An overview of sparse convex optimization

Jacob Mantjitji Modiba

Submitted in partial fulfillment of the requirements for the degree

Magister Scientiae (Mathematical Statistics)

In the Department of Statistics
In the Faculty of Natural and Agricultural Sciences
University of Pretoria

January 2018

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

I, *Jacob Mantjitji Modiba,* declare that this mini-dissertation (100 credits), which I hereby submit for the degree Magister Scientiae in Mathematical Statistics at the University of Pretoria, is my own work and has not previously been submitted by me for a degree at this or any other tertiary institution.

Signature:
Date: January 2018

Summary

Sparse estimation methods are aimed at using or obtaining parsimonious representations of data or models. Optimization is finding the best solution to the function that you want to optimize or seeking values of a variable that leads to an optimal value of the function that is to be optimized. Many real life problems involves solving systems of equations (data set with high dimension) by optimization i.e. by putting certain constraints on the variable of interest or the function of the variable of interest. For instance, multiple linear regression deals with solving the system of equations $Y = X\beta$ for β by minimizing the sum of squared errors i.e. by minimizing $\|\beta\|_2^2$. In this mini-dissertation we discuss the algorithms used in sparse convex optimization. Consider the following system of equations,

$$b = Ax$$

where A is an $m \times n$ matrix with more columns than rows ($m < n$), b and x are $m \times 1$ and $n \times 1$ vectors respectively. To solve for x in this system, will result in infinitely many solutions since the system has more columns than rows and is hence under-determined. If one has prior knowledge that x is sparse, the problem can be turned into an optimization problem. This results in solving the above system by minimizing the number of non-zero elements in x. The problem can be expressed as follows,

$$\min_{Ax=b} \|x\|_0 .$$

This is a non-convex problem and it is known that non-convex problems are hard to solve. One way to approximate the above problem in a convex manner is to use a method called basis pursuit. Basis pursuit minimizes the sum of the absolute value of the elements in x i.e.$\|x\|_1$ instead of $\|x\|_0$. The problem can be expressed as follows,

$$\min_{Ax=b} \|x\|_1 .$$

This becomes a sparse convex optimization problem and it has been proven that it provides a good approximation to the original solution. Basis pursuit is not the only method to be considered here. We will compare different type of methods for solving sparse optimization problems to see which of the methods gives the best results i.e.

approximates the ℓ_0-minimization best. These methods include orthogonal matching pursuit, convex relaxation, iterative hard threshold and iterative soft threshold. The sparse estimation method can also be used for variable selection or for reducing the dimension of the data. In the context of regression, sparse methods are used to find the simplest representation of the model at hand, and to make the prediction more interpretable and easy to compute.

We look at a sudoku problem transformed into a system of linear equations in the following manner:

$$Ax = \begin{bmatrix} A_{row} \\ A_{col} \\ A_{box} \\ A_{cell} \\ A_{clue} \end{bmatrix} x = \begin{bmatrix} 1 \\ 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix} = b$$

where A is the matrix containing clues, x is the solution vector of the sudoku problem and b is the indicator vector of ones, where 1 represents that the constraints are true. We applied the algorithms to 100 unique sudoku problems with 17 clues. The times taken to solve the 100 sudoku puzzles were recorded, analyzed and compared, for all algorithms.

Acknowledgements

I would like to thank my supervisor Dr Fabris-Rotelli for all the support and guidance. I would also like to thank the sponsor for my masters programme, namely STATOMET and CAIR. A poster was presented at the 2017 South African Statistical Association hosted by the University of the Free State.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

1.1 Why sparse convex optimization?

Convex optimization techniques are mostly preferred over non-convex optimization. This is due to the fact that convex optimization always results in a global optimum solution [4], that is if there are many local optima, all of these local optima will give the same minimum or maximum value. When optimizing a non-convex function with many local optima, it is difficult and time consuming to find out which of the local optima is a global optimum [4]. For the above reason, optimizing a convex function is faster as compared to optimizing a non-convex function.

Suppose we have a system of linear equations where there are more unknowns than the equations. This type of system leads to infinitely many solutions. If one has prior knowledge that the solution is sparse this problem can be treated as an optimization problem with a unique solution. Next, the mathematics of the elements of optimization, convexity, and sparsity, are discussed, in that order.

1.2 Optimization

Definition 1. Optimization [10] aims to find the best solution to the function that is to be optimized or seeking values of a variable that leads to an optimal value of the function that is to be optimized. A solution for an optimization problem is optimal if it minimizes or maximizes the function in study.

Optimization techniques are applied in many areas for example economics, and strategic planning [4]. In economics optimization can be used for maximizing the profit or for minimizing the costs. In a simple form optimization can be used for finding the minimum or maximum value of a parabola in its domain. As an example, suppose we have a parabola of the form $f(x) = x^2 + 1$ in the domain $[-2, 2]$ and we want to find an x value that minimizes the function. The following table represents the values of f for x in the domain.

x	-2	-1	0	1	2
$f(x)$	5	2	1	2	5

The function is minimum at $x = 0$, with a minimum value of $f(0) = 1$. This means that $f(x)$ is greater than or equal to 1 for all x in the domain $[-2, 2]$.

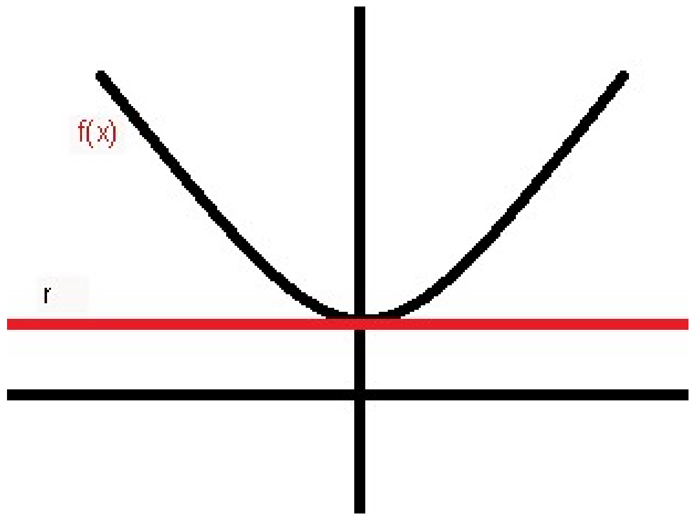


Figure 1.1: Plot of $f(x)$.

Figure 1.1 represents the plot $f(x) = x^2 + 1$. The reference line represented by r is at $f(0) = 1$. This line shows that $f(x)$ is on or above 1 and that the minimum value for f is 1.

Another example of optimization is finding the minimum or maximum value of a function under some certain constraints. For example, regression and linear programming. The first step in optimization problems is identifying the constraints and the objects that need to be optimized. We should note that the solution to an optimization problem should always satisfy the objective of optimization and the constraints, if the solution does not satisfy the constraints we say the solution is not feasible.

Definition 2. A mathematical optimization problem or simply an optimization problem is formulated as follows,

$$\min_{f_i(x) \leq b_i} f_0(x), \qquad i = 1, ..., m \tag{1.1}$$

where the function $f_0 : \mathbb{R}^n \to \mathbb{R}$ is the objective function, the functions $f_i : \mathbb{R}^n \to \mathbb{R}$ for $i = 1, \dots, m$ are the constraints functions, $x = (x_1, \dots, x_n)^T$ is the optimization variable and $b_1, \dots, b_m$ are the limits or bounds of the constraints [4].

In problem (1.1) we want to minimize $f_0(x)$ subject to the constraints $f(x_i) \leq b_i$. A vector x^* is called optimal, or a solution of problem (1.1), if it has the smallest objective value among all vectors that satisfy the constraints [4]. A point x is feasible if it satisfies the the constraints $f(x_i) \leq b_i$ for $i = 1, \ldots, m$ [4], and a set of all feasible solutions is called the feasible region. A simple example of mathematical optimization is linear programming, that is, the case of linear objective and constraint functions.

1.3 Convexity

1.3.1 Convex sets

Definition 3. A subset S of some vector space is convex if the line segment joining any two points p and q in S is contained in the set S, or equivalently, a set is convex if all the points on the line segment connecting any two points is S are contained in S. Mathematically, the subset S is convex if

$$L = \alpha p + (1 - \alpha)q \in S \tag{1.2}$$

where α, $\beta \in [0, 1]$ and $\alpha + \beta = 1$ [4].

To explain equation (1.2), suppose that we have a subset S of some vector space and we choose any two points p and q in S. Suppose any line segment joining p and q is represented by h. A subset S is convex if all the points lying on h are contained in S. Any point on a straight line h can be represented by $L = \alpha p + (1 - \alpha)q$ for $0 \leq \alpha \leq 1$, such that $L = p$ if $\alpha = 1$ and $L = p$ if $\alpha = 0$ otherwise $p < L < q$. Therefore a subset S is convex if $L = \alpha p + (1 - \alpha)q \in S$.

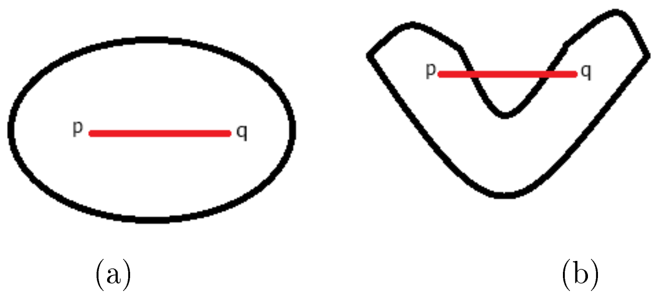


Figure 1.2: Example of (a) convex set and (b) non-convex set

Figure 1.2(a) represents an example of a convex set. Here p and q are any points in the set in (a). From Figure 1.2(a) we can see that the line segment joining the points p and q is contained in the set. Therefore by definition the set in (a) is convex. Figure 1.2(b) represents an example of a non-convex set. Here p and q are any points in the set in (b). From Figure 1.2(b) we can see that the line segment joining the points p

and q is at some point outside the set. This does not satisfy the definition of a convex set. Therefore the set in (b) is not convex.

1.3.2 Convex functions

Definition 4. If a straight line connecting two points on the function is on or above that function, we say that the function is convex [4]. Mathematically, the function f is convex if the following property holds,

$$f(\alpha x + (1-\alpha)y) \leq \alpha f(x) + (1-\alpha) f(y), \tag{1.3}$$

where x and y are in the domain of the function f, $\alpha \in [0,1]$. A function is concave if it satisfies the following property,

$$f(\alpha x + (1-\alpha)y) \geq \alpha f(x) + (1-\alpha) f(y). \tag{1.4}$$

Multiplying (1.3) by -1 leads to

$$-f(\alpha x + (1-\alpha)y) \geq \alpha\left[-f(x)\right] + (1-\alpha)\left[-f(y)\right].$$

This proves that if f is convex then $-f$ is concave. This relationship allows us to study either concave functions or convex functions. Here we will only study convex functions since it does not add value by studying both of them.

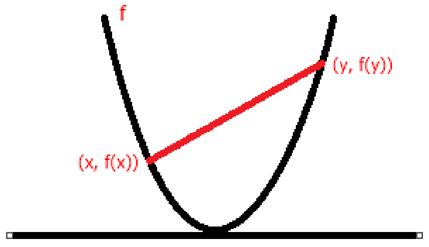


Figure 1.3: Example of a convex function

To explain equation (1.3), consider Figure 1.3. Suppose we choose any point $z = \alpha x + (1-\alpha)y$ that lies on the straight line connecting the points $(x, f(x))$ and $(y, f(y))$. The point is chosen such that if $\alpha = 0$ then $z = y$ and if $\alpha = 1$ then $z = x$, otherwise $x < z < y$. Let $h(z)$ be the height of the line segment connecting $(x, f(x))$ and $(y, f(y))$

at the point $z = \alpha x + (1-\alpha)y$. Since the line segment is linear, we have

$$\begin{aligned} h(z) &= h(\alpha x + (1-\alpha)y) \\ &= \alpha h(x) + (1-\alpha)h(y). \end{aligned}$$

Here $h(x)$ and $h(y)$ are the heights of the line segment at points x and y respectively. In Figure 1.3 the heights at point x and y are represented by $f(x)$ and $f(y)$ respectively. This implies that $h(x) = f(x)$ and $h(y) = f(y)$. Therefore $h(z) = \alpha f(x) + (1-\alpha)f(y)$. The function f is convex if the height of the line segment joining any two points in the domain of f is greater or equal to the height of f at the same point. i.e. f is convex if

$$f(\alpha x + (1-\alpha)y) \leq \alpha f(x) + (1-\alpha)f(y).$$

In Figure 1.3 the line joining any two points in the domain of f will always be above f, therefore the function is convex. Convex functions have an advantage over non-convex function since any local optimum is also a global optimum [4].

Definition 5. A function is strictly convex if a straight line connecting two points on the function is strictly above that function. i.e. if,

$$f(\alpha x + (1-\alpha)y) < \alpha f(x) + (1-\alpha)f(y), \tag{1.5}$$

where x and y are in the domain of the function f, $\alpha \in [0,1]$ [4].

A function that is strictly convex results in the unique global optimum. A non-convex function has more than one local optimum, and to find a global optima on a function with a high dimension is difficult. Therefore the use of convex functions for optimization yields the best solutions [4]. Note that all linear functions are both convex and concave. Suppose we have a linear function $f(z) = mz + c$ and we choose any two points in the domain of $f(z)$, say x and y. Any point on the straight line connecting the points $(x, f(x))$ and $(y, f(y))$ is represented by $p = \alpha x + (1-\alpha)y$ then,

$$\begin{aligned} f(\alpha x + (1-\alpha)y) &= m(\alpha x + (1-\alpha)y) + c \\ &= \alpha mx + (1-\alpha)my + c \end{aligned}$$

by adding $(\alpha c - \alpha c)$ and $[(1-\alpha)c - (1-\alpha)c]$ on the left hand side we get,

$$\begin{aligned} f(\alpha x + (1-\alpha)y) &= \alpha mx + \alpha c - \alpha c + (1-\alpha)my + (1-\alpha)c - (1-\alpha)c + c \\ &= \alpha(mx + c) + (1-\alpha)(my + c) - \alpha c - c + \alpha c + c \\ &= \alpha f(x) + (1-\alpha)f(y) \end{aligned}$$

This satisfies both equations (1.3) and (1.4).

Definition 6. A loosely convex function is a convex function that has multiple local optima which are all global optima [4].

Figure 1.4 represents a function that is loosely convex. The lowest that f can achieve is at $f(z)$ where z is any x value between a and b. This shows that all x values between a and b are local minima and since they are minimum at the same f value, they are all global optima.

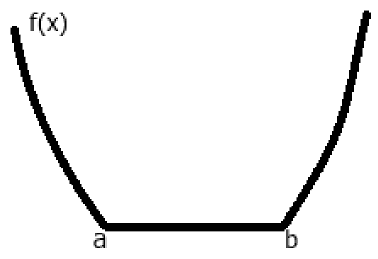


Figure 1.4: Example of a loosely convex function

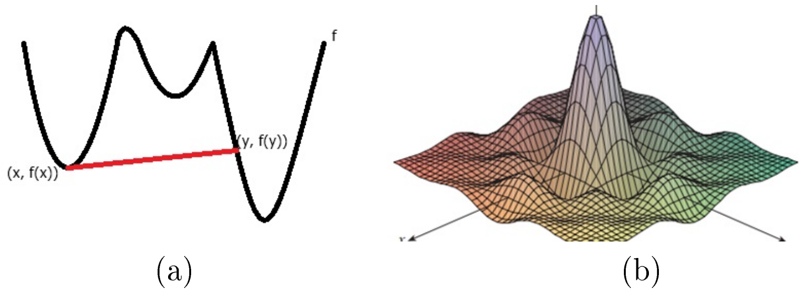


Figure 1.5: Example of a non-convex function (a) in 2D (b) in 3D [37]

Figure 1.5(a) shows an example of non-convex functions. This function is non-convex since the line joining any two points $(x, f(x))$ and $(y, f(y))$ is, at some point below the function, specifically for $x \in [-2, 1.8]$. Non-convex functions have more than one local optima and not all of them share the same f value. Therefore not all local optima in a non-convex function are global. To find a global optima, one needs to search through all the local optima to find out which one gives the optimum f value (minimum or maximum). Figure 1.5(a) has three local minima. To find the global minimum, we need to check which x value give the lowest f value. Since this is a $2D$ function, it is easier to see from the plot which x value is a global minimum. When you dealing with functions of more than one variable as in Figure 1.5(b) it is difficult to find a global optimum straight from the plot, some functions are even difficult to plot. To find a global optimum, one needs to test whether all local minima are global. This process takes time. On the other hand, for a convex function any local optima will also be globally optimum [4]. Although when using a computer it is possible to find a solution for a non-convex optimization problem, it will take more time to find the solution as compared to when solving a convex optimization problem. If the function is strictly convex, then the global optimum will be unique. Even though a loosely convex function has many local optima, we know that they are all global since they share the same optimum f value. Therefore the efficiency in time of a convex optimization problem is much better than for a non-convex optimization problem.

1.4 Sparsity

Sparsity refers to extent to which a measure (such as a matrix or a vector) contains a null, where a null value can be a missing value or a zero value. A dataset with many null values is called sparse. The concept of sparsity is used in many fields of study for example face recognition [41], and image processing [16]. In machine learning, sparsity has played an important role in developing algorithms such as matrix factorization [20], and support vector machines [24]. A vector or a matrix is said to be sparse if most of its elements are zeros, where most is measured by the ℓ_0-norm. There are different types of sparse matrix representation such as, compressed sparse row and compressed sparse column representation [5]. We provide a formal definition.

Definition 7. A vector x of dimension $n \times 1$ is k-sparse if at most k of its elements are non-zero for $1 \leq k \leq n$ [7]. In terms of norms, x is k-sparse if $||x||_0 \leq k$ [7].

The ℓ_0-norm is the theoretical measure for how sparse a vector is. For example, if x and y are two $n \times 1$ vectors then x is called more sparse than y if $\|x\|_0 < \|y\|_0$. There are various reasons for using sparse vectors. For example, sparse solutions are easy to interpret and they have an advantage in terms of computational purposes. In regression, the sparse representation is used to represent complex models with fewer information than we originally have i.e. with fewer regression parameters [22]. In compressed sensing if we know that the signal has sparse representation, the signal can be recovered from a small number of non-adaptive measurements [15]. One of the objectives in multivariate analysis is to estimate the covariance matrix. When there are more variables than observations it is difficult to estimate the covariance matrix, but if one imposes sparsity on the covariance matrix it can be estimated uniquely [23]. Suppose that we want to solve a system of equations $Ax = b$ where matrix A is an $m \times n$ matrix with more columns than rows i.e. the system has more unknowns than equations. Solving this system of equations will result in infinitely many solutions, but if one considers only sparse solutions, we can find a unique solution [7].

1.5 Conclusion

This chapter has introduced briefly the concepts of optimization, convexity and sparsity. In what follows, the following aims of this mini-dissertation will be expanded on:

- Present algorithms for sparse convex optimization.
- Present two algorithms for non-convex sparse optimization.
- Discuss the advantages and disadvantages of each algorithm presented.
- Implement the algorithms to solve sudoku problem.
- Finally, compare the performance of sparse convex optimization to sparse non-convex optimization on the application at hand.

In the next chapter we will discuss the mathematical concepts that will be needed for discussing the algorithms presented in Chapter 3.

Chapter 2

Definitions and notation

In this chapter we provide the mathematics necessary for the optimization algorithms presented in Chapter 3.

2.1 The ℓ_p-norm of a vector

Definition 8. A norm of a vector is the magnitude or length of that vector. Suppose $x = (x_1, x_2 \ldots, x_n)$ is a vector in some vector space V then we symbolize its norm by $\|x\|$. The following are the properties that define a norm of a vector in a vector space V [19],

i) $\|x\| \geq 0 \ \forall x \in V$ and $\|x\| = 0$ iff $x = 0$

ii) $||\alpha x|| = |\alpha|.\|x\| \ \forall \alpha \in R, \ x \in V$ (absolute homogeneity) and

iii) $\|x + x\| \leq \|x\| + \|x\| \ \forall x, y \in V$ (triangle inequality)

An ℓ_p-norm of a vector x is defined by the following formula.

$$\|x\|_p = \left(\sum_{i=1}^{n} |x_i|^p \right)^{1/p}, \tag{2.1}$$

where n is the number elements in x [19].

Theorem 9. *$\|x\|_p$ for $p \geq 1$ are norms.*

Proof. Suppose we have $x, y \in V$ with $\alpha \in \mathbb{R}$. Then,

- since $|x| \geq 0$ then $\|x\|_p \geq 0$

- $$\begin{aligned} \|\alpha x\|_p &= \left(\sum_{i=1}^{n} |\alpha x_i|^p\right)^{1/p} \\ &= \left(\sum_{i=1}^{n} |\alpha|^p \mid x_i|^p\right)^{1/p} \\ &= |\alpha| \parallel x \parallel_p \end{aligned}$$

- $$\begin{aligned} \|x+y\|_p &= \left(\sum_{i=1}^{n} |x_i + y_i|^p\right)^{1/p} \\ &\leq \left(\sum_{i=1}^{n} |x_i|^p\right)^{1/p} + \left(\sum_{i=1}^{n} |y_i|^p\right)^{1/p} \quad \text{Minkowski inequality} \\ &\leq \|x\|_p + \|y\|_p \end{aligned}$$

For $0 < p < 1$ equation (2.1) no longer satisfies the triangle inequality since Minkowski's inequality only holds for $p \geq 1$ [29].

Theorem 10. *All norms are convex functions.*

Proof. Suppose $f(z) = \|z\|$ is a norm in some vector space V. This implies that for $x, y \in V$, $k \in \mathbb{R}$ and $0 \leq \alpha \leq 1$ we have $\|kz\| = |k|\|z\|$ (homogeneity property) and $\|x+y\| \leq \|x\| + \|y\|$ (triangle inequality).
If we choose a point $\alpha x + (1-\alpha)y$ that lies on a straight line connecting x and y, then

$$\begin{aligned} f(\alpha x + (1-\alpha)y) &= \|\alpha x + (1-\alpha)y\| \\ &\leq \|\alpha x\| + \|(1-\alpha)y\| \text{ triangle inequality} \\ &= |\alpha|\|x\| + |1-\alpha|\|y\| \text{ homogeneity property} \\ &= \alpha f(x) + (1-\alpha)f(y) \text{ since } \alpha \geq 0, \end{aligned}$$

showing that $f(x)$ is convex.

Definition 11. The ℓ_0-norm is defined to be the number of non-zero elements in a vector. Mathematically it is defined as follows,

$$\|x\|_0 = \#\{x_i \mid x_i \neq 0\} \tag{2.2}$$

where x is some vector and x_i are the elements of x.

Theorem 12. *The ℓ_0-norm is however not a norm despite its name.*

Proof. To prove that a function is not a norm, it is sufficient to prove that the properties of norms are false. We show that ℓ_0 does not satisfy the homogeneity property of a norm. If we multiply a vector with a non-zero constant, the zero elements of that vector will remain zero since any number multiplied by zero is zero. This means that the number of non-zero elements will stay the same, so that we have

$$\|\alpha x\|_0 \neq \alpha \|x\|_0. \tag{2.3}$$

This proves that ℓ_0 does not satisfy the homogeneity property and hence it is not a norm. □

Here is an example. Suppose we have a vector $x = \begin{pmatrix} 1 & 2 & 4 & 3 & 0 & 0 \end{pmatrix}^T$. Then $\|x\|_0 = 4$ (the number of non zero elements in x). Suppose we multiply x with some scalar $\alpha \neq 0$ in $\mathbb{R}$, then we will have $\alpha x = \begin{pmatrix} \alpha 1 & \alpha 2 & \alpha 4 & \alpha 3 & 0 & 0 \end{pmatrix}$ so that $\|\alpha x\|_0 = 4$. Therefore $\|\alpha x\|_0 = \|x\|_0$, hence ℓ_0 is not a norm. In general, we have for $\alpha \neq 0$,

$$\|\alpha x\|_0 = \|x\|_0. \tag{2.4}$$

Theorem 13. *The ℓ_0-norm is not a convex function.*

Proof. Suppose we have two vectors $x : n \times 1$ with $\|x\|_0 = m$ and $y : n \times 1$ with $\|y\|_0 = z$. since x has m non-zero elements and y has z non-zero elements, the maximum number of non-zero that $x + y$ can have is $m + z$. This is because the number of non-zero elements of $x + y$ is bounded by the number of non-zero element of x plus the number of non zero element of y. Therefore we have,

$$\|x + y\|_0 \leq \|x\|_0 + \|y\|_0.$$

This means that ℓ_0 satisfies the triangle inequality. Let $f(z) = \|z\|_0$. Let $x, y \in V$ and $0 \leq \alpha \leq 1$ an element of real numbers. If we choose a point $\alpha x + (1 - \alpha)y$ on the straight line connecting x and y then,

$$\begin{aligned} f(\alpha x + (1 - \alpha)y) &= \|\alpha x + (1 - \alpha)y\|_0 \\ &\leq \|\alpha x\|_0 + \|(1 - \alpha)y\|_0 \ \text{(Triangle inequality)} \\ &= \|x\|_0 + \|y\|_0 \text{ by equation (2.4)} \\ &= f(x) + f(y). \end{aligned}$$

Thus ℓ_0 does not satisfy the definition of convex function. □

The ℓ_1-norm ($p = 1$), defined to be the sum of the absolute value of the elements in a vector i.e.

$$\|x\|_1 = \sum_{i=1}^{n} |x_i| \tag{2.5}$$

where x is some vector and x_i are the elements of x, is however a convex function.

Theorem 14. *The ℓ_1-norm is a convex function*

Proof. Let $f(z) = \|z\|_1$ be a norm of vector z. Since $f(z)$ is a norm, the following applies,

- $\|\alpha z\|_1 = |\alpha|\|z\|$ (homogeneity property)
- $\|x + y\|_1 \leq \|x\|_1 + \|y\|_1$ (triangle inequality)

Suppose that $x,\ y\ \in\ V$. If we choose a point $\alpha x + (1-\alpha)y$ on the straight line connecting the points x and y then,

$$\begin{aligned} f(\alpha x + (1-\alpha)y) &= \| \alpha x + (1-\alpha)y \|_1 \\ &\leq \| \alpha x \|_1 + \| (1-\alpha)y \|_1 \ \text{(triangle inequality)} \\ &= |\alpha| \ \| x \|_1 + |1-\alpha| \ \| y \|_1 \ \text{(homogeneity property)} \\ &= \alpha f(x) + (1-\alpha) f(y) \, \text{for} \, \alpha \geq 0 \end{aligned}$$

Therefore the ℓ_1 -norm is convex. Note that in fact we already proved that all ℓ_p for $p \geq 1$ are norms and we proved that all norms are convex, therefore ℓ_1 -norm is convex.

We focus only on the ℓ_0-norm and ℓ_1-norm as they are the role players in sparse convex optimization. Figure 2.1 shows the graphical representations of the ℓ_p norm as p increases from zero to infinity. Figure 2.1 indicate that the ℓ_0 is the limit of the function (2.1) as p goes to zero, i.e.

$$\begin{aligned} \|x\|_0 &= \lim_{p \to 0} \|x\|_p^p \\ &= \lim_{p \to 0} \sum_{i=1}^{n} |x_i|^p. \end{aligned}$$

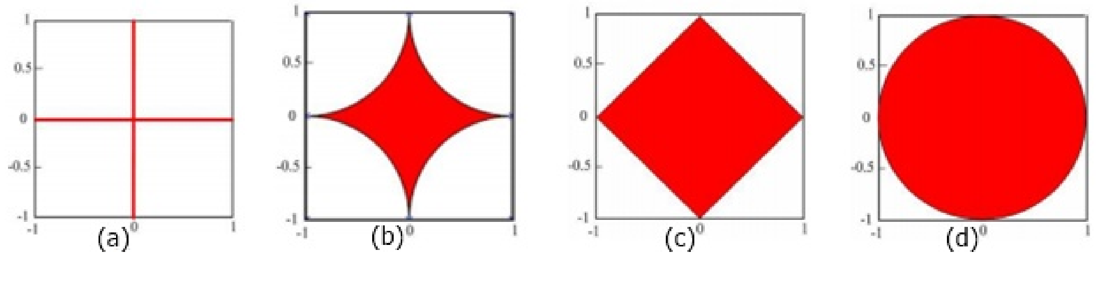


Figure 2.1: Geometric representation of ℓ_p in 2D for (a) $p = 0$ (b) $0 < p < 1$ (c) $p = 1$ (d) $p = 2$ [43]

In Figure 2.1 we can see that the unit ball of ℓ_p expands away from the axis as p goes to infinity, and gets closer to the axis as p gets closer to zero and lies on the axis when p equals zero. For ℓ_p with $p \leq 1$ (Figure 2.1 (a), (b), and (c)) we can see that the only time an objective line touches ℓ_p once is at the axis, and we know that when a point is on one axis the value for the other axis is zero. This shows that ℓ_p with $p \leq 1$ promotes sparsity. This will be explained further in chapter 3. This is one of the reasons, if not the only, why the ℓ_1-norm is chosen over the norms with $p > 1$ when we want to approximate the ℓ_0-minimization problem in sparse optimization. This will be explained further in Chapter 3 when we compare ℓ_0 and ℓ_1.

2.2 Properties of ℓ_p-norms

Consider an ℓ_p-norm of some vector x in 1D, and suppose $f_p(x)$ is the function of the ℓ_p-norm i.e.

$$f_p(x) = \|x\|_p^p = |x_i|^p.$$

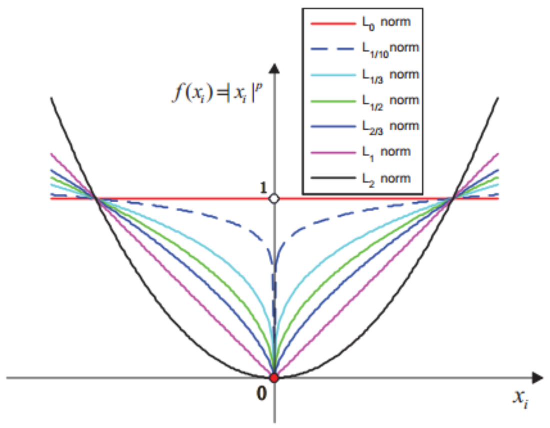


Figure 2.2: Geometric representation of ℓ_p 1D for (a) $p = 0$ (b) $0 < p < 1$ (c) $p = 1$ (d) $p = 2$ [43]

From Figure 2.2 we note the following.

1. The ℓ_0 function f_0

- is non-convex, since a line connecting two points, say $(-x, 1)$ and $(x, 1)$ for some positive x, will contain a point $(0, 1)$ where ℓ_0 is not defined at 0,
- has a discontinuity at 0
- is not a globally differentiable function since it is not differentiable at 0. The tangent line at the point $(0, 0)$ has an undefined gradient.

2. The ℓ_p $(0 < p < 1)$ function f_p
   - is nonconvex, since a line joining points $(1, 1)$ and $(0, 0)$ is below the function.
   - is not a globally differentiable function since it is not differentiable at 0, The tangent line at the point $(0, 0)$ has an undefined gradient.
3. The ℓ_1 function f_1
   - is convex, since a line connecting any two points in the domain of ℓ_1 will always be on ℓ_1
   - is continuous
   - is not a global differentiable function since it is not differentiable at 0.
4. The ℓ_2 function f_2
   - is convex since a line connecting any two points in the domain of ℓ_1 will always be on or above ℓ_1,
   - is continuous
   - is a globally differentiable function.

We discuss the geometrical representation of the norm. We have noticed that all norms with $p \geq 1$ are convex and thus satisfy the uniqueness property needed for the sparse optimization techniques. We note that the ℓ_1-norm is the first convex norm closest to the ℓ_0-norm. The ℓ_1 ball is close to the axes, therefore it has a higher chance of returning a sparse solution than norms with p larger than 1 since sparsity is obtained on the axes. This is one of the why reasons ℓ_1-norm is used to approximate ℓ_0-norm.

2.3 Restricted isometry property

If we choose to replace (relax) the ℓ_0-norm by the ℓ_1-norm, we we are not guaranteed a unique or best solution. The matrix of constraints needs to follow certain rules so that the solution to the ℓ_1-norm minimization problem can be the best and unique approximation to the solution of the ℓ_0-norm minimization problem. One of the properties is the restricted isometry property, and it is said that if the matrix of constraints follow the restricted isometry property then the solution to the ℓ_1-norm minimization problem is the best approximation to the solution of the ℓ_0-norm minimization problem [8]. Details of this theory will be discussed further in Chapter 3.

Definition 15. For each integers $k = 1, 2, 3....$ we define the k-restricted isometry constant δ_k of a matrix A as the smallest quantity such that

$$(1 - \delta_k) \|c\|_2^2 \leq \|Ac\|_2^2 \leq (1 + \delta_k) \|c\|_2^2$$

holds for all k-sparse vectors c [8]. This property essentially requires that every set of columns with the number of non-zero elements that are less than k approximately behaves like an orthonormal system [8]. So basically if x is k-sparse the k-restricted isometry constant δ_k of A [8] is the smallest constant such that

$$(1 - \delta_k) \|x\|_2^2 \leq \|Ax\|_2^2 \leq (1 + \delta_k) \|x\|_2^2.$$

2.4 Coherence of a matrix

In Chapter 3 we discuss greedy algorithms. Greedy algorithms solve an optimization problem by selecting rows in the matrix of constraints in order to reach an optimal solution. The method selects rows that are furthest to the residual of the previous iteration. This means we select the row that has the largest distance away from the residual in the previous iteration. We call this maximum distance the coherence.

Definition 16. Suppose we are given two orthonormal bases[1] A_1 and A_2 and $A = [A_1, A_2]$. The coherence [9] of a matrix A, is the maximum absolute value of the dot product of any two different columns a_i and a_j of the matrices A_1 and A_2, it is denoted by μ and it is defined mathematically as follows [9],

$$\mu = \max |\langle a_i, a_j \rangle| \text{ with } i \neq j. \tag{2.6}$$

In simple terms, the coherence measures the largest correlation between any two elements of A_1 and A_2 [9].

2.5 Sparse convex optimization in statistics

Regression problems with many explanatory variables occur in a wide variety of scientific fields. This problem can be dealt with by performing a statistical model selection to find an optimum solution that is simple and also provide a good predictive performance. In such problems penalized regression can be used. In penalized regression all the explanatory variables are kept in the model but the regression coefficients are

[1]A basis with unit vectors that are orthogonal to each other.

constrained by shrinking them towards zero. In regression terms, penalized regression estimates a regression model by minimizing the sum of squared errors subject to the penalty on the coefficients. The LASSO is one of the examples of penalized regression. The LASSO is formulated as follows,

$$\min \frac{1}{2} \|Ax - b\|_2^2 + \lambda \|x\|_1$$

where $\lambda > 0$ is called the Lagrange multiplier and it controls the level of shrinkage. The vector of coefficients will be sparse since some coefficients shrink towards zero. The expression $\|Ax - b\|_2^2 + \lambda \|x\|_1$ is convex since the ℓ_1-norm and the ℓ_1-norm are both convex. Methods for sparse convex optimization can be used to solve such problems.

2.6 Sparse convex optimization problem

For an algorithm to be a convex optimization technique it needs to solve the ℓ_1 minimization problem instead of the ℓ_0 minimization problem and both the constraints and the objective function should be convex [10]. An algorithm is a sparse optimization algorithm if it returns a sparse solution [1].

2.7 Conclusion

We have now discussed the theory needed for the purpose of sparse convex optimization techniques. In the next chapter we discuss the theory of sparse convex optimization as well as provide algorithms currently used to solve the optimization problem.

Chapter 3

Sparse convex optimization

This chapter discusses the concept of sparse convex optimization together with the algorithms. We first discuss the concept of the ℓ_0 minimization problem and explain why the ℓ_1 minimization problem is used to approximate the ℓ_0 minimization problem. We then discuss the algorithms for solving the ℓ_1 minimization and two algorithms for solving the ℓ_0 minimization so that in Section 4 we can compare the results for sparse convex optimization algorithms with sparse non-convex optimization algorithms. For an algorithm to be a convex optimization technique it needs to solve the ℓ_1 minimization problem instead of the ℓ_0 minimization problem and both the constraints and the objective function should be convex. An algorithm is a sparse optimization algorithm if it returns a sparse solution.

3.1 The ℓ_0 minimization problems

Suppose we have an optimization problem as follows

$$\min_{f_i(x) \leq b_i} f_0(x), \qquad i = 1, ..., m. \tag{3.1}$$

A convex optimization problem [10] is a mathematical optimization problem for which the objective and the constrained function are convex, that is, if $f_i(x)$ for $i = 0, ..., m$ satisfy equation

$$f_i(\alpha x + (1-\alpha)y) \leq \alpha f_i(x) + (1-\alpha) f_i(y).$$

A powerful property of convex optimization is that any local optimal solution is also a global solution as explained in Chapter 1 [4].
Consider a system of linear equations given by,

$$Ax = b, \tag{3.2}$$

where A is a $m \times n$ matrix with $m < n$ and x and b are $n \times 1$ and $m \times 1$ vectors respectively. This type of system of equations is called underdetermined since it has more unknowns than the equations ($m < n$). Using Gaussian elimination to solve this system of equations will result in infinitely many solutions. The most common way to solve this system is to use a least squares approach as follows,

$$\min \|b - Ax\|_2^2, \tag{3.3}$$

which results in x being obtained as,

$$x = \left(A^T A\right)^{-1} A^T b. \tag{3.4}$$

However, when the matrix A is underdetermined the inverse $\left(A^T A\right)^{-1}$ does not exist, hence the solution does not exist. If we have prior knowledge that x is sparse, then the problem in equation (3.2) is considered as a sparse optimization problem and in this case the ℓ_0 minimization problem can be used since the ℓ_0 minimization problem always results in a unique sparse solution (more in Section 3.2).

To incorporate sparsity we represent b as a linear combination of fewer columns of A [18] i.e. to obtain a sparse solution. The solution to this system of equations then yields the minimum number of coefficients to represent b. This procedure is known as sparse approximation. If a sparse vector x is found, it is called the sparse representation of b. In general sparse representations are not unique, but there are certain conditions that makes them unique [9], (see Section 3.2).

The ℓ_0 minimization problem is represented mathematically as follows,

$$\min_{Ax=y} \|x\|_0 \tag{3.5}$$

where $\|x\|_0$ is the number of non-zero elements in the vector x [38]. We however solve the system $Ax = y$ such that the zero norm of x is as small as possible. It can be further specified that we want to solve for x such that the sum of squares of the errors is small and at the same time x should be sparse, where the sum of squares of the errors is $\|y - Ax\|_2^2$. Therefore problem (3.5) can sometimes be written as follows,

$$\min_{\|x\|_0 \leq k} \|y - Ax\|_2^2 \tag{3.6}$$

where k is the level of sparsity of x [43]. This is referred to as sparse approximation. We want a vector x that has the minimum number of non-zero elements and at the same time satisfies equation (3.2). The ℓ_0 minimization problem is convex if both$\|x\|_0$

and $Ax = b$ are convex[1], that is if both $\|x\|_0$ and $Ax = y$ satisfies equation (1.3). $Ax = b$ satisfies equation (1.3) but $\|x\|_0$ does not, since $\|\alpha x\|_0 \neq \alpha\|x\|_0$ as discussed in Chapter 2. Therefore the ℓ_0 minimization problem is non-convex. It is known that non-convex optimization problems are computationally difficult to solve, one of the reasons being that the non-convex optimization problem may results in many local optima and it maybe difficult and time consuming to determine the global optimum from this local optima [32]. Equation (3.2) can also be written in Lagrange form i.e. in terms of minimizing both the sum of squared errors and the number of non-zero elements in x as follows,

$$\min \frac{1}{2}\|Ax - b\|_2^2 + \lambda\|x\|_0 \tag{3.7}$$

where $\lambda > 0$ is called the Lagrange multiplier [43]. This is called the unconstrained minimization problem [43]. We use equation (3.7) to develop some of the algorithms needed in this chapter i.e. iterative thresholding. There are many methods that can be used to solve the above problems, which instead of finding the k-columns to represent b at the same time, they find the columns one at a time i.e. iteratively. Solving non-convex functions is computationally hard for reasons explained in Section 1.3, for this reason we relax the ℓ_0-norm by a convex norm that is closest to it i.e. the ℓ_1-norm. This will be explained further in Section 3.4.6.2.

3.2 Uniqueness of the solution

Sparse solutions in general are not unique since different methods have different selection properties, but there are certain conditions that yield a unique solution of problem (3.2), namely

- Suppose that any $2k$ columns of the $m \times n$ matrix A are linearly independent $m \geq 2$. Then, any k-sparse vector x can be reconstructed uniquely. It follows that k-sparse representations are unique if and only if any $(2k)$-column submatrix of A are linearly independent [40].
- A sparse representation x of b is unique if $\|x\|_0 \leq \frac{\text{spark}(A)}{2}$ where spark(A) is the spark[2] of matrix A [14].
- Suppose A has a coherence μ (see Definition 16). If x is a sparse solution to problem (3.2) and $\|x\|_0 < \frac{1}{2}\left(\mu^{-1} + 1\right)$ then x is a unique solution to problem (3.2) [9]. The smaller the coherence the fewer columns needed from A to represent b [9].

[1]A system of linear equation is convex since all linear functions are convex

[2]Spark of A is the smallest number of columns in A that are linearly dependent

- A stronger condition is that, if the matrix A follows a restricted isometry property (has a small isometric constant δ_k) (see Definition 15) then the solution to problem (3.2) is unique. This means that every subset of k or fewer columns is approximately an orthonormal system [8]. We will loosely say that a matrix A obeys the RIP of order k if δ_k is not too close to one [9].

3.3 From ℓ_0 to ℓ_1

As stated solving the ℓ_0 minimization problem is computationally hard since the problem is not convex. This is because non-convex functions may have many local optima and it will take time to determine which of the local optima is a global optimum (this is explained in detail in Section 1.3). The norms ℓ_p for $0 < p < 1$ are also non-convex. The closest norm to ℓ_0 that is convex is the ℓ_1-norm. Therefore the ℓ_1-norm minimization problem is chosen since it is a good approximation to the ℓ_0 minimization problem [32]. The ℓ_1 ball is close to the axes, therefore it has a higher chance of returning a sparse solution than norms with p larger than 1 since sparsity is obtained on the axes. For this reason, we replace the ℓ_0-norm by the ℓ_1-norm. The ℓ_1-norm minimization problem is an approximation to the ℓ_0-norm minimization problem, which has been proven in [32]. The new problem is thus to solve for x such that the sum of the absolute value of the elements of x is as small as possible. Mathematically this is represented as,

$$\min_{Ax=y} \|x\|_1, \tag{3.8}$$

where $\|x\|_1$ is the ℓ_1-norm.

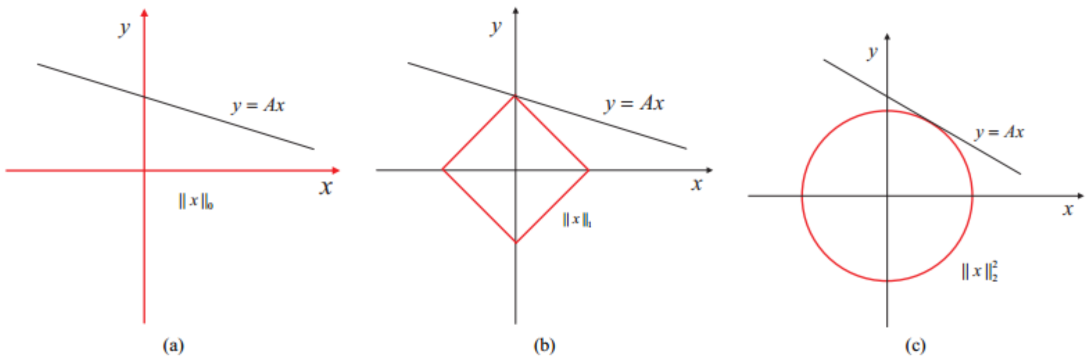


Figure 3.1: Visual representation of solutions of ℓ_p-norm in 2D for (a) $p = 0$ (b) $p = 1$ (c) $p = 2$ [43]

Consider the Figure 3.1. The black line represents the constraint function $Ax = b$ and the red ball represents the objective function $\|x\|_p$ i.e. the norm. The optimal solution x is obtained when the constraint function intercepts the objective function, and for x to be sparse they should intercept on one of the axes (for 2D only). Suppose we have a line $y = Ax$ (which is a hyper-plane in higher dimensions), and all possible solutions lie on it [43]. If we inflate the unit ball until it touches the line $y = Ax$ at some point, this point is the minimization to problem [43]. In Figure 3.1(a) we note that the objective function lies on the axes and we know that if we have the xy axes, any point lying on the axes will have $x = 0$ (the y-axes) or $y = 0$ (the x-axes). This means that the constraint function will always intercept the objective function on the one of the axes and thus lead to a sparse optimum solution. This gives a clear geometrical representation of why the ℓ_0-minimization problem always returns a sparse solution. In Figure 3.1(b) the objective function has the shape of a cube with the corners on the axes. This means that we will get a sparse solution when the constraint function intercepts the objective function on the corners i.e. on the axes. Due to the shape of the ℓ_1 ball the probability of the constraint function intercepting the objective function is high. This promotes sparsity because the only time the constrained function will intercept the objective function is at the corners. This means that the ℓ_1-minimization problem has a high chance of returning a sparse optimum solution. Figure 3.1(c) shows that for norms with $p > 1$ the probability of obtaining a sparse solution is low compared to norms with $p \leq 1$ since the constraint function can touch the objective function at anywhere besides on the axes. Even though they do not return sparse solutions, they still return an optimal solution.

3.4 Obtaining convexity

There are various classes of techniques for solving sparse convex optimization problems, most of which give good results, but are difficult to compute and also take time. Here we will focus on the class of techniques that are relatively easy to implement, give good results and are commonly used [31]. This class of techniques are convex relaxation, greedy algorithms, iterative thresholding and combinatorial algorithms. The following are the methods that we will discuss briefly.

1. **Convex relaxation -** Convex relaxation solves the ℓ_0-minimization problem by relaxing the use of ℓ_0 to the convex ℓ_1 instead (the basis pursuit or ℓ_1-norm minimization problem) and solve the convex program with algorithms that exploit the problem structure e.g. linear programming [11]. The most common linear programming algorithms that are used to solve the ℓ_1-norm minimization problem are the simplex method and the interior-point method.

2. **Greedy algorithms -** Greedy algorithms operate by the choice that seems to be the best at that moment and continue doing so until an optimal solution is found. Assume that you have an objective function that needs to be optimized (either

maximized or minimized) at a given point. At each step a greedy algorithm chooses a solution that seems the best at that moment and it continues doing so in each step until it finds the the optimal global solution of the function. This means that it makes a locally-optimal choice in the hope that this choice will lead to a globally-optimal solution. The general framework in greedy techniques is 1) to select an element and 2) to update the coefficients. The greedy strategy actually cannot directly solve the optimization problem and it only seeks an approximate solution [43].

3. **Iterative thresholding** - Given that x is sparse, iterative thresholding iteratively calculates x, and in each iteration it sets some elements of x to zero using what is called a threshold function. The iteration stops when there is convergence [31]. This will be explained in detail in the Section 3.7. The thresholding function depends upon the number of iterations and problem setup at hand [31].

4. **Combinatorial algorithms -** This class of algorithms recovers a sparse x through a two step process. It selects the necessary columns of the matrix of the constraints matrix (for sparsity purpose) then estimates the solution using selected columns. Note that the columns here are selected as a group instead of one at a time as in greedy algorithms. Even though they are extremely fast and efficient, as compared to convex relaxation or greedy algorithms, they require specific pattern in the measurements matrix, namely sparsity [31], which is why it is not commonly used. In this mini-dissertation we will not consider this algorithm.

We now look at each of these methods in detail.

3.5 Convex relaxation

Convex relaxation solves equation (3.2) by relaxing the use of ℓ_0 to the convex ℓ_1 instead and solves the convex program with algorithms that exploit the problem structure [11]. Even though the solution of the ℓ_0 minimization problem is unique, the solution of the ℓ_1 minimization problem is not since various methods selects the sparse solution differently. Since the solution to the ℓ_1 minimization problem approximates the solution to the ℓ_0 minimization problem, we are interested in the solution with the smallest norm. First we cover linear programming since to solve an ℓ_1-minimization problem we first need to convert it into a linear program.

3.5.1 Linear programming

3.5.1.1 Introduction

A linear program is a constrained optimization problem in which the objective function and each of the constraints are linear [11]. A linear program problem can be written in the following form. Let $c = (c_1, c_2, ..., c_n)^T$, $s = (s_1, s_2, ..., s_m)^T$ and $\alpha = (\alpha_1, \alpha_2, ..., \alpha_n)^T$ be real valued vectors and $\Phi = [\phi_{ij}]$ an $m \times n$ matrix. Then the problem

$$\min_{\alpha} \; c_1\alpha_1 + c_2\alpha_2 + c_3\alpha_3 + ... + c_n\alpha_n$$

subject to

$$\begin{gathered} \phi_{i1}\alpha_1 + \phi_{i2}\alpha_2 + ... + \phi_{in}\alpha_n \leq s_i \, , \; 1 \leq i \leq m_1 \\ \alpha_1 \geq 0, \; \alpha_2 \geq 0, ..., \alpha_n \geq 0 \\ \phi_{j1}\alpha_1 + \phi_{j2}\alpha_2 + ... + \phi_{jn}\alpha_n = s_j \, , \; m_1 < j \leq m_2 \end{gathered}$$

is called a linear program, where $m_1 + m_2 = m$ [33].

3.5.1.2 The standard form

The best way to work with a linear program is to put it in its standard form. The following is the standard form of a linear program.

$$\min_{\Phi\alpha = s} c^T\alpha, \quad \alpha \geq 0, \tag{3.9}$$

where $c^T\alpha$ is the objective function, $\Phi\alpha = s$ is a collection of equality constraints and $\alpha \geq 0$ is a set of bounds [33]. All variables are constrained to be non-negative. Then α is considered a feasible solution to problem (3.9) if $c^T\alpha$ is minimized and both conditions $\Phi\alpha = s$ and $\alpha \geq 0$ are satisfied. A collection of feasible solutions is called the feasible set.

3.5.1.3 Writing a linear program in standard form

Any linear program can be written in the standard form. It is best to know how to change a linear program to a standard form since this is the first step of the simplex method which is a primary method for solving linear programs. The following are the rules for changing any linear program to a standard one.

- **Change inequalities to equalities** –The inequality of the form

$$a_1x_1 + a_2x_2 + ... + a_nx_n \leq b \tag{3.10}$$

  can be changed to an equality by adding a slack variable u. A slack variable is the difference between the right hand side and the left hand side of the inequality.

so we have $a_1x_1 + a_2x_2 + ... + a_nx_n + u = b$. If we have an inequality of the form $a_1x_1 + a_2x_2 + ... + a_nx_n \geq b$, we can simply multiply the left hand side and the right hand side of this inequality by -1 to have it in the form of inequality (3.10), then we proceed as we did above [33].

- **Non-negative variables** – Some variables in the linear program are not constrained to be non-negative. In this case the unconstrained variable can be replaced by the difference of two non-negative variables,

$$x_j = x_j^+ - x_j^-$$

  where $x_j^+ \geq 0$ and $x_j^- \geq 0$, then from this we can adjust our objective and constrained functions [33].

- **Negative variables** – Some variables are constrained to be negative i.e. $x_j \leq 0$ in inequality (8). In this case we change $x_j \leq 0$ to $-x_j \geq 0$ and let $y_j = -x_j$ therefore $y_j \geq 0$. From here we substitute x_j by y_j in the objective and constrained functions.

3.5.2 Duality

Definition 17. The dual of a linear program is the transpose of that linear program with the constrained signs inverted [33].

Any given linear program has a related linear program called the dual. Duality is important since we can convert a maximization problem into a minimization problem and then use the simplex method to find the solution. The original linear program is called the primal. For every linear program of the form [33]

$$\min_{\Phi\alpha \geq s} c^T\alpha, \quad \alpha \geq 0$$

called the primal, and there exists a dual linear program of the form

$$\max_{y^T\Phi \leq c} y^T s.$$

Consider the standard form of a linear program in equation (3.9). To change Equation (3.9) into its dual form, we know that $\Phi\alpha = s$ is the same as $\Phi\alpha \leq s$ and $\Phi\alpha \geq s$. Therefore Equation (3.9) can be written as,

$$\min_{\substack{\Phi\alpha \geq s \\ -\Phi\alpha \geq -s}} \quad c^T\alpha \ \ \alpha \geq 0.$$

Using the above definition of a dual, we can write the dual of equation (3.9) as

$$\max_{\Phi^T(u-v)\le c} u^T s - v^T \underline{s} \;\; u \ge 0,\, v \ge 0$$

where u and v are the dual variables associated with the two constraints (see Table 3.1). Let $y = u - v$ then,

$$\max_{\Phi^T y \le c} y^T s. \tag{3.11}$$

We can convert program (3.11) to a standard form by adding a slack variable, say g, to the set of constraints, resulting in the following program

$$\max_{\Phi^T y + Ig = \alpha} y^T s,\;\; g \ge 0. \tag{3.12}$$

We should note that it is always best to transform a primal into its dual by first making sure that the primal is in its standard form. The following is a standard form of a minimization problem

$$\min_x \; z = c_1x_1 + c_2x_2 + c_3x_3 + \cdots + c_nx_n$$

subject to

$$\begin{aligned} \phi_{11}\alpha_1 + \phi_{12}\alpha_2 + \cdots + \phi_{1n}\alpha_n &\ge b_1 \\ \phi_{21}\alpha_1 + \phi_{22}\alpha_2 + \cdots + \phi_{2n}\alpha_n &\ge b_2 \\ &\vdots \\ \phi_{m1}\alpha_1 + \phi_{m2}\alpha_2 + \cdots + \phi_{mn}\alpha_n &\ge b_m, \end{aligned} \tag{3.13}$$

and for a maximization problem, this is the standard form

$$\min_x \; z = c_1x_1 + c_2x_2 + c_3x_3 + \cdots + c_nx_n$$

subject to

$$\begin{aligned} \phi_{11}\alpha_1 + \phi_{12}\alpha_2 + \cdots + \phi_{1n}\alpha_n &\le b_1 \\ \phi_{21}\alpha_1 + \phi_{22}\alpha_2 + \cdots + \phi_{2n}\alpha_n &\le b_2 \\ &\vdots \\ \phi_{m1}\alpha_1 + \phi_{m2}\alpha_2 + \cdots + \phi_{mn}\alpha_n &\le b_m. \end{aligned} \tag{3.14}$$

Note that for a minimization problem the constraint are either $\ge$ and for a maximization problem we either have $\le$. In Table 3.1, the first column represents the new variables to be considered and the first row represents the variables from our original problem. The last row and column represent the sign and the right hand side of the new and old constraints consecutively. The new variables have the same sign as the original variables. The generalization of the primal dual relationship is summarized in Table 3.1.

	α_1	α_1	$\cdots$	α_1	
y_1	ϕ_{11}	ϕ_{12}	$\cdots$	ϕ_{1n}	$\leq s_1$
y_1	ϕ_{21}	ϕ_{22}	$\cdots$	ϕ_{2n}	$\leq s_2$
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
y_1	ϕ_{m1}	ϕ_{m2}	$\cdots$	ϕ_{mn}	$\leq s_m$
	$\geq c_1$	$\geq c_1$	$\cdots$	$\geq c_1$	

Table 3.1: Primal-dual relationship

As an example consider the following linear program,

$$\min_{\alpha} \; 2\alpha_1 + 2\alpha_2$$

subject to

$$4\alpha_1 + 2\alpha_2 \geq 3$$
$$4\alpha + 3\alpha \geq 5$$
$$4\alpha_1 + 3\alpha_2 \geq 8$$

and we want to find the dual. First we can create a table similar to Table 3.1,

	α_1	α_2	
y_1	4	2	≥ 3
y_2	1	3	≥ 5
y_3	4	3	≥ 8
	≤ 2	≤ 2	

Therefore the dual form will be

$$\min_{y} \; 3y_1 + 5y_2 + 8y_3$$

subject to

$$4y_1 + y_2 + 4y_3 \leq 2$$
$$2y_1 + 3y_2 + 43y \leq 2.$$

In the next section methods for solving a linear program will be discussed. There are many methods for solving linear programs, such interior point method [33] and graphical method [13], but here we will only discuss the simplex since it the most commonly used method.

3.5.3 The simplex method

3.5.3.1 Maximization problem

The simplex method is an iterative procedure for solving a linear program. Most commonly, the simplex method is used to solve maximization problems in linear programming. In this section we will explain how to solve and use a maximization problem to solve a minimization problem. Consider the following linear programming problem

$$\max_{x}\ z = c_1x_1 + c_2x_2 + c_3x_3 + \cdots + c_nx_n$$

subject to

$$\begin{aligned}\phi_{11}\alpha_1 + \phi_{12}\alpha_2 + \cdots + \phi_{1n}\alpha_n &\le b_1\\ \phi_{21}\alpha_1 + \phi_{22}\alpha_2 + \cdots + \phi_{2n}\alpha_n &\le b_2\\ &\vdots\\ \phi_{m1}\alpha_1 + \phi_{m2}\alpha_2 + \cdots + \phi_{mn}\alpha_n &\le b_m\end{aligned} \tag{3.15}$$

$\alpha_i \ge 0\ \forall i$'s. The first step in using the simplex method is to change the inequality constrained to equality constraint by adding a slack, a surplus or an artificial variable. Table 3.2 shows the rules for converting inequality constraints to equality constraints.

Inequality ($\le$)	Add a slack variable (s_1)
Inequality ($\ge$)	Add a slack variable (s_2) and an artificial variable (M_1)
Equality ($=$)	Add an artificial variable (M_2)

Table 3.2: Methods for changing inequality constraints to equality constraints

A slack variable represents unused resources in the inequality and a surplus variable is the opposite/negative of the slack variable. Since both the slack and the surplus variables do not contribute to the objective function we do not include them in the objective function. An artificial variable does not have a true meaning, it is simply used to set up a basic feasible solution when we start the simplex method. An artificial variable should not appear in the final solution of the problem. To make sure that the artificial variable does not appear in the final solution of the problem we assign it a very large coefficient when we include it in the objective function and instead of specifying a large number for this coefficient we just use M to represent a large coefficient. The next step is to write the objective function in the form $z - cx_1 - c_2x_2 - c_3x_3 - \cdots - c_nx_n = 0$. The linear program in (3.15) will be as follows.

$$\min_{x}\ z - cx_1 - c_2x_2 - c_3x_3 - \cdots - c_nx_n + 0s_1 + 0s_2 + ... + 0s_m = 0$$

subject to

$$\begin{aligned}\phi_{11}\alpha_1 + \phi_{12}\alpha_2 + \cdots + \phi_{1n}\alpha_n + s_1 &= b_1\\ \phi_{21}\alpha_1 + \phi_{22}\alpha_2 + \cdots + \phi_{2n}\alpha_n + s_2 &= b_2\\ &\vdots\\ \phi_{m1}\alpha_1 + \phi_{m2}\alpha_2 + \cdots + \phi_{mn}\alpha_{n+}s_m &= b\end{aligned}$$

Variables that are non-zero are called basic variables and those that are zero are called non-basic variables. Non-basic variables correspond to the zero solution to the system of equations. A solution to a linear program problem will consist only of the variables x_i for $i = 1, 2 \ldots, n$ and s_j for $j = 1, 2, \ldots, m$ where at most m variables are not zero (non-basic variables), i.e. the solution will be $(\alpha_1, \alpha_2, \ldots, \alpha_n, s_1, s_2, \ldots, s_m)$. This solution is called a basic solution of a linear program [33]. If all the variables in the basic solution are non-negative the solution is called a basic feasible solution. Gaussian elimination is applied to the so called simplex table to solve the linear program. For our linear program the simplex table will look like that in Table 3.3.

	α_1	α_2	$\cdots$	α_n	s_1	s_2	$\cdots$	s_m	b
s_1	ϕ_{11}	ϕ_{12}	$\cdots$	ϕ_{1n}	1	0	$\cdots$	0	b_2
s_2	ϕ_{21}	ϕ_{22}	$\cdots$	ϕ_{2n}	0	1	$\cdots$	0	b_1
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
s_m	ϕ_{m1}	ϕ_{m2}	$\cdots$	ϕ_{mn}	0	0	$\cdots$	1	b_1
z	$-c_1$	$-c_2$	$\cdots$	$-c_n$	0	0	$\cdots$	0	0

Table 3.3: The simplex table

All variables corresponding to a column with zeros and a one are basic variables and their values corresponds to the b elements that are in the same row as the element one in that column, and the remaining columns are non-basic variable and the have a value of zero initially. We start the simplex method at the origin, i.e. at $\alpha_i = 0\ \forall i$. Therefore $s_j = b_j\ \forall j$. The initial solution for the initial table will be

$$(\alpha_1\, \alpha_2 \cdots \alpha_n\, s_1\, s_2\, s_m) = (0\, 0 \cdots 0\, b_1\, b_2 \cdots b_m)\,.$$

The value of the objective function corresponds to the value at the interception of the last row and the last column, which is currently zero since we start at the origin. The last row represents the negative of the coefficients of the α_i in the objective function and the solution is optimal if their values are zero or positive. In our case the solution is not optimal since we have negative elements in the last row. To improve the solution we choose a variable from the non-basic variables to enter the basic variables; this variable is called the entering variable. Since we will have too many variables for the basic solution, one variable from the basic variables set must be taken to the non-basic set, and this variable is called the departing variable. The entering variable is the variable corresponding to the largest negative value in the last row and the column corresponding to that variable is called the pivot column. The departing variable corresponds to the variables with the smallest ratio $\frac{b_i}{\phi_{ij}}$ where ϕ_{ij} are in the pivot column. The value in both the pivot row and the pivot column is called the pivot quantity. The entering variable is the pivot row divided by the pivot quantity and the remaining rows are changed using Gaussian elimination in such a way that all values above and below the pivot quantity are zero. This process is repeated until we do not have any negative value in the last row. As an example consider the following linear program and we use the simplex method to solve it.

$$\max_{y} z = 3y_1 + y_2$$

subject to

$$\begin{aligned} y_1 + 2y_2 &\leq 4, \\ y_1 + y_2 &\leq 3, \\ y_1, y_2 &\geq 0. \end{aligned}$$

Then the simplex table will be as follows

	y_1	y_2	s_1	s_1	b	ratio
s_1	1	2	1	0	4	$\frac{4}{1} = 4$
s_2	1	1	0	1	3	$\frac{3}{1} = 3$
z	-3	-1	0	0	0	

The initial solution to the simplex table is $\begin{pmatrix} 0 & 0 & 4 & 3 \end{pmatrix}$. It is clear that -3 is the largest negative value, therefore the y_1 is the entering variable and the corresponding column is the pivot table. We divide the elements of the b column with the corresponding values of the pivot column. The smallest non-negative ratio is $\frac{3}{1} = 3$ and it is with the s_2 row, therefore s_2 is the departing variable and its row is the pivot row. The pivot quantity is 1 since it is in both the pivot row, s_2 and the pivot column, y_1. The entering variable will then be row s_2 divided by the pivot quantity, we then label the row y_1. Then we perform row operations on both row one and row three in such a way that the element below and above the pivot quantity are zero. With the first row we perform the operation $R_1 - R_2 \to R_1$ and with the third row we perform the operation $3R_2 + R_3 \to R_3$. The simplex table now looks as follows.

	y_1	y_2	s_1	s_2	b
s_1	0	1	1	-1	1
y_1	1	1	0	1	3
z	0	2	0	3	9

Since all the elements of the last row are positive the solution to this table is optimal. Variables y_1 and s_1 are basic variables since columns corresponding to them contains zeros and a one and their values corresponds to the b elements that are in the same row as the one in that column. In this case $y_1 = 3$ and $s_1 = 1$. Therefore variables y_2 and s_2 are non-basic and their values are zero. The solution then becomes $\begin{pmatrix} 3 & 0 & 1 & 0 \end{pmatrix}$, but since our original problem only deals with y_1 and y_2, our solution will be $y_1 = 3$ and $y_2 = 0$ and $z = 9$. The value for z can be verified by substituting the values of

y_1 and y_2 in the objective function i.e. $z = 3(3) + (0) = 9$. This means that z has a maximum of 9 and this happens when it is at a point $y_1 = 3$ and $y_2 = 0$. We summarize the algorithm here.

Algorithm 3.1 The simplex method

- **Step 1** Change the constraints into equalities.
- **Step 2** Set up the simplex table.
- **Step 3** Find the pivot column - correspond to the column with the minimum negative value in the last row.
- **Step 4** Find the pivot row - correspond to the row with the smallest ratio $\frac{b_i}{\phi_{ij}}$ where ϕ_{ij} are in the pivot column.
- **Step 5** Set up the new simplex matrix.
- **Step 6** If the last row has negative values return to step 3.

3.5.3.2 Minimization problem

We will solve the minimization problem using the primal-dual relationship called the Von Neumann Duality Principle [26]. It states that, the objective value w of a minimization problem in standard form has a minimum value if and only if the objective value z of the dual maximization problem has a maximum value. Moreover, the minimum value of w is equal to the maximum value of z. This means that to find the solution for a minimization problem you need to find the solution to the maximization problem of the dual. The first step is to convert the minimization problem into its dual, then apply the simplex method. The only difference is reading the solution on the simplex table. Here the solution for the minimization problem are the last values in the slack variables column. Here is an example,

$$\min_x \; z = 4x_1 + 3x_2$$

subject to

$$\begin{aligned} x_1 + x_2 &\geq 3 \\ 2x_1 + x_2 &\geq 1 \\ x_1, x_2 &\geq 0. \end{aligned}$$

We find the dual as following table.

	x_1	x_2	
y_1	1	1	≥ 3
y_2	2	1	≥ 1
	≤ 4	≤ 3	

Therefore the dual will be,

$$\max_{y} \; w = 3y_1 + y_2$$

subject to

$$\begin{aligned} y_1 + 2y_2 &\leq 4 \\ y_1 + y_2 &\leq 3 \\ y_1, y_2 &\geq 0. \end{aligned}$$

Here y_1 and y_2 take the sign of x_1 and x_2 therefore the constraints $x_1, x_2 \geq 0$ is not represented in the table. This is exactly the same as the example in Section 3.5.4, therefore we jump to the final simplex table.

	y_1	y_2	s_1	s_1	b
s_1	0	1	1	-1	1
s_2	1	1	0	1	3
w	0	2	**0**	**3**	9

the solution to the maximization problem is $y_1 = 3$ and $y_2 = 1$. The solution to the minimization problem are the last values in the slack variable columns i.e. $x_1 = 0$ and $x_2 = 3$, but the value of the objective function is still 9 just like the primal-dual relationship suggested. When we verify the value for the objective function we substitute $x_1 = 0$ and $x_2 = 3$ in the objective function of the primal i.e. $z = 4(0) + 3(3) = 9$.

3.5.4 The ℓ_1-norm minimization problem (Basis pursuit)

3.5.4.1 Theory

Basis pursuit is a principle for decomposing a signal into an "optimal" superposition of dictionary elements, where optimal means having the smallest ℓ_1 norm of coefficients among all such decompositions [11]. The ℓ_1 minimization problem is represented mathematically as in equation (3.8). Since both the functions $\|x\|_1$ and $Ax = y$ satisfies equation (1.3), the ℓ_1-norm minimization problem is convex and hence has a unique solution. Due to the shape of the ℓ_1 ball (see Figure 3.1), the ℓ_1-minimization problem promotes sparsity [16]. To use the method of basis pursuit, we need to transform equation (3.8) to a linear program and use methods for solving linear programs to find the optimal solution. We formulate the method of basis pursuit as a linear program as follows. Let $\Phi = (A, -A)$ and $\alpha = \begin{bmatrix} u \\ v \end{bmatrix}$ such that $x = u - v$ where u, v both have dimensions $n \times 1$, [11]. Therefore Φ will have dimension $m \times 2n$ and α will have dimension $2n \times 1$. Let $c = \begin{pmatrix} 1 & 1 \end{pmatrix}$ and $s = y$, where 1 is a vector of ones with dimension $n \times 1$. Then the linear program can be formulated as follows,

$$\min_{\Phi\alpha = s} c^T\alpha \quad \alpha \geq 0. \tag{3.16}$$

We therefore solve for α by solving equation (3.16). Then the solution to the ℓ_1-minimization problem can be solved using the relationship $x = u - v$. Equation (3.8) is equivalent to the following

$$\min \left(1^T u + 1^T v\right) \text{ such that } (Au - Av) = s. \tag{3.17}$$

Equation $(Au - Av) = s$ is linear since it is a system of linear equations. The expression $1^T u + 1^T v$ is also linear since it is a linear combination of linear functions. Therefore equation (3.8) is linear and hence convex since all linear functions are convex.

3.5.4.2 Algorithm

The algorithm for the ℓ_1-minimization problem is just the same as the one in Section 3.5.5.

3.6 Greedy algorithms

3.6.1 Introduction

The idea in greedy algorithms for finding sparse solution is also to solve the ℓ_0-minimization problem, but as explained before the ℓ_0-minimization problem is non-convex and therefore NP-hard. The greedy algorithms alleviate this problem by approximating the solution [43]. A greedy algorithm solves the ℓ_0 minimization directly, therefore, it is not a convex approach, however it gives sparse solution since it selects necessary columns in the matrix of constraints, and use those columns to approximate the solution [21]. Greedy pursuit methods are defined to be those methods that approximate the sparse solution x iteratively. They consist of three basic steps. Firstly, the x is set to a zero vector. Secondly, these methods estimate a set of non-zero components of x by iteratively adding new components that are deemed to be non-zero. Thirdly, the values for all non-zero components are optimized. The greedy strategy provides a special way to obtain an approximate sparse representation solution [43]. The greedy strategy actually cannot directly solve the optimization problem so it only seeks an approximate solution for equation (3.5) [43]. In each iteration the greedy algorithms find the global optimum solution which will lead to achieving the optimal holistic solution and if we want a $k-$sparse solution the greedy algorithm will stop at the k-th column leading to a k-sparse solution [43]. We define greedy algorithms so that we can compare the results of sparse convex optimization techniques to the results of sparse non-convex optimization techniques. We discuss two such convex algorithms next.

3.6.2 Matching pursuit

3.6.2.1 Theory

The main idea of the matching pursuit (MP) is to interactively choose the best column from the the matrix A based on a certain similarity measurement to approximately obtain the sparse solution [43]. Suppose $y = Ax$ as explained in Section 2.1, with $A = (a_1, a_2, ..., a_n)$ and $\|a_i\|_2 = 1$ for $i = 1, 2, ..., n$. let R_0 be the initial residual (this initialized at the beginning of the iterations). First, matching pursuit chooses the best column a_j that satisfies the following property called the mutual incoherence [6], namely

$$|\langle R_0, a_j\rangle| = \sup|\langle R_0, a_i\rangle| \tag{3.18}$$

where a_j is the chosen column from the matrix A and j represent the iterations. Matching pursuit correlates the residual R, with all the columns of A and then chooses the column that contribute the most in terms of correlation. Therefore y can be decomposed as follows:

$$\begin{aligned} y &= \langle y, a_j\rangle a_j + R_1 \\ &= \langle R_0, a_j\rangle a_j + R_1, \end{aligned} \tag{3.19}$$

so that the next residual can be calculated using the formula $R_1 = R_0 - \langle R_0, a_j\rangle a_j$. By introducing the square of the ℓ_2-norm on both side of equation (3.19), we can write it in the following way

$$\|y\|_2^2 = |\langle y, a_j\rangle|^2 + \|R_1\|_2^2. \tag{3.20}$$

Here R_1 is the representation residual by using a_j to represent R_1, and $\langle R_0, a_j\rangle a_j$ is the orthogonal projection of y onto a_j, considering the fact that a_j is orthogonal to R_1 [43]. Basically matching pursuit finds the best column in A that satisfies the mutual incohorence and then uses the corresponding residual as the next approximation target and it continues until the stopping criteria has been met. At the t^{th} iteration we have the following,

$$\begin{aligned} |\langle R_t, a_t\rangle| &= \sup|\langle R_t, a_i\rangle| \\ R_t &= \langle R_t, a_t\rangle a_t + R_{t+1} \\ \|R_t\|_2^2 &= |\langle R_t, a_t\rangle|^2 + \|R_{t+1}\|_2^2. \end{aligned} \tag{3.21}$$

At the k^{th} iteration (k-sparse) $\|R_k\|_2 \leq c$ where c is a some small constant. In the second and next iterations the dot product of the residuals with the columns that have already been chosen may not be computed as they are orthogonal to that residual and the dot product will be zero. Therefore y can be represented as follows,

$$y = \sum_{l=0}^{k-1} \langle R_{j,} a_j \rangle \, a_j + R_k \tag{3.22}$$

where $k < n$. Equation (3.22) is the representation of y with only a few columns of A. The algorithm is summarized here.

Algorithm 3.2 Matching pursuit algorithm

```
• Step 1 Input matrix $A$ and vector $y$
• Step 2 Initialize $D_0 = \emptyset, \;\; R_0 = y$
• Step 3 Compute $\langle R_0, a_i \rangle = R_0^T a_i$
• Step 4 Choose $a_{S_k}$ such that $|\langle R_0, a_{S_k} \rangle| = \max |\langle R_0, a_i \rangle|$ and update $D_0$ as $D_1 = D_0 \cup a_{S_k}$
• Step 5 Calculate the estimate $\widetilde{x}$ of $x$ by minimizing $\|y - D_1 \widetilde{x}\|_2^2$ i.e $\widetilde{x} = (D_1^T D_1)^{-1} D_1^T y$ then $R_1 = R_0 - \langle R_0, a_{S_t} \rangle \, a_{S_t}$.
• Step 6 Repeat from step 3 until $\|R_k\| \leq c$ with $c$ a small constant and $r_k$ the residual at the $k$-th iteration
• Step 7 Output $D_k$ and $\widetilde{x}$
```

3.6.2.2 Complications

In matching pursuit after subtracting the projection from the residual may result in a subsequent residual which is not orthogonal to the previously chosen column(s) i.e. $R_1 = R_0 - \langle R_0, a_j \rangle \, a_j$ may not be orthogonal to a_j. This means that computing R_1 introduces some component that is not orthogonal to the span of the first column chosen. This causes the solution to take time to converge. This happens because the columns in A are not mutually orthogonal. When there is no orthogonality between R_1 and a_{S_k} the iteration might choose the same column more than once. When there is orthogonality, the dot product between R_1 and a_{S_k} is zero, so the probability of choosing the same column more than once is small since we are choosing a column that has a maximum dot product with the residual. Consider the following example.

$$A = \begin{pmatrix} 1 & 0.5 & -1.4 \\ 0 & 0.8 & -1.4 \end{pmatrix} \quad y = \begin{pmatrix} 1 \\ 1.4 \end{pmatrix}.$$

We initialize the residual matrix R_0 to y. The first step is to calculate the absolute value of the dot product between the columns of A and R_0. We get $|\langle R_0, A\rangle| = \left|R_0^T A\right| = \begin{pmatrix} 1 & 1.62 & 3.36 \end{pmatrix}$. Here 3.36 is the maximum of $|\langle R_0, A\rangle|$, therefore the third vector $\begin{pmatrix} -1.4 \\ -1.4 \end{pmatrix}$ is the first chosen vector. The next residual becomes

$$R_1 = R_0 - \left\langle R_0, \begin{pmatrix} -1.4 \\ -1.4 \end{pmatrix} \right\rangle = \begin{pmatrix} -3.704 \\ -3.304 \end{pmatrix}.$$

Since two vectors are orthogonal if their dot product is zero. Then $\left\langle R_1, \begin{pmatrix} -1.4 \\ -1.4 \end{pmatrix} \right\rangle = 9.8112$ which is not zero, so R_1 is not orthogonal to $a_3 = \begin{pmatrix} -1.4 \\ -1.4 \end{pmatrix}$. This shows that the matching pursuit computes residuals that are not orthogonal to the span of previously chosen columns. This complication is dealt with by the use of orthogonal matching pursuit.

3.6.3 Orthogonal matching pursuit

3.6.3.1 Theory

Matching pursuit was introduced in this mini-dissertation so that it will be easier to understand the concept of orthogonal matching (OMP) pursuit and where it comes from. Orthogonal matching pursuit is an improvement of matching pursuit. The orthogonal matching pursuit employs the process of orthogonalization to guarantee the orthogonal direction of projection in each iteration [43]. Orthogonal matching pursuit is an iterative greedy algorithm that selects at each step the column, which is most correlated with the current residuals [6]. Different from matching pursuit, orthogonal matching pursuit never reselects a column and the residual at any iteration is always orthogonal to all currently selected columns in the dictionary. Another difference is that OMP minimizes the coefficients for all selected columns at iteration k, while MP only updates the coefficient of the most recently selected column [21]. Orthogonal matching pursuit is similar to matching pursuit, the only difference is the calculations of residual since orthogonal matching pursuit makes sure that the residual is orthogonal to the span of all the chosen vectors by projecting the residual to the vector space that is orthogonal to the vector space spanned by all the chosen columns. At the k^{th} iteration orthogonal matching pursuit selects the column a_{S_k} as part of the representation of y from the matrix A that satisfies,

$$|\langle R_k, a_{S_k}\rangle| = \max |\langle R_k, a_i\rangle| , \tag{3.23}$$

where S_k is the index set tracking the indices of all selected columns up to iteration k, and the residual is calculated as follows

$$R_k = (I - P_{S_k})R_{k-1} \tag{3.24}$$

where P_{S_k} (called the projection matrix) is the orthogonal projection onto the vector space spanned by all the columns that are already chosen. If D_k is a matrix made up of previously chosen columns at iteration k then

$$P_{S_k} = D_k(D_k^T D_k)^{-1} D_k^T. \tag{3.25}$$

One can always find the estimated coefficient, $\widetilde{x}$ at each iteration by solving the following least squares problem,

$$\min \|y - D_k\widetilde{x}\|_2^2 \tag{3.26}$$

and then use that estimate to calculate the residual. Then

$$\widetilde{x} = (D_k^T D_k)^{-1} D_k^T y \tag{3.27}$$

and the residual can be calculated as

$$R_i = y - D_k\widetilde{x}. \tag{3.28}$$

The process continues until the stopping criteria has been met. We should note that when calculating the residual using equation (3.24) and equation (3.28) we will not receive the equivalent solutions but the results using both methods are similar. They both select the same columns that are going to be used to represents y, but the estimate for the vector x is different. Using the projection matrix to calculate the residuals results in a smaller error than using the least squares method. The algorithm is summarized here.

Algorithm 3.3 Orthogonal matching pursuit algorithm

```
• Step 1 Input matrix $A$ and vector $y$
• Step 2 Initialize $D_0 = \emptyset$ $r_0 = y$ $R_0 = r_0$
• Step 3 Compute $\langle r_0, a_i \rangle = r_0^T a_i$
• Step 4 Choose $a_{S_k}$ such that $|\langle r_0, a_{S_k} \rangle| = \max |\langle r_0, a_i \rangle|$ and update $D_0$ as $D_1 = D_0 \cup a_{S_k}$
• Step 5 Calculate the estimate $\widetilde{x}$ of $x$ by minimizing $\|y - D_t\widetilde{x}\|_2^2$ i.e $\widetilde{x} = (D_1^T D_1)^{-1} D_1^T y$ then $r_1 = y - D_1\widetilde{x}$ and update $R_0$ as $R_1 = R_0 \cup r_1$
• Step 6 Repeat from step 3 until $\|r_k\| \leq c$ with $c$ a small constant and $r_k$ the residual at the $k$-th iteration
• Step 7 Output $D_k$ and $\widetilde{x}$
```

Consider the example in Section 4.3.2.3. We are now solving the problem using the orthogonal matching pursuit to stress the fact that the residual is orthogonal to the vector space spanned by the chosen columns.

$$A = \begin{pmatrix} 1 & 0.5 & -1.4 \\ 0 & 0.8 & -1.4 \end{pmatrix} \quad y = \begin{pmatrix} 1 \\ 1.4 \end{pmatrix}$$

We initialize the residual matrix R_0 to y. The first step is to calculate the absolute value of the dot product between the columns of A and R_0. We get $|\langle R_0, A \rangle| = \left| R_0^T A \right| = \begin{pmatrix} 1 & 1.62 & 3.36 \end{pmatrix}$. Here again 3.36 is the maximum element of $|\langle R_0, A \rangle|$, therefore the third vector $\begin{pmatrix} -1.4 \\ -1.4 \end{pmatrix}$ is the first chosen vector. This time the residual is calculated as

$$r_1 = (I - P_{S_k}) r_0$$

where

$$P_{S_k} = D_1 (D_1^T D_1)^{-1} D_1^T$$

and $D_1 = \begin{pmatrix} -1.4 \\ -1.4 \end{pmatrix}$ therefore $r_1 = \begin{pmatrix} -0.2 \\ 0.2 \end{pmatrix}$. To see that r_1 to a_{S_1} are orthogonal we prove that their dot product equals zero i.e. $r_1^T a_{S_1} = \begin{pmatrix} -0.2 & 0.2 \end{pmatrix} \begin{pmatrix} -1.4 \\ -1.4 \end{pmatrix} = 0$. This shows that in orthogonal matching pursuit the residual is always orthogonal to the vector space spanned by the columns that are already chosen.

3.6.3.2 The maximum of k (for k-sparse)

We aim to write y with only a few columns of the matrix A. That is we want x to be k-sparse with $k < n$. A has dimension $m \times n$ with $m < n$. Greedy algorithms solve the ℓ_1-norm minimization problem by choosing one column in each iteration. In each iteration, after the column is chosen it is stored in a matrix D, the estimate of x is computed by minimizing the sum of squares of the residual i.e.

$$\hat{x} = (D^T D)^{-1} D^T y. \tag{3.29}$$

Equation (3.29) has a solution if and only if $D^T D$ is invertible (non-singular). In this case for $D^T D$ to be invertible D must not be under-determined (if rank$(D) = m \leq n$). Since we are choosing one column at a time, the matrix D is updated by one column at a time and it will not be under-determined until m columns are added. After adding the m^{th} column the matrix D will have a dimension $m \times m$ and adding more column makes D under-determined and causing equation (3.29) to have no solution. This means that the programs will run for m iterations or less depending on the stopping criterion. Therefore the maximum sparsity x can get is m i.e. $k \leq m$. Consider the graph in Figure 3.2.

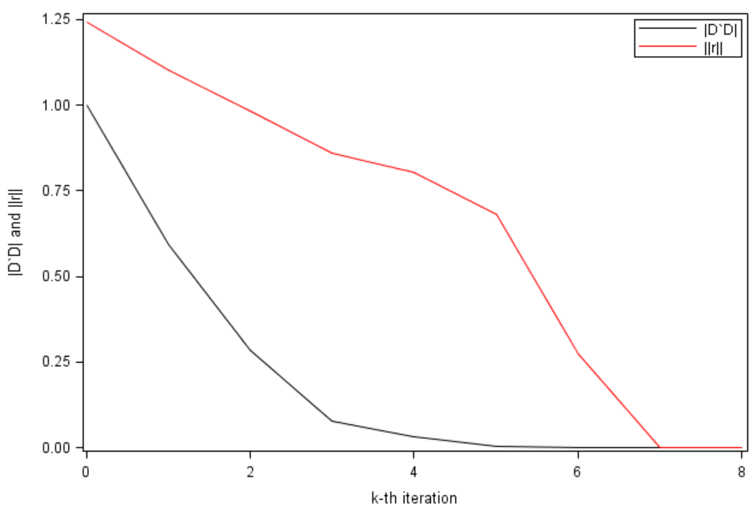


Figure 3.2: The plot of k vs the determinant and the sum of squared errors of the residual with $m = 8$ and $n = 10$

In Figure 3.2, k is the level of sparsity and the number of iterations, the blue line is the determinant of a $D^T D$ matrix for each iteration and the red line is the ℓ_2-nom (sum of squared error) of the residual at each iteration. The graph shows that as the number of iterations increases the determinant gets closer to zero and eventually becomes zero at $k = m$. It means that the more we add columns to the matrix D the more we decrease its determinant. This shows that k can go to a maximum of m iterations i.e. k has an upper bound of m. As k increases the norm of the residual decreases. We are using the norm of a residual as our stopping criteria i.e. we stop the iterations if the norm of the residual in that iteration is less than some small constant c. And on the graph the norm of the residual is small at iteration m, so the iteration stops at $k = 8$ which can also be seen on the plot of the determinant. This applies to both matching pursuit and orthogonal matching pursuit.

3.7 Iterative thresholding

3.7.1 Introduction

Iterative thresholding also approximates the ℓ_0 minimization problem by solving the ℓ_1 minimization problem. Iterative thresholding is a convex approach since it solve a convex minimization problem. For the solution to be sparse, iterative thresholding uses what is called the threshold operator to select which elements of the solution are going to be replaced by zero. The threshold operator we are going discuss are hard threshold and soft threshold.

3.7.2 Iterative hard thresholding

Iterative hard thresholding is used to solve equation (3.8) recursively. This method is used for solving the under-determined system of equations of the form $Ax = b$. To solve the system of equations we know that $A^TAx = A^Tb$ which leads to a fixed point equation $x = (I - A^TA)x + A^Tb$. Classical iterative methods suggests to define a sequence (x_i) by the recursion $x_{i+1} = (I - A^TA)x_i + A^Tb$ [17]. Since we want a k-sparse solution, in each iteration we will only select the k largest elements of $(I - A^TA)x + A^Tb$. This leads to the following formula [3],

$$x_{i+1} = H_k\left[Ix_i + A^T(b - Ax_i)\right] \tag{3.30}$$

where

$$H_k(\underline{z}) = \begin{cases} 0 & |z_i| < \lambda \\ z_i & |z_i| > \lambda \end{cases}.$$

Note that

$$\begin{aligned}(I - A^TA)x_i + A^Tb &= Ix_i - A^TAx_i + A^Tb \\ &= Ix_i + A^T(b - Ax_i).\end{aligned}$$

Here $H_k(z)$ is called the hard threshold operator which is a non-linear operator that sets all but the largest (in absolute value) k elements of z to zero. This is demonstrated in Figure 3.3.

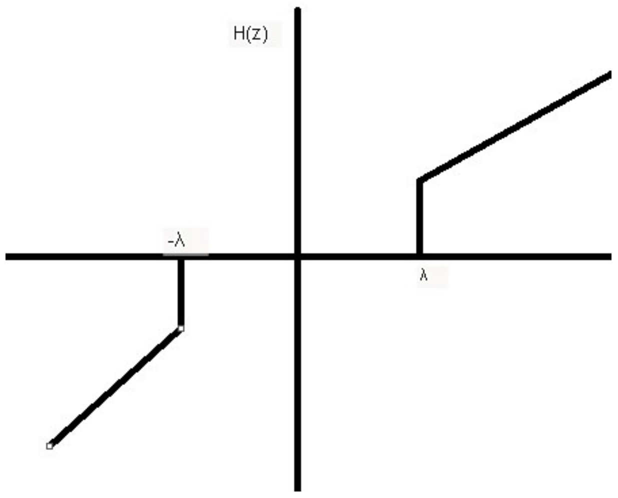


Figure 3.3: The plot of the hard threshold operator $H_k(z)$

From the graph in Figure 3.3 we can see that the hard threshold operator $H_k(z)$ is a non-linear function. Iterative hard thresholding guarantees convergence to a local minimum if $\|A\|_2 < 1$ [3]. The algorithm is summarized here,

Algorithm 3.4 Iterative hard thresholding, $H_k(z)$

- **Step 1** Input matrix A and vector b
- **Step 2** Initialize $x_0 = 0$
- **Step 3** Compute $x_{i+1} = H_k\left[Ix_i + A^T(b - Ax_i)\right]$
- **Step 4** Repeat step 3 until $|x_i - x_{i+1}|$ is close to zero.
- **Step 5** Output x

3.7.3 Iterative soft thresholding

Definition 18. A soft threshold function is a non-linear function of the form

$$\mathrm{soft}(x,T) = \begin{cases} x+T & x \le -T \\ 0 & |x| \le T \\ x-T & x \ge T \end{cases} \tag{3.31}$$

where T is the threshold parameter. The function can also be written in the form

$$\mathrm{soft}(x,T) = \mathrm{sign}(x)\max(0,|x|-T).$$

Figure 3.4 shows the soft threshold function.

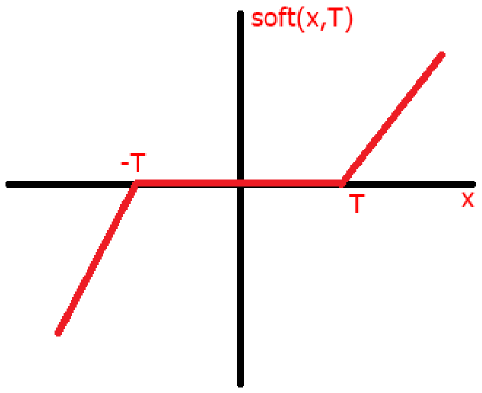


Figure 3.4: The plot of the soft threshold operator

This function sets values small values (less than T in absolute value) to zero and shrinks large values (greater than T in absolute value) towards zero, and this is why it is sometimes called the shrinkage operator [34]. We next discuss what is called the majorization minimization which we are going to use to derive the soft threshold function for solving the ℓ_1 minimization problem.

Majorization minimization

Definition 19. Suppose we have two functions $f : w \to w$ and $g : w \to w$. The function f majorizes g if $f(x) \geq g(x)$ for all $x \in w$ [12].

If we have a function $J(x)$ that we want to minimize, we can choose a function $G(x)$ that majorizes $J(x)$. That is $G(x) \geq J(x)$, where the inequality is true at some initialized point x_k called the minimizer. Then we minimize $G(x)$ at x_k to find x_{k+1}, i.e. we find x_{k+1} such that $G(x_{k+1}) \leq G(x_k)$ [35]. In the second iteration we set x_{k+1} as our minimizer and find x_{k+2} such that $G(x_{k+2}) \leq G(x_{k+1})$. This continues until there is convergence. Since $G(x)$ changes in each iteration we can denote it as $G_k(x)$ and we can represent it as follows [35],

$$G_k(x) = J(x) + \text{non-negative function of } x \tag{3.32}$$

Suppose we want to minimize the following,

$$O(x) = \|y - Ax\|_2^2 + \lambda\|x\|_1 \tag{3.33}$$

where $||y - Ax||_2^2$ is the sum of squared errors, $||x||_1$ a penalty function and λ the shrinkage parameter. The shrinkage parameter controls the strength of the penalty function, i.e. it controls the sparsity level. If $\lambda = 0$ no element in x will be zero and λ increases more elements are penalized to zero. To minimize a function it is sufficient to find its derivative and set that derivative to zero. Equation (3.33) is not differentiable since the penalty function i.e. the ℓ_1-norm is not differentiable, as explained in Section 2.2. To minimize equation (3.33) we will use majorization minimization. Define $G_k(x)$ to be

$$G_k(x) = O(x) + (x - x_k)^T(\alpha I - A^T A)(x - x_k)$$

where $G_k(x) \geq O(x)$ since we are adding a positive function to $O(x)$. The equality here is at $x = x_k$, that is $G_k(x_k) = O(x_k)$ [34]. Substituting $O(x)$ we get

$$G_k(x) = \|y - Ax\|_2^2 + \lambda\|x\|_1 + (x - x_k)^T(\alpha I - A^T A)(x - x_k). \tag{3.34}$$

To make sure that the function $G_k(x)$ is positive, the scalar α is chosen such that α is greater or equal to the maximum eigenvalue of $A^T A$ [35]. First let us consider the case where the case where $A = I$, that is,

$$O(x) = \|y - x\|_2^2 + \lambda\|x\|_1. \tag{3.35}$$

Expanding $O(x)$ gives,

$$O(x) = (y_1 - x_1)^2 + \lambda|x_1|_1 + (y_2 - x_2)^2 + \lambda|x_2|_2 + ... + (y_N - x_N)^2 + \lambda|x_N|_1.$$

Therefore we can minimize $O(x)$ by minimizing the term $(y_i - x_i)^2 + \lambda|x_i|_1$. Let $f(x) = (y - x)^2 + \lambda|x|$. Then,

$$f'(x) = -2(y - x) + \lambda\text{sign}(x).$$

Setting $f'(x) = 0$ we get,

$$y = x + \frac{\lambda}{2}\text{sign}(x). \tag{3.36}$$

Therefore solving for x is the same as applying the threshold function with a threshold of $\frac{\lambda}{2}$. Therefore we have

$$x = \text{soft}\left(y, \frac{\lambda}{2}\right).$$

Back to equation (3.34) we have

$$G_k(x) = \alpha\|x_k + \frac{1}{\alpha}A^T(y - Ax_k) - x\|_2^2 + \lambda\|x\|_1.$$

Minimizing $G_k(x)$ is the same as minimizing $\frac{1}{\alpha}G_k(x)$. Therefore we get [35],

$$\frac{1}{\alpha}G_k(x) = \|x_k + \frac{1}{\alpha}A^T(y - Ax_k) - x\|_2^2 + \frac{\lambda}{\alpha}\|x\|_1. \tag{3.37}$$

Comparing equation (3.37) with equation (3.35) we get that minimizing equation (3.37) is the same applying the soft threshold function to $x_k + \frac{1}{\alpha}A^T(y - Ax_k)$ with a threshold of $\frac{\lambda}{\alpha}$. Therefore we have

$$x_{k+1} = \text{soft}\left(x_k + \frac{1}{\alpha}A^T(y - Ax_k), \frac{\lambda}{\alpha}\right)$$

where $\alpha \geq \text{maxeig}(A^TA)$, where maxeig is the maximum eigenvalue. Therefore α should be greater than the maximum eigenvalue of A^TA [35].

Algorithm 3.5 Iterative hard thresholding, $H_k(z)$

```
• Step 1 Input matrix $A$, vector $b$ and the threshold $T = \frac{\lambda}{\alpha}$
• Step 2 Initialize $x_0 = 0$
• Step 3 Compute $x_{i+1} = \text{soft}(Ix_i + A^T(b - Ax_i), T)$
• Step 4 Repeat step 3 until convergence.
• Step 5 Output $x$
```

3.8 Conclusion

In this section we discussed the algorithms for solving sparse convex optimization. Of those algorithms basis pursuit, iterative hard thresholding and iterative soft thresholding are sparse convex optimization techniques while matching pursuit and orthogonal matching pursuit are sparse non-convex optimization techniques. For an algorithm to be a convex optimization technique it needs to solve the ℓ_1 minimization problem instead of the ℓ_0 minimization problem and both the constraints and the objective function should be convex. An algorithm is a sparse optimization algorithm if it returns a sparse solution.

Basis pursuit, iterative hard thresholding and iterative soft thresholding are convex optimization algorithms since they approximate problem (3.5) by minimizing ℓ_1 instead, and it has been shown that ℓ_1 is a convex function. Matching pursuit and orthogonal matching pursuit are non-convex optimization algorithms since they directly approximate problem (3.5), which is a non-convex problem. All of the algorithms we discussed in this section return sparse solutions, therefore they are all sparse optimization algorithms. In the next chapter we discuss the game sudoku and the application of the techniques discussed in Chapter 3 on a sudoku problem.

Chapter 4

Application

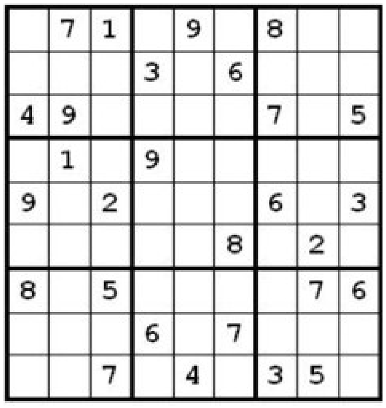
4.1 The sudoku problem

A sudoku problem is a 9×9 grid consisting of 9 3×3 grids, where a single player inserts numbers one to nine in each cell of the grid in such a way that all rows, all columns and all 3×3 grids contains numbers one to nine. The puzzle comes with some of the cells already filled in and these values are called clues. The number of clues for a sudoku problem to have a unique solution is not known. McQuire et al [28] proved that with 16 clues a unique solution is not possible. There however do exist problems with unique solutions having 17 clues. However, it is not guaranteed that all 17 clue sudoku problems will have a unique solution. To summarise, the rules for solving a sudoku problem are as follows:

- All rows should contain numbers one to nine.
- All columns should contain numbers one to nine.
- All three by three regions should contain numbers one to nine.
- All cells should be filled.

The modern sudoku was most likely designed anonymously by Howard Garns, a 74-year-old retired architect and freelance puzzle constructor from Connersville, Indiana, and first published in 1979 by Dell Magazines as Number Place (the earliest known examples of modern Sudoku) [27]. The puzzle was introduced in Japan by Nikoli in the paper Monthly Nikolist in April 1984 as Sūji wa dokushin ni kagiru, which also can be translated as the digits must be single or the digits are limited to one occurrence. At a later date, the name was abbreviated to sudoku by Maki Kaji [30]. A 9×9 sudoku puzzle with 3×3 grids is the most common sudoku puzzle but other variations exist [42]. There are different ways of solving a sudoku puzzle mathematically, this includes recursive backtracking [36] and integer programming [2]. In this mini-desertation we use the sparse optimization techniques to solve the the sudoku puzzle. We thus write the sudoku puzzle as the minimization problem.

4.2 Sudoku as a linear program



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

Figure 4.1: A 9×9 sudoku puzzle

Suppose we have a 9×9 sudoku puzzle, say S. The contents of each cell of S can be represented as S_n where $S_n \in \{1, 2, \ldots 9\}$ and $n \in \{1, 2, \ldots 81\}$, where the cells are numbered left to right and then top to bottom (a raster scan). Figure 4.1 represents an example of a 9×9 Sudoku puzzle. Let $i = (I(S_n = 1), I(S_n = 2), \ldots, I(S_n = 9))$ be an indicator vector [1], where

$$I(S_n = k) = \begin{cases} 1 & \text{if } S_n = k \\ 0 & \text{Otherwise.} \end{cases}$$

This indicator vector helps us to code the integers 1 to 9 as follows [39],

integer	binary vector	integer	binary vector
1	$(1,0,0,0,0,0,0,0,0)$	6	$(0,0,0,0,0,1,0,0,0)$
2	$(0,1,0,0,0,0,0,0,0)$	7	$(0,0,0,0,0,0,1,0,0)$
3	$(0,0,1,0,0,0,0,0,0)$	8	$(0,0,0,0,0,0,0,1,0)$
4	$(0,0,0,1,0,0,0,0,0)$	9	$(0,0,0,0,0,0,0,0,1)$
5	$(0,0,0,0,1,0,0,0,0)$		

Every entry in the 9×9 sudoku puzzle thus has a 9 dimensional variable representation, as represented above. Therefore we will have 729 variables $(9 \times 9 \times 9)$ for the puzzle [39]. We will denote the solution to our sudoku puzzle as $x_{729 \times 1} = [i_1, i_{2,}, \ldots, i_{81}]$. The solution of the sudoku puzzle must satisfy all the rules. In the form of a linear program we represent the sudoku rules as our constraints. The rules of the sudoku are as follows.

- **Rule 1:** All rows must contain all numbers 1 to 9. Each row constraint can be represented as a linear combination of x as follows.

$$\begin{aligned} \begin{bmatrix} I_{9\times9} & I_{9\times9} & I_{9\times9} & I_{9\times9} & I_{9\times9} & I_{9\times9} & I_{9\times9} & I_{9\times9} & I_{9\times9} & O_{9\times648} \end{bmatrix} x &= \underline{1} \\ \begin{bmatrix} O_{9\times81} & I_{9\times9} & I_{9\times9} & I_{9\times9} & I_{9\times9} & \dots & I_{9\times9} & I_{9\times9} & I_{9\times9} & O_{9\times567} \end{bmatrix} x &= \underline{1} \\ &\vdots \\ \begin{bmatrix} O_{9\times648} & I_{9\times9} & I_{9\times9} & I_{9\times9} & I_{9\times9} & I_{9\times9} & I_{9\times9} & I_{9\times9} & I_{9\times9} & I_{9\times9} \end{bmatrix} x &= \underline{1} \end{aligned}.$$

where $\underline{1} : 9 \times 1 = \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix}$, $I_{n\times n}$ an $n \times n$ identity matrix and $O_{m\times n}$ a matrix containing zeros.

The first equation represents the constraints that the first row contains numbers 1 to 9, and the further equations represent the constraints for the remaining 8 rows. The constraints are setup in such a way that $\underline{1} : 9 \times 1$ represents an indicator function, where 1 represents that the constraint is true and 0 otherwise. In this case it is a vector of ones only since the constraints should always be true.

- **Rule 2:** All columns must contain all numbers 1 to 9. Each column constraint can be represented as a linear combination of x as follows

$$\begin{aligned} [I_{9\times9}O_{9\times72}\dots I_{9\times9}O_{9\times72}]\, x &= \underline{1} \\ [O_{9\times9}I_{9\times9}O_{9\times72}\dots\ I_{9\times9}O_{9\times72}]\, x &= \underline{1} \\ &\vdots \\ [O_{9\times72}I_{9\times9}O_{9\times72}\dots I_{9\times9}O_{9\times72}]\, x &= \underline{1} \end{aligned}$$

The first equation represents the constraints that the first column contains numbers 1 to 9, and the equations that follow represents the constraints for the remaining 8 rows. The equation $[I_{9\times9}O_{9\times72}\dots I_{9\times9}O_{9\times72}]\, x$ represents a statement that the first column contains numbers 1 to 9, and by equating it to an indicator vector of ones means that the the statement is true.

- **Rule 3:** All 3×3 box must contain all numbers 1 to 9. Each box constraint can be represented as a linear combination of x as follows.

$$\begin{gathered}
\begin{bmatrix} J_{9\times27}O_{9\times54} & J_{9\times27}O_{9\times54} & J_{9\times27}O_{9\times540} \end{bmatrix} x = \underline{1} \\
\begin{bmatrix} O_{9\times27} & J_{9\times27}O_{9\times54} & J_{9\times27}O_{9\times54} & J_{9\times27}O_{9\times513} \end{bmatrix} x = \underline{1} \\
\begin{bmatrix} O_{9\times54}J_{9\times27}O_{9\times54} & J_{9\times27}O_{9\times54} & J_{9\times27}O_{9\times486} \end{bmatrix} x = \underline{1} \\
\begin{bmatrix} O_{9\times243} & J_{9\times27}O_{9\times54} & J_{9\times27}O_{9\times54} & J_{9\times27}O_{9\times293} \end{bmatrix} x = \underline{1} \\
\begin{bmatrix} O_{9\times270} & J_{9\times27}O_{9\times54} & J_{9\times27}O_{9\times54} & J_{9\times27}O_{9\times270} \end{bmatrix} x = \underline{1} \ . \\
\begin{bmatrix} O_{9\times297}J_{9\times27}O_{9\times54} & J_{9\times27}O_{9\times54} & J_{9\times27}O_{9\times243} \end{bmatrix} x = \underline{1} \\
\begin{bmatrix} O_{9\times486} & J_{9\times27}O_{9\times54} & J_{9\times27}O_{9\times54} & J_{9\times27}O_{9\times54} \end{bmatrix} x = \underline{1} \\
\begin{bmatrix} O_{9\times513} & J_{9\times27}O_{9\times54} & J_{9\times27}O_{9\times54} & J_{9\times27}O_{9\times27} \end{bmatrix} x = \underline{1} \\
\begin{bmatrix} O_{9\times540} & J_{9\times27}O_{9\times54} & J_{9\times27}O_{9\times54} & J_{9\times27} \end{bmatrix} x = \underline{1}
\end{gathered}$$

where $J_{9\times27} = I_{9\times9}I_{9\times9}I_{9\times9}$. Each equation represents a constraint that each box contains numbers 1 to 9, starting from the box on the top left corner to the right. Again here a vector of ones represents that a box constraint is true.

- **Rule 4:** All cells should be filled. An example of representing a cell constraint is as follows. The constraints that cells one and two are filled

$$\begin{gathered}
[1\,1\,1\,1\,1\,1\,1\,1\,1\,0\,0\,0\,0\,0\,0\,0\,0\,0\ldots0\,0]x = 1 \\
[0\,0\,0\,0\,0\,0\,0\,0\,0\,1\,1\,1\,1\,1\,1\,1\,1\,1\,0\ldots0\,0]x = 1
\end{gathered}$$

  We continue like this for all 81 cells [39]. The ones in this statement represents that the first cell should contain one of the numbers 1 to 9, and the second statement that cell 2 should contain one of the numbers 1 to 9.

- **Rule 5:** The clues can also be written as a linear combination x. As an example, the clue that cell 2 contains 5 and the clue that cell 1 contains 5 are as follows,

$$\begin{gathered}
[0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,1\,0\ldots0\,0]x \\
[0\,0\,0\,0\,1\,0\,0\,0\,0\,0\,0\,0\,0\ldots0\,0]x
\end{gathered}$$

Therefore the linear system can be represented as follows,

$$Ax = \begin{bmatrix} A_{row} \\ A_{col} \\ A_{box} \\ A_{cell} \\ A_{clue} \end{bmatrix} x = \begin{bmatrix} 1 \\ 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix} = b$$

where the matrix A is a matrix containing clues. The size of the matrix A will be of size $(4N^2 + c) \times N^3$ where c is the number of clues [1].

4.3 Application

4.3.1 Introduction

In this section we apply the algorithms explained in Chapter 3 on 100 9×9 unique sudoku problems with 17 clues. Therefore a solution vector x will be of size 729×1 and the matrix of constraints, A, will be of size 341×729. When solving for x, we cannot always find a solution that only has zeros and ones. This may happen depending on the algorithms used. In the case where the solution is not zeros and ones, we transform x in the following manner,

$$T(i) = \begin{cases} 1, & i = \text{maxindex}(T) \\ 0, & \text{otherwise} \end{cases}$$

where maxindex(y) is the position of the maximum value for y [39]. As an example, suppose the first 9 elements of x are T = (0.1, 0.01, 0.5, 0.03, 0.2, 0.05, 0.34, 0.25, 0.43), i.e. the first element of a sudoku puzzle. Then after the transformation we will have T = (0, 0, 1, 0, 0, 0, 0, 0, 0) which means that the first element of the sudoku puzzle is 3. We thus solve the following system of linear equations using the algorithms explained in Chapter 3:

$$Ax = \begin{bmatrix} A_{row} \\ A_{col} \\ A_{box} \\ A_{cell} \\ A_{clue} \end{bmatrix} x = \begin{bmatrix} 1 \\ 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix} = b.$$

4.3.2 ℓ_1-minimization (Basis pursuit)

For ℓ_1-minimization we solve the following minimization problem,

$$\begin{matrix} \min \|x\|_1 \\ Ax = y \end{matrix},$$

by first transforming the problem into a linear program. We then find x by solving the following linear program:

$$\min_{\Phi\alpha = s} c^T\alpha \quad \alpha \geq 0$$

where $\alpha = \begin{bmatrix} u \\ v \end{bmatrix}$ and $x = u - v$.

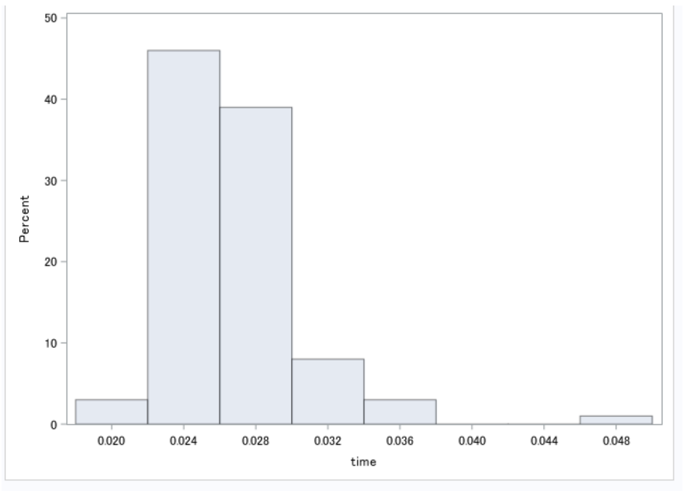


Figure 4.2: Histogram of times in seconds to complete a sudoku problem

Figure 4.2 represents the histogram of the times taken to solve the 100 sudoku puzzles. In this plot we note that most values are clustered together in the range 0.024 to 0.032 seconds. This illustrates that the time taken to solve one sudoku puzzle for this method is approximately the same and hence the algorithm is consistent.

Minimum time	maximum time	average time	standard deviation
$5.55556 \times e^{-6}$	$1.333361111 \times e^{-5}$	$7.32222222 \times e^{-6}$	$1.0597777778 \times e^{-6}$

Table 4.1: Summary statistics for time (seconds) taken to solve 100 sudoku puzzles

Table 4.1 represents summary statistics for the time taken to solve the puzzles. The average time to solve one sudoku puzzle was $7.32222222 \times e^{-6}$ with a standard deviation of $1.0597777778 \times e^{-6}$ hours. A small standard deviation indicates that the times taken to solve 100 sudoku puzzles are close to the average time of $7.32222222 \times e^{-6}$ hours. This means that each different sudoku problem was solved in approximately $7.32222222 \times e^{-6}$ hours. The minimum and maximum time to solve one Sudoku puzzle are $5.55556 \times e^{-6}$ hours and $1.333361111 \times e^{-5}$ hours respectively.

4.3.3 Iterative soft thresholding

For iterative soft threshold we use the residual error, and this is defined to be the true value for b minus the estimated value for b i.e. residuals $= b - A\hat{x}$ where $\hat{x}$. Since the residual is a vector, we can stop the iteration when the maximum of the elements of the residual is small. We thus solve

$$\min \|Ax - y\|_2^2 + \lambda \|x\|_1 .$$

For iterative soft thresholding, we calculated the value for x in the following manner

$$x_{k+1} = \text{soft}\left(x_k + \frac{1}{\alpha}A^T(y - Ax_k), \frac{\lambda}{\alpha}\right).$$

We chose α to be the maximum of the eigenvalues A^TA [35]. To get a good estimate of the solution, we need to select the best λ, which controls the level of sparsity. We selected λ using trial and error. We measure the performance of the algorithm using the time taken to solve the puzzles and the percentage of correct cells in the sudoku puzzle. The percentage of correct cells in the sudoku puzzle is defined as:

$$\%\text{correct} = \frac{\text{number of correct cells}}{\text{number of cells}} \times 100.$$

Since there are 81 cells, if %correct equals 100, then all cells have been estimated correctly. With this algorithm all the sudoku puzzles have been solved correctly.

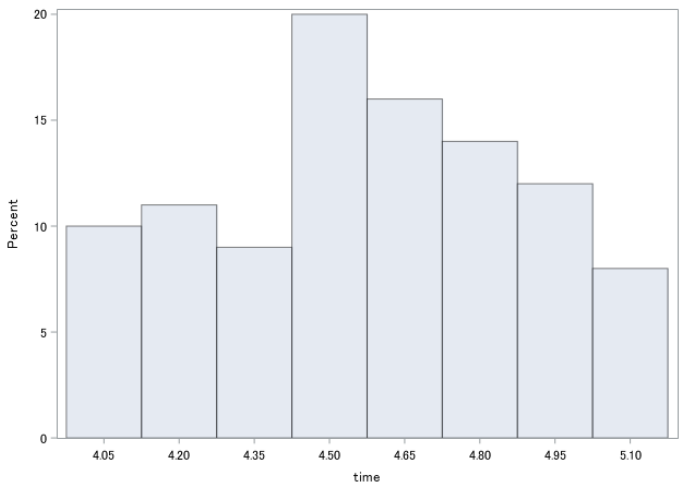


Figure 4.3: Histogram of time in hours to complete a sudoku problem

Figure 4.3 represents the histogram of times taken to solve 100 sudoku puzzles. In this plot we note that most values are clustered together in the range 4.5 to 4.95 seconds. This illustrate that the time taken to solve one sudoku puzzle for this method is approximately the same and hence the algorithm is consistent.

Minimum time	maximum time	average time	standard deviation
4.01147	5.09826	4.574537	0.30793

Table 4.2: Summary statistics for time (hours) taken to solve 100 sudoku sudoku puzzles

Table 4.2 represents summary statistics for the times taken to solve the puzzles. The average time to solve one sudoku puzzle was 4.8 hours with a standard deviation of 0.30793 hours. The standard deviation is small with respect to hours. A small standard deviation indicates that the times taken to solve 100 sudoku puzzles are close to the average time of 4.574537 hours. This means that each different sudoku problem was solved in approximately 4.574537 hours. The minimum and maximum time taken to solve one sudoku puzzle is 4.01147 hours and 5.09826 hours respectively

4.3.4 Iterative hard thresholding

For iterative hard threshold we want to solve the following minimization problem for x,

$$\min \|Ax - y\|_2^2 + \lambda \|x\|_1 .$$

We solve for x iteratively using the following

$$x_{i+1} = H_k \left[I x_i + A^T(b - Ax_i) \right]$$

where

$$H_k(\underset{\sim}{z}) = \begin{cases} 0 & |z_i| < \lambda \\ z_i & |z_i| > \lambda \end{cases} .$$

Here λ was selected by trial and error. With this algorithm, all the sudoku puzzles can be solved exactly.

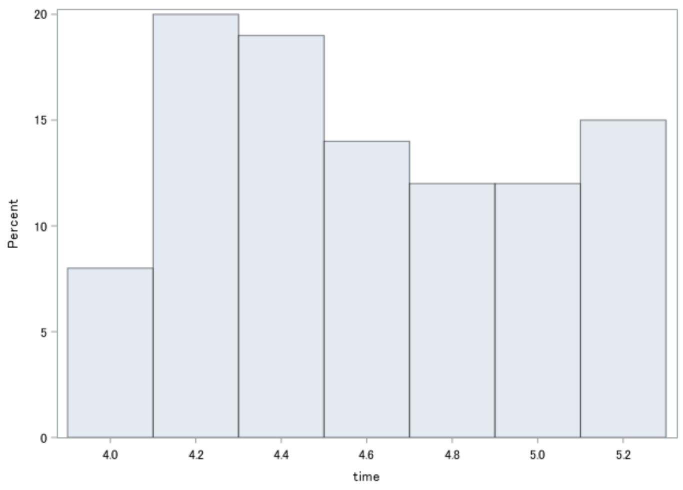


Figure 4.4: Histogram of time in hours to complete a sudoku problem

Figure 4.4 represents the distribution of the time taken to solve one sudoku puzzle. In this plot we note that most values are clustered together in the range 4.2 to 4.6 hours. This illustrate that the time taken to solve one sudoku puzzle for this method is approximately the same and hence the algorithm is consistent.

Minimum time	maximum time	average time	standard deviation
4.00137	5.28171	4.599476	0.387

Table 4.3: Summary statistics for time (hours) taken to solve 100 sudoku puzzles

Table 4.3 represents summary statistics for the time taken to solve the puzzles. The average time to solve one sudoku puzzle was 4.599476 hours with a standard deviation of 0.387 seconds. The standard deviation is small with respect to hours. A small standard deviation indicates that the times taken to solve 100 sudoku puzzles are close to the average time of 4.599476 hours. This means that each different sudoku problem was solved in approximately 4.599476 hours. The minimum and maximum time taken to solve the puzzles are 4.00137 and 5.28171 seconds.

4.3.5 Orthogonal Matching pursuit

Orthogonal matching pursuit iteratively selects columns that are correlated with the current residuals the best. At each iteration, only one column is selected. This is done by selecting columns that satisfy the following.

$$|\langle R_k, a_{S_k}\rangle| = \sup |\langle R_k, a_i\rangle| .$$

We then use the selected columns to estimate x. For instance, suppose D represents a matrix made up of the columns that are selected. Then we estimate x as follows:

$$\tilde{x} = (D^T D)^{-1} D^T b.$$

This will have a solution if $(D^T D)^{-1}$ exists. In Section 4.3.3.2 we explained that $(D^T D)^{-1}$ exists when $D : m \times n$ must not be under-determined (if $\text{rank}(D) = m \leq n$). In our case we have $Ax = b$, where $A : 325 \times 729$ is a matrix of the sudoku restrictions. Since there are 729 columns in A and in each iteration we are selecting one column, there will be less than 729 iterations. This is because after we are finished with 729 columns there will no longer be any more columns to choose from. Since we are adding one column at a time in D, D will be under-determined until we add the 325th column. This means that our algorithm will run for 325 iterations. From 326 on wards $(D^T D)^{-1}$ will no longer exists. We could make use of the generalized inverse but this does not provide a unique solution. We used the percentage of elements in the sudoku table that are predicted correctly and the time taken to solve the puzzles to measure the performance of the algorithm. Percentage of elements in the sudoku table that are predicted correctly is defined as follows:

$$\%\text{correct} = \frac{\text{number of correct cells}}{\text{number of cells}} \times 100.$$

To get a threshold in each iteration, we calculated the percentage of correct solutions and we chose the x value in the iteration that gives the best proportion of correct solutions. With this algorithm not all of the Sudoku puzzles were solved exactly. This is due to the fact that non-convex methods may converge to a solution that is not global.

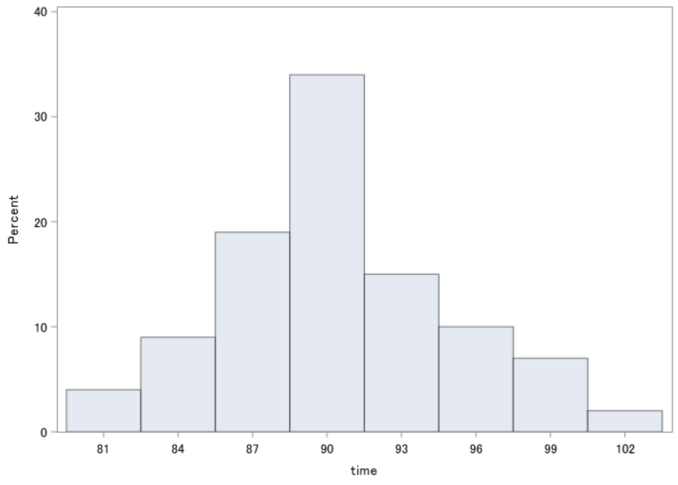


Figure 4.5: Histogram of time in seconds to complete a sudoku problem

Figure 4.5 represent the distribution of the time taken to solve one sudoku puzzle. Table 4.4 represents summary statistics for the time taken to solve the puzzles.

Minimum time	maximum time	average time	standard deviation
0.0222383333	0.02833583333	0.025151069444	0.001253716667

Table 4.4: Summary statistics for time taken to solve 100 sudoku puzzles in seconds

The average time to solve one sudoku puzzle was 0.025 hours with a standard deviation of 0.00125 hours. The standard deviation is small with respect to hours. A small standard deviation indicates that the times taken to solve 100 sudoku puzzles are close to the average time of 0.025 hours. This means that each different sudoku problem was solved in approximately 0.025 hours. For each puzzle, we deleted the values that were solved incorrectly and then ran the algorithm again. However, this technique did not provide better results.

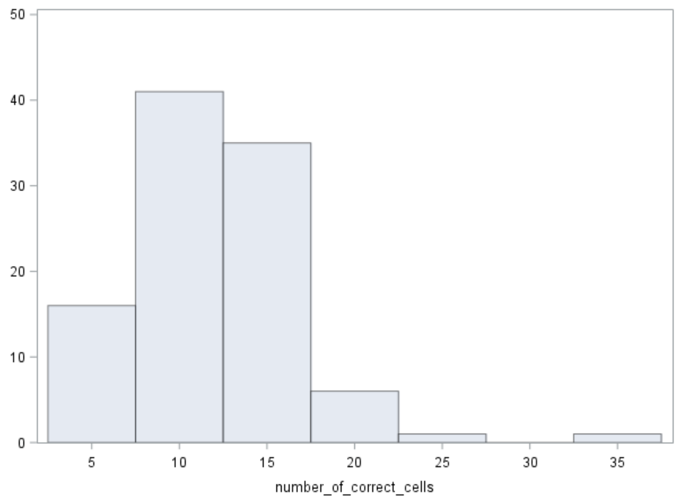


Figure 4.6: Histogram of the number of correct cells for 100 sudoku problems

Figure 4.6 represents a histogram of the number of correct cells for each of the 100 sudoku problem. Figure 4.6 shows that more data lies below 20. This means that for most puzzles, the orthogonal matching pursuit algorithm managed to solve less than 20 cells of the 64 unknowns correctly. The minimum number of correct cells is 5 and the maximum number of correct cells is 35

4.3.6 Discussion

	Number of solved puzzles	Total time(sec)	Average time(sec)
ℓ_1-minimization (BP)	100	2.6359999	0.02636
IST	100	27447.2191	274.47222
IHT	100	27576.856	275.96856
OMP	0	9054.385	90.54385

Table 4.5: Summary results of all algorithms

Table 4.5 represents the summary for all the different methods we considered in this mini-dissertation. The algorithms are ordered according to the number of correct solutions and average time, with the first order being the correct solution. Based on our results, it has been noted that the ℓ_1-minimization algorithm is the best method for solving the sudoku puzzles since it solved all the sudoku puzzles correctly in a short amount of time (0.026 seconds per puzzle on average). Iterative hard threshold and iterative soft threshold solved all the sudoku puzzles correctly but it took a long amount of time to converge (4.85 hours and 4.97 hours per puzzle on average respectively). Orthogonal matching pursuit is a non-convex method since it estimates the ℓ_0-minimization problem directly. It took 90.66 seconds to solve each sudoku puzzle, which is faster than iterative soft threshold and iterative hard threshold, but the problem is that not all cells are estimated correctly. The second step for orthogonal matching pursuit improved the results but it still did not estimate all cells correctly.

Chapter 5

Conclusion

This mini-dissertation provided an overview of sparse optimization algorithms. The algorithms considered here were, the ℓ_1-minimization, iterative hard thresholding, iterative soft thresholding and orthogonal matching pursuit. We then applied the algorithms to 100 9×9 unique sudoku puzzles each with 17 clues. The times taken to solve the 100 sudoku puzzles were recorded, analyzed and compared. The performance of the algorithm was measured by the number of sudoku puzzles solved completely and the time taken to solve the puzzle. It has been noted that the ℓ_1-minimization performed better than iterative hard thresholding, iterative soft thresholding and orthogonal matching pursuit since it solves all the sudoku puzzles completely in a short period of time.

We presented an algorithm for non-convex sparse optimization, namely orthogonal matching. This algorithm was included to show that for a non-convex algorithm it may take time to converge to a global solution. We discussed the advantages and disadvantages of each algorithm presented. We implemented the algorithms to solve 100 sudoku puzzles. We finally, compared the performance of sparse convex optimization to sparse non-convex optimization on the application at hand. We noted that the algorithms for convex optimization solved the puzzles completely while the algorithm for non-convex optimization did not. This shows that for the problem at hand convex optimization methods performed better.

For iterative hard thresholding and iterative soft thresholding, we needed to find a suitable threshold λ. In this mini-dissertation we found the threshold by trial and error, but there are various methods for finding the threshold. One of the methods is called cross-validation [25]. In this method we ran the algorithm for different values of λ and each time we calculated the percentage of correct solutions. We then chose the λ that returns the best percentage of correct solutions. For future work, we can look at sudoku puzzles with non-unique solutions and also compare the performance of convex and non-convex algorithms on sudoku puzzles with non-unique solutions.

Bibliography

[1] P Babu, K Pelckmans, P Stoica, and J. Li. Linear systems, sparse solution, and sudoku. *IEEE Signal Processing Letters*, 17(1), January 2010.

[2] A Bartlett, TP Chartier, AN Langville, and TD Rankin. An integer programming model for the sudoku problem. *Journal of Online Mathematics and its Applications*, 8(1), 2008.

[3] T Blumensath and M Davies. Iterative thresholding for sparse approximations. *Fourier Analysis and Applications, Special Issue on Sparsity*, 2008.

[4] S Boyd and L Vaddenberghe. *Convex Optimization.* Cambridge University Press, 2014.

[5] A Buluç, JT Fineman, M Frigo, JR Gilbert, and CE Leiserson. Parallel sparse matrix-vector and matrix-transpose-vector multiplication using compressed sparse blocks. In *Proceedings of the twenty-first Annual Symposium on Parallelism in Algorithms and Architectures*, pages 233–244. ACM, 2009.

[6] TT Cai and L Wang. Orthogonal matching pursuit for sparse signal recovery with noise. *Transactions on Information Theory*, 57(7), July 2011.

[7] EJ Candes. Mathematics of sparsity (and few other things). *Proceedings of the International Congress of Mathematicians, Seoul, South Korea*, 2014.

[8] EJ Candes and T Tao. Decoding by linear programming. *IEEE Transactions on Information Theory*, 51(12):489–509, 2005.

[9] E.J Candes and M.B Wakin. An introduction to compressive sensing. *IEEE Signal Processing Magazine*, (21), March 2008.

[10] M Cavazzuti. *Optimization Methods: From Theory to Design Scientific and Technological Aspects in Mechanics.* Springer Science & Business Media, 2012.

[11] S.S Chen, D.L Donoho, and M.A Saunders. Atomic decomposition by basis pursuit. *SIAM REV Society for Industrial and Applied Mathematics*, 43(1):129–159, 2001.

[12] P Cholak, N Greenberg, and JS Miller. Uniform almost everywhere domination. *arXiv:math/0506019v2*, November 2005.

[13] G Dantzig. *Linear programming and extensions.* Princeton University Press, 2016.

[14] D Donoho, M Elad, and V Temlyakov. Stable recovery of sparse overcomplete representations in the presence of noise. *IEEE Transactions on Information Theory*, 52(1), 2006.

[15] DL Donoho. Compressed sensing. *IEEE Transactions on Information Theory*, 52(4):1289–1306, April 2006.

[16] YC Eldar and G Kutyniok. *Compressed sensing: theory and applications.* Cambridge University Press, 2012.

[17] S Foucart. Hard thresholding pursuit: an algorithm for compressive sensing. *SIAM Journal on Numerical Analysis*, pages 2543–2563, 2011.

[18] R Gagan, S Arabinda, and Sahoo. A comparative study of some greedy pursuit algorithms for sparse approximation. *Signal Processing Conference, 2009 17th European.*, pages 398–402, 2009.

[19] J Gallier and J Quaintance. *Fundamentals of Linear Algebra and Optimization.* University of Pennsylvania, 2017.

[20] A Gupta, G Karypis, and V Kumar. Highly scalable parallel algorithms for sparse matrix factorization. *Parallel and Distributed Systems, IEEE Transactions*, 8(5):502–520, May 1997.

[21] R Huamin, P Hong, OS Ingvor, and M Thomas B. Greedy vs. l_1 convex optimization in sparse coding : Comparative study in abnormal event detection. *Paper presented at ICML '15 workshop, Lille, France*, 2015.

[22] M Jaggi. *Sparse convex optimization methods for machine learning.* PhD thesis, ETH Zurich, 2011.

[23] F Jianqing, Lv Jinchi, and Lei Q. Sparse high-dimensional models in economics. *Annual Review of Economics*, pages 291–317, September 2011.

[24] L Jiao, L Bo, and L Wang. Fast sparse approximation for least squares support vector machine. *Neural Networks, IEEE Transactions*, 18(3):685–697, May 2007.

[25] R Kohavi. A study of cross-validation and bootstrap for accuracy estimation and model selection. *Appears in the international joint conference on artificial intelligence(IJCAI)*, 1995.

[26] R Larson. *Elementary Linear Algebra.* Cengage Learning, 1999.

[27] G Lev. The answer men. *Time New york*, March 2013.

[28] G McGuire, B Tugemann, and G Civario. There is no 16-clue sudoku: Solving the sudoku minimum number of clues problem via hitting set enumeration. *arXiv:1201.0749*, August 2013.

[29] HP Mulholland. On generalizations of minkowski's inequality in the form of a triangle inequality. *Proceedings of the London Mathematical Society*, Proceedings of the London Mathematical Society(1):294–307, June 1949.

[30] E Pegg. Ed Pegg Jr's math games: Sudoku variations. *MAA online. The Mathematical Association of America*, October 2006.

[31] S Qaisar, RM Bila, W Iqbal, M Naureen, and S Lee. Compressive sensing: From theory to applications, a survey. *Journal of Communications and Networks*, 15(5):443–456, 2013.

[32] C Ramirez, V Kreinovich, and M Argaez. Why l_1 is a good approximation to l_0 : A geometric explanation. *Journal of Uncertain Systems*, 7, 2013.

[33] R Robere. Interior point methods and linear programming. *University of Toronto*, 2012.

[34] I Selesnick. Penalty and shrinkage functions for sparse signal processing. *Connexions*, 11, 2012.

[35] Ivan W Selesnick. Sparse signal restoration. *Connexions*, pages 1–13, 2009.

[36] PJ Simha, KV Suraj, and T Ahobala. Recognition of numbers and position using image processing techniques for solving sudoku puzzles. In *Advances in Engineering, Science and Management (ICAESM), 2012 International Conference on*, pages 1–5. IEEE, 2012.

[37] J Stewart. *Calculus: early transcendentals.* Cengage Learning, 2010.

[38] Y Tang, Z Wu, and C Zhu. An improved strategy for solving sudoku by sparse optimization methods. arXiv preprint arXiv:1507.05995, 2015.

[39] Y Tang, Z Wu, and C Zhu. An improved strategy for solving sudoku by sparse optimization methods. *arXiv:1507.05995*, 2015.

[40] JA Tropp and SJ Wright. Computational methods for sparse solution of linear inverse problems. *Proceedings of the IEEE*, 98(6):948–958, 2010.

[41] J Wright, A Yang, A Ganesh, S Sastry, and Y Ma. Robust face recognition via sparse representation. *Pattern Analysis and Machine Intelligence, IEEE Transactions*, 2008.

[42] B Youngkyun, K Bokyeong, Y Seongchul, and C Donguk. Effects of two types of sudoku puzzles on students logical thinking. In *Proceedings of the second European conference on games based learning*, pages 19–24. Academic Publishing Limited UK, 2008.

[43] Z Zhang, Y Xu, J Yang, X Li, and D Zhang. A survey of sparse representation: algorithms and applications. arXiv:1602.07017v1 [cs.CV], February 2016.

Appendix

The ℓ_1-minimization algorithm

```
proc iml; use datamy;
read all into jac;
use solu; read all into solu;
/* Start timer */ %let _timer_start = %sysfunc(datetime());
*----------------------row constraints-------------;
xx=repeat(I(9),1,9); do i=81 to (648-81) by 81;
        c=c//(J(9,i,0)||xx||j(9,648-i,0));
end; Ar=(xx||j(9,648,0))//c//(j(9,648,0)||xx);
*--------------------column constraints------------;
xxx=I(9)||J(9,72,0);
c1=repeat(xxx,1,9);
do i=9 to (72-9) by 9;
        cc=cc//(j(9,i,0)||(repeat(xxx,1,8))||I(9)||j(9,72-i,0));
end;
Ac=c1//cc//repeat((j(9,72,0)||I(9)),1,9);
*--------------------box constraints---------------------------;
jo=repeat(I(9),1,3)||j(9,54,0);
b1=repeat(jo,1,2)||repeat(I(9),1,3)||j(9,540,0);
do i=27 to (2*27) by 27;
        bbb=bbb//(j(9,i,0)||repeat(jo,1,2)||repeat(I(9),1,3)
    ||j(9,540-i,0));
end;
aba1=(j(9,243,0)||repeat(jo,1,2)||repeat(I(9),1,3)||j(9,297,0));
aba2=(j(9,270,0)||repeat(jo,1,2)||repeat(I(9),1,3)||j(9,270,0));
aba3=(j(9,297,0)||repeat(jo,1,2)||repeat(I(9),1,3)||j(9,243,0));
ab1=(j(9,486,0)||repeat(jo,1,2)||repeat(I(9),1,3)||j(9,54,0));
ab2=(j(9,513,0)||repeat(jo,1,2)||repeat(I(9),1,3)||j(9,27,0));
ab3=j(9,540,0)||repeat(jo,1,2)||repeat(I(9),1,3);
Ab=b1//bbb//aba1//aba2//aba3//ab1//ab2//ab3;
*--------------------all cell filled constraints---------------;
ccc=j(1,9,1);
do i=1 to (729) by 9;
   cc1=j(1,729,0);
        cc1[,i:i+8]=ccc;
        Acf=Acf//cc1; end;
*----------------------clues constraints-------------;
```

```
do all=1 to 100;
solution=solu[all,]`;
AAA=jac[all,]`;
do abb=1 to (nrow(AAA));
     if AAA[abb]^=0 then clu=clu//(j(1,(abb-1)*9+(AAA[abb]-1),0)||
     1||j(1,728-((abb-1)*9+(AAA[abb]-1)),0));
end;
ss=clu;
free clu;
*-------------------matrix of constraints and vector b---------;
A=Ar//Ac//Ab//Acf//ss; b=j(nrow(A),1,1);
coef=A||(-A);    *phi;
object=j(ncol(coef),1,1);    *c;
b=j(nrow(A),1,1);     *b;
*-------------------------optimizing-----------------------;
call lpsolve(rc,objv,x,dual,rd,object,coef,b); dua=dual`;
sol_xx=x[1:729,]-x[730:1458,]; xx_sol=shape(sol_xx,81,9);
maxx=xx_sol[,<>]; maxind=xx_sol[,<:>]; check=(solution=maxind);
perc_corr=perc_corr//((sum(check))/nrow(check))*100;
alll=alll//all;
dur = dur//(datetime() - &_timer_start);
end;
*-----------------------uncumulate time---------------;
time_in_sec=dur[1]//j(nrow(dur)-1,1,1);
do zip=2 to (nrow(dur));
  time_in_sec[zip]=dur[zip]-dur[zip-1];
end;
total_time_in_sec=sum(time_in_sec);
av_time_in_sec=mean(time_in_sec);
std_time_in_sec=std(time_in_sec);
print alll perc_corr time_in_sec ;
print total_time_in_sec av_time_in_sec std_time_in_sec;
quit;
```

Iterative soft threshold algorithm

```
proc iml;
use datamy;
read all into jac; use sol;
read all into solu;
```

```
/* Start timer */ %let _timer_start = %sysfunc(datetime());
*-----------------------row constraints--------------;
xx=repeat(I(9),1,9);
do i=81 to (648-81) by 81;
        c=c//(J(9,i,0)||xx||j(9,648-i,0));
end;
Ar=(xx||j(9,648,0))//c//(j(9,648,0)||xx);
*---------------------column constraints-------------;
xxx=I(9)||J(9,72,0); c1=repeat(xxx,1,9);
do i=9 to (72-9) by 9;
        cc=cc//(j(9,i,0)||(repeat(xxx,1,8))||I(9)||j(9,72-i,0));
end; Ac=c1//cc//repeat((j(9,72,0)||I(9)),1,9);
*---------------------box constraints--------------------------;
jo=repeat(I(9),1,3)||j(9,54,0);
b1=repeat(jo,1,2)||repeat(I(9),1,3)||j(9,540,0);
do i=27 to (2*27) by 27;
        bbb=bbb//(j(9,i,0)||repeat(jo,1,2)
    ||repeat(I(9),1,3)||j(9,540-i,0));
end;
aba1=(j(9,243,0)||repeat(jo,1,2)||repeat(I(9),1,3)||j(9,297,0));
aba2=(j(9,270,0)||repeat(jo,1,2)||repeat(I(9),1,3)||j(9,270,0));
aba3=(j(9,297,0)||repeat(jo,1,2)||repeat(I(9),1,3)||j(9,243,0));
ab1=(j(9,486,0)||repeat(jo,1,2)||repeat(I(9),1,3)||j(9,54,0));
ab2=(j(9,513,0)||repeat(jo,1,2)||repeat(I(9),1,3)||j(9,27,0));
ab3=j(9,540,0)||repeat(jo,1,2)||repeat(I(9),1,3);
Ab=b1//bbb//aba1//aba2//aba3//ab1//ab2//ab3;
*---------------------all cell filled constraints---------------;
ccc=j(1,9,1);
do i=1 to (729) by 9;
   cc1=j(1,729,0);
        cc1[,i:i+8]=ccc;
        Acf=Acf//cc1; end;
*---------------------------clues constraints-----------------------;
do all=1 to 2;
solution=solu[all,]`;
AAA=jac[all,]`;
do abb=1 to 81;
    if AAA[abb]^=0 then clu=clu//(j(1,(abb-1)*9+(AAA[abb]-1),0)
    ||1||j(1,728-((abb-1)*9+(AAA[abb]-1)),0));
```

```
end;
ss=clu;
free clu;
*——————————————————matrix of constraints and vector b————————;
A=Ar//Ac//Ab//Acf//ss;
b=j(nrow(A),1,1);
*==========iterative soft thresholding===================;
x = A; d1=norm(x,"L2");
A=x/d1; aa=(t(A))*A;
call eigen(eigv,eigve,aa);
al=max(eigv);
*lam=95;
do lam=-30 to 30 by 5;
        x_0 = j(ncol(x),1,0.5);
        T = lam/al;      st=10000000000;
        sto=0.00000000000000000000000000000001;
        m=2;
        ite=0;
    do i=1 to m while(st>sto);
                        xk = x_0+(ginv(A))*(1/al)*(b-A*x_0);
                        do ii=1 to nrow(xk);
                                if (xk[ii]<(-T)) then xk[ii]=xk[ii]+T;
                                if (xk[ii]>(T)) then xk[ii]=xk[ii]-T;
                                else xk[ii]=0;
                        end;
                y_sparse=A*(xk);
                st=max(abs(b-y_sparse));
                nor=norm(xk,"L1");
                x_0=xk;
                ite=ite+1;
        end;
        error=error//st;
        sod=shape(xk,81,9);
    maxind=sod[,<:>];
        check=(solution=maxind);
correct=(sum(check))/nrow(check);
savecorr=savecorr//correct;
        norm=norm//nor;
end;
```

```
erro1=erro1//min(error);
normL1=min(norm);
perc_corr=perc_corr//(max(savecorr))*100;
Num_correct=Num_correct//(max(savecorr))*81;
free step;
free savecorr;
free norm;
dur = dur//(datetime() - &_timer_start);
end;
*----------summary of time statistics------------;
time_in_sec=dur[1]//j(nrow(dur)-1,1,1);
do zip=2 to (nrow(dur));
  time_in_sec[zip]=dur[zip]-dur[zip-1];
end;
total_time_in_sec=sum(time_in_sec);
av_time_in_sec=mean(time_in_sec);
std_time_in_sec=std(time_in_sec);
print perc_corr Num_correct erro1 time_in_sec;
print total_time_in_sec av_time_in_sec std_time_in_sec;
quit;
```

Iterative hard threshold

```
proc iml;
use datamy;
read all into jac;
use sol;
read all into solu;
/* Start timer */ %let _timer_start = %sysfunc(datetime());
*----------------------row constraints-------------;
xx=repeat(I(9),1,9);
do i=81 to (648-81) by 81;
        c=c//(J(9,i,0)||xx||j(9,648-i,0));
end; Ar=(xx||j(9,648,0))//c//(j(9,648,0)||xx);
*--------------------column constraints------------;
xxx=I(9)||J(9,72,0);
c1=repeat(xxx,1,9);
do i=9 to (72-9) by 9;
        cc=cc//(j(9,i,0)||(repeat(xxx,1,8))||I(9)||j(9,72-i,0));
end;
```

```
Ac=c1//cc//repeat((j(9,72,0)||I(9)),1,9);
*--------------------box constraints----------------------------;
jo=repeat(I(9),1,3)||j(9,54,0);
b1=repeat(jo,1,2)||repeat(I(9),1,3)||j(9,540,0);
do i=27 to (2*27) by 27;
        bbb=bbb//(j(9,i,0)||repeat(jo,1,2)
    ||repeat(I(9),1,3)||j(9,540-i,0));
end;
aba1=(j(9,243,0)||repeat(jo,1,2)||repeat(I(9),1,3)||j(9,297,0));
aba2=(j(9,270,0)||repeat(jo,1,2)||repeat(I(9),1,3)||j(9,270,0));
aba3=(j(9,297,0)||repeat(jo,1,2)||repeat(I(9),1,3)||j(9,243,0));
ab1=(j(9,486,0)||repeat(jo,1,2)||repeat(I(9),1,3)||j(9,54,0));
ab2=(j(9,513,0)||repeat(jo,1,2)||repeat(I(9),1,3)||j(9,27,0));
ab3=j(9,540,0)||repeat(jo,1,2)||repeat(I(9),1,3);
Ab=b1//bbb//aba1//aba2//aba3//ab1//ab2//ab3;
*--------------------all cell filled constraints---------------;
ccc=j(1,9,1);
do i=1 to (729) by 9;
   cc1=j(1,729,0);
        cc1[,i:i+8]=ccc;
        Acf=Acf//cc1;
end;
*--------------------clues constraints-------------;
do all=1 to 2;
solution=solu[all,]`;
AAA=jac[all,]`;
do abb=1 to (nrow(AAA));
    if AAA[abb]^=0 then clu=clu//(j(1,(abb-1)*9+(AAA[abb]-1),0)
    ||1||j(1,728-((abb-1)*9+(AAA[abb]-1)),0));
end;
ss=clu;
free clu;
*-------------------matrix of constraints and vector b---------;
A=Ar//Ac//Ab//Acf//ss;
b=j(nrow(A),1,1);
x_sol=ginv(A)*b;
sol_x=shape(x_sol,81,9);
minxsol=sol_x[,<:>];
sud=shape(minxsol,9,9);
```

```
*=========iterative hard thresholding============;
x = A;
d1=norm(x,"L2");
A=x/d1;
do k=0 to 1 by 0.1;
        x_0 = j(ncol(x),1,0);
        *k = 0.07;
        st=10000000000;
        sto=0.05;
        m=2;
        ite=0;
    do i=1 to m while(st>sto);
                        xk = x_0+(ginv(A))*(b-A*x_0);
                in = ((abs(xk))>k);
                x_k = (in)#(xk);
                y_sparse=A*(x_k);
                st=max(abs(b-y_sparse));
                nor=norm(x_k,"L1");
                x_0=x_k;
                ite=ite+1;
        end;
        error=error//st;
        x_sparse=shape(x_k,81,9);
        maxx=x_sparse[,<>];
    maxind=x_sparse[,<:>];
        check=(solution=maxind);
correct=(sum(check))/nrow(check);
savecorr=savecorr//correct;
end;
sudoku=shape(maxind,9,9);
perc_corr=perc_corr//((max(savecorr))*100);
Num_correct=Num_correct//((max(savecorr))*81);
error1=error1//min(error); free error;
free savecorr;
dur = dur//(datetime() - &_timer_start);
end;
*--------------------summary of time statistics -----------;
time_in_sec=dur[1]//j(nrow(dur)-1,1,1);
do zip=2 to (nrow(dur));
```

```
  time_in_sec[zip]=dur[zip]-dur[zip-1];
end; total_time_in_sec=sum(time_in_sec);
av_time_in_sec=mean(time_in_sec);
std_time_in_sec=std(time_in_sec);
print perc_corr Num_correct error1 time_in_sec;
print total_time_in_sec av_time_in_sec std_time_in_sec;
quit;
```

Orthogonal matching pursuit

```
proc iml;
use datamy;
read all into jac;
use sol;
read all into solu;
/* Start timer */ %let _timer_start = %sysfunc(datetime());
*--------------------------row constraints-----------------;
xx=repeat(I(9),1,9);
do i=81 to (648-81) by 81;
         c=c//(J(9,i,0)||xx||j(9,648-i,0));
end; Ar=(xx||j(9,648,0))//c//(j(9,648,0)||xx);
*------------------------column constraints---------------;
xxx=I(9)||J(9,72,0);
c1=repeat(xxx,1,9);
do i=9 to (72-9) by 9;
         cc=cc//(j(9,i,0)||(repeat(xxx,1,8))||I(9)||j(9,72-i,0));
end;
Ac=c1//cc//repeat((j(9,72,0)||I(9)),1,9);
*------------------------box constraints-------------------------------------;
jo=repeat(I(9),1,3)||j(9,54,0);
b1=repeat(jo,1,2)||repeat(I(9),1,3)||j(9,540,0);
do i=27 to (2*27) by 27;
         bbb=bbb//(j(9,i,0)||repeat(jo,1,2)||repeat(I(9),1,3)
    ||j(9,540-i,0));
end;
aba1=(j(9,243,0)||repeat(jo,1,2)||repeat(I(9),1,3)||j(9,297,0));
aba2=(j(9,270,0)||repeat(jo,1,2)||repeat(I(9),1,3)||j(9,270,0));
aba3=(j(9,297,0)||repeat(jo,1,2)||repeat(I(9),1,3)||j(9,243,0));
ab1=(j(9,486,0)||repeat(jo,1,2)||repeat(I(9),1,3)||j(9,54,0));
ab2=(j(9,513,0)||repeat(jo,1,2)||repeat(I(9),1,3)||j(9,27,0));
```

```
ab3=j(9,540,0)||repeat(jo,1,2)||repeat(I(9),1,3);
Ab=b1//bbb//aba1//aba2//aba3//ab1//ab2//ab3;
*———————————————all cell filled constraints————————————;
ccc=j(1,9,1);
do i=1 to (729) by 9;
   cc1=j(1,729,0);
        cc1[,i:i+8]=ccc;
        Acf=Acf//cc1;
end;
*———————————————clues constraints—————————;
do all=1 to 100;
solution=solu[all,]`;
AAA=jac[all,]`;
do abb=1 to (nrow(AAA));
     if AAA[abb]^=0 then clu=clu//(j(1,(abb-1)*9+(AAA[abb]-1),0)
     ||1||j(1,728-((abb-1)*9+(AAA[abb]-1)),0));
end;
ss=clu;
free clu;
*————————————matrix of constraints and vector b————————;
A=Ar//Ac//Ab//Acf//ss; b=j(nrow(A),1,1);
*——————————Orthogonal matching pursuit————————;
z=A;
y=b;
c=0.000000000000000000000000000000000000001;
n=ncol(z);
*p=nrow(y);
*p=81;
*____________________normalizing A___________________;
do j=1 to n; n1=norm(z[,j],"L2"); d1=d1||n1; end; A=z/d1;
*________________orthogonal matching pursuit__________________;
do p=1 to 200;
r0=y; indicator=1;
sol_x=j(729,1,0);
normr=1000000000000000000000000000;
        do i=1 to p while(normr>c);
                dot_product=abs((r0`)*A);
                loc=dot_product[<:>];
                sol_x[loc]=1;
```

```
                    aj=A[,loc];
                    D=D||aj; *storing chosen columns;
                    DD=D`;
                    x_hat=(ginv((DD)*D))*(D`)*y;
                    res=y-(D)*x_hat;
          r0=res;
                    normr=(norm(res,"L2"));
          end;
          do sss=1 to nrow(sol_x);
                    if sol_x[sss]=1 then sol_x[sss]= x_hat[indicator];
                    if sol_x[sss]^=0 then indicator=indicator+1;
          end;
          err=err//normr;
          xs_sol=shape(sol_x,81,9);
          locmax=xs_sol[,<:>];
          newda=newda||locmax;
          corr=(solution=locmax);
          sol_corr=sol_corr//(sum(corr))/(nrow(corr));
     free D; end;
loca=sol_corr[<:>];
newdata=newdata//newda[,loca];
perc_correct=perc_correct//(max(sol_corr)*100);
Num_correct=Num_correct//(max(sol_corr)*81);
errorL=errorL//(min(err)); free clue;
free d1;
free err;
free sol_corr;
free newda;
dur = dur//(datetime() - &_timer_start);
end;
*-------------------------summary of time statistics ---------------;
time_in_sec=dur[1]//j(nrow(dur)-1,1,1);
do zip=2 to (nrow(dur));
  time_in_sec[zip]=dur[zip]-dur[zip-1];
end;
total_time_in_sec=sum(time_in_sec);
av_time_in_sec=mean(time_in_sec);
std_time_in_sec=std(time_in_sec);
print perc_correct Num_correct errorL time_in_sec;
print total_time_in_sec av_time_in_sec std_time_in_sec;
*-----------------------------create data set--------------------;
create sasuser.data from newdata[colname={'sol'}];
append from newdata;
run;
quit;
```