

Department of Mathematics

Final Project For A Graduate In Applied Math B.SC

*Douglas Rachford Method For Solving Convex and Non – Convex
Feasibility Problems*

Advisor: Assoc. Prof. Aviv Gibali

Author: Samuel Zeldin Kogan

Date: 5th March, 2023.

Acknowledgements

I want to give a special thanks to Assoc. Prof. Aviv Gibali, who is also my advisor in this project, him for his support throughout the making of this article, giving me lots of his time to assist me in finding good sources of information, and helping me navigate through the process of making this final project, in how to do it.

I am not sure I would be able to complete this project without his psychological assistance aswell, motivating me and pushing me to finish this project.

Table of Content

Abstract

1. Preliminaries

- Definitions

2. Introduction:

- History
- Idea
- Applications
- Advantages and Limitations

3. Douglas Rachford Algorithm:

- Classic Douglas Rachford
- Cyclic Douglas Rachford
- Theorems about convergence
- Algorithms used in our Numerical Experiments

4. Numerical Experiments:

- **4.1. Convex Examples:**
 - 3 Two Dimensional Lines
 - 2 Two Dimensional Balls
 - 3 Two Dimensional Balls
- **4.2. Nonconvex Example:**
 - Two Dimensional Ball and a Line using DR
 - Two Dimensional Ball and a line using Von Neumann

5. Space Product Formulation

- Diagonal Set D
- Set of sets product C
- Theorems and proof
- Model Formulation and its importance on results

6. Applications of Douglas Rachford method

- **6.1. List of Applications**
- **6.2. Sudoku**

- Sudoku modelled as an Integer Problem
- Sudoku modelled as a zero-one (binary) Problem
- Results of the Douglas-Rachford algorithm on Sudoku puzzles
- **6.3. Magic Squares**
 - Modeling
 - Magic Squares Modelled as an Integer Problem
 - Magic Squares modelled as a zero-one (binary) problem

7. Implentation and Experimentation on Magic Squares

- Implementation of the Integer Case
- Algorithms used in an attempt to build Magic Squares
- Results

8. Conclusions

9. Biblography

Abstract

In this article, we'll discuss The Douglas Rachford Method, some of it's history, and it's applications. We'll be seeing different examples of how the Method works, and we'll also show some variations of this method that can be very useful, in different cases.

We'll also see some applications of this method, including finding Magic Squares, solving Sudoku riddles, solving Nonograms, and many more problems in math.

First we'll start with some definitions and theorems, that are the basis and the framework that we lay our algorithm on.

Then I will introduce the Douglas Rachford algorithm, giving a brief summary of what is the Douglas Rachford algorithm, it's history, the idea behind it, some of it's applications, and also investigate it's limitations.

1.Preliminaries

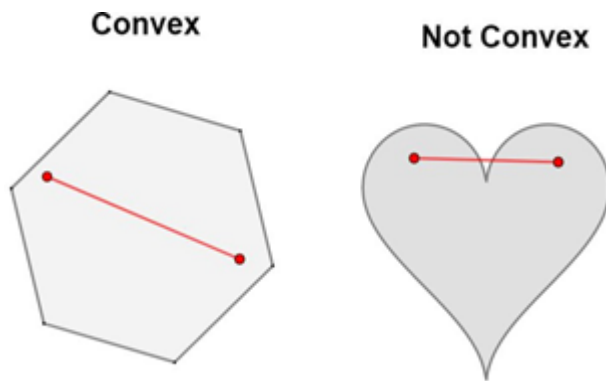
Definition 1 – Convex Set

Let H be a vector space and let $C \subseteq H$. We say that C is convex if :

$$\forall c \in C,$$

$$\lambda c + (1 - \lambda)c \in C \quad \forall \lambda \in [0,1].$$

In another words, one could say that a set is convex, if every set of points on the line connecting two points that are in the set is also in the set.



Definition 2 – Closest Point Projection

Let $C \subset \mathbb{R}^m$ be some closed set.

The closest point projection in C is a set valued mapping $P_C: \mathbb{R}^m \rightarrow C$, which assigns for any $x \in \mathbb{R}^m$

a point in C , denoted by $P_C(x)$, and is characterized by:

$$P_C(x) := \left\{ z \in C: \|x - z\| = \inf_{c \in C} \|x - c\| \right\}$$

Definition 3 – Reflection of a Point

The reflection of point x in the set C is also a set – valued mapping $R_C: \mathbb{R}^m \rightarrow C$ defined by:

$$R_C(x) = 2 * P_C(x) - x$$

2. Introduction – The Douglas Rachford Method

History

The Douglas – Rachford algorithm, which is also known as the Peaceman – Rachford splitting algorithm or the Forward – backward algorithm, is an iterative method for solving convex optimization problems.

The method was first proposed by R. Douglas and H. H. Rachford in 1956 to solve partial differential equations, and later it was rediscovered and popularized in the optimization community in the 1970s and 80s, and is still used to great success to this day.

Idea

The idea of the algorithm is to iteratively solve two simpler optimization subproblems, which are typically easier to solve than the original problem.

Applications

The Douglas – Rachford algorithm has a wide range of applications in many fields, including combinatorial problems such as Magic Squares, Sudoku, Nonograms, also Signal Processing, image analysis, machine learning, and many more.

It can be used to solve a variety of optimization problems, such as sparse signal recovery, compressed sensing, and others.

Advantages and Limitations

One of the advantage of the Douglas Rachford algorithm is that it can handle complex constraints and nonsmooth objective functions, which are common in many optimization problems.

Although it is very successful and useful in many fields of math, the Douglas Rachford does have its limitations, and in some cases, can converge very slowly, or may not converge at all for non convex problems.

Throughout time people came up with different variations and modifications of the algorithm, to address these limitations, including accelerated variants to have faster convergence, and some preconditioned versions, that work only when some preconditions are met.

3. Douglas Rachford Method

3.1. The classic Douglas – Rachford Method

The classic Douglas –

Rachford method was originally introduced in connection with PDEs

Arising in heat conduction [1]. Convergence of this method was later proven in [2], who also proposed this method to find zeros of the sum of two maximal monotone operators.

Given two subsets A, B of hilbert space \mathcal{H} , the method iterates by repeatedly applying the 2 – set Douglas – Rachford Operator:

$$T_{A,B} := \frac{I + R_B R_A}{2}$$

Where I denotes the identity mapping, $R_A(x)$ denotes the reflection of a point $x \in \mathcal{H}$ relative to the set A .

The reflection can be defined as:

$$R_A(x) := 2P_A(x) - x$$

Where $P_A(x)$ is the closest point projection of the point x onto the set A , that is:

$$P_A(x) := \left\{ z \in A : \|x - z\| = \inf_{a \in A} \|x - a\| \right\}$$

*In general, the projection P_A is a set valued map. If A is closed and convex, the projection is uniquely defined for every point in \mathcal{H} , thus yielding a single valued mapping. (See [3, **Theorem 4.5.1**])*

Applied to closed and convex sets, convergence is well understood and can be explained by using the theory of (firmly) nonexpansive mappings.

Theorem 1

Let $A, B \subseteq \mathcal{H}$ be closed and convex sets with nonempty intersection. $A \cap B \neq \emptyset$.

For any $x_0 \in \mathcal{H}$, set $x_{n+1} = T_{A,B}x_n$.

Then (x_n) converges weakly to a point x such that $P_Ax \in A \cap B$

Note:

(x_n) converges to a point x , but x is not necessarily belonging to $A \cap B$.

It is what's called the shadow sequence (P_Ax_n) that converges to a point P_Ax that does belong to $A \cap B$.

3.2. The Cyclic Douglas – Rachford Method

There are many possible generalizations of the classic Douglas – Rachford iteration.

Given three sets A, B, C and $x_0 \in \mathcal{H}$,

an obvious candidate is the iteration defined repeatedly

by setting $x_{n+1} := T_{A,B,C}x_n$ where

$$T_{A,B,C} := \frac{I + R_C R_B R_A}{2}$$

For closed and convex sets, like $T_{A,B}$, the mapping $T_{A,B,C}$ is firmly nonexpansive, and has at least one fixed point provided $A \cap B \cap C \neq \emptyset$.

Using a well known theorem of Opial (See [4], **Theorem 1**), (x_n) can be shown to converge weakly to a fixed point,

However, attempting to obtain a point in the intersection of the sets using said fixed point has been, so far, unsuccessful. We will illustrate this failure in the next Example:

Instead, Borwein and Tam (See [5]) considered a cyclic application of 2 – set Douglas Rachford

operators. given N sets C_1, C_2, \dots, C_N , and $x_0 \in \mathcal{H}$, their cyclic Douglas rachford scheme iterates by repeatedly setting $x_{n+1} := T_{[C_1, C_2, \dots, C_N]}x_n$

where

$T_{[C_1, C_2, \dots, C_N]}$ denotes the cyclic Douglas – Rachford operator defined by:

$$T_{[C_1, C_2, \dots, C_N]} = T_{C_N, C_1} T_{C_{N-1}, C_N} \dots T_{C_2, C_3} T_{C_1, C_2}.$$

In the consistent case, the iterations behave analogously to the classical Douglas – Rachford Scheme.

Theorem 2 (Cyclic Douglas – Rachford)

Let $C_1, C_2, \dots, C_n \subseteq \mathcal{H}$ be closed and convex sets with a nonempty intersection.

For any $x_0 \in \mathcal{H}$, set $x_{n+1} = T_{[C_1, C_2, \dots, C_N]}x_n$.

Then (x_n) converges weakly to a point x such that $P_{C_i}x = P_{C_j}x$ for all indices i, j .

Moreover, $P_{C_j}x \in \bigcap_{i=1}^N C_i$ for each index j .

3.3 Douglas Rachford Algorithm to find the Intersection of N balls in \mathbb{R}^m

Douglas Rachford can be used to find the intersection of N balls in \mathbb{R}^m .

To demonstrate this, we're going to be dealing with \mathbb{R}^2 , and give two examples.

One example is an example of 2 circles with exactly one intersecting point, and we show how taking an arbitrary point P , and running the DR iterations on this point P , slowly converges to the one intersection point.

We'll also demonstrate the same thing for 3 Circles in \mathbb{R}^2 , Using the cyclic Douglas Rachford method. But before doing so, we need a mathematical foundation of how to calculate a projection of an arbitrary point P , relative to a circle with center C and radius R .

Finding the projection of an arbitrary point P onto a circle with center C and radius R

We know from geometry, that a tangent to a circle is perpendicular to the radius at the tangency point. We can use this idea to find the projection of every point P , relative to the given circle.

We draw a straight line from C to P , and this straight line is perpendicular to the tangent line at the intersection point of the Line we made from C to P , and the circumference of the circle. Call this tangency point O .

Therefore, the projection of point P onto the circle, is exactly the point O .

We can use algebra to find explicitly the point O :

We know that a straight line is given by $y = mx + b$, and a circle with center (x_0, y_0) with radius r is given by:

$$(x - x_0)^2 + (y - y_0)^2 = r^2$$

We want to find where the straight line and the circle intersect, so we plug in $y = mx + b$ in the circle equation.

We'll get two results, which makes sense since a straight line that goes through the middle of the circle, will intersect the circle at two points. Afterwards, we will choose the closest point.

$$\begin{aligned}(x - x_0)^2 + (y - y_0)^2 &= r^2 \\ (x - x_0)^2 + (mx + b - y_0)^2 &= r^2\end{aligned}$$

$$\begin{aligned}
(mx + b - y_0)^2 &= (mx + b - y_0) * (mx + b - y_0) = m^2x^2 + mbx - my_0x + mbx + b^2 - by_0 - \\
&my_0x - y_0b + y_0^2 \\
&= m^2x^2 + b^2 + y_0^2 + 2mbx - 2my_0x - 2by_0
\end{aligned}$$

$$\begin{aligned}
(x - x_0)^2 + (mx + b - y_0)^2 &= \\
&= x^2 - 2x_0x + x_0^2 + m^2x^2 + b^2 + y_0^2 + 2mbx - 2my_0x - 2by_0 \\
&= x^2(m^2 + 1) + x(2mb - 2my_0 - 2x_0) + (x_0^2 + b^2 + y_0^2 - 2by_0) = r^2
\end{aligned}$$

$$\Rightarrow (m^2 + 1) * x^2 + (2mb - 2my_0 - 2x_0) * x + (x_0^2 + b^2 + y_0^2 - 2by_0 - r^2) = 0$$

So we have our quadratic formula $Ax^2 + Bx + C = 0$

$$\begin{aligned}
B^2 &= (2mb - 2my_0 - 2x_0)^2 = (2mb - 2my_0 - 2x_0) * (2mb - 2my_0 - 2x_0) = \\
&= 4m^2b^2 - 4m^2by_0 - 4mbx_0 - 4m^2by_0 + 4m^2y_0^2 + 4my_0x_0 - 4mbx_0 + 4my_0x_0 + 4x_0^2 \\
&= 4m^2b^2 + 4m^2y_0^2 + 4x_0^2 - 8m^2by_0 - 8mbx_0 + 8my_0x_0
\end{aligned}$$

$$x_{1,2} = \frac{\left((-2mb + 2my_0 + 2x_0) \pm \sqrt{4m^2b^2 + 4m^2y_0^2 + 4x_0^2 - 8m^2by_0 - 8mbx_0 + 8my_0x_0 - 4(m^2 + 1)(x_0^2 + b^2 + y_0^2 - 2by_0 - r^2)} \right)}{2(m^2 + 1)}$$

$$x_{1,2} = \frac{\left((-mb + my_0 + x_0) \pm \sqrt{m^2b^2 + m^2y_0^2 + x_0^2 - 2m^2by_0 - 2mbx_0 + 2my_0x_0 - m^2x_0^2 - m^2b^2 - m^2y_0^2 + 2m^2by_0 + m^2r^2 - x_0^2 - b^2 - y_0^2 + 2by_0 + r^2} \right)}{m^2 + 1}$$

$$x_{1,2} = \frac{\left((-mb + my_0 + x_0) \pm \sqrt{m^2r^2 + 2my_0x_0 + 2by_0 + r^2 - 2mbx_0 - m^2x_0^2 - b^2 - y_0^2} \right)}{m^2 + 1}$$

We had found a formula, that gives us two X coordinates, where the line $y=mx+b$ intersects the circle.

Then, we can just plug these x 's to one of the equations, preferably $y = mx +$

b because it's

easier to calculate, to find the corresponding y values.

so we get two points: $(x_1, y_1), (x_2, y_2)$.

Then to decide which is the desired point, we take the one that is closer to P . We did this by

calculating the distance from p to (x_1, y_1) calling it $dist1$, and from p to (x_2, y_2) calling it $dist2$,

and then deciding which is the right point, by taking the one that is responsible for the smaller distance out of $dist1$ and $dist2$. Basically we're taking $\min(dist1, dist2)$.

Note that the cyclic Douglas Rachford algorithm is used when we have atleast 3 sets, and the classic DR is used when we have two sets. Moreover, when dealing with the classic Douglas – Rachford method,

we have to keep in mind that the iterations do not give us a point belonging to the intersection of the sets, but rather they give us a point, whose projection on the first set A , is in the intersection.

3.4. Algorithms used in our Numerical Experiments:

I've written different algorithms for different problems. But the main idea of the algorithms, is to be able to calculate

the projection of a point, onto a set. The idea behind the algorithms is different for the case of having to project a point onto a line, or onto a circle.

The list of the algorithms presented is in the order of their usage in the Numerical Experiments Section of this article

Algorithm 1 – Used to plot the 3 Lines and x_0 in the 3 Line example

```
function [] = plotLinesPoint(p)

x=p(1);
y=p(2);
x1=linspace(-4,4);

y1=(1/sqrt(3))*x1;
y2=(-1/sqrt(3))*x1;

plot(x1, y1, 'r-', x1, y2, 'g--', x, y, 'o')
xlabel(0);

axis equal

end
```

Algorithm 2 – Used to calculate the projection of a point, onto a line.

Used in the 3 Line Example.

```

function [proj] = lineProjection(m,b,p)

x=p(1);
y=p(2);
values=linspace(-2,2);
f=m*values+b;

perpSlope=-1/m;

yInt=-perpSlope*x+y;

xIntersection=(yInt-b)/(m-perpSlope);
yIntersection=perpSlope*xIntersection+yInt;

proj(1)=xIntersection;
proj(2)=yIntersection;

end

```

*Algorithm 3 – Used to calculate the reflection of a point onto a line.
used in the 3 Line Example.*

```

function [ref] = lineReflection(m,b,p)

proj=lineProjection(m,b,p);
ref=2*proj-p;

end

```


Algorithm 4 – Used to calculate one full cyclic Iteration of the Cyclic Douglas Rachford variation. Used in the 3 Line Example.

```
function [point] = circularIteration(x0)

    %%% We know that first we have T_AB
    %Then T_BC
    %Then T_CA

    hold on % Incase we iterate a few times, so we keep the graph
    %Calculating t_AB
    x=x0(1); y=x0(2);
    refA=[-x, y];
    refB=lineReflection(1/sqrt(3),0,refA);

    tAB=(x0+refB)/2;

    hold on
    plot([x0(1) tAB(1)], [x0(2) tAB(2)], 'black');

    %Calculating t_BC
    x0=tAB;
    refB=lineReflection(1/sqrt(3),0,x0);
    refC=lineReflection(-1/sqrt(3),0,refB);

    tBC=(x0+refC)/2;

    hold on
    plot([x0(1) tBC(1)], [x0(2) tBC(2)], 'black');

    %Calculating t_CA
    x0=tBC;
    refC=lineReflection(-1/sqrt(3),0, x0);
    x=refC(1); y=refC(2);
    refA=[-x y];
    tCA=(x0+refA)/2;

    hold on
    plot([x0(1) tCA(1)], [x0(2) tCA(2)], 'black');

    point=tCA;

end
```

Algorithm 5 – Used to find the projection of a point, onto a 2D Ball.

Used in every example involving balls.

```
function [Intersection] = twoDcircleProjection(r, center, p)

% This function calculates the projection of an arbitrary point p = [x y]
% on the circle with the radius r and some center [x0 y0]

x0=center(1);
y0=center(2);

x=p(1);
y=p(2);

%check if the Point is inside or outside the circle
dist=sqrt((x-x0)^2+(y-y0)^2);
if dist>r

if x0==x %If the point is directly above the circle
    Intersection=[x0 (y0+r)];

    else
    %Slope
    m=(y-y0)/(x-x0);
    b=y-m*x;

    %Math explained in article

    xInt1=(-m*b+m*y0+x0+sqrt(m^2*r^2+2*m*y0*x0+2*b*y0+r^2-2*m*b*x0-m^2*x0^2-b^2-y0^2))/(m^2+1);
    xInt2=(-m*b+m*y0+x0-sqrt(m^2*r^2+2*m*y0*x0+2*b*y0+r^2-2*m*b*x0-m^2*x0^2-b^2-y0^2))/(m^2+1);

    yInt1=m*xInt1+b;
    yInt2=m*xInt2+b;

    %Now check which point is closer;

    dist1=sqrt((x-xInt1)^2+(y-yInt1)^2);
    dist2=sqrt((x-xInt2)^2+(y-yInt2)^2);

    %Pick the closer distance.
    Intersection=[xInt1 yInt1];
    if dist1>dist2
        Intersection=[xInt2 yInt2];
    end

end

else
    Intersection=p;

end

end
```

Algorithm 6 –

Used to find the reflection of a point, relative to a 2D Ball. Used in every example involving balls.

```
function [ref] = twoDcircleReflection(r,center,p)

%Gives a reflection of an arbitrary point p relative to a circle with
%radius R and some center.

proj=twoDcircleProjection(r,center,p);
ref=2*proj-p;

|
end
```

Algorithm 7:

Used to showcase the Classic Douglas Rachford iterations, having two 2D balls.

```
function [point] = CircleCircularDR(r1, r2, center1, center2, p)
%Denote A the circle made by r1 and center1
%Denote B the circle made by r2 and center2

%The DR Theorem states that
%Let A,B be two subsets of H, A,B closed and convex.
%Let  $x_{(n+1)}=tAB(x_n)$ , then  $x_n$  converges to a point  $x$  such that  $p_A(x)$ 
%belongs to the intersection
x=p(1);
y=p(2);

proj1=twoDcircleProjection(r1,center1,p);
hold on
plot(proj1(1),proj1(2),'o');
cx1=center1(1);
cy1=center1(2);

cx2=center2(1);
cy2=center2(2);

theta=linspace(0,2*pi);
x1=cx1+r1*cos(theta);
y1=cy1+r1*sin(theta);

x2=cx2+r2*cos(theta);
y2=cy2+r2*sin(theta);

hold on
plot(x1, y1);
hold on
plot(x2, y2);
hold on
plot(x,y,'o');

refA=twoDcircleReflection(r1,center1,p);
refB=twoDcircleReflection(r2,center2,refA);

point=(p+refB)/2;

%so point is  $x_{(n+1)}$ , we need now to project this point on A
proj=twoDcircleProjection(r1, center1, point);

hold on
plot(point(1), point(2), 'o');
hold on
plot([x point(1)], [y point(2)], 'red'); % Plot the progression of  $x_n$  to  $x_{(n+1)}$ 
hold on
plot([proj1(1) proj(1)], [proj1(2) proj(2)], 'blue'); % Plot the progression of the shadow sequence
hold on
plot(proj(1), proj(2), 'o');

end
```

Algorithm 8:

Used to showcase the Cyclic Douglas Rachford iterations, involving three 2D Balls. This algorithm *can be generalized for N Balls, $N \geq 3$. But I didn't do this. The idea is the same for $N > 3$, we'll just have to add more sets and run more calculations of the cycle.*

```
function [point] = threeCircleCircularDR(r1, r2, r3, center1,center2,center3, p)

%% We know that first we have T_AB
%Then T_BC
%Then T_CA

%we'll call the
%circle made by r1 and center1 =: A
%circle made by r2 and center2 =: B
%circle made by r3 and center3 =: C
x0=p;

cx1=center1(1);
cy1=center1(2);

cx2=center2(1);
cy2=center2(2);

cx3=center3(1);
cy3=center3(2);

theta=linspace(0,2*pi);
x1=cx1+r1*cos(theta);
y1=cy1+r1*sin(theta);

x2=cx2+r2*cos(theta);
y2=cy2+r2*sin(theta);

x3=cx3+r3*cos(theta);
y3=cy3+r3*sin(theta);

hold on
plot(x1, y1);
hold on
plot(x2, y2);
hold on
plot(x3,y3);
hold on
plot(x0(1),x0(2),'o');

%calculating tAB
refA=twoCircleReflection(r1, center1, p);
refB=twoCircleReflection(r2,center2, refA);

tAB=(p+refB)/2;

%hold on
%plot([x0(1) tAB(1)], [x0(2) tAB(2)], 'red');

%calculating tBC
x0=tAB;
refB=twoCircleReflection(r2,center2,x0);
refC=twoCircleReflection(r3,center3,refB);

tBC=(x0+refC)/2;

%hold on
%plot([x0(1) tBC(1)], [x0(2) tBC(2)], 'blue');

%calculating tCA
x0=tBC;
refC=twoCircleReflection(r3, center3, x0);
```

```

x0=tBC;
refC=twoDcircleReflection(r3,center3,x0);
refA=twoDcircleReflection(r1,center1,refC);

tCA=(x0+refA)/2;

hold on
plot([p(1) tCA(1)], [p(2) tCA(2)], 'black');

point = tCA;

end

```

Algorithm 9:

Used in the nonconvex example of a circle and a line in two dimensions. This is the DR algorithm.

```

function [point] = lineCircleDR(m,b,r,center,p)

x1=linspace(-5,5);
y1=m*x1+b;
theta=linspace(0,2*pi);

x2=center(1)+r*cos(theta);
y2=center(2)+r*sin(theta);

hold on
plot(x1,y1,'r-');
hold on
plot(x2,y2);
hold on
plot(p(1),p(2),'o');

projA1=twoDcircleProjection(r,center,p);
hold on
plot(projA1(1),projA1(2),'o');
%Denote the circle by A
%Denote the line by B
refA=twoDcircleReflection(r, center,p);
refB=lineReflection(m,b,refA);

tAB=(p+refB)/2;
hold on
plot(tAB(1),tAB(2),'o');
point=tAB; %this is x_(n+1), now need to project it to A to get the shadow sequence
projA2=twoDcircleProjection(r,center,point);
hold on
plot(projA2(1),projA2(2),'o');
hold on
plot([p(1) point(1)], [p(2) point(2)], 'black');
hold on
plot([projA1(1) projA2(1)], [projA1(2) projA2(2)], 'red');

end

```

Algorithm 10 – Von Neumann Alternating Projection Method

Used as an example of another algorithm that converges to the intersection point of a circle and a straight line, using a different method, called the Von Neumann Alternating Projection method.

```
function [point] = vonNeumann(m,b,r,center,p)

x1=linspace(-5,5);
y1=m*x1+b;
theta=linspace(0,2*pi);

x2=center(1)+r*cos(theta);
y2=center(2)+r*sin(theta);

hold on
plot(x1,y1,'r-');
hold on
plot(x2,y2);
hold on
plot(p(1),p(2),'o');

projA=twoDCircleProjection(r,center,p);
hold on
plot([p(1) projA(1)], [p(2) projA(2)], 'red');
projB=lineProjection(m,b,projA);
point=projB;
hold on
plot([point(1) projA(1)], [point(2) projA(2)], 'blue');
hold on
plot(point(1),point(2),'o');

hold on
%plot([p(1) point(1)], [p(2) point(2)], 'black');

end
```

Using what we have until now, we can start with some Numerical Examples.

4. Numerical Experiments

Example 1 (Failure of the three set Douglas – Rachford iterations.)

We give an example showing the iteration described previously, can fail to find a feasible point.

Consider the one – dimensional subspaces $A, B, C \subset \mathbb{R}^2$ defined by:

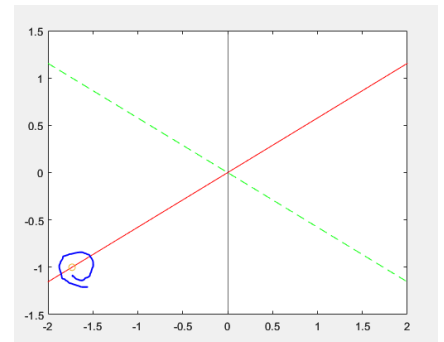
$$A := \{\lambda(0,1): \lambda \in \mathbb{R}\}$$

$$B := \{\lambda(\sqrt{3}, 1): \lambda \in \mathbb{R}\}$$

$$C := \{\lambda(-\sqrt{3}, 1): \lambda \in \mathbb{R}\}.$$

Then $A \cap B \cap C = \{(0,0)\}$.

Let $x_0 = (-\sqrt{3}, -1)$.



In the figure above we have the three subspaces, and the yellow circle, marked by a blue circle, is the point $x_0 = (-\sqrt{3}, -1)$

We know that in the three set douglas – rachford, the douglas rachford operator is defined as follows:

$$x_{n+1} = T_{A,B,C}x_n$$

where

$$T_{A,B,C} := \frac{I + R_C R_B R_A}{2}$$

Reflecting x_0 relative to A in matlab seems difficult, as it's difficult to define a function that is only vertical, since it has infinite slope, it cannot be defined as $y = mx + b$ like the other lines.

But since we know that A is a vertical line, reflecting relative to it is straightforward,

$R_A(x_0) = (\sqrt{3}, -1)$, That is, just changing the sign of the x value.

Now we'll show that

$x_0 \in \text{Fix}(R_C R_B R_A)$, which will yield to the fact that $x_0 \in \text{Fix}\left(\frac{I + R_C R_B R_A}{2}\right)$,

meaning that x_0 is a fixed point of the three set douglas rachford operator in this scenario.

To show that, we'll calculate R_B and R_C , and to calculate R_B and R_C , we will use the relationship between the reflection of a point and it's projection, and we'll also use a function that I built in matlab that gets as an input, the slope m and intersection b of a line, and a point p_1 , and returns another point p_2 , which is the projection of p_1 on the line represented by m and b .

We know that if we have a vector v and a line l ,
then

$\text{Ref}_l(v) = 2\text{Proj}_l(v) - v$, Meaning that the reflection of v on l
is 2 times the projection of v on l , minus the vector v .

We can use our function and this relationship to find R_B and R_A .

$$R_B(x_0) = 2P_B(x_0) - x_0$$

$$R_A(x_0) = 2P_A(x_0) - x_0$$

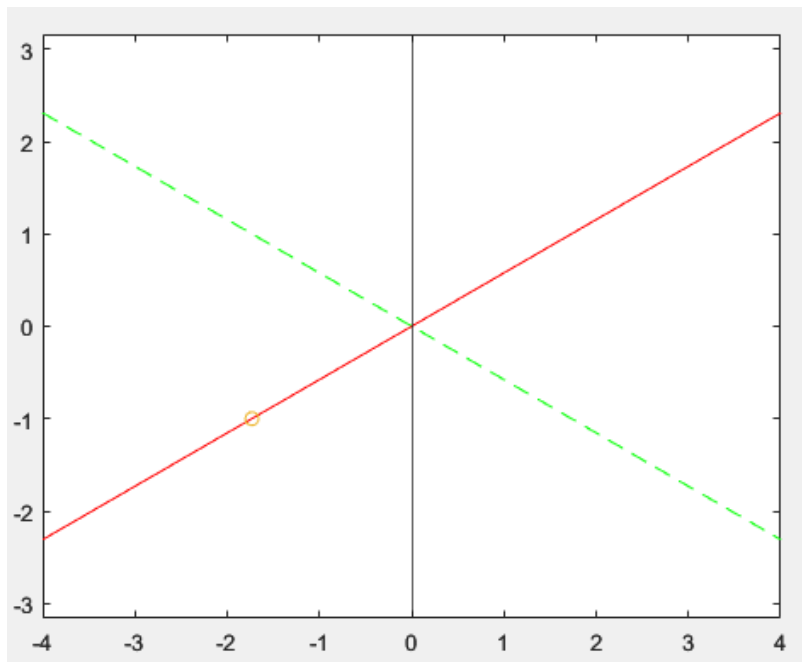
In our case,

we could write B and C as:

$$B: y_1 = \frac{1}{\sqrt{3}}x$$

$$C: y_2 = -\frac{1}{\sqrt{3}}x$$

Let's do this:

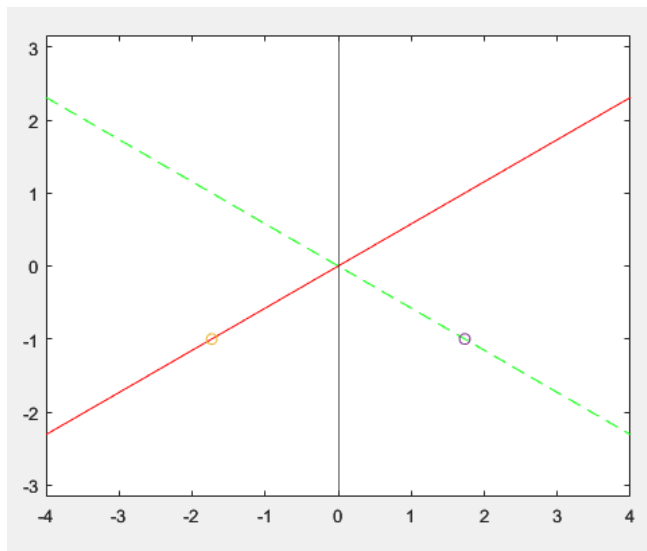


That's our initial figure, with all the sets, and x_0 marked as a small yellow circle on the red line.

Now we want $R_A x_0$, since A is the vertical line, we'll just calculate it's reflection straightforward.

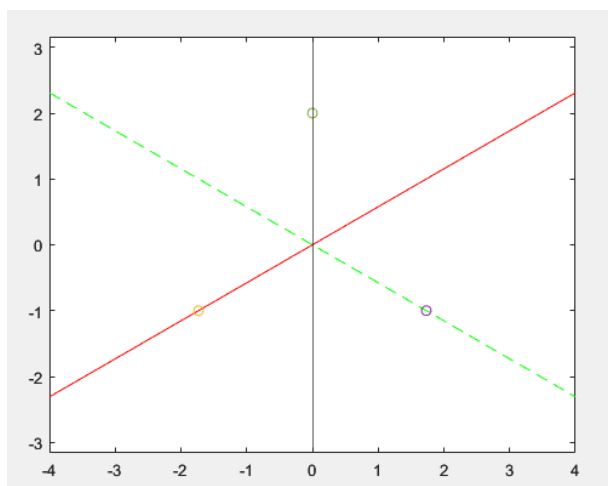
since x_0 is $(-\sqrt{3}, -1)$, then $R_A(x_0) = (\sqrt{3}, -1)$.

Let's add this to the plot:



Now we want $R_B R_A x_0$, which just means,
to reflect the last point we got which is $(\sqrt{3}, -1)$, relative to B .

And to do that, I've made a function that returns me the point which is the reflection of a point relative to a set.



And finally, we want $R_C R_B R_A x_0$, which just means to reflect our last point which is $(0,2)$, relative to C .

And once again I'll do this using the function i've written to find a reflection.

And we see using the function I've made, that the output (which is the reflected point), is $(-\sqrt{3}, -1)$, which is just x_0 .

$R_C R_B R_A x_0 = x_0$, meaning that $x_0 \in \text{Fix } R_C R_B R_A x_0$. Therefore $x_0 \in \text{Fix } \frac{I + R_C R_B R_A}{2}$

The code I've used

```

>> plotLinesPoint([-sqrt(3) -1])
>> hold on
>> plot(sqrt(3),-1,'o')
>> refB=lineReflection(1/sqrt(3),0, [sqrt(3) -1]);
>> x1=refB(1), y1=refB(2);

x1 =

    -6.6613e-16

>> hold on
>> plot(x1,y1, 'o')
>> refC=lineReflection(-1/sqrt(3),0, [0 2]);
>> x2=refC(1); y2=refC(2);
>> x1, y2

x1 =

    -6.6613e-16

y2 =

    -1.0000

>> x2, y2

x2 =

    -1.7321

y2 =

    -1.0000

```

Meaning that x_0 is a fixed point of the operator.

Meaning that x_0 converges to x_0 , no matter how many time we use this operator.

By douglas rachford theorem,

x_0 converges slowly to a point x such that $P_A x \in A \cap B$.

In our case,

$A \cap B \cap C = \{(0,0)\}$, and x_0 does converge, to itself, since it's a fixed point.

But if we check

$P_A x_0 = \text{Projection of } x_0 \text{ relative to a vertical line} = (0, -1) \neq (0,0)$

$P_B x_0 = \text{Using our algorithm of calculating a projection} = x_0 = (-\sqrt{3}, -1)$

$P_C x_0 = \text{Using our algorithm of calculating a projection} = \left(-\frac{\sqrt{3}}{2}, \frac{1}{2}\right)$

```
>> projB=lineProjection(1/sqrt(3),0, [-sqrt(3),-1]);
>> projC=lineProjection(-1/sqrt(3),0, [-sqrt(3), -1]);
>> projB

projB =

    -1.7321    -1.0000

>> projC

projC =

    -0.8660     0.5000

>> [
```

And we see that the Douglas – Rachford algorithm fails for the three set method.

Example 2 (Example 1 Revisited)

Now we revisit Example 1, but we're using a different approach.

Consider the cyclic Douglas –

Rachford scheme applied to the sets of the Previous example.

As before, let $x_0 = (-\sqrt{3}, -1)$.

By theorem 2, the sequence (x_n) converges to a point x such that

$$P_A x = P_B x = P_C x = (0,0)$$

We know that for two sets, A and B ,

$$T_{A,B} = \frac{I + R_B R_A}{2} \Rightarrow$$

$$T_{A,B} x = \frac{x}{2} + \frac{R_B R_A x}{2}$$

In our example, we have only 3 sets, A, B and C .

So starting with x_0 ,

Our circular iterations are:

$$T_{A,B} x_0$$

$$T_{B,C} T_{A,B} x_0$$

$$T_{C,A} T_{B,C} T_{A,B} x_0$$

and so on ...

In order to do this simulation effectively, I've written a code that simulates one full cyclic

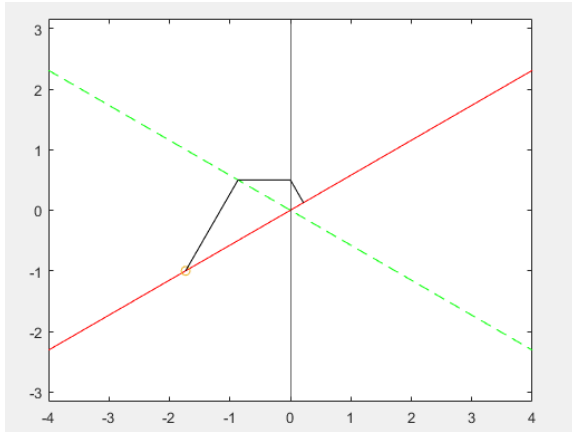
Iteration at a time. The code gets as an input a point x_0 , and calculates for us

$T_{C,A} T_{B,C} T_{A,B} x_0$, as well as drawing the lines from x_0 to $T_{A,B} x_0$, from

$T_{A,B} x_0$ to $T_{B,C} T_{A,B} x_0$ and so on.

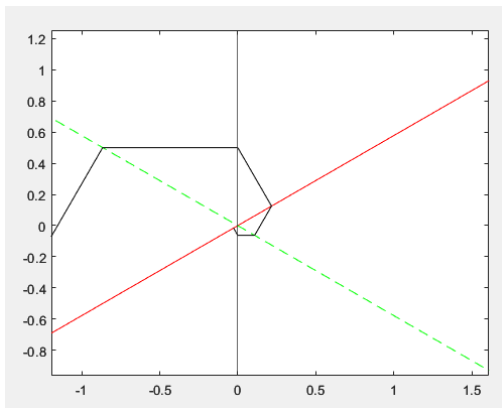
Then we could let $x_1 = T_{C,A} T_{B,C} T_{A,B} x_0$ and input x_1 for the next cyclic Iteration.

After one run of the circular Iteration (Input is $x_0 = (-\sqrt{3}, -1)$)

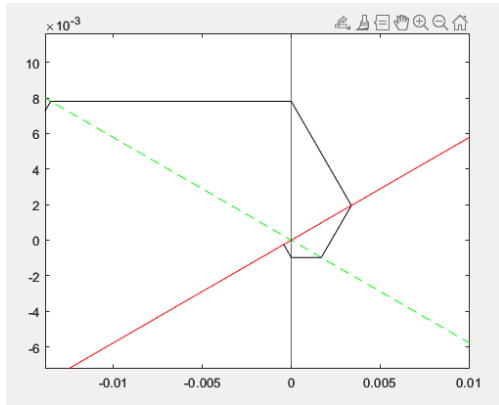


After two runs of the circularIteration (Input is $x_1 = T_{C,A}T_{B,C}T_{A,B}x_0$)

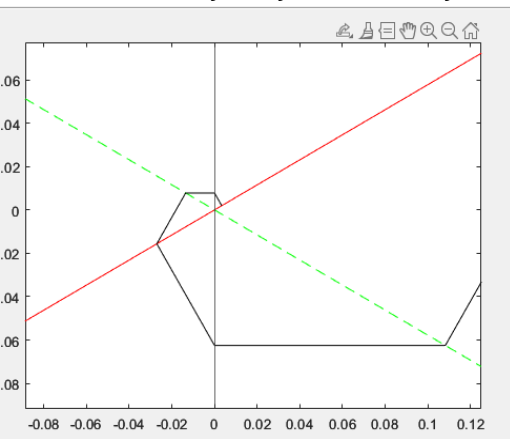
We can see that each run of adds 3 lines, that is because we have 3 Douglas Rachford operators to calculate



After three runs of the circularIteration (Input is $x_2 = T_{C,A}T_{B,C}T_{A,B}x_1$)



After four runs of the circularIteration (Input is $x_3 = T_{C,A}T_{B,C}T_{A,B}x_2$)



We can see that the algorithm converges in a spirally way to the intersection point.

We could run more iterations, and by the theorem, eventually we'll get to a point x such that $P_Ax = P_Bx = P_Cx = (0,0)$.

We can also see in the figure, that the spiral converges to the intersection point.

Example 3 – 2 Balls Scenario (using classic Douglas Rachford)

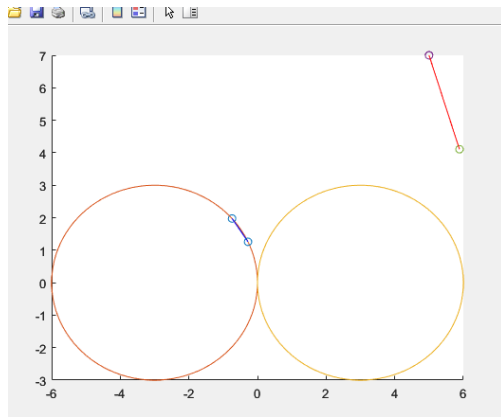
In red lines we see the progression of the sequence (x_n) , where $x_{n+1} = T_{A,B}x_n$

In blue lines we see the progression of the shadow sequence $(P_A x_n)$, where we first need to calculate x_{n+1} and then we project it to A to find the next element in the shadow sequence.

First run of the algorithm:

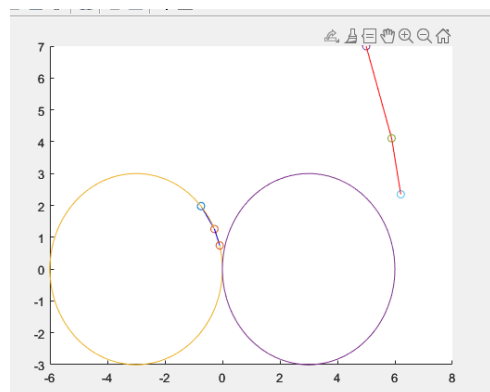
Inputs:

$(r_1 = 3, r_2 = 3, \text{center1} = (-3,0), \text{center2} = (3,0), \text{and } p_0 = (5,7))$

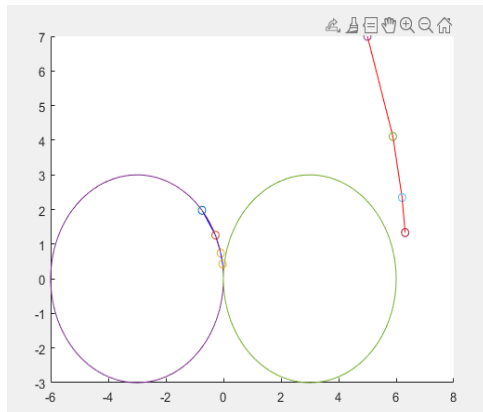


Second run of the algorithm:

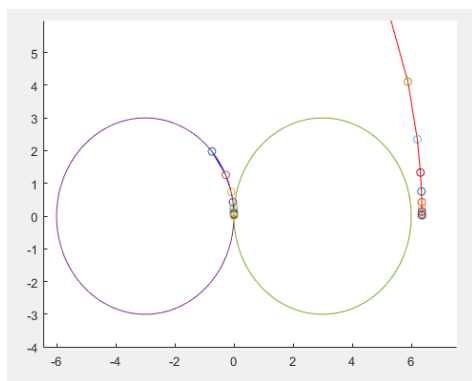
The inputs are the same for every iteration, except for the point p_0 which changes every iteration, and in each new iteration, we put the new p_0 as an input.



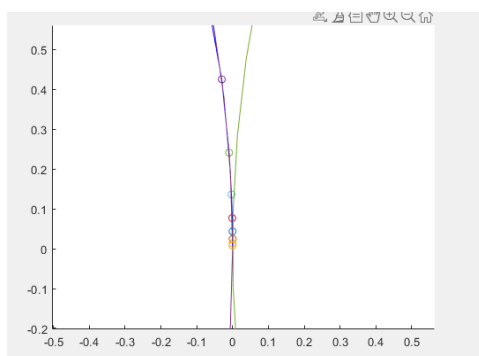
Third run of the algorithm:



10th Run of the algorithm:



*Zooming in to see the progression of the Shadow Sequence, we can see
That the shadow sequence does slowly converge to a point that belongs to the intersection
of the two sets.*



Example 4 – 3 Balls Scenario (using cyclic Douglas Rachford)

By Theorem 2, for the cyclic Douglas Rachford,

The sequence (x_n) converges to a point x such that $P_A x = P_B x = P_C x$, and $P_j x \in A \cap B \cap C \forall j \in \{A, B, C\}$.

As inputs for the algorithm,

I gave 3 circles (3 sets):

$A :=$ Circle with Radius 3 and center $(-3,0)$

$B :=$ Circle with Radius 3 and center $(3,0)$

$C :=$ Circle with Radius 3 and center $(0,3)$

and an Initial point $P = (5,7)$

A very specific choice to have only one intersection point at $A \cap B \cap C = \{(0,0)\}$.

Since it's the cyclic DR method, each iteration requires 3 calculations: T_{AB} , T_{BC} , T_{CA} ,

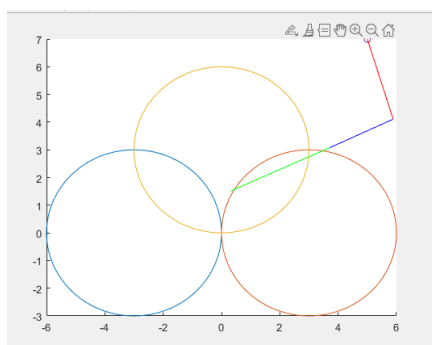
so in the plot there are also different colored lines:

Red for T_{AB} of the current iteration,

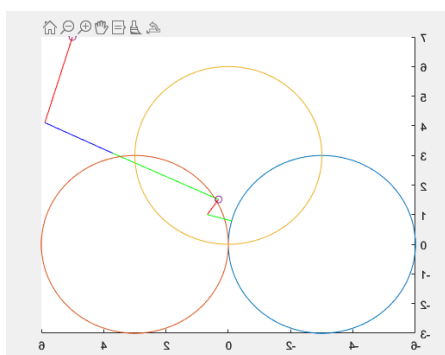
Blue for T_{BC} of the current iteration,

and Green for T_{CA} of the current iteration.

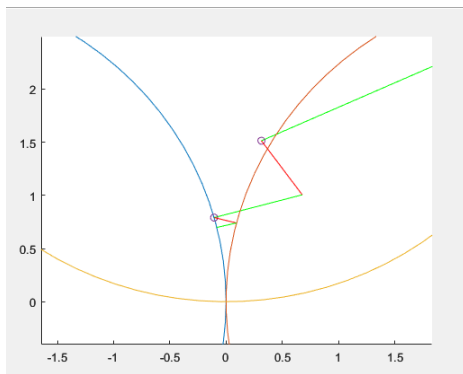
First iteration:



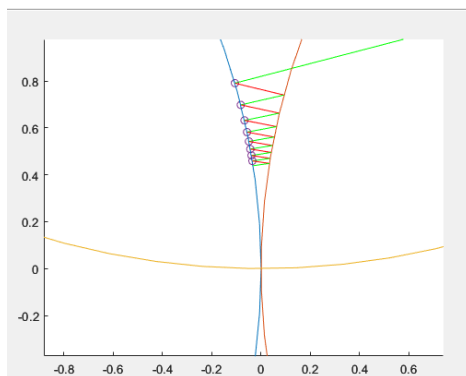
Second Iteration:



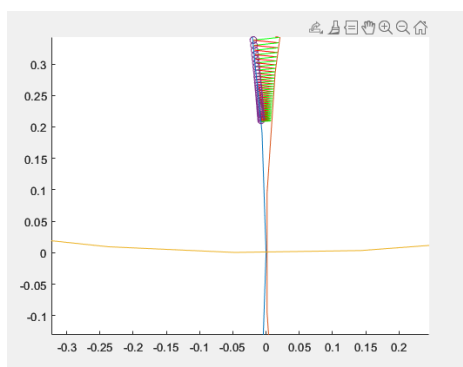
Third Iteration:



10th Iteration:



50th Iteration:



We can see that the sequence (x_n) does slowly converge to some value x , such that $P_A x = P_B x = P_C x = \text{Their intersection point} = (0,0)$.

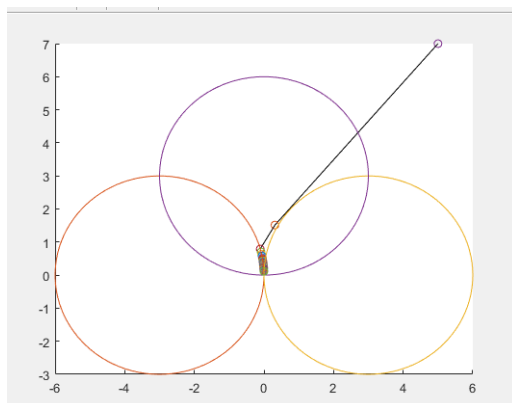
We can also see that the blue line has disappeared only after one iteration.

The reason for that is the fact that if an element is already in a set, then it's projection or reflection onto that set, is just the same point.

Same iterations just instead of connecting x_0 to $T_{A,B}x_0$ to $T_{B,C}T_{A,B}x_0$ and so on, we immediately connect x_0 to $T_{C,A}T_{B,C}T_{A,B}x_0$:



After 100 + Iterations:



Example 5 – Nonconvex example of a Circle and a Line

In Black lines – progression of (x_n)

In Blue Lines – progression of the shadow sequence $(P_A x_n)$,

We see that the sequence (x_n) does slowly converge to a point x , such that

$P_A x \in A \cap B$.

The initial inputs are:

$m = 0$ (line with slope 0)

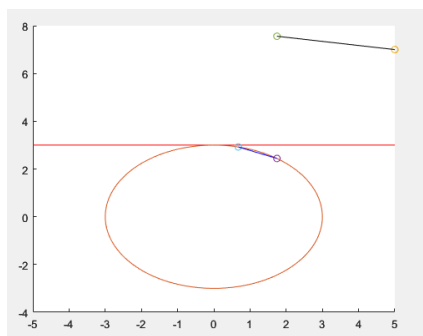
$b = 3$ (y intersect of the line)

$r = 3$ (radius)

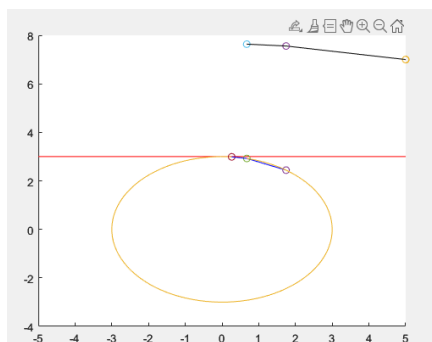
center = (0,0), center of the circle

$p = (5,7)$, $p = x_0$

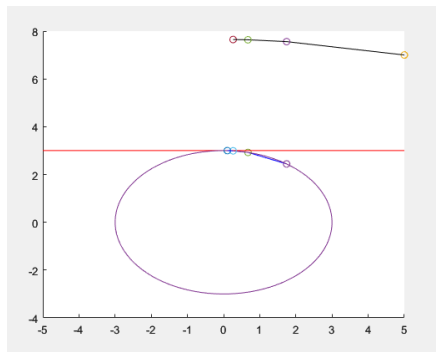
First Iteration:



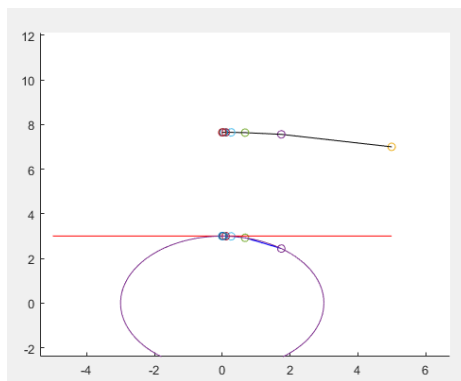
Second Iteration:



Third Iteration:

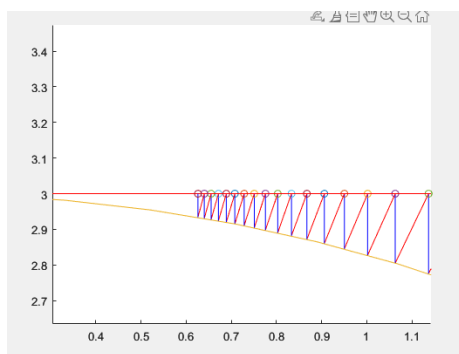
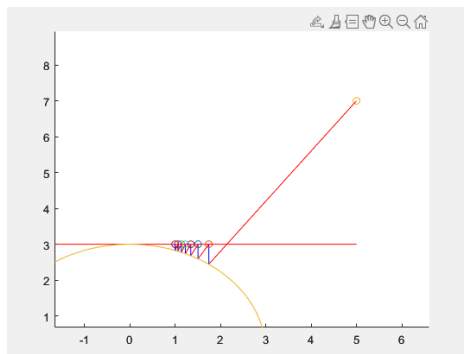


10th Iteration:



We see that the shadow sequence does converge to the point of intersection of the two sets

Example 6 – Von Neumann Alternating Projection Method



We can see that this algorithm also converges to the intersection point of the circle and the straight line.

5. Feasibility Problems in the Product Space

Given $C_1, C_2, \dots, C_n \subset \mathbb{R}^m$, the feasibility problem asks:

$$\text{Find } x \in \bigcap_{i=1}^N C_i \subset (\mathbb{R}^m)^N$$

A great many optimization and reconstruction problems, both continuous and combinatorial can be cast within this framework.

What's good about this formulation, is not just that the original feasibility problem, can be translated into a feasibility problem containing two sets, as we've seen above, that in that case the classic douglas rachford algorithm does converge.

but also that the projections onto the sets which we'll soon see, C_1, \dots, C_N , have a closed form of $P_{C_1}, P_{C_2}, \dots, P_{C_N}$, as yet again, soon will be shown.

Define two sets $C, D \subset (\mathbb{R}^m)^N$ by

$C := \prod_{i=1}^N C_i$, the cartesian product of the sets C_i ,

Such that

$C := \{(c_1, c_2, \dots, c_N) \in (\mathbb{R}^m)^N \mid c_i \in C_i \subset \mathbb{R}^m \forall i \in \{1, \dots, N\}\},$

$D := \{(x, x, \dots, x) \in (\mathbb{R}^m)^N : x \in \mathbb{R}^m\}.$

While the set D , the diagonal, is always a closed subspace, the properties of C are largely inherited.

For instance, when C_1, C_2, \dots, C_n are closed and convex, so is C .

Consider, now, the equivalent feasibility problem:

(4) Find $x \in C \cap D \subset (\mathbb{R}^m)^N$

Equivalent in the sense that

$$x \in \bigcap_{i=1}^N C_i \Leftrightarrow (x, x, \dots, x) \in C \cap D.$$

Moreover, knowing the projections onto C_1, C_2, \dots, C_n , the projections onto C and D can be easily computed. The proof has recourse to the standard characterization of orthogonal projection,

$p = P_D x \Leftrightarrow \langle x - p, d \rangle = 0$ for all $d \in D$.

Now we'll shown an example, why this formulation is very useful to find the solution for the feasibility problem.

We'll show how it works in two dimensions, but the idea could be generalized and extended for every whole dimension.

Let's start with two sets:

$$C_1 = [2,8] \subset \mathbb{R}^1$$

$$C_2 = [3,6] \subset \mathbb{R}^1$$

we want to find x such that $x \in C_1 \cap C_2$,

$$C_1 \cap C_2 = [3,6] \subset \mathbb{R}^1$$

so every x in $[3,6]$ will belong to that intersection.

When we define $C := C_1 \times C_2$, we get that $C \subset$

\mathbb{R}^2 , and in fact, we get a rectangular shape in \mathbb{R}^2 , which is the set $C_1 \times C_2$.

and $D := \{(x, x, \dots, x) \in (\mathbb{R}^m)^N : x \in \mathbb{R}^m\}$ in the two dimensional ,

we'll get a function $x = y$, which is a vector of type the form (x, x) , where $x \in \mathbb{R}$.

Now, notice that the intersection of the sets is one dimensional, and that their product is two dimensional, and so is $y = x$, or D .

but finding a vector $(x, x, \dots, x) \in C \cap D$, or in our case,

$(x, x) \in (C_1 \times C_2) \cap D$, we get precisely all the vectors of the form (c_1, c_1) , where $c_1 \in [3,6]$,

So we can translate back from \mathbb{R}^2 , to find our x that is in \mathbb{R} that belongs to the intersection of $C_1 \cap C_2$.

The space product formulation allows us to find the intersection of sets,

by going to a higher dimension, and using the diagonal set to get a vector in that higher dimension, whose all entries are the same,

and we can take the entries back to the lower dimension, to find the intersecting point, in the lower dimension.

Theorem 3 (Product Projections)

Let $x = (x_1, \dots, x_N) \in (\mathbb{R}^m)^N$.

Then

$$P_D x = \left(\frac{1}{N} \sum_{i=1}^N x_i, \dots, \frac{1}{N} \sum_{i=1}^N x_i \right),$$

and if $P_{C_1}(x_1), \dots, P_{C_N}(x_N)$ are nonempty then

$$P_C(x) = \prod_{i=1}^N P_{C_i}(x_i).$$

For clarification:

$$x = (x_1, \dots, x_N) \in (\mathbb{R}^m)^N \text{ means that } x_i \in \mathbb{R}^m \forall i, \text{ which means that } x_i = \begin{pmatrix} c_1 \\ c_2 \\ \vdots \\ c_m \end{pmatrix} \in \mathbb{R}^m, c_j \in$$

$$\mathbb{R} \forall i, j.$$

Proof

For the set D:

Let $(p, \dots, p) \in D$ be the projection of x onto D . For any $d \in \mathbb{R}^m$, one has $(d, \dots, d) \in D$.

Now

$$\begin{aligned} 0 &= \langle x - (p, \dots, p), (d, \dots, d) \rangle = \langle (x_1, x_2, \dots, x_N) - (p, \dots, p), (d, \dots, d) \rangle = \\ &= \langle x_1 - p, d \rangle + \langle x_2 - p, d \rangle + \dots + \langle x_N - p, d \rangle = \\ &= \sum_{i=1}^N \langle x_i - p, d \rangle = \langle \sum_{i=1}^N (x_i - p), d \rangle = \langle \sum_{i=1}^N x_i - Np, d \rangle = 0 \end{aligned}$$

$$\text{Therefore } \sum_{i=1}^N x_i - Np = 0 \Leftrightarrow p = \frac{1}{N} * \sum_{i=1}^N x_i.$$

This proves the projection onto D .

for the set C:

We now prove the projection formula for C .

We'll do this by first, prove that every $x = (x_1, x_2, \dots, x_N) \in (\mathbb{R}^m)^N$ such that $x \in \prod_{i=1}^N P_{C_i}(x_i) \Rightarrow x \in P_C(x)$ meaning that $\prod_{i=1}^N P_{C_i}(x_i) \subseteq P_C x$ and then we'll show the opposite.

For any $c = (c_1, \dots, c_N) \in C$ and $p = (p_1, \dots, p_N) \in \prod_{i=1}^N P_{C_i}(x_i) \subseteq C$,

$$\|x - c\|^2 = \sum_{i=1}^N \|x_i - c_i\|^2 \geq \sum_{i=1}^N \|x_i - p_i\|^2 = \|x - p\|^2.$$

Since $P_C(x) \subseteq C$, this shows $\prod_{i=1}^N P_{C_i}(x_i) \subseteq P_C x$

to clarify,

$$p = (p_1, \dots, p_N) \in \prod_{i=1}^N P_{C_i}(x_i) \subseteq C$$

means that $p_i \in P_{C_i}(x_i) \in \mathbb{R}^m \ \forall i$.

And

$\prod_{i=1}^N P_{C_i}(x_i)$ is indeed a subset set or equal to C ,

because $C := \prod_{i=1}^N C_i$

and each $P_{C_i}(x_i) \subseteq C_i$

Conversly,

let $p = (p_1, \dots, p_N) \in P_C(x)$ and suppose by contradiction that $p_j \notin P_{C_j}(x_j)$ for some j .

Define $q := (q_1, \dots, q_N) \in (\mathbb{R}^m)^N$ where $q_j \in P_{C_j}(x_j)$ and $q_i = p_i$ if $i \neq j$.

Then

$$\|x - p\|^2 = \sum_{i=1}^N \|x_i - p_i\|^2 > \sum_{i=1}^N \|x_i - q_i\|^2 = \|x - q\|^2$$

since $q \in C$, we conclude that $p \notin P_C x$. Whis is a contradiction, therefore $p_j \in P_{C_j}(x_j)$.

This completes the proof.

6. Applications of the Douglas Rachford Algorithm

6.1. List of Applications

- 1. Protein folding and graph coloring problems were first studied via Douglas – Rachford methods in [6] and [7], respectively.*
- 2. Image retrieval and phase reconstruction problems are analyzed in some detail on [8, 9]. The bit retrieval problem is considered in [7].*
- 3. Mastrix completion problems were studied using Douglas – Rachford methods in [10]. This includes finding various types of hadamard matrices, and construction of low – rank distance matrices. For a survey of matrix completion problem, see [11].*
- 4. The N queens problem, which requests the placement of N queens on a $N \times N$ chessboard, is studied and solved in [12].*
- 5. Boolean satisfiability is treated in in [7, 13]. Note that the three variable case, 3 – SAT, was the first problem to be shown NP – complete [14].*
- 6. TextaVex is an edge – matching puzzle, whose NP – completeness is discusses in [15], was studied in [16]. Problems up to size 4×4 could be solved in an average of 200 iterations. There are $10^{2n(n+1)}$ base – 10 $n \times n$ boards, with $n = 3$ being the most popular.*
- 7. Solutions of (very large) Sudoku puzzles have been studied in [12, 7]. For a discussion of NP – completeness of determining solvability of Sudoku see [17]. The effective solution of Sudoku pzuzles forms the basis of the next section, which is our main section.*
- 8. Nonograms [18, 19] are a more recent NP – complete Japanese puzzle which were shown to be solved with Douglas – Rachford methods.*

6.2 Douglas Rachford Algorithm – Sudoku

Theorem 1 only guarantees the global convergence of Douglas Rachford algorithms for convex sets. In spite of this, the algorithm has been successfully applied as a heuristic for solving many nonconvex problems, especially those of combinatorial type [20, 21].

In most applications in the nonconvex setting, the constraint sets satisfy some type of regularity property and local convergence can be proved [22, 7, 23].

Either way, the results on global behavior are yet limited to very specific sets [24], and we cannot conclude the good performance of the algorithm in the nonconvex setting.

As mentioned before, the efficiency and performance of an algorithm, depends on the way it is formulated.

It is already known that two algorithms might do the same thing exactly, but one will take much more time and space.

So when we use the Douglas Rachford algorithm to a nonconvex setting, finding a suitable formulation can be crucial

for it's success as a heuristic. Sometimes the formulation of a problem can successfully solve the problem always, sometimes, and even never, two different formulations of essentially the same problem, can yield very different results.

We'll briefly discuss an interesting example regarding those results.

Sudoku, which will also lead us to the topic of Magic Squares, which are very similar in nature.

Solving Sudoku puzzles with Douglas Rachford was proposed in [7], and also later analyzed in [12] and [20].

A sudoku is a 9 by 9 grid, divided into 3 by 3 subgrids, with some entries already prefilled.

The goal is to fill all the remaining entries in such a way that:

every row, column, and subgrids, contain every digit from 1 to 9 exactly once.

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

(a) Unsolved Sudoku

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

(b) Solved Sudoku

Figure 6: Example of Sudoku

Sudokus, are actually matrix completion problems, which we can model as a feasibility problem.

There are different ways to choose the constraint sets C_1, \dots, C_N , in such a way that $\bigcap_{i=1}^N C_i$ coincides with the unique solution of the Sudoku.

Eithe way, as we've already seen before, in order to apply the Douglas – Rachford method,

the sets must be chosen in such a way that the projections upon them can be easily computed, ideally having a closed form.

We'll show two ways of Modelling Sudoku, one as an integer feasibility problem, and the other as a binary feasibility problem.

6.2.1 Sudoku Modeled as an Integer Problem.

It is fairly simple to model sudoku as an Integer Problem.

Denote S by the partially filled 9 by 9 matrix representing the unsolved Sudoku.

Let $J \subset \{1,2, \dots, 9\}^2$ be the set of indicies for which S is already filled, and let $A_{i,j}$ denote the $(i,j)_{\text{th}}$ entry of the matrix A .

If we denote by C the set of vectors which are permutations of $1,2, \dots, 9$, then a matrix $A \in \mathbb{R}^{9 \times 9}$ is a solution to the

Sudoku only and only if $A \in C_1 \cap C_2 \cap C_3 \cap C_4$ where

$$C_1 = \{A \in \mathbb{R}^{9 \times 9} | \text{Each row of } A \in C\}$$

$$C_2 = \{A \in \mathbb{R}^{9 \times 9} | \text{Each column of } A \in C\}$$

$$C_3 = \{A \in \mathbb{R}^{9 \times 9} | \text{Each subgrid of } A \in C\}$$

$$C_4 = \{A \in \mathbb{R}^{9 \times 9} | A_{i,j} = S_{(i,j)} \forall (i,j) \in J\}.$$

The projection onto C_4 can be calculated component wise, Where the projections of C_1, C_2 and C_3 , are determined by the next result:

Proposition 2

Denote by $C \in \mathbb{R}^m$ the set of vectors whose entries are all permutations of $c_1, \dots, c_m \in \mathbb{R}$. Then for any $x \in \mathbb{R}^m$ one has that:

$$P_C(x) = [C]_x,$$

Where $[C]_x$ denotes the set of vectors in C such that

$y \in [C]_x$ if the i_{th} largest entry of y , is indexed the same as the i_{th} largest entry of x .

Proof:

For any $c \in C$:

$$\begin{aligned} \|x - c\|^2 &= \|x\|^2 + \|c\|^2 - 2x^T c = \|[x]\|^2 + \|[c]\|^2 - 2x^T c \geq \|[x]\|^2 + \|[c]\|^2 - 2[x]^T [c] = \\ \|[x] - [c]\|^2 &= \|x - y\|^2 \text{ for } y \in [C]_x \end{aligned}$$

Remark 2.1:

Proposition 2 suggests the following algorithm in order to compute a projection of x onto C .

Since the projection is in general not unique,

we want to find the nearest point, p , in the set of projections or some other alternative.

For convenience, given a vector $y \in (\mathbb{R}^2)^n$, we denote the projections onto the first and second product coordinates by Q and S , respectively. That is, if $y = ((x_1, c_1), (x_2, c_2), \dots, (x_n, c_n)) \in (\mathbb{R}^2)^n$, then $Qy = (x_1, \dots, x_n)$, $Sy = (c_1, \dots, c_n)$

We can now state the Algorithm

Algorithm 1 Projection: Input: $x \in \mathbb{R}^n$ and $c_1, \dots, c_n \in \mathbb{R}$.

1. For comfortability, reorder $\{c_n\}$ such that $c_i \leq c_{i+1} \forall i \in [1, \dots, n-1]$.
2. Set $y = ((x_1, c_1), (x_2, c_2), \dots, (x_n, c_n)) \in (\mathbb{R}^2)^n$
3. Set z to be a vector with the same components as y such that Qz is in non – increasing order.
4. Output: $p = Sz$

6.2.2 Sudoku Modeled as a zero – one problem.

Another way to formalize the Sudoku problem is a zero – one problem, as follows:

We'll reformulate the matrix $A \in \mathbb{R}^{9 \times 9}$ as $B \in \mathbb{R}^{9 \times 9 \times 9}$ such that:

$$B(i, j, k) = \begin{cases} 1, & \text{if } A(i, j) = k \\ 0, & \text{otherwise} \end{cases}$$

This reformulation transforms the entries into a 3 – dimensional zero – one array, which can be thought of as a cube, seen in figure 7.

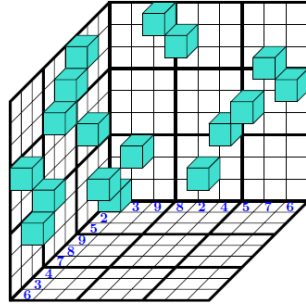


Figure 7: Zero-one representation of a Sudoku puzzle

Denote by S' the $9 \times 9 \times 9$ zero – one array corresponding to the unsolved sudoku puzzle S .

Let $I := \{1, 2, \dots, 9\}$ and $J' \subseteq$

I^3 be the set of indices for which S' is filled, and let \mathcal{B} denote the 9 – dimensional standard basis.

The four constraints of the integer problem, now become the following constraints:

$$C_1 := \{B \in \mathbb{R}^{9 \times 9 \times 9} : B(i, :, k) \in \mathcal{B} \forall i, k \in I\}$$

$$C_2 := \{B \in \mathbb{R}^{9 \times 9 \times 9} : B(:, j, k) \in \mathcal{B} \forall j, k \in I\}$$

$$C_3 := \{B \in \mathbb{R}^{9 \times 9 \times 9} : \text{vec} B_{3i+1:3(i+1), 3j+1:3(j+1), k} \in \mathcal{B} \text{ for } i, j = 0, 1, 2 \text{ and } k \in I\}$$

$$C_4 := \{B \in \mathbb{R}^{9 \times 9 \times 9} : B(i, j, k) = 1 \forall (i, j, k) \in J'\}$$

where $\text{vec} A$ is the vectorization of A by columns.

Further, since each cell of the sudoku puzzle must be filled exactly by one number, we must add the additional constraint:

$$C_5 := \{B \in \mathbb{R}^{9 \times 9 \times 9} : B(i, j, :) \in \mathcal{B} \forall i, j \in I\}$$

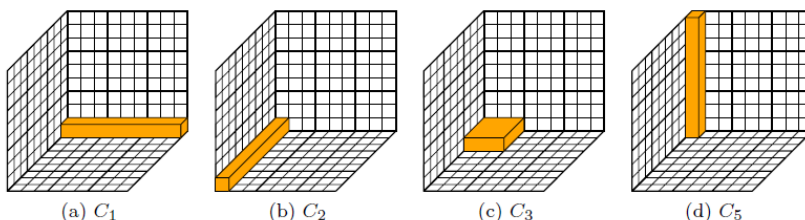


Figure 8: Visualization of the constraints used for modeling Sudoku as a zero-

Then, B completes S' (And thus solves the sudoku) if and only if $B \in \bigcap_{i=1}^5 C_i$.

Again, the projection onto C_4 can be calculated componnet wise, while the projections onto C_1, C_2, C_3, C_5 can be easily calculated using Propisition 2.

6.2.3 Performance of the Douglas – Rachford method on Sudoku Puzzles

As observed in [25], the Douglas –

Rachford method was totally ineffective for solving the integer formulation.

On the other hand, it was seen that the algorithm is very successful when it is applied to the binary formulation,

being able to solve nearly all the instances in all the Sudoku libraries presented in [25],

Even in the one library that it was the most unsuccessful,

the top95 library, it had a success rate of 87%.

Another way to formulate Sudoku as feasibility problems was tested in [20,26],

based on the fact that an unsolved sudoku can be viewed as a graph precoloring problem.

The rank formulation presented in [26] had a success rate of 100% in the top95

library, and no sudoku has been found so far for which the Douglas Rachford method fails to find it's solution for every initial point.

Therefore, we can conclude that the way we formalize our problem,

has huge impacts on the results we'll get from the Douglas Rachford method,

when applied in the nonconvex set.

6.3. Finding Magic Squares as a feasibility problem

In this section, which is also the main section, where we discuss our main objective of this project, to find magic squares, as a feasibility problem. All the information that was given about Projections and Sudoku is very relevant to this next section.

We'll propose two different methods to solve magic squares as a feasibility problem, one is an integer formulation, and the other one is a binary formulation.

Both are inspired by the testst that were done on Sudoku.

To formulate the search, we first thing we need to do is to choose some sets, whose intersection, includes all the properties that define a magic square of order n .

These properties are:

The sum of each row, column, both main diagonals, is constant, also known as the Magic Constant $M = \frac{n(n^2+1)}{2}$,

and it must contain all numbers between 1 and n^2 . Note that if our goal would be to fill a prefilled Magic Square, we'd need to add additional constraints that fix those prefilled numbers.

A natural approach to look for magic squares, would be through the use of Integer Formulation.

Afterwards, we'll show how to formulate the search for magic squares as a Binary Problem.

6.3.1 Magic Squares Modeled as Integer Problems

Denote \mathcal{P} the set of permutations of $1, 2, \dots, n^2$, and $I := \{1, 2, \dots, n\}$,

Then $A \in \mathbb{R}^{n \times n}$ is a magic square if and only if $A \in \bigcap_{i=1}^5 C_i$, where

$$C_1 := \{A \in \mathbb{R}^{n \times n} : \sum_{j \in I} A_{i,j} = c \ \forall i \in I\}$$

$$C_2 := \{A \in \mathbb{R}^{n \times n} : \sum_{i \in I} A_{i,j} = c \ \forall j \in I\}$$

$$C_3 := \{A \in \mathbb{R}^{n \times n} : \sum_{i \in I} A_{i,i} = c\}$$

$$C_4 := \{A \in \mathbb{R}^{n \times n} : \sum_{i \in I} A_{n+1-i,i} = c\}$$

$$C_5 := \{A \in \mathbb{R}^{n \times n} : \text{vec} A \in \mathcal{P}\}$$

The first four sets are clearly convex, as I shall prove for one of the sets.

The proofs for the other is similar.

Claim: C_1 is convex.

Proof:

Take $A, B \in C_1$, then

The sum of each row of A is equal to c .

The sum of each row of B is equal to c .

Take $\lambda \in [0, 1]$, then:

$$\lambda * A + (1 - \lambda) * B$$

Since we know that each row in A equals c ,

then each row in $\lambda * A$ is $\lambda * c$

And for $(1 - \lambda)B$ the sum of each row is $(1 - \lambda)c$

So if we add up $\lambda * A + (1 - \lambda) * B$ via rows,

we'll get for each row the sum $\lambda c + c - \lambda c = c$

and therefore C_1 is convex.

DONE.

Same for C_2, C_3 and C_4 .

Their projection operators have a closed form determined by the next result:

Proposition 3:

Consider $S = \{x \in \mathbb{R}^m : \sum_{i=1}^m x_i = c\}$. For any $x \in \mathbb{R}^m$.

$$P_S(x) = x + \frac{1}{m}(c - \sum_{i=1}^m x_i)e \text{ where } e = \begin{pmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{pmatrix}.$$

Proof:

This follows from the standard formula for the orthogonal projection, onto a hyperplane, since $S = \{x \in \mathbb{R}^m, \langle x, e \rangle = c\}$.

Thereby, the projections onto each one of these sets are given by:

$$P_{C_1}(A) = A + \frac{1}{n} \begin{pmatrix} c - \sum_{i=1}^n A_{1,i} & \dots & c - \sum_{i=1}^n A_{1,i} \\ & \ddots & \\ c - \sum_{i=1}^n A_{n,i} & \dots & c - \sum_{i=1}^n A_{n,i} \end{pmatrix}$$

$$P_{C_2}(A) = A + \frac{1}{n} \begin{pmatrix} c - \sum_{i=1}^n A_{i,1} & \dots & c - \sum_{i=1}^n A_{i,1} \\ & \ddots & \\ c - \sum_{i=1}^n A_{i,n} & \dots & c - \sum_{i=1}^n A_{i,n} \end{pmatrix}$$

$$P_{C_3}(A) = A + \frac{1}{n}(c - \sum_{i=1}^n A_{i,i})I_n$$

$$P_{C_4}(A) = A + \frac{1}{n}(c - \sum_{i=1}^n A_{i,n+i-1})I_n^T$$

Finally, C_5 is a nonconvex set containing all matrices whose entries are permutations of $1, 2, \dots, n^2$, so its projection operator is determined by Proposition 2. Therefore, given a $n \times n$ matrix, to compute its projection onto C_5 ,

we just need to sort the elements of the matrix in ascending order, and place the number 1 in the cell that contains the smallest number, 2 in the cell that contains the next smallest number, and so on.

If it happens that there are two equal elements, then the projection of the matrix won't be unique.

Completing a Partially Filled Magic Square

If M is a partially complete matrix, representing a partially complete magic square, denote by $J \subseteq I^2$ the set of indices for which M is prefilled.

In order to find a completion of the magic square, we need to add the following constraint:

$$C_6 := \{A \in \mathbb{R}^{n \times n} : A_{i,j} = M_{i,j} \ \forall (i,j) \in J\}$$

Therefore, A completes M if and only if, $A \in \bigcap_{i=1}^6 C_i$

The projection onto C_6 is given componentwise by:

$$P_{C_6}(A) = \begin{cases} M_{i,j}, & (i,j) \in J \\ A_{i,j} & \text{otherwise} \end{cases} \quad \forall (i,j) \in I^2$$

6.3.2. Magic Squares Modeled as Binary Problems

In order to model the search for magic squares, using a binary feasibility problem, we'll reformulate $A \in \mathbb{R}^{n \times n}$ as

$$B \in \mathbb{R}^{n \times n \times n^2} \text{ s.t. } B(i, j, k) = \begin{cases} 1, & \text{if } A(i, j) \geq k \\ 0, & \text{otherwise} \end{cases}$$

That way, we transform the entries of our matrix, into 3 dimensional zero – one arrays, and each number in the magic square can be thought of as a pillar made up of single cubes.

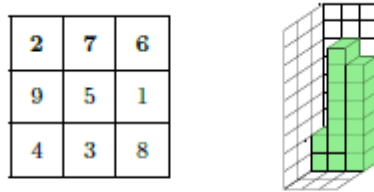


Figure 9: Representation of the numbers in the first row of a 3×3 magic square as columns of small cubes for the binary formulation

The constraints we had in the Integer Feasibility Problem now become the following constraints:

$$C_1 := \left\{ B \in \mathbb{R}^{n \times n \times n^2} : \sum_{j=1}^n \sum_{k=1}^{n^2} B_{i,j,k} = c \quad \forall i \in I \right\}$$

$$C_2 := \left\{ B \in \mathbb{R}^{n \times n \times n^2} : \sum_{i=1}^n \sum_{k=1}^{n^2} B_{i,j,k} = c \quad \forall j \in I \right\}$$

$$C_3 := \left\{ B \in \mathbb{R}^{n \times n \times n^2} : \sum_{i=1}^n \sum_{k=1}^{n^2} B_{i,i,k} = c \right\}$$

$$C_4 := \left\{ B \in \mathbb{R}^{n \times n \times n^2} : \sum_{i=1}^n \sum_{k=1}^{n^2} B_{i,n-i+1,k} = c \right\}$$

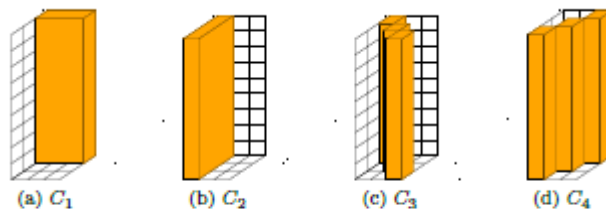


Figure 10: Visualization of the constraints used for modeling magic squares as a zero-one program. Each colored block must be formed by exactly c cubes (ones)

Constraint 5 that we had for the integer feasibility problem now becomes the following intersection of

two sets:

$$C_5 := \{B \in \{0,1\}^{n \times n \times n^2} : \sum_{i=1}^n \sum_{j=1}^n B(i, j, k) = n^2 - k + 1, \forall k \in [1, 2, \dots, n^2]\}$$

$$C_6 := \{B \in \{0,1\}^{n \times n \times n^2} : \text{vec} B_{i,j,:} \in \mathcal{B} \forall i, j \in I\}$$

where $\mathcal{B} := \{[1, 1, \dots, 1], [1, 1, \dots, 0], \dots, [1, 0, 0, \dots, 0]\}$ is a base of vectors of \mathbb{R}^{n^2} .

On the one hand, constraint 5 guarantees that the first floor of B is all filled with ones, the second floor must have

$n^2 - 1$ ones, and so on until the last floor, the n_{th}^2 floor will have exactly 1 one.

On the other hand, constraint 6 guarantess that the matrix is formed by pillars of ones standing on the floor, so if there is a one in an entry, all the elements below it must be ones too.

Therefore, B is magic square if and only if

$$B \in \bigcap_{i=1}^6 C_i$$

The projections onto the first four sets are given, and can be calculated using Proposition 3.

$$P_{C_1}(B) = B + \frac{1}{n^3} \begin{pmatrix} u_1 & \dots & u_1 \\ u_2 & \dots & u_2 \\ \cdot & & \cdot \\ \cdot & & \cdot \\ u_n & \dots & u_n \end{pmatrix}$$

$$P_{C_2}(B) = B + \frac{1}{n^3} \begin{pmatrix} v_1 & v_2 & \dots & v_n \\ v_1 & v_2 & \dots & v_n \\ \cdot & & & \cdot \\ \cdot & & & \cdot \\ v_1 & v_2 & \dots & v_n \end{pmatrix}$$

$$P_{C_3}(B) = B + \frac{1}{n^3} \left(c - \sum_{i=1}^n \sum_{k=1}^{n^2} B_{iik} \right) \begin{pmatrix} e & 0_{n^2} & \dots & 0_{n^2} \\ 0_{n^2} & e & \dots & 0_{n^2} \\ \cdot & & & \cdot \\ \cdot & & & \cdot \\ 0_{n^2} & \dots & 0_{n^2} & e \end{pmatrix}$$

$$P_{C_4}(B) = B + \frac{1}{n^3} \left(c - \sum_{i=1}^n \sum_{k=1}^{n^2} B_{i, n-i+1, k} \right) \begin{pmatrix} 0_{n^2} & 0_{n^2} & \dots & e \\ 0_{n^2} & \dots & e & 0_{n^2} \\ \cdot & & & \cdot \\ \cdot & & & \cdot \\ e & 0_{n^2} & \dots & 0_{n^2} \end{pmatrix}$$

where $e = (1, \dots, 1) \mathbb{R}^{n^2}$ and

$$u_p := \left(c - \sum_{j=1}^n \sum_{k=1}^{n^2} B_{p,j,k}, \dots, c - \sum_{j=1}^n \sum_{k=1}^{n^2} B_{p,j,k} \right) \in \mathbb{R}^{n^2}$$

$$v_p := \left(c - \sum_{i=1}^n \sum_{k=1}^{n^2} B_{p,j,k}, \dots, c - \sum_{i=1}^n \sum_{k=1}^{n^2} B_{p,j,k} \right) \in \mathbb{R}^{n^2}$$

$$\forall i,j \in I:$$

$$P_{C_6}(B)_{i,j} = \operatorname{argmin}_{b \in \mathcal{B}} \left| B_{i,j} - b \right|$$

*And if we want to complete a partially filled magic square,
it is analogous to the integer problems, so we'll not get into that.*

7 Implementation and Experimentation of Magic Squares

In order to implement the algorithms we've learned, we'll use Matlab as our workplace.

We'll implement both formulations, the integer one and the binary one, and compare how different formulations affect the results of the algorithm.

For each formulation, we'd like our algorithm to run on the same space product formulation we've already discussed.

As for a termination criteria, we'll either terminate after 30 minutes of runtime, or after finding a solution.

By Theorem 1, we know that the Shadow Sequene $\{P_D(x_n)\}$ converges weakly to $P_D(x^) \in C \cap D$,*

in the case where $C \cap D \neq \emptyset$. But we know that $C \cap D \neq \emptyset$ since Magic Squares do in fact exist.

Therefore by this theorem, The set $\{P_D(x_n)\}$ converges to a magic square.

7.1 Implementation of the Integer Feasibility Problem:

Take $x_0 := (y, y, y, y, y) \in D$ for some $y \in \mathbb{R}^{n \times n}$ randomly chosen with entries $[0,1]$.

We know already how to calculate $P_D(x_n)$ and $P_C(x_n)$, since the closed forms are given in Propositions 2 and 3.

We also know that $x_{n+1} = x_n + P_D(2P_C(x_n) - x_n) - P_C(x_n)$

And for our algorithm not to run infinitely long, we'll set a limit to how long the algorithm will run, or how close to convergence we'd like to be. Meaning, we can limit our algorithm to work let's say 30 minutes, or we can also stop the algorithm when the following upholds:

$$\left\| \text{round}(P_D(x_n)) - P_{C_i}(\text{round}(P_D(x_n))) \right\| \leq$$

$0.05 \quad \forall i$, where $\text{round}(\cdot)$ gives the nearest rounded integer, if it's a vector,

then it is done componentwise. This also means that $P_D(x_n)$ is very close to every set, meaning that $P_D(x_n)$ is close to intersect every set.

7.2 Algorithms for Magic Squares

Main Algorithm:

Input: n , Output: n by n magic Square.

We used the douglas –

rachford algorithm here, given by DROperator function. We used the DR + Proj variant, after 400,800,1600,3200, ... iterations.

After calculating our current Diagonal Matrix, we check how close it is to the set C , and if it is close enough, we can conclude that our Diagonal Matrix is a Magic Square.

```
function [o1,o2,o3] = MagicSquare(dim)

y=randi([0 1],dim);
xn=zeros(dim,dim,4);
for i=1:5
    xn(:, :, i)=y;
end
|
t=0;
k=1;
j=1;
cnt=0;

tic
while toc<3 && k<100000
    if k==400*power(2,t)
        t=t+1;

        xn=ProjectionD(DROperator(xn));
        pD=ProjectionD(xn);
        pCpD=ProjectionC(pD);
        RpCpD=ProjectionC(round(pD));

        for i=1:5
            if norm(round(pD(:, :, i)) - (RpCpD(:, :, i))) <=.05
                cnt=cnt+1;
            end
        end

        else

            xn=DROperator(xn);
            pD=ProjectionD(xn);
            pCpD=ProjectionC(pD);
            RpCpD=ProjectionC(round(pD));

            for i=1:5
                if norm(round(pD(:, :, i)) - (RpCpD(:, :, i))) <=.05
                    cnt=cnt+1;
                end
            end

            end

            k=k+1;
            if cnt==5
                k=100000;
                o1=round(pD);
                o2=round(pCpD);
            else
                cnt =0;
            end

        end

        o1=round(pD);
        o2=round(pCpD);
        o3=RpCpD;
        cnt
    end
end
```

Side Algorithms:

Logic of the calculation of the projection onto C_5 :

First we create an ascending array of $[1, \dots, n^2]$.

Then we take our current matrix, transform it into a column vector, and sort it in an ascending order,

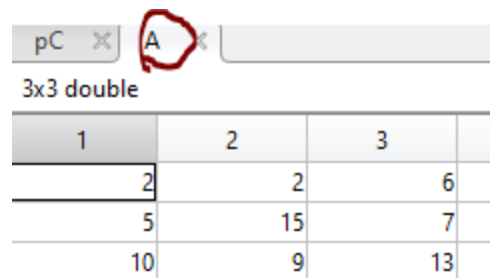
Then for the smallest number in the matrix, we assign the number 1,

the next smallest number gets the number 2, and so on.

If we have two equal numbers in the matrix, then we assign the numbers by the order that the numbers in the matrix were read.

In the end, if our main matrix was $\begin{pmatrix} 2 & 2 & 6 \\ 5 & 15 & 7 \\ 10 & 9 & 13 \end{pmatrix}$, Then the projection of this matrix onto C_5 is $\begin{pmatrix} 1 & 2 & 4 \\ 3 & 9 & 5 \\ 7 & 6 & 8 \end{pmatrix}$.

Which is indeed what we get in Matlab:



1	2	3
2	2	6
5	15	7
10	9	13

`val(:, :, 5) =`

1	2	4
3	9	5
7	6	8

Find Projections onto C:

```
function [pC] = ProjectionC(x)

n=size(x,1);
c=n*(n^2+1)/2;
pC=zeros(n,n,5);

rowsums=zeros(n,1);
columnsums=zeros(n,1);

% Calculating the sum of each row, column, and the diagonals of the matrix.
for i=1:n
    rowsums(i)=sum(x(i,:),1);
    columnsums(i)=sum(x(:,i),2);
end

mainDiagonalSum=sum(diag(x(:, :, 3)));
antiDiagonalSum=sum(diag(flip(x(:, :, 4))));

% Calculating the projections of y onto each of the convex subsets
% We have a closed form.
v1=ones(n,1)*c-rowsums;
v2=ones(1,n)*c-columnsums';

pC1=x(:, :, 1)+(1/n)*repmat(v1,1,n);
pC2=x(:, :, 2)+(1/n)*repmat(v2,n,1);
pC3=x(:, :, 3)+(1/n)*(c-mainDiagonalSum)*eye(n);
pC4=x(:, :, 4)+(1/n)*(c-antiDiagonalSum)*flip(eye(n)); %% (Inverse ID)

v=[1:n*n]';
B=x(:, :, 5)';
B=B(:);
sortB=sort(B); % Ascending Order
proj5Vec=zeros(1,n*n);

% An algorithm to calculate the projection of y onto C5,
% Logic of the algorithm explained in presentation
for i=1:n*n
    f=find(sortB==B(i),1);
    proj5Vec(i)=v(f);
    v(f)=[];
    sortB(f)=[];
end

proj5Vec=reshape(proj5Vec, [n n]);
pC5=proj5Vec';

pC(:, :, 1)=pC1;
pC(:, :, 2)=pC2;
pC(:, :, 3)=pC3;
pC(:, :, 4)=pC4;
pC(:, :, 5)=pC5;
```


Find Projections onto D:

```
function [pD] = ProjectionD(x)

n=size(x,1);

matSum=0;
D=zeros(n,n,5);
for i=1:n
    matSum=matSum+x(:, :, i) % Sum of vectors = A vector
end

matAvg=matSum/n; % Average of those vectors

for i=1:5
    D(:, :, i)=matAvg;
end

pD=D; % 5 times, since we have 5 constraints.
```

Iteration Algorithm:

```
function [x] = DROperator(x0)

ProjC=ProjectionC(x0);
ProjD=ProjectionD(2*ProjC-x0);
x=x0+ProjD-ProjC;
```

7.3 Results

We let the algorithm run either until 10000 iterations have been done, or 30 seconds have passed.

We generate a sequence of elements $\{P_D(x_n)\}$, which:

By Theorem 1,

1. If $A \cap B \neq$

\emptyset , then $\{x_n\}$ is weakly convergent to a point x^ and $\{P_A(x_n)\}$ is weakly convergent to $P_A(x^*) \in A \cap B$.*

We know that Magic Squares do exist. So $D \cap C$ must be non – empty.

As a consequence of this theorem, in our results, we looked at the sequence $\{P_D(x_n)\}$,

And also at the sequence $\{P_C(P_D(x_n))\}$

which was given as a variant in 3. By the theorem, $\{P_D(x_n)\}$ should converge to a magic square.

In all of our results, we haven't received even a single magic square when we looked at $\{P_D(x_n)\}$.

It seems that $P_D(x_n)$ converges to a matrix that belongs to $C_1 \dots \cap$

C_4 , but not to C_5 , meaning the elements

of $P_D(x_n)$ are not a permutation of 1 to n^2 .

That means that the matrix we get does uphold the criteria that each row sums up to the magic constant, so does each

column, and both main diagonals. But the matrix isn't a permutation of 1 to n^2 ,

therefore it is not a magic square.

The sole property that our final matrix is missing, is the property that the matrix also belongs to C_5 , which means that the matrix is a permutation of 1 to n^2 .

The conclusion was either that it didn't work because C_5 was nonconvex so the theorem didn't apply, or that we wrote the algorithm to calculate the projection onto C_5 is wrong, which doesn't seem to be the behavior of C_5 on different matrices. The algorithm to calculate the projection onto C_5 was given in Proposition 2, but perhaps I have misunderstood the logic of this algorithm. Because it does seem, that the way I understood the logic of how to project onto C_5 , and how I wrote the algorithm relative to my understanding, is correct, but perhaps I've misunderstood.

8. Conclusions

To conclude our work, we've seen a few examples of how the Classic Douglas-Rachford and the Cyclic Douglas-Rachford algorithms iterate. We've seen that in some cases, like the 3 Convex sets example, the Classic Douglas-Rachford failed to converge, while the variation of the classic Douglas-Rachford algorithm, the cyclic Douglas-Rachford algorithm, did indeed converge to the point which is the intersection of all the sets. We've seen that the theory and theorems that we've stated, appear to be true in our examples.

What we can conclude from that, is that how we define our iterations, the order we define them, can change a problem from being feasible using a method, to unfeasible, and vice versa.

We've seen that a very useful notion, the Space Product Formulation, can take advantage of the Douglas-Rachford theorem for 2 sets, by creating 2 other sets, called the D (Diagonal) and C sets, which can help us deal with convex and nonconvex problems involving more than 2 sets, potentially, a very big amount of sets, which as a result, we've seen that many valuable mathematical problems from lots of fields, specifically combinatorial problems is what we've went deeper into, even these types of problems, can be solved, and we can find a solution, using the Douglas-Rachford method.

In our magic square case, our algorithm didn't manage to give us a full legit completed Magic Square, which was disappointing. My conclusions as to why this has happened, are written in the result part of the Magic Squares section (section 7).

For the future of the field of Optimization, both in convex and nonconvex settings, I believe we can expect the Douglas-Rachford algorithm to still be successful in many cases, and I also believe that people, with the need to solve different feasibility problems, will come up with various variations of the Classic Douglas Rachford algorithm,

To give an answer to the situations where the Classic Douglas Rachford algorithm does not give us The result, like shown in Example 1 with the 3 Lines.

Bibliography

1. Douglas, J., Rachford, H.: On the numerical solution of heat conduction problems in two (1956) and three space variables.
<https://www.ams.org/journals/tran/1956-082-02/S0002-9947-1956-0084194-4/>
2. Lions, P., Mercier, B.: Splitting algorithms for the sum of two nonlinear operators. (1979)
<https://www.jstor.org/stable/2156649>
3. Borwein, J., Zhu, Q.: Techniques of variational analysis. CMS books in mathematics. (2005)
4. Opial, Z.: Weak convergence of the sequence of successive approximation for nonexpansive mappings. (1967)
<https://www.semanticscholar.org/paper/Weak-convergence-of-the-sequence-of-successive-for-Opial/0f2f3bf20641922294f079923ff36872fcc0a860>
5. Borwein, J., Tam, M.: A cyclic Douglas-Rachford Iteration scheme. (2013)
<https://www.proquest.com/openview/7895f874977fad8b8bedb7f67c8e9afa/1.pdf?pq-origsite=gscholar&cbl=48247>
6. Elser, V., Rankenburg, I.: Deconstructing the energy landscape: Constraint-based algorithms for folding heteropolymers (2006)
<https://journals.aps.org/pre/abstract/10.1103/PhysRevE.73.026702>
7. Elser, V., Rankenburg, I., Thibault, P.: Searching with iterated maps (2007)
<https://www.pnas.org/doi/10.1073/pnas.0606359104>
8. Bauschke, H., Combettes, P., Luke, D.: Phase Retrieval, error reduction algorithm, and Fienup variants: A view from convex optimization. (2002)
<https://opg.optica.org/josaa/abstract.cfm?uri=josaa-19-7-1334>
9. Bauschke, H., Combettes, P., Luke, D.: Hybrid projection-reflection method for phase retrieval. (2003)
<https://opg.optica.org/josaa/abstract.cfm?uri=josaa-20-6-1025>
10. Aragon Artacho, F., Borwein, J., Tam, M.: Douglas-Rachjford feasibility methods for matrix completion problems. (2013)

<http://arxiv.org/abs/1308.4243>

11. Johnson, C.: Matrix completion problems: A survey. (1990)
12. Schaad, J.: Modeling the 8-queens problem and sudoku using an algorithm based on projections onto nonconvex sets. (2010)
<https://open.library.ubc.ca/soa/cIRcle/collections/ubctheses/24/items/1.0071292>
13. Gravel, S., Elser, V.: Divide and concur: A general approach to constraint satisfaction. (2008)
<https://pubmed.ncbi.nlm.nih.gov/18851188/>
14. Garey, R., Johnson, D.: Computers and Intractability: A guide to the Theory of NP-Completeness. (A series of books in Mathematical science)
15. Takenaga, Y., Walsh, T.: Tetravex is NP-Complete. (2006)
<https://arxiv.org/abs/0903.1147>
16. Bansal, P.: Code for solving Tetravex using Douglas-Rachford Algorithm (2010)
<http://people.ok.ubc.ca/bauschke/Pulkit/pulkitreport.pdf>
17. Takayuki, Y., Takahiro, S.: Complexity and completeness of finding another solution and its application to puzzles. (2003)
<https://www.semanticscholar.org/paper/Complexity-and-Completeness-of-Finding-Another-and-Yato-Seta/aa8091f44bd25d23bb53fb9af6257dfeba2355dc>
18. Nagao, T., Ueda, N.: NP-Completeness results for nonogram via parsimonious reductions. (2012)
19. Van Rijn, J.: Playing games: The complexity of Klondike, Mahjongh, nonograms and animal chess. (2012)
https://theses.liacs.nl/pdf/2012-01JanvanRijn_2.pdf
20. Francisco J. Aragon Artacho, Jonathan M. Borwein, Matthew K. Tam.: Recent Results on Douglas–Rachford Methods for Combinatorial Optimization Problems. (2018)
<https://arxiv.org/pdf/1305.2657.pdf>
21. H.H. Bausche.: On the local convergence of the Douglas-Rachford algorithm (2014)

https://www.researchgate.net/publication/259893303_On_the_local_convergence_of_the_Douglas-Rachford_algorithm

22. Robert Hesse, D. Russell Luke.: Nonconvex Nontions of regularity and convergence of fundamental algorithms for feasibility problems. (2013)
<https://arxiv.org/abs/1212.3349>
23. Francisco J. Aragon Artacho, Jonathan M. Borwein, Matthew K. Tam.: Global Behavior of the Douglas-Rachford method for a nonconvex Feasibility problem. (2015)
<https://arxiv.org/abs/1506.09026>
24. Joel Benoist.: The Douglas-Rachford algorithm for the case of the sphere and the line. (2015)
<https://link.springer.com/article/10.1007/s10898-015-0296-1>
25. Jared Weed - Applications of AI for Magic Squares
<https://arxiv.org/pdf/1602.01401.pdf>
26. Peyman F., Ramin. J.: An Introduction to Magic Squares and Their Physical Applications (2015)
https://www.researchgate.net/publication/297731505_An_Introduction_to_Magic_Squares_and_Their_Physical_Applications