : (x_1, \ldots, x_n) \mapsto (x_n, x_1, \ldots, x_{n-1})$ satisfies $\|Rx\| = \|x\|$ so that, using linearity of R,

$$\|R(x - y)\| = \|Rx - Ry\| = \|x - y\|,$$

and hence R is nonexpansive.

Example 2.34 (Linear operator on ℓ_2)**.** Recall that an element x in ℓ_2 is of the form $x = (x_n) = (x_1, x_2, \ldots)$ such that

$$\|(x_n)\| = \sqrt{\sum_{n=1}^{\infty} {x_n}^2} < +\infty \tag{2.79}$$

We define a linear operator $T : \ell_2 \to \ell_2$ by

$$Tx := (\frac{x_n}{n}) = (x_1, \frac{x_2}{2}, \frac{x_3}{3}, \ldots, \frac{x_n}{n}, \ldots), \quad \forall x = (x_n) = (x_1, x_2, \ldots) \in \ell_2. \tag{2.80}$$

So we have $(\forall x = (x_n) \in \ell_2)$

$$\|Tx\|^2 = \sum_{i=1}^{\infty} (\frac{x_i^2}{i^2}) \leq \sum_{i=1}^{\infty} (x_i^2) = \|x\|^2 \quad \text{(by (2.79))}. \tag{2.81}$$

Taking the square root on both sides,

$$\|Tx\| \leq \|x\|, \quad \forall x \in \ell_2. \tag{2.82}$$

Now by linearity of T we have

$$\|Tx - Ty\| = \|T(x - y)\| \leq \|x - y\|, \quad \forall x, y \in \ell_2. \tag{2.83}$$

Therefore, T is nonexpansive.

Definition 2.35 (Continuity)**.** Let $T : X \to X$. T is said to be continuous if for all sequences $(x_n) \in X$ with $x_n \to x$, then $Tx_n \to Tx$.

Alternatively, given an arbitrary element $x \in X$, for all $\epsilon > 0$ there exists $\delta > 0$ such that $\|Tx - Ty\| < \epsilon$ whenever $y \in X$ with $\|x - y\| < \delta$.

Proposition 2.36. *If F is L-Lipschitz, then F is continuous.*

Proof. The result is clear when $L = 0$. So assume $L > 0$. Let $c \in X$. We will show that F is continuous at c. Given $\varepsilon > 0$, we need to find $\delta > 0$ such that $\|F(x) - F(c)\| < \varepsilon$ when $\|x - c\| < \delta$.

We now set $\delta := \dfrac{\varepsilon}{L} > 0$.

Then $(\forall x \in X)$,

$$\|x - c\| < \delta \Rightarrow \|F(x) - F(c)\| \leq L\|x - c\| \leq L\delta = \varepsilon. \tag{2.84}$$

So $\|F(x) - F(c)\| < \varepsilon$ as desired. □

Theorem 2.37. *The composition of nonexpansive operators is nonexpansive.*

Proof. Let T and N be nonexpansive, and let x and y be in X. Then,

$$\|T(Nx) - T(Ny)\| \leq \|Nx - Ny\| \leq \|x - y\|. \tag{2.85}$$

□

Definition 2.38 (Firmly nonexpansive)**.** Let $T : X \to X$. We say T is firmly nonexpansive if

$$(\forall x \in X)(\forall y \in X) \quad \|Tx - Ty\|^2 \leq \langle x - y, Tx - Ty\rangle$$

Theorem 2.39. *Let $T : X \to X$. Then the following statements are equivalent:*

1. *T is firmly nonexpansive.*
2. *$\|Tx - Ty\|^2 + \|(x - Tx) - (y - Ty)\|^2 \leq \|x - y\|^2 \quad \forall x, \forall y.$*
3. *$\|(2Tx - x) - (2Ty - y)\| \leq \|x - y\| \quad \forall x, \forall y.$*
4. *$\mathrm{Id} - T$ is firmly nonexpansive*
5. *$T = \frac{1}{2}N + \frac{1}{2}\mathrm{Id}$, where N is nonexpansive.*

Proof. We rewrite the statements above by letting $a = x - y$, $b = Tx - Ty$. The following statements are equivalent:

1. $\|b\|^2 \leq \langle a, b\rangle$.
2. $\|b\|^2 + \|a - b\|^2 \leq \|a\|^2$.
3. $\|2b - a\| \leq \|a\|$.
4. $\mathrm{Id} - T$ is firmly nonexpansive.
5. $T = \frac{1}{2}N + \frac{1}{2}\mathrm{Id}$, where N is nonexpansive

$$\begin{aligned}
2 &\iff \|b\|^2 + \|a - b\|^2 \leq \|a\|^2 && (2.86)\\
&\iff \|b\|^2 + \|a\|^2 - 2\langle a, b\rangle + \|b\|^2 \leq \|a\|^2 && (2.87)\\
&\iff 2\|b\|^2 \leq 2\langle a, b\rangle && (2.88)\\
&\iff \|b\|^2 \leq \langle a, b\rangle && (2.89)\\
&\iff 1 && (2.90)
\end{aligned}$$

$$\begin{aligned}
3 &\iff \|2b - a\|^2 \leq \|a\|^2 && (2.91)\\
&\iff 4\|b\|^2 - 4\langle a, b\rangle + \|a\|^2 \leq \|a\|^2 && (2.92)\\
&\iff 4\|b\|^2 \leq 4\langle a, b\rangle && (2.93)\\
&\iff \|b\|^2 \leq \langle a, b\rangle && (2.94)\\
&\iff 1 && (2.95)
\end{aligned}$$

$$4 \iff \|(\mathrm{Id}-T)x-(\mathrm{Id}-T)y\|^2 \leq \langle x-y,(\mathrm{Id}-T)x-(\mathrm{Id}-T)y\rangle \tag{2.96}$$
$$\iff \|a-b\|^2 \leq \langle a, a-b\rangle \tag{2.97}$$
$$\iff \|a\|^2 - 2\langle a,b\rangle + \|b\|^2 \leq \langle a,a\rangle - \langle a,b\rangle \tag{2.98}$$
$$\iff \|a\|^2 + \|b\|^2 \leq \|a\|^2 - \langle a,b\rangle + 2\langle a,b\rangle \tag{2.99}$$
$$\iff \|b\|^2 \leq \langle a,b\rangle \tag{2.100}$$
$$\iff 1 \tag{2.101}$$

Note that

$$T = \frac{1}{2}\mathrm{Id} + \frac{1}{2}N \iff N = 2T - \mathrm{Id}\,. \tag{2.102}$$

Hence,

$$3 \iff N \text{ is nonexpansive} \tag{2.103}$$
$$\iff 5. \tag{2.104}$$

□

The last equivalence is a tool to create new firmly nonexpansive mappings from nonexpansive mappings.

Example 2.40. We define A on ℓ_2 as follows:

$$A : (x_1, x_2, x_3, \ldots) \longmapsto (x_1, \frac{x_2}{2}, \frac{x_3}{3}, \ldots). \tag{2.105}$$

If we take $A + \mathrm{Id}$ we get

$$A + \mathrm{Id} : (x_1, x_2, x_3, \ldots) \longmapsto (2x_1, \frac{3}{2}x_2, \frac{4}{3}x_3, \ldots). \tag{2.106}$$

We now define F to be the inverse,

$$F := (A+\mathrm{Id})^{-1} : (\xi_1, \xi_2, \xi_3, \ldots) \longmapsto (\frac{1}{2}\xi_1, \frac{2}{3}\xi_2, \frac{3}{4}\xi_3, \ldots). \tag{2.107}$$

We will now show that $F := (A+\mathrm{Id})^{-1}$ is firmly nonexpansive. By (2.107), F is linear. We will now show that

$$\|Fx\|^2 \leq \langle Fx, x\rangle \quad \forall x, \tag{2.108}$$

which, by linearity of F, is equivalent to the firm nonexpansivity of F. Let $y = Fx$. Then $x = F^{-1}y = (A + \mathrm{Id})y$. Then $\|Fx\|^2 = \|y\|^2$, as they are equal, and since $\langle y, Ay\rangle \geq 0$,

$$\langle Fx, x\rangle = \langle y, (A + \mathrm{Id})y\rangle = \langle y, Ay\rangle + \|y\|^2 \geq \|y\|^2 = \|Fx\|^2. \tag{2.109}$$

We have shown that $\langle Fx, x\rangle \geq \|Fx\|^2$ and therefore F is firmly nonexpansive.

Corollary 2.41. *All firmly nonexpansive mappings are nonexpansive.*

Proof. For T, a firmly nonexpansive mapping, we know that $\forall x, \forall y$

$$\|T(x) - T(y)\|^2 \leq \langle T(x) - T(y), x - y\rangle \leq \|Tx - Ty\| \cdot \|x - y\|. \tag{2.110}$$

We now consider two cases. Case 1: $\|T(x) - T(y)\| = 0$.

$$\|T(x) - T(y)\| = 0 \leq \|x - y\| \tag{2.111}$$

as norms are always greater or equal to zero. So we have,

$$\|T(x) - T(y)\| \leq \|x - y\|, \tag{2.112}$$

therefore T is nonexpansive, as desired. Case 2: $\|T(x) - T(y)\| > 0$. Again we have

$$\|T(x) - T(y)\|^2 \leq \|T(x) - T(y)\| \cdot \|x - y\|. \tag{2.113}$$

Divide both sides by $\|T(x) - T(y)\|$ to obtain,

$$\|T(x) - T(y)\| \leq \|x - y\|, \tag{2.114}$$

as desired. □

2.7 Projections

The main focus of this section is to provide an introduction to projections onto convex sets.

Definition 2.42. Let $C \subseteq X$, $x \in X$, and $c^* \in C$. Then c^* is the **projection** of x onto C if

$$(\forall c \in C) \quad \|x - c^*\| \leq \|x - c\|.$$

In other words, c^* is a closest point to $x \in C$.

The associated operator $P_C : X \rightrightarrows C : x \mapsto \{c^* \in C |\ c^* \text{ is a projection of } x \text{ onto } C\ \}$ is called the **projection operator** or **projector**. If $P_C x = \{c^*\}$ we also write $c^* = P_C x$ rather than $\{c^*\} = P_C x$, which is a slight abuse of notation.

Lemma 2.43. *Let u and v be in X. Then the following holds:*

$$\langle u, v\rangle \leq 0 \Longleftrightarrow (\forall \alpha \in \mathbb{R}_+)\|u\| \leq \|u - \alpha v\| \Longleftrightarrow (\forall \alpha \in [0,1])\|u\| \leq \|u - \alpha v\|.$$

Proof. Observe that

$$(\forall \alpha \in \mathbb{R}) \quad \|u - \alpha v\|^2 - \|u\|^2 = \alpha(\alpha\|v\|^2 - 2\langle u, v\rangle). \tag{2.115}$$

Hence, for forward implications follow immediately. Conversely, if for every $\alpha \in]0,1]$, $\|u\| \leq \|u - \alpha v\|$, then (2.115) implies that $\langle u, v\rangle \leq \alpha\|v\|^2/2$. As $\alpha \downarrow 0$, we obtain $\langle u, v\rangle \leq 0$. □

Theorem 2.44 (The Projection Theorem)**.** *Let C be a nonempty closed convex subset of X. Then $\forall x \in X$ there exists a unique $c^* \in C$ such that $\|x - c^*\| = d_C(x)$. Moreover, c^* is characterized by*

$$c^* \in C \tag{2.116}$$

and

$$(\forall c \in C) \quad \langle c - c^*, x - c^*\rangle \leq 0. \tag{2.117}$$

Proof. Fix $x \in X$. Pick a sequence (c_n) in C such that $\|x - c_n\| \longrightarrow d_C(x)$. By the parallelogram law,

$$\|c_n - c_m\|^2 = 2\|c_n - x\|^2 + 2\|c_m - x\|^2 - \|(c_n + c_m) - 2x\|^2 \tag{2.118}$$

$$= 2\|c_n - x\|^2 + 2\|c_m - x\|^2 - 4\|\frac{c_n + c_m}{2} - x\|^2 \tag{2.119}$$

$$\leq 2\|c_n - x\|^2 + 2\|c_m - x\|^2 - 4(d_C(x))^2; \tag{2.120}$$

as $d_C(x) \leq \|x - \dfrac{c_n + c_m}{2}\|$. Now as $n, m \longrightarrow \infty$

$$\longrightarrow 2(d_C^2(x)) + 2(d_C(x))^2 - 4(d_C(x))^2 = 0. \tag{2.121}$$

Therefore (c_n) is Cauchy. Hence (c_n) is Cauchy and thus convergent as X is a Hilbert space and thus complete:

$$c_n \longrightarrow c^* \in X. \tag{2.122}$$

But (c_n) lies in C and C is closed, so $c^* \in C$ as well. Now since $c_n \longrightarrow c^*$ we also have that $x - c_n \longrightarrow x - c^*$ and thus $\|x - c_n\| \longrightarrow \|x - c^*\|$. On the other hand, $\|x - c_n\| \to d_C(x)$ by assumptions. Since limits are unique $\|x - c^*\| = d_C(x)$ so $c^* \in P_C x$. We have shown existence. Now we need to show uniqueness. We let c' be another nearest point in $P_C x$. We need only show that $c^* = c'$. Set $(c'_n) = (c^*, c', c^*, c', ...)$, then $\|x - c'_n\| = d_C(x)$. By (2.119) we deduce that $\|c^* - c'\|^2 = 0$ i.e., $c^* = c'$.

Now for the characterization. Pick $c \in C$ and set $c_\alpha = \alpha c + (1-\alpha)c^* \in C$ where $\alpha \in [0, 1]$. Then, using Lemma 2.43 (with $u = x - c^*$ and $v = c - c^*$), we obtain

$$\|x - c^*\| = d_C(x) \tag{2.123}$$

$$\iff (\forall c \in C)(\forall \alpha \in [0,1]) \quad \|x - c^*\| \leq \|x - c_\alpha\| = \|(x - c^*) - \alpha(c - c^*)\| \tag{2.124}$$

$$\iff (\forall c \in C)\langle x - c^*, c - c^*\rangle \leq 0. \tag{2.125}$$

□

We now give a few examples of projections.

Example 2.45 (Hyperplane)**.** (See [Deu01, page 99].) If $a \in X \backslash \{0\}, \beta \in \mathbb{R}$ and $C = \{x \in X | \langle a, x\rangle = \beta\}$ then

$$P_C x = x - \frac{(\langle a, x\rangle - \beta)}{\|a\|^2} a \tag{2.126}$$

Proof. We shall use the characterization of the Projection theorem. It can be easily shown that our set C is closed and convex. Next we call our projection candidate c^* and check that $c^* \in C$.

$$\langle a, x - \frac{(\langle a, x\rangle - \beta)}{\|a\|^2} a\rangle = \langle a, x\rangle - \frac{(\langle a, x\rangle - \beta)}{\|a\|^2}\langle a, a\rangle = \beta. \tag{2.127}$$

Next we take $c \in C$, which means that $\langle a, c\rangle = \beta$. Then, by simplifying and recalling that $\langle a, c\rangle = \beta$ and $\langle a, a\rangle = \|a\|^2$,

$$\langle c - (x - \frac{\langle a, x\rangle - \beta}{\|a\|^2}a), \frac{(\langle a, x\rangle - \beta)}{\|a\|^2}a\rangle \tag{2.128}$$

$$= \frac{\langle a, x\rangle - \beta}{\|a\|^2}(\langle c, a\rangle - \langle x, a\rangle + \frac{(\langle a, x\rangle - \beta)}{\|a\|^2}\langle a, a\rangle) \tag{2.129}$$

$$= \frac{\langle a, x\rangle - \beta}{\|a\|^2}(\beta - \langle x, a\rangle + \langle a, x\rangle - \beta) \tag{2.130}$$

$$= \frac{\langle a, x\rangle - \beta}{\|a\|^2}(0) \tag{2.131}$$

$$= 0 \tag{2.132}$$

$$\leq 0. \tag{2.133}$$

Since the two conditions for the characterization are satisfied, our candidate c^* is indeed the projection $P_C x$ □

Example 2.46 (The unit ball)**.** We provide the projection onto the unit ball, $C := B(0; 1) = \{x | \|x\| \leq 1\}$:

$$P_C x = \frac{x}{\max\{\|x\|, 1\}}, \quad \forall x \in X. \tag{2.134}$$

Proof. C is closed and convex this implies that our projection is a singleton by Theorem 2.44. We denote c^* our projection candidate in (2.134). To prove this is a projection we need to show two things:

1. $c^* \in C$
2. $\langle c - c^*, x - c^*\rangle \leq 0 \quad \forall c \in C, \quad \forall x \in X$

Case 1: Assume that $x \in C$. This means that x is in the unit ball, so $\max\{\|x\|, 1\} = 1$ and hence $c^* = x$.

1. Trivial by assumptions.
2. $\langle c - c^*, x - c^*\rangle = \langle c - x, x - x\rangle = \langle c - c^*, 0\rangle = 0 \leq 0.$

Case 2: Assume that $x \notin C$.
1. $x \notin C$ implies $\|x\| > 1$ and $c^* = \frac{x}{\|x\|}$ by definition. We also have that $\|c^*\| = 1$ so $c^* \in C$:
2. $\forall c \in C$ we have $\|c\| \leq 1$ by definition

$$\langle c - c^*, x - c^* \rangle = \langle c - \frac{x}{\|x\|}, x - \frac{x}{\|x\|} \rangle \tag{2.135}$$

$$= \langle c - \frac{x}{\|x\|}, (1 - \frac{1}{\|x\|})x \rangle = \langle c, (1 - \frac{1}{\|x\|})x \rangle - \langle \frac{x}{\|x\|}, (1 - \frac{1}{\|x\|})x \rangle. \tag{2.136}$$

Now since

$$\frac{(1 - \frac{1}{\|x\|})}{\|x\|} \langle x, x \rangle = \frac{(1 - \frac{1}{\|x\|})}{\|x\|} \|x\|^2 = \|x\|(1 - \frac{1}{\|x\|}) = \|x\| - 1 > 0, \tag{2.137}$$

we have that

$$\langle c - c^*, x - c^* \rangle = \tag{2.138}$$

$$\langle c, (1 - \frac{1}{\|x\|})x \rangle - \|x\|(1 - \frac{1}{\|x\|}) \tag{2.139}$$

$$\leq \|c\|\|x\|(1 - \frac{1}{\|x\|}) - \|x\|(1 - \frac{1}{\|x\|}) \tag{2.140}$$

by the Cauchy-Schwarz inequality. So this

$$= (\|x\| - 1)(\|c\| - 1) \leq 0, \tag{2.141}$$

as $\|x\| - 1 > 0$ and $\|c\| - 1 \leq 0$. □

Example 2.47 (The Product Space)**.** We show that the projection onto the product set, $C := C_1 \times C_2 \times \ldots \times C_n$ is:

$$P_C(x_1, x_2, \ldots, x_n) = (P_{C_1}(x_1), P_{C_2}(x_2), \ldots, P_{C_n}(x_n)). \tag{2.142}$$

Proof. Clearly, $P_{C_i}(x_i) \in C_i$. Then $(P_{C_1}(x_1), P_{C_2}(x_2), \ldots, P_{C_n}(x_n)) \in C_1 \times C_2 \times \cdots \times C_n = C$. Now $\forall c \in C$, let $c = (c_1, c_2, \ldots, c_n)$ with $c_i \in C_i$. We will show that

$$\langle c - ((P_{C_1}(x_1), \ldots, P_{C_n}(x_n)), (x_1, x_2, \ldots, x_n) - (P_{C_1}(x_1), \ldots, P_{C_n}(x_n)) \rangle \leq 0 \tag{2.143}$$

Now,

$$\langle (c_1, \ldots, c_n) - (P_{C_1}(x_1), \ldots, P_{C_n}(x_n)), (x_1, \ldots, x_n) - (P_{C_1}(x_1), \ldots, P_{C_n}(x_n)) \rangle \tag{2.144}$$

$$= \langle (c_1 - P_{C_1}(x_1), \ldots, c_n - P_{C_n}(x_n)), (x_1 - P_{C_1}(x_1), \ldots, x_n - P_{C_n}(x_n)) \rangle \tag{2.145}$$

$$= \langle c_1 - P_{C_1}(x_1), x_1 - P_{C_1}(x_1) \rangle + \ldots + \langle c_n - P_{C_n}(x_n), x_n - P_{C_n}(x_n) \rangle \leq 0, \tag{2.146}$$

because every term above is less than or equal to 0 by the Theorem 2.44. □

Example 2.48 (The Diagonal Set)**.** We show that the projection onto the Diagonal Set, $C := \{(x, x, \ldots, x) \in X^n\}$ is:

$$P_C(x_1, x_2, \ldots, x_n) = (\overline{x}, \overline{x}, \ldots, \overline{x}), \tag{2.147}$$

where $\overline{x} = \frac{1}{n} \sum x_i$.

Proof. C is a closed and convex set. Clearly, $(\overline{x}, \ldots, \overline{x}) \in C$. Let $c \in C$, say $c = (x, x, \ldots, x)$ for some $x \in X$. Then

$$\langle c - (\overline{x}, \ldots, \overline{x}), (x_1, x_2, \ldots, x_n) - (\overline{x}, \overline{x}, \ldots, \overline{x}) \rangle \tag{2.148}$$

$$= \langle (x, x, \ldots, x) - (\overline{x}, \ldots, \overline{x}), (x_1, x_2, \ldots, x_n) - (\overline{x}, \overline{x}, \ldots, \overline{x}) \rangle \tag{2.149}$$

$$= \langle (x - \overline{x}, \ldots, x - \overline{x}), (x_1 - \overline{x}, \ldots, x_n - \overline{x}) \rangle \tag{2.150}$$

$$= \langle x - \overline{x}, x_1 - \overline{x} \rangle + \ldots + \langle x - \overline{x}, x_n - \overline{x} \rangle \tag{2.151}$$

$$= \langle x - \overline{x}, (x_1 - \overline{x}) + \ldots + (x_n - \overline{x}) \rangle \tag{2.152}$$

$$= \langle x - \overline{x}, (x_1 + \ldots + x_n) - n\overline{x} \rangle \tag{2.153}$$

$$= \langle x - \overline{x}, n\overline{x} - n\overline{x} \rangle \tag{2.154}$$

$$= \langle x - \overline{x}, 0 \rangle \tag{2.155}$$

$$= 0 \tag{2.156}$$

$$\leq 0. \tag{2.157}$$

The conclusion follows from the Projection theorem. □

2.8 Properties of the Projector

Theorem 2.49 (Projections are firmly nonexpansive)**.** *Suppose C is a nonempty closed convex set in the Hilbert space X. Then the projection P_C onto C is firmly nonexpansive:*

$$\|P_C x - P_C y\|^2 \leq \langle x - y, P_C x - P_C y \rangle, \quad \forall x, y \in C \tag{2.158}$$

for all $x, y \in X$.

Proof. By (2.117), we have that

$$\begin{aligned}
\|P_C x - P_C y\|^2 &= \langle P_C x - P_C y, P_C x - P_C y\rangle \\
&\leq \langle P_C x - P_C y, P_C x - P_C y\rangle + \langle x - P_C x, P_C x - P_C y\rangle \qquad (2.159)\\
&= \langle x - P_C y, P_C x - P_C y\rangle \qquad (2.160)\\
&\leq \langle x - P_C y, P_C x - P_C y\rangle + \langle P_C y - y, P_C x - P_C y\rangle \qquad (2.161)\\
&= \langle x - y, P_C x - P_C y\rangle.
\end{aligned}$$

So P_C is firmly nonexpansive as desired. □

Corollary 2.50. *Suppose* C *is a nonempty closed convex set in the Hilbert space* X*. Then* P_C *is nonexpansive.*

Proof. P_C is firmly nonexpansive and therefore by Corollary 2.41 nonexpansive. □

Remark 2.51. Since P_C is nonexpansive, it is 1-Lipschitz and hence continuous.

Remark 2.52. Since projection operators are nonexpansive, the composition of projection operators is nonexpansive.

2.9 The Reflector

Definition 2.53. Let B be a subset of X. The reflector is defined as $R_B := 2P_B - \mathrm{Id}$.

Proposition 2.54. *Let* $B \subseteq X$ *be a nonempty closed convex set. Then the reflector* R_B *is nonexpansive.*

Proof. Recall Theorem 2.39 and Theorem 2.49, so $P_B = \dfrac{2P_B - \mathrm{Id} + \mathrm{Id}}{2} = \dfrac{R_B + \mathrm{Id}}{2}$ is firmly nonexpansive.

This means that R_B is nonexpansive if and only if P_B is firmly nonexpansive. The good news is that all projections are firmly nonexpansive by Theorem 2.49. □

Proposition 2.55. *The composition of reflectors is nonexpansive.*

Proof. We proved that the composition of nonexpansive operators is nonexpansive Theorem 2.37. Since each reflector is nonexpansive by Proposition 2.54, we are done. □

2.10 Miscellaneous

Theorem 2.56. *We claim if $x_n \to x$, then (x_n) is bounded.*

Proof. For $\varepsilon = 1$, $\exists N$ such that $\forall n \geq N$ we have

$$\|x_n - x\| < \varepsilon = 1, \tag{2.162}$$

by the definition of a limit. By the triangle inequality,

$$\|x_n\| - \|x\| \leq \|x_n - x\| \leq 1 \quad \forall n \geq N. \tag{2.163}$$

Thus,

$$\|x_n\| \leq \|x\| + 1 \quad \forall n \geq N. \tag{2.164}$$

Next we define M to be $\max\{\|x\| + 1, \|x_1\|, \|x_2\|, ..., \|x_{N-1}\|\}$. Thus

$$\|x_n\| \leq M, \quad \forall n. \tag{2.165}$$

□

Theorem 2.57. *We claim if (x_n) is bounded, and B is a nonempty closed convex set, then $(P_B(x_n))$ is also bounded.*

Proof. Fix y, since P_B is firmly nonexpansive by Theorem 2.49 and thus nonexpansive by Theorem 2.41 we have,

$$\|P_B(x_n) - P_B(y)\| \leq \|x_n - y\| \leq \|x_n\| + \|y\|. \tag{2.166}$$

Since $\|(x_n)\|$ is bounded we have,

$$\|x_n\| + \|y\| \leq M + \|y\| \tag{2.167}$$

for some $M > 0$. By the triangle inequality,

$$\|P_B(x_n)\| - \|P_B(y)\| \leq \|P_B(x_n) - P_B(y)\| \leq M + \|y\|. \tag{2.168}$$

Therefore, $\|PB(x_n)\| \leq \|P_B y\| + M + \|y\|$ and $(P_B(x_n))$ is bounded. □

Definition 2.58 (Weakly convergent sequence)**.** We say $x_n \rightharpoonup x \iff \langle x_n, z\rangle \longrightarrow \langle x, z\rangle \quad \forall z \in X$.

Definition 2.59 (Weak cluster point)**.** A vector z is a weak cluster point of a sequence (x_n) if there is a subsequence (x_{k_n}) of (x_n) such that $x_{k_n} \rightharpoonup z$.

Chapter 3

Projection Methods

In this chapter we show that Elser's Difference Map algorithm is equivalent to the Douglas-Rachford and Fienup Hybrid Input-Output (HIO) algorithms. Throughout this chapter A and B are nonempty fixed convex and closed sets in X a real Hilbert space, and P_A,P_B,R_A, and R_B are the associated projections and reflectors onto the sets A and B.

3.1 Elser's Difference Map

Definition 3.1. Elser's Difference Map [ERT07, page 419] is defined by

$$D(x) = x + \beta[P_A \circ g_B(x) - P_B \circ f_A(x)] \tag{3.1}$$

where

$$f_A(x) = P_A(x) - (P_A(x) - x)/\beta \tag{3.2}$$

and

$$g_B(x) = P_B(x) + (P_B(x) - x)/\beta, \tag{3.3}$$

where β is a real parameter that is not equal to zero.

Proposition 3.2. *Elser's Difference Map, for $\beta = 1$, can be written in the form*

$$D = P_A(2P_B - \mathrm{Id}) + (\mathrm{Id} - P_B). \tag{3.4}$$

This is the same as Fienup's HIO [Fie82, page 2763] and the Douglas-Rachford algorithm [LM79].

Proof. The Difference Map, when $\beta = 1$, becomes

$$D(x) = x + [P_A \circ g_B(x) - P_B \circ f_A(x)] \tag{3.5}$$

where

$$f_A(x) = P_A(x) - (P_A(x) - x) = x \tag{3.6}$$

and

$$g_B(x) = P_B(x) + (P_B(x) - x) = 2P_B(x) - x. \tag{3.7}$$

We now plug into (3.5) giving:

$$D(x) = x + P_A(2P_B(x) - x) - P_B(x). \tag{3.8}$$

Hence

$$D = P_A(2P_B - \mathrm{Id}) + (\mathrm{Id} - P_B), \tag{3.9}$$

as desired. □

Lemma 3.3. *If A is a closed subspace, in such a case P_A is linear by [Deu01, 5.13(1) page 79], we can write $D = P_A(2P_B - \mathrm{Id}) + (\mathrm{Id} - P_B)$ in a more symmetric fashion:*

$$D = P_A P_B + (\mathrm{Id} - P_A)(\mathrm{Id} - P_B). \tag{3.10}$$

Proof. Indeed,

$$\begin{aligned}
D &= P_A(2P_B - \mathrm{Id}) + (\mathrm{Id} - P_B) && (3.11)\\
&= 2P_A P_B - P_A + \mathrm{Id} - P_B \quad (2P_A P_B = P_A P_B + P_A P_B) && (3.12)\\
&= P_A P_B + P_A P_B - P_A + \mathrm{Id} - P_B && (3.13)\\
&= P_A P_B + P_A(P_B - \mathrm{Id}) + (\mathrm{Id} - P_B) && (3.14)\\
&= P_A P_B - P_A(\mathrm{Id} - P_B) + (\mathrm{Id} - P_B) && (3.15)\\
&= P_A P_B + (-P_A + \mathrm{Id})(\mathrm{Id} - P_B) && (3.16)\\
&= P_A P_B + (\mathrm{Id} - P_A)(\mathrm{Id} - P_B) && (3.17)
\end{aligned}$$

as desired. □

3.2 Douglas-Rachford and Fienup HIO

Proposition 3.4. *The Fienup's Hybrid Input-Output Algorithm and Douglas-Rachford Algorithm is described by*

$$x_{n+1} = \Big(P_A(2P_B - \mathrm{Id}) + (\mathrm{Id} - P_B)\Big)(x_n). \tag{3.18}$$

For brevity, we set

$$T = P_A(2P_B - \mathrm{Id}) + (\mathrm{Id} - P_B). \tag{3.19}$$

Then

$$T = \frac{1}{2}(R_A R_B + \mathrm{Id}), \tag{3.20}$$

where $R_A = 2P_A - \mathrm{Id}$ and $R_B = 2P_B - \mathrm{Id}$.

Proof. Indeed,

$$T = P_A(2P_B - \mathrm{Id}) + (\mathrm{Id} - P_B) \tag{3.21}$$
$$= (P_A R_B + \mathrm{Id} - P_B) \quad (2P_B - \mathrm{Id}\,\text{replaced with } R_B) \tag{3.22}$$
$$= \frac{1}{2}(2P_A R_B + 2\,\mathrm{Id} - 2P_B) \quad (\text{factor out } \frac{1}{2}) \tag{3.23}$$
$$= \frac{1}{2}(2P_A R_B + \mathrm{Id} + \mathrm{Id} - 2P_B) \quad (\text{write } 2\,\mathrm{Id}\,\text{as } \mathrm{Id} + \mathrm{Id} = 2\,\mathrm{Id}) \tag{3.24}$$
$$= \frac{1}{2}(2P_A R_B + \mathrm{Id} - R_B) \quad (2P_B - \mathrm{Id}\,\text{replaced with } R_B) \tag{3.25}$$
$$= \frac{1}{2}(2P_A R_B - R_B + \mathrm{Id}) \quad (\text{rearrange}) \tag{3.26}$$
$$= \frac{1}{2}((2P_A - \mathrm{Id})R_B + \mathrm{Id}) \quad (\text{factor out } R_B) \tag{3.27}$$
$$= \frac{1}{2}(R_A R_B + \mathrm{Id}) \tag{3.28}$$

as desired. □

3.3 Fixed Point Results

Theorem 3.5. *Set* $T = P_A(2P_B - \mathrm{Id}) + (\mathrm{Id} - P_B)$. *Then*

$$P_B(\mathrm{Fix}\,T) = A \cap B \subseteq \mathrm{Fix}\,T \tag{3.29}$$

Proof. Fix $x \in X$, and write as $x = b + q$, where $b = P_B(x)$ and hence $q = x - b$. Then

$$x = T(x) \quad \text{definition of a fixed point} \tag{3.30}$$
$$\iff x = P_A(2P_B - \mathrm{Id})(x) + (\mathrm{Id} - P_B)(x) \tag{3.31}$$
$$\iff b + q = P_A(2b - (b + q)) + b + q - b \quad \text{substitution} \tag{3.32}$$
$$\iff b + q = P_A(2b - b - q) + q \quad \text{simplify} \tag{3.33}$$
$$\iff b = P_A(b - q) \quad \text{recall } b = P_B(x) \tag{3.34}$$
$$\iff P_B(x) = P_A(P_B(x) - (x - P_B(x))) \quad \text{substitution} \tag{3.35}$$
$$\iff P_B(x) = P_A(P_B(x) - x + P_B(x)) \quad \text{simplify} \tag{3.36}$$
$$\iff P_B(x) = P_A(2P_B(x) - x) \quad \text{simplify} \tag{3.37}$$
$$\iff P_B(x) = P_A(R_B(x)). \tag{3.38}$$

All together this means that $x = T(x) \iff P_B(x) = P_A(R_B(x))$, $P_B(x) \in B$ in which case $P_B(x) \in A$. So when $x \in \mathrm{Fix}\,T$, we have $P_B(x) \in B \cap A$, and

since $x = T(x)$ and x was an arbitrary fixed point we have

$$\boxed{P_B(\operatorname{Fix} T) \subseteq A \cap B.} \tag{3.39}$$

We now prove that $A \cap B \subseteq \operatorname{Fix} T$. $\forall x \in A \cap B$ we have $P_A(x) = x$ and $P_B(x) = x$ so

$$\begin{aligned} T(x) &= P_A(2P_B - \mathrm{Id})(x) + (\mathrm{Id} - P_B)(x) \quad \text{expand} && (3.40)\\ &= P_A(2P_B(x) - x) + (x - P_B(x)) && (3.41)\\ &= P_A(2x - x) + (x - x) && (3.42)\\ &= P_A(x) && (3.43)\\ &= x. && (3.44) \end{aligned}$$

Hence

$$\boxed{A \cap B \subseteq \operatorname{Fix} T.} \tag{3.45}$$

Now apply P_B to (3.45) to deduce that

$$A \cap B = P_B(A \cap B) \subseteq P_B(\operatorname{Fix} T). \tag{3.46}$$

Combining (3.39) and (3.46), we see that

$$\boxed{P_B(\operatorname{Fix} T) = A \cap B.} \tag{3.47}$$

□

3.4 The Main Result

We next present the main convergence result. We show what happens in a finite dimensional Hilbert space and then give an example about why the proof fails when we try to extend to infinite dimensions.

Throughout the remainder of this chapter, we set (see Proposition 3.4)

$$\boxed{T = \frac{1}{2}(R_A R_B + \mathrm{Id}) = P_A(2P_B - \mathrm{Id}) + (\mathrm{Id} - P_B).} \tag{3.48}$$

Fact 3.6 (Opial)**.** [Opi67] *Suppose that T is a firmly nonexpansive mapping from X to X with $\operatorname{Fix} T \neq \varnothing$. Then for every $x \in X$, the sequence $(T^n x)$ weakly converges to some point in Fix T.*

Theorem 3.7. *Suppose that $A \cap B \neq \varnothing$. Then T is firmly nonexpansive and the sequence (x_n) generated by*

$$x_{n+1} = \frac{1}{2}(R_A R_B + \mathrm{Id})(x_n) = T(x_n) \tag{3.49}$$

converges weakly to some point $x \in \mathrm{Fix}\, T$, and $P_B(x) \in A \cap B$. Moreover, the sequence $(P_B(x_n))$ is bounded and every weak cluster point of $(P_B(x_n))$ lies in $A \cap B$.

Proof. Fix T contains $A \cap B \neq \varnothing$ by Theorem 3.5. The sequence

$$(x_n) = (T^n(x_0)) \tag{3.50}$$

converges weakly to some fixed point x of T by Fact 3.6. Since (x_n) is bounded, it follows $(P_B(x_n))$ is bounded by Theorem 2.57. $R_A R_B$ is nonexpansive by Proposition 2.54 and Theorem 2.37. This implies that $T = \frac{\mathrm{Id} + R_A R_B}{2}$ is firmly nonexpansive by Theorem 2.39. Thus,

$$\|T(x) - T(y)\|^2 + \|(\mathrm{Id} - T)(x) - (\mathrm{Id} - T)(y)\|^2 \leq \|x - y\|^2 \quad \forall x, y \in X. \tag{3.51}$$

Pick $x = x_n$ and $y = x$ a fixed point. Then we get

$$\|x_n - x\|^2 \geq \|T(x_n) - T(x)\|^2 + \|x_n - T(x_n) - x + T(x)\|^2 \tag{3.52}$$

$$= \|x_{n+1} - x\|^2 + \|x_n - x_{n+1} - x + x\|^2 \tag{3.53}$$

$$= \|x_{n+1} - x\|^2 + \|x_n - x_{n+1}\|^2 \tag{3.54}$$

So

$$\sum_{n=1}^{m} (\|x_n - x\|^2 - \|x_{n+1} - x\|^2) \geq \sum_{n=1}^{m} \|x_n - x_{n+1}\|^2. \tag{3.55}$$

The summation on the left is

$$\|x_1 - x\|^2 - \|x_{m+1} - x\|^2, \tag{3.56}$$

which is less than or equal to $\|x_1 - x\|^2 < +\infty$. Hence

$$\forall m \quad \sum_{n=1}^{m} \|x_n - x_{n+1}\|^2 \leq \|x_1 - x\|^2 < +\infty \tag{3.57}$$

since $\|x_1 - x\|^2$ is a constant we have

$$\sum_{n=1}^{\infty} \|x_n - x_{n+1}\|^2 \leq \|x_1 - x\|^2 < +\infty. \tag{3.58}$$

Hence, $\|x_n - x_{n+1}\|^2 \to 0$, thus, $\|x_n - x_{n+1}\| \to 0$ i.e.,

$$x_n - x_{n+1} \to 0. \tag{3.59}$$

It follows that

$$\begin{aligned} x_n - x_{n+1} &= x_n - P_A(2P_B - \mathrm{Id})x_n - (\mathrm{Id} - P_B)x_n && (3.60)\\ &= P_B x_n - P_A(2P_B - \mathrm{Id})x_n \to 0. && (3.61) \end{aligned}$$

In infinite dimensions, it is a fact that if $(P_B x_n)_{n\in\mathbb{N}}$ is bounded then it has weak cluster points by [Deu01, page 205, 9.12] and Definition 2.59, say

$$P_B x_{k_n} \rightharpoonup z. \tag{3.62}$$

Now $(P_B x_n)_{n\in\mathbb{N}}$ lies in B and B is weakly closed by [Rud91, page 66, Theorem 3.12 corollaries (a)] (B is weakly closed since B is closed and convex) so its weak limit $z \in B$. We have

$$P_B x_n = P_B x_n - P_A(2P_B - \mathrm{Id})x_n + P_A(2P_B - \mathrm{Id})x_n, \tag{3.63}$$

recall that $P_B x_n - P_A(2P_B - \mathrm{Id})x_n \to 0$. Thus

$$P_A(2P_B - \mathrm{Id})x_{k_n} \rightharpoonup z. \tag{3.64}$$

Since A is also weakly closed we have that $z \in A$. Altogether we have $z \in A \cap B$. □

The statement of Theorem 3.7 is even simpler in finite dimensions.

Theorem 3.8. *Suppose that X is finite dimensional and $A \cap B \neq \varnothing$. Then the sequence (x_n) generated by*

$$x_{n+1} = Tx_n, \quad \text{where } T := \frac{1}{2}(R_A R_B + \mathrm{Id}) \tag{3.65}$$

converges to some point $x \in \operatorname{Fix} T$, and $P_B(x) \in A \cap B$. Moreover,

$$P_B(x_n) \to P_B(x) \in A \cap B. \tag{3.66}$$

Proof. Note that T is a firmly nonexpansive operator, because it is a projection by Theorem 2.49. So

$$x_n \to x = Tx \Longrightarrow P_B x_n \to P_B x \tag{3.67}$$

as P_B is continuous. We know $P_B x \in P_B(\operatorname{Fix} T) = A \cap B$ by Theorem 3.5. □

Remark 3.9. The proof of the Theorem 3.8 does not work in infinite dimensions because P_B may fail to be weakly continuous.

Zarantonello [Zar71, page 245], showed that if

$$x_n \rightharpoonup x \not\Rightarrow P_B x_n \rightharpoonup P_B x. \tag{3.68}$$

Let B be the unit-ball in ℓ_2. Then let

$$x_n = e_1 + e_n \rightharpoonup e_1 = P_B e_1 \text{ as } e_1 \in B. \tag{3.69}$$

It is a fact that if we are outside the unit ball the closed-form projection is $\frac{x}{\|x\|}$ by Example 2.46. So if $n \geq 2$, then

$$P_B x_n = \frac{e_1 + e_n}{\|e_1 + e_n\|} = \frac{e_1 + e_n}{\sqrt{2}} \rightharpoonup \frac{e_1}{\sqrt{2}}. \tag{3.70}$$

So on one hand we have

$$x_n \rightharpoonup e_1, \tag{3.71}$$

but on the other we have

$$P_B x_n \rightharpoonup \frac{e_1}{\sqrt{2}} \neq e_1 = P_B e_1. \tag{3.72}$$

Altogether, P_B is not weakly continuous.

Chapter 4

Monotone Operators

Throughout this chapter, X is a finite-dimensional real Hilbert space with inner product $\langle \cdot, \cdot \rangle$ and induced norm $\| \cdot \|$.

$\boxed{C \text{ and } D \text{ are two closed convex sets in } X \text{ such that } C \cap D \neq \varnothing.}$

Recall that if $A\colon X \to X$ is linear, then A is automatically continuous (see, e.g., [Kre78, Page 94]).

4.1 Definitions

We first introduce some fundamental definitions. The arrow "$\to$" is used for a single-valued mapping, whereas "$\rightrightarrows$" denotes a set-valued mapping, i.e. $X \to \mathcal{P}(X)$, the power set of X.

Definition 4.1. Let $T\colon X \rightrightarrows X$. We say T is *monotone* if

$$\langle x^* - y^*, x - y \rangle \geq 0, \tag{4.1}$$

whenever $x^* \in T(x), y^* \in T(y)$.

Proposition 4.2. *Let $A\colon X \to X$ be linear. Then A is monotone, if and only if,*

$$\langle x, Ax \rangle \geq 0 \quad \forall x \in X. \tag{4.2}$$

Proof. "$\Rightarrow$" For every $x \in X$, by monotonicity and linearity of A we have

$$\langle Ax, x \rangle = \langle Ax - A0, x - 0 \rangle \geq 0. \tag{4.3}$$

(4.3) holds by $A0 = 0$ (since A is linear).

"$\Leftarrow$" For every $x, y \in X$, since $\langle x, Ax \rangle \geq 0$, we have

$$\big\langle Ax - Ay, x - y \big\rangle = \big\langle A(x - y), x - y \big\rangle \geq 0. \tag{4.4}$$

□

Definition 4.3 (Adjoint operator)**.** Let $A : X \to X$ be continuous and linear. We say A^* is the adjoint operator of A if

$$\langle x, Ay \rangle = \langle A^* x, y \rangle \quad \forall x, y \in X. \tag{4.5}$$

Remark 4.4. We know A^* exists and is unique, see [Kre78, Theorem 3.9-2].

Remark 4.5. We know $A^{**} = A$, see [Deu01, Lemma 8.29 (3)].

Remark 4.6. If $A \in \mathbb{R}^{n \times n}$ then $A^* = A^T$, see [Mey00, Page 84].

Definition 4.7 (Symmetric)**.** Let $A : X \to X$ be linear. We say that A is *symmetric* if $A^* = A$.

Definition 4.8 (Symmetric Part)**.** Let $A : X \to X$ be linear. We say A_+ is the symmetric part of A and is defined as follows:

$$A_+ = \frac{A + A^*}{2}. \tag{4.6}$$

Proposition 4.9. *Let $A : X \to X$ be linear. Then A_+ is symmetric.*

Proof. $A_+^* = (\frac{A+A^*}{2})^* = \frac{A^*+A^{**}}{2} = \frac{A^*+A}{2} = A_+$. □

Definition 4.10 (Antisymmetric)**.** Let $A : X \to X$ be linear. We say that A is *antisymmetric* if $A^* = -A$.

Definition 4.11 (Antisymmetric Part)**.** Let $A : X \to X$ be linear. We say A_0 is the antisymmetric part of A and is defined as follows:

$$A_0 = \frac{A - A^*}{2}. \tag{4.7}$$

Proposition 4.12. *Let $A : X \to X$ be linear. Then A_0 is antisymmetric.*

Proof. $A_0^* = (\frac{A-A^*}{2})^* = \frac{A^*-A^{**}}{2} = \frac{A^*-A}{2} = -(\frac{A-A^*}{2}) = -A_0$. □

Lemma 4.13. *Let $A: X \to X$ be linear. Then A is monotone $\Leftrightarrow$ A^* is monotone.*

Proof. By Proposition 4.2 we have that,

$$\begin{aligned} A \text{ is monotone} &\iff \langle x, Ax \rangle \geq 0 \quad \forall x && (4.8)\\ &\iff \langle Ax, x \rangle \geq 0 \quad \forall x && (4.9)\\ &\iff \langle x, A^* x \rangle \geq 0 \quad \forall x && (4.10)\\ &\iff A^* \text{ is monotone.} && (4.11) \end{aligned}$$

□

Definition 4.14. Let $A\colon X \to X$ be linear. We set

$$\boxed{q_A\colon X \to \mathbb{R}\colon x \mapsto \frac{1}{2}\langle x, Ax\rangle.} \tag{4.12}$$

Proposition 4.15. *Let $A\colon X \to X$ be linear. Then $\nabla q_A = A_+$.*

Proof. For $x \in X$ and $h \in X\backslash\{0\}$, we have,

$$\frac{q_A(x+h) - q_A(x) - \langle A_+x, h\rangle}{\|h\|} \tag{4.13}$$

$$= \frac{q_A(x) + q_A(h) + \frac{1}{2}\langle x, Ah\rangle + \frac{1}{2}\langle Ax, h\rangle - q_A(x) - \langle A_+x, h\rangle}{\|h\|} \tag{4.14}$$

$$= \frac{q_A(h) + \frac{1}{2}\langle A^*x, h\rangle + \frac{1}{2}\langle Ax, h\rangle - \langle A_+x, h\rangle}{\|h\|} \tag{4.15}$$

$$= \frac{q_A(h) + \langle A_+x, h\rangle - \langle A_+x, h\rangle}{\|h\|} \tag{4.16}$$

$$= \frac{q_A(h)}{\|h\|} \tag{4.17}$$

$$= \frac{\frac{1}{2}\langle h, Ah\rangle}{\|h\|}. \tag{4.18}$$

Since A is continuous, we have, for $M = \|A\|$,

$$\|Ah\| \leq M\|h\|, \quad \forall h \in X. \tag{4.19}$$

Then, by Cauchy-Schwarz we get

$$0 \leq \left|\frac{q_A(x+h) - q_A(x) - \langle A_+x, h\rangle}{\|h\|}\right| \tag{4.20}$$

$$= \left|\frac{\frac{1}{2}\langle h, Ah\rangle}{\|h\|}\right| \tag{4.21}$$

$$\leq \frac{\frac{1}{2}\|h\| \cdot M \cdot \|h\|}{\|h\|} = \frac{1}{2}M\|h\| \to 0 \quad \text{as } \|h\| \to 0^+. \tag{4.22}$$

By the Squeeze theorem we get,

$$\lim_{\|h\|\to 0^+} \frac{q_A(x+h) - q_A(x) - \langle A_+x, h\rangle}{\|h\|} = 0 \tag{4.23}$$

i.e., $\nabla q_A = A_+$. So we have proven that q_A is Fréchet differentiable at x and $\nabla q_A = A_+$. □

Proposition 4.16. *Let $A\colon X \to X$ be linear. Then A is monotone if and only if $q_A \geq 0$.*

Proof. See Proposition 4.2. □

Proposition 4.17. *Let $A : X \to X$ be linear. Then $q_A = q_{A_+}$.*

Proof. Let $x \in X$. Then,

$$\begin{aligned} 2q_{A_+}(x) &= \langle A_+x, x\rangle = \langle \tfrac{A^*+A}{2}x, x\rangle \\ &= \langle \tfrac{A^*x}{2}, x\rangle + \langle \tfrac{Ax}{2}, x\rangle = \langle \tfrac{x}{2}, Ax\rangle + \langle \tfrac{Ax}{2}, x\rangle \\ &= \langle Ax, x\rangle = 2q_A(x). \end{aligned}$$

□

Proposition 4.18. *Let $A : X \to X$ be linear. Then A is monotone if and only if A_+ is monotone.*

Proof. By Proposition 4.2 we have that

$$\begin{aligned} A \text{ is monotone } &\iff q_A \geq 0 \quad \text{by Proposition 4.16} && (4.24)\\ &\iff q_{A_+} \geq 0 \quad \text{by Proposition 4.17} && (4.25)\\ &\iff A_+ \text{ is monotone by Proposition 4.16 .} && (4.26) \end{aligned}$$

□

Definition 4.19. Let $T\colon X \rightrightarrows X$ be monotone. We call T *maximal monotone* if for every $(y, y^*) \notin \operatorname{gra} T$ there exists $(x, x^*) \in \operatorname{gra} T$ with $\langle x - y,\ x^* - y^*\rangle < 0$.

Fact 4.20. *Let $A\colon X \rightrightarrows X$ be maximal monotone and $(x_0, x_0^*) \in X \times X$. Let $\widetilde{A}\colon X \rightrightarrows X$ such that $\operatorname{gra}\widetilde{A} = \operatorname{gra} A - (x_0, x_0^*)$ (i.e., a rigid translation of $\operatorname{gra} A$). Then $\widetilde{A}$ is maximal monotone.*

Proof. Follows directly from Definition 4.19. □

Lemma 4.21. *Every continuous monotone operator $A\colon X \to X$ is maximal monotone.*

Proof. [RW98, Example 12.7]. □

Definition 4.22 (Subdifferential)**.** Let $f\colon X \to]-\infty,+\infty]$ and let $x \in X$ be such $f(x) < +\infty$. Then the *subdifferential* of f at x is

$$\partial f(x) = \{s \in X \mid f(y) \geq f(x) + \langle s, y - x\rangle, \quad \forall y \in X\}. \tag{4.27}$$

The elements in $\partial f(x)$ are called *subgradients* of f at x.

See [Roc97, Part V] for basic properties of the subdifferential operator.

Fact 4.23. *$\partial f(x) = \nabla f(x)$ if f is Gateaux differentiable, see [Zăl02, Theorem 2.4.4(i)].*

Example 4.24. Let $X = \mathbb{R}$ and suppose $f(x) = |x|$. Then,

$$\partial f(x) = \begin{cases} \{-1\} & \text{if } x < 0; \\ [-1,+1] & \text{if } x = 0; \\ \{1\} & \text{if } x > 0. \end{cases} \tag{4.28}$$

Proof. Let $x < 0$ and $s \in \partial f(x)$. We have

$$\langle s, y - x\rangle \leq f(y) - f(x) = |y| - |x|. \tag{4.29}$$

Let $z \in X$ and $y = x + tz$, where $t > 0$. By (4.29),

$$\langle s, tz\rangle \leq |x + tz| - |x| = |x + tz| - (-x). \tag{4.30}$$

Since $x < 0$, $x + tz < 0$ when t is small. Thus by (4.30),

$$\langle s, tz\rangle \leq -(x + tz) - (-x) = -tz \tag{4.31}$$

so

$$\langle s, z\rangle \leq -z. \tag{4.32}$$

So plug in $z = \pm 1$ and we get that $s = -1$. This shows that when $x < 0$ the subdifferential is the set $\{-1\}$. Now let $x = 0$ and $s \in \partial f(x) = \partial f(0)$. We have

$$\begin{aligned}
\langle s, y - 0\rangle \leq f(y) - f(0) = |y| &\iff \langle s, y\rangle \leq |y| \quad \forall y && (4.33)\\
&\iff \langle s, y\rangle \leq |y| \quad \forall y \neq 0 && (4.34)\\
&\iff \langle s, \frac{y}{|y|}\rangle \leq 1 \quad \forall y \neq 0 && (4.35)\\
&\iff \max\{\langle s, 1\rangle, \langle s, -1\rangle\} \leq 1 && (4.36)\\
&\iff \max\{s, -s\rangle\} \leq 1 && (4.37)\\
&\iff |s| \leq 1 && (4.38)\\
&\iff s \in [-1, 1]. && (4.39)
\end{aligned}$$

This shows that when $x = 0$ the subdifferential is the set $[-1, 1]$.

Finally, let $x > 0$ and $s \in \partial f(x)$. We have

$$\langle s, y - x\rangle \leq f(y) - f(x) = |y| - |x|. \tag{4.40}$$

Let $z \in X$ and $y = x + tz$, where $t > 0$. By (4.40),

$$\langle s, tz\rangle \leq |x + tz| - |x| = |x + tz| - x. \tag{4.41}$$

Since $x > 0$, $x + tz > 0$ when t is small. Thus by (4.41),

$$\langle s, tz\rangle \leq (x + tz) - x = tz \tag{4.42}$$

so

$$\langle s, z\rangle \leq z. \tag{4.43}$$

So plug in $z = \pm 1$ and we get that $s = 1$. This shows that when $x > 0$ the subdifferential is the set $\{1\}$.

□

Fact 4.25 (Rockafellar)**.** *Let $f : X \to]-\infty, +\infty]$ be proper lower semicontinuous and convex. Then ∂f is maximal monotone.*

Proof. See [Sim98, Page 113] or [Roc70]. □

Definition 4.26 (Positive-semidefinite matrix)**.** The matrix $A \in \mathbb{R}^{n\times n}$ is positive semidefinitite, written $A \succeq 0$, if A is symmetric, i.e. $A = A^T$ and

$$\langle x, Ax\rangle \geq 0 \quad \forall x. \tag{4.44}$$

Fact 4.27. *In the 2×2 case it can be checked that*

$$\begin{pmatrix} a & b \\ b & c \end{pmatrix} \succeq 0 \iff a \geq 0, c \geq 0, ac - b^2 \geq 0, \tag{4.45}$$

see [Mey00, Page 566].

Definition 4.28 (Resolvent)**.** Let M be maximal monotone and $J_M = (\mathrm{Id} + M)^{-1}$. Then J_M is called the resolvent of M.

Fact 4.29 (Minty)**.** *Let $T : X \to X$. Then*

$$T \text{is firmly nonexpansive} \iff T = (\mathrm{Id} + M)^{-1}, \text{where } M \text{is maximal monotone,} \tag{4.46}$$

see [EB92, Theorem 2].

The above fact can also be found in [Min62]. We can see from the above that we have another way to characterize firmly nonexpansive mappings.

Definition 4.30 (Indicator Function)**.** Let $S \subseteq X$, then the indicator function is defined by

$$x \mapsto \iota_S(x) = \begin{cases} 0, & \text{if } x \in S; \\ \infty, & \text{otherwise.} \end{cases}$$

Definition 4.31 (Proximal Mapping)**.** Let $T : X \to X$ be firmly nonexpansive, so that $T = (\mathrm{Id} + M)^{-1}$, where M is maximal monotone, see Fact 4.29. If $M = \partial f$ for some function f that is convex, lower semicontinuous and proper, then T is called a proximal mapping.

Remark 4.32 (Projection). Let $T : X \to X$ be firmly nonexpansive, say $T = (\mathrm{Id} + M)^{-1}$, where M is maximal monotone, see Fact 4.29. If $M = \partial \iota_C$, the subdifferential of the indicator function, then $T = P_C$.

Projection mappings are a subset of proximal mappings and proximal mappings are subsets of firmly nonexpansive mappings.

Lemma 4.33. *Every proximal mapping is a subdifferential operator.*

Proof. Let T be a proximal mapping. Then by definition, the sum rule [Zăl02, Theorem 2.8.7(iii)], and [Roc97, Theorem 23.8],

$$\begin{aligned} T &= (\mathrm{Id} + \partial f)^{-1} && (4.47)\\ &= (\nabla \frac{1}{2}\|\cdot\|^2 + \partial f)^{-1} && (4.48)\\ &= (\partial(\frac{1}{2}\|\cdot\|^2 + f))^{-1} && (4.49)\\ &= \partial(\frac{1}{2}\|\cdot\|^2 + f)^*, && (4.50) \end{aligned}$$

where $g^*(x) = \sup_{y \in X}(\langle x, y\rangle - g(y))$ is the Fenchel conjugate of g, see [Roc97, Chapter 12] for basic properties. This clearly shows that the proximal mapping is a subdifferential. □

Lemma 4.34. *The projection operator P_C is the gradient of $\frac{1}{2}\|\cdot\| - \frac{1}{2}d_C^2$.*

Proof. We use [RW98, Theorem 2.26] throughout this proof. If we set f to be the indicator function and $\lambda = 1$, then by [RW98, Definition 1.22] we get

when $x \in X$,

$$e_1\iota_C(x) = \inf_{\omega\in X} \iota_g(\omega) + \frac{1}{2}\|\omega - x\|^2 \tag{4.51}$$

$$= \inf_{\omega\in C} \frac{1}{2}\|\omega - x\|^2 \tag{4.52}$$

$$= \frac{1}{2}d_C^2(x), \tag{4.53}$$

where $e_1 g(x) = \inf_{\omega\in X} g(\omega) + \frac{1}{2}\|\omega - x\|^2$ is the Moreau envelope of g and $P_1 g(x)$ is the corresponding set of minimizers. Again by [RW98, Definition 1.22] we get,

$$P_1\iota_C(x) = \operatorname*{argmin}_{\omega\in X} \iota_C + \frac{1}{2}\|\omega - x\|^2 \tag{4.54}$$

$$= \operatorname*{argmin}_{\omega\in C} \frac{1}{2}\|\omega - x\|^2 \tag{4.55}$$

$$= \operatorname*{argmin}_{\omega\in C} \frac{1}{2}d_C^2(x) \tag{4.56}$$

$$= P_C(x). \tag{4.57}$$

Now by [RW98, Theorem 2.26b] we have,

$$\nabla e_1\iota_C = \mathrm{Id} - P_C. \tag{4.58}$$

By rearranging, then using (4.53) we have

$$P_C = \mathrm{Id} - \nabla e_1\iota_C \tag{4.59}$$

$$= \mathrm{Id} - \nabla(\frac{1}{2}d_C^2) \tag{4.60}$$

$$= \nabla(\frac{1}{2}\|\cdot\|^2) - \nabla(\frac{1}{2}d_C^2) \tag{4.61}$$

$$= \nabla(\frac{1}{2}\|\cdot\|^2 - \frac{1}{2}d_C^2), \tag{4.62}$$

which clearly shows that P_C is in fact a gradient as desired. □

Proposition 4.35. *Let $B : X \to X$ be linear. Then $B = \partial g \iff B = B^*$, in which case $B = \nabla q_B$ and $g = q_B + c$, where $c \in \mathbb{R}$.*

Proof. "⇐": $B = B^* \implies B = B_+$ now by Proposition 4.15 $\implies B = B_+ = \nabla q_B$. "⇒": $B = \nabla g \implies \nabla^2 g = B$ is symmetric [MN88, Theorem 4 Page 120]. □

Lemma 4.36. *Let T be linear and a proximal mapping. Then $T = T^*$.*

Proof. By assumption, there exists a lower semicontinuous convex function f such that

$$T = (\mathrm{Id} + \partial f)^{-1}. \tag{4.63}$$

By Proposition 4.35, and the fact that Id is symmetric,

$$\mathrm{Id} = \nabla q_{\mathrm{Id}}. \tag{4.64}$$

By (4.63),

$$T = (\nabla q_{\mathrm{Id}} + \partial f)^{-1} = (\partial(f + q_{\mathrm{Id}}))^{-1} \tag{4.65}$$

by [Roc97, Theorem 23.8]. Let

$$g = (f + q_{\mathrm{Id}})^* \tag{4.66}$$

then

$$\partial g = (\partial(f + q_{\mathrm{Id}}))^{-1} \tag{4.67}$$

by [Roc97, Corollary 23.5.2]. By (4.65), we have

$$T = \partial g. \tag{4.68}$$

Then by Proposition 4.35, $T = T^*$. □

Below is an alternative proof provided by Liangjin Yao. We first note that if

$$T = (\mathrm{Id} + \partial f)^{-1} \tag{4.69}$$

then

$$T^{-1} = \mathrm{Id} + \partial f \tag{4.70}$$

and

$$T^{-1} - \mathrm{Id} = \partial f. \tag{4.71}$$

Since T^{-1} and Id are linear relations, so is $T^{-1} - \mathrm{Id}$ and hence $\mathrm{gra}(T^{-1} - \mathrm{Id})$ is a linear subspace. Then

$$(\partial f)^* = \partial f \tag{4.72}$$

by [BWY09, Lemma 4.1]. Now

$$T = (\mathrm{Id} + \partial f)^{-1} \tag{4.73}$$

and

$$\begin{aligned} T^* &= [(\mathrm{Id}+\partial f)^{-1}]^* && (4.74)\\ &= [(\mathrm{Id}+\partial f)^*]^{-1} \quad \text{by [Cro98, Proposition III 1.3b]} && (4.75)\\ &= [(\mathrm{Id})^* + (\partial f)^*]^{-1} \quad \text{by [Bor83, Theorem 7.4]} && (4.76)\\ &= (\mathrm{Id}+\partial f)^{-1} \quad \text{by (4.72)} && (4.77)\\ &= T. && (4.78) \end{aligned}$$

Lemma 4.37. *Let* $P = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$ *such that* P *is a proximal mapping. Then* $b = c$.

Proof. By Lemma 4.36, $P = P^*$, hence $b = c$. □

Lemma 4.38. *(See also [RW98, Example 12.7].) Let* $A : X \to X$ *be continuous and monotone. Then* A *is maximal monotone.*

Proof. Let $(x, x^*) \in X \times X$ be such that

$$\langle x - y, x^* - Ay\rangle \geq 0, \quad \forall y \in X. \tag{4.79}$$

It suffices to show that $x^* = Ax$. Let $z \in X$ and $t > 0$. Let $y = x + tz \in X$. Then

$$\begin{aligned} &\langle x - (x+tz), x^* - A(x+tz)\rangle \geq 0 && (4.80)\\ \iff &\langle -tz, x^* - A(x+tz)\rangle \geq 0 && (4.81)\\ \iff &\langle A(x+tz) - x^*, tz\rangle \geq 0 \quad \text{divide both sides by } t && (4.82)\\ \iff &\langle A(x+tz) - x^*, z\rangle \geq 0. && (4.83) \end{aligned}$$

Let $t \to 0^+$. Since A is continuous,

$$\langle Ax - x^*, z\rangle \geq 0, \quad \forall z \in X. \tag{4.84}$$

Set $z = x^* - Ax$. Then (4.84) yields,

$$0 \geq \|Ax - x^*\|^2 \iff \langle Ax - x^*, x^* - Ax\rangle \geq 0. \tag{4.85}$$

So, $\|Ax - x^*\| = 0$, i.e., $x^* = Ax$. □

Corollary 4.39. *Let* $A : X \to X$ *be linear. Then* A *is maximal monotone if and only if* $\forall x \quad \langle x, Ax\rangle \geq 0$.

Proof. We have $\forall x \quad \langle x, Ax\rangle \geq 0 \iff A$ is monotone by Proposition 4.2. Now since X is finite-dimensional, this implies that A is continuous by [Kre78, Page 94]. Therefore, since A monotone and continuous, it is maximal monotone by Lemma 4.38. □

4.2 Eckstein's Observation

Recall that R_C and R_D are the reflectors of given subsets C and D.

Definition 4.40. Following Eckstein in his thesis, for any maximal monotone operators A and B, and $\lambda > 0$ we define

$$G_{\lambda,A,B} = J_{\lambda A}(2J_{\lambda B} - \mathrm{Id}) + (\mathrm{Id} - J_{\lambda B}). \tag{4.86}$$

Remark 4.41. It is important to note that $J_{\lambda A} = (\mathrm{Id} + \lambda A)^{-1}$ is the resolvent of λA by Definition 4.28 as A is maximal monotone,

Fact 4.42. *(See [Eck89, Corollary 4.21].) Let $G^{-1}_{\lambda,A,B} = S_{\lambda,A,B} + \mathrm{Id}$, then $G_{\lambda,A,B}$ is firmly nonexpansive.*

Definition 4.43 (Splitting Operator)**.** (See [Eck89, Page 137].)

$$S_{\lambda,A,B} = G^{-1}_{\lambda,A,B} - \mathrm{Id}\,. \tag{4.87}$$

In [Eck89, Corollary 4.21] we see a fairly complicated proof of Fact 4.42. However, just like we proved that T is firmly nonexpansive in Theorem 3.7, it may be proved that $G_{\lambda,A,B}$ is firmly nonexpansive by replacing projections with appropriate resolvents. See also [Com04] and [EF98] for further information on the Douglas-Rachford and related methods involving (firmly) nonexpansive operators.

Eckstein was the first to observe the following [Eck89, Proposition 4.10].

Proposition 4.44. *There exist maximal monotone operators A and B on $\mathbb{R}^2$ and a constant $\lambda > 0$ such that A and B are both subdifferentials of convex functions, but $S_{\lambda,A,B}$ is not the subdifferential of any convex function.*

Proof. Let

$$A = \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix} \tag{4.88}$$

$$B = \begin{pmatrix} 2 & 0 \\ 0 & 1 \end{pmatrix} \tag{4.89}$$

$$\lambda = \frac{1}{3}. \tag{4.90}$$

By Fact 4.27, A and $B \succeq 0$. Then, by Proposition 4.35,

$$A = \partial f, \quad \text{where } f(x) = \frac{1}{2} x^T \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix} x \tag{4.91}$$

$$B = \partial g, \quad \text{where } g(x) = \frac{1}{2} x^T \begin{pmatrix} 2 & 0 \\ 0 & 1 \end{pmatrix} x. \tag{4.92}$$

By direct calculation,

$$J_{\lambda A} = (\mathrm{Id} + \lambda A)^{-1} = \begin{pmatrix} \frac{5}{8} & \frac{-1}{8} \\ \frac{-1}{8} & \frac{5}{8} \end{pmatrix} \tag{4.93}$$

$$J_{\lambda B} = (\mathrm{Id} + \lambda B)^{-1} = \begin{pmatrix} \frac{3}{5} & 0 \\ 0 & \frac{3}{4} \end{pmatrix}. \tag{4.94}$$

Now Eckstein writes

$$G_{\lambda,A,B} = J_{\lambda A}(2J_{\lambda B} - \mathrm{Id}) + (\mathrm{Id} - J_{\lambda B}) = \begin{pmatrix} \frac{1}{2} & \frac{1}{16} \\ \frac{1}{10} & \frac{11}{16} \end{pmatrix} \tag{4.95}$$

and

$$S_{\lambda,A,B} = G_{\lambda,A,B}^{-1} - \mathrm{Id} = \begin{pmatrix} \frac{28}{27} & \frac{-5}{27} \\ \frac{-8}{27} & \frac{13}{27} \end{pmatrix}. \tag{4.96}$$

Since $G_{\lambda,A,B}$ is not symmetric, it cannot be a proximal mapping: indeed, G is firmly nonexpansive by Fact 4.42 and if G were a proximal map, it would by symmetric by Lemma 4.37, but it is not. □

Eckstein made a numerical error in (4.95) and as a result (4.96). The matrices in fact are

$$G_{\lambda,A,B} = J_{\lambda A}(2J_{\lambda B} - \mathrm{Id}) + (\mathrm{Id} - J_{\lambda B}) = \begin{pmatrix} \frac{21}{40} & \frac{-1}{16} \\ \frac{-1}{40} & \frac{9}{16} \end{pmatrix} \tag{4.97}$$

and

$$S_{\lambda,A,B} = G_{\lambda,A,B}^{-1} - \mathrm{Id} = \begin{pmatrix} \frac{43}{47} & \frac{10}{47} \\ \frac{4}{47} & \frac{37}{47} \end{pmatrix}, \tag{4.98}$$

however, Eckstein's conclusion remains valid (see appendix for the calculation details). In Example 4.46, we provide an example which isn't prone to numerical errors as it is much simpler.

Lemma 4.45. *Let $T : X \to X$ be linear. If*

$$T = \frac{\mathrm{Id} + R_D R_C}{2} \tag{4.99}$$

then,

$$R_D R_C \text{ is symmetric} \iff T \text{ is symmetric.} \tag{4.100}$$

Proof.

$$\begin{aligned} R_D R_C \text{ is symmetric} &\iff (R_D R_C)^* = R_D R_C && (4.101)\\ &\iff \mathrm{Id} + (R_D R_C)^* = \mathrm{Id} + R_D R_C && (4.102)\\ &\iff (\mathrm{Id} + R_D R_C)^* = \mathrm{Id} + R_D R_C && (4.103)\\ &\iff \frac{(\mathrm{Id} + R_D R_C)^*}{2} = \frac{\mathrm{Id} + R_D R_C}{2} && (4.104)\\ &\iff T^* = T && (4.105) \end{aligned}$$

□

Example 4.46. Set

$$T = \frac{\mathrm{Id} + R_D R_C}{2}. \tag{4.106}$$

Then we have the implications if T is a proximal mapping then T is symmetric $\iff$ $R_D R_C$ is symmetric, by Lemma 4.36 and Lemma 4.45.

Consider,

$$C = \{(x, 0) | x \in \mathbb{R}\} \quad \text{the x-axis in } \mathbb{R}^2 \tag{4.107}$$

$$D = \{(x, x) | x \in \mathbb{R}\} \quad \text{the diagonal in } \mathbb{R}^2. \tag{4.108}$$

We have

$$P_C = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}, P_D = \begin{pmatrix} \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} \end{pmatrix}, \mathrm{Id} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}. \tag{4.109}$$

Now the reflectors onto set C and D are defined as follows:

$$R_C = 2P_C - \mathrm{Id} = 2\begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} - \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \tag{4.110}$$

$$R_D = 2P_D - \mathrm{Id} = 2\begin{pmatrix} \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} \end{pmatrix} - \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}. \tag{4.111}$$

Furthermore,

$$R_D R_C = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}\begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} \tag{4.112}$$

which is clearly not symmetric! Therefore, T is a firmly nonexpansive mapping that is not a proximal mapping even though it was constructed from two proximal mappings.

Now

$$T = \frac{\mathrm{Id} + R_D R_C}{2} = \frac{\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} + \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}}{2} = \begin{pmatrix} \frac{1}{2} & -\frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} \end{pmatrix} \tag{4.113}$$

$$T = \frac{\mathrm{Id} + R_D R_C}{2} = (\mathrm{Id} + M)^{-1} \tag{4.114}$$

where M is maximal monotone, and hence

$$M = T^{-1} - \mathrm{Id}\,. \tag{4.115}$$

We calculate T^{-1} directly from T,

$$T^{-1} = \begin{pmatrix} 1 & 1 \\ -1 & 1 \end{pmatrix}. \tag{4.116}$$

And finally M,

$$M = T^{-1} - \mathrm{Id} = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}. \tag{4.117}$$

This is clearly not symmetric and does not come from a subdifferential operator, by Proposition 4.35 applied to M.

Eckstein in his thesis pointed out that the T in Douglas-Rachford (aka Difference map or Lions-Mercier) is not necessarily a proximal map even if it is constructed from two proximal mappings. (However, his example is more complicated and has a numerical error, compared to the simpler example we just showed).

This shows that you truly need monotone operator theory to understand Douglas-Rachford or the Difference Map, even if you started with the projection operators since the resolvent is not a proximal mapping (the monotone operator part is not a subdifferential operator).

Chapter 5

8 Queens

The eight queens puzzle is the problem of putting eight chess queens on an 8×8 chessboard such that none of them is able to capture any other using the standard chess queen's moves. The queens must be placed in such a way that no two queens would be able to attack each other. Thus, a solution requires that no two queens share the same row, column, or diagonal. The eight queens puzzle is an example of the more general n queens puzzle of placing n queens on an $n \times n$ chessboard, where solutions exist only for $n = 1$ or $n \geq 4$. This is a new application of Elser's Difference Map. From a complexity point of view, the n queens puzzle is easy — in fact, it can be solved in linear time, see [AY89] for details. Our interest in it stems from a modeling perspective; it will make the modeling of Sudoku in the next chapter easier to understand.

5.1 Queen's Role in Chess

The queen can be moved any number of unoccupied squares in a straight line vertically, horizontally, or diagonally, thus combining the moves of the rook and bishop. The queen captures by occupying the square on which an opponent's piece sits, see Figure 5.1 (from [Que]).

5.2 8 Queens History

The puzzle was originally proposed in 1850 by Franz Nauck. Nauck also extended the puzzle to the n queens problem (on an $n \times n$ chess-board, how can n queens be placed) [RBC74, page 166]. In 1874, S. Günther proposed a method of finding solutions by using determinants, and J.W.L. Glaisher refined this approach [RBC74, page 166-167].

This puzzle even appeared in the popular early 1990s computer game The 7th Guest.

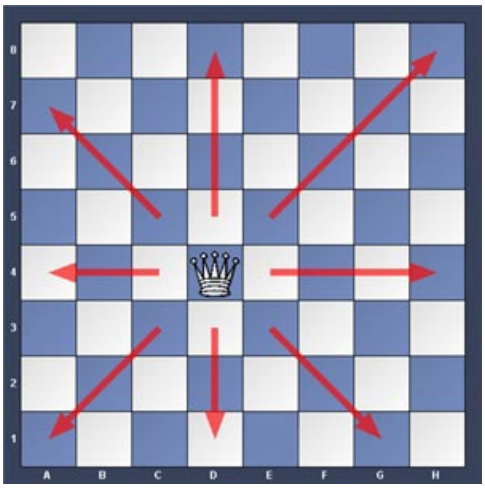
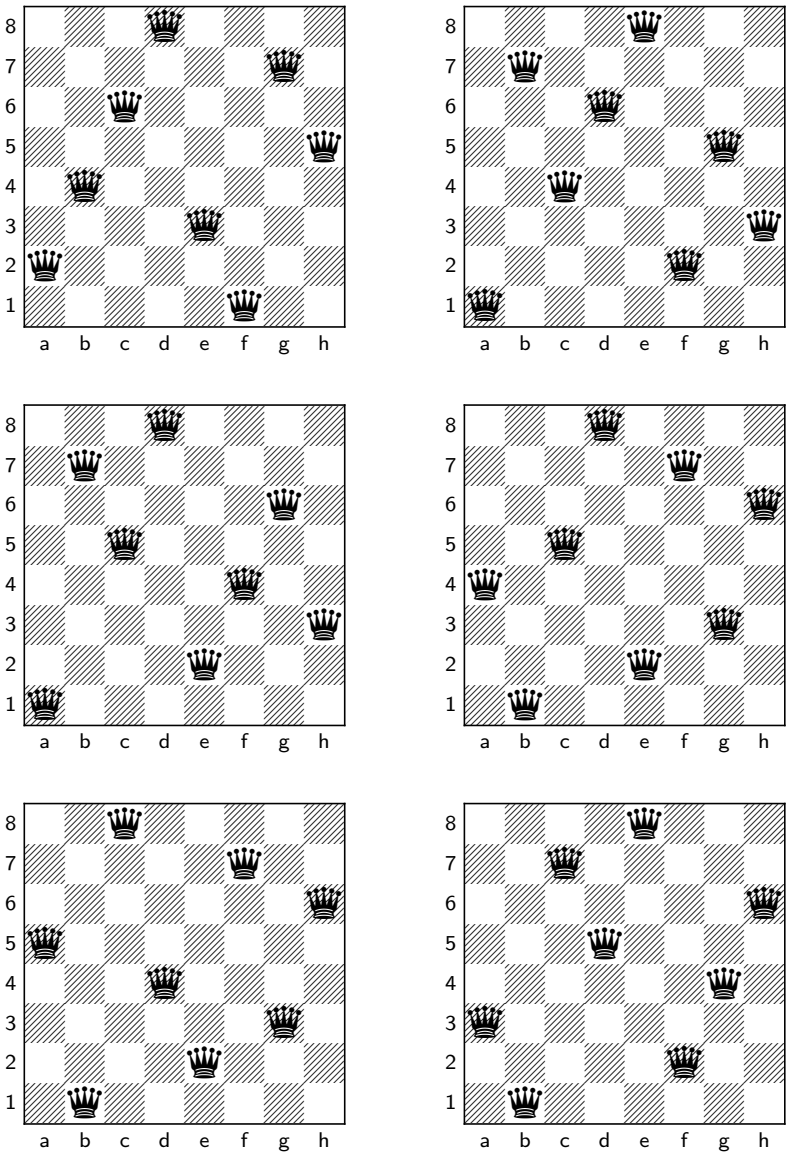


Figure 5.1: Queen moves

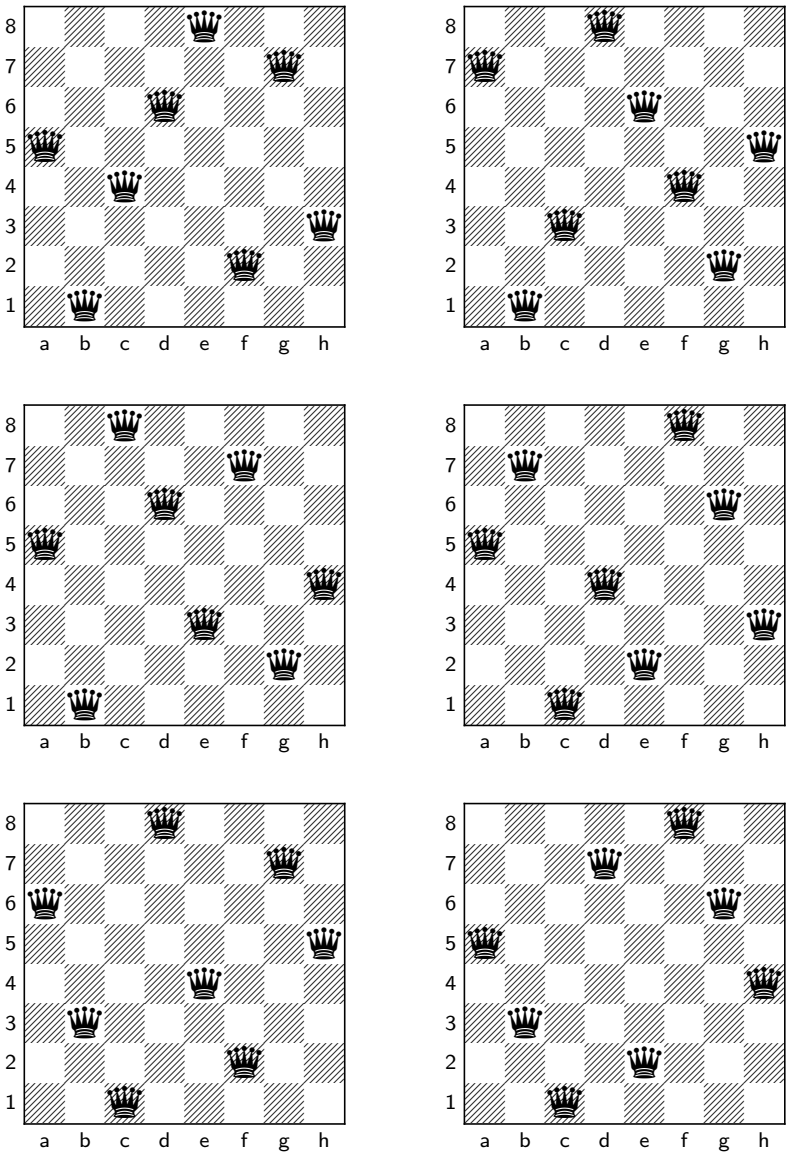
5.3 Solutions to 8 Queens

The eight queens puzzle has 92 distinct solutions. If solutions that differ only by symmetry operations (rotations and reflections) of the board are counted as one, the puzzle has 12 unique (or fundamental) solutions [RBC74, page 177].

12 Unique solutions to the 8 Queens problem



12 Unique solutions to the 8 Queens problem



5.4 Modeling 8 Queens

We begin by thinking of the 8 queens as a 8×8 binary matrix. We represent a queen on a square with a 1 and a blank square with a 0. Our underlying

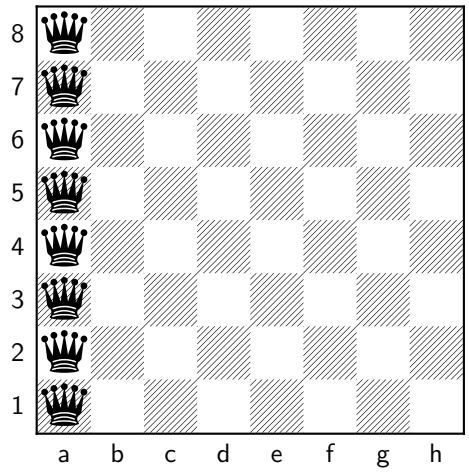
Hilbert space is $X = \mathbb{R}^{8^2}$ or $X = \mathbb{R}^{64}$.

5.5 Row Constraints

Each row can only have one queen in it. In each row we are only allowed 1 one the rest are zeros. We call this constraint C_1.

$$C_1 = \{x \in \mathbb{R}^{8\times 8} | \text{ every row of } x \text{ is a unit vector}\}. \tag{5.1}$$

The row constraint ensures that each row has only one queen in it.

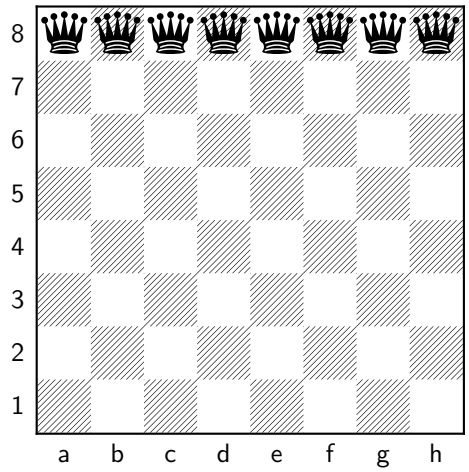


5.6 Column Constraints

Each column can only have one queen in it. In each column we are only allowed 1 one the rest are zeros. We call this constraint C_2

$$C_2 = \{x \in \mathbb{R}^{8\times 8} | \text{ every column } x \text{ is a unit vector}\}. \tag{5.2}$$

The column constraint ensures that each column has only one queen in it.

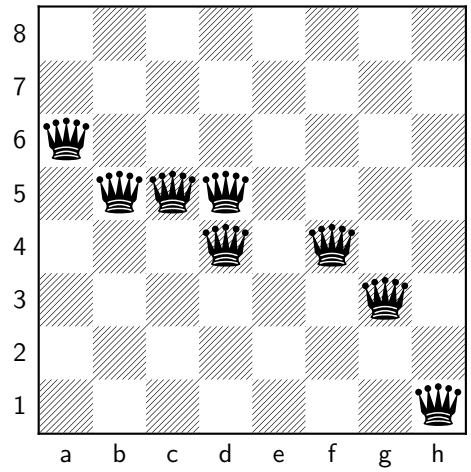


5.7 Positive Slope Constraints

Every positively sloped diagonal must have at most one queen in it. In each positively sloped diagonal we are only allowed at most one 1. We call this constraint C_3. We can think of these as diagonals, or bishop moves. There are 15 positively sloped diagonals.

$$C_3 = \{x \in \mathbb{R}^{8\times 8} | \text{ every positive diagonal of } x \text{ is a unit vector or the 0 vector}\}. \tag{5.3}$$

The positive slope constraint ensures that each positive diagonal has at most one queen in it.

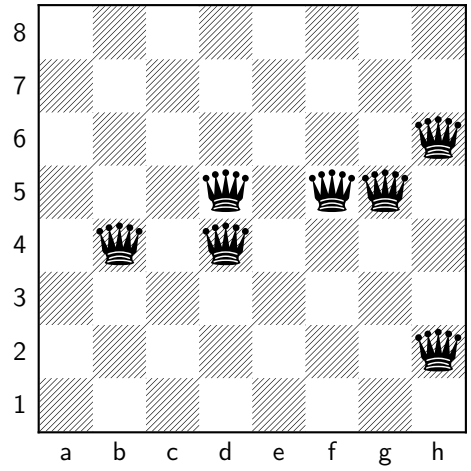


5.8 Negative Slope Constraints

As with the positive slope constraint, the negative sloped constraint must have at most one queen in it. Again we are only allowed at most one 1. We call this constraint C_4. There are 15 negatively sloped diagonals.

$$C_4 = \{x \in \mathbb{R}^{8\times 8} | \text{ every negative diagonal of } x \text{ is a unit vector or the 0 vector}\}. \tag{5.4}$$

The negative slope constraint ensures that each negative diagonal has at most one queen in it.



5.9 Projection onto the Nearest Unit Vector

Denote the standard unit vectors in $\mathbb{R}^n$ by e_i. Let $C = \{e_1, \ldots, e_n\}$ be a very nonconvex set in X. Then

$$P_C x = \{e_i \mid x_i = \max\{x_1, \ldots, x_n\}\}. \tag{5.5}$$

Proof. The set C has a finite number of elements so $P_C x \neq \varnothing$. Let $i \in \{1, \ldots, n\}$. Then

$$\begin{aligned}
e_i \in P_C x &\iff \|x - e_i\| \leq \|x - e_j\| \quad \forall j && (5.6)\\
&\iff \|x - e_i\|^2 \leq \|x - e_j\|^2 \quad \forall j && (5.7)\\
&\iff \|x\|^2 - 2\langle x, e_i\rangle + \|e_i\|^2 \leq \|x\|^2 - 2\langle x, e_j\rangle + \|e_j\|^2 \quad \forall j && (5.8)\\
&\iff -2\langle x, e_i\rangle \leq -2\langle x, e_j\rangle \quad \forall j && (5.9)\\
&\iff \langle x, e_j\rangle \leq \langle x, e_i\rangle \quad \forall j && (5.10)\\
&\iff x_j \leq x_i \quad \forall j. && (5.11)
\end{aligned}$$

□

This projection is very easy to compute, especially for a computer as it picks the biggest component, sets it to 1 and the others to zero. All the constraints can be represented as unit vectors. For the diagonals we compare the projection with the zero vector and choose the nearest one. In general we pick the biggest component in our vector, set it to 1 and the others to 0. In the case of a tie we pick the lowest component.

The constraints C_1,C_2,C_3, and C_4 can be applied to the 8 queens matrix as projections.

$$\begin{pmatrix}
.3 & .8 & .3 & .2 & .4 & .5 & .5 & .1\\
.1 & .1 & .1 & .1 & .1 & .1 & .1 & .1\\
.2 & .7 & .4 & .2 & .7 & .8 & .5 & .9\\
.9 & .5 & .5 & .5 & .5 & .5 & .4 & .5\\
.4 & .2 & 0 & 0 & 0 & 0 & .2 & .3\\
1 & .1 & .1 & .3 & .2 & .6 & .5 & .6\\
-6 & .8 & .1 & .2 & .3 & .4 & .7 & .9\\
.3 & .4 & .7 & .4 & .2 & 1 & .8 & 2
\end{pmatrix} \mapsto \begin{pmatrix}
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0\\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1\\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{pmatrix}$$

The matrix on the right is $P_{C_1} x$ where x is the matrix on the left.

The negative slope constraint is exactly the same as the positive constraint except in the negative direction. If we rotate the chessboard 90

degrees in a counterclockwise direction, apply the positive slope constraint and rotate it 90 degrees clockwise back we get the negative slope constraint. In fact this is how the algorithm is implemented.

$$\begin{pmatrix} .3 & 0 & .3 & .2 & .4 & .5 & .5 & .1 \\ 0 & .1 & .1 & .1 & .1 & .1 & .1 & .1 \\ .2 & .7 & .4 & .2 & .7 & .8 & .7 & .9 \\ .9 & .5 & .5 & .5 & .5 & .5 & .4 & .5 \\ .4 & .2 & 0 & 0 & 0 & 0 & .2 & 0 \\ 1 & .1 & .1 & .3 & .2 & .6 & 0 & .6 \\ -6 & .9 & .1 & .2 & .3 & 0 & .7 & .9 \\ .3 & .4 & .7 & .4 & 0 & 1 & .8 & 0 \end{pmatrix} \mapsto \begin{pmatrix} 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

The matrix on the right is $P_{C_3}x$ where x is the matrix on the left.

5.10 Iterations

Recall that Elser's Difference Map with $\beta = 1$, Lions-Mercier and Douglas Rachford algorithms can be defined as

$$T = \frac{\mathrm{Id} + R_A R_B}{2}, \tag{5.12}$$

where R_A and R_B are reflectors onto their respective sets. The solution to our 8 Queens problem is in $C_1 \cap C_2 \cap C_3 \cap C_4$. We have 4 constraints and our algorithm can only handle two. In order to address this we use the product space and the diagonal space previously defined by Definitions 2.25 and 2.27 respectively. We'll work in the Hilbert space

$$X \times X \times X \times X, \text{ recall } X = \mathbb{R}^{8^2}. \tag{5.13}$$

We also define

$$C = C_1 \times C_2 \times C_3 \times C_4 = \{(c_1, c_2, c_3, c_4) | c_i \in C_i\} \subseteq X^4. \tag{5.14}$$

Now the Diagonal,

$$\Delta = \{(x_1, x_2, x_3, x_4) \mid x_1 = x_2 = x_3 = x_4 \in X\} \tag{5.15}$$

Then

$$\Delta \cap C = \{(c_1, c_2, c_3, c_4) \mid c_1 = c_2 = c_3 = c_4, c_i \in C_i\} \tag{5.16}$$

and get that

$$x \in C_1 \cap C_2 \cap C_3 \cap C_4 \iff (x, x, x, x) \in C \cap \Delta. \tag{5.17}$$

Our solutions lie in the intersection of the 4 constraints. The reflectors used in the Difference Map are two times the projection minus the identity. In our model, we have two projections. The projection onto C, our constraints and the projection onto the diagonal. The projection onto the diagonal is described by Example 2.48:

$$P_\Delta(x_1, x_2, x_3, x_4) = (y, y, y, y) \text{ where } y = \frac{x_1 + x_2 + x_3 + x_4}{4}. \tag{5.18}$$

The projection onto the constraints is described by Example 2.47:

$$P_C = P_{C_1 \times C_2 \times C_3 \times C_4}(x_1, x_2, x_3, x_4) = (P_{C_1} x_1, P_{C_2} x_2, P_{C_3} x_3, P_{C_4} x_4). \tag{5.19}$$

Finally, we define the reflectors and use the Difference Map,

$$R_\Delta := 2P_\Delta - \mathrm{Id} \tag{5.20}$$

$$R_C := 2P_C - \mathrm{Id}\,. \tag{5.21}$$

The iterations are described as

$$\mathbf{x}_{n+1} = \left(\frac{\mathrm{Id} + R_C R_\triangle}{2} \right) \mathbf{x}_n. \tag{5.22}$$

We round one of $P_\triangle \mathbf{x}_n$ to a 0-1 matrix, call the result $\mathbf{y}_n$. Then we monitor $\sum_{i=1}^{4} \|\mathbf{y}_n - P_{C_i}\mathbf{y}_n\|^2$, which is 0 if and only if $\mathbf{y}_n$ is a solution.

5.11 8 Queens Example

In this example we start by placing 8 queens to the top left corner- a ridiculous starting position. As we see in Figure 5.3, after 30 iterations the board is more spread out. We have a queen in every row. Also notice that most diagonal constraints are satisfied. Notice that there are more than 8 queens on the board at this time. This is because the algorithm looks at the entries in each cells and rounds them, if the entry is greater or equal to 0.5, we place a queens in that entry, otherwise it is empty.

Finally after 96 iterations we see in Figure 5.6, we have a solution.

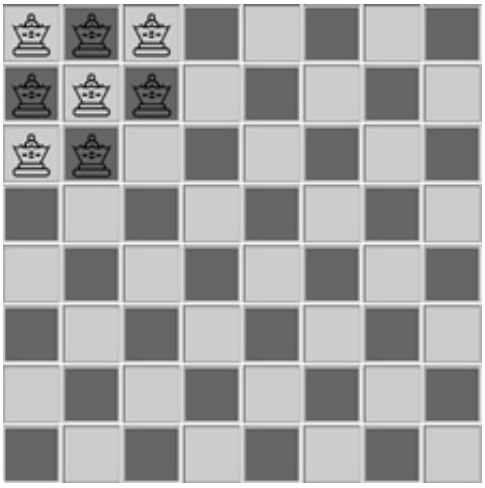


Figure 5.2: Initial Starting Position

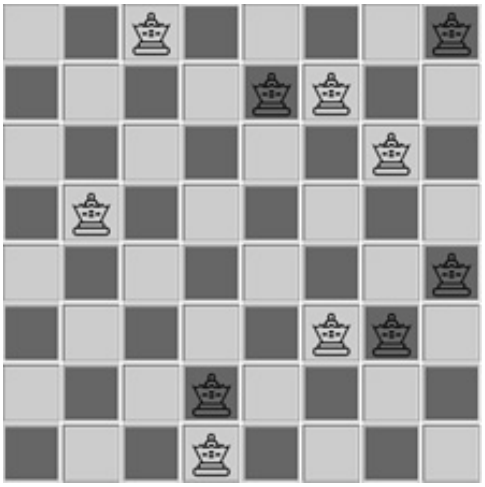


Figure 5.3: 30 iterations

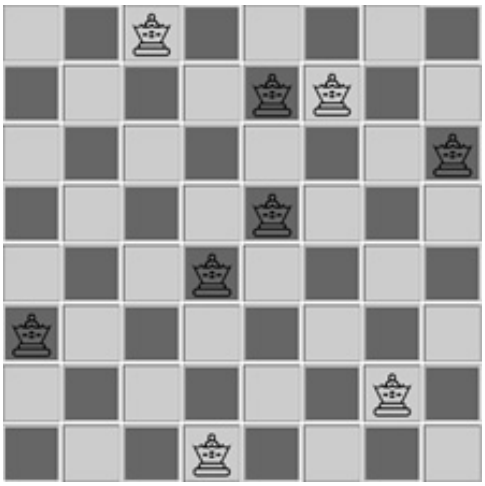


Figure 5.4: 60 iterations

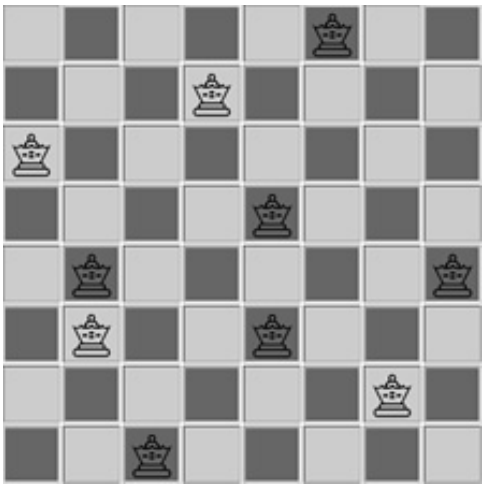


Figure 5.5: 90 iterations

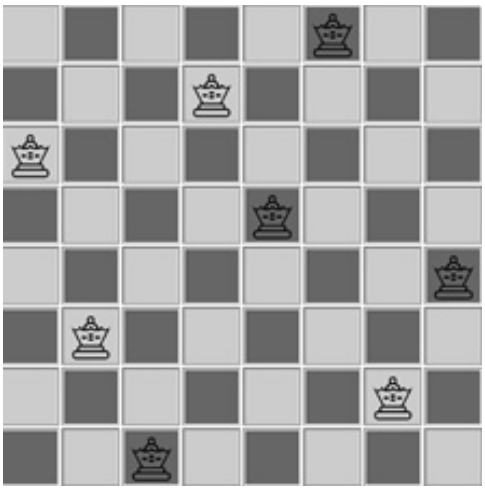


Figure 5.6: Solved in 96 iterations

Chapter 6

Sudoku

We have proved some results about projections and convergence if our sets are convex. There is no theory about convergence if our sets are non-convex. In fact, it is known to fail. The method of alternating projections fails very quickly if our sets are non-convex- it gets stuck between two points and bounces back and forth endlessly. Sudoku uses integers from 1 to 9 which are a very non-convex set. However, when we apply Elser's difference map to our very non-convex Sudoku constraint sets, it converges (very quickly even); which I find amazing. From a complexity point of view, Sudoku is NP-complete, see [Epp05] for details. Below are the full details on how to model Sudoku to solve it using Elser's Difference Map.

6.1 What is Sudoku?

Sudoku literally means "number single". It was derived from the Japanese phrase "Suuji wa dokushin ni kagiru". This phrase translates as "the numbers must occur only once" or "the numbers must be single". Sudoku is a logic-based number placement puzzle. The objective of Sudoku is to fill a 9×9 grid so that each column, each row, and each of the nine 3×3 boxes contains the digits from 1 to 9, only one time each. The game is a type of Latin Square game with the additional constraint of the 3×3 offices. Because of this Leonard Euler is often incorrectly cited as the father of Sudoku. Sudoku was actually invented by an American architect named Howard Garns in 1979; he called it "Number Place". Sudoku is now published and trademarked by Nikoli of Japan and given its name in 1986 [BT09, Pages 6–10], [Eri10]. A sample Sudoku puzzle and its solution is given below in 6.1 and 6.1 respectively (pictures from [Wik]).

6.2 Modeling Sudoku

We begin by thinking of the Sudoku as a 9×9 integer matrix. Think of each cell as an elevator shaft, where the integer number of the cell is the floor. If

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Figure 6.1: A Sample Sudoku Puzzle

5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9	2	4	8	5	6
9	6	1	5	3	7	2	8	4
2	8	7	4	1	9	6	3	5
3	4	5	2	8	6	1	7	9

Figure 6.2: A Sample Sudoku Solution

the floor is say 3, we have 0,0,1,0,0,0,0,0,0 on the floors of the elevator shaft. If we do this for every cell, we add another dimension to our 9×9 matrix. We now have a $9 \times 9 \times 9$ cube! We set all the blank entries to 1, but the starting position does not matter. Let Q be a cube in X described by

$$Q(i, j, k), \text{ where } \{i, j, k\} \subseteq \{1, \ldots, 9\}. \tag{6.1}$$

So an elevator shaft is $Q(i, j, :)$ (i.e., the 3rd component is free using Matlab notation). We aim to place unit vectors into each elevator shaft. Our space $X = \mathbb{R}^{9^3}$ or $X = \mathbb{R}^{729}$. We now describe our constraints, which are similar to those in the 8 Queens problem.

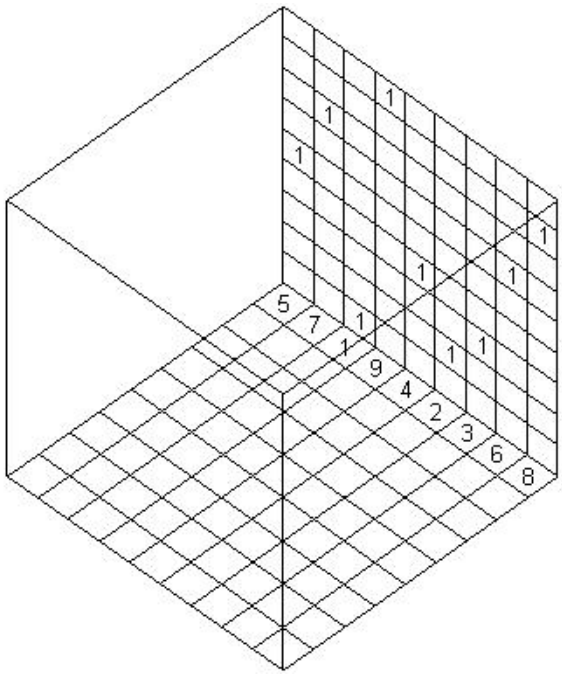


Figure 6.3: 9 Sudoku Elavator Shafts

6.3 Constraints

6.3.1 North-South Hallways

Each cell of every column must not repeat any of the numbers 1 to 9. Let's call these North-South hallways. In each hallway we are only allowed at

most one 1 and the rest are zeros. There are 81 North-South hallways, 9 per floor. Therefore, our constraint becomes

$$C_1 = \left\{ Q \in \mathbb{R}^{9^3} \middle| Q(i,:,k) \text{ is a stardard unit vector} \quad \forall i,k \right\}. \tag{6.2}$$

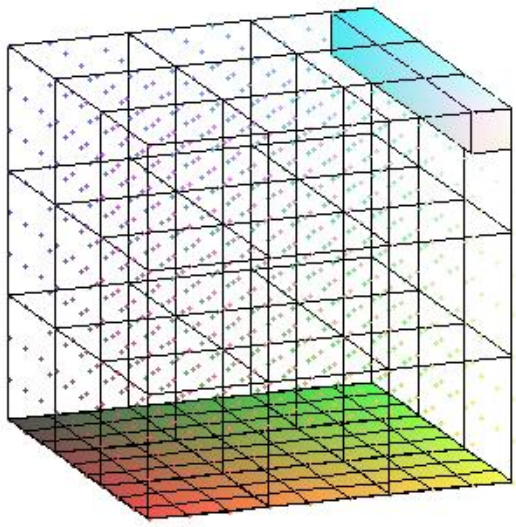


Figure 6.4: A Sodoku Hallway

6.3.2 East-West Hallways

Each cell of every row must not repeat any of the numbers 1 to 9. Let's call these East - West hallways. In each hallway we are only allowed at most one 1 and the rest are zeros. There are 81 East-West hallways, 9 per floor. Therefore, our constraint becomes

$$C_2 = \left\{ Q \in \mathbb{R}^{9^3} \middle| Q(:,j,k) \text{ is a stardard unit vector} \quad \forall j,k \right\}. \tag{6.3}$$

6.3.3 Meeting Rooms

Every 3×3 square on each floor must not repeat any of the numbers 1 to 9; we call these meeting rooms. In each meeting room we are only allowed at most 1 one the rest are zeros. There are 81 meeting rooms, 9 per floor. We define the following:

$$E = \{A \in \mathbb{R}^{3\times 3} | \text{ all entries are zero except one entry equals zero}\}, \quad (6.4)$$

$$J = \{\{1,2,3\},\{4,5,6\},\{7,8,9\}\}, \quad (6.5)$$

and, for every $Q \in \mathbb{R}^{9^3}$, $I_1 \in J$, $I_2 \in J$, and $k \in \{1,\ldots,9\}$, set

$$M_{I_1,I_2,k}(Q) = \left\{A \in \mathbb{R}^{3\times 3} | A_{i \bmod 3, j \bmod 3} = Q(i,j,k) \quad \forall j \in I_1 \forall j \in I_2\right\}. \quad (6.6)$$

Therefore, our constraint set is

$$C_3 = \left\{Q \in \mathbb{R}^{9^3} | \forall k, \forall I_1 \in J, \forall I_2 \in J, M_{I_1,I_2,k}(Q) \in E\right\}. \quad (6.7)$$

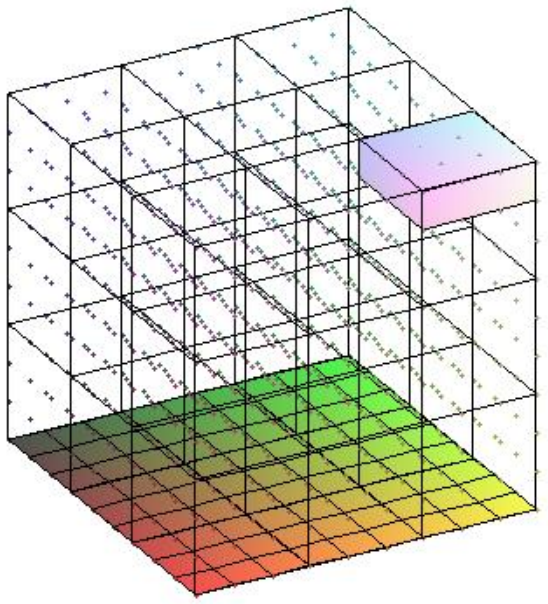


Figure 6.5: A Sodoku Meeting Room

As you can see from the picture the meeting room is not a typical unit vector. We can think of it as a fancy unit vector where we write it as a 3×3 matrix instead of the usual 1×9 vector we are used to.

6.3.4 The Given Sudoku

Constraint 4 is the given Sudoku problem. We call this constraint C_4. This is the set of all cubes $Q \in \mathbb{R}^{9^3}$ such that if the (i, j) entry is prescribed and equal to $\gamma_{i,j}$ then $Q(i, j, :)$ is equal to the $\gamma_{i,j}$-unit vector; otherwise, $Q(i, j, :)$ is any unit vector.

6.4 Iterations

Recall that Elser's Difference Map with $\beta = 1$, Lions-Mercier and Douglas Rachford algorithms can be defined as

$$T = \frac{\mathrm{Id} + R_A R_B}{2}, \tag{6.8}$$

where R_A and R_B are reflectors onto their respective sets. The solution to our Sudoku problem is in $C_1 \cap C_2 \cap C_3 \cap C_4$. We have 4 constraints and our algorithm can only handle two. As with 8 Queens, we use the product space and the diagonal, which we have previous defined. We work in the Hilbert product space

$$X \times X \times X \times X, \text{ recall } X = \mathbb{R}^{9^3}. \tag{6.9}$$

We also define

$$C = C_1 \times C_2 \times C_3 \times C_4 = \{(c_1, c_2, c_3, c_4) | c_i \in C_i\} \subseteq X^4, \tag{6.10}$$

and the Diagonal,

$$\Delta = \{(x_1, x_2, x_3, x_4) \mid x_1 = x_2 = x_3 = x_4 \in X\}. \tag{6.11}$$

Then

$$\Delta \cap C = \{(c_1, c_2, c_3, c_4) \mid c_1 = c_2 = c_3 = c_4, c_i \in C_i\} \tag{6.12}$$

and hence

$$x \in C_1 \cap C_2 \cap C_3 \cap C_4 \iff (x, x, x, x) \in C \cap \Delta. \tag{6.13}$$

Our unique solution (assuming a well posed Sudoku puzzle) lies in the intersection of the 4 constraints. The reflectors used in the Difference Map

are two times the projection minus the identity. In our model, we have two projections. The projection onto C, our constraints and the projection onto the diagonal. The projection onto the diagonal is described by Example 2.48:

$$P_{\Delta}(x_1, x_2, x_3, x_4) = (y, y, y, y) \text{ where } y = \frac{x_1 + x_2 + x_3 + x_4}{4}. \tag{6.14}$$

The projection onto the constraints is described by Example 2.47:

$$P_C = P_{C_1 \times C_2 \times C_3 \times C_4}(x_1, x_2, x_3, x_4) = (P_{C_1} x_1, P_{C_2} x_2, P_{C_3} x_3, P_{C_4} x_4). \tag{6.15}$$

Finally, we define the reflectors and use the Difference Map,

$$R_{\Delta} = 2P_{\Delta} - \mathrm{Id} \tag{6.16}$$

$$R_C := 2P_C - \mathrm{Id}\,. \tag{6.17}$$

The iterations are described as

$$\mathbf{x}_{n+1} = \left(\frac{\mathrm{Id} + R_C R_{\triangle}}{2} \right) \mathbf{x}_n. \tag{6.18}$$

We now apply the exact same algorithm described in details with the 8-Queens example. Now we pick a cube from our diagonal and convert the cube into a matrix by projecting onto the nearest unit vector of each elevator shaft. Convergence is monitored similarly.

It should be noted that the iterations are in our cube space and in order to view them in a 9×9 matrix that is Sodoku, we need to flatten the cube. That is take the $9 \times 9 \times 9$ binary cube and convert it to a 9×9 integer matrix by converting each elevator shaft containing a standard unit vector to its corresponding integer- see Figure 6.3.

6.5 Sudoku Example

Arto Inkala, a Finnish mathematician, is an enthusiastic puzzle and problem solver. He completed his doctoral thesis in applied mathematics in 2001. He creates very difficult Sudoku puzzles. Inkala claims that his "mathematic background has been great help when creating the most difficult Sudokus by [using] genetic algorithms, different optimizing methods and artificial intelligence. On Inkala's website (`www.aisudoku.com`), he shares 10 of his difficult Sudoku puzzles. According to the Finnish puzzle maker, he calls the most difficult Sudoku puzzle ever "AI Escargot", because it looks like

a snail. The AI comes from Inkala's initials. He calls solving it is like an intellectual culinary pleasure.

We use the difference map method described above to solve all 10 of Inkala's Sudoku puzzle. The online implementation of the Sudoku solver can be found at `www.schaad.ca`, the author's website. We see that AI Escargot was solved in 6627 iterations whereas AI labryinth took over 12000 iterations. Even though Inkala claims that AI Escargot is the world's hardest Sudoku puzzle, our algorithm solves it faster than other Sudokus created by Inkala. The algorithm managed to solve AI honeypot in only 208 iterations! The Sudoku problems and pictures in the left column are from `www.aisudoku.com`, whereas the Sudoku solutions in the right column are original work.

AI escargot solved in 6627 iterations

1					7		9	
	3			2				8
		9	6			5		
		5	3			9		
	1			8				2
6					4			
3							1	
	4							7
		7				3		

1	6	2	8	5	*7*	4	*9*	3
5	*3*	4	1	*2*	9	6	7	*8*
7	8	*9*	*6*	4	3	*5*	2	1
4	7	*5*	*3*	1	2	*9*	8	6
9	*1*	3	5	*8*	6	7	4	*2*
6	2	8	7	9	*4*	1	3	5
3	5	6	4	7	8	2	*1*	9
2	*4*	1	9	3	5	8	6	*7*
8	9	*7*	2	6	1	*3*	5	4

AI killer application solved in 882 iterations

							7	
	6			1				4
		3	4			2		
8					3		5	
		2	9			7		
	4			8				9
	2			6				7
			1			9		
7					8		6	

9	5	4	8	2	6	1	*7*	3
2	*6*	8	3	*1*	7	5	9	*4*
1	7	*3*	*4*	9	5	*2*	8	6
8	1	9	7	4	*3*	6	*5*	2
6	3	*2*	*9*	5	1	*7*	4	8
5	*4*	7	6	*8*	2	3	1	*9*
4	*2*	1	5	*6*	9	8	3	*7*
3	8	6	*1*	7	4	*9*	2	5
7	9	5	2	3	*8*	4	*6*	1

AI lucky diamond solved in 276 iterations

1			5			4		
		9		3				
	7				8			5
		1					3	
8			6			5		
	9				7			8
		4		2			1	
2			8			6		
					1			2

1	2	8	*5*	7	6	*4*	9	3
5	4	*9*	1	*3*	2	7	8	6
3	*7*	6	9	4	*8*	1	2	*5*
7	6	*1*	2	8	5	9	*3*	4
8	3	2	*6*	9	4	*5*	7	1
4	*9*	5	3	1	*7*	2	6	*8*
6	5	*4*	7	*2*	3	8	*1*	9
2	1	3	*8*	5	9	*6*	4	7
9	8	7	4	6	*1*	3	5	*2*

AI worm hole solved in 4998 iterations

	8							1
		7			4		2	
6			3			7		
		2			9			
1				6				8
	3		4					
		1	7			6		
	9				8			5
							4	

9	*8*	4	2	7	6	3	5	*1*
3	1	*7*	9	5	*4*	8	*2*	6
6	2	5	*3*	8	1	*7*	9	4
5	6	*2*	8	3	*9*	4	1	7
1	4	9	5	*6*	7	2	3	*8*
7	*3*	8	*4*	1	2	5	6	9
4	5	*1*	*7*	9	3	*6*	8	2
2	*9*	3	6	4	*8*	1	7	*5*
8	7	6	1	2	5	9	*4*	3

AI labyrinth solved in 12025 iterations

1			4			8		
	4			3				9
		9			6		5	
	5		3					
					1	6		
				7				2
		4		1		9		
7			8					4
	2				4		8	

1	6	3	*4*	9	5	*8*	2	7
8	*4*	5	2	*3*	7	1	6	*9*
2	7	*9*	1	8	*6*	4	*5*	3
4	*5*	2	*3*	6	8	7	9	1
9	3	7	5	2	*1*	*6*	4	8
6	1	8	*7*	4	9	5	3	*2*
3	8	*4*	6	*1*	2	*9*	7	5
7	9	6	*8*	5	3	2	1	*4*
5	*2*	1	9	7	*4*	3	*8*	6

AI circles solved in 2410 iterations

		5			9	7		
	6						2	
1			8					6
	1		7					4
		7		6			3	
6					3	2		
					6		4	
	9			5		1		
8			1					2

4	8	*5*	6	2	*9*	*7*	1	3
7	*6*	9	3	4	1	8	*2*	5
1	3	2	*8*	7	5	4	9	*6*
5	*1*	3	*7*	9	2	6	8	*4*
9	2	*7*	4	*6*	8	5	*3*	1
6	4	8	5	1	*3*	*2*	7	9
2	5	1	9	8	*6*	3	*4*	7
3	*9*	4	2	*5*	7	*1*	6	8
8	7	6	*1*	3	4	9	5	*2*

AI squadron solved in 3252 iterations

6						2		
	9				1			5
		8		3			4	
					2			1
5			6			9		
		7		9				
	7				3			2
			4			5		
		6		7			8	

6	5	3	7	4	9	*2*	1	8
7	*9*	4	8	2	*1*	6	3	*5*
1	2	*8*	5	*3*	6	7	*4*	9
4	6	9	3	5	*2*	8	7	*1*
5	3	1	*6*	8	7	*9*	2	4
2	8	*7*	1	*9*	4	3	5	6
8	*7*	5	9	1	*3*	4	6	*2*
3	1	2	*4*	6	8	*5*	9	7
9	4	*6*	2	*7*	5	1	*8*	3

AI honeypot solved in 208 iterations

1							6	
			1					3
		5			2	9		
		9			1			
7				4			8	
	3		5					2
5			4					6
		8		6			7	
	7				5			

1	8	2	3	9	4	5	*6*	7
9	6	7	*1*	5	8	2	4	*3*
3	4	*5*	6	7	*2*	*9*	1	8
8	2	*9*	7	3	*1*	6	5	4
7	5	6	2	*4*	9	3	*8*	1
4	*3*	1	*5*	8	6	7	9	*2*
5	9	3	*4*	1	7	8	2	*6*
2	1	*8*	9	*6*	3	4	*7*	5
6	*7*	4	8	2	*5*	1	3	9

AI tweezers solved in 688 iterations

				1				4
	3		2					
6					8		9	
		7		6				5
9					5		8	
			8			4		
	4		9			1		
7					2		4	
		5		3				7

2	7	9	3	*1*	6	8	5	*4*
5	*3*	8	*2*	9	4	7	1	6
6	1	4	5	7	*8*	3	*9*	2
4	8	*7*	1	*6*	9	2	3	*5*
9	2	3	7	4	*5*	6	*8*	1
1	5	6	*8*	2	3	*4*	7	9
3	*4*	2	*9*	5	7	*1*	6	8
7	9	1	6	8	*2*	5	*4*	3
8	6	*5*	4	*3*	1	9	2	*7*

AI broken brick solved in 1598 iterations

4				6			7	
						6		
	3				2			1
7					8	5		
	1		4					
	2		9	5				
						7		5
		9	1				3	
		3		4			8	

4	5	1	8	*6*	3	9	*7*	2
9	8	2	7	1	4	*6*	5	3
6	*3*	7	5	9	*2*	8	4	*1*
7	9	6	3	2	*8*	*5*	1	4
3	*1*	5	*4*	7	6	2	9	8
8	*2*	4	*9*	*5*	1	3	6	7
1	4	8	6	3	9	*7*	2	*5*
2	7	*9*	*1*	8	5	4	*3*	6
5	6	*3*	2	*4*	7	1	*8*	9

In the following figure, we show how the online solver solves AI Escargot. It takes 6627 iterations and we take snapshots every 1000 iteration until we come to a solution.

1					7		9	
	3			2				8
			9	6			5	
			5	3			9	
	1			8				2
6					4			
3							1	
	4							7
			7				3	

Current Counter 1000
168547293
534129768
729638541
475312986
913586472
682954135
396752814
241893657
857461329

Current Counter 2000
186457293
536921748
429633541
275312984
714585632
628794175
358766419
841263867
967118326

Current Counter 3000
166457293
534921178
279638544
475312986
713785462
628294731
396875415
541293627
257146329

Current Counter 4000
156837294
734529168
829641573
475312981
913786442
682194135
398275614
241963857
567451329

Current Counter 5000
184437296
434925168
729648541
275362941
413589752
698714735
352872414
841263657
867142329

Current Counter 6000
126857493
534129678
879633541
725361984
413985762
698274135
352798416
941263257
867142359

Current Counter 6627
162857493
534129678
789643521
475312986
913586742
628794135
356478219
241935867
897261354

Figure 6.6: Solution to Escargot

6.6 Sudoku Distance

The next figure is very interesting. At each iteration, we compute a distance from the solution. At each iteration, our algorithm displays the current iterate as 9×9 matrix which represents an attempt at a Sudoku puzzle solution. If we let the current iterate be the matrix A, and let the solution to the Sudoku puzzle be B, then we can compute the difference which we call matrix C. The difference is computed by subtracting each element, i.e. $c_{ij} = a_{ij} - b_{ij}$, where $i, j \in 1, 2, \ldots, 9$. We calculate the distance as follows,

$$\|C\| = \sqrt{\sum_{i,j} c_{ij}^2} \tag{6.19}$$

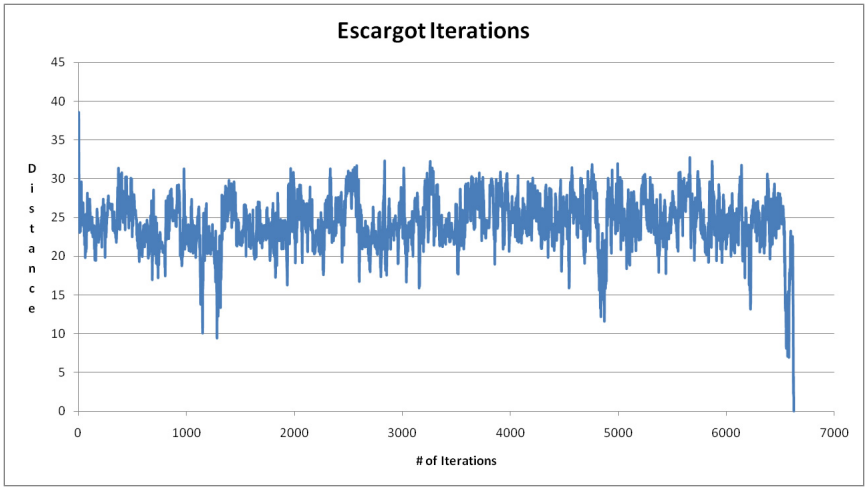


Figure 6.7: Distance versus number of iterations

The graph is not monotone. It shows that Elser's Difference Map sometimes gets close to a solution, but needs to go further away to actually get the solution. There are a few times, around 1100, 1200, 4800, and 6300 iterations where we get fairly close to a solution, but then we get farther away before finding a solution. Also notice how quickly we converge to a solution at the end.

Chapter 7

Conclusion

We have shown that Elser's Difference Map is equivalent to the Douglas-Rachford and Fienup's Hybrid Input-Output algorithms, where $\beta = 1$. We showed how to model the 8-queens problem and following Elser, we modeled Sudoku as well. We hope that the detailed expository in this thesis will help readers to understand modeling feasibility problems such as Sudoku and 8-queens.

We used several very non-convex sets and applied the theory of convex sets to converge to a solution. There is no theory for the nonconvex case, although we have demonstrated the algorithm seems to work. Some future research might include discovering convergence proofs for the nonconvex case, starting with simple examples in the Euclidean plane or building on the paper [LLM09] by Lewis, Luke and Malick.

Finally, we showed that the operator governing the Douglas-Rachford iteration need not be a proximal map even when the two involved resolvents are; this refines an observation of Eckstein and underlines the need for monotone operator theory.

Bibliography

[AY89] Bruce Abramson and Moti Yung. Divide and conquer under global constraints: A solution to the n-queens problem. *Journal of Parallel and Distributed Computing*, 6(3):649 – 662, 1989. Available from: http://www.sciencedirect.com/science/article/B6WKJ-4BRJHWS-13/2/44424d257adca8566925a5ca897293e9. → pages 47

[BCL02] Heinz H. Bauschke, Patrick L. Combettes, and D. Russell Luke. Phase retrieval, error reduction algorithm, and Fienup variants: a view from convex optimization. *J. Opt. Soc. Amer. A*, 19(7):1334–1345, 2002. Available from: http://dx.doi.org/10.1364/JOSAA.19.001334. → pages 1

[bir09] Interdisciplinary workshop on fixed-point algorithms for inverse problems in science and engineering, November 2009. Available from: http://www.birs.ca/birspages.php?task=displayevent&event_id=09w5006. → pages 2

[Bor83] J. M. Borwein. Adjoint process duality. *Math. Oper. Res.*, 8(3):403–434, 1983. Available from: http://dx.doi.org/10.1287/moor.8.3.403. → pages 42

[BT09] Seymour S. Block and Santiago A. Tavares. *Before Sudoku*. Oxford University Press, Oxford, 2009. The world of magic squares. → pages 60

[BWY09] Heinz H. Bauschke, Xianfu Wang, and Liangjin Yao. On Borwein-Wiersma decompositions of monotone linear relations, 2009. Available from: http://www.citebase.org/abstract?id=oai:arXiv.org:0912.2772. → pages 41

[Com04] Patrick L. Combettes. Solving monotone inclusions via compositions of nonexpansive averaged operators. *Optimization*, 53(5-6):475–504, 2004. Available from: http://dx.doi.org/10.1080/02331930412331327157. → pages 43

[Cro98] Ronald Cross. *Multivalued linear operators*, volume 213 of *Monographs and Textbooks in Pure and Applied Mathematics*. Marcel Dekker Inc., New York, 1998. → pages 42

[Deu01] Frank Deutsch. *Best approximation in inner product spaces*. CMS Books in Mathematics/Ouvrages de Mathématiques de la SMC, 7. Springer-Verlag, New York, 2001. → pages 20, 27, 31, 34

[EB92] Jonathan Eckstein and Dimitri P. Bertsekas. On the Douglas-Rachford splitting method and the proximal point algorithm for maximal monotone operators. *Math. Programming*, 55(3, Ser. A):293–318, 1992. Available from: `http://dx.doi.org/10.1007/BF01581204`. → pages 38

[Eck89] Jonathan Eckstein. *Splitting Methods for Monotone Operators with Applications to Parallel Optimization*. PhD dissertation, Massachusetts Institute of Technology, Cambridge, Massachusetts, June 1989. → pages 43

[EF98] Jonathan Eckstein and Michael C. Ferris. Operator-splitting methods for monotone affine variational inequalities, with a parallel application to optimal control. *INFORMS J. Comput.*, 10(2):218–235, 1998. → pages 43

[Epp05] David Eppstein. Nonrepetitive paths and cycles in graphs with application to sudoku. *CoRR*, abs/cs/0507053, 2005. → pages 60

[Eri10] Martin Erickson. *Pearls of discrete mathematics*. Discrete Mathematics and its Applications (Boca Raton). CRC Press, Boca Raton, FL, 2010. → pages 60

[ERT07] V. Elser, I. Rankenburg, and P. Thibault. Searching with iterated maps. *Proc. Natl. Acad. Sci. USA*, 104(2):418–423, 2007. Available from: `http://dx.doi.org/10.1073/pnas.0606359104`. → pages 1, 26

[Fie82] J. R. Fienup. Phase retrieval algorithms: a comparison. *Appl. Opt.*, 21(15):2758–2769, 1982. Available from: `http://ao.osa.org/abstract.cfm?URI=ao-21-15-2758`. → pages 26

[GE08] Simon Gravel and Veit Elser. Divide and concur: A general approach to constraint satisfaction. *Phys. Rev. E*, 78(3):036706, Sep 2008. → pages 1

[ICC07] Second mathematical programming society international conference on continuous optimization, August 2007. Available from: `http://iccopt-mopta.mcmaster.ca/`. → pages 1

[Kre78] Erwin Kreyszig. *Introductory functional analysis with applications*. John Wiley & Sons, New York-London-Sydney, 1978. → pages 7, 33, 34, 42

[LLM09] A. S. Lewis, D. R. Luke, and J. Malick. Local linear convergence for alternating and averaged nonconvex projections. *Found. Comput. Math.*, 9(4):485–513, 2009. Available from: `http://dx.doi.org/10.1007/s10208-008-9036-y`. → pages 75

[LM79] P.-L. Lions and B. Mercier. Splitting algorithms for the sum of two nonlinear operators. *SIAM J. Numer. Anal.*, 16(6):964–979, 1979. Available from: `http://dx.doi.org/10.1137/0716071`. → pages 26

[Mey00] Carl Meyer. *Matrix analysis and applied linear algebra*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 2000. With 1 CD-ROM (Windows, Macintosh and UNIX) and a solutions manual (iv+171 pp.). → pages 4, 34, 38

[Min62] George J. Minty. Monotone (nonlinear) operators in Hilbert space. *Duke Math. J.*, 29:341–346, 1962. → pages 39

[MN88] Jan R. Magnus and Heinz Neudecker. *Matrix differential calculus with applications in statistics and econometrics*. Wiley Series in Probability and Mathematical Statistics: Applied Probability and Statistics. John Wiley & Sons Ltd., Chichester, 1988. → pages 40

[Opi67] Zdzisław Opial. Weak convergence of the sequence of successive approximations for nonexpansive mappings. *Bull. Amer. Math. Soc.*, 73:591–597, 1967. → pages 29

[Que] Queen moves. Available from: `http://www.chessdryad.com/education/magictheater/queens/index.htm`. → pages 47

[RBC74] W. W. Rouse Ball and H. S. M. Coxeter. *Mathematical recreations & essays*. University of Toronto Press, Toronto, Ont., twelfth edition, 1974. → pages 47, 48

[Reh] Julie Rehmeyer. The Sudoku solution. *The Science News*. Available from: `http://sciencenews.org/view/generic/id/39529/title/The_Sudoku_solution`. → pages 1

[Roc70] R. T. Rockafellar. On the maximal monotonicity of subdifferential mappings. *Pacific J. Math.*, 33:209–216, 1970. → pages 38

[Roc97] R. Tyrrell Rockafellar. *Convex analysis*. Princeton Landmarks in Mathematics. Princeton University Press, Princeton, NJ, 1997. Reprint of the 1970 original, Princeton Paperbacks. → pages 37, 39, 41

[Rud91] Walter Rudin. *Functional analysis*. International Series in Pure and Applied Mathematics. McGraw-Hill Inc., New York, second edition, 1991. → pages 31

[RW98] R. Tyrrell Rockafellar and Roger J-B. Wets. *Variational analysis*, volume 317 of *Grundlehren der Mathematischen Wissenschaften [Fundamental Principles of Mathematical Sciences]*. Springer-Verlag, Berlin, 1998. → pages 36, 39, 40, 42

[Sim98] Stephen Simons. *Minimax and monotonicity*, volume 1693 of *Lecture Notes in Mathematics*. Springer-Verlag, Berlin, 1998. → pages 38

[Wik] Queen moves. Available from: `http://en.wikipedia.org/wiki/Sudoku`. → pages 60

[Zăl02] C. Zălinescu. *Convex analysis in general vector spaces*. World Scientific Publishing Co. Inc., River Edge, NJ, 2002. → pages 37, 39

[Zar71] Eduardo H. Zarantonello. Projections on convex sets in Hilbert space and spectral theory. I. Projections on convex sets. In *Contributions to nonlinear functional analysis (Proc. Sympos., Math. Res. Center, Univ. Wisconsin, Madison, Wis., 1971)*, pages 237–341. Academic Press, New York, 1971. → pages 32

Appendix A

8 Queens PHP Code

```
<?php

// This code is prototype and is provided as is under the GNU General Public License
// Author: Jason Schaad August 11, 2010

$n=8;

function checkqueens($matrix) {


  // Round the matrix
  for ($x=1;$x<=8;$x++) {
    for ($y=1;$y<=8;$y++) {
      if ($matrix[$x][$y] >=0.5) {
        $matrix[$x][$y]=1;
      }
    }
  }


// PrintMatrix($matrix);
  $total=0;

  // HORIZONTAL

  for ($x=1;$x<=8;$x++) {
    $sum=0;

    for ($y=1;$y<=8;$y++) {
      $sum+=$matrix[$x][$y];
    }

    if ($sum==1) {
      $total++;
    }

  }

  // VERTICAL
```

```
for ($x=1;$x<=8;$x++) {
  $sum=0;

  for ($y=1;$y<=8;$y++) {
    $sum+=$matrix[$y][$x];
  }

  if ($sum==1) {
    $total++;
  }

}

$check=1;

// POSITIVE

if (($matrix[1][1])>1){$check=-1;}
if (($matrix[2][1]+$matrix[1][2])>1){$check=-1;}
if (($matrix[3][1]+$matrix[2][2]+$matrix[1][3]) > 1){$check=-1;}
if (($matrix[4][1]+$matrix[3][2]+$matrix[3][3]+$matrix[1][4])>1){$check=-1;}
if (($matrix[5][1]+$matrix[4][2]+$matrix[3][3]+$matrix[2][4]+
  $matrix[1][5])>1){$check=-1;}
if (($matrix[6][1]+$matrix[5][2]+$matrix[4][3]+$matrix[3][4]+
  $matrix[2][5]+$matrix[1][6])>1) {$check=-1;}
if (($matrix[7][1]+$matrix[6][2]+$matrix[5][3]+$matrix[4][4]+
  $matrix[3][5]+$matrix[2][6]+$matrix[1][7])> 1){$check=-1;}
if (($matrix[8][1]+$matrix[7][2]+$matrix[6][3]+$matrix[5][4]+
  $matrix[4][5]+$matrix[3][6]+$matrix[2][7]+$matrix[1][8])>1) {$check=-1;}
if (($matrix[8][2]+$matrix[7][3]+$matrix[6][4]+$matrix[5][5]+
  $matrix[4][6]+$matrix[3][7]+$matrix[2][8]) > 1) {$check=-1;}
if (($matrix[8][3]+$matrix[7][4]+$matrix[6][5]+$matrix[5][6]+
  $matrix[4][7]+$matrix[3][8])>1) {$check=-1;}
if (($matrix[8][4]+$matrix[7][5]+$matrix[6][6]+$matrix[5][7]+
  $matrix[4][8])>1) {$check=-1;}
if (($matrix[8][5]+$matrix[7][6]+$matrix[6][7]+$matrix[5][8])>1) {$check=-1;}
if (($matrix[8][6]+$matrix[7][7]+$matrix[6][8])>1){$check=-1;}
if (($matrix[8][7]+$matrix[7][8])>1){$check=-1;}
if (($matrix[8][8])>1){$check=-1;}

// NEGATIVE

if (($matrix[8][1]) > 1) {$check=-1;}
if (($matrix[7][1]+$matrix[8][2])>1){$check=-1;}
if (($matrix[6][1]+$matrix[7][2]+$matrix[8][3])>1){$check=-1;}
if (($matrix[5][1]+$matrix[6][2]+$matrix[7][3]+$matrix[8][4])>1){$check=-1;}
if (($matrix[4][1]+$matrix[5][2]+$matrix[6][3]+$matrix[7][4]+
  $matrix[8][5])>1){$check=-1;}
```

```
  if (($matrix[3][1]+$matrix[4][2]+$matrix[5][3]+$matrix[6][4]+
    $matrix[7][5]+$matrix[8][6])>1) {$check=-1;}
  if (($matrix[2][1]+$matrix[3][2]+$matrix[4][3]+$matrix[5][4]+
    $matrix[6][5]+$matrix[7][6]+$matrix[8][7])>1){$check=-1;}
  if (($matrix[1][1]+$matrix[2][2]+$matrix[3][3]+$matrix[4][4]+
    $matrix[5][5]+$matrix[6][6]+$matrix[7][7]+$matrix[8][8])>1){$check=-1;}
  if (($matrix[1][2]+$matrix[2][3]+$matrix[3][4]+$matrix[4][5]+
    $matrix[5][6]+$matrix[6][7]+$matrix[7][8])>1){$check=-1;}
  if (($matrix[1][3]+$matrix[2][4]+$matrix[3][5]+$matrix[4][6]+
    $matrix[5][7]+$matrix[6][8])>1){$check=-1;}
  if (($matrix[1][4]+$matrix[2][5]+$matrix[3][6]+$matrix[4][7]+
    $matrix[5][8])>1){$check=-1;}
  if (($matrix[1][5]+$matrix[2][6]+$matrix[3][7]+$matrix[4][8])>1){$check=-1;}
  if (($matrix[1][6]+$matrix[2][7]+$matrix[3][8])>1) {$check=-1;}
  if (($matrix[1][7]+$matrix[2][8])>1){$check=-1;}
  if (($matrix[1][8])>1){$check=-1;}

  if (($total==16) && ($check==1)) {
    return 1;
  }
  else {
    return 0;
  }

}

function chessround($x, $y) {
  if ($y == 1 ) {

    if ($x >= 0.5) {
      return "<img height=50 width=50 src=3.png>";
    }
    else {
      return "<img height=50 width=50 src=2.png>";
    }

  }
  else if ($y ==0) {

    if ($x >= 0.5) {
      return "<img height=50 width=50 src=4.png>";
    }
    else {
      return "<img height=50 width=50 src=1.png>";
    }


  }
}
```

```
function PrintChessMatrix($matrix) {
  echo "<table border=1>";
  $n=sizeof($matrix);
  for ( $x = 1; $x <= $n; $x++) {
  echo "<tr>";
    for ( $y = 1; $y <= $n; $y++) {
      if (($x % 2)==1) {
        if (($y % 2)==1) {
          echo "<td bgcolor=#999999>".chessround($matrix[$x][$y],1)."</td>";
        }
        else {
          echo "<td bgcolor=#FFFFFF>".chessround($matrix[$x][$y],0)."</td>";
        }
      }
      else {
        if (($y % 2)==0) {
          echo "<td bgcolor=#999999>".chessround($matrix[$x][$y],1)."</td>";
        }
        else {
          echo "<td bgcolor=#FFFFFF>".chessround($matrix[$x][$y],0)."</td>";
        }
      }
    }
    echo "</tr>";
  }
  echo "</table>";
} //end function


function PrintMatrix($matrix) {
  echo "<table border=1>";
  $n=sizeof($matrix);

  for ( $x = 1; $x <= $n; $x++) {
    echo "<tr>";
    for ( $y = 1; $y <= $n; $y++) {
      if (($x % 2)==1) {
        if (($y % 2)==1) {
          echo "<td bgcolor=#999999>".$matrix[$x][$y]."</td>";
        }
        else {
          echo "<td bgcolor=#FFFFFF>".$matrix[$x][$y]."</td>";
        }
      }
      else {
        if (($y % 2)==0) {
          echo "<td bgcolor=#999999>".$matrix[$x][$y]."</td>";
        }
```

```
        else {
          echo "<td bgcolor=#FFFFFF>".$matrix[$x][$y]."</td>";
        }
      }
    }
    echo "</tr>";
  }
  echo "</table>";
 } //end function



function makeunitcol ($matrix,$colnumber,$entrytomakezero) {
  $n=sizeof($matrix);

  for ($x=1; $x<=$n; $x++) {
    $matrix[$x][$colnumber]=0;
  }

  if ($entrytomakezero>0) {
    $matrix[round($entrytomakezero)][$colnumber]=1;
  }
  return $matrix;
}

function makeunitrow ($matrix,$rownumber,$entrytomakezero) {
  $n=sizeof($matrix);

  for ($y=1; $y<=$n; $y++) {
    $matrix[$rownumber][$y]=0;
  }

  if ($entrytomakezero>0) {
    $matrix[$rownumber][round($entrytomakezero)]=1;
  }
  return $matrix;
}


function columnproj ($matrix) {
  $n=sizeof($matrix);
  for ( $y = 1; $y <= $n; $y++) {
    $imax=1;
    $entrymax=$matrix[1][$y];
    for ( $x = 1; $x <= $n; $x++) {
      if ($matrix[$x][$y]>$entrymax) {
        $imax = $x;
        $entrymax= $matrix[$x][$y];
      }
```

```
    }
    $matrix=makeunitcol($matrix,$y,$imax);
  }
  RETURN $matrix;
} //end function

function rowproj ($matrix) {
  $n=sizeof($matrix);
  for ( $x = 1; $x <= $n; $x++) {
    $imax=1;
    $entrymax=$matrix[$x][1];
    for ( $y = 1; $y <= $n; $y++) {
      if ($matrix[$x][$y]>$entrymax) {
        $imax = $y;
        $entrymax= $matrix[$x][$y];
      }
    }
    $matrix=makeunitrow($matrix,$x,$imax);
  }
  RETURN $matrix;
} //end function

function postiveslopeproj ($matrix) {

  $v1[1]=$matrix[1][1];
  $v2[1]=$matrix[2][1];$v2[2]=$matrix[1][2];
  $v3[1]=$matrix[3][1];$v3[2]=$matrix[2][2];$v3[3]=$matrix[1][3];
  $v4[1]=$matrix[4][1];$v4[2]=$matrix[3][2];$v4[3]=$matrix[2][3];
  $v4[4]=$matrix[1][4];
  $v5[1]=$matrix[5][1];$v5[2]=$matrix[4][2];$v5[3]=$matrix[3][3];
  $v5[4]=$matrix[2][4];$v5[5]=$matrix[1][5];
  $v6[1]=$matrix[6][1];$v6[2]=$matrix[5][2];$v6[3]=$matrix[4][3];
  $v6[4]=$matrix[3][4];$v6[5]=$matrix[2][5];$v6[6]=$matrix[1][6];
  $v7[1]=$matrix[7][1];$v7[2]=$matrix[6][2];$v7[3]=$matrix[5][3];
  $v7[4]=$matrix[4][4];$v7[5]=$matrix[3][5];$v7[6]=$matrix[2][6];
  $v7[7]=$matrix[1][7];
  $v8[1]=$matrix[8][1];$v8[2]=$matrix[7][2];$v8[3]=$matrix[6][3];
  $v8[4]=$matrix[5][4];$v8[5]=$matrix[4][5];$v8[6]=$matrix[3][6];
  $v8[7]=$matrix[2][7];$v8[8]=$matrix[1][8];
  $v9[1]=$matrix[8][2];$v9[2]=$matrix[7][3];$v9[3]=$matrix[6][4];
  $v9[4]=$matrix[5][5];$v9[5]=$matrix[4][6];$v9[6]=$matrix[3][7];
  $v9[7]=$matrix[2][8];
  $v10[1]=$matrix[8][3];$v10[2]=$matrix[7][4];$v10[3]=$matrix[6][5];
  $v10[4]=$matrix[5][6];$v10[5]=$matrix[4][7];$v10[6]=$matrix[3][8];
  $v11[1]=$matrix[8][4];$v11[2]=$matrix[7][5];$v11[3]=$matrix[6][6];
  $v11[4]=$matrix[5][7];$v11[5]=$matrix[4][8];
  $v12[1]=$matrix[8][5];$v12[2]=$matrix[7][6];$v12[3]=$matrix[6][7];
  $v12[4]=$matrix[5][8];
  $v13[1]=$matrix[8][6];$v13[2]=$matrix[7][7];$v13[3]=$matrix[6][8];
```

```
$v14[1]=$matrix[8][7];$v14[2]=$matrix[7][8];
$v15[1]=$matrix[8][8];

if ($v1[1]>=0.5) {
  $matrix[1][1]=1;
}
else {
  $matrix[1][1]=0;
}

$diagnames = array('v2','v3','v4','v5','v6','v7','v8','v9','v10','v11',
  'v12','v13','v14');
$unitnames= array('w2','w3','w4','w5','w6','w7','w8','w9','w10','w11',
  'w12','w13','w14');
for ($outsidecounter=2; $outsidecounter<=8; $outsidecounter++) {
  $leftsum=0;
  $rightsum=0;
  ${$unitnames[$outsidecounter-2]}=
    unitproj(${$diagnames[$outsidecounter-2]},$outsidecounter);
  for ($insidecounter=1; $insidecounter<=$outsidecounter; $insidecounter++) {
    $leftsum=$leftsum+pow(${$diagnames[$outsidecounter-2]}[$insidecounter],2);
    $rightsum=$rightsum+pow(${$diagnames[$outsidecounter-2]}[$insidecounter]-
      ${$unitnames[$outsidecounter-2]}[$insidecounter],2);
  }
  //echo " LS-> ".$leftsum." <-RS-> ".$rightsum." <- ";
  if (sqrt($leftsum) < sqrt($rightsum)) {
    //echo " All Zeros ";
    for ($num1=$outsidecounter; $num1>=1; $num1--) {
      for ($num2=1; $num2<=$outsidecounter; $num2++) {
        if (($num1+$num2)==($outsidecounter+1)) {
          $matrix[$num1][$num2]=0;
        }
      }
    }
  }
  else {
    //echo " Unit Vector ";
    $unitvectorcounter=0;
    for ($num1=$outsidecounter; $num1>=1; $num1--) {
      for ($num2=1; $num2<=$outsidecounter; $num2++) {
        if (($num1+$num2)==($outsidecounter+1)) {
          $unitvectorcounter++;
          $matrix[$num1][$num2]=
            ${$unitnames[$outsidecounter-2]}[$unitvectorcounter];
        }
      }
    }
  }
}
```

```
  // ***************** NOW WE do the remaining 7 diagonals *****************
  if ($v15[1]>=0.5) {
    $matrix[8][8]=1;
  }
  else {
    $matrix[8][8]=0;
  }

  for ($outsidecounter=14; $outsidecounter>=9; $outsidecounter--) {
    $leftsum=0;
    $rightsum=0;
    ${$unitnames[$outsidecounter-2]}=
      unitproj(${$diagnames[$outsidecounter-2]},(16-$outsidecounter));
    for ($insidecounter=1; $insidecounter<=(16-$outsidecounter); $insidecounter++) {
      $leftsum=$leftsum+pow(${$diagnames[$outsidecounter-2]}[$insidecounter],2);
      $rightsum=$rightsum+pow(${$diagnames[$outsidecounter-2]}[$insidecounter]-
        ${$unitnames[$outsidecounter-2]}[$insidecounter],2);
    }

    if (sqrt($leftsum) < sqrt($rightsum)) {
      //echo " All Zeros ";
      for ($num1=1; $num1<=8; $num1++) {
        for ($num2=1; $num2<=8; $num2++) {
          if (($num1+$num2)==($outsidecounter+1)) {
            $matrix[$num2][$num1]=0;
          }
        }
      }
    }
    else {
      //echo " Unit Vector ";
      $unitvectorcounter=0;
      for ($num1=1; $num1<=8; $num1++) {
        for ($num2=1; $num2<=8; $num2++) {
          if (($num1+$num2)==($outsidecounter+1)) {
            $unitvectorcounter++;
            $matrix[$num2][$num1]=
              ${$unitnames[$outsidecounter-2]}[$unitvectorcounter];
          }
        }
      }
    }
  }

RETURN $matrix;
} //end function
```

```
function clockwiserotation ($matrix) {
  for ($x=1; $x<=8; $x++) {
    for ($y=1; $y<=8; $y++) {
      $newmatrix[$x][$y]=$matrix[9-$y][$x];
    }
  }

  RETURN $newmatrix;
} //end function


function counterclockwiserotation ($matrix) {
  for ($x=1; $x<=8; $x++) {
    for ($y=1; $y<=8; $y++) {
      $newmatrix[9-$y][$x]=$matrix[$x][$y];
    }
  }

  RETURN $newmatrix;
} //end function


function unitproj ($v,$size) {
  $imax=1;
  $entrymax = $v[1];
  for ($i=2; $i<=$size; $i++) {
    if ($v[$i]>$entrymax) {
      $imax = $i;
      $entrymax =$v[$i];
    }
  }
  RETURN makeunit($imax,$size);
}

function makeunit ($i,$size) {
  for ($x=1; $x<=$size; $x++) {
    $v[$x]=0;
  }

  if ($i>0) {
    $v[round($i)]=1;
  }
  return $v;
}

function addcubes($M1, $M2) {
  for ($i=1; $i<=8; $i++) {
    for ($j=1; $j<=8; $j++) {
      $S[$i][$j]=$M1[$i][$j] + $M2[$i][$j];
```

```
    }
  }

  return $S;
}

function scalmultcube ($alpha, $T) {
  for ($i=1; $i<=8; $i++) {
    for ($j=1; $j<=8; $j++) {
      $S[$i][$j]=$alpha*$T[$i][$j];
    }
  }

  return $S;
} // end function


function diagproj ($T1, $T2, $T3, $T4) {
  $T=addcubes($T1,$T2);
  $T=addcubes($T,$T3);
  $T=addcubes($T,$T4);
  $T=scalmultcube(0.25,$T);
  return $T;
}

function new1 ($T1, $T2, $T3, $T4) {
  $PD=diagproj($T1,$T2,$T3,$T4);
  $S1 = scalmultcube(2,$PD);
  $S2 = scalmultcube(-1,$T1);
  $S3 = addcubes($S1,$S2);
  $S4 = columnproj($S3);
  $S5 = addcubes($T1,scalmultcube(-1,$PD));
  $S = addcubes($S4,$S5);
  return $S;
}

function new2 ($T1, $T2, $T3, $T4) {
  $PD=diagproj($T1,$T2,$T3,$T4);
  $S1 = scalmultcube(2,$PD);
  $S2 = scalmultcube(-1,$T2);
  $S3 = addcubes($S1,$S2);
  $S4 = rowproj($S3);
  $S5 = addcubes($T2,scalmultcube(-1,$PD));
  $S = addcubes($S4,$S5);
  return $S;
}

function new3 ($T1, $T2, $T3, $T4) {
  $PD=diagproj($T1,$T2,$T3,$T4);
```

```
  $S1 = scalmultcube(2,$PD);
  $S2 = scalmultcube(-1,$T3);
  $S3 = addcubes($S1,$S2);
  $S4 = postiveslopeproj($S3);
  $S5 = addcubes($T3,scalmultcube(-1,$PD));
  $S = addcubes($S4,$S5);
  return $S;
}

function new4 ($T1, $T2, $T3, $T4) {
  global $origS;
  $PD=diagproj($T1,$T2,$T3,$T4);
  $S1 = scalmultcube(2,$PD);
  $S2 = scalmultcube(-1,$T4);
  $S3 = addcubes($S1,$S2);
  $S4 = counterclockwiserotation(postiveslopeproj(clockwiserotation($S3)));
  $S5 = addcubes($T4,scalmultcube(-1,$PD));
  $S = addcubes($S4,$S5);
  return $S;
}

$origMatrix[1][1]=$_POST['11'];
$origMatrix[1][2]=$_POST['12'];
$origMatrix[1][3]=$_POST['13'];
$origMatrix[1][4]=$_POST['14'];
$origMatrix[1][5]=$_POST['15'];
$origMatrix[1][6]=$_POST['16'];
$origMatrix[1][7]=$_POST['17'];
$origMatrix[1][8]=$_POST['18'];

$origMatrix[2][1]=$_POST['21'];
$origMatrix[2][2]=$_POST['22'];
$origMatrix[2][3]=$_POST['23'];
$origMatrix[2][4]=$_POST['24'];
$origMatrix[2][5]=$_POST['25'];
$origMatrix[2][6]=$_POST['26'];
$origMatrix[2][7]=$_POST['27'];
$origMatrix[2][8]=$_POST['28'];

$origMatrix[3][1]=$_POST['31'];
$origMatrix[3][2]=$_POST['32'];
$origMatrix[3][3]=$_POST['33'];
$origMatrix[3][4]=$_POST['34'];
$origMatrix[3][5]=$_POST['35'];
$origMatrix[3][6]=$_POST['36'];
$origMatrix[3][7]=$_POST['37'];
$origMatrix[3][8]=$_POST['38'];

$origMatrix[4][1]=$_POST['41'];
```

```
$origMatrix[4][2]=$_POST['42'];
$origMatrix[4][3]=$_POST['43'];
$origMatrix[4][4]=$_POST['44'];
$origMatrix[4][5]=$_POST['45'];
$origMatrix[4][6]=$_POST['46'];
$origMatrix[4][7]=$_POST['47'];
$origMatrix[4][8]=$_POST['48'];

$origMatrix[5][1]=$_POST['51'];
$origMatrix[5][2]=$_POST['52'];
$origMatrix[5][3]=$_POST['53'];
$origMatrix[5][4]=$_POST['54'];
$origMatrix[5][5]=$_POST['55'];
$origMatrix[5][6]=$_POST['56'];
$origMatrix[5][7]=$_POST['57'];
$origMatrix[5][8]=$_POST['58'];

$origMatrix[6][1]=$_POST['61'];
$origMatrix[6][2]=$_POST['62'];
$origMatrix[6][3]=$_POST['63'];
$origMatrix[6][4]=$_POST['64'];
$origMatrix[6][5]=$_POST['65'];
$origMatrix[6][6]=$_POST['66'];
$origMatrix[6][7]=$_POST['67'];
$origMatrix[6][8]=$_POST['68'];

$origMatrix[7][1]=$_POST['71'];
$origMatrix[7][2]=$_POST['72'];
$origMatrix[7][3]=$_POST['73'];
$origMatrix[7][4]=$_POST['74'];
$origMatrix[7][5]=$_POST['75'];
$origMatrix[7][6]=$_POST['76'];
$origMatrix[7][7]=$_POST['77'];
$origMatrix[7][8]=$_POST['78'];

$origMatrix[8][1]=$_POST['81'];
$origMatrix[8][2]=$_POST['82'];
$origMatrix[8][3]=$_POST['83'];
$origMatrix[8][4]=$_POST['84'];
$origMatrix[8][5]=$_POST['85'];
$origMatrix[8][6]=$_POST['86'];
$origMatrix[8][7]=$_POST['87'];
$origMatrix[8][8]=$_POST['88'];
$iteration=20000;


echo "<table border=2 cellpadding=5 cellspacing=5>";
echo "<tr><th>Iteration</th><th>Chess Board</th><th>";
echo "Current Matrix";
```

```
echo "</th>";
echo "</tr>";
echo "<tr><td>0</td><td>";
PrintChessMatrix($origMatrix);
echo "</td><td>";
PrintMatrix($origMatrix);
echo"</td>";
$T1old=$origMatrix;
$T2old=$origMatrix;
$T3old=$origMatrix;;
$T4old=$origMatrix;
$Tadaold=diagproj($T1old,$T2old,$T3old,$T4old);


$solved=0;
$i=0;

while (!$solved) {
  $T1=new1($T1old,$T2old,$T3old,$T4old);
  $T2=new2($T1old,$T2old,$T3old,$T4old);
  $T3=new3($T1old,$T2old,$T3old,$T4old);
  $T4=new4($T1old,$T2old,$T3old,$T4old);
  $Tada=diagproj($T1,$T2,$T3,$T4);

  if (checkqueens($Tada)) {
    $solved=1;
  }

  echo "<tr><td>$i</td><td>";
  PrintChessMatrix($Tada);
  echo "</td><td>";
  PrintMatrix($Tada);
  echo "</td>";
  echo "</tr>";
  $i++;

  $T1old = $T1;
  $T2old = $T2;
  $T3old = $T3;
  $T4old = $T4;
}
echo "<big>Success!  Solved in ".($i-1). " iterations.</big>";
?>
```

Appendix B

Sudoku PHP Code

We first have sudokufunctions.php, then our sudoku code.

```
<?php

// This code is prototype and is provided as is under the GNU General Public License
// Author: Jason Schaad August 11, 2010

function makeunit ($i) {
  // returns a vector (array)
  // $i should be a float

  $v[1]=0;$v[2]=0;$v[3]=0;$v[4]=0;$v[5]=0;$v[6]=0;$v[7]=0;$v[8]=0;$v[9]=0;
  // easier way to initialize an array
  if ($i>0) {
    $v[round($i)]=1;
  }
  return $v;
}


function unitproj ($v)  {
  // input $v is a vector
  // finds biggest component and makes 1 and the rest 0
  //e.g. [0.4, .6, .7, .8, 0, 0, 0, 0, 0] = [0,0,0,1,0,0,0,0,0]
  // returns a vector (array)
  // relies on makeunit

  $imax=1;
  $entrymax = $v[1];
  for ($i=2; $i<=9; $i++) {
    if ($v[$i]>$entrymax) {
      $imax = $i;
      $entrymax =$v[$i];
    }
  }
  return makeunit($imax);
}

function addcubes ($T1, $T2) {
  // add 2 9x9x9 cubes
```

```
  // returns the "sum cube"

  for ($i=1; $i<=9; $i++) {
    for ($j=1; $j<=9; $j++) {
      for ($k=1; $k<=9; $k++) {
          $S[$i][$j][$k]=$T1[$i][$j][$k] + $T2[$i][$j][$k];
      }
    }
  }

return $S;


}

function printcube($cube) {
  // this function prints the cube to the screen
  // not included in unit tests as it is never used in my actual sudoku code
  // it was used for debugging when I was creating the code
  for ($x=1; $x<=9; $x++) {
    for ($y=1; $y<=9; $y++) {
      for ($z=1; $z<=9; $z++) {
        echo "(".$x.",".$y.",".$z.") =".$cube[$x][$y][$z]."<br>";
      }
    }
  }
}

function printarray($square) {
  // this function prints a 2 dimensional array to the screen (our "flattened" Sudoku)
  // there is no need for a unit test for this code as it outputs to the screen only
  for ($x=1; $x<=9; $x++) {
    for ($y=1; $y<=9; $y++) {
      echo $square[$x][$y]." ";
    }
  echo "<br />";
  }
}

function columnproj ($Tcube) {
  // finds the biggest entry in every column of our sudoku cube
  // and makes it into a unit vector - returns a cube
  for ($j=1; $j<=9; $j++) {
    for ($k=1; $k<=9; $k++) {
      for ($i=1; $i<=9; $i++)
        {
          $tempv[$i]=$Tcube[$i][$j][$k];
        }
      $tempw=unitproj($tempv);
```

```
      for ($i=1; $i<=9; $i++) {
        $cprojTcube[$i][$j][$k]=$tempw[$i];
      }
    }
  }
return $cprojTcube;
}

function rowproj ($Tcube) {
  // finds the biggest entry in every row of our sudoku cube
  // and makes it into a unit vector which it returns
  for ($i=1; $i<=9; $i++) {
    for ($k=1; $k<=9; $k++) {
      for ($j=1; $j<=9; $j++) {
        $tempv[$j]=$Tcube[$i][$j][$k];
      }
      $tempw=unitproj($tempv);
      for ($j=1; $j<=9; $j++) {
        $rprojTcube[$i][$j][$k]=$tempw[$j];
      }
    }
  }
return $rprojTcube;
}

function meetproj ($Tcube) {
  // This function calculates a "unit vector" based on the
  // meeting rooms (the 3 x 3 rooms) it uses meetpro2 function
  // to do alot of the work (loops are good.. and so is re-using your code)
  // initialize mprojTcube to have all 0's

  for ($i=1; $i<=9; $i++) {
      for ($j=1; $j<=9; $j++) {
        for ($k=1; $k<=9; $k++) {
          $mprojTcube[$i][$j][$k]=0;
        }
      }
  }

  // 1-1 subcube
  $mprojTcube=meetproj2($mprojTcube,$Tcube,1,1);
  // 1-2 subcube
  $mprojTcube=meetproj2($mprojTcube,$Tcube,4,1);
  // 1-3 subcube
  $mprojTcube=meetproj2($mprojTcube,$Tcube,7,1);
  // 2-1 subcube
  $mprojTcube=meetproj2($mprojTcube,$Tcube,1,4);
  // 2-2 subcube
  $mprojTcube=meetproj2($mprojTcube,$Tcube,4,4);
```

```
  // 2-3 subcube
  $mprojTcube=meetproj2($mprojTcube,$Tcube,7,4);
  // 3-1 subcube
  $mprojTcube=meetproj2($mprojTcube,$Tcube,1,7);
  // 3-2 subcube
  $mprojTcube=meetproj2($mprojTcube,$Tcube,4,7);
  // 3-3 subcube
  $mprojTcube=meetproj2($mprojTcube,$Tcube,7,7);

  return $mprojTcube;
}

function meetproj2($mprojTcube,$Tcube,$index1,$index2) {
  // This function calculates a "unit vector" based on the
  // meeting rooms (the 3 x 3 rooms)
  // this is coded using loops and the variables $index1 and $index
  // 2 help get the indices correct for the funny "unit vector"

  for ($k=1; $k<=9; $k++) {
    for ($j=0; $j<3; $j++) {
      for ($i=0;$i<3; $i++) {
        $tempv[1+$i+3*$j]=$Tcube[$index2+$j][$index1+$i][$k];
      }
    }
    $tempw = unitproj($tempv);
    for ($j=0; $j<3; $j++) {
      for ($i=0;$i<3; $i++) {
        $mprojTcube[$index2+$j][$index1+$i][$k]=$tempw[1+$i+3*$j];
      }
    }
  }
  return $mprojTcube;
}


function scalmultcube ($alpha, $T) {
  // Takes as input a cube and a scalar and multiples them - output is a cube
  for ($i=1; $i<=9; $i++) {
    for ($j=1; $j<=9; $j++) {
      for ($k=1; $k<=9; $k++) {
        $S[$i][$j][$k]=$alpha*$T[$i][$j][$k];
      }
    }
  }

return $S;

} // end function
```

```
function thesame ($T1, $T2) {
  // compares two arrays - if they are the same return true, otherwise false
  $counter=0;
  for ($i=1; $i<=9; $i++) {
    for ($j=1; $j<=9; $j++) {
      if ($T1[$i][$j]!=$T2[$i][$j]) {
        $counter++;

      }
    }
  }
  if ($counter>0) {
    return FALSE;
  }
  else {
    return TRUE;
  }
}

function givenproj ($Tcube, $S) {
  // this function uses the original sudoku that was given to calculate
  // a projection (our given Sudoku constraint
  // - which is just a 9 by 9 array not a cube).  Returns a projected cube.

  $gprojTcube=$Tcube;
  for ($i=1; $i<=9; $i++) {
    for ($j=1; $j<=9; $j++) {
      if ($S[$i][$j]>0) {
        $tempw=makeunit($S[$i][$j]);
        for ($k=1; $k<=9; $k++) {
          $gprojTcube[$i][$j][$k]=$tempw[$k];
        }
      }
      if ($S[$i][$j]==0) {
        for ($k=1; $k<=9; $k++) {
          $tempv[$k]=$Tcube[$i][$j][$k];
        }
        $tempw=unitproj($tempv);
        for ($k=1; $k<=9; $k++)  {
          $gprojTcube[$i][$j][$k]=$tempw[$k];
        }
      }
    }
  }
  return $gprojTcube;
} // end function

function makearray ($Tcube) {
  // flattens a cube (9x9x9) into an array (9x9) so we can print it to the screen
```

```
  for ($i=1; $i<=9; $i++) {
    for ($j=1; $j<=9; $j++) {
      $temp=1;
      $tempval=$Tcube[$i][$j][1];
      for ($k=2; $k<=9; $k++) {
        if ($Tcube[$i][$j][$k]>$tempval) {
          $temp=$k;
          $tempval=$Tcube[$i][$j][$k];
        }
      }
      $A[$i][$j]=$temp;
    }
  }
  return $A;
}

function stuckinloop ($i) {
  // no need to test this function, it is too simple.
  if ($i>100000) {
    throw new Exception('We are stuck in a loop with over 10000 iterations.');
  }


}

function diagproj ($T1, $T2, $T3, $T4) {
  // take four cubes and computes the diagonal
  $T=addcubes($T1,$T2);
  $T=addcubes($T,$T3);
  $T=addcubes($T,$T4);
  $T=scalmultcube(0.25,$T);
  return $T;
}

function new1 ($T1, $T2, $T3, $T4)  {
  // creates a new cube with the column projection
  $PD=diagproj($T1,$T2,$T3,$T4);
  $S1 = scalmultcube(2,$PD);
  $S2 = scalmultcube(-1,$T1);
  $S3 = addcubes($S1,$S2);
  $S4 = columnproj($S3);
  $S5 = addcubes($T1,scalmultcube(-1,$PD));
  $S = addcubes($S4,$S5);
  return $S;
}

function new2 ($T1, $T2, $T3, $T4)  {
  // creates a new cube the row projection
  $PD=diagproj($T1,$T2,$T3,$T4);
```

```
  $S1 = scalmultcube(2,$PD);
  $S2 = scalmultcube(-1,$T2);
  $S3 = addcubes($S1,$S2);
  $S4 = rowproj($S3);
  $S5 = addcubes($T2,scalmultcube(-1,$PD));
  $S = addcubes($S4,$S5);
  return $S;
}

function new3 ($T1, $T2, $T3, $T4) {
  // creates a new cube with the meet projection
  $PD=diagproj($T1,$T2,$T3,$T4);
  $S1 = scalmultcube(2,$PD);
  $S2 = scalmultcube(-1,$T3);
  $S3 = addcubes($S1,$S2);
  $S4 = meetproj($S3);
  $S5 = addcubes($T3,scalmultcube(-1,$PD));
  $S = addcubes($S4,$S5);
  return $S;
}

function new4 ($T1, $T2, $T3, $T4) {
  // creates a new cube with the given projection
  global $origS;
  $PD=diagproj($T1,$T2,$T3,$T4);
  $S1 = scalmultcube(2,$PD);
  $S2 = scalmultcube(-1,$T4);
  $S3 = addcubes($S1,$S2);
  $S4 = givenproj($S3,$origS);
  $S5 = addcubes($T4,scalmultcube(-1,$PD));
  $S = addcubes($S4,$S5);
  return $S;
}

function meetproj22 ($Tcube) {
  // This function calculates a "unit vector" based on the
  // meeting rooms (the 3 x 3 rooms)
  // I coded it differently (using more loops),
  // but it made it much harder to read.  We can see where an
  // actual meeting room is located in our cube this way!
  // 1-1 subcube
  for ($k=1; $k<=9; $k++) {
    $tempv[1] = $Tcube[1][1][$k];
    $tempv[2] = $Tcube[1][2][$k];
    $tempv[3] = $Tcube[1][3][$k];
    $tempv[4] = $Tcube[2][1][$k];
    $tempv[5] = $Tcube[2][2][$k];
    $tempv[6] = $Tcube[2][3][$k];
    $tempv[7] = $Tcube[3][1][$k];
```

```
    $tempv[8] = $Tcube[3][2][$k];
    $tempv[9] = $Tcube[3][3][$k];
    $tempw = unitproj($tempv);
    $mprojTcube[1][1][$k] = $tempw[1];
    $mprojTcube[1][2][$k] = $tempw[2];
    $mprojTcube[1][3][$k] = $tempw[3];
    $mprojTcube[2][1][$k] = $tempw[4];
    $mprojTcube[2][2][$k] = $tempw[5];
    $mprojTcube[2][3][$k] = $tempw[6];
    $mprojTcube[3][1][$k] = $tempw[7];
    $mprojTcube[3][2][$k] = $tempw[8];
    $mprojTcube[3][3][$k] = $tempw[9];
}
// 1-2 subcube
for ($k=1; $k<=9; $k++) {
    $tempv[1] = $Tcube[1][4][$k];
    $tempv[2] = $Tcube[1][5][$k];
    $tempv[3] = $Tcube[1][6][$k];
    $tempv[4] = $Tcube[2][4][$k];
    $tempv[5] = $Tcube[2][5][$k];
    $tempv[6] = $Tcube[2][6][$k];
    $tempv[7] = $Tcube[3][4][$k];
    $tempv[8] = $Tcube[3][5][$k];
    $tempv[9] = $Tcube[3][6][$k];
    $tempw = unitproj($tempv);
    $mprojTcube[1][4][$k] = $tempw[1];
    $mprojTcube[1][5][$k] = $tempw[2];
    $mprojTcube[1][6][$k] = $tempw[3];
    $mprojTcube[2][4][$k] = $tempw[4];
    $mprojTcube[2][5][$k] = $tempw[5];
    $mprojTcube[2][6][$k] = $tempw[6];
    $mprojTcube[3][4][$k] = $tempw[7];
    $mprojTcube[3][5][$k] = $tempw[8];
    $mprojTcube[3][6][$k] = $tempw[9];
}
// 1-3 subcube
for ($k=1; $k<=9; $k++) {
    $tempv[1] = $Tcube[1][7][$k];
    $tempv[2] = $Tcube[1][8][$k];
    $tempv[3] = $Tcube[1][9][$k];
    $tempv[4] = $Tcube[2][7][$k];
    $tempv[5] = $Tcube[2][8][$k];
    $tempv[6] = $Tcube[2][9][$k];
    $tempv[7] = $Tcube[3][7][$k];
    $tempv[8] = $Tcube[3][8][$k];
    $tempv[9] = $Tcube[3][9][$k];
    $tempw = unitproj($tempv);
    $mprojTcube[1][7][$k] = $tempw[1];
    $mprojTcube[1][8][$k] = $tempw[2];
```

```
    $mprojTcube[1][9][$k] = $tempw[3];
    $mprojTcube[2][7][$k] = $tempw[4];
    $mprojTcube[2][8][$k] = $tempw[5];
    $mprojTcube[2][9][$k] = $tempw[6];
    $mprojTcube[3][7][$k] = $tempw[7];
    $mprojTcube[3][8][$k] = $tempw[8];
    $mprojTcube[3][9][$k] = $tempw[9];
  }
  // 2-1 subcube
  for ($k=1; $k<=9; $k++) {
    $tempv[1] = $Tcube[4][1][$k];
    $tempv[2] = $Tcube[4][2][$k];
    $tempv[3] = $Tcube[4][3][$k];
    $tempv[4] = $Tcube[5][1][$k];
    $tempv[5] = $Tcube[5][2][$k];
    $tempv[6] = $Tcube[5][3][$k];
    $tempv[7] = $Tcube[6][1][$k];
    $tempv[8] = $Tcube[6][2][$k];
    $tempv[9] = $Tcube[6][3][$k];
    $tempw = unitproj($tempv);
    $mprojTcube[4][1][$k] = $tempw[1];
    $mprojTcube[4][2][$k] = $tempw[2];
    $mprojTcube[4][3][$k] = $tempw[3];
    $mprojTcube[5][1][$k] = $tempw[4];
    $mprojTcube[5][2][$k] = $tempw[5];
    $mprojTcube[5][3][$k] = $tempw[6];
    $mprojTcube[6][1][$k] = $tempw[7];
    $mprojTcube[6][2][$k] = $tempw[8];
    $mprojTcube[6][3][$k] = $tempw[9];
  }
  // 2-2 subcube
  for ($k=1; $k<=9; $k++) {
    $tempv[1] = $Tcube[4][4][$k];
    $tempv[2] = $Tcube[4][5][$k];
    $tempv[3] = $Tcube[4][6][$k];
    $tempv[4] = $Tcube[5][4][$k];
    $tempv[5] = $Tcube[5][5][$k];
    $tempv[6] = $Tcube[5][6][$k];
    $tempv[7] = $Tcube[6][4][$k];
    $tempv[8] = $Tcube[6][5][$k];
    $tempv[9] = $Tcube[6][6][$k];
    $tempw = unitproj($tempv);
    $mprojTcube[4][4][$k] = $tempw[1];
    $mprojTcube[4][5][$k] = $tempw[2];
    $mprojTcube[4][6][$k] = $tempw[3];
    $mprojTcube[5][4][$k] = $tempw[4];
    $mprojTcube[5][5][$k] = $tempw[5];
    $mprojTcube[5][6][$k] = $tempw[6];
    $mprojTcube[6][4][$k] = $tempw[7];
```

```
  $mprojTcube[6][5][$k] = $tempw[8];
  $mprojTcube[6][6][$k] = $tempw[9];
}
// 2-3 subcube
for ($k=1; $k<=9; $k++) {
  $tempv[1] = $Tcube[4][7][$k];
  $tempv[2] = $Tcube[4][8][$k];
  $tempv[3] = $Tcube[4][9][$k];
  $tempv[4] = $Tcube[5][7][$k];
  $tempv[5] = $Tcube[5][8][$k];
  $tempv[6] = $Tcube[5][9][$k];
  $tempv[7] = $Tcube[6][7][$k];
  $tempv[8] = $Tcube[6][8][$k];
  $tempv[9] = $Tcube[6][9][$k];
  $tempw = unitproj($tempv);
  $mprojTcube[4][7][$k] = $tempw[1];
  $mprojTcube[4][8][$k] = $tempw[2];
  $mprojTcube[4][9][$k] = $tempw[3];
  $mprojTcube[5][7][$k] = $tempw[4];
  $mprojTcube[5][8][$k] = $tempw[5];
  $mprojTcube[5][9][$k] = $tempw[6];
  $mprojTcube[6][7][$k] = $tempw[7];
  $mprojTcube[6][8][$k] = $tempw[8];
  $mprojTcube[6][9][$k] = $tempw[9];
}
// 3-1 subcube
for ($k=1; $k<=9; $k++) {
  $tempv[1] = $Tcube[7][1][$k];
  $tempv[2] = $Tcube[7][2][$k];
  $tempv[3] = $Tcube[7][3][$k];
  $tempv[4] = $Tcube[8][1][$k];
  $tempv[5] = $Tcube[8][2][$k];
  $tempv[6] = $Tcube[8][3][$k];
  $tempv[7] = $Tcube[9][1][$k];
  $tempv[8] = $Tcube[9][2][$k];
  $tempv[9] = $Tcube[9][3][$k];
  $tempw = unitproj($tempv);
  $mprojTcube[7][1][$k] = $tempw[1];
  $mprojTcube[7][2][$k] = $tempw[2];
  $mprojTcube[7][3][$k] = $tempw[3];
  $mprojTcube[8][1][$k] = $tempw[4];
  $mprojTcube[8][2][$k] = $tempw[5];
  $mprojTcube[8][3][$k] = $tempw[6];
  $mprojTcube[9][1][$k] = $tempw[7];
  $mprojTcube[9][2][$k] = $tempw[8];
  $mprojTcube[9][3][$k] = $tempw[9];
}
// 3-2 subcube
for ($k=1; $k<=9; $k++) {
```

```
    $tempv[1] = $Tcube[7][4][$k];
    $tempv[2] = $Tcube[7][5][$k];
    $tempv[3] = $Tcube[7][6][$k];
    $tempv[4] = $Tcube[8][4][$k];
    $tempv[5] = $Tcube[8][5][$k];
    $tempv[6] = $Tcube[8][6][$k];
    $tempv[7] = $Tcube[9][4][$k];
    $tempv[8] = $Tcube[9][5][$k];
    $tempv[9] = $Tcube[9][6][$k];
    $tempw = unitproj($tempv);
    $mprojTcube[7][4][$k] = $tempw[1];
    $mprojTcube[7][5][$k] = $tempw[2];
    $mprojTcube[7][6][$k] = $tempw[3];
    $mprojTcube[8][4][$k] = $tempw[4];
    $mprojTcube[8][5][$k] = $tempw[5];
    $mprojTcube[8][6][$k] = $tempw[6];
    $mprojTcube[9][4][$k] = $tempw[7];
    $mprojTcube[9][5][$k] = $tempw[8];
    $mprojTcube[9][6][$k] = $tempw[9];
  }
  // 3-3 subcube
  for ($k=1; $k<=9; $k++) {
    $tempv[1] = $Tcube[7][7][$k];
    $tempv[2] = $Tcube[7][8][$k];
    $tempv[3] = $Tcube[7][9][$k];
    $tempv[4] = $Tcube[8][7][$k];
    $tempv[5] = $Tcube[8][8][$k];
    $tempv[6] = $Tcube[8][9][$k];
    $tempv[7] = $Tcube[9][7][$k];
    $tempv[8] = $Tcube[9][8][$k];
    $tempv[9] = $Tcube[9][9][$k];
    $tempw = unitproj($tempv);
    $mprojTcube[7][7][$k] = $tempw[1];
    $mprojTcube[7][8][$k] = $tempw[2];
    $mprojTcube[7][9][$k] = $tempw[3];
    $mprojTcube[8][7][$k] = $tempw[4];
    $mprojTcube[8][8][$k] = $tempw[5];
    $mprojTcube[8][9][$k] = $tempw[6];
    $mprojTcube[9][7][$k] = $tempw[7];
    $mprojTcube[9][8][$k] = $tempw[8];
    $mprojTcube[9][9][$k] = $tempw[9];
  }
  return $mprojTcube;
}


function checkMeetingRoom($sudoku,$index_x, $index_y) {
  // sudoku is the 9x9 array and index1 and index2 are the starting points
  //[1][1] [1][2] [1][3]
```

```
  //[2][1] [2][2] [2][3]
  //[3][1] [3][2] [3][3]

  //$one=array (1=>1,1,1,2,2,2,3,3,3);
//  $two=array (1=>1,2,3,1,2,3,1,2,3);

  $one=array (1=>$index_y,$index_y,$index_y,($index_y+1),($index_y+1),
    ($index_y+1),($index_y+2),($index_y+2),($index_y+2));
  $two=array (1=>($index_x),($index_x+1),($index_x+2),($index_x),($index_x+1),
    ($index_x+2),($index_x),($index_x+1),($index_x+2));

  //print_r($one);
  //print_r($two);

  for ($i=1; $i<=9; $i++) {
      for ($j=1; $j<=9; $j++) {
        if ($i!=$j) {
          if ($sudoku[$one[$i]][$two[$i]]==$sudoku[$one[$j]][$two[$j]]) {
            return 0;
          }
        }
      }
  }


  return 1;
}




function isValidSudoku($original,$sudoku) {
  // checks to see if the sudoku is valid - returns true if valid, false otherwise
  // input is the original constraints and the sudoku to check

  // Check if we meet the ROW constraints (1 to 9 in every row)
  // first cehck for duplicates
  for ($k=1; $k<=9; $k++) {
    for ($i=1; $i<=9; $i++) {
      for ($j=1; $j<=9; $j++) {
        if ($i!=$j) {
          if ($sudoku[$k][$i]==$sudoku[$k][$j]) {
            //echo "Failed row constraint<br />";
            return 0;

          }
        }
      }
```

```
    }
  }

  // Check if we meet the COLUMN constraints (1 to 9 in every row)
  // first cehck for duplicates
  for ($k=1; $k<=9; $k++) {
    for ($i=1; $i<=9; $i++) {
      for ($j=1; $j<=9; $j++) {
        if ($i!=$j) {
          if ($sudoku[$i][$k]==$sudoku[$j][$k]) {
            //echo "Failed column constraint<br />";
            return 0;

          }
        }
      }
    }
  }

  //Check MEETING ROOM constraints (1 to 9 in every meeting room)
  if (!checkMeetingRoom($sudoku,1,1)) { return 0;}
  if (!checkMeetingRoom($sudoku,4,1)) { return 0;}
  if (!checkMeetingRoom($sudoku,7,1)) { return 0;}
  if (!checkMeetingRoom($sudoku,1,4)) { return 0;}
  if (!checkMeetingRoom($sudoku,4,4)) { return 0;}
  if (!checkMeetingRoom($sudoku,7,4)) { return 0;}
  if (!checkMeetingRoom($sudoku,1,7)) { return 0;}
  if (!checkMeetingRoom($sudoku,4,7)) { return 0;}
  if (!checkMeetingRoom($sudoku,7,7)) { return 0;}

  //Lastly we check to make sure we have the given constraints
  for ($i=1; $i<=9; $i++) {
    for ($j=1; $j<=9; $j++) {
      $new_array[$i][$j]=$original[$i][$j]*$sudoku[$i][$j];
      $new_array[$i][$j]=sqrt($new_array[$i][$j]);
    }
  }

  //printarray($new_array);
  //echo "<br />";
  //printarray($original);
  if (!thesame($new_array,$original)) {
    return 0;
  }


  return 1;
}
```

```
<?php

set_time_limit(0);

// Report simple running errors
error_reporting(E_ERROR | E_WARNING | E_PARSE);

// Verbose mode (shows all details)
$details=$_POST['details'];

if ($details != 1) {
  $details=0;
}

// Pick up the form variables
for ($x=1; $x<=9; $x++) {
  for($y=1; $y<=9; $y++) {
    $origS[$x][$y]=$_POST[$x.$y];
  }
}

// Display the initial Sudoku puzzle we will solve
// We also initialize a GLOBAL variable called origS
// which is our original Sudoku (which is a constraint)
echo "<table border=1>";
for ($x=1; $x<=9; $x++) {
  echo "<tr>";
  for($y=1; $y<=9; $y++) {
      echo "<td>";
      if ($origS[$x][$y]=="") {
        echo " ";
      }
      else {
        echo $origS[$x][$y];
      }
    echo "</td>";
  }
  echo "</tr>";
}

echo "</table>";

// include the functions to be used in sodoku
include('sudokufunctions.php');

// initialize the scube 9x9x9 array to all zeros
for ($x=1; $x<=9; $x++) {
  for ($y=1; $y<=9; $y++) {
    for ($z=1; $z<=9; $z++) {
```

```
      $Scube[$x][$y][$z]=0;
    }
  }
}

// Get a starting position based on the original sudoku given
// (the commented out line it when we start at random positions - for research)
for ($x=1; $x<=9; $x++) {
  for ($y=1; $y<=9; $y++) {
    $tempv=makeunit($origS[$x][$y]);
    for ($z=1; $z<=9; $z++) {
      $Scube[$x][$y][$z]=$tempv[$z];
      //$Scube[$x][$y][$z]= rand(-10000, 10000);
    }
  }
}

echo "<br /><br />";



// We now start the iteration process

// $Scube is our starting position as defined above
$T1old=$Scube;
$T2old=$Scube;
$T3old=$Scube;
$T4old=$Scube;
$Tadaold=diagproj($T1old,$T2old,$T3old,$T4old);

$zerosinarow = 0;
$i=1;
echo "working";
while (!isValidSudoku($origS,$mTada)) {
  // the stopping criteria is when there is not change
  // in the cubes for 10 iterations
  // we clone 4 cubes and make the diagonal
  $T1=new1($T1old,$T2old,$T3old,$T4old);
  $T2=new2($T1old,$T2old,$T3old,$T4old);
  $T3=new3($T1old,$T2old,$T3old,$T4old);
  $T4=new4($T1old,$T2old,$T3old,$T4old);
  $Tada=diagproj($T1,$T2,$T3,$T4);
  $mTada=makearray($Tada);
  if ($details==1) {
    echo "Current Counter ".$i,"<br>";
    printarray($mTada);
  }
  else {
    echo ".";
```

```
  }

  $T1old = $T1;
  $T2old = $T2;
  $T3old = $T3;
  $T4old = $T4;
  $mTadaold = $mTada;
  $mTadaoldmatrix = $mTadamatrix;
  $i++;


  try {
     stuckinloop($i);
   } catch (Exception $e) {
    $badsudoku = print_r($origS, true);
    error_log($e->getMessage()."\n",3, "my-errors.log");
    error_log($badsudoku."\n", 3, "my-errors.log");
    echo "Too many iterations - we are stuck.. maybe a bad Sudoku?";
    return;
  }
}

if ($details==0) {echo "<br>Solved in ".$i." iterations:<br>"; printarray($mTada);}

?>
```

Appendix C

Eckstein Calculation Details

In chapter 4, we showed that Eckstein made a numerical error in (4.95) and as a result (4.96). The matrices in fact are

$$G_{\lambda,A,B} = J_{\lambda A}(2J_{\lambda B} - \mathrm{Id}) + (\mathrm{Id} - J_{\lambda B}) = \begin{pmatrix} \frac{21}{40} & \frac{-1}{16} \\ \frac{-1}{40} & \frac{9}{16} \end{pmatrix} \tag{C.1}$$

and

$$S_{\lambda,A,B} = G_{\lambda,A,B}^{-1} - \mathrm{Id} = \begin{pmatrix} \frac{43}{47} & \frac{10}{47} \\ \frac{4}{47} & \frac{37}{47} \end{pmatrix}. \tag{C.2}$$

Below are the details of the calculation.

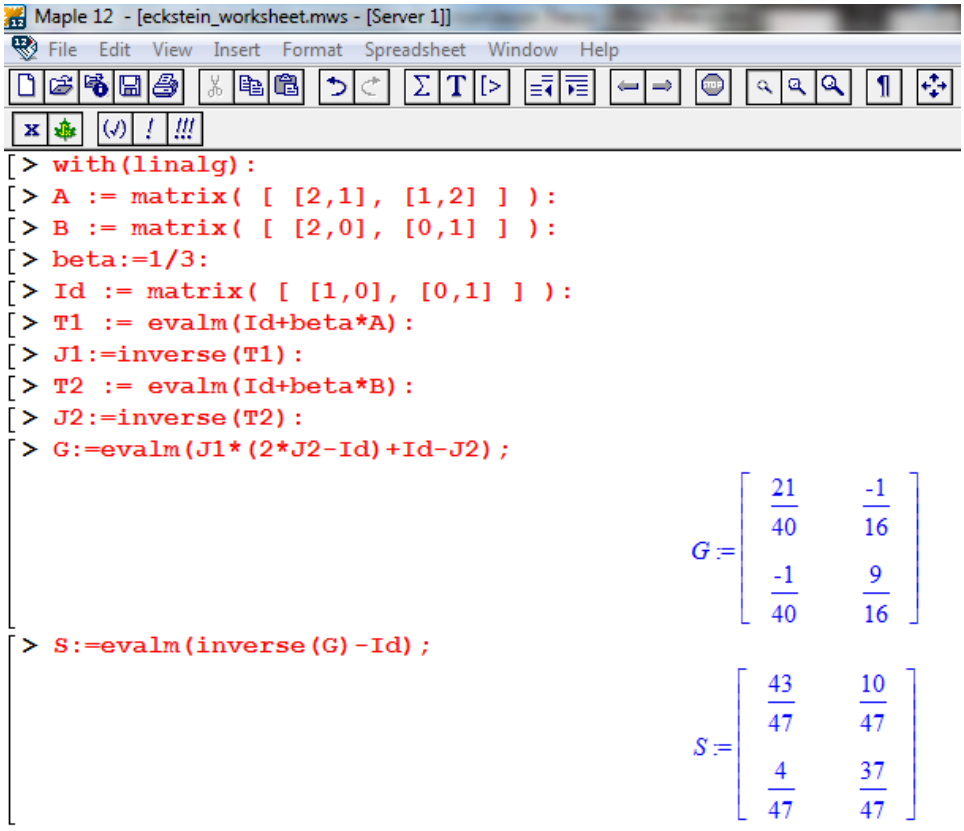


Figure C.1: Calculation Details